University of Southampton

Faculty of Social Sciences

School of Economic, Social & Political Sciences

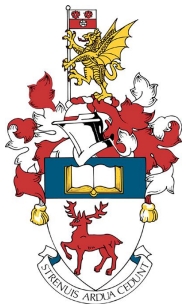
Department of Economics

Deep learning in econometrics: theory and applications

Tullio Mancini

A thesis for the degree of *Doctor of Philosophy*

May 2021



University of Southampton

Abstract

Faculty of Social Sciences
School of Economic, Social & Political Sciences
Department of Economics

Doctor of Philosophy

Deep learning in econometrics: theory and applications

by Tullio Mancini

This thesis concentrates on implementing deep learning methodologies for econometrics. Among the supervised machine learning toolbox, deep neural networks are the most ubiquitous ones. By being the least restricted nonlinear functions that one could implement, deep learning models allow learning complex signals from the infinitely versatile big data. Recent evidence from the literature shows how, notwithstanding the well-known issues around neural networks specification for causal problems, deep learning methods (among other machine learning methodologies) can be regarded as a powerful nonparametric tool for addressing not only prediction but also causal problems. Chapter 2 develops a new methodology for detecting Granger causality in nonlinear multivariate time series using deep neural networks coupled with Lasso methods. After defining the optimal neural network architecture by maximizing the transfer of information between input and output variables, the novel two-stage procedure applies a sparse double group lasso penalty function to detect Granger causality. The methodology is applied to the Tobalaba network of renewable energy companies showing an increase in the connectivity among the network members after the introduction of the blockchain platform. Chapter 3 proposes a constrained maximization for the identification of an optimal neural network architecture of a given size. The optimal architecture obtains from maximizing the minimum number of linear regions approximated by a deep ReLu neural network. A Monte Carlo simulation illustrates the optimal architecture's outperformance against cross-validated methods for linear and nonlinear prediction models. Chapter 4 proposes a suitable method for constructing prediction intervals for the output of both deep and shallow neural networks. The proposed methodology adapts the extremely randomized trees method to construct ensembles of neural networks. The Monte Carlo simulation shows a good performance of the novel methodology not only in terms of out-of-sample uncertainty estimation but also in terms of out-of-sample accuracy. Finally, chapters 5 and 6 apply the novel deep learning methodologies –proposed in the previous chapters– to environmental economics. More specifically, chapter 5 uses ReLu deep neural networks to predict the CO_2 emissions associated with Bitcoin mining showing that the fossil fuel emission associated with Bitcoin mining, for the year 2018, is higher than the annual levels of fossil fuel emissions of some U.S. states such as Maine, New Hampshire, and South Dakota. Lastly, chapter 6 uses both deep and shallow neural networks to construct environmental Engel curves for the U.S. for the years 1984 and 2012. The empirical results show that richer households pollute more, the pollution content of consumption increases at a lower rate than income, and that the pollution content of consumption grows at a decreasing rate. Finally, Appendix A, B, and C provide a brief theoretical introduction to feedforward, convolutional, and recurrent neural networks to allow the reader to understand the similarities and the differences between the different deep learning methods; and three different empirical applications (focused on regression, image classification, and text generation) are implemented to show the strength and power of the different classes of deep networks.

Contents

List of Equations

List of Theorems and Properties

List of Algorithms

List of Tables

List of Figures

Declaration of Authorship

I, Tullio Mancini, declare that this thesis entitled *Deep learning in econometrics: theory and applications* and the work presented in the thesis are both my own and have been generated by me as the result of my own original research

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:
   (a) Calvo-Pardo, H.F., Mancini, T. and Olmo, J. (2020) "Neural Network Models for Empirical Finance" *Journal of Risk and Financial Management*; 13(11), p. 265.
   (b) Calvo-Pardo, H., Mancini, T. and Olmo, J. (2021) "Granger causality detection in high-dimensional systems using feedforward neural networks" *International Journal of Forecasting*; 37(2), p. 920-940.

Signed: ..... Date: .....

List of Abbreviations

DF ..... Dickey-Fuller

DGP ..... Data generating process

DNN ..... Deep neural network

ECC ..... Environmental Engel curve

FAVAR ..... Factor-augmented vector autoregressive

FNN ..... Feedforward neural network

KPSS ..... Kwiatkowski-Phillips-Schmidt-Shin

IAM ..... Integrated assessment models

Lasso ..... Least absolute shrinkage and selection operator

MAE ..... Mean Absolute Error

MCS ..... Monte Carlo Simulation

MSE ..... Mean Squared Error

MSFE ..... Mean Square Forecast Error

NRMSE ..... Normalized Root Mean Squared Error

OOB ..... Out-of-Bag

OLS ..... Ordinary Least Square

ReLu ..... Rectified Linear Unit

RMSE ..... Root Mean Squared Error

SCC ..... Social Cost of Carbon

Tanh ..... Hyperbolic tangent activation function

PCA ..... Principal component analysis

PoW ..... Proof-of-Work

PoA ..... Proof-of-Authority

PWL ..... Piecewise linear

SLSQP ..... Sequential least square programming

VAR ..... Vector autoregressive

Glossary

Activation Function: Defines the output of the hidden nodes. The *post-activation value* introduces the nonlinearity in the system.

Adversarial Example: Small variations in the neural network inputs that result in the wrong output from the network.

Bias: Term used for the intercept terms in the affine transformations identifying neural networks.

Deep Network: Neural network with more than one hidden layer.

Epoch: Number of passes performed by the machine learning algorithm through the entire training set.

Feedforward neural network: The information flows from the input layer, thorough the hidden layers, and finally to the output layer. There are no feedback connections that feed the output of the model back to the input layer (Goodfellow et al., 2016).

Fully connected neural network: When all the weights are different from 0; that is, there are no initial sparse connections.

Hidden layers: Intermediate layers of a neural network, through learning algorithm they perform the neural computation.

Input layer: First layer of a neural network, defined by *passive* nodes that pass the observed data into the model.

Neural Network: Composition of functions; defined by Goodfellow et al. (2016) as "*Directed acyclic graph describing how the functions are composed together. Given three functions we define a neural network as* $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$".

Out-of-Bag error: Allows out-of-sample model evaluation using the observations not sampled during bootstrapping.

Output layer: Final layer of a neural network, the choice of the activation function depends on the output to be returned.

Random Forest: Ensemble learner that fits decision trees on the bootstrapped dataset.

ReLu Function: Piecewise linear function that returns 0 if the input is negative, otherwise the input itself if positive.

Shallow Network: Neural network with only one hidden layer.

Sigmoidal function: Defined as a *squashing* function, is a mathematical function having a "S"-shaped curve.

"To understand its [cerebral cortex] proper function we need to know what it computes. Its output is some function of its input. As yet we do not know, even for the simplest structure, what that function is"

W. S, McCulloch, 1950

CHAPTER 1

Introduction to the Thesis

Chapter Abstract

The present chapter discusses the importance of deep learning methods in both economics and econometrics, and it summarizes the contributions that the present thesis provides to the modern econometric literature focusing on feedforward neural network methods for regression. It analyzes the literature on VAR estimation and Granger causality detection in high-dimensional nonlinear systems; the importance of identifying the optimal neural network structure not only in terms of out-of-sample accuracy but also in terms of causal analysis; and the estimation of the uncertainty around deep learning predictions. Following, the contributions and the methodologies of each chapter are summarized.

1.1 Problem Statement

Due to recent developments in digitalization, the world is awash with big and complicated data, and researchers are trying to make sense out of it. The computer scientists, statistics, and economics (more recently) communities use sophisticated tools from supervised machine learning (SML) to learn from the infinitely versatile big data. Among the SML toolbox, deep learning methods are the most notorious ones. Deep learning models allow learning complex signals by composing simple nonlinear functions into multiple hierarchical levels of representations of the raw data. Depending on the type and structure of the data analyzed, practitioners and scholars recognize and implement three distinct classes of deep neural networks[1]: 1) *deep feedforward neural networks* that have been extensively used for signal processing in intricate structures in large datasets (e.g., Chakraborty et al., 1992; and Kaastra and Boyd, 1996), 2) *deep convolutional neural networks* that have brought breakthroughs in image and video processing (e.g., Lawrence et al., 1997; and Krizhevsky et al., 2017), and 3) *deep recurrent neural networks* that provide state-of-the-art tools for the analysis of sequential time series, text, and speech data (e.g., Mikolov et al. (2011); Connor et al., 1994; and Che et al., 2018). An intuitive definition of deep learning –based on the aforementioned idea of function composition– will now be provided. It is well known that, given a set of N input-output pairs $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$, the nonlinear relation between $\mathbf{X}$ and y can be defined by the following linear basis function model:

$$f(\mathbf{X}) = \mathbf{b} + \mathbf{W}^{\intercal}\theta_n^{\mathbf{W}_n,\mathbf{b}_n}(\mathbf{X}) \tag{1.1}$$

with $\theta(\mathbf{X})$ an element-wise nonlinear function (e.g., logistic sigomid) on the linear affine transformation of the form $\mathbf{W}^{\intercal}\mathbf{X} + \mathbf{b}$, $\mathbf{W}$ some $Q \times D$ matrix, and $\mathbf{b}$ a vector with D elements. By defining a hierarchical composition of n nonlinear basis functions on the input $\mathbf{X}$ as $\boldsymbol{\Xi} = \theta_n^{\mathbf{W}_n,\mathbf{b}_n} \circ \cdots \circ \theta_1^{\mathbf{W}_1,\mathbf{b}_1}(\mathbf{X})$, it is possible to represent a deep neural network (feedforward neural network) as:

$$f(\mathbf{X}) = \mathbf{b} + \mathbf{W}^{\intercal}\theta_n^{\mathbf{W}_n,\mathbf{b}_n}(\boldsymbol{\Xi}) \tag{1.2}$$

As an illustrative example, Figure 1.1 reports a representation of a sequential two hidden layers feedforward neural network with five and three hidden nodes (layer-wise widths) respectively, for $\mathbf{x}_i \in \mathbb{R}^d$.

A defining characteristic of deep feedforward neural networks is their ability to accommodate a large set of potential predictor variables and to approximate complex unknown functional forms of the data. In particular, the universal approximation theorem –initially proposed by Cybenko in 1989– states that a sufficiently large neural network will approximate any underlying function with an approximately small error, making no longer necessary to construct an ad hoc model for the specific nonlinearity to be learned. Starting from these

[1] See Schmidhuber, 2015; and LeCun, et al., 2015 for overviews of the topic.

premises, it is possible to understand why the methodological literature in economics and econometrics has started focusing on the topic, resulting in a rapid adoption of the methodology also in empirical work (Athey and Imbens, 2019). Among many, noteworthy examples are Kaji et al. (2018) that use Generative Adversarial Networks (Goodfellow et al., 2014) for structural estimation where the likelihood function is often intractable or does not have a closed form solution; Farrell et al. (2019) who, by defining new rates of convergence for deep feedforward neural networks, establish valid two-step inference; Hruschka (1993) that uses artificial neural networks to model market response functions as opposed to classical econometric models; and Gu et al. (2020) that –by performing a comparative analysis of different machine learning methods– show the superior performance of portfolio strategies based on the forecasted conditional means of the asset returns obtained from deep feedforward neural networks[2].

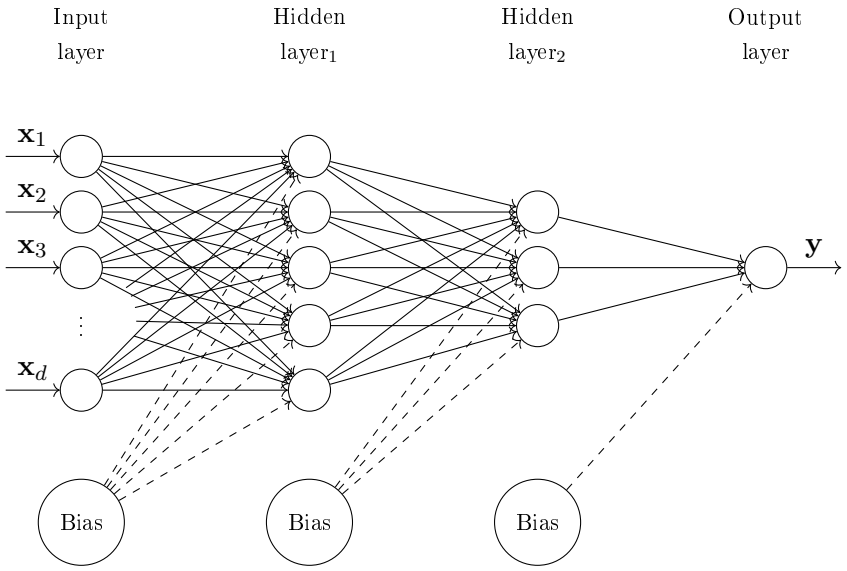


Figure 1.1: Deep Feedforward neural network for regression. It comprises two hidden layers with a number of hidden nodes (layer wise widths) equal to five and three respectively.

Notwithstanding the advancements made by the current literature in accommodating SML methods to the economics and econometrics literature, numerous facets have not been addressed yet. Thus, the present thesis, focusing on feedforward neural networks, contributes to the deep learning literature by proposing suitable methods for Granger causality detection, structure identification, and uncertainty estimation in high-dimensional nonlinear systems. One of the main impediments for neural networks to be considered a standard tool for time series analysis is the lack of interpretation. In fact, due to the tangled web of interacting nodes between and across layers, it is difficult to quantify exactly the effects of the inputs. The second chapter adds interpretability to neural network structures into time series settings

[2] For a detailed review of SML methods for economists, the interested reader is referred to Athey and Imbens (2019); and Mullainathan and Spiess (2017).

by imposing a mapping between the "original" variables and the nodes in the first hidden layer; consequentially, the magnitude of the weights associated with the nodes in the first layer determines the presence of Granger causality between input and output variables. The novel methodology proposes a sparse double group lasso penalty function that allows for the estimation of the weights characterizing the transfer of information through the neural network and model selection –Granger causality and lag selection.

In their seminal paper, Farrell et al. (2019) establish rates of convergence for deep feedforward neural networks, which is of great importance when focusing on causal inference. The authors highlight that –just as for classic nonparametric modeling– it is the choice of the tuning parameters of the neural network that determines the rates of convergence. For this reason, as the current literature in deep learning has not yet shown that a neural network structure (in terms of width and depth) can return an optimal approximation, they are only able to prove a bound with a slower than optimal rate. The third chapter –by building on the literature focusing on the approximation power of deep neural networks– contributes to the advancements made by Farrell et al. (2019) by proposing a novel methodology for the identification of a neural network structure (width and depth) that maximizes the approximation power of a neural network with a given size. The optimal architecture obtains from maximizing the lower bound on the maximal number of linear regions approximated by a deep neural network with ReLu activation function. In other words –similarly to the recursive binary splitting algorithm in regression/classification trees– the proposed optimization ensures an optimal allocation of hidden nodes across hidden layer that maximizes the efficiency in which a neural network architecture divides the feature space in sub-regions.

By quoting Athey and Imbens (2019) "*We view econometrics, as in essence, decision making under uncertainty*", one can understand the importance of proposing a suitable method for constructing prediction intervals for the output of neural networks. The fourth chapter adds to the literature on deep learning by proposing a novel methodology for the construction of a finite-sample approximation of the prediction intervals by adapting the extremely randomized trees method, originally developed for random forests, to construct an ensemble of neural networks. The extra-randomness introduced in the ensemble reduces the variance of the predictions and yields gains in out-of-sample accuracy.

Earth's climate has been changing throughout history. All the large-scale species extinctions observed in the past 500 million years have been caused by either excessive global cooling or warming. If most of these observed climate changes are attributed to the change in the amount of solar energy received by the Earth due to changes in the Earth's orbit, the current climate warming (starting from 1900) is characterized by an unprecedented increasing trend. Being the anthropogenic increase in greenhouse emissions the most likely cause of this concerning trend, the Paris Accord Agreement at COP21 imposes to limit the temperature increase to less than 2°C. As a result, it has become socially imperative for policymakers to pursue zero-net carbon emissions policies (as an example, China's plan for zero-net carbon emissions by 2060). Kleinberg et al. (2015) show how machine learning methods can be

used to address the often-neglected (as highlighted by the authors, empirical policy research focuses on causal inference) prediction policy problems (e.g., predicting unemployment spell length) producing not only policy impact but also economic insights. Similarly, the present thesis –focusing on environmental economics– argues that deep learning methods can enable timeless public decision-making regarding pressing complex social issues. In particular, the fifth chapter adds to the literature by proposing a deep learning method for the estimation of the carbon footprint of Bitcoin mining. Mora et al. (2018) forecast that Bitcoin's cumulative emission alone –due to the energy-intensive validation protocol– could violate the Paris agreement by 2040. If one considers the growing interest of national governments in cryptocurrencies (e.g., China) and the possibility of issuing financial instruments solely on blockchain (e.g., Bank of Australia and World Bank *bond-i*), the correct prediction of CO_2 emission with relative uncertainty measures becomes pivotal for efficient policymaking. Finally, chapter 6 replicates the study by Levinson and O'Brien (2019) on Environmental Engel curves (ECC) using deep learning methods. The study of the EECs allows understanding why –notwithstanding the increase in the real value of the U.S. production– the pollution emitted by American households has declined significantly.

1.2 Literature Review

The present section reviews the main body of the literature related to the present thesis. First, the importance of analyzing the dynamic relation of time series is discussed and framed within a high-dimensional nonlinear setting; the contributions proposed by the current literature – with relative limitations– are also reported. Second, the importance of structure identification is also reviewed. Following, the importance of uncertainty estimation in econometrics and in deep learning settings is examined.

1.2.1 Granger causality

The concept of causality introduced by Wiener (1956) and Granger (1969) constitutes a basic notion for analyzing dynamic relationships between time series. In particular, Granger (1969) states that time series $\mathbf{x}_{tj}$ does not strictly Granger cause time series $\mathbf{x}_{ti}$ if $g(\mathbf{x}_{ti}|\mathbf{I}_{t-1}) = g(\mathbf{x}_{ti}|\mathbf{I}_{t-1} - \mathbf{x}_{t-1,j})$, with $\mathbf{I}_{t-1}$ being the lagged information set available at time t. In other words, the study of Granger causality is based on detecting whether considering past realizations of $\mathbf{x}_{tj}$ helps in forecasting $\mathbf{x}_{ti}$. A natural parametric setting to assess for the presence of Granger causality in a multivariate setting is the family of Vector Autoregressive (VAR) models introduced by Sims (1980). However, the usage of this parametric model relies on the important assumption that there is no omitted variable bias. The violation of this assumption led to empirical problems not only in terms of policy evaluation (the well-known problem of the "prize puzzle" discussed in Christiano et al., 1999) but also in terms of accurate forecasting (and thus correct Granger causality detection[3]). As explained in Sims (1998), the key to addressing these empirical problems is to increase the information set included in the VAR

[3] Based on the above definition, one could notice how the correct detection of Granger causality depends on the correct specification of the information set I_{t-1} considered.

systems; as a consequence, Sims and coauthors started considering larger VARs, moving from the six-and eight-variables VARs usually adopted to the thirteen and eighteen-variables VARs in Leeper et al. (1996).

However, increasing the number of variables in VARs can lead to over-parametrization in large dimensions; in fact, the number of VAR coefficients increases as the square of the number of variables in the multivariate time series. If one considers that –in reality– Federal Reserve (FED) economists analyze thousands of variables when preparing for meetings of the Open Market Committee (Stock and Watson, 2005), the "naive" inclusion of a higher number of variables in the VAR system becomes a non-feasible solution. Thus, the literature has proposed different procedures to overcome the "profligate parametrization" that can affect high-dimensional VARs, and they can broadly be classified into two main groups: i) dimensionality reduction and ii) sparsity induction via convex regularizers.

In the first group, the literature tries to solve the over-parametrization that may affect VARs in high-dimensional settings by limiting the number of variables considered in the multivariate time series without reducing the information set available. In his Ph.D. thesis, Geweke (1977) introduces the concept of dynamic factor analysis. The underlying theory in dynamic factor models (DFM) is that it is possible to summarize the information set describing a large number of time series using a small number of indices or *factors* (see, for example, Stock and Watson, 2002). It follows that, if a small number of factors can effectively summarize the information contained in a large number of time series, it is possible to increase the information set of a standard VAR –without increasing the number of degrees-of-freedom– by augmenting the model with a set of estimated factors. One of the most notorious approaches is implemented by Bernanke et al. (2005) that propose a factor-augmented VAR model (FAVAR) where the factors are estimated using principal component analysis (PCA)[4]. By including the principal components (PC), the FAVAR model is then able to capture macroeconomic aggregated information while remaining tractable in terms of the number of parameters included (examples of empirical applications are Jurado et al., 2015; Boivin et al., 2010; Bianchi et al., 2009; and Forni and Gambetti, 2010). One of the main limitations of the aforementioned factor-augmented VAR models is the assumption that the true data generating process determining the interactions between the variables is linear, or more generally known. To overcome this limitation, Babikir and Mwambi (2016) propose to augment the factors to multivariate feedforward neural networks that, being universal approximators, can model both linear and nonlinear data generating processes. If both parametric and non-parametric FAVAR models provide a suitable solution to improve the forecasting performance of VAR models in high-dimensional systems, it is also true that they do not allow for the correct detection of Granger causality in high-dimensional VARs. In particular, the main limitations (for the parametric and non-parametric FAVARs) lie in the loss of interpretability due to the transformations

[4] Forni et al. (2000) prove that dynamic (the estimated factors are modeled as following a linear dynamic process such as VAR or AR) PCA provides a consistent estimation of the common factors as both the dimension p and the time T increase. Forni et al. (2004) further show that estimation consistency holds also if both p and $T \to \infty$, and $n/T \to 0$.

involved in the factor computation that make impossible to track the Granger causal interactions between the "original" multivariate time series, and in the absence of a suitable method for the estimation of Granger causal interactions when the multivariate time series is modeled using feedforward neural networks.

Recently, the statistical and machine learning literature has focused on imposing sparsity in the estimated model coefficients via convex loss functions such as the least absolute shrinkage and selector operator (Lasso; Tibshirani, 1996). Following, Yuan and Lin (2006) propose a more sophisticated version called Group Lasso, which imposes sparsity in the model parameters by jointly deleting groups of coefficients. However, none of these approaches takes into consideration the structure of the time dependence in multivariate time series. To overcome this limitation and to accommodate penalty functions that allow not only variable but also lag selection, Nicholson et al. (2014) propose a Hierarchical Group Lasso for the estimation of VARs in high-dimensional systems (see also Song and Bickel, 2011; Callot and Kock, 2014; and Skripnikov and Michailidis, 2019). Most importantly, Hecq et al. (2019) propose a first step towards correct inferential procedures by extending the post-double selection approach of Belloni et al. (2014) to Granger causality detection in linear sparse high-dimensional VARs. However, the sparse high-dimensional VARs proposed by the literature still rely on the assumption that the underlying data generating process of the multivariate time series is linear or known. A powerful methodology for forecasting nonlinear multivariate time series is neural networks (see, for example, Chakraborty et al., 1992; and Kaastra and Boyd, 1996). As previously mentioned, the main impediment for neural networks to be considered as a standard tool for time series analysis is the lack of interpretation. However, recent work by Scardapane et al. (2017) adds interpretability to neural network structures by imposing a mapping between the set of regressors and the hidden nodes in the first hidden layer. The authors state that for sequential feedforward neural networks pruning nodes in the first hidden layer is equivalent to deleting variables from a regression model. Tank et al. (2018) are the first authors to apply Scardapane et al. (2017)'s strategy to Granger causality detection in high-dimensional nonlinear time series. In particular, these authors extend the Hierarchical Group Lasso proposed by Nicholson et al. (2014) to the weights of neural networks. However, their work does not analyze the impact of the architecture of the neural network (in terms of both depth and width) on the correct detection of Granger causality.

Chapter 2 contributes to the literature on VARs estimation and Granger causality detection in high-dimensional nonlinear systems by proposing to model the multivariate time series using element-wise feedforward neural networks with the novel sparse double group lasso that allows not only for variable but also for lag selection. The proposed methodology is compared with sparsity induction methods for detection of Granger causality in high-dimensional settings (linear VAR by Nicholson et al., 2014; and neural network approach by Tank et al., 2018). Our methodology differentiates from Tank et al. (2018) in two main aspects. First, we propose an algorithm for the identification of the optimal neural network structure (optimality for Granger causality detection equals maximization of information transfer through the neural network), and second, we propose a different lasso penalty function. One of the

main contribution of the chapter is to highlight the pivotal role that the identification of the correct (maximizes the forecasting accuracy) neural network's structure plays in detecting Granger causality. Therefore, the following subsection will focus on the relevant literature analyzing the role that the neural network structure (in terms of width and depth) plays in the approximation power of deep learning models.

1.2.2 Optimal Structure Identification

The success of deep learning methods in high-dimensional linear and nonlinear problems such as pattern recognition, biomedical diagnosis, and text processing (see also Schmidhuber, 2015; and LeCun et al., 2015 for review on the topic) relies on their ability to approximate the unknown functional form $y = f(\mathbf{X})$. The *universal approximation theorem* by Cybenko (1989) states that a sufficiently large shallow (only one hidden layer) feedforward neural network with sigmoidal activation functions can approximate any Borel measurable function with approximately small error. Leshno et al. (1993) extend the results of Cybenko (1989) to deep (more than one hidden layer) feedforward neural networks as long as the activation functions are bounded and not polynomial. Following, Hornik (1991), in his seminal paper, extends the results of Cybenko (1989) by proving that a sufficiently large neural network with *any* bounded and nonconstant activation function is a universal approximator. Thus, focusing on shallow neural networks, one could notice how the universal approximation theorem implies fitting a neural network with a number of hidden nodes (width) greater than the number of input nodes (dimensions of the training set), leading to the possible problem of overfitting. However, Mei et al. (2018) complete the results by Hornik (1991) by proving that a shallow network with a width much greater than the number of input nodes, and trained using stochastic gradient descent, will converge to a solution close to the optimum; implying that even if the number of nodes grows to infinity, a shallow neural network will never overfit. Finally, Lu et al. (2017) extend the universal approximation theorem by Cybenko (1989) to width-bounded feedforward neural networks with rectified linear unit (ReLu) activation function and a minimum depth equal to $d + 4$ with d the input dimension. Thus, one could infer how the approximation power of both deep and shallow networks depends not only on the activation function used but also on the correct specification of the dimension (sufficiently large given the particular underlying data generating process analyzed) of the neural network.

The results reported in the above paragraph imply that both shallow and deep neural networks –for a given activation function used and with a sufficiently large number of hidden nodes– are universal approximators. The implications of these results are noteworthy: notwithstanding the underlying data generating process, a shallow or deep feedforward neural network with a sufficiently large number of hidden nodes will be able to approximate, accurately, the underlying function. Thus, it is no longer necessary to construct an *ad hoc* model for the specific functional form to be learned.

As explained in chapter 3, the output of a ReLu neural network (with identity output function) can be regarded as a weighted sum of indicator functions that take as input hierar-

chical nonlinear representations of the input space. A similar formulation is obtained when considering random forests where the recursive binary partition algorithm –used to train the individual trees– is designed to identify the optimal number of non-overlapping regions that ensures optimality in predicting the target variable. When fitting random forests, the depth of the individual trees defines the bias-variance trade-off: an increase in depth leads to an increase in the number of non-overlapping regions being identified, leading to a higher approximation power that –depending on the complexity of the underlying data generating process– can be translated to either lower bias or higher variance.

The depth and the width of ReLu neural networks play a similar role to the depth of the regression trees comprising random forests: they define the number of linear (sub-regions) approximating the target function. Therefore, depending on the underlying data generating process analyzed, an increase in the number of linear regions being approximated by shallow or deep ReLu neural networks can lead to a decrease in bias or increase in variance. However, when analyzing neural networks, the weighted sum does not consider non-overlapping regions where the predicted output is the mean response value, but it applies to interacting hyperplanes that take as input values folded representations of the input space. This final aspect has an important impact on the bias-variance trade-off: Pascanu et al. (2013) explain how the interactions among the different hyperplanes (or sub-regions) ensure a good generalization performance. Consequently, the risk of overfitting (high variance) increases at a slower rate when increasing the depth/width of neural networks as opposed to increasing the depth of regression trees.

In other words, the correct specification of the width and the depth of the neural network, together with other factors, ensures an efficient balance between variance and bias. This final aspect is analyzed in the research conducted by Kraus et al. (2020) where the authors show how out-of-box architectures (i.e., sub-optimal identification of the number of sub-regions approximating the target function) leads to underfitting (high bias). Similarly, empirical research (see, for example, Mesnil et al., 2011; Kim and Gofman, 2018; and Pasupa and Sunhem, 2016) show how in some instances, shallow structures outperform deeper structures (deep structures lead to high variance due to a number of sub-regions being higher than the optimal one) while in others deeper structures return the best out-of-sample accuracy (shallow neural networks are not able to approximate the underlying data generating process correctly). It is standard in the machine learning literature on feedforward neural networks to use k-folds cross-validation methods to choose the width and the depth of the network. However, the performance of cross-validation methods depends on (I) the dimensions of the hyperspace (in this specific case, possible neural network architectures) –if too few are tuned, we may miss the global optimum or if too many we may overfit (Rao et al., 2008); (II) the observations must be *i.i.d.* and the distribution of the target variable must be similar across different folds; and (III) the number of observations available.

To not rely on heuristic approaches for the identification of the optimal neural network structure, recent literature on deep learning has been focusing on understanding the role

that depth and width of a given architecture play in the approximation power of ReLu neural networks. Eckle and Schmidt-Hieber (2019) show how ReLu neural networks –being a composition of piecewise linear (PWL) functions– are also PWL functions. This final aspect implies that ReLu neural networks can be described by the number of linear regions that they can approximate. This concept is further clarified in Farrell et al. (2019) when comparing neural networks to more classic nonparametric techniques. In particular, one could notice that smoothing splines are usually defined by the spline basis (smoothing parameter) and by the number of knots (tuning parameters). In kernel regression, the shape of the kernel constitutes the smoothing parameter, while the bandwidth defines the tuning parameter. When analyzing support vector machines (SVMs), the shape of the kernel constitutes the smoothing parameter, while the C-parameter the tuning parameter. Lastly, a similar comparison could be extended to random forests where the dimension of the random subset of features, used in the greedy algorithm, constitutes the smoothing parameter, while the depth and the number of trees the tuning parameter. Similarly, when focusing on neural networks, the type of connections (graph structure) and the activation function define the smoothing parameters, while the width and depth of the neural network are the tuning parameters. Finally, Arora et al. (2018) show how the number of hidden nodes used to train the network defines the number of linear regions approximated by a ReLu feedforward neural network

It is from this final aspect that Montufar et al. (2014), Pascanu et al. (2013), and Raghu et al. (2017) state that the number of linear regions approximated by a ReLu neural network defines the model flexibility and thus, the complexity of the unknown function that can be approximated. These authors, starting from the results of Zaslavky (1975) on the number of linear regions defining an arrangement of hyperplanes, define the number of linear regions approximated by a shallow ReLu neural network (Pascanu et al., 2013), and the lower (Montufar et al., 2014) and upper (Raghu et al., 2017) bounds on the maximal number of linear regions represented by a deep ReLu neural network. Based on their results, the authors show how an increase in depth in a deep ReLu network leads to a gain in accuracy always greater than the one obtained from an increase in width; or similarly, that for a given number of hidden nodes, deeper structures will always outperform shallower ones.

Yet, in some empirical applications, shallow networks are shown to outperform deep networks (see, for example, Pasupa and Sunhem, 2016; and Kim and Gofman, 2018). Starting from the results of Pascanu et al. (2013) and Montufar et al. (2014), chapter 3 shows that under certain conditions, a shallow network could provide a better approximation of the unknown function $f(\mathbf{X})$, and that the arrangement of hidden nodes across the layers of a neural network influences the approximation power of the model. Based on these results and on the relevant literature on linear regions approximation, one could conclude that it exists an optimal neural network architecture (in terms of depth and width) that maximizes the number of linear regions approximated and thus, the approximation power of the model. Chapter 3 builds on top of this intuition by proposing a constrained maximization for the identification of the optimal neural network architecture.

Based on the previous sections, one could see that proposing a methodology for structure identification (that ensures optimal approximation) allows not only correct Granger causality detection (chapter 2) and correct causal inference, but also maximization of the out-of-sample accuracy of a neural network. However, as explained by Hüllermeier and Waegeman (2020) and Pearce et al. (2018), out-of-sample pointwise accuracy is not enough; the predictions and forecasts of neural network models need to be supported by measures of uncertainty. Thus, the following subsection will focus on uncertainty estimation in deep learning.

1.2.3 Uncertainty and Deep Learning

As previously mentioned, pointwise accuracy is not enough. A trustworthy representation of the uncertainty in deep learning methods can be considered pivotal when the neural networks are applied to medicine (Yang et al., 2009; Lambrou et al., 2011), or to anomaly detection, optimal resource allocation, and budget planning (Zhu and Laptev, 2017), or surgical robots, self-driving cars and smart grids (Varsheny and Alemzadeth, 2017). Similarly, starting from Athey and Imbens (2019), uncertainty estimation can be considered crucial in economics and decision making. However, the literature focusing on estimating the uncertainty of deep learning models around their predictions is still in its infancy.

When focusing on uncertainty estimation for deep learning models, the literature (see, for example, Pearce et al., 2018; and Heskes, 1997) distinguish between three main sources of uncertainty: model misspecification or bias which captures how closely $\hat{f}(\mathbf{X})$ can approximate the true $f(\mathbf{X})$, assuming infinite training data; data uncertainty or *aleatoric uncertainty* which is the irreducible noise due to an inherently stochastic process in the training sample; and parameter uncertainty or *epistemic uncertainty* which captures the uncertainty around the estimates of the model parameters. Based on the previous subsection, one could understand why, when focusing on deep learning models, the model misspecification is usually ignored by the literature; in fact, if the training data are assumed infinite, a sufficiently large neural network is a universal approximator that represents exactly $f(\mathbf{X})$. The distinction between epistemic and aleatoric uncertainty is extremely relevant for deep learning models. As an example, the well-known problem of *adversarial examples* by Papernot et al. (2017) –deep learning models are usually characterized by drastic changes in their performance when small perturbations are engineered to the input data– implies high variability in the parameter estimates.

A first step towards estimating the uncertainty around the predictions of deep learning models is proposed by Hwang and Ding (1997) in their seminal paper. In particular, these authors and the literature focusing on the delta method (see, among the others, Ungar et al., 1996; and De vieaux et al., 1998) –starting from the literature on prediction intervals for nonlinear regression (see Seber and Wild (1989) for a complete argumentation)– propose asymptotically valid prediction intervals when the feedforward neural network is trained to convergence. However, due to the increasing complexity of the data to be analyzed, and the associated increase in the complexity of the models proposed by the literature in deep learn-

ing[5], the delta method firstly introduced by Hwang and Ding (1997) is not widely adopted by the empirical literature. The main limitation lies in the correct computation of the Jacobian matrix required to estimate the prediction interval. Tibshirani (1996) explains how the probability of computation error increases with the number of parameters estimated in the model. Additionally, De vieaux et al. (1998) also explain how the near singularities in the model parameters due to overfitting or due to a small sample size make the computation of the Jacobian matrix unfeasible.

For these reasons, recent literature has been proposing finite-sample approximations of the prediction intervals of deep neural networks. Heskes (1997) proposes to estimate the epistemic uncertainty using bootstrap methods. In particular, multiple feedforward neural networks, with different parameters initialization, are trained on different resampled versions of the training data[6]. The second moment of the approximate predictive distribution allows the computation of parameter uncertainty. The aleatoric or data noise uncertainty is then estimated as the variance of the residuals in a hold-out set (if it is assumed homoscedasticity) or using the Mean-Variance Estimation (MVE)[7] proposed by (Nix and Weigend, 1994), if it is assumed heteroscedasticity. By assuming independence among the two different sources of uncertainty (Pearce et al., 2018), the total variance to be used for the construction of the prediction interval is then obtained as the sum of the epistemic and aleatoric uncertainties. However, the bootstrap method has some inherent limitations: (I) it requires the assumption that the observations are independent and identically distributed; (II) each neural network is trained only on 63% of the train data (see Lee et al., 2015); and (III) it is computationally intensive. Additionally, empirical research by Lee et al. (2015) and Lakshminarayanan et al. (2017) show how data resampling deteriorates the out-of-sample estimation accuracy and the uncertainty estimation of bootstrap-based methods.

Based on the aforementioned limitations, the literature focusing on uncertainty estimation for deep learning methods has been widely adopting the Monte Carlo (MC) dropout proposed by Gal and Ghahramani (2016a). The methodology, originally proposed for the approximation of the posterior distribution for Bayesian neural networks (Denker and LeCun, 1991), involves fitting a neural network with dropout (Srivastava et al., 2014) not only at training but also during test phase. As a result, the *stochastic forward passes* at test time allow approximating the predictive distribution of a neural network without fitting a different model for each forward pass (reducing the computation requirements) and without resampling the original dataset. The epistemic uncertainty will be then defined by the second moment of the obtained approximate predictive distribution. As for the bootstrap method, the aleatoric uncertainty is then estimated as the variance of the residuals in a hold-out sample (Zhu and Laptev, 2017) or via the MVE (Serpell et al., 2019). One of the main limitations of the MC dropout is that the model parameters are fixed across the random samples implying that the cross-

[5] One could think of the AlexNet (Krizhevsky et al., 2017) that is defined by 60 million parameters and 650,000 hidden neurons.

[6] See Tibshirani (1996) for the bootstrap pairs and bootstrap residuals algorithms.

[7] It involves fitting a neural network with two output nodes: one for the means and the other for the variance of a normal distribution.

correlation between predictions is perfect; as a result, the MC dropout will not benefit (as in the bootstrap-based approaches) from the reduction in the variance subsequent to model averaging. Additionally, being the epistemic uncertainty determined solely by the chosen Bernoulli distribution, the correct uncertainty estimation depends significantly upon the right choice of p (as opposed to both bootstrap and delta methods). Chapter 4 proposes a novel methodology for uncertainty estimation for deep learning models that extends the extra-tree algorithm of Geurts et al. (2006) to feedforward neural networks.

1.3 Thesis Structure

The following section outlines the contributions of each chapter to the current literature by summarizing not only the novel methodologies developed but also the empirical findings.

1.3.1 Chapter 2

This chapter proposes a novel methodology for the detection of Granger causality in mean for vector autoregressive models in which the dynamic dependence structure is unknown and can take very general forms accommodating high-dimensional (in terms of both number of variables and lags) linear and nonlinear functional forms. The chapter builds on the recent work of Scardapane et al. (2017) that adds interpretability to neural network models by imposing a mapping between the variables of a multivariate system and the nodes in the first hidden layer of a neural network. In particular, a given variable is considered not significant if the weights connecting the relative input node to the hidden nodes in the first hidden layer are jointly equal to zero (the output of the neural network is invariant to the analyzed variable).

This chapter applies Scardapane et al. (2017)'s strategy and extends it to a multivariate time series setting. In particular, we construct a neural network with the input nodes defined by the lagged values of all the variables in the multivariate time series, and the output node by the vector of the dependent variable. The magnitude of the weights connecting the input layer to the first hidden layer determines the presence of Granger causality between input and output variables and lag selection. More formally, a particular input variable will be Granger causing an output variable if all the weights connecting the input nodes corresponding to all the considered lags of the given input node to the first hidden layer are jointly different from zero. Similarly, a given lag will be relevant if all the weights connecting the corresponding input node to the first hidden layer are jointly different from zero. Additionally, the chapter also shows that the optimal choice of the number of nodes in the first hidden layer reduces type I and type II errors when detecting Granger causality in neural networks.

Thus, the chapter proposes a two-stage procedure for the detection of Granger causality in neural networks. The first stage extends Montgomery and Eledath (1995)'s algorithm to deep neural networks and defines the optimal neural network architecture (number of nodes per hidden layer) by maximizing the mutual information transfer/minimizing the information loss through the network. Once the optimal number of nodes in the intermediate hidden layers is

obtained, a novel sparse double group lasso (proposed by Simon et al., 2013) penalty function is applied to estimate the weights in the feedforward neural network. The first group of the penalty function considers the weights associated with all possible lags of a given regressor and allows Granger causality detection. The second group considers all the weights associated with a given lag of a specific regressor and allows detecting the optimal number of lags of a given input variable.

The chapter also discusses results on parameter identification and model selection consistency as the sample size increases. The derived conditions for model selection consistency coincide with those found in the literature when the number of lags and variables is fixed (Fan and Li, 2001), and extend the literature by proving model selection consistency when the number of lags is allowed to increase with the sample size.

A comprehensive Monte Carlo simulation analyzes the impact that the correct structure identification plays in detecting Granger causality in terms of type I and type II errors. Additionally, it compares the performance of the proposed sparse double group lasso against a hierarchical group lasso (Nicholson et al., 2014) penalty function. Finally, it analyzes model selection consistency for increasing sample sizes. The simulation results show the good performance of the proposed methodology for the detection of Granger causality in terms of probability of type I and type II errors for both short and long-range (10 lags) dependence and for both linear and nonlinear data generating process.

Lastly, building on the recent literature on social and financial networks that identifies the presence of connections in a network via the presence of Granger causality (Billio et al., 2012; and Hecq et al., 2019), the chapter applies the novel methodology for the detection of Granger causal relationships between the financial returns of the energy companies trading in the Tobalaba network. The Tobalaba network is a test-net provided by the Energy Web Foundation (2018) that allows the stipulation of smart contracts between energy companies via a blockchain platform. Starting from the research of the World Bank Group (2018), according to which smart contracts increase the number of bilateral transactions and diversify the market structure, the empirical application analyzes the impact that the Tobalaba network had on the topology of the network comprised by the member of the blockchain platform. To do so, two Granger causal networks (before and after the introduction of Tobalaba) are constructed applying the novel two-stage algorithm. The empirical results, validated via a Diebold-Mariano (1995) test, show an increase in the number of interactions among the network members after the introduction of Tobalaba.

1.3.2 Chapter 3

The results from chapter 2 show the importance that structure identification plays on the correct detection of Granger causality. To identify a neural network architecture that maximizes the forecasting accuracy (and consequentially optimal for Granger causality detection), the previous chapter extends the algorithm of Montgomery and Eledath (1995) to deep feedfor-

ward neural network structure. However, the algorithm presents some limitations: it does not allow for depth selection, and it considers sigmoidal activation functions (which are shown to underperform when applied to extremely deep neural network architectures). For this reason, the present chapter focuses on proposing an alternative procedure for optimal architecture identification that allows not only width but also depth selection, and that applies to neural networks fitted using ReLu activation functions (considered the standard in the literature on deep learning, see Goodfellow et al., 2016).

In particular, this chapter proposes a constrained maximization for identifying a neural network structure that maximizes the approximation accuracy of a given neural network for a given number of nodes. The optimal neural network architecture obtains from maximizing the minimum number of linear regions approximated by a ReLu neural network with ReLu activation function, for a given number of nodes. Thus, based on recent work by Pascanu et al. (2013) and Montufar et al. (2014), the obtained neural network architecture will maximize –for a given number of nodes– the minimum expressive power of a deep neural network. The proposed maximization optimally allocates the hidden nodes across hidden layers (and thus, it selects the optimal width and depth of the network) to maximize the flexibility of the neural network in approximating the unknown data generating process, and thus the estimation accuracy. More specifically, the optimal neural network architecture is obtained by maximizing the lower bound on the maximal number of linear regions (Montufar et al., 2014) approximated by a ReLu DNN. The constraints applied to the maximization ensure a layer-wise width greater than the input dimension (due to the binomial coefficient in the analyzed lower bound) and a maximum number of hidden nodes. The latter constraint (maximum number of hidden nodes) directly depends on the data generating process analyzed: more complicated functional forms will naturally summon bigger structures.

The proposed maximization is solved in two stages: a first step that, given a number of hidden nodes and maximal depth (number of hidden layers), identifies the optimal layer-wise widths (number of hidden nodes per hidden layers); a second stage that, given the optimal layer-wise widths for different depths, identifies the optimal depth of the network with associated optimal widths. The optimal number of hidden nodes depends on the particular data generating process considered; therefore, cross-validation methods should be adopted to determine the optimal number of parameters to be adopted for given optimal architectures. By optimizing the width and the depth prior to training architectures of a given size, the proposed maximization substantially saves on computing time involved in the fine-tuning the optimal neural network, while ensuring an improvement in the predictive ability of the neural network.

Based on the theoretical results of Montufar et al. (2014) and Pascanu et al. (2013), the present chapter also proposes a set of properties that ensures the potential outperformance of a shallow network over a deep neural network (with the same number of hidden nodes), contributing on the literature focusing on the gains (in terms of out-of-sample estimation) when the depth of a neural network is increased as opposed to its width.

The chapter also reports a Monte Carlo exercise to analyze the predictive ability of the proposed methodology. In particular, by considering both linear and nonlinear data generating processes, the out-of-sample performance of a neural network with an architecture defined via the novel maximization is compared against an OLS estimator (which by definition is the best benchmark when a linear data generating process is considered) and against a neural network with the same depth, and the width of which is computed using k-folds cross-validation with randomized grid-search. To ensure the robustness of the simulation exercise, the optimization is conducted over different maximum numbers of hidden nodes allowed in the ReLu DNN. The chapter also proposes the derivation of a test statistic for the pair-wise comparison of the different approaches implemented. The simulation results show that a neural network whose architecture is identified with the proposed maximization (*optimal* neural network) is comparable in terms of predictive accuracy to a BLUE estimator when the underline data generating process is linear. Additionally, the optimal neural network is shown to outperform a neural network with a structure obtained via 3-folds cross-validation with a randomized grid-search. Additionally, by considering neural networks with different sizes (different numbers of hidden nodes), the Monte Carlo simulation results also show the existence of an optimal number of hidden nodes for a given data generating process.

Finally, the novel maximization is also evaluated empirically by focusing on the Boston Housing dataset originally studied by Harrison and Rubinfeld (1978). Being this dataset widely adopted by the literature focusing on deep learning methods, it allows comparing the performance of the novel methodology with state-of-the-art machine learning methods (see for example, Al Bataineh and Kaur, 2018; Papadopoulos and Haralambous, 2011; Granitto et al., 2001; Nado et al., 2018; Bakker and Heskes, 2003; and Zhou et al., 2001). The empirical results show that optimally selecting the neural network structure ensures an improvement in prediction accuracy (MSE) in between 28.55% and 47.32%.

1.3.3 Chapter 4

Chapter 3 proposes a constrained maximization that provides not only an alternative to the algorithm for architecture selection analyzed in chapter 2, but it also ensures maximization of the estimation accuracy. However, as previously stated, estimation accuracy is not enough; it is also necessary to provide suitable methods for the estimation of the uncertainty around the predictions of deep neural networks. For this reason, the present chapter proposes a novel algorithm for uncertainty estimation in a deep learning framework.

To provide a suitable method for the construction of prediction intervals for the output of neural networks, the chapter extends the extremely randomized trees method (Geurts et al., 2006) originally developed for random forests to construct ensembles of neural networks by means of a fixed Bernoulli mask applied prior to training. More specifically, T different sub-networks with randomized architectures (each individual network will be characterized by the same depth but different layer-wise widths) are independently trained on the same dataset. In doing so, the proposed algorithm –by introducing an additional randomization scheme to the

predictions obtained by the individual learners of the ensemble– ensures independence between the members of the ensemble while reducing the variance associated with the predictions. Following the empirical findings of Lee et al. (2015) and Lakshminarayanan et al. (2017), the proposed algorithm is expected to outperform bootstrap-based approaches not only in terms of out-of-sample accuracy but also of uncertainty estimation by ensuring independence in the ensemble of neural networks without performing data resampling.

An extensive Monte Carlo exercise is carried out to evaluate the out-of-sample performance and the empirical coverage rates of the proposed extra-neural network algorithm, of the naive bootstrap approach, and of the MC dropout proposed by Gal and Ghahramani (2016a). The different methodologies are analyzed for both deep and shallow networks, for both linear and nonlinear data generating processes, by assuming either known or unknown true neural network structure, for different dropout rates, and different significance levels. The simulation results show that the three methodologies analyzed return empirical coverage rates approximately equal to the theoretical ones for nominal values equal to 0.01 and 0.05; for prediction intervals constructed at 0.10 significance level, the extra-neural network is shown to outperform both MC dropout and bootstrap. Additionally, the simulation results show how the extra-neural network algorithm returns correct prediction intervals for different dropout rates, as opposed to the MC dropout that returns correct coverage rates only for the dropout rate that maximizes the out-of-sample accuracy.

Finally, to allow for comparability with state-of-the-art algorithms, the proposed methodology is also evaluated empirically using the experimental settings of Hernández-Lobato and Adams (2015). The empirical results show how the novel algorithm, when applied to large dimensional datasets, outperforms state-of-the-art methods such as the variational inference method of Graves (2011) and the probabilistic backpropagation of Hernández-Lobato and Adams (2015) in terms of out-of-sample accuracy.

1.3.4 Chapter 5

Chapter 5 conducts an empirical investigation of the CO_2 emissions associated with Bitcoin mining using deep learning methods and adopting the theoretical results from chapter 3 for optimal architecture identification and from chapter 4 for uncertainty estimation. The current literature focusing on the estimation of the Carbon footprint of Bitcoin mining, such as the Cambridge Bitcoin Electricity Consumption Index (CBECI) and Stoll et al. (2019), estimates the greenhouse emission starting from the seminal paper by Hayes (2017). In particular, assuming that the Bitcoin production resembles a competitive market, where risk-neutral rational miners will produce until their marginal costs equal the value of their expected marginal product, it is possible to derive the upper bound on the energy consumption associated with Bitcoin mining. Similarly, by assuming that only the most efficient miners (in terms of the number of Bitcoins mined per electricity consumed) operate in the market, it is possible to compute the lower bound on the energy consumption.

Following, the standard procedure in the literature consists of multiplicating the aggregate level of energy consumption by the weighted average of CO_2 emission factors of the countries considered to obtain the upper and lower bounds on the overall carbon footprint of Bitcoin mining (*top-down approach*). This approach has two inherently problems: (I) it does not follow the 2006 IPCC Guidelines for National Green House Gas Inventories reported in the second Volume, according to which the source of consumption must be multiplied by the corresponding emission factor when computing the emission of greenhouse gas from stationary sources. Since Bitcoin mining spans many different countries, the contribution of the miners located in each country to the overall network hashrate is needed to construct a country-specific upper and lower limits of electricity consumption that can be aggregated to a world total, (II) the uncertainty associated with the CO_2 emission from Bitcoin mining captured by the difference between Hayes (2017)'s lower and upper bound is extremely large.

The present chapter uses deep learning methods to address both problems. First, it defines the target of the deep neural network by developing a new *bottom-up* approach that, by exploiting the exact geographic location of Bitcoin miners and the relative market shares of the ASIC miners producers, is able to construct an aggregate level of CO_2 emission starting from country-specific information (in doing so, it is also possible to determine the share of clean energy –with an emission factor equal to zero– employed in Bitcoin mining). Second, it measures the uncertainty around the point predictions of the neural networks using the MC dropout proposed by Gal and Ghahramani (2016a), which ultimately narrows down the difference between upper and lower bounds of the Bitcoin mining carbon footprint.

Starting from Hayes (2017)'s insights, it is believed that the mining of Bitcoins (and thus associated electricity consumption and CO_2 emissions) is mainly driven by the current market price of the cryptocurrency. Therefore, starting from Kristoufek (2015), Liu and Tsivinsky (2018), McNally et al. (2018) and Jang and Lee (2017), the factors considered as input data range from (a) predictors of the Bitcoin price (with the exception of network-specific information), (b) factors driving investors' attention to the cryptocurrency (e.g., Bitcoin as a store of value as opposed to gold), (c) exchange rates with other currencies (is the increase in Bitcoin price due to an increase in its intrinsic value or due to a devaluation of the dollar?), (d) supply factors for the production of the ASIC mining chips, (d) macroeconomic indicators, (e) and the market attention (number of Google searches for the word "Bitcoin"). Factors associated with the blockchain network (network hashrate, and difficulty) are excluded for two main reasons: (i) the network hashrate is used to compute the target variable and as such cannot be used among the input variables (ii) the network difficulty is highly correlated with the network hashrate and as such, its inclusion could lead to the same problem.

Based on the out-of-sample accuracy, the fitted neural network (with architecture selected via the maximization proposed in chapter 3) is shown to outperform the two adopted benchmarks: random forest (Breiman, 2001) and a deep neural network with the same depth but cross-validated widths. When aggregating the daily predictions, the yearly CO_2 estimates [and associated 0.95 prediction intervals] are 2.77 $[1.98, 3.56]$ $MtCO_2$ for the year 2017, 16.08

[14.19, 17.97] $MtCO_2$ for 2018, and 14.99 [13.25, 16.73] $MtCO_2$ for 2019. To provide an order of magnitude, the estimated levels of CO_2 show how the Carbon footprint of Bitcoin mining is higher than the annual level of fossil fuel emissions of some U.S. states such as Maine, New Hampshire, Rhode Island, or South Dakota and of some smaller countries such as Bolivia, Sudan, or Lebanon (*Global Carbon Atlas*).

1.3.5 Chapter 6

Chapter 6 conducts an empirical investigation focused on the functional form existing between household specific information and pollution content of consumption in the U.S. for the year 1984 and 2012 –via the environmental Engel curves proposed by Levinson and O'Brien (2019)– considering five major pollutants: carbon monoxide (CO), sulfur dioxide (SO_2), volatile organic compounds (VOC), nitrogen oxides (NO), and particulates smaller than 10 microns (PM10).

The environmental Engel curves (ECCs) were first proposed as a structural approach to estimate the relationship between household income and pollution, holding price constant. Levinson and O'Brien (2019) construct ECCs separately for the indirect emissions of U.S. households –over the period 1984 and 2012– for the aforementioned pollutants. In particular, the authors estimate two versions of the ECCs: one based solely on income information, and another one that considers eighteen household-specific information. These authors find that the ECCs shift down and become more concave over time, implying that the mix of goods consumed by households is less polluting and that the pollution content of consumption increases with income at a decreasing rate. Secondly, the ECCs are upward sloping, showing that richer households pollute more. Lastly, the ECCs are characterized by income elasticities smaller than one implying that richer households consume a mix of goods whose consumption content is smaller for lower income households.

However, as mentioned in Levinson and O'Brien (2019), one of the main impediments associated with the correct construction and the study of the ECCs is the absence of a theoretical framework that identifies the income-pollution relationship. To overcome this limitation, the authors suggest that the ECCs should be estimated with as few restrictions as possible. As a result, after estimating the ECCs using standard OLS regression, they propose to model the income-pollution relationship using nonparametric Kernel regression models. Yet, Kernel models constitute a valuable solution when the ECCs are constructed considering only household income, due to their non-feasibility in higher dimensions (see also the curse of dimensionality by Sims, 1980).

The aim of the present chapter is to study the robustness of the results of Levinson and O'Brien (2019) to the choice of the functional form of the ECCs. Previously mentioned advances in the machine learning literature have shown that both (sufficiently large) deep and shallow neural networks –by being universal approximators– are able to accurately approximate the underlying function notwithstanding the type of nonlinearity to be learned.

Therefore, by being no longer necessary to construct an ad hoc model for the specific nonlinearity to be modeled, both shallow and deep neural networks can be considered as the least restricted nonlinear functions to be implemented.

In addition to predict nonparametrically the pollution content of household consumption, we apply the MC dropout of Gal and Ghahramani (2016a) and the extra-neural network approach implemented in chapter 4 to construct intervals around the model predictions. The fitted shallow and deep neural networks exhibit low mean square prediction errors, and most importantly, the constructed prediction intervals are characterized by accurate coverage probabilities. The estimated intervals show that the ECCs are upward sloping, have income elasticities lower than one, and are concave for all the analyzed pollutants.

1.3.6 Appendix

As mentioned in subsection 1.1, practitioners and scholars –when focusing on deep learning methods– distinguish between feedforward neural networks, convolutional neural networks, and recurrent neural networks. Both recurrent and convolutional neural networks can be considered special cases of the more general specification of feedforward neural networks. For completeness, and in order to allow the reader to understand the similarities and the differences between the different deep learning methods adopted by empirical researchers, Appendix A, B, and C provide a brief theoretical introduction to feedforward, convolutional, and recurrent neural networks focusing on their structure, characteristics, functions, advantages, and disadvantages. Additionally, three distinct empirical applications –concentrated in showing the strength, power, and versatility of each class of deep network– are also reported. In Appendix A, a deep feedforward neural network is trained in order to construct investment portfolios based on the forecasted conditional mean and volatility of the stock returns; in Appendix B, a deep convolutional neural network is implemented for glaucoma detection from fundus images, and the learning process is evaluated with the support of an expert's opinion; in Appendix C, a deep recurrent neural network is trained on "An inquiry into the Nature and Causes of the Wealth of Nations" by Adam Smith (1776) and used for text generation.

CHAPTER 2

Granger causality detection in high-dimensional systems using feedforward neural networks

Chapter Abstract

This chapter proposes a novel methodology to detect Granger causality in mean in vector autoregressive settings using feedforward neural networks. The approach accommodates unknown dependence structures between the elements of high-dimensional multivariate time series with weak and strong persistence. To do this, we propose a two-stage procedure. First, we maximize the transfer of information between input and output variables in the network to obtain an optimal number of nodes in the intermediate hidden layers. Second, we apply a novel sparse double group lasso penalty function to identify the variables that have predictive ability and, hence, Granger cause the others. The penalty function inducing sparsity is applied to the weights characterizing the nodes of the neural network. We show the correct identification of these weights for increasing sample sizes. We apply this method to the recently created Tobalaba network of renewable energy companies and show the increase in connectivity between companies after the creation of the network using Granger causality measures to map the connections.

2.1 Introduction

The concept of causality introduced by Wiener (1956) and Granger (1969) constitutes a basic notion for analyzing dynamic relationships between time series. In studying this type of statistical causality, predictability is the central issue, which is of great importance to economists, policymakers, and investors. A broad definition of Granger causality is based on detecting whether a variable or group of variables helps to reduce the mean square forecast error of a univariate or multivariate prediction, see Geweke (1982, 1984), Dufour and Taamouti (2010), and more recently, Song and Taamouti (2018) in a high-dimensional setting.

A natural parametric setting to assess for the presence of predictive ability in a multivariate setting is the family of Vector Autoregressive (VAR) models introduced by Sims (1980) in a seminal paper. The choice of this parametric approach has two inherent problems. The occurrence of overparametrization in large dimensions and the incorrect specification of the relationship between the variables in the linear VAR model if the true data generating process determining the interactions between the variables is nonlinear or, more generally, unknown. The different procedures suggested by the literature to overcome the "profligate parametrization" that can affect high-dimensional VARs are classified as dimensionality reduction and sparsity induction via convex regularizers. In the first group the literature tries to solve the overparametrization that may affect VARs by reducing the dimensionality of vector time series models such as canonical correlation analysis (Box and Tiao, 1977), factor models (Peña and Box, 1987), Bayesian models (e.g., Banbura et al., 2010; Koop, 2013), principal component analysis (Stock and Watson, 2002), and generalized dynamic factor models (Forni et al., 2000), among many other statistical techniques. The main limitation of these approaches lies on the loss of interpretability due to the transformations involved under most of the methods that make impossible to track the Granger causal interactions between the "original" multivariate time series (Géron, 2017).

Recently, the statistical and machine learning literature have instead focused on imposing sparsity in the estimated model coefficients through the use of convex loss functions such as the least absolute shrinkage and selector operator (or Lasso hereafter; Tibshirani, 1996). The primitive version of this approach reduces the dimension of the problem by deleting individual regressors. More sophisticated versions such as the Group Lasso penalty (Yuan and Lin, 2006) reduce the dimension of the problem by jointly deleting groups of variables. None of these approaches takes, however, explicit consideration of the structure of the dependence in multivariate time series processes. In particular, these methods do not consider the pivotal role that the correct specification of the order of the VARs plays in detecting Granger causal interactions. To overcome this limitation and accommodate penalty functions that explicitly consider the appropriate number of lags in the system Nicholson et al. (2014) suggest a Hierarchical Group Lasso approach that allows not only for automatic variable selection but also for automatic lag selection. Another noteworthy example of Granger causality discovery in high-dimensional linear VARs is the paper by Skripnikov and Michailidis (2019) where the authors propose a generalized sparse fused lasso optimization criterion for jointly estimating

multivariate VARs. The novel lasso-based optimization procedure developed by these authors allows not only to introduce sparsity but also to encourage similarities between transition matrices - ultimately allowing for both joint estimation and Granger causality detection in multiple VARs. The current literature has proposed robust procedures for the estimation of Granger causality in high-dimensional linear VARs by sparsity induction via convex regularizers; however, the identification of correct inferential procedures based on Granger causality testing remains not fully untangled[8]. A first step towards correct inferential procedures is proposed by Hecq et al. (2019) that extend the post-double selection approach of Belloni et al. (2014) to Granger causality testing in linear sparse high-dimensional VARs[9]. This procedure allows retaining the correct size after the variable selection of the lasso and is shown to perform well for different data generating processes.

The presence of nonlinearities in the dynamic relationship between the variables is another problem not properly studied yet in the analysis of Granger causality. Taamouti et al. (2014) propose nonparametric estimation and inference for conditional density based Granger causality measures that quantify linear and nonlinear Granger causalities. These authors transform the Granger causality measures in terms of copula densities. More recently, Song and Taamouti (2018) propose model-free measures for Granger causality in mean between random variables. Unlike the existing measures, these methods are able to detect and quantify nonlinear causal effects. The new measures are based on nonparametric regressions and are consistently estimated by replacing the unknown mean square forecast errors by their nonparametric kernel estimates.

In his seminal paper, Granger (1969) states that if it exists lagged causality from time series $\mathbf{x}_{jt}$ to time series $\mathbf{x}_{it}$, including the lagged values of time series $\mathbf{x}_{jt}$ will ensure a higher accuracy when forecasting time series $\mathbf{x}_{it}$[10]. A powerful methodology for prediction in regression models and, more specifically, forecasting multivariate time series is neural networks. Empirical researches show that Artificial Neural Networks are characterized by high accuracy when used to forecast nonlinear multivariate time series (Chakraborty et al., 1992; Kaastra and Boyd, 1996)[11]. More generally, deep learning methods based on training large neural networks have proved very successful in many high-dimensional problems such as pattern recognition, biomedical diagnosis, and others, see Schmidhuber (2015) and LeCun et al. (2015) for overviews of the topic. Athey and Imbens (2019) provide a recent literature review of applications and contributions, to and from economics and econometrics.

The main impediment for neural network models to be considered as a standard tool for

[8] Wilms et al. (2016) propose a bootstrap based Granger causality test which, however, ignores the uncertainty regarding the selection step and thus, does not account for post-selection issues.

[9] Another important example that can be found in the literature is the research conducted by Song and Taamouti (2018). The authors propose correct inferential procedures for Granger causality testing in high dimensional systems modeled by factor models as opposed to high dimensional VARs.

[10] As specified in Diebold (1997) Granger causality can be regarded as "*predictive causality*" which is an abbreviation for "$\mathbf{x}_{jt}$ *contains useful information for predicting* $\mathbf{x}_{it}$"; which is different from the more known form of causality.

[11] *Universal Approximation Theorem* by Cybenko (1989) analyzed by Hornik (1991).

time series analysis is the lack of interpretation. This is due to the fact that the effects of inputs are difficult to quantify exactly due to the tangled web of interacting nodes between and across hidden layers. There is, however, some recent progress in this area. In particular, Scardapane et al. (2017) propose a methodology that adds interpretability to neural network structures by imposing a mapping between the "original" variables and the nodes of the first hidden layer of the neural network. An "original" variable in a model is considered as irrelevant if the corresponding nodes carrying information from the variable to the neural network are pruned. Pruning nodes in the first layer of the neural network is equivalent to deleting variables from a regression model. Tank et al. (2018) are the first authors to apply this strategy to the specific time series problem of detecting Granger causality. These authors use a hierarchical group lasso penalty function (see Yuan and Lin, 2006) to the weights of the neural network; but their work remains silent on the impact of the architecture of the network on Granger causality discovery.

The aim of the current chapter is to propose a methodology based on neural networks to detect Granger causality in mean for vectors of variables in which the dynamic dependence structure is unknown and can take very general forms accommodating, in turn, linear and nonlinear VAR models with a potentially high-dimensional number of variables and lags. In contrast to most of the literature on neural networks, we add interpretability to the neural network by applying Scardapane et al. (2017)'s strategy to a time series setting. More specifically, we construct a neural network with input layer given by the vector of *regressors* and output layer given by the vector of *dependent* variables. The magnitude of the weights associated with the nodes in the first layer determines the presence of Granger causality between input and output variables. More formally, the interpretability of the network is given by the existence of a mapping between the regressors and the nodes in the first hidden layer. A particular input variable will be relevant for predicting an output variable if there are connections from the corresponding input node to any node in the first hidden layer. Granger causality of a variable involves checking the connections between all possible lags and all possible nodes in the first hidden layer, such that a variable will not Granger cause another variable if there are no connections leaving from any of the input variables to any of the nodes in the first hidden layer. In contrast, the number of nodes in the intermediate hidden layers is not directly related to the definition and interpretation of Granger causality. The relevant intermediate nodes are obtained from optimizing the flow of information from the input variables to the output variables in the neural network - maximizing the mutual information transfer/minimizing the information loss. More formally, we show that the optimal choice of the number of nodes in the intermediate hidden layers improves model selection - reduces type I and II errors in the Granger causality detection methods.

Our method allows for a large number of variables and lags. In this setting, the number of input nodes can be very large rendering standard estimation and model selection methods unfeasible. We propose instead a novel sparse double group lasso penalty function that allows for the estimation of the weights characterizing the transfer of information through the neural network and model selection - Granger causality and lag selection. Our double group lasso

penalty function considers all possible lags of a specific regressor and all possible nodes of the first hidden layer connecting to such regressor as a first group. The second group considers separately all possible nodes of the first hidden layer connecting to a specific lag of a specific regressor. This is the proposed approach to detect the optimal number of lags of a given input variable influencing each output variable.

Our sparse double group lasso penalty function extends the penalty functions proposed in Simon et al. (2013) for multivariate regression models and Scardapane et al. (2017) for neural networks. Both hierarchical and sparse group lasso procedures for detecting Granger causality allow specifying a different number of lags across variables in the vector. However, in contrast to the hierarchical group lasso, our novel objective function imposes a lower penalty function on the parameters of the model at the same time as guaranteeing model selection consistency. By doing so, we make sure we exclude those variables without the ability to predict the response variables, without excluding important interactions between the variables once a group is not deleted; that is, once a variable is shown to Granger cause another variable.

To the best of our knowledge, this - together with Tank et al. (2018) - is the first study that considers Granger causality in a very general setting - unknown dependence structure between the variables - using neural networks. Our method differentiates from Tank et al. (2018) study in two main aspects. First, we propose an optimal network structure, obtained from applying Montgomery and Eledath (1995)'s algorithm, and second, we consider a different lasso penalty function that operates differently from the hierarchical group lasso. That is, we only use in each hidden layer those nodes that carry information from the input layer to the output layer removing unnecessary nodes. The optimality of the neural network has a direct effect on the properties of our Granger causality procedure. In particular, we reduce the type I error, interpreted in this context as spurious Granger causality. An excessive number of nodes can lead to lasso type penalty functions that identify spuriously non-existing interactions among the input nodes.

The chapter also discusses results on parameter identification and model selection consistency as the sample size increases. We derive the conditions that determine the inclusion or not of a parameter or group of parameters in the model. Our conditions for model selection coincide with those found in the literature for model selection consistency when the number of variables and the number of lags are fixed; in particular, we obtain $\lambda = o(1/T)$ with T the sample size, see Fan and Li (2001). Nevertheless, our procedure also achieves model selection consistency when the number of lags k increases with the sample size. In order to guarantee this, we impose $\lambda = o\left(\frac{1}{\sqrt{k_T T}}\right)$, with k_T the number of lags of the input variables.

A comprehensive Monte-Carlo simulation exercise –that shows the performance of our methodology to detect Granger causality– is also conducted. First, we assess the type I and type II errors of our detection procedure in finite samples and compare it against a method that does not optimize the structure of the neural network. Second, we compare the performance of our proposed sparse double group lasso against the hierarchical group lasso.

Both sets of results provide clear evidence of the outstanding performance of our methodology for detecting Granger causality in terms of probability of type I and type II errors. This result holds for short and long range dependence and for linear and nonlinear VAR specifications. We also show the consistency of our approach for model selection for increasing sample sizes.

The suggested methodology is then applied to detect the interconnections between energy companies trading in the recently created Tobalaba network. The Tobalaba network is a test-net provided by the energy Web foundation (2018) that connects renewable energy companies via a blockchain platform. More specifically, we exploit recent work on social and financial networks that identifies the presence of connections in a network through the presence of Granger causality between their nodes, see Billio et al (2012) and Hecq et al. (2019). In our setting, we propose our two-stage neural network approach for detecting Granger causal relationships between the financial returns of the energy companies trading in the Tobalaba network.

The World Bank Group (2018) argues that the decentralization, disintermediation, increase in information symmetry, and cost reduction via smart contracts will allow smaller participants entering the market, increasing the number of bilateral transactions and ultimately diversifying the market structure. The objective of our application is to corroborate the World Bank Group's hypotheses by gauging the interconnectivity between energy firms before and after the introduction of Tobalaba. To do this, we construct two Granger causal networks (before and after the introduction of Tobalaba) applying to each vertex the proposed algorithm. The empirical study reveals an increase in the number of connections among the members of the Tobalaba network after the introduction of the blockchain platform. We explore the implications of our methodology for forecasting purposes. To do this, we implement Diebold-Mariano (1995) predictive ability test and find overwhelming empirical evidence supporting the outperformance in predictive ability of our approach compared to VAR models of different dimensions.

The rest of the chapter is organized as follows: Section 2.2 presents the structure of the neural network and formulates the Granger causality detection procedure in this setting. Section 2.3 discusses estimation and model selection using a two-stage procedure, based on a novel sparse double group lasso penalty function. Section 2.4 discusses parameter identification and model selection consistency when the number of lags is fixed and also when it increases with the sample size. Section 2.6 presents a Monte-Carlo simulation exercise that provides empirical evidence in finite samples of the performance of our method to detect Granger causality and model selection consistency for increasing sample sizes. In Section 2.7, we apply our novel procedure for detecting Granger causality to the financial returns of the set of renewable energy firms trading in the recently created Tobalaba network. Section 2.8 concludes.

2.2 Granger causality in neural networks

Let $\{\mathbf{x}_t \in \mathbb{R}^p\}_{t=1}^T$ denote a p-dimensional vector time series of length T, with p the number of variables defining the multivariate time series. Our goal is to study Granger causality in mean. For this, the relevant loss function is the mean square forecast error. The vector of random variables $\mathbf{x}_t$ evolves according to the following dynamics, that are defined componentwise. Thus, for each x_{it}:

$$x_{it} = g_i(\mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-k}) + \epsilon_{it}, \text{ for } i = 1, \ldots, p, \tag{2.1}$$

where $g_i(\cdot)$ is a function that captures the dependence structure between the dependent variable x_{it} and the lags of the vector $\mathbf{x}_t$. The quantity ϵ_{it} is a martingale difference sequence satisfying that $E[\epsilon_{it} \mid \Im_{t-1}] = 0$, with $\Im_{t-1}$ denoting the sigma-algebra containing all the information available to the individual at time t. We further assume that the sigma-algebra $\Im_{t-1}$ can be approximated by the finite set $\mathbf{X}_{t-1}$, with $\mathbf{X}_{t-1} = [\mathbf{x}_{t-1}...\mathbf{x}_{t-k}]$ a matrix of dimension $(p \times k)$ containing the relevant information set. Then, $\Im_{t-1} \equiv \mathbf{X}_{t-1}$ such that $E[x_{it} \mid \Im_{t-1}] = g_i(\mathbf{X}_{t-1})$. In addition, it is also implicit that $k = k_i$ as we investigate the possibility to use different lags for different components.

There are different approaches to model the function $g_i(\mathbf{X}_{t-1})$ for $i = 1, \ldots, p$. This chapter builds on the recent literature introducing techniques for adding interpretability to neural networks and proposes a feedforward neural network to model each function $g_i(\cdot)$ separately. Each feedforward neural network has N hidden layers, and the vector $\mathbf{h}_n$ denotes the values of the hidden layers obtained from z_n hidden nodes in the n^{th} hidden layer. We use $g_i(\mathbf{X}_{t-1}; {}^i\mathbf{W}, \mathbf{z})$ to denote the function $g_i(\mathbf{X}_{t-1})$. By doing so, we incorporate as additional arguments of the function the matrix ${}^i\mathbf{W} = [{}^i\mathbf{W}^1; \ldots; {}^i\mathbf{W}^N]$ that contains all the weights with the information carried through the nodes in the hidden layers for predicting the output variable x_{it}, and the vector $\mathbf{z} = (z_1, \ldots, z_N)^\top$ that contains the number of nodes in each hidden layer.

In this framework, we propose $i = 1, \ldots, p$ different neural networks to measure the relationship between each variable x_{it} and the matrix of input variables $\mathbf{X}_{t-1}$. Furthermore, each submatrix ${}^i\mathbf{W}^1$ contains the weights associated with the nodes in the first hidden layer. These weights connect the vector of input variables to the first hidden layer with z_1 nodes. To fix ideas, let us focus on the output variable x_{it}. In this case, the matrix of weights relevant for gauging Granger causality and characterizing the first hidden layer is

$${}^i\mathbf{W}^1_{(z_1 \times kp)} = \begin{bmatrix} {}^iw_{11}^{1(1)} & \ldots & {}^iw_{11}^{1(k)} & & {}^iw_{1p}^{1(1)} & \ldots & {}^iw_{1p}^{1(k)} \\ \ldots & & \ldots & \ldots & \ldots & & \ldots \\ {}^iw_{z_1 1}^{1(1)} & \ldots & {}^iw_{z_1 1}^{1(k)} & & {}^iw_{z_1 p}^{1(1)} & \ldots & {}^iw_{z_1 p}^{1(k)} \end{bmatrix}$$

For the intermediate hidden layers the matrices ${}^i\mathbf{W}^n$, with $n = 2, \ldots, N$, are constructed similarly; however, in this case the connections are between a layer of z_{n-1} nodes and a layer of z_n nodes such that the dimensions of the matrix ${}^i\mathbf{W}^n$ need to be adapted. In what follows,

we drop the superscript i and denote the matrix with the weights of the neural network as $\mathbf{W} = [\mathbf{W}^1; \ldots; \mathbf{W}^N]$.

The information in a neural network flows across layers by means of activation functions θ. Let $\widetilde{\mathbf{x}}_{t-1}$ be a vector of dimension $kp \times 1$ that stacks all the elements of the matrix $\mathbf{X}_{t-1}$ from the input variables. For a given bias parameter $\mathbf{b}_1 \in \mathbb{R}^{z_1}$ and activation vector-valued function $\theta(\cdot) : \mathbb{R}^{z_1} \to \mathbb{R}^{z_1}$, the values at the first hidden layer $\mathbf{h}_1 \in \mathbb{R}^{z_1}$ are

$$\begin{aligned} \mathbf{h}_1 &= \theta\left(\mathbf{W}^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{b}_1\right) \\ &= \theta\left(\begin{array}{c} \ldots \\ \sum_{j=1}^{p}\sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \\ \ldots \end{array}\right) \end{aligned} \tag{2.2}$$

The values of the activation functions at the intermediate hidden layers, $\mathbf{h}_n \in \mathbb{R}^{z_n}$, are given by

$$\mathbf{h}_n = \theta\left(\mathbf{W}^n \mathbf{h}_{n-1} + \mathbf{b}_n\right), n = 2 \ldots N, \tag{2.3}$$

where $\mathbf{b}_n \in \mathbb{R}^{z_n}, \mathbf{h}_{n-1} \in \mathbb{R}^{z_{n-1}}$ and hence $\mathbf{W}^n \in \mathbb{R}^{z_n} \times \mathbb{R}^{z_{n-1}}$ is the matrix of weights in hidden layer n with z_n rows and z_{n-1} columns. Since the vector-valued activation function $\theta(\cdot)$ proceeds element-wise, we denote by $\theta_z(\cdot) : \mathbb{R} \to \mathbb{R}$ with $z = 1, \ldots, z_n$, each component of it corresponding to each node in hidden layer n. Therefore, the one-period ahead forecast of the time series x_{it} is

$$g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) = \omega_O^\top \mathbf{h}_N + b_O, \tag{2.4}$$

where b_O is a constant; $\omega_O \in \mathbb{R}^{z_N}$ is the vector of weights connecting the last hidden layer N to the output node, and $\mathbf{h}_N \in \mathbb{R}^{z_N}$ is the vector of values at the last hidden layer, which can be expressed in terms of the input data as

$$\mathbf{h}_N = \theta\left(\mathbf{W}^N ... \theta\left(\mathbf{W}^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{b}_1\right) ... + \mathbf{b}_N\right) \tag{2.5}$$

where $\mathbf{W}^N \in \mathbb{R}^{z_N} \times \mathbb{R}^{z_{N-1}}$ and $\theta(\cdot) : \mathbb{R}^{z_N} \to \mathbb{R}^{z_N}$. Hence,

$$g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) = \omega_O^\top \theta\left(\mathbf{W}^N ... \theta\left(\begin{array}{c} \ldots \\ \sum_{j=1}^{p}\sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \\ \ldots \end{array}\right) ... + \mathbf{b}_N\right) + b_O. \tag{2.6}$$

Equation (2.6) expresses the function $g_i(\cdot)$ in terms of a vector-valued activation function $\theta(\cdot)$ applied to a linear combination of the input nodes. This equation adds interpretability to the neural network through the connections between the input variables and the nodes in the first hidden layer. A variable x_{jt} does not Granger-cause the variable x_{it} if all the weights connecting all the lags of x_{jt} in the model and all the nodes in the first hidden layer are zero.

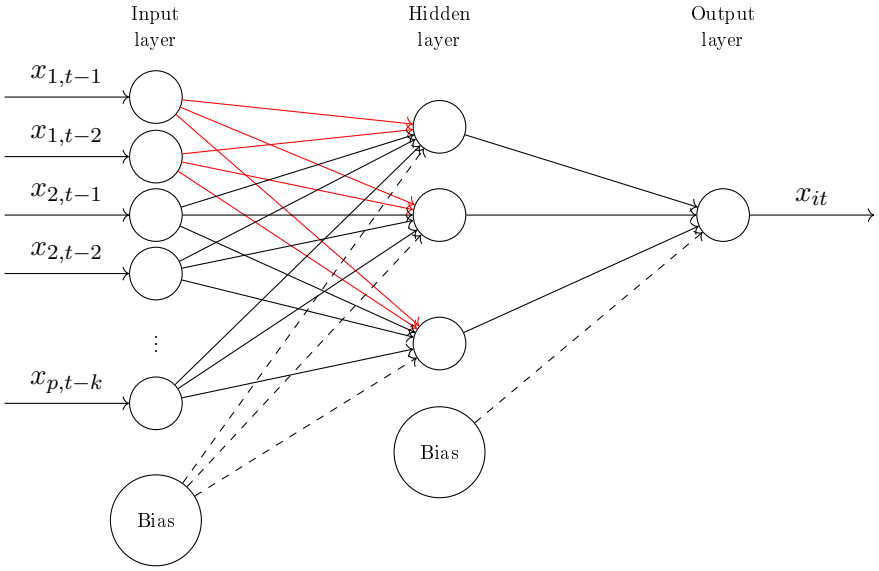


Figure 2.1: Neural Granger causality

Section 2.3 provides a formal parametric definition of Granger causality in a fully connected neural network framework. The null hypothesis of no Granger causality of x_{jt} to x_{it} is

$$\mathcal{H}_0 : w_{1j}^{1(1)} = \ldots = w_{1j}^{1(k)} = \ldots w_{z_1 j}^{1(1)} = \ldots = w_{z_1 j}^{1(k)} = 0, \tag{2.7}$$

and the alternative is

$$\mathcal{H}_A : \text{some } w_{nj}^{1(l)} \neq 0, \text{ for } n = 1, \ldots, z_1 \text{ and } l = 1, \ldots, k. \tag{2.8}$$

Figure 2.1 provides a visual representation of the intuition behind the test for Granger causality based on feedforward neural networks for a multivariate time series with one hidden layer. If the Group Lasso penalty penalizes to zero the weights highlighted in red, the variable x_{1t} does not Granger cause series x_{it}. Thus, Granger causality is inferred from the sparsity introduced in the first layer by the Group lasso penalty.

2.3 Fully Connected Neural Network

In extreme cases, it is possible that rejecting the null hypothesis (2.7) in a sparse connected deep feedforward neural network does not imply causality in the sense of Granger. To rule out this possibility, the null hypothesis (2.7) is assessed in fully connected neural networks.

The following extreme case is reported as an explanatory example of the failure of our null hypothesis to capture Granger causal relationships between a set of variables. Given three random variables defined as $\mathbf{Y}_t \in \mathbb{R}$, $\mathbf{X}_t \in \mathbb{R}^{n_x}$, and $\mathbf{Z}_t \in \mathbb{R}^{n_z}$, we define $\mathcal{S}_{t-1}$ as the informa-

tion set containing $\mathbf{Y}_{t-1}, \cdots, \mathbf{Y}_{t-k}$, $\mathbf{X}_{t-1}, \cdots, \mathbf{X}_{t-k}$, and $\mathbf{Z}_{t-1}, \cdots, \mathbf{Z}_{t-k}$, and the restricted information set $\Im_{-\mathbf{X},t-1}$ as the one consisting of $\mathbf{Y}_{t-1}, \cdots, \mathbf{Y}_{t-k}$, and $\mathbf{Z}_{t-1}, \cdots, \mathbf{Z}_{t-k}$. $\mathbf{X}_t$ does not Granger cause $\mathbf{Y}_t$ is and only if

$$\mathbb{E}\{[\mathbf{Y}_t - \mathbb{E}(\mathbf{Y}_t|\Im_{t-1})]^2\} < \mathbb{E}\{[\mathbf{Y}_t - \mathbb{E}(\mathbf{Y}_t|\Im_{-\mathbf{X},t-1})]^2\} \tag{2.9}$$

For simplicity, we will consider a neural network with a single hidden layer defined as:

$$y_t = \omega_O^\top \left[\theta\left(\mathbf{W}_x^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{W}_z^1 \widetilde{\mathbf{z}}_{t-1} + \mathbf{b}_1\right)\right] + b_O \tag{2.10}$$

where $\widetilde{\mathbf{x}}_{t-1}$ is a vector of dimension $kn_x \times 1$, $\widetilde{\mathbf{z}}_{t-1}$ of dimension $kn_z \times 1$, $\mathbf{W}_x^1 \in \mathbb{R}^{z_1} \times \mathbb{R}^{kn_x}$, $\mathbf{W}_z^1 \in \mathbb{R}^{z_1} \times \mathbb{R}^{kn_z}$, and $\omega_O \in \mathbb{R}^{z_1}$.

Assuming that the network is modeled with the sparse representation (reported also in Figure 2.2) where the input $\widetilde{\mathbf{x}}_{t-1}$ is loaded only to h_1, $\mathbf{W}_x^1$ is imposed equal to zero for all the hidden nodes h_i for $i = 2, \cdots, z_1$. Based on Equation 2.10, it is possible to distinguish the output of h_1 to y_t, from all the other hidden nodes in the first hidden layer as:

$$y_t = \boldsymbol{\omega}_{O,1}^\top \left[\boldsymbol{\theta}\left(\mathbf{W}_x^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{W}_z^1 \widetilde{\mathbf{z}}_{t-1} + \mathbf{b}_1\right)\right] + \boldsymbol{\omega}_{O,2:z_1}^\top \left[\boldsymbol{\theta}\left(\mathbf{W}_x^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{W}_z^1 \widetilde{\mathbf{z}}_{t-1} + \mathbf{b}_1\right)\right] + b_O \tag{2.11}$$

where $\boldsymbol{\omega}_{O,1} \in \mathbb{R}$, and $\boldsymbol{\omega}_{O,2:z_1} \in \mathbb{R}^{(z_1-1)}$. As the aforementioned sparse representation imposes $\mathbf{W}_x^1 \neq 0$ only for h_1, it is possible to express Equation (2.11) as:

$$y_t = \boldsymbol{\omega}_{O,1}^\top \left[\boldsymbol{\theta}\left(\mathbf{W}_x^1 \widetilde{\mathbf{x}}_{t-1} + \mathbf{W}_z^1 \widetilde{\mathbf{z}}_{t-1} + \mathbf{b}_1\right)\right] + \boldsymbol{\omega}_{O,2:z_1}^\top \left[\boldsymbol{\theta}\left(\mathbf{W}_z^1 \widetilde{\mathbf{z}}_{t-1} + \mathbf{b}_1\right)\right] + b_O \tag{2.12}$$

Being this the case, if $\boldsymbol{\omega}_{O,1} = 0$, $\mathbf{X}_t$ does not Granger cause $\mathbf{Y}_t$ even if $\mathbf{W}_x^1 \neq 0$. Therefore, one may presume that rejecting the null hypothesis (2.7) does not imply causality.

However, it is important to specify that this extreme case scenario can occur only in the case of sparse connected neural network, and that it is extremely unlikely that $\omega_{O,1} = 0$ without imposing a lasso regularization on $\mathbf{W}^n$ for $n > 1$. This is still true even when the *vanishing gradient* problem affecting deep learning is not properly accounted for, as the weight initialization is different from zero. To avoid this possibility, we embed our testing methodology in fully connected feedforward neural networks.

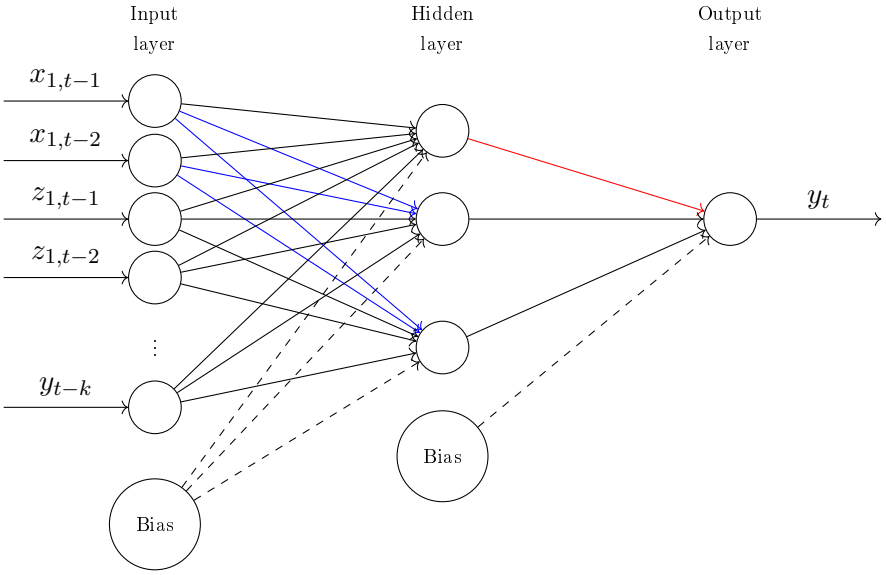


Figure 2.2: Sparse connected neural network, in blue the missing connections are reported

As the depth of the network increases, the conditions that have to occur in a sparse connected neural network, become even more unlikely. To conclude, even if in extreme and rare conditions, the authors acknowledge this eventuality, and for this reason only fully connected neural networks will be considered.

2.4 Estimation and model selection

We propose a methodology to detect Granger causality using a neural network for each element of the vector $\mathbf{x}_t$. An important aspect –that is analyzed more in details in subsection 2.4.2– is that the correct identification of the neural network structure plays a crucial role in the correct identification of Granger causal interactions. Intuitively, when performing group lasso in neural networks, the groups' dimensions are defined within the model by the analyst. By increasing (or decreasing) the number of hidden nodes in the first hidden layer, it is possible to increase (or decrease) the groups' dimension without changing the interpretation assigned to a particular group. Specifying a sub-optimal neural network structure could lead to either spurious Granger causality or missed Granger causality detection. For this reason, our procedure is done in two stages. For a given sample, we obtain first the optimal number of nodes in the feedforward neural network by minimizing the information loss across layers. We focus on the first hidden layer z_1 given that in the second stage, we plug-in the optimal quantity z_1 in a lasso type function penalizing the weights of the neural network. Minimization of the corresponding regularization problem has two objectives. First, it uncovers the input variables that are relevant for forecasting the output variables (Granger causality) and, second, it allows us to establish the optimal number of lags affecting the mean square forecast error.

Our approach differentiates from the recent literature on interpretable neural networks, see Hastie et al. (2005), Scardapane et al. (2017) or Tank et al. (2018), in two main aspects. First, in our case, concretely, as per Algorithm 1, we choose the optimal number of nodes in all hidden layers that minimize the information loss through the neural network deep architecture. This has been done in the literature by maximizing the mutual information transfer between input and output nodes, see (Schreiber, 2000) or, similarly, by minimizing the loss of information through the neural network, see de Veciana and Zakhor (1992), Montgomery and Eledath (1995), Reed et al. (1995), or more recently, Urban (2017). In order to fully exploit these theories and construct an optimal architecture for the neural network that minimizes the information loss we need to introduce uncertainty into the neural network. This is done by injecting noise into the model. In this case, the optimal neural network is constructed through noise jittering.

Second, we propose a regularization function that extends the mean square error loss function for fitting a neural network by penalizing the weights associated with the nodes in the first hidden layer. In contrast to the literature, we propose a double group lasso regularization that penalizes Granger causal relations separately across groups and lag selection within groups.

2.4.1 Stage 1: Choosing the optimal neural network

In the first stage, we optimize the neural network by choosing the number of nodes per hidden layer that maximizes the transfer of information/minimize the information loss. To do this, we follow the above literature and inject noise into the neural network. In what follows, we adapt these methods to our setting. Our base loss function is the sample mean square error, that is defined as

$$\frac{1}{T}\left\|x_{it} - g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z})\right\|_2^2. \tag{2.13}$$

In order to be able to apply the different information criteria above, we introduce noise into the system by constructing noisy replicas of our sample, as in de Veciana and Zakhor (1992), Montgomery and Eledath (1995), Reed et al. (1995) or Urban (2017). This approach can be interpreted as a procedure to regularize the neural network applied to a population and not only to a given sample. Let $x_{jt}^* = x_{jt} + v_{jt}$, with v_{jt} an *i.i.d.* realization of a $\mathcal{N}(0, \sigma_v^2)$ random variable, and let $\mathbf{X}_{t-1}^*$ be the corresponding matrix. The objective function (2.13) applied to these iid random copies of the original observations becomes

$$\frac{1}{T}\sum_{t=1}^{T}\left(x_{it} - g_i(\mathbf{X}_{t-1}^*; \mathbf{W}, \mathbf{z})\right)^2. \tag{2.14}$$

In what follows, we decompose the mean square error (2.14) into two components: a first component given by the mean square error of the original data and a second component given by introducing noise into the model. To do this, we consider first the case of a single hidden

layer $\mathbf{W} = \mathbf{W}^1$. Let the objective function be $g_i\left(\mathbf{X}^*_{t-1}; \mathbf{W}, \mathbf{z}\right) = \omega_O^\top \mathbf{h}_1^* + b_O$ with

$$\mathbf{h}_1^* = \theta \begin{pmatrix} \dots \\ \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} + \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l} \\ \dots \end{pmatrix}. \tag{2.15}$$

For simplicity, we work with the activation function element-wise, such that $g_i\left(\mathbf{X}^*_{t-1}; \mathbf{W}, \mathbf{z}\right) = \sum_{z=1}^{z_1} \omega_{Oz} \theta_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} + \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l} \right) + b_O$. Applying a Taylor expansion of first order to each activation function $\theta_z(\cdot)$ around the deterministic component $\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z}$, we obtain

$$\begin{aligned} \theta_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} + \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l} \right) &\approx \theta_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \\ &+ \dot{\theta}_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l}, \end{aligned} \tag{2.16}$$

with $\dot{\theta}_z(\cdot)$ the first derivative of $\theta_z(\cdot)$. Note that for standard activation functions proposed in the related literature, the second derivative of $\theta(\cdot)$ is close to zero along the support of the function, therefore, a first order expansion is sufficient to approximate very accurately the activation function. Then, the objective function becomes

$$g_i\left(\mathbf{X}^*_{t-1}; \mathbf{W}, \mathbf{z}\right) \approx \omega_O^\top h_1 + b_O + \sum_{z=1}^{Z_1} \omega_{Oz} \dot{\theta}_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l},$$

and the loss function (2.14) can be decomposed as

$$\begin{aligned} &\frac{1}{T} \sum_{t=1}^{T} \left(x_{it} - g_i(\mathbf{X}^*_{t-1}; \mathbf{W}, \mathbf{z}) \right)^2 \\ &+ \frac{1}{T} \sum_{t=1}^{T} \left(\sum_{z=1}^{z_1} \omega_{Oz} \dot{\theta}_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l} \right)^2 \\ &- \frac{2}{T} \sum_{t=1}^{T} \left(x_{it} - g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) \right) \sum_{z=1}^{z_1} \omega_{Oz} \dot{\theta}_z \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} v_{j,t-l}. \end{aligned}$$

Note that the elements inside the outer sum over t are independent but not identically distributed. The randomness is introduced through v_t in all cases so the mean is zero but the variance varies for each observation depending on the value of $\dot{\theta}(\cdot)$ and the weights ω_O. In this case, we can apply the law of large numbers for independent but not identically distributed

random variables, and write the preceding function as the sum of the population mean square error and an additional regularization component. More specifically, as $T \to \infty$, the previous expression converges in probability to the following population quantity:

$$E\left[(x_{it} - g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}))^2\right] + \sigma_v^2 \sum_{z=1}^{z_1} \omega_{Oz} \dot{\theta}_z^2 \left(\sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{j,t-l} + b_{1z} \right) \sum_{j=1}^{p} \sum_{l=1}^{k} \left(w_{zj}^{1(l)} \right)^2 \quad (2.17)$$

The variance of the innovation term, σ_v^2, can be interpreted as the tuning parameter of a regularization component given by the first derivative of the activation function and the magnitude of the weights.

The objective of this procedure is to minimize the noise transmitted through the neural network. This can be done in two ways: (i) minimizing the nodes operating in the linear region of the activation function and (ii) minimizing the weight values in the network. If a node is operating in the saturation region then its output will not be affected as much by the noise. Figure 2.3 illustrates the different regions. It is also clear that large weights, $\mathbf{W}$, will also amplify the noise of the output. Also, as noted by Montgomery and Eledath (1995), small weight values in the first hidden layer tend to keep nodes in the linear region so we may only want to minimize the outgoing weights from each node. Furthermore, the choice of the activation function is another factor to consider. The choice of the *tanh* function, defined as:

$$\theta(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.18)$$

guarantees that a node operating in the middle of the linear region has an average activation close to zero. In this case, removing a hidden node does not affect the training as much as under other activation functions.

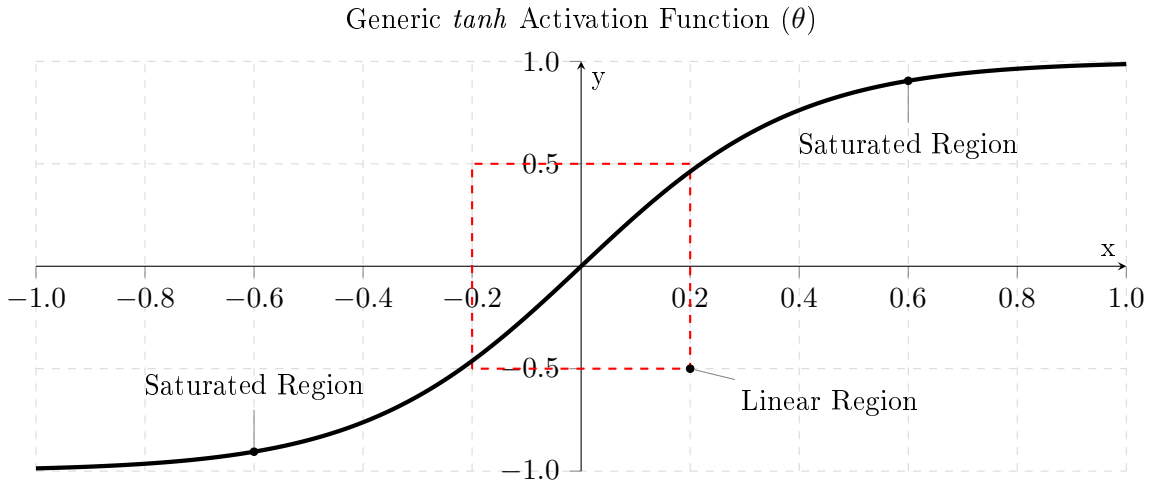


Figure 2.3: Linear and saturated regions for a generic *tanh* activation function θ

We use these arguments and focus on the significance of each node in each hidden layer rather than on minimizing formally expression (2.17). In the second stage of our procedure,

we will formally minimize the mean square error under Lasso regularization when a VAR structure is considered. In this stage, we assess, indirectly, the contribution of each node to the noise in the neural network by applying a version of the pruning algorithm called Dynamic Node Removal developed in Montgomery and Eledah (1995). This method removes hidden units as training progresses. The idea is to keep those nodes that contribute to transmitting information and delete those that transmit noise. The algorithm penalizes the nodes operating in the linear region (low confidence) of $\theta_z(\cdot)$, while accounting for the magnitude of the outgoing weights of the hidden nodes. In our setting, at each Epoch (defined as the pass that the machine learning algorithm has completed), the objective function is

$$S_{1z} = \frac{\sigma_v^2}{pkT}\sum_{t=1}^{T}\left(\kappa tanh^2\left(b_{1z} + \sum_{j=1}^{p}\sum_{l=1}^{k} w_{zj}^{1(l)} x_{jt-l}\right) + \mu\sum_{\widetilde{z}=1}^{z_2}\left(\mathbf{w}_{z,\widetilde{z}}^2\right)^2\right), \tag{2.19}$$

with $\mathbf{w}_{z,\widetilde{z}}^2$ denoting a vector of weights of dimension $(1 \times z_2)$ connecting nodes $z = 1, \ldots, z_1$ in hidden layer 1 to nodes $\widetilde{z} = 1, \ldots, z_2$ in hidden layer 2; κ and μ are tuning parameters. The objective function is standardized by dividing by the number of observations and the number of nodes in the input layer.

This function measures the *quality* of the nodes in the neural network with regards to information transfer. Thus, if the magnitude of this function is lower than a given threshold χ, then the hidden node is pruned. The choice of the *tanh* activation function over other activation functions such as the *ReLu*, *Exponential ReLu*, or *sigmoidal* ensures satisfying the *principle of minimum information loss*. As we are considering a supervised neural network, the minimization of the entropy must ensure the minimization of the output error. Penalizing nodes that operate in the linear region of $\theta(\cdot)$ (element-wise) is equivalent to penalizing nodes the output of which is approximately zero and, thus, nodes that have little impact on the final output of the feedforward neural network. However, as highlighted by Goodfellow et al. (2016) and by Géron (2017), sigmoidal activation functions saturate for high or low values of $\mathbf{h}_n$, incurring in the possible problem of exploding gradient and limiting the training of the neural network (Glorot and Bengio, 2010)[12].

Unfortunately, for neural networks comprised of more than one hidden layer, minimizing the mutual information transfer by minimizing the mean square error of the *noisy* version of the data is even more challenging. In this case, we extend the Dynamic Node Removal of Montgomery and Eledah (1995) to higher layers:

$$S_{nz} = \frac{\sigma_v^2}{z_{n-1}}\left(\kappa tanh^2\left(b_{nz} + \mathbf{W}_z^n \mathbf{h}_{n-1}\right) + \mu\sum_{\widetilde{z}=1}^{z_{n+1}}(\mathbf{w}_{z,\widetilde{z}}^{n+1})^2\right) \tag{2.20}$$

with $\mathbf{w}_{z,\widetilde{z}}^{n+1}$ a row vector of dimension $(1 \times z_{n+1})$ of matrix $\mathbf{W}^{n+1}$ connecting the node

[12] For saturated values of θ at 0 or 1 the derivative is extremely close to 0, leaving no gradient to propagate through the neural network (Géron, 2017).

$z = 1, \ldots, z_n$ in hidden layer n to node $\widetilde{z} = 1...z_{n+1}$ in hidden layer $n+1$.[13] The algorithm that we propose for pruning the neural network is detailed in Algorithm 1.

Having defined the first step, it is now possible to discuss how the variance-bias trade-off is taken into consideration. In particular, one must consider that the objective function being minimized in the first step is the mean squared error (the nodes' quality is measured at each epoch while training the neural network), ensuring an increase in the goodness of fit (decreasing the bias), and that the regularization imposed by noise jittering ensures good generalization performance by controlling the variance (avoids overfitting by training the network on a population rather than on a specific sample).

2.4.2 Stage 2: Model selection

In this section, we adapt the sparse group lasso proposed by Simon et al. (2013) to neural networks. Our penalty function extends Simon et al. (2013) by considering a double group lasso penalty function. In our case the groups are defined not only by the nodes in the first hidden layer but also by the number of lags of each input variable.

Algorithm 1 Optimal neural network - pruning method

INPUT: Vector of all input variables, Gaussian noise $v \sim \mathcal{N}(0, \sigma_v^2)$
OUTPUT: Pruned Feed Forward Neural Network that maximizes the mutual information transfer.

1: **procedure** N HIDDEN LAYER EXERCISE

2:

3: Set $\chi = 0.001$ (see Montgomery and Eledath, 1995)

4: For each epoch E, calculate the significance of the function $h_{n,z}$ as:

$$S_{1z} = \frac{\sigma_v^2}{pT} \sum_{t=1}^{T} \left(\kappa tanh^2 \left(b_{1z} + \sum_{j=1}^{p} \sum_{l=1}^{k} w_{zj}^{1(l)} x_{jt-l} \right) + \mu \sum_{\widetilde{z}=1}^{z_2} \left(\mathbf{w}_{z,\widetilde{z}}^2 \right)^2 \right), \tag{2.21}$$

if the feedforward neural network contains one hidden layer, and

$$S_{nz} = \frac{\sigma_v^2}{z_{n-1}} \left(\kappa tanh^2 \left(b_{nz} + \mathbf{W}_z^n \mathbf{h}_{n-1} \right) + \mu \sum_{\widetilde{z}=1}^{z_{n+1}} (\mathbf{w}_{z,\widetilde{z}}^{n+1})^2 \right), \tag{2.22}$$

for multilayer neural networks.

5: If $S_{nz} \leq \chi$ for $n = 1, \ldots, N$, remove h_{nz}

6: $\chi = \chi * 0.0001$

7: Repeat $4 - 6$ until $E =$ Maximum Epochs

[13] We should note that the time index is implicit in the objective function S_{nz}. All observations $\{\mathbf{x}_t\}_{t=1}^T$ are used to compute the objective function.

Sparse double group lasso penalty

The regularization component of our objective function is divided into two components. The first component is comprised by groups of size kz_1, with k the number of lags[14] and z_1 the number of nodes in the first hidden layer. This component is specific to the Granger causality detection problem. This function penalizes as a group those weights that are associated with a specific input variable and all its lags. To do this, we use the Frobenius norm that extends the L_2 norm to matrices. The penalty function for a specific input variable takes a value of zero if the Frobenius norm is zero:

$$\sum_{j=1}^{p} \left\| \mathbf{W}_j^1 \right\|_F = \sum_{j=1}^{p} \left[\sum_{z=1}^{z_1} \sum_{l=1}^{k} \left(w_{zj}^{1(l)} \right)^2 \right]^{1/2}. \tag{2.23}$$

The second component is comprised by groups of size z_1. This component is used for detecting the optimal number of lags. This function penalizes as a group those weights that are associated with a specific lag l of a given input variable. To do this, we use the L_2 norm penalizing jointly all the nodes in the first hidden layer corresponding to that lag:

$$\sum_{j=1}^{p} \sum_{l=1}^{k} \left\| \mathbf{W}_j^{1(l)} \right\|_2 = \sum_{j=1}^{p} \sum_{l=1}^{k} \left[\sum_{z=1}^{z_1} \left(w_{zj}^{1(l)} \right)^2 \right]^{1/2} \tag{2.24}$$

It is possible to illustrate the groupings determined by the first (in blue) and second component (in black) :

$$\begin{bmatrix} {}_i w_{11}^{1(1)} & \dots & {}_i w_{11}^{1(k)} & {}_i w_{12}^{1(1)} & \dots & {}_i w_{12}^{1(k)} & \dots & {}_i w_{1p}^{1(1)} & \dots & {}_i w_{1p}^{1(k)} \\ \dots & & \dots & \dots & & \dots & & \dots & & \dots \\ {}_i w_{z_1 1}^{1(1)} & \dots & {}_i w_{z_1 1}^{1(k)} & {}_i w_{z_1 2}^{1(1)} & \dots & {}_i w_{z_1 2}^{1(k)} & & {}_i w_{z_1 p}^{1(1)} & \dots & {}_i w_{z_1 p}^{1(k)} \end{bmatrix}$$

In this setting, for a given output variable x_{it}, we propose the following regularization function:

$$P(\mathbf{W}^1, z_1; \lambda, \alpha) = \lambda(1-\alpha)\sqrt{kz_1} \sum_{j=1}^{p} \left\| \mathbf{W}_j^1 \right\|_F + \lambda\alpha\sqrt{z_1} \sum_{j=1}^{p} \sum_{l=1}^{k} \left\| \mathbf{w}_j^{1(l)} \right\|_2 \tag{2.25}$$

with z_1 being the optimal number of nodes, $\sqrt{kz_1}$ and $\sqrt{z_1}$ being the square of the group dimensions of the matrix used for the detection of Granger causality, and the vector used for lag detection. Based on the above penalty function, we propose the following objective function:

$$\min_{\mathbf{W}} \left\{ \frac{1}{2T} \left\| \mathbf{Y}_i - g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) \right\|_2^2 + P(\mathbf{W}^1, z_1; \lambda, \alpha) \right\} \tag{2.26}$$

[14] We allow for different number of lags across input variables x_{jt}.

where $\mathbf{Y}_i \equiv [\mathbf{x}_{i,1}...\mathbf{x}_{i,T}]$. Expression (2.26) has a 'sparse double group lasso' penalty because it contains features of both sparse lasso and group lasso penalty functions. The above discussion clearly shows that both penalty terms differently penalize different groups of variables. The second component introduces sparsity in the first component by adding shrinkage in each of the vectors comprising the matrices $\left\|\mathbf{W}_j^1\right\|_F$ before checking if all the parameters in the matrices are zero.

The level of sparsity induced by the group lasso depends on the level of λ; the higher the λ, the lower the number of groups selected. The parameter $\alpha \in [0, 1]$ is a tuning parameter that, similarly to λ, defines the level of sparsity induced into the system. When $\alpha = 0$, the sparse double group lasso reduces to the group lasso. It is also possible to notice the relation that subsists between the adapted sparse group lasso and the hierarchical group lasso when the number of lags is one and there is no lag selection. In particular, imposing $\alpha = 0$ suggests that no lag selection should be performed and thus that $k = 1$, implying equivalence between the objective function (2.26) and the hierarchical group lasso proposed by Nicholson et al. (2014) and Tank et al. (2018), as conveyed by the function (2.32) below.

In our context, it is important to note that the dimension of the groups, when the group lasso is applied in a feedforward neural network, is a quantity that is determined within the model. More specifically, the number of groups in the first hidden layer is established in stage 1 through the mutual information optimization procedure. The quantity z_1 is obtained from this stage. Given the soft thresholding of the group lasso, a higher number of nodes will lead to a lower level of sparsity for a fixed level of λ. In this setting it is important to choose a suitable number of groups in the first hidden layer. Otherwise, a suboptimal choice $\underline{z}_1$ different from z_1 introduces two effects. First, if $\underline{z}_1 > z_1$, there are more nodes than needed and some of them do not carry information between the input variables and the output variables. In this case, the Granger causality procedures based on neural networks, see Tank et al. (2018), will lead to spurious Granger causality as a result of the activation of more nodes than necessary, increasing the type I error. A second effect is due to introducing additional terms in the regularization component of the objective function $P(\mathbf{W}^1, \underline{z}_1; \lambda, \alpha)$ given by the difference between $\underline{z}_1$ and z_1. This effect can increase the severity of the penalty and lead to delete weights that are indeed relevant in forecasting the output variable $x_{i,t}$. In this case, the suboptimal choice of the number of nodes in the first hidden layer can lead to an increase in type II error; spuriously rejecting that a given variable Granger-cause another one, when in fact it does. These effects will be analyzed in a simulation study.

Before introducing the algorithm for the detection of Granger causality we discuss the role of deeper layers on the correct detection of Granger causality. As the group lasso depends on the size of z_1 and not on the width of deeper architectures, it is expected that the layer wise widths of z_N - for $N > 1$ - will not impact on the correct detection of Granger causality. Intuitively, when fitting a deep neural network, an increase in the number of hidden nodes in the first hidden layer leads to an increase in the assumed interactions among the different input nodes (and thus in the group sizes); conversely, an increase in depth leads to an improvement

in the fit of the network by increasing the number of nonlinearities captured by the network, without affecting the assumed interactions among input nodes. Therefore, the weights in the first layer take care of the potential relationships (Granger Causal interactions) between the variables in the system and the weights in higher layers introduce further flexibility into the model and improve the goodness of fit.

Algorithm 2 Algorithm for the Detection of Granger Causality

INPUT: Dependent and Independent variables.
OUTPUT: Predicted dependent variable, and Granger causal relations.

procedure BASED ON ALGORITHM 1

Initialize an over-specified Deep Neural Network.

Definition of the Optimal ν^2.
Divide the dataset in training and test set.
Given the structure of the network, cross-validate the optimal ν^2.

Definition Optimal Structure of the Network
$\chi = 0.001$.
Set E = Maximum Epochs.
while (Epoch $<$ E) **do**
    Generate $v \sim \mathcal{N}(0, \sigma_v^2)$
    Calculate node significance applying Equation 2.22.
    Remove insignificant nodes.
    Update χ.
Algorithm 1 will return the pruned Neural Network

Granger Causality x
Define the number of hidden nodes and layers from previous steps.
Fit the feedforward neural network with objective function 2.26.

Another approach to understanding the impact that deeper hidden layers have on the estimation of Granger causality starts from the research of Lee et al. (2015). In their work, the authors propose to model multivariate or univariate time series by estimating a linear specification which is then augmented by a neural network to capture any remaining nonlinearity. Therefore, the novel methodology starts by modeling the time series with a linear model to which –depending on the data– nonlinear elements are incrementally added. Similarly, when analyzing the methodology proposed in the present chapter, the relevant layer for the detection of Granger causality is the first hidden layer to which it is possible to incrementally add hidden layers that –depending on the underlying data generating process– increase the out-of-sample estimation accuracy of the neural network.

Algorithm for the Detection of Granger Causality

In what follows, we present the algorithm that implements the methodology above for the detection of Granger causality when a feedforward neural network is fitted.

By applying Algorithm 2, this chapter extends the current literature about Granger causality discovery via neural networks by defining a new objective function that allows not only the discovery of the Granger causal interactions but also of the optimal lag length. The combination of these two aspects should ensure low type I and type II errors when testing for Granger causality[15].

The following subsection reviews two alternative penalty functions recently proposed in the neural network literature introducing lasso penalty functions for model selection.

2.4.3 Interpretable neural networks

Advances in neural networks allow us to propose a feedforward neural network for the detection of Granger causality in large systems. The main difference with previous models based on neural networks is the possibility of interpreting the intermediate steps in the making of the model predictions. To do this in a Granger causality setting, we interpret the connections between the nodes in the first hidden layer and the input variables. The interpretability of the neural network is made formal by adapting lasso type regularization functions to a neural network setting. Rather than penalizing the parameters of standard regression models, we propose a model that penalizes the weights in the nodes of the first hidden layer of the neural network. The absence of Granger causality is interpreted as a lack of connections between a given input variable and the set of nodes in the first hidden layer.

Interpretable neural networks are briefly discussed in Hastie et al. (2005), in more detail in Scardapane et al. (2017) and adapted to the detection of Granger causality using a hierarchical lasso penalty function in Tank et al. (2018). To provide a suitable background to our proposed regularization function above, we discuss in this section the regulation functions proposed in these pioneering studies adapted to our VAR setting. In this way, we can compare our novel objective function to the related literature.

[15] Another important factor that could impact on the interpretation of Granger causality from a neural network relies on the correct combination of activation function and weight initialization. The exploding gradient problem can reduce the efficacy of the group lasso detection of Granger causality by limiting the training of $\mathbf{W}^1$. Similarly, if the rectified linear unit activation function is used, the "*dying ReLu*" problem could lead to spurious node selection. The impact of the correct engineering of the feedforward neural network in terms of the combination of activation function and weight initialization is beyond the scope of this chapter.

Scardapane et al. (2017): These authors propose the following penalty function:

$$P(\lambda, \mathbf{W}) = \lambda \sum_{i=1}^{p} \sqrt{kz_1} \sum_{j=1}^{p} \left\| \mathbf{W}_j^1 \right\|_F + \lambda \sum_{i=1}^{p} \sum_{n=2}^{N} \sum_{z=1}^{z_{n-1}} \sqrt{z_n} \left\| \mathbf{W}_z^n \right\|_2 + \\ + \lambda \sum_{n=1}^{N} \sqrt{z_n} \left\| \mathbf{b}_n \right\|_2 + \lambda \left\| \mathbf{W} \right\|_1 \tag{2.27}$$

This penalty term weights 'equally' each component of the penalty by $\lambda > 0$.[16] The first component:

$$\sum_{j=1}^{p} \left\| \mathbf{W}_j^1 \right\|_F = \sum_{j=1}^{p} \left[\sum_{z=1}^{z_1} \sum_{l=1}^{k} \left(w_{zj}^{1(l)} \right)^2 \right]^{1/2} \tag{2.28}$$

is identical to the first component of the penalty function (2.26), penalizing the coefficients of the input layer ($n = 1$) across lags and nodes for each time series j. In our framework, there are two 'groups' of size kz_1 and z_1, respectively.

The second and third components penalize the 'adaptable features' of the neural network, $\{\mathbf{W}^n, \mathbf{b}_n\}_{n=1}^{N}$. The second function penalizes the vector of all outgoing connections from each node z_{n-1} in each hidden layer $n \neq 1$ such that

$$\sum_{n=2}^{N} \sum_{z=1}^{z_{n-1}} \sqrt{z_n} \left\| \mathbf{W}_z^n \right\|_2 = \sum_{n=2}^{N} \sum_{z=1}^{z_{n-1}} \sqrt{z_n} \left(\sum_{\tilde{z}=1}^{z_n} (\mathbf{w}_{\tilde{z}z}^n)^2 \right)^{1/2} \tag{2.29}$$

for each time series i. Intuitively, this corresponds to column-wise penalization of column vectors of matrix $\mathbf{W}^n, n \neq 1$. The third term penalizes the biases $\{\mathbf{b}_n\}_{n=1}^{N+1}$, where $N+1 \equiv O$ (output node), of the neural network across layers n:

$$\sum_{n=1}^{N} \sqrt{z_n} \left\| \mathbf{b}_n \right\|_2 = \sum_{n=1}^{N} \sqrt{z_n} \left(\sum_{z=1}^{z_n} b_{zn}^2 \right)^{1/2} \tag{2.30}$$

Finally, the fourth term penalizes the absolute value of the coefficients of the matrix $\mathbf{W} = [... \left[{}^i\mathbf{W}^1 \ldots {}^i\mathbf{W}^N \right] ...]$ for all time series i, i.e:

$$\left\| \mathbf{W} \right\|_1 = \sum_{i=1}^{p} \sum_{n=1}^{N} \sum_{z=1}^{z_{n-1}} \sum_{j=1}^{p} \sum_{l=1}^{k} \left| {}^i w_{zj}^{n(l)} \right| \tag{2.31}$$

Importantly, it is this last constraint that does not allow Scardapane et al.'s (2017) optimization problem to 'decouple' across the rows of the output variable $\mathbf{x}_t$, and be solved 'in parallel'. That is, we propose a different neural network for each of the p output variables in the system. For given network architectures, Farrell et al. (2018) obtain conditions for valid inference over $(\mathbf{W}, \mathbf{b})$ in non-regularized feedforward neural networks.

[16] Scardapane et al. (2017) argue that after experimenting with simulation results, weighting the components differently does not make a difference.

Hierarchical group lasso in neural networks: Tank et al. (2018) also base their definition of Granger causality on the invariance of the neural network to x_{jt}. In particular, these authors adapt a hierarchical group lasso objective function previously proposed for high-dimensional linear VAR models by Nicholson et al. (2014). Tank et al. (2018) propose a hierarchical group lasso function that allows for Granger causality detection and also for automatic lag selection. The objective function is

$$\min_{\mathbf{W}} \frac{1}{2T}||\mathbf{Y}_i - g_i(\mathbf{X}_{t-1};\mathbf{W})||_2^2 + \lambda \sum_{j=1}^{p}\sum_{l=1}^{k} ||\mathbf{w}_j^{1(l:k)}||_F, \tag{2.32}$$

where $\mathbf{w}_j^{1(l:k)} = [w_{ij}^{1(l)}...w_{ij}^{1(k)}]$. This is a hierarchical group penalty in the sense that if $\mathbf{w}_{ij}^{1(l:k)} = \mathbf{0}$ then for all $l' > l, \mathbf{w}_{ij}^{1(l':k)} = \mathbf{0}$. We use $g_i(\mathbf{X}_{t-1};\mathbf{W})$ in (2.32) to differentiate from our multilayer perceptron function $g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})$ that chooses the number of nodes z_n strategically. In the hierarchical group lasso setting, the quantity $\mathbf{z}$ is a vector of nuisance parameters that is taken as given in the optimization problem.

Therefore, the methodology proposed by Tank et al. (2018) differs from the methodology introduced in this chapter not only in terms of the regularization considered, but also in terms of the identification of the parameters affecting Granger causality via a feedforward neural network. In particular, as discussed previously, the underestimation or overestimation of the number of hidden nodes in the first hidden layer, due to the exogenous dimension of $\mathbf{h}_1$, can lead to an increase in either aggregate type I or type II errors. Also, expression (2.26) allows performing a lag selection strategy similar to Tank et al. (2018) but with a lower level of penalty given by not using the hierarchical structure. For each group, the optimal lag will be identified by the highest nonzero lag length l'. Lag lengths higher than l' will have the L_2 norms equal to zero and can be considered jointly non-significant.

2.5 Parameter identification and model selection

The aim of this section is to assess the correct identification of the parameters characterizing the objective function (2.26) under the null hypothesis of no Granger causality. To do this, we explore the conditions obtained from our objective function that leads us to delete irrelevant weights (nodes and input variables). We consider these conditions and, in particular, the role of λ and α in introducing sparsity into the regularization problem.

There are kz_1p parameters in the objective function (2.26) indexed by $w_{zj}^{1(l)}$, with $z = 1,\ldots,z_1$, $j = 1,\ldots,p$ and $l = 1,\ldots,k$. These parameters constitute a group, denoted by the matrix $\mathbf{W}_j^1$, of size kz_1. The objective function is convex implying that the solution $\widehat{\mathbf{W}}_j^1$ to the minimization problem is characterized by the first order conditions of the problem. These conditions are given by:

$$\frac{1}{T}\frac{\partial g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})}{\partial \mathbf{W}_j^1}\left(\mathbf{Y}_i - g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})\right) = \lambda(1-\alpha)\sqrt{z_1 k}\mathbf{u}_1 + \lambda\alpha\sqrt{z_1}\mathbf{u}_2, \tag{2.33}$$

where $\frac{\partial g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})}{\partial \mathbf{W}_j^1}$ is the first derivative of the function $g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})$ with respect to the parameters in matrix $\mathbf{W}_j^1$; $\mathbf{u}_1 = \frac{\widehat{\mathbf{W}}_j^1}{\|\widehat{\mathbf{W}}_j^1\|_F}$ if $\widehat{\mathbf{W}}_j^1 \neq 0$, and $\mathbf{u}_1$ is a matrix inside a unit ball such that $\|\mathbf{u}_1\|_F \leq 1$, if $\widehat{\mathbf{W}}_j^1 = 0$. The definition of $\mathbf{u}_2$ is similar but replacing the matrix $\mathbf{W}_j^1$ by the vector $\mathbf{w}_j^{1(l)}$ and the Frobenius norm by the L_2 norm.

The null hypothesis $\mathcal{H}_0$ of no Granger causality of x_{jt} to x_{it} corresponds to $\mathbf{W}_j^1 = 0$. The corresponding estimate from the objective function (2.26) must be zero in order for the lasso penalty to delete the parameter. The first order conditions with $\widehat{\mathbf{W}}_j^1 = 0$ must satisfy the condition

$$\frac{1}{T}\left\|\frac{\partial g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})}{\partial \mathbf{W}_j^1}\left(\mathbf{Y}_i - g_i(\mathbf{X}_{t-1};\mathbf{W},\mathbf{z})\right)\right\|_F \leq \lambda(1-\alpha)\sqrt{z_1 k} + \lambda\alpha\sqrt{z_1}. \tag{2.34}$$

This inequality shows the contribution of λ and α to the condition that keeps a group inactive, that is, the condition that allows us to assume $\mathbf{W}_j^1 = 0$, and hence, not rejecting the null hypothesis that the variable x_{jt} does not Granger cause x_{it}.

For problems in which the number of lags k is fixed, it is sufficient to impose $\lambda = o(1/T)$ in order for this condition to be satisfied for increasing sample sizes, see Fan and Li (2001). In this scenario, our model selection strategy is consistent; that is, it deletes those weights that do not influence the output variables. For high-dimensional problems in which the number of lags also grows to infinity with the sample size, $k = k_T$, we must impose a tighter convergence of the tuning parameter λ. In our problem it is sufficient to have $\lambda = o\left(\frac{1}{\sqrt{k_T}T}\right)$, with $k_T/T \to 0$. Alternatively, we can assume that α also converges to zero such that for high-dimensional problems, we have $\lambda = o(1/T)$ and $1-\alpha = o(1/\sqrt{T})$. These two conditions are sufficient to guarantee the correct selection of the parameters as $T \to \infty$. In practice, for a given sample size, these parameters are optimized by cross-validation.

The first order conditions of the objective function (2.26) can also give some insight into the sparsity of the vector $\mathbf{w}_j^{1(l)}$ within the matrix $\mathbf{W}_j^1$ when some of the elements of the matrix are nonzero. In this case, the group corresponding to the input variable x_{jt} is not rejected and the question of interest is to select those lags influencing the forecast of x_{it} and, by doing so, the optimal number of lags that should be included in the model. In this case, sparsity is given by subsets of parameters in the matrix $\mathbf{W}_j^1$ that are actually zero. The aim of our objective function is to be able to delete these parameters.

More formally, if $\mathbf{W}_j^1 \neq 0$, the corresponding first order conditions of the objective function (2.26) if $x_{j,t-l}$ does not influence x_{it} must satisfy, for $\widehat{\mathbf{w}}_j^{1(l)} = 0$, the following condition:

$$\left\| \frac{\partial g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z})}{\partial \mathbf{w}_j^{1(l)}} \left(\mathbf{Y}_i - g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) \right) \right\|_2 \leq T\lambda\alpha\sqrt{z_1}. \tag{2.35}$$

In this case it is sufficient to assume $\lambda = o(1/T)$ and α constant for the model to delete those parameters that are zero within a larger group and achieve model selection consistency.

In this setting, there is another source of sparsity within groups. Thus, we can consider parameters that are zero within the group of parameters that determine the relevance of a lag. More specifically, we can have $\mathbf{w}_j^{1(l)} \neq 0$ but some parameters of this vector being equal to zero. To address this case, a possibility is to extend the objective function (2.26) to include a further penalty function $\left\|\mathbf{W}^1\right\|_1$ inducing sparsity at the individual level. Although this additional penalty would allow us to correctly detect those weights that are zero if the vector $\mathbf{w}_j^{1(l)}$ is at least partially nonzero, this would increase the computational complexity of the method. More importantly, the marginal benefit of including those terms would be negligible from the point of view of model selection since the parameters that would be rightly identified as zero would correspond to connections between input nodes and specific nodes in the first hidden layer. The interpretation of these connections is not important once we accept that for some nodes in the first hidden layer the weights are different from zero, $\mathbf{w}_j^{1(l)} \neq 0$, and, therefore, carry information relevant for Granger causality and lag selection. For this reason, we do not pursue the identification of these parameters further.

Finally, we should mention that in contrast to lasso penalty functions expanding least squares procedures and well-behaved likelihood functions, see Yuan and Lin (2006), Zhou and Zhu (2010), Simon et al. (2013), Nicholson et al. (2014), among other leading examples, the objective function (2.26) is highly nonlinear due to the presence of nonlinear activation functions θ in each hidden layer that characterizes the function $g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z})$. This implies that it is not possible to derive in closed form the estimates of the weights different from zero characterizing the objective function (2.26).

2.6 Simulation Study

This section is divided into three blocks. First, we assess the probability of type I and type II errors of our procedure for detecting Granger causality in finite samples and compare it against a detection method that does not optimize the structure of the neural network. More specifically, we aim to report the fraction of times the null hypothesis of no Granger causality is rejected when there is no Granger causality in the data generating process (type I error). For the power analysis, we aim to report the fraction of times the null hypothesis is rejected when there is indeed Granger causality in the data generating process (1 - prob. type II

error)[17]. To assess the impact of the structure of the network on the type I and type II error probabilities, we impose $\alpha = 0$ in our objective function (2.26). By doing so, we restrict to a single group lasso penalty function that only focuses on detecting Granger causality without considering the optimal number of lags. Second, we relax the assumption of $\alpha = 0$, and we compare the performance of the newly proposed two step procedure with sparse group lasso against the hierarchical group lasso that does not optimize the structure of neural network (Tank et al., 2018). Last but not least, we assess the consistency of our approach for model selection when the sample size increases.

In this simulation experiment we focus on testing hypothesis (2.7), that is, Granger causality of variable x_{jt} on variable x_{it}. Our method develops a feedforward neural network for each output variable separately. For simplicity, we consider one output variable, x_{1t}, that is assumed to be endogenous and the remaining variables $x_{2t}, \ldots, x_{pt}$ are exogenous variables exhibiting different levels of persistence. The hypothesis of Granger causality (2.7) on x_{1t} is tested for each of the p variables in the system. By doing so, we are able to control for the structure of the neural network and simplify the simulation design.

2.6.1 Simulation design

The simulation experiment is conducted for three different data generating processes: a linear process exhibiting short-range persistence, a nonlinear VAR model exhibiting short-range persistence and a linear process exhibiting long-range persistence. The three models are

$$x_{1t} = a^{(10)} + \mathbf{a}^{(11)}\mathbf{x}_{t-1} + \epsilon_{1t}; \tag{2.36}$$

$$x_{1t} = a^{(10)} + \mathbf{a}^{(11)}\mathbf{x}_{t-1} * \tau_{t-1} + \epsilon_{1t}; \tag{2.37}$$

and

$$x_{1t} = a^{(10)} + \mathbf{a}^{(11)}\mathbf{x}_{t-1} + ...\mathbf{a}^{(1k)}\mathbf{x}_{t-k} + \epsilon_{1t}. \tag{2.38}$$

[17] The main difference with standard size and power exercises is that our method does not rely on a critical value provided by an asymptotic or simulated distribution but by a sparsity induction method. Hence, in contrast to standard testing methods, the simulated probability of type I error of our method may be close to zero.

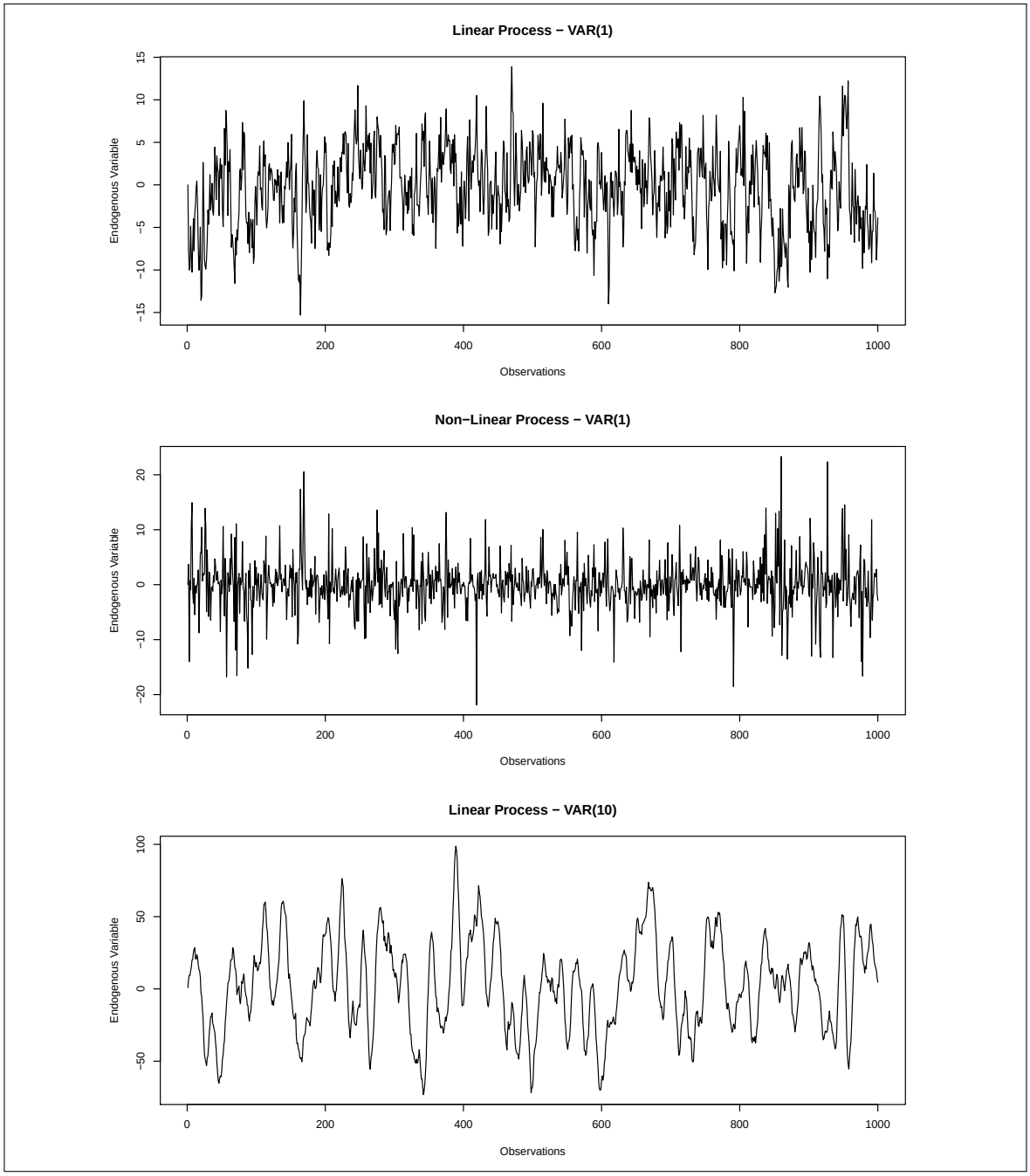


Figure 2.4: Dynamics of x_{1t} for one random sample of each of the three data generating processes considered. Top row for model (2.36), middle row for model (2.37) and bottom row for model (2.38).

The remaining variables x_{it} for $i = 2, \ldots, p$ are exogenous and exhibit autoregressive dynamics modeled as:

$$x_{it} = a^{(i0)} + a^{(i1)} x_{i,t-1} + \epsilon_{it}. \tag{2.39}$$

with the error terms ϵ_{it} being cross-sectionally, serially uncorrelated, and distributed as $\mathcal{N}(0, 1)$.

The vector $\mathbf{x}_{t-1}$ is of dimension $p = 39$; each $a^{(1,l)}$ is a $1 \times p$ vector for $l = 1, \ldots, k$ that captures the dependence structure in the linear model. The model that reflects short-range

dependence corresponds to $k = 1$ and long-range dependence corresponds to $k = 10$. For simplicity, in the latter case, we consider $\mathbf{a}^{(1,1)} = \cdots = \mathbf{a}^{(1,10)}$. Our proposed approach allows us to simultaneously assess the type I and type II error probabilities of the Granger causality detection method. The first element of $\mathbf{a}^{(1,1)}$, denoted as $a_1^{(1,1)}$, captures the persistence of x_{1t} over time; the elements $a_j^{(1,1)}$ for $j = 2, \ldots, 20$ are associated with the variables that exhibit Granger causality on x_{1t}. The coefficients are defined as follows; $a_1^{(1,1)} = 0.80$ and $\mathbf{a}_j^{(1,1)} = \{0.70, -0.30, -0.50, 0.87, 0.97, -0.65, 0.40, 0.23, -0.85, 0.76, -0.12, 0.54,$
$0.69, -0.79, 0.90, -0.90, 0.63, 0.88, 0.50\}$ for $j = 2, \ldots, 20$. These coefficients are obtained randomly from a uniform $U(-1, 1)$ distribution and guarantee the stationarity of the output variable x_{1t}. The remaining 19 parameters corresponding to $a_j^{(1,1)}$ for $j = 21, \ldots, 39$ are zero and are associated with the variables that do not Granger cause x_{1t}. In this way, our testing procedure for the hypothesis test (2.7) assesses the type II error of the test when it is applied to each of the first 19 variables, plus $x_{1,t-1}$, and assesses the type I error of the test when it is applied to the remaining 19 input variables.

We consider four different scenarios for modeling the dynamics of the exogenous variables. These scenarios are given by *i*) no persistence (white noise for all input variables), *ii*) autoregressive persistence for $j = 2, \ldots, 20$ (Granger causal variables) and no persistence for $j = 21, \ldots, 39$ (no Granger causal variables), *iii*) no persistence for $j = 2, \ldots, 20$ and autoregressive persistence for $j = 21, \ldots, 39$, and *iv*) autoregressive persistence for both sets of variables indexed by $j = 2, \ldots, 20$ and $j = 21, \ldots, 39$. In what follows, we report the results for scenarios *ii*) and *iv*); the other two scenarios are very similar and are available from the authors upon request. The autoregressive parameters of the input variables $\{x_{jt}\}_{j=2}^{p=20}$ exhibiting Granger causality are randomly drawn from a Uniform distribution $U \sim (0, 1)$ such that for each variable x_{jt} we have $a^{(j,1)} = \{0.63, 0.67, 0.65, 0.60, 0.71, 0.54, 0.63, 0.61, 0.85, 0.69, 0.68, 0.88, 0.78,$
$0.59, 0.58, 0.89, 0.71, 0.66, 0.69\}$. Case *ii*) above assumes that the last 19 variables are random noise processes generated from a $\mathcal{N}(0, 1)$ random variable. Case *iv*) considers, instead, the same autoregressive dynamics for the last 19 variables as for the first 19 variables. By doing so, this case allows us to compare the type I and type II error probabilities of the detection procedures for a given persistence parameter, and repeat this exercise for 19 different input variables.

For the nonlinear case, we consider the VAR(1) model (2.38). The dependence structure in this scenario is completed by including the interaction between the lags of $\mathbf{x}_t$ and a random variable τ_t. We assume this variable to be *i.i.d.*, following a normal distribution $\mathcal{N}(0.5, 1)$, and independent of the error term in Equation 2.37. As in the linear case, we will entertain separately the case of white noise input variables (scenario *ii*) above) and autoregressive input variables (scenario *iv*) above). By doing so, we can assess the ability of our testing procedure to detect Granger causality under nonlinearity and different dynamics in the data.

The choice of parameters above ensures the stationarity of the variable x_{1t} across data generating processes. Figure 2.4 reports an example of the different simulated processes.

2.6.2 Empirical type I and type II error probabilities

We study first the impact of the correct specification of the structure of the neural network on the probability of the type I error and the corresponding power analysis (1-prob. type II error) of the testing procedures, by focusing on linear and nonlinear processes with short persistence ($k = 1$). From the data generating process described by Equation (2.36) and (2.37), it is possible to see that no interaction subsists between the x_{it} for $i = 2, \ldots, p$. From Equation (2.2), one could notice that this particular data generating process is correctly represented by the input of the vector-valued activation function $\theta(\cdot)$ when $\mathbf{W}^1$ is a vector of dimension $(1 \times p)$ and thus, when $z_1 = 1$.

Once the structure of the neural network is identified, to study the impact of the correct specification of the structure of the network on Granger causality discovery, it is necessary to isolate the group discovery from lag discovery. Knowledge of the data generating process allows us using the correct regularization function (2.26) that only penalizes for Granger causality and not for the number of lags. This is possible in (2.26) by considering $\alpha = 0$. In this case, our proposed objective function is a single group lasso penalty function. This restriction allows isolating the impact of the structure of the network on the performance of the group lasso applied on the weights that connect the input layer to the first hidden layer. In order to study the impact of the structure of the neural network, the different structures reported in Table 2.1 will be analyzed.

Table 2.1: p denotes the dimension of the vector $\mathbf{x_t}$; No GC denotes the number of variables that do not Granger cause x_{1t}. $\underline{z}_1$ and $\underline{z}_2$ denote the starting values of the number of nodes in the two hidden layers.

Linear
$X_t = A_0 + A_1 X_{t-1} + \epsilon_t$

VAR(1)				**VAR(10)**	
p	20	p	20	p	20
No GC	19	No GC	19	No GC	19
$\underline{z}_1$ and $\underline{z}_2$	[1,1]	$\underline{z}_1$ and $\underline{z}_2$	[2,1]	$\underline{z}_1$ and $\underline{z}_2$	[5,10]
p	20	p	20		
No GC	19	No GC	19		
$\underline{z}_1$ and $\underline{z}_2$	[1,2]	$\underline{z}_1$ and $\underline{z}_2$	[5,10]		

Non Linear
$X_t = A_0 + A_1 X_{t-1} * \tau_{t-1} + \epsilon_t$

VAR(1)				**VAR(10)**	
p	20	p	20	p	20
No GC	19	No GC	19	No GC	19
$\underline{z}_1$ and $\underline{z}_2$	[1,1]	$\underline{z}_1$ and $\underline{z}_2$	[2,1]	$\underline{z}_1$ and $\underline{z}_2$	[5,10]
p	20	p	20		
No GC	19	No GC	19		
$\underline{z}_1$ and $\underline{z}_2$	[1,2]	$\underline{z}_1$ and $\underline{z}_2$	[5,10]		

The parameters in Table 2.1 determine the data generating processes but also the architecture

of the neural network. We fix the number of intermediate hidden layers equal to two[18] and give different starting values $\underline{z}_1$ and $\underline{z}_2$ to the number of nodes in each hidden layer. In order to assess the performance of our methodology for detecting Granger causal interactions, we propose a Monte Carlo simulation exercise to compute the empirical probability of the type I error and the corresponding power analysis for different sample sizes ($T = 500, 1000$). We should note that considering small samples such as $T = 100$ is not feasible in our simulation exercise due to the large number of regressors entertained in the data generating processes and the choice of a lasso penalty function. To obtain the finite-sample probability of the type I error and the corresponding empirical power of our testing procedure, we simulate $B = 100$ iterations of the above data generating processes with the parameters introduced in Table 2.1. For each iteration, we compute a neural network and decide whether there is Granger causality from the variables x_{jt} for $j = 1, \ldots, p$ on the output random variable x_{1t}. This simulation procedure is repeated B times to obtain the fraction of times the null hypothesis $\mathcal{H}_0$ in (2.7) is rejected. We construct the variable D_{jb} that takes a value of zero if the variable x_{jt} does not Granger cause the random variable x_{1t}, and one otherwise. The variable $\bar{D}_j$ is the fraction of time out of B simulations that the null hypothesis is rejected.

Table 2.1 shows that our data generating processes are characterized by 20 variables exhibiting Granger causality, with the first one being an autoregressive component, and the remaining 19 variables being exogenous. By doing so, our experiment studies simultaneously 19 exercises, given by 19 variables with different persistence parameters, for assessing the probability of type I error and 19 exercises for assessing the empirical power (1- probability of type II error). There is an additional exercise that tests the ability of our testing framework for capturing serial dependence of the output variable. The choice of different values for the quantities $a^{(1,j)}$ and $a^{(j,1)}$, for different values of j, allow us to gauge the ability of the test to detect Granger causality depending on the magnitude of the relationship between x_{jt} and x_{1t}, and the time series persistence of each x_{jt}, respectively. We choose this scenario to assess the performance of the model in large systems given by many variables and many lags. A suitable testing procedure should have values of $\bar{D}_j$ close to one if the variable x_{jt} Granger causes the variable x_{1t} - one minus the probability of type II error - and a value close to zero if the variable x_{jt} does not Granger cause x_{1t} - probability of type I error.

Linear VAR(1) model:

The following bar chart reports the fraction of rejections out of B times of the null hypothesis (2.7) for each of the input variables in the system. The data generating process in this case is model (2.36) with a set of exogenous regressors that are persistent (case *iv*) above).

[18] Unreported simulations show that the results are very similar for different numbers of hidden layers. In this chapter, we do not explore further the possibility of considering more than two hidden layers in the construction of the feedforward neural network.

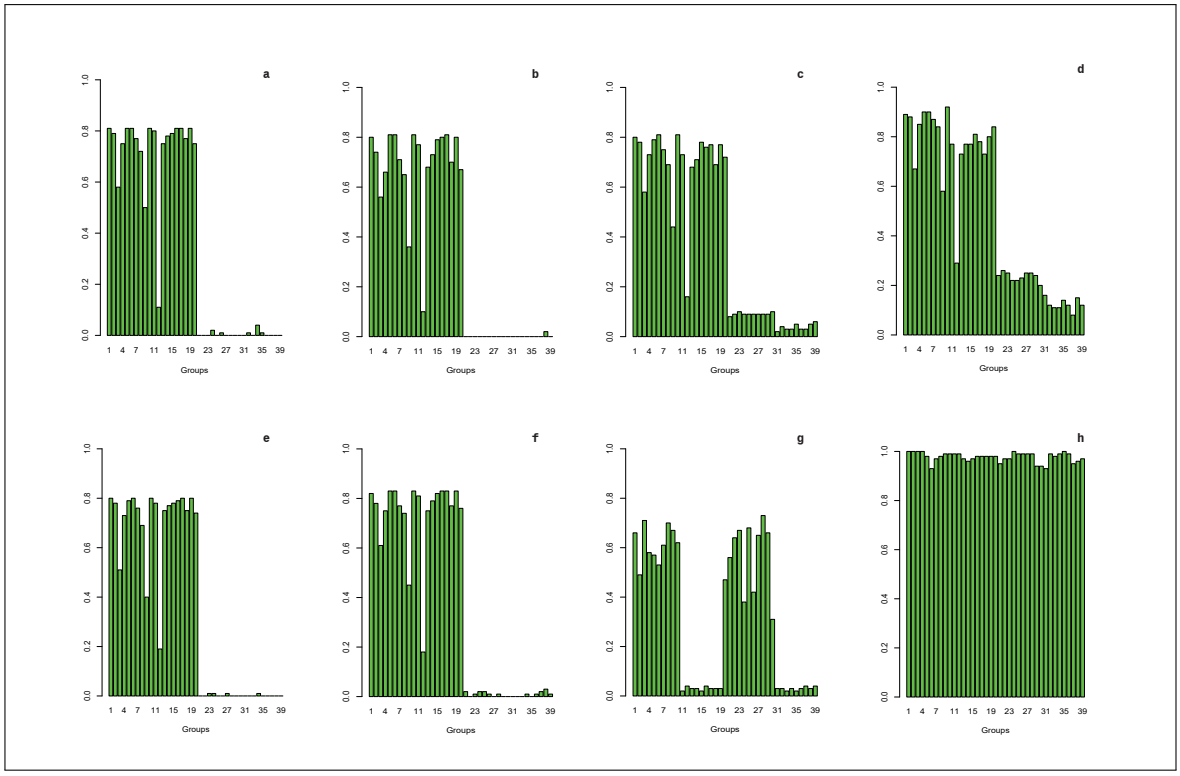


Figure 2.5: Simulated rejection probabilities for the linear VAR(1) when the sample size is T = 500 and the input variables exhibit autoregressive persistence. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure. Both procedures start with the same initial structure for the feedforward neural network. Subfigures a and e consider the structure $\underline{z}_1 = 1$; $\underline{z}_2 = 1$; subfigures b and f consider the structure $\underline{z}_1 = 1$ and $\underline{z}_2 = 2$; subfigures c and g correspond to $\underline{z}_1 = 2$ and $\underline{z}_2 = 1$; and subfigures d and h correspond to $\underline{z}_1 = 5$ and $\underline{z}_2 = 10$.

The x-axis in Figure 2.5 indexes the input variables between 1 and 39, with the first 20 variables exhibiting Granger causality on x_{1t} and the last 19 variables being those variables that do not affect x_{1t}. The bar chart for each value in the x-axis denotes a rejection probability that denotes the power of the detection procedure (1-probability type II error) for the first 20 variables and estimates of the probability of type I error for the remaining 19 variables. Interestingly, both optimized and non-optimized network structures initialized with the pairs $(1, 1)$ and $(1, 2)$ for the number of nodes in the hidden layers provide a perfect cut-off point between input variables. This means that the probability of the type I and type II errors is zero or close to zero. This result shows an outstanding performance of the test in classifying Granger causality since not only the ability to reject the null hypothesis when the alternative holds is high but also the test hardly exhibits type I error. Variation in the probability of type I and type II errors across variables is due to the effect of the magnitude of the parameters $a^{(1l)}$ defining the dependence structure. The lowest proportion is associated with $j = 12$ that corresponds to an input variable with coefficient closer to 0. This result is shared by both

types of feedforward network structures and does not depend on whether the first hidden layer is optimized or not. In this case, the robustness of the result is because the initial value of the number of hidden nodes is the optimal one.

This result changes abruptly when the initializing parameters are not optimal. This case is shown in subfigures (c) and (g) for $\underline{z}_1 = 2$ and $\underline{z}_2 = 1$, and (d) and (h) for $\underline{z}_1 = 5$ and $\underline{z}_2 = 10$. In this scenario, there are important differences between the optimized two-stage testing procedure (in the top row) and the non-optimized testing procedure (bottom row). The effect of the first-stage in the optimized testing procedure is clear. More specifically, the implementation of the algorithm in Montgomery and Eledath (1995) allows us to reduce the number of relevant nodes in the first hidden layer. In this case the testing procedure has considerable probability of type I and type II errors, although different from zero for some variables, it is still close to zero in many cases. The success of the testing procedure in identifying no Granger causality depends on the persistence of the autoregressive parameters for the exogenous variables and also the success of the algorithm to reduce the number of superfluous nodes. Thus the optimized method works better when the initial number of nodes is closer to the optimal coefficient than when it is far (see subfigure (d)). We include this case as an illustration of cases where the number of initial nodes in the first hidden layer is far from the target. We also note the complete failure of the non-optimized testing procedure in the latter case (subfigure (h)) when the number of initial nodes is not optimized.

Figure 2.6 repeats the same exercise for the case when the set of input variables not Granger causing x_{1t} are white noises (case *ii*) above). The sample size is $T = 500$ as before. The results are very similar across different choices of the number of nodes in the hidden layers and confirm the strong performance of our two-stage testing procedure and the poor performance of the non-optimized method that chooses an arbitrary number of nodes in the first layer when the initial number is not optimal, that is, it is different from one. The comparison of Figures 2.5 and 2.6 shows no effect of the persistence of the exogenous variables on the Granger causality tests.

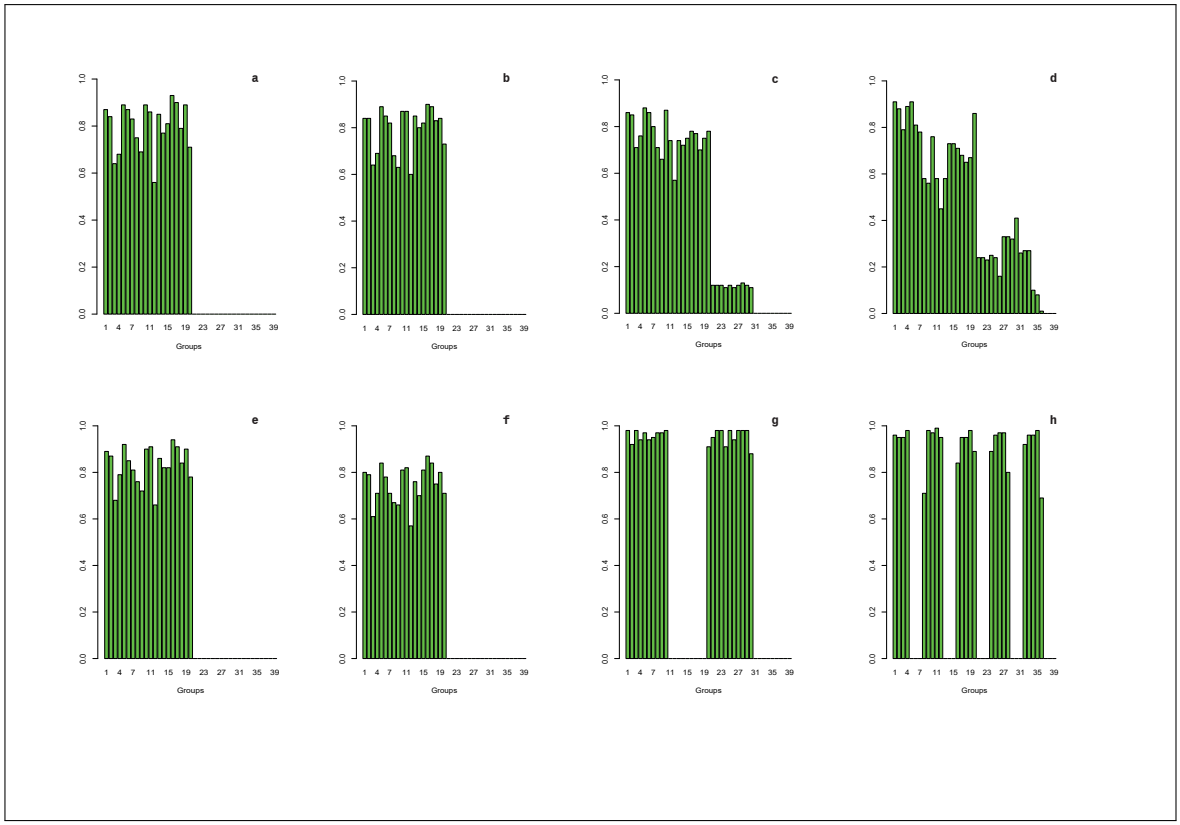


Figure 2.6: Simulated rejection probabilities for the linear VAR(1) when the sample size is T = 500 and the last 19 input variables are a white noise. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Figure 2.7 shows the same qualitative results for the case of persistence in the regressors when the sample size increases to $T = 1000$[19]. In this case, the probability of type II error of the test is closer to zero for most input variables and the probability of type I error of the test is smaller than for $T = 500$. It is interesting to note that the probability of type I error of the test depends more on the distance of the initial number of nodes in the first hidden layer from the optimal number than in the number of observations used for training the network. Nevertheless, we observe a considerable improvement in case (d) for $T = 1000$ compared to case (d) for $T = 500$.

[19] Unreported results available from the authors upon request illustrate the case for white noise input variables and $T = 1000$. The results are qualitatively very similar to the rejection probabilities in Figure 2.7.

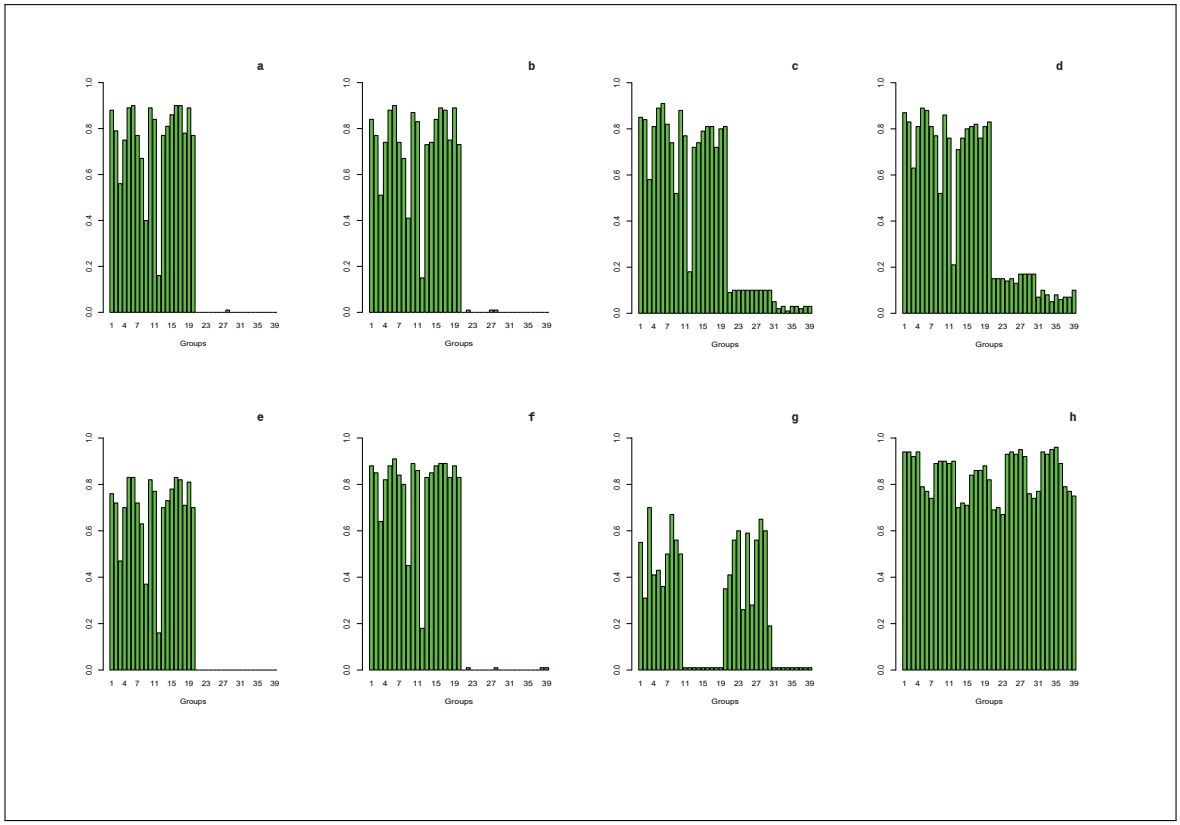


Figure 2.7: Simulated rejection probabilities for the Linear VAR(1) when the sample size is T = 1000 and the input variables exhibit autoregressive persistence. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Non-Linear VAR(1) model:

This block analyzes the type I error probability and the corresponding power analysis of the Granger causality tests when the data generating process is (2.37). Figure 2.8 studies the nonlinear VAR(1) model when τ_t is a random error with no persistence following a $\mathcal{N}(0.5, 1)$ distribution, and the variables without Granger Causality are assumed to be white noise (case ii) above). The figure shows similar patterns to the previous exercises. The testing procedure is able to identify Granger causality when there exists. Both optimized and non-optimized feedforward neural networks work very well when the initial value of the number of nodes in the first hidden layer is close to the optimal one. The optimized method also performs very well in case (c), reporting values of the probability of type I error close to zero. In case (d), the performance of the test is not as reliable as in the first cases; however, it still represents a massive improvement compared to the non-optimized testing procedure. We should note that the initial number of nodes in the latter example is far from the target. This phenomenon seems to have an important effect on the size of the testing procedures.

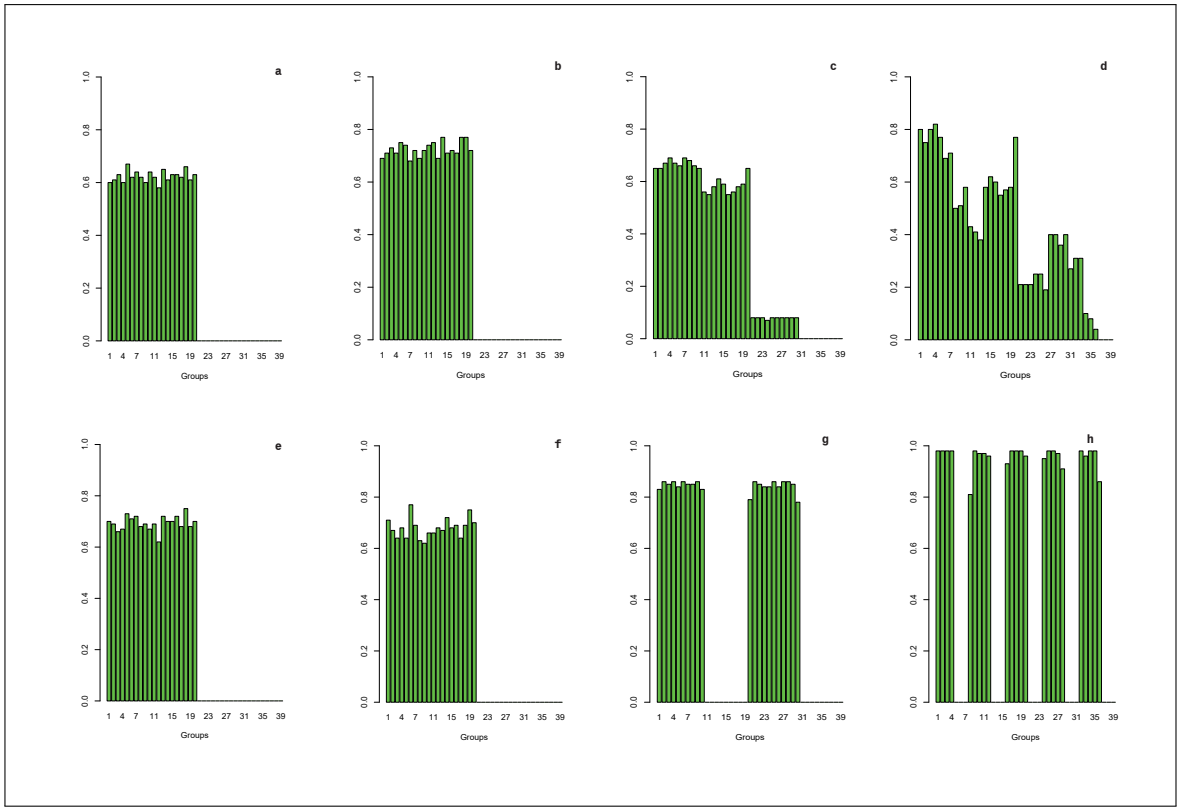


Figure 2.8: Simulated rejection probabilities for the nonlinear VAR(1) when the sample size is $T = 500$ and the last 19 input variables are a white noise. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Figure 2.9 reports the probability of type I error and the corresponding power analysis for the nonlinear VAR(1) exercise for $T = 1000$. In this case there is an improvement (decrease) in both probabilities of type I and type II errors, however, as mentioned above, the improvement depends more on the correct choice of nodes in the first hidden layer than in the amount of available information offered by an increasing sample size.

In both exercises, we should stress that the data generating process is very nonlinear and the dependence between variables leading to Granger causality is masked to a large extent by the interaction with the random variable τ_t. Standard testing procedures based on Wald type tests or likelihood ratio tests would fail completely in this setting unless the data generating process is known to the econometrician. In contrast, our approach is very general and does not rely on the specific parametric form of the VAR model.

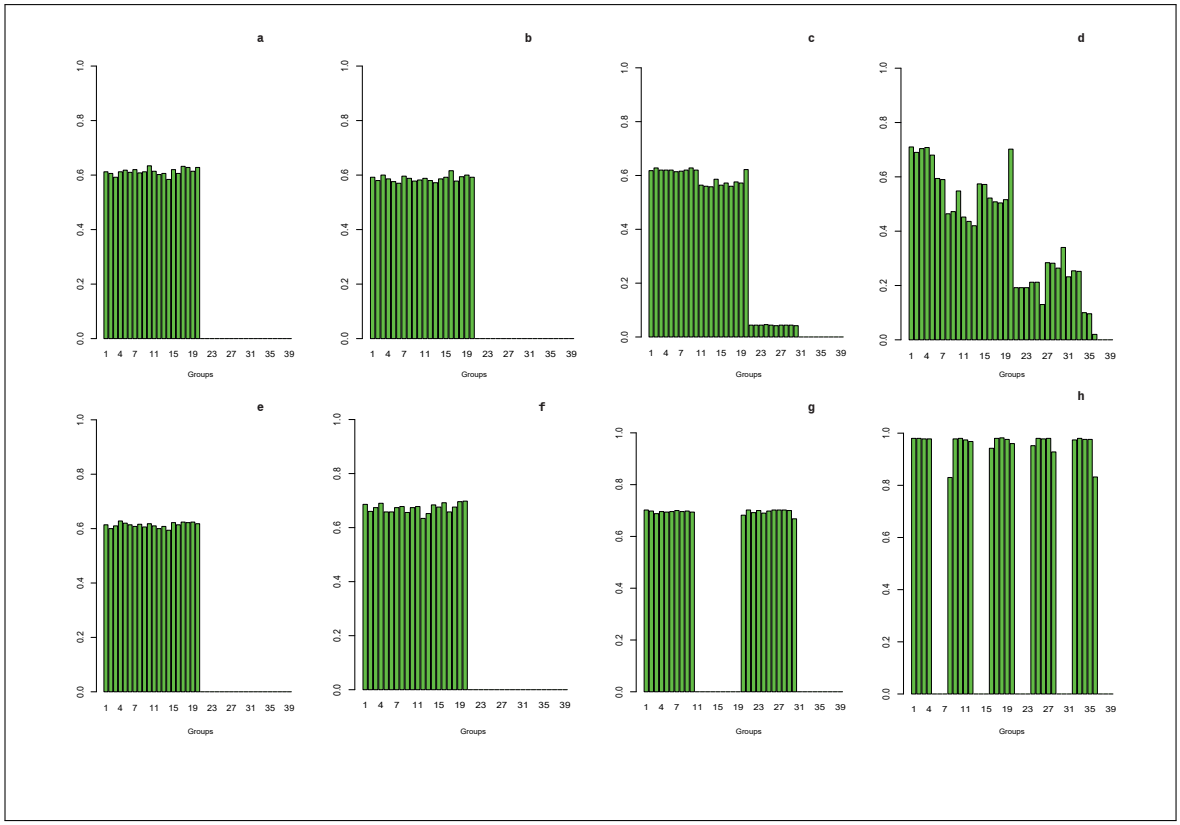


Figure 2.9: Simulated rejection probabilities for the Non-Linear VAR(1) when the sample size is T = 1000 and the last 19 input variables are a white noise. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

The next exercise studies the performance of the probability of type I and type II errors when the variables that do not Granger cause x_{1t} exhibit persistence (case iv) above). Figure 2.10 shows similar patterns to previous exercises. The testing procedure is able to identify Granger causality when there exists; however, the probability of type I error of the test is considerably larger than in the linear case and the nonlinear case with white noise variables. There are also important differences between the optimized and non-optimized methods when the initial number of hidden nodes is different from one, see subfigures (c) and (g), and (d) and (h). In particular, the non-optimized method reports values of probability of type I error close to one across input variables invalidating any conclusion obtained from the test.

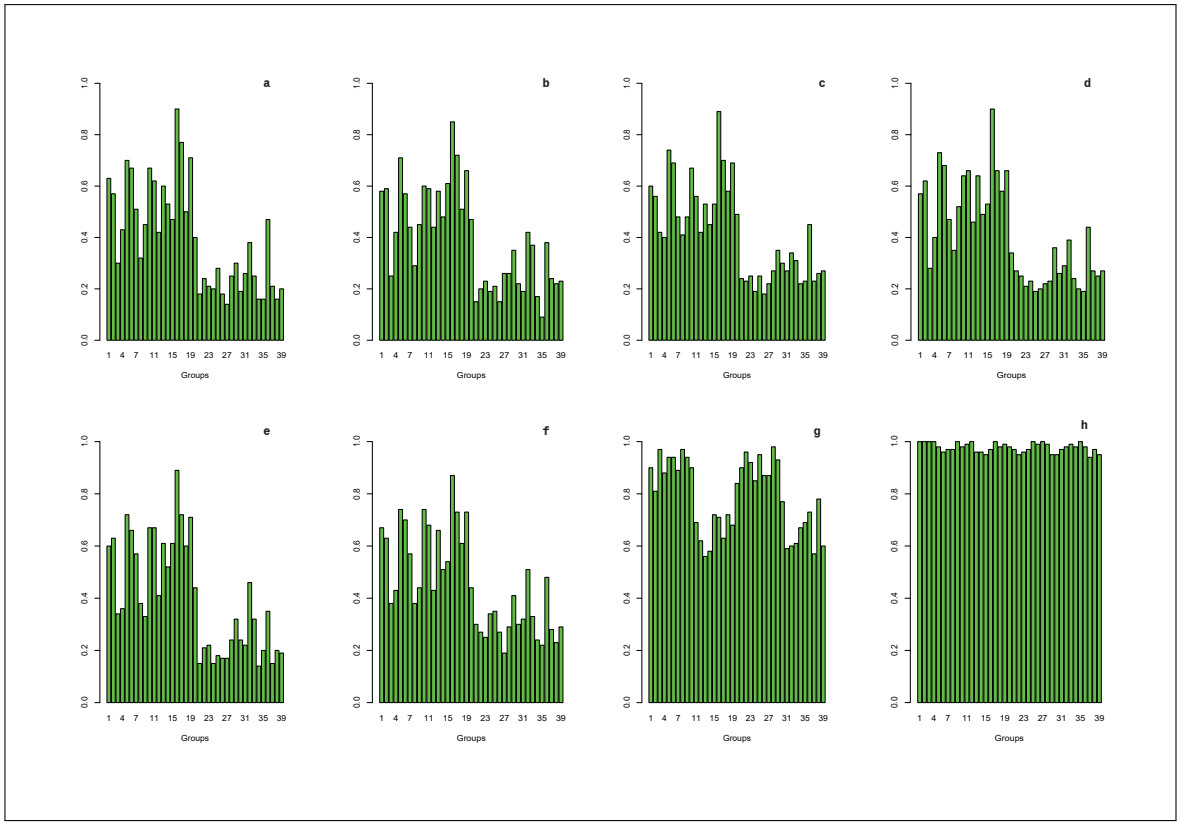


Figure 2.10: Simulated rejection probabilities for the Non-Linear VAR(1) when the sample size is T = 500 and the input variables exhibit autoregressive persistence. Subfigures a and e consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures b and f ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures c and g ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures d and h ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Figure 2.11 reports the nonlinear case (2.37) for $T = 1000$. In contrast to previous scenarios, we observe a significant improvement of the test when the sample size increases. The probability of type I error is considerably lower and guarantees that the empirical power obtained under the alternative hypothesis is not spurious. Surprisingly, one minus the probability of type II error is not as high as before, and yields low values when the parameters associated with the input variables exhibiting Granger causality are close to zero.

The empirical findings observed in the nonlinear case show an important effect of the persistence of the exogenous variables on the probabilities of type I and type II errors. The optimized two-stage testing procedure performs better under the absence of persistence in the variables that do not Granger cause x_{1t}.

We should also note the sensitivity of the type I and type II error probabilities to variations in the expected value, μ, of the random variable τ_t. For $\mu \geq 0.8$ the probabilities of type I and type II errors in the nonlinear case with persistence (worst case scenario) decrease significantly, performing as well as for the linear case. This behavior can be justified by the level of noise in the data generating process; an increase in the difference between the stochastic behaviors of τ_t and of the regressors x_{jt} will reduce the nuisance in the system, and it will ensure an easier identification of the separate effects of τ_{t-1} and x_{t-1} on x_{1t}.

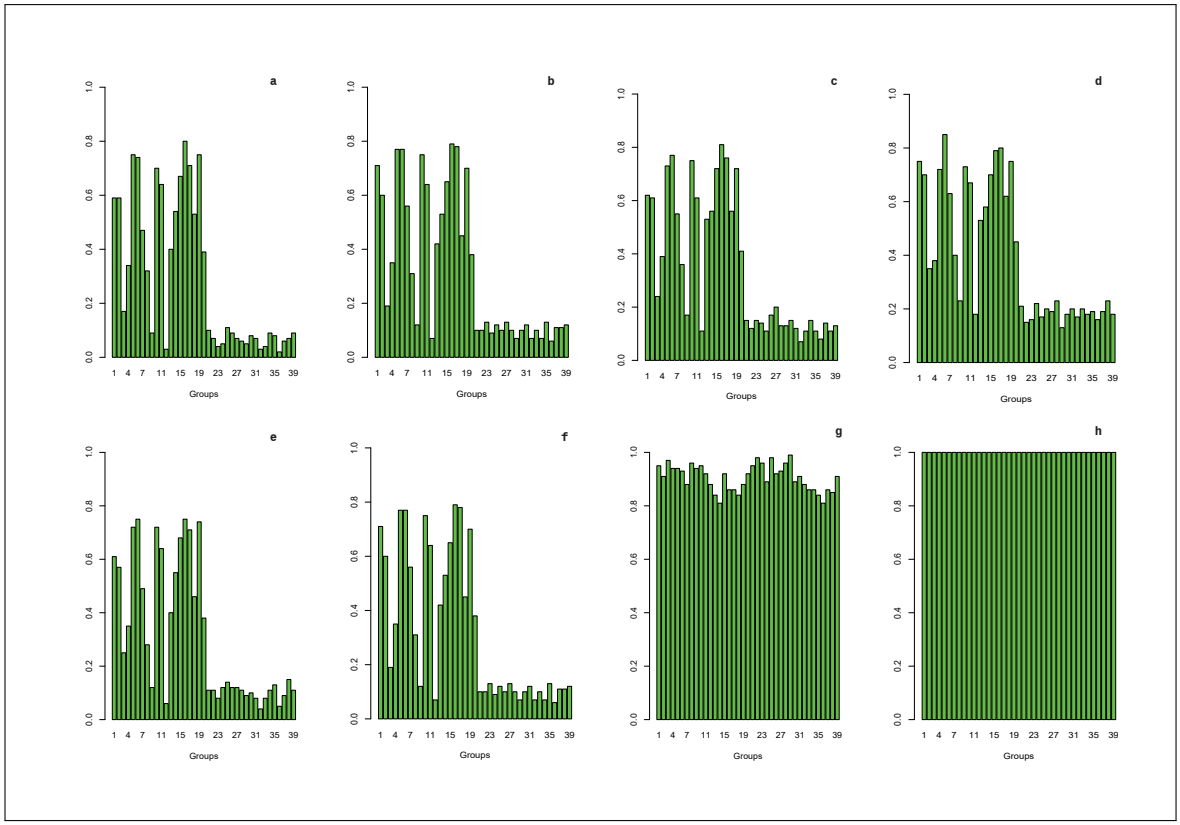


Figure 2.11: Simulated rejection probabilities for the Non-Linear VAR(1) when the sample size is T = 1000 and the input variables exhibit autoregressive persistence. Subfigures *a* and *e* consider the structure ($\underline{z}_1 = 1$; $\underline{z}_2 = 1$); subfigures *b* and *f* ($\underline{z}_1 = 1$; $\underline{z}_2 = 2$); subfigures *c* and *g* ($\underline{z}_1 = 2$; $\underline{z}_2 = 1$); and subfigures *d* and *h* ($\underline{z}_1 = 5$; $\underline{z}_2 = 10$). Top row figures correspond to the optimized two-stage procedure and bottom figures to the corresponding non-optimized procedure.

Linear VAR(10) model:

Once the impact of the correct structure of the neural network on Granger causality discovery is analyzed, we relax the assumption of $\alpha = 0$ and extend the previous simulation scheme by considering an endogenous variable x_{1t} driven by ten lags.

The following exercise extends the previous simulation scheme by considering an endogenous variable x_{1t} driven by ten lags. In particular, we generate process (2.38) and impose the same parameter structure $\mathbf{a}^{(1,1)} = \cdots = \mathbf{a}^{(1,10)}$ across lags. This simulation exercise is richer than for the VAR(1) case as it allows us exploring the performance of the Granger causality tests when there is more than one lag Granger causing the output variable x_{1t}. In this setting, we consider two competing models: our two-stage testing procedure proposed in expression (2.26) that is based on a double group lasso penalty function and the hierarchical group lasso penalty function discussed above. In contrast to previous cases, the success of feedforward neural network procedures is measured by their performance in terms of the probabilities of type I and type II errors and also on their ability to detect the correct number of lags, given by $k = 10$ in this simulation exercise.

The results of the simulations for cases *ii*) and *iv*) above are reported in Figure 2.12 for $T = 500$ and Figure 2.13 for $T = 1000$, respectively. In particular, the last 19 variables in

subfigures *a-b* are exogenous variables following independent white noise processes such that the reported proportions correspond to the probability of type I error when the input variables follow independent white noise processes. In contrast, the last 19 variables in subfigures *c-d* are exogenous variables that exhibit stationary autoregressive persistence. In these cases, the reported proportions correspond to the probability of type I error under different degrees of persistence of the exogenous variables. In both sets of experiments, the first 19 variables for assessing the power of the detection procedure are stationary variables with different degrees of persistence. More specifically, subfigure *a* and *a.1* consider the case of the double group lasso penalty function and white noise exogenous variables. Subfigures *b* and *b.1* consider the hierarchical group lasso penalty function with white noise exogenous variables. Subfigures *c* and *c.1* present the double group lasso when the exogenous variables are persistent. Subfigures *d* and *d.1* consider the hierarchical group lasso under persistence of the exogenous variables.

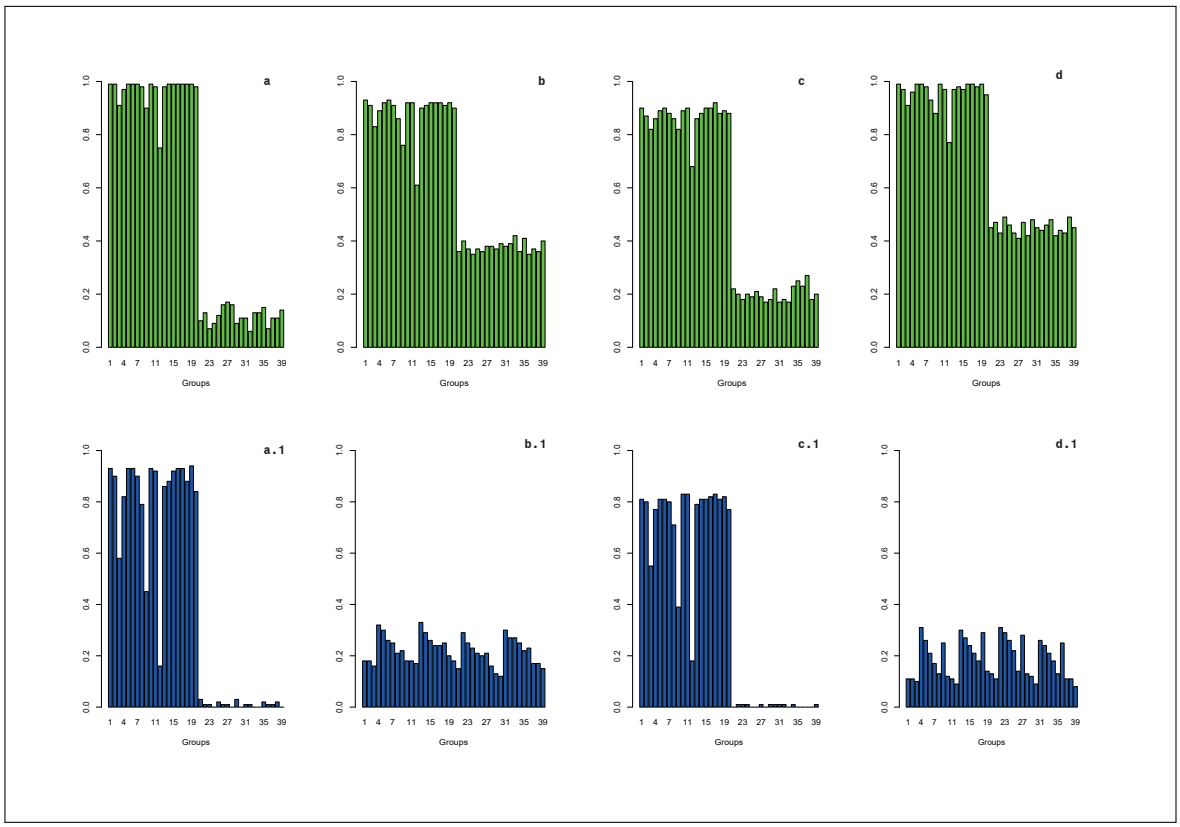


Figure 2.12: Simulated rejection probabilities for the linear VAR(10) for a sample size of T = 500 when the last 19 input variables are a white noise. For each process and each penalty, the proportions of the group identification (top row), and correct lag detection (bottom row) are reported. Subfigure *a* and *a.1* consider the case of double group lasso and white noise exogenous variables; subfigures *b* and *b.1* hierarchical group lasso and white noise exogenous variables; Subfigures *c* and *c.1* double group lasso and persistent exogenous variables; Subfigures *d* and *d.1* hierarchical group lasso and persistent exogenous variables.

The top row in Figure 2.12 reports the rejection probabilities for the null hypothesis (2.7). The bottom row reports the fraction of times the correct number of lags ($k = 10$) is estimated for each input variable. For each process and penalty function, the proportions of the group identification (top row), and correct lag detection (bottom row) are reported.

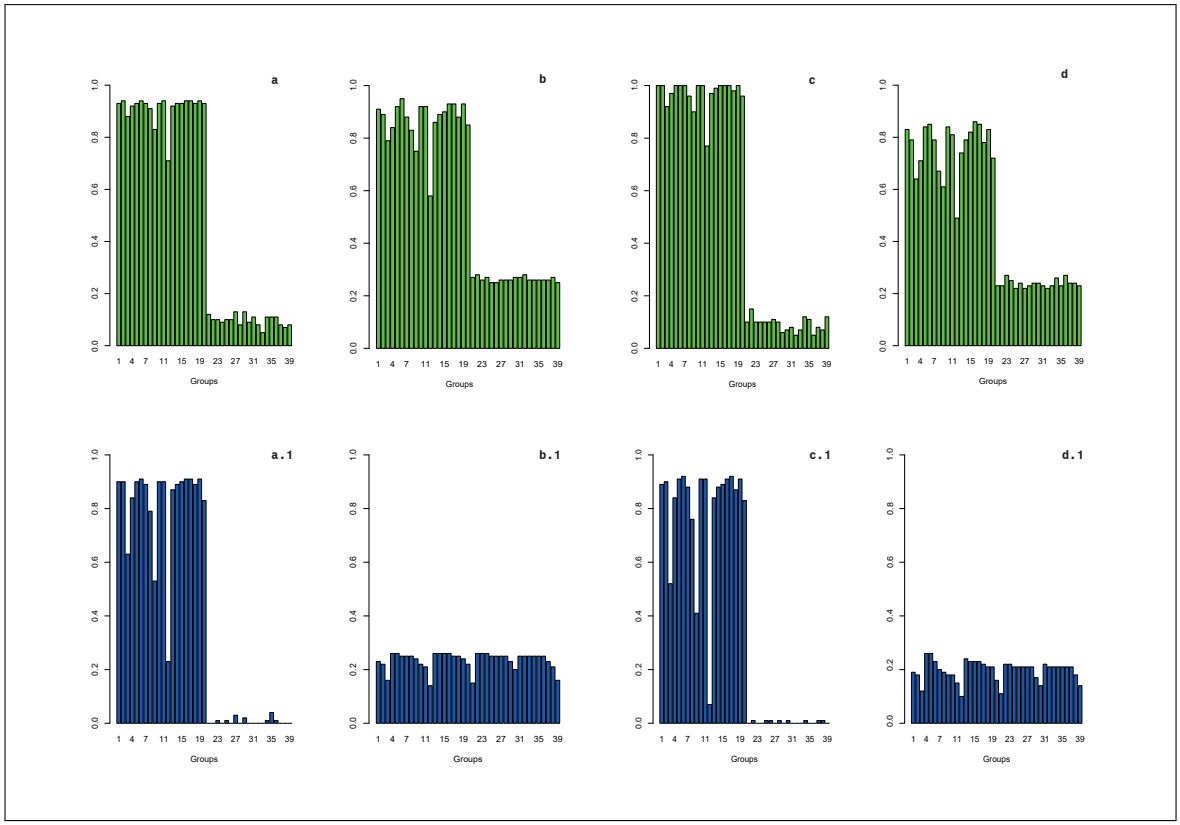


Figure 2.13: Simulated rejection probabilities for the Linear VAR(10) when a sample size of T = 1000 and the last 19 input variables are a white noise. For each process and each penalty, the proportions of the group identification (top row), and correct lag detection (bottom row) are reported. Subfigure *a* and *a.1* consider the case of double group lasso and white noise exogenous variables; subfigures *b* and *b.1* hierarchical group lasso and white noise exogenous variables; Subfigures *c* and *c.1* double group lasso and persistent exogenous variables; Subfigures *d* and *d.1* hierarchical group lasso and persistent exogenous variables.

The results from both experiments show overwhelming evidence on the outperformance of the double group lasso regularization function for both scenarios (white noise and persistent regressors) in terms of type I and type II error probabilities. It is worth noting the good performance of the double group lasso procedure with regards to the probability of type I error of the test in subfigure *c*. The type I error probability takes on low values even under the presence of persistence in the non Granger causal regressors.

The results in the bottom row also suggest a very good performance of the double group lasso function compared to the hierarchical lasso. The accuracy of the method decreases for those input variables that are characterized by a small persistence parameter. The hierarchical lasso penalty function underestimates the correct number of lags across the set of input variables, in contrast, the double group lasso penalty function provides robust results for the correct number of lags in most cases.

2.6.3 Model Selection Consistency

The simulation study also verifies that the the *oracle* property of consistent variable selection is satisfied (Fan and Li, 2001). Let $\hat{M}_T$ be the set of parameter estimates that satisfy

$\widehat{\mathbf{W}}(\delta) = 0$, and M_0 the set of corresponding model parameters that are actually equal to zero. More formally, Leeb and Pötscher (2005) assert that a procedure δ is consistent if i) $\lim_{T\to\infty} P(\hat{M}_T = M_0) = 1$, and ii) $\lim_{T\to\infty} P(\hat{M}_T \neq M_0) = 0$, with $P(\cdot)$ denoting the probability function. Section 4 studies these properties theoretically from the first order conditions of the objective function (2.26). In particular, we show that the tuning parameter λ needs to satisfy the condition $\lambda = o(1/T)$ in order to guarantee model selection consistency if the number of lags k is fixed. Alternatively, if the number of lags is allowed to increase with the sample size, then, the condition $\lambda = o\left(\frac{1}{\sqrt{k_T T}}\right)$ is sufficient to guarantee model selection consistency.

Table 2.2: The Table reports the correct and incorrect number of null coefficients. We apply this procedure to the linear and nonlinear VAR(1) processes and case *iv* are considered. On the right of the Table, the figures corresponding to the oracle estimator are reported.

Linear

	T = 500	**T = 1000**	**T = 2000**	**T = 3000**	**Oracle**
[1,1]					
Correct	18.94	18.98	19	19	19
Non Correct	5.88	4.46	3.02	3.00	0
[2,1]					
Correct	17.42	17.55	18.00	18.68	19
Non Correct	5.86	4.63	4.26	3.56	0

NonLinear

	T = 500	**T = 1000**	**T = 2000**	**T = 3000**	**Oracle**
[1,1]					
Correct	14.04	16.88	17.74	18.31	19
Non Correct	8.35	9.96	9.91	9.02	0
[2,1]					
Correct	13.8	16.53	17.62	18.17	19
Non Correct	10	9.62	9.80	9.60	0

To add further empirical evidence to the above findings on the consistency of the objective function (2.26) for the detection of Granger causal relationships and the number of relevant lags, we carry out a second simulation experiment. Following Fan and Li (2001) and Leeb and Pötscher (2005), a Monte Carlo simulation is adopted to assess empirically model selection consistency. Our simulation experiment considers a system of p variables in the data generating processes; hence, an *oracle* procedure will correctly identify, as the sample size increases, the number of variables that satisfy the null hypothesis and the number of variables that are significant, that is, Granger cause the output variable. Hence, if our Granger causality test satisfies the *oracle* property, it will correctly identify, as the sample size increases, 19 null coefficients for those variables that are not significant and zero null coefficients for those variables that are significant. For the purpose of the simulation, we consider 100 simulated linear and nonlinear VAR(1) processes with autoregressive persistence in all of the input variables -

case *iv*) above - and different sample sizes $T = 500, 1000, 2000, 3000$. The *oracle* property of consistent variable selection is tested for two settings: when the number of nodes in the first hidden layer corresponds to the optimal value - $\underline{z}_1 = 1$ - and when $\underline{z}_2 = 2$. In the latter case, the two-stage method for Granger causality detection improves the properties of the test with respect to the naive approach, that is shown to perform very poorly in the above exercises. For each sample size considered, the average number of correctly and incorrectly identified zero coefficients is reported, see also Fan and Li (2006) for a similar procedure, in Table 2.2.

The results show that for the linear VAR(1) and $\underline{z}_1 = 1$ the correct number of zeros reaches its maximum - 19 - for $T = 2000$, and the incorrect number of zeros decreases as the sample size increases to a minimum of 3 for $T = 3000$. Also, when the initial number of nodes in the first hidden layer is $\underline{z}_1 = 2$, the correct number of zeros increases as the sample size increases to a maximum that reaches 19 for $T = 3000$, and the incorrect number of zeros decreases to a minimum near 4. These results show that the *oracle* property of consistency holds for our group lasso regularization function when the correct number of nodes is specified and, also, for our two-stage approach that optimizes the architecture of the neural network.

To assess this property in more challenging settings, we repeat the experiment for the nonlinear VAR(1) process. In this case, the number of correctly classified variables increases to a maximum of approximately 18 when $T = 3000$, and the incorrect number of zero coefficients decreases to a minimum of around 9 when $T = 3000$. As the incorrect number of zero coefficients decreases to zero at a slower rate, one could conclude that the penalty is over-conservative. Overall, these results are also supportive of the good performance of the two-stage Granger causality tests proposed in this study. Nevertheless, these simulations provide some evidence of the challenges involved in determining the variables with predictive ability under very general nonlinear settings characterized by unknown forms of Granger causality.

2.7 Empirical Analysis: Tobalaba Network

The Energy Web Foundation provides the Energy sector a blockchain-based test network with Proof-of-Authority[20] consensus mechanism: the Tobalaba test network (Energy Web Foundation, 2018). The World Bank Group (2018) highlights the benefit of a distributed ledger technology over the traditional centralized ledgers: it allows decentralization and disintermediation, it guarantees information symmetry due to the verifiable audit of transactions of both physical and digital assets, and it ensures cost reduction and associated increase in the speed for the stipulation of contracts via the smart contracts. J.P. Morgan (2018) argues that the automation and the disintermediation arising from the application of blockchain technologies will automate functions necessary to participate in the market, expanding the access also to smaller participants. The Energy Web Foundation (2018) states that smart contracts, by automating bilateral transactions, will allow for a greater diversity of market structure. The resulting information symmetry will allow tracing in the carbon and renewable energy market,

[20] Transactions are validated through validators, reducing the energy impact.

credit ownership with lower costs and higher accuracy (Energy Web foundation, 2018).

Using these arguments, one should expect an increase in the stipulation of contracts, and thus in the interactions among the members of the Tobalaba network. The increase of the connections (unilateral or bilateral) is justified by a higher transparency of credit and asset ownership, by the automation of the execution of smart contracts (not feasible for centralized ledgers), and by the reduction of transaction costs due to disintermediation. The aim of this section is to explore this empirically. To do this, we use recent work on social and financial networks, see Billio et al. (2012) and Hecq et al. (2019), that establish connections in a network through the presence of Granger causality between the variables characterizing the nodes. These authors explore Granger causality between pairs of variables. In this application, we broaden the analysis of Granger causality defining a network, and consider equation (2.4), reproduced here again:

$$x_{it} = g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z}) + \epsilon_{it}, \text{ for } i = 1, \ldots, p,$$

where $g_i(\mathbf{X}_{t-1}; \mathbf{W}, \mathbf{z})$ captures the multivariate dynamic structure between the percentage cumulative log-returns over a 30-minute-time window for the firms below. We model the multivariate dependence component-wise using a neural network for each company $i = 1, \ldots, p$.

2.7.1 Data

Table 2.3: Names, market, and a short description of the core activities of the companies considered in this study.

Company Name	Market	Core	Tick
Acciona	Spain	Renewable energy	ANA
Aes Corp	USA	Electricity sell, mines coal, alternative source of energy	AES
Centrica	London	Home and business energy solution	CAN
Duke Energy	USA	Manage portfolio of natural gas supply and delivery	DUK
Engie Brasil Energia	Brazil	Exploration, production and trading of electricity and natural gas	EGIE3
Equinor Asa	Oslo	Develops oil, gas, wind and solar energy projects	EQNR
Exelon Corp	USA	Distributes energy to Illinois and Pennsylvania	EXC
Fluvius	Luxembourg	Renewable energy distribution Network	FLUVIU
General Electric	USA	Diversified technology	GE
Iberdrola	Spain	Generates, distributes, trade electricity	IBE
Innogy	Germany	Manages plans to generate power from renewable energy	IGY
Itron	USA	Collecting, communicating analyzing electric data	ITRI
PG&E Corp	USA	Holding company that provides natural gas and electricity	PCG
Royal Dutch Shell	London	Explores, produces, refines petroleum	RDSA
Siemens	Germany	Engineering and manufacturing company	SIE
Total Sa	Euronext Paris	Explores for producers, refines, transports, and market oil and natural gas	FP
Wipro	India	E-commerce, data warehousing, system administration	WPRO

Intraday prices in 30-minute intervals for the companies reported in Table 2.3 over the period 09/05/2016 to 10/05/2019 are collected from Bloomberg. Of the 70 companies belonging to the Tobalaba Network, only those reported in Table 2.3 are considered[21]. We exclude

[21] The rest of the companies are excluded for two main reasons: they are either not public, or due to the high number of missing at random observations.

companies listed in different time zones and nonlisted companies. Our dataset is divided in two periods - before the introduction of Tobalaba (09/05/2016 - 29/03/2018) - and after the creation of Tobalaba (26/10/2018 - 10/05/2019). A time interval between the two subsets is left in order to allow the creation of the connections between the members of the Energy Web blockchain.

The missing values in the dataset are completed using the MissForest algorithm (Stekhoven, 2013). The maximum number of trees to be grown in each forest is set equal to 500, the maximum number of nodes for each tree is equal to 100, and the maximum number of iterations is 50. The MissForest algorithm does not make any assumption about the distribution of the variables as it involves estimating the missing values by fitting a random forest trained on the observed values. The Out-Of-Bag (OOB) estimates of the imputation error in terms of Normalized Root Mean Squared Error (NRMSE) for the two subsamples is 0.01438 and 0.012984, respectively. The returns are then computed from the intra-day prices.

Table 2.4 reports the exploratory data analysis conducted for both subsamples for each individual series considered. There is a general increase in the mean and standard deviation of the returns for each company. In all cases, the Dickey-Fuller test rejects at 0.05 significance level the null hypothesis of unit root, and we fail to reject at 0.05 significance level the null hypothesis of stationarity of the KPSS test, showing that for both subsamples all series considered are stationary.

Table 2.4: Exploratory data analysis for the series considered. Due to the number of observations, the Kolmogorov-Sminrov test for normality is adopted in the first sub-sample. Moreover, the test statistics and associated p-values of the Dickey-Fuller and the KPSS tests for stationarity are reported.

(09/05/2016 - 29/03/2018)																	
	ANA	AES	CAN	DUK	EGIE3	EQNR	EXC	FLUVIU	GE	IBE	IGY	ITRI	PGC	RDSNA	SIE	FP	WPRO
Mean	-0.0015	-0.0005	-0.0045	-0.0005	0.0000	0.0032	0.0013	0.0003	-0.0092	-0.0003	-0.0007	0.0054	0.0063	0.0017	0.0013	0.0009	0.0007
Std. Deviation	0.3127	0.7086	0.4612	0.5685	0.7603	0.4184	0.6126	0.0667	1.0939	0.2951	0.5256	1.6895	1.5639	0.3069	0.3286	0.3069	1.1897
Min	-8.3657	-8.5695	-17.1965	-6.6416	-9.6321	-3.6688	-5.2387	-0.6217	-13.8826	-12.8402	-7.6234	-34.4226	-32.8006	-8.5474	-8.7049	-9.5013	-11.4161
Max	3.7182	10.7099	14.3815	6.5628	10.5759	4.2803	5.4795	0.6585	21.9190	3.3580	7.5691	34.7628	33.9859	5.2412	6.7774	4.9290	10.5179
Skweness	-1.7878	-0.1906	-5.8680	-0.2275	0.0911	0.0493	-0.3339	0.3446	0.6017	-9.3001	-0.0698	0.8649	0.7876	-1.5988	-1.8681	-2.7896	-0.3433
Kurtosis	72.9088	35.5819	423.4997	28.9009	36.5251	15.1058	17.3413	36.5411	66.8865	419.7467	53.2526	101.3114	122.7723	93.5393	129.5589	119.0220	19.4230
Kolm. t. stat.	0.2872	0.2576	0.2696	0.2835	0.2342	0.2718	0.2648	0.4399	0.2652	0.3070	0.2935	0.2257	0.2221	0.3017	0.3033	0.2940	0.2167
Kolm. p-value	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
DF t. stat.	-47.2078	-50.6489	-46.6261	-50.4284	-50.6923	-57.6236	-52.4630	-54.1766	-48.6654	-50.2723	-51.1224	-47.7949	-48.3805	-48.1052	-49.8299	-49.9683	-50.6796
DF p-value	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
KPSS stat	0.1414	0.0155	0.1606	0.0630	0.0626	0.0605	0.0343	0.0539	0.3718	0.1060	0.0176	0.0387	0.0589	0.0456	0.3180	0.0333	0.0472
KPSS p-value	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.0893	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
(26/10/2018 - 10/05/2019)																	
	ANA	AES	CNA	DUK	EGIE3	EQNR	EXC	FLUVIU	GE	IBE	IGY	ITRI	PCG	RDSA	SIE	FP	WPRO
Mean	0.0079	0.0016	-0.0145	-0.0006	0.0060	-0.0022	0.0022	0.0003	0.0104	0.0053	-0.0007	0.0055	-0.0111	0.0027	0.0028	-0.0013	-0.0031
Std. Deviation	2.1697	1.5338	2.4323	0.8389	2.8989	1.3800	0.8406	0.0656	3.8024	1.3659	0.5781	2.2506	11.7503	1.2577	1.3416	1.1393	2.0681
Min	-27.0106	-17.2372	-24.6358	-6.1559	-24.1935	-12.3153	-8.8194	-0.4471	-41.8479	-11.1166	-5.4102	-21.9921	-116.7995	-9.2351	-11.5270	-9.5761	-23.3458
Max	26.1996	18.7566	24.8999	6.9390	22.7915	12.3872	8.6189	0.3615	40.9288	11.0580	5.2053	22.4000	85.3738	9.6994	11.5836	9.9737	25.3803
Skweness	-0.2296	0.7305	-0.0618	-0.3039	-0.1741	0.2325	0.5157	-0.2896	0.0196	0.0515	-0.2343	-0.2159	-1.1079	0.1642	0.0447	0.2635	-1.1434
Kurtosis	47.4203	52.9197	34.5855	21.1393	30.9132	37.4122	31.7782	15.4785	33.8791	29.2221	27.9577	26.2839	24.8550	16.2953	25.5945	22.2098	49.7950
Shapiro t. stat.	0.3710	0.4670	0.3945	0.5968	0.4429	0.4528	0.5746	0.6277	0.5099	0.4557	0.5111	0.5600	0.4822	0.5319	0.4741	0.5222	0.4525
Shapiro p-value	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
DF t. stat.	-24.4804	-24.8217	-22.9151	-24.9950	-23.7368	-22.8723	-24.7452	-18.4441	-24.4323	-23.9530	-23.4678	-23.4181	-24.3258	-23.0250	-22.8387	-22.7410	-23.9251
DF p-value	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
KPSS t. stat.	0.0192	0.0497	0.0646	0.0171	0.0145	0.0176	0.0192	0.0640	0.0141	0.0208	0.0191	0.0356	0.0162	0.0101	0.0316	0.0275	0.0343
KPSS p-value	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000

2.7.2 Empirical Results

Figure 2.14 shows the network topologies induced by fitting model (2.4) to detect Granger causality between the $p = 17$ firms considered in our sample before and after the introduction

of Tobalaba. The method to detect Granger causality is based on using an optimized neural network and the objective function (2.26). Thus, the network is constructed by fitting 17 feedforward neural networks with a double group lasso penalty function to the weights that connect input nodes to the nodes in the first hidden layer. Each component-wise feedforward neural network has company x_{it} in the output layer and the lagged values of companies x_{jt} with $j \neq i$ in the input layer. The edges for each vertex are identified by the Granger causal interactions defined by the sparsity induced in the objective function (2.26).

To train the component-wise feedforward neural networks, we apply the Adam optimizer with constant learning rate. Different learning rates (0.0001, 0.001, 0.01, and 0.1) are tuned and the optimal learning rate for both before and after the introduction of Tobalaba is 0.1. The initial number of hidden nodes is set equal to $\underline{z}_1 = 30$ and $\underline{z}_2 = 35$. Following Montgomery and Eledath (1995), the tuning parameters of the algorithm are $\kappa = 1$, $\mu = 0.2$, $\chi = 0.000001$ and the number of epochs is 7000. The same parameters are adopted for both subsamples. The optimal combination of α and λ is obtained by cross-validation. The domain of the two hyper-parameters was discussed earlier. As in the simulation exercise, we consider 75% of the dataset to train the network and 25% of the sample to obtain the out-of-sample RMSE associated with each combination.

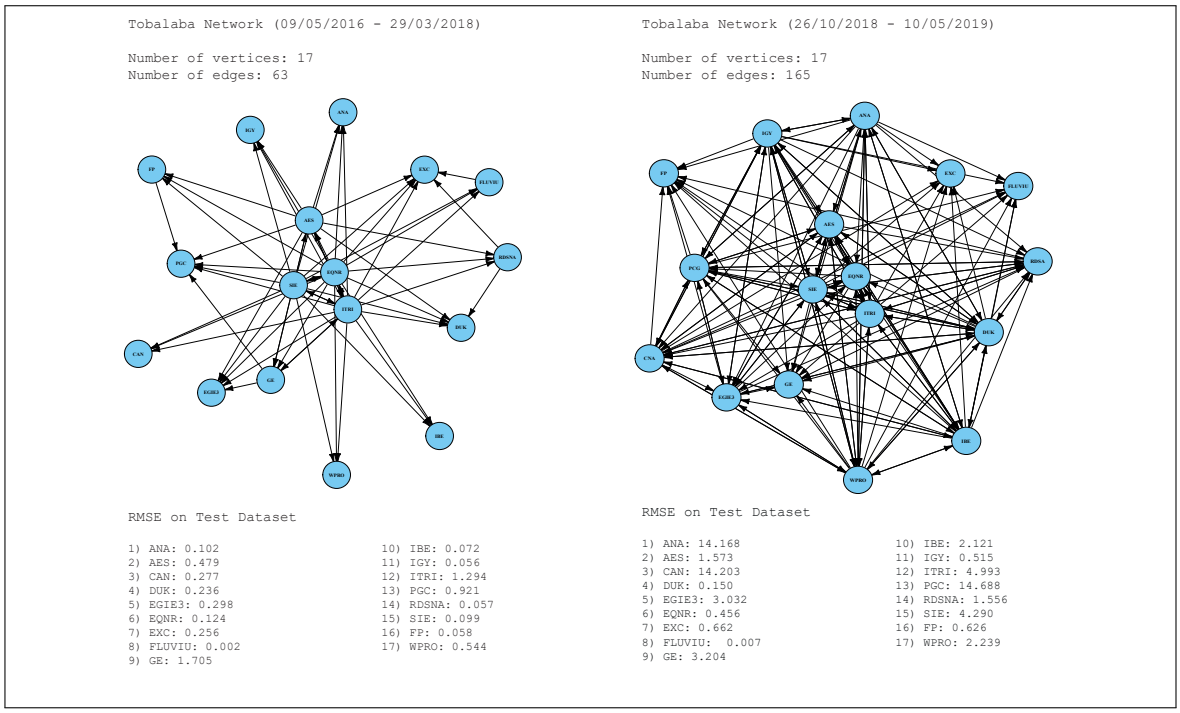


Figure 2.14: Granger causal networks before and after the introduction of Tobalaba. The out-of-sample RMSE is reported for each company.

Figure 2.14 shows an increase in the number of edges in the network after the introduction of Tobalaba (from 63 to 165) and, in particular, there is an increase in bi-directional edges. This finding clearly reveals the increase in connections after the introduction of the platform and can be justified by the introduction of the distributed ledgers and smart contracts that

allow stipulating significantly more contracts due to the reduction of transaction costs, increase in information, and absence of intermediaries.

Centrality Measures

In this section, we study different centrality measures to interpret the results with respect to the importance of the firms in the Tobalaba platform. Table 2.5 reports for each network different measures of the degree centrality, the betweenness centrality, the eigen centrality, the page rank, the in-degree and out-degree centrality reported below. The different measures reported in Table 2.5 are used to identify the central nodes in the two uncovered networks; the different centrality measures allow overcoming the absence of a general definition of centrality (Rodrigues, 2019).

Table 2.5: Centrality Measures: Degree centrality, betweenness centrality, eigen centrality, and page rank before and after the introduction of the Tobalaba network.

	ANA	AES	CNA	DUK	EGIE3	EQNR	EXC	FLUVIU	GE	IBE	IGY	ITRI	PCG	RDSA	SIE	FP	WPRO
(09/05/2016 - 29/03/2018)																	
Degree	4	13	3	5	5	18	6	4	7	3	4	19	6	5	17	4	3
In-Degree	4	3	3	5	5	2	6	3	4	3	4	4	6	3	2	3	3
Out-Degree	0	10	0	0	0	16	0	1	3	0	0	15	0	2	15	1	0
Betweennes	0.0000	1.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	19.3333	0.0000	0.0000	0.6667	0.0000	0.0000
Eigen	0.3533	0.7329	0.2802	0.3878	0.4069	0.9191	0.4199	0.3221	0.5372	0.2802	0.3533	1.0000	0.4366	0.3451	0.8904	0.2971	0.2802
Page Rank	0.0515	0.0475	0.0475	0.0724	0.0661	0.0451	0.1127	0.0475	0.0515	0.0475	0.0515	0.0626	0.1069	0.0490	0.0450	0.0480	0.0475
(26/10/2018 - 10/05/2019)																	
Degree	23	20	22	27	20	12	8	9	11	25	25	25	24	18	25	11	25
In-Degree	7	10	10	11	10	10	8	9	11	9	9	9	11	11	10	11	9
Out-Degree	16	10	12	16	10	2	0	0	0	16	16	16	13	7	15	0	16
Betweennes	0.7100	1.1445	3.0056	5.2286	1.0215	0.3270	0.0000	0.0000	0.0000	2.1262	2.4052	3.5334	17.3520	0.9917	4.5448	0.0000	2.6096
Eigen	0.8644	0.8216	0.8305	1.0000	0.7846	0.4841	0.3453	0.3879	0.4423	0.9353	0.9337	0.9306	0.9055	0.7214	0.9254	0.4586	0.9301
Page Rank	0.0448	0.0561	0.0593	0.0648	0.0561	0.0554	0.0459	0.0513	0.0829	0.0533	0.0533	0.0563	0.0830	0.0609	0.0606	0.0635	0.0526

Looking at degree centrality, defined as the number of connections relative to each node, we observe an increase in the number of links for each vertex after the introduction of Tobalaba. Before the introduction of Tobalaba, the companies *AES*, *EQNR*, *ITRI*, and *SIE* were the central nodes, while after the introduction of Tobalaba the number of central nodes increases drastically to 10[22]. However, as pointed out by Rodrigues (2019), degree centrality should be considered as a local centrality measure that does not take into account the density of the links among different nodes. In Table 2.5 we also report the in-degree and out-degree centrality statistics relevant in directed networks. The in-degree centrality defines how prominent a node is and the out-degree centrality measures the centrality of a node in the network. The reported out-degree centrality measures confirm the conclusions drawn from the other centrality measures analyzed: the introduction of the new blockchain platform increases the number of central nodes from 4 to 12. More interestingly, the directed measures of degree centrality provide useful insights regarding the interactions among the members of the network. After the introduction of Tobalaba all nodes become more receptive given by an increase in the in-degree centrality for all the analyzed companies. However, the out-degree centrality for EXC, $FLUVIU$, GE, and FP is still zero, in contrast to all the other members of the network that increase the out-degree statistic after the introduction of Tobalaba. Looking at the core activities of the members of the network, we note that EXC and $FLUVIU$ are

[22] Degree centrality higher than 20.

retail distributors of energy and as such, are expected to receive a high number of incoming transactions from companies that are either producers of energy or of the infrastructures used for distribution.

The betweenness centrality (unweighted) quantifies the importance of a node in connecting to other vertexes (Bloch et al., 2017). Table 2.5 also shows that the degree of centrality, with the exception of EXC, *FLUVIU*, *GE*, and *FP*, changes after the introduction of Tobalaba. Before the introduction of Tobalaba, the betweenness centrality identifies *ITRI* as the primary central node, implying that a removal of *ITRI* from the network would have implied a disruption of the overall network activity. Conversely, after the introduction of Tobalaba the primary central node is *PCG*. It is also interesting to see how, after the introduction of Tobalaba, the number of nodes that influence the flow of information circulating through the network increases. The betweenness centrality for the majority of the vertexes in the network is zero before the introduction of Tobalaba and increases, in most cases, after the introduction of the blockchain platform.

Eigenvector centrality (Bonacich, 1987) takes into account not only the connections of the particular node but also how many links the connected neighbors have. In other words, it measures the "*prestige*" (Bloch et al., 2017) of a node. Before the introduction of Tobalaba, the eigenvector centrality confirms the conclusions drawn from the betweenness centrality. After the introduction of Tobalaba, *DUK* is identified as the primary central node. Also in this case, it is possible to note how the degree of centrality increases for all the members of the network, with the companies *ANA*, *AES*, *CNA*, *IBE*, *IGY*, *ITRI*, *PCG*, *SIE*, and *WPRO* characterised by an eigenvalue of centrality close to unity.

Last but not least, the page rank is also analyzed. The page rank is a variant of the eigenvector centrality that takes into account also the directions of the different links. Before the introduction of Tobalaba, the page rank identifies *PCG* and *EXC* as central nodes; after the introduction of Tobalaba, the degree of centrality of the different nodes becomes more uniform, reducing the spread in page rank across the different members.

To summarize, the different centrality measures confirm an increase in the degree of centrality associated with each vertex of the Granger causal network after the introduction of Tobalaba. The increase in the number of central nodes can be associated with an increase in the number of bilateral transactions due to the newly adopted blockchain technology, thereby reducing the overall network reliance on a single central vertex, increasing its activity, robustness and reliability[23].

[23] It is worth noting that the increase in the number of central nodes, and thus in the number of critical companies and connections, may have a significant impact on the study of cascading failures based on the dependency risk methodology of Kotzanikolaou et al. (2013). The study of cascade failure and risk transmission will be object of future research.

Structure of neural network

Table 2.6 reports the optimal α, λ, and structure of the component-wise feedforward neural networks fitted to construct the networks reported in Figure 2.14. These results highlight the sensitivity of the structure of the neural network to the amount of information transmitted through it, and hence, the importance of constructing an optimal neural network prior to uncovering the presence of predictive ability between the variables. Before the introduction of Tobalaba the optimal number of nodes in the first and second hidden layers is sensibly lower than the number of hidden nodes after the introduction of Tobalaba. After the introduction of Tobalaba, the larger number of hidden nodes captures the higher number of interactions that arise between firms due to the increase in the number of bilateral (decentralized) transactions, that increases the interdependencies between operating firms, which naturally lead to a 'more dense' network architecture to capture them. The optimal construction of the neural network obtained from applying the algorithm of Montgomery and Eledath (1995) guarantees that the network does not propagate noise through the neural network and only meaningful information for the analysis of Granger causality and the predictive ability of the variables.

Table 2.6: Optimal λ and α returned from cross-validation for each of the fitted Feedforward Neural Networks. The structure of the Network selected by the Algorithm of Montgomery and Eledath (1995) is also reported.

	ANA	AES	CNA	DUK	EGIE3	EQNR	EXC	FLUVIU	GE	IBE	IGY	ITRI	PCG	RDSA	SIE	FP	WPRO
(09/05/2016 - 29/03/2018)																	
z_1	3	6	2	1	1	11	3	2	7	2	3	20	3	3	1	5	11
z_2	2	3	1	1	1	4	1	2	1	1	2	4	2	1	1	1	3
α	0.1	0.3	0.2	0.2	0.3	0.4	0.1	0.3	0.3	0.1	0.1	0.2	0.3	0.3	0.4	0.3	0.3
λ	0.3	0.2	0.7	1	0.8	0.3	0.4	0.5	0.3	0.9	0.5	0.4	0.9	0.5	0.2	0.7	0.6
(26/10/2018 - 10/05/2019)																	
z_1	28	28	17	24	24	19	15	2	29	19	2	27	30	20	17	21	18
z_2	7	8	3	7	6	4	4	2	11	5	1	7	11	5	4	5	4
α	0.2	0.4	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.1	0.3	0.1	0.2	0.1	0.1	0.3	0.3
λ	0.4	0.9	0.4	0.3	0.5	0.6	0.9	0.5	1	0.1	0.1	0.1	0.1	0.4	0.2	1	0.3

To add robustness to the results of this exercise, we also consider a reduced dataset. In particular, both subsamples are reduced to the first 25% of the observations. In these cases, the number of edges observed before the introduction of Tobalaba is 63 and after the introduction of Tobalaba is 143. These results corroborate previous findings: there is no change in the number of connections and interactions between the firms before the introduction of Tobalaba; conversely, after the introduction of Tobalaba, when the first 25% of the dataset is used, we observe a reduction in the number of edges compared to Figure 2.14, showing that the number of interactions between the firms has increased over time.

Forecast Accuracy

The original definition of lagged causality (Granger, 1969), $x_{j,t-l} \Rightarrow x_{it}$, involves an increase in forecast accuracy of time series x_{it} given the lagged values of the time series x_{jt}. The edges of the Granger causal network reported above are identified by the Granger causal interactions discovered by the objective function (2.26). Once the parameters are estimated and the weights penalized, it is possible to forecast out of sample. Consequentially, the

Granger causal network that we have uncovered in this empirical exercise can be justified as a framework for improving the forecasts of a multivariate time series of log returns of 17 firms.

We formalize this claim by comparing the component-wise forecast accuracy of the feed-forward neural network against several benchmark models. To obtain the one-step ahead forecasts, a rolling window approach is implemented. We compare the predictive ability of the constructed Tobalaba network against the different benchmark models by applying a one-sided Diebold-Mariano test (1995). The hypothesis of predictive ability can be written in terms of the mean square forecast error (MSFE) between both predictive models. For each i, we have

$$\mathcal{H}_0 : MSFE^i_{nn} \geq MSFE^i_{VAR}, \tag{2.40}$$

and the alternative is

$$\mathcal{H}_A : MSFE^i_{nn} < MSFE^i_{VAR}, \tag{2.41}$$

with $MSFE^i_{nn}$ denoting the mean square forecast error for the prediction obtained from neural networks and $MSFE^i_{VAR}$ the corresponding quantity obtained from the alternative models. Table 2.7 reports the test-statistics and the p-value of the one-sided Diebold-Mariano test (1995) for different benchmark models.

Table 2.7: Test statistics and p-values for the one sided Diebold-Mariano (1995) test.

	ANA	AES	CNA	DUK	EGIE3	EQNR	EXC	FLUVIU	GE	IBE	IGY	ITRI	PCG	RDSA	SIE	FP	WPRO
(09/05/2016 - 29/03/2018)																	
VAR(10) - AIC																	
DM t-stat	0.7109	3.5557	-0.3475	4.1242	3.4742	2.7946	1.8170	1.7894	3.3899	-0.1324	5.9087	4.2003	3.7315	0.6025	1.1613	0.5325	2.8867
P-value	0.2398	0.0003	0.6354	<.0001	0.0005	0.0034	0.0367	0.0390	0.0006	0.5526	<.0001	<.0001	0.0002	0.2744	0.1248	0.2980	0.0025
VAR(10) - SC/BIC																	
DM t-stat	0.8314	1.4418	-0.3375	5.0691	3.2742	1.8498	3.7063	-0.0756	1.2011	-0.2029	5.1596	4.0094	4.4937	0.6554	0.5775	0.97435	2.4512
P-value	0.2043	0.0770	0.6316	<.0001	0.0009	0.0343	0.0002	0.5301	0.1169	0.5801	<.0001	<.0001	<.0001	0.2572	0.2827	0.1666	0.0008
VAR(10) - H. Lasso																	
DM t-stat	0.5715	2.8837	3.2669	4.6904	5.5815	1.8014	4.3471	-1.3607	1.8002	-0.4502	5.9700	5.7816	2.2178	-0.3449	1.0025	-0.7035	6.7251
P-value	0.2848	0.0026	0.0008	<.0001	<.0001	0.038	<.0001	0.9110	0.0381	0.6730	<.0001	<.0001	0.0149	0.6344	0.1598	0.7580	<.0001
ARIMA																	
DM t-stat	0.2950	1.6813	-0.1464	4.0966	1.3429	0.6443	4.2751	-1.1433	3.3477	3.1960	1.9294	2.4249	1.5788	1.7826	1.9389	1.5764	4.4666
P-value	0.3844	0.04862	0.558	<.0001	0.0918	0.2608	<.0001	0.8761	0.0006	0.0011	0.0289	0.0009	0.0598	0.0395	0.0283	0.0598	<.0001
(26/10/2018 - 10/05/2019)																	
VAR(10) - AIC																	
DM t-stat	4.4708	5.4614	4.4851	5.3019	7.1449	5.2264	5.3268	3.7235	6.3085	4.6958	4.6995	4.0900	5.8294	6.0783	3.7364	5.5485	5.7272
P-value	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	0.0002	<.0001	<.0001	<.0001	<.0001	<.0001	<.0001	0.0002	<.0001	<.0001
VAR(10) - SC/BIC																	
DM t-stat	5.5601	3.5373	4.1454	3.5823	4.2906	2.3727	4.8180	3.5271	5.7450	4.8650	3.6427	4.6288	4.4794	4.9785	3.3436	4.1410	5.3268
P-value	<.0001	0.0004	<.0001	0.0003	<.0001	0.0102	<.0001	0.0004	<.0001	<.0001	0.0003	<.0001	<.0001	<.0001	0.0007	<.0001	<.0001
VAR(10) - H. Lasso																	
DM t-stat	4.1515	-1.3573	2.6058	0.0533	3.8638	2.5957	1.6365	-0.36215	-0.29011	3.9205	-1.5090	7.9074	2.1622	1.7082	3.9536	-0.5101	3.7499
P-value	<.0001	0.9105	0.0056	0.4788	0.0001	0.0057	0.0531	0.6408	0.6137	0.0001	0.9321	<.0001	0.01703	0.0460	<.0001	0.6942	0.0002
ARIMA																	
DM t-stat	-0.6447	1.3833	1.2962	3.2208	2.3887	1.6395	3.3682	0.87364	4.8382	-0.4006	2.6907	2.2173	0.7802	1.8874	0.9494	0.3256	3.4211
P-value	0.7394	0.0855	0.0996	0.0009	0.0098	0.0523	0.0006	0.1927	<.0001	0.6550	0.0044	0.0149	0.2190	0.0316	0.1729	0.3728	0.0005

In the absence of a relevant model for an unknown data generating process, the linear VAR(K) is chosen as the first benchmark as it can be considered the best linear approximation of a process that may be nonlinear. Moreover, Plagborg-Møller and Wolf (2019) show how, for a large number of lags, a VAR(K) is as robust to nonlinearities as the linear projection.

The maximum lag-length allowed in the VAR(K) is 10; the optimal lag is selected using the AIC scores. The top panel in Table 2.7 corresponds to the period before the introduction of the Tobalaba network and the bottom panel corresponds to the period afterward. For the first period, we fail to reject the null hypothesis of equal forecast ability in six cases, at a 0.05 significance level. However, after the introduction of Tobalaba, the null hypothesis is rejected in all cases. The higher forecast accuracy of the feedforward neural network for each vertex is a result of including nonlinear interactions between the variables and also a potentially larger persistence compared to the VAR(10) model.

As a robustness exercise, we also propose three different benchmark models that compete against our neural network specification. First, we select the optimal lag of our VAR(K) model using the BIC score instead of the AIC score[24]. In addition to the model obtained from the BIC score, we also consider two alternative benchmarks given by a linear VAR(K) model estimated using component-wise hierarchical group lasso, as in Nicholson et al. (2014). This benchmark defines the best linear VAR alternative that can be considered in such high-dimensional multivariate time series. This model is, therefore, a suitable alternative strategy in large dimensions for our feedforward neural network. Finally, we also consider an ARIMA(p, d, q) process; this model does not accommodate any feedback effect from other input variables and, hence, fails to incorporate Granger causal interactions. The predictive ability of these models is compared, as before, using Diebold-Mariano (1995) tests in Table 2.7. The choice of these benchmarks allows us to understand different aspects regarding the performance of the proposed methodology in uncovering Granger causal relations.

The results show that the forecasts of the VAR(K) model obtained using the BIC score have similar predictive ability to the VAR(K) model using the AIC score. Both models fare poorly with respect to the feedforward neural network in terms of predictive ability. The second benchmark is given by a high-dimensional VAR - optimized over one-step ahead forecasts - with the component-wise hierarchical group lasso proposed by Nicholson et al. (2014). Being both procedures - Nicholson et al. (2014)'s VAR benchmark and our methodology - able to accommodate high-dimensional systems, the results in Table 2.7 report statistically significant differences in one-step ahead forecasts between the two models. These results provide evidence that the neural network is able to outperform state-of-the-art high-dimensional linear VARs with induced sparsity via convex regularizers. The main reason for this is the ability of feedforward neural network models to capture the nonlinearities in the underlying data generating process. Finally, an ARIMA(p, d, q) is used to further validate the Granger causal network discovered with our novel methodology against a time series linear model exhibiting no Granger causality. The results reported in the bottom panels of Table 2.7 provide further empirical support to the feedforward neural network model.

[24] Lütkepohl (1985) shows - in his simulation study with VAR models - that the BIC outperforms other model selection criteria by choosing the correct autoregressive order and by returning the smallest mean squared forecast error from one-step ahead forecasts

2.8 Conclusions

This chapter has proposed a new methodology for the detection of Granger causality in a vector autoregressive setting using feedforward neural networks. To do this, we have constructed an optimal neural network that maximizes the mutual information transfer between input and output nodes. In a second stage, we propose a novel objective function that introduces sparsity in high-dimensional systems and controls for the number of connections between the input variables and the nodes in the first hidden layer. The newly proposed objective function detects the Granger causal interactions between the variables and also the optimal lag length associated with each input variable, allowing different lag orders for each endogenous time series.

The simulation study shows in finite samples the importance of using an optimal network structure to reduce type I and type II errors. In particular, we show that the number of nodes in the first hidden layer has a significant impact on the correct detection of Granger causal interactions. Our simulations also show the consistency of the algorithm used to detect the optimal number of nodes in each hidden layer as the sample size increases. We compare the performance of our approach against a hierarchical group lasso penalty function. The results show clear evidence of the outperformance of our method to detect Granger causality.

The empirical application shows that after the introduction of the Tobalaba network there is an increase in the number of edges among the 17 companies studied. Moreover, the centrality measures obtained show an increase in the number of central nodes in the network after the introduction of the new platform. Our results demonstrate how the introduction of the blockchain platform has changed the structure of the connections between the firms trading in the platform due to the introduction of smart contracts and disintermediation. The application of the Diebold-Mariano test (1995) shows that the Granger causal network constructed using the algorithm proposed in this chapter outperforms, in terms of forecast accuracy, several linear VAR(K) models in low and high dimensions, and provide empirical evidence on the importance of our nonlinear method for forecasting.

Being the chapter focused on Granger causality and thus forecasting, estimation and inference within regularized neural networks was not analyzed. The recent contribution by Hecq et al. (2019) develops an LM test for Granger causality in high-dimensional VAR models based on penalized least squares estimations. To obtain a test retaining the appropriate size after the variable selection done by the lasso, these authors propose a post-double selection procedure to partial out effects of nuisance variables and establish its uniform asymptotic validity. Although the method performs very well in high-dimensional settings, it is devised for linear parametric settings. In contrast, the method presented in this chapter based on detecting Granger causality through sparsity induction presents a nice alternative that works in more general settings. Future research will extend the current work to the derivation of nonasymptotic bounds for regularized and non-regularized neural networks (Farrell et al., 2018), as well as the limiting distributions for the two-step estimator proposed in this chapter.

Another potential extension of the current research is to couple our methodology to detect Granger causality with the graphic theory introduced by Eichler (2007) and Eichler and Didelez (2012). The main advantages of the analysis of Granger causality in graph theory are the possibility of visualizing the complex dependence structures that may underline multivariate time series, and a definition of Granger causality that can be applied to multivariate time series with nonlinear dependencies. Therefore, by analyzing the matrix of the weights and error terms of a VAR defined by our feedforward neural networks approach, it may be possible to define the directed and undirected edges in a mixed path diagram in high-dimensional and potentially nonlinear time series.

CHAPTER 3

Optimal deep neural networks by maximization of the approximation power

Chapter Abstract

This chapter proposes an optimal architecture for deep neural networks of given size. The optimal architecture obtains from maximizing the minimum number of linear regions approximated by a deep neural network with a ReLu activation function. The accuracy of the approximation function relies on the neural network structure, characterized by the number, dependence and hierarchy between the nodes within and across layers. For a given number of nodes, we show how the accuracy of the approximation improves as we optimally choose the width and depth of the network. More complex datasets naturally summon bigger-sized architectures that perform better applying our optimization procedure. A Monte Carlo simulation exercise illustrates the outperformance of the optimized architecture against cross-validation methods and gridsearch for linear and nonlinear prediction models. The application of this methodology to the Boston Housing dataset confirms empirically the outperformance of our method against state-of the-art machine learning models.

3.1 Introduction

Neural networks and, more specifically, deep learning models are extremely popular in high-dimensional problems such as pattern recognition, biomedical diagnosis, and others (see Schmidhuber, 2015; and LeCun, et al. 2015 for overviews of the topic). The success of these techniques rests in their ability to approximate complex unknown functional forms for the relationship between the outcome variable and the predictors. The Universal Approximation Theorem (Cybenko, 1989) states that a shallow feedforward neural network with enough hidden units and sigmoidal activation function can approximate any Borel measurable function with an arbitrarily small error. The implications of this result are noteworthy: notwithstanding the type of nonlinearity that characterizes the data generating process, a sufficiently large shallow feedforward neural network will be able to approximate, accurately, the underlying function (Goodfellow et al., 2016). In this setting, it is no longer necessary to construct an *ad hoc* model for the specific nonlinearity to be learned.

Universal approximation theorems have been proved for both shallow (one hidden layer) and deep (more than one hidden layer) learning networks. Although the accuracy of the approximation is not well understood yet, it is widely accepted in the literature that deeper models can reduce the number of computational units (hidden units) required to approximate the target function by a factor that is exponential in the depth of the feedforward neural network (Montufar et al., 2014). Similarly, based on the literature of piecewise functions and function oscillation, Tewlgarsky (2016) shows that - conversely to shallow networks - the function composition characterizing deeper architectures returns highly oscillatory functions and thus, it is better suited to fit an unknown target function compared to shallow structures of similar size. Deeper architectures, by capturing and learning the repeated regularities or hierarchical structures in the data, allow learning the underlying function with a lower level of model complexity (Aggarwal, 2018). Poggio et al. (2017) state that since shallow networks can be considered a special case of deep networks, any continuous function that can be represented by a shallow network (universal approximator) can also be represented by a deep network arbitrarily well. Hanin and Sellke (2017) show that deep feedforward neural networks with a ReLu activation function (ReLu DNN) and bounded width are universal approximators.

Based on the Universal Approximation Theorem, it is possible to state that the approximation power of neural networks depends on the number of hidden units and on their allocation across layers, or network's depth (Montufar et al., 2014; Pascanu et al., 2013; Tewlgarsky, 2016; Raghu et al, 2017). The correct specification of the width and depth of the network, together with other factors, ensures an efficient balance between the *variance* and the *bias* of the model and thus, it ensures that the model will not over- or under-fit the underlying function. It is standard in the literature on DNN to use k-fold cross-validation methods to determine the width and depth of the network. In this scenario both variables are treated as model hyperparameters that are optimized while training. However, the performance of cross-validation approaches depends on (i) the number of variants of the learning models - if

too few models are tuned we may miss the global optimum or if too many we may over-fit (Rao et al., 2008); (ii) the optimum division of the folds - observations must be *i.i.d.* and the distribution of the target variable must be similar across different folds; and (iii) the number of observations available - too few and it is not possible to correctly implement k-fold cross-validation (among the possible solutions it is important to mention the Out-of-Bag procedure that allows out-of-sample model evaluation using the observations not sampled during bootstrapping).

In this chapter, we show that it is possible to optimize the network architecture based on recent theoretical contributions that characterize the minimum expressive power of DNN with ReLu activation functions. Pascanu et al. (2013) and Montufar et al. (2014) provide a characterization of the lower bound for the number of affine regions approximated by a ReLu DNN[25]. The number of linear regions of the functions that can be approximated is a measure of the flexibility of the network architecture under consideration. For network architectures of a given size, we show that it is possible to optimize their width and depth by allocating the hidden units in such a way that the minimum number of linear regions approximated is maximized, effectively forcing the network architecture to be maximally flexible when approximating the unknown target function. Maximum flexibility is achieved when, for a given number of hidden nodes, the recursive folding of the input space is optimized, ensuring maximum efficiency in approximating the underlying data generating process. In doing so, we reduce the set of hyperparameters to be determined by cross-validation to the learning rate, number of epochs and drop-out rates. By optimizing width and depth prior to training architectures of a given size, our proposed method substantially saves upon the necessary time and computing power involved in fine tuning while training. More importantly, we reduce the approximation error and improve, in turn, the predictive ability of the DNN. Naturally, bigger and more complex datasets call for bigger sized network architectures in terms of hidden units.

The predictive ability of our approach is superior to state-of-the-art methods widely used in the DNN literature such as cross-validation and randomized gridsearch. To show this, we carry out a Monte Carlo exercise that evaluates the out-of-sample performance of deep neural networks equipped with our optimal architecture against different benchmarks, depending on the data generating process considered. The main competitors that we consider for nonlinear data generating processes are k-fold cross-validation and randomized gridsearch. The simulation section also considers a linear data generating process that helps us understand the limits of our optimization procedure. We do this by comparing the predictions of our optimal DNN against the predictions of ordinary least square (OLS) methods, that are known to be best linear unbiased estimators. Interestingly, the out-of-sample mean square prediction error and mean absolute prediction error obtained from our ReLu DNN remain competitive and are comparable to those obtained from OLS methods in this setting.

[25] The upper bound, or maximal number of linear regions of a function approximated by a network architecture with rectified linear units of size N, is equal to 2^N (Proposition 4 in Montufar et al., 2014). Therefore, it is common to shallow and deep architectures of equal size N.

The methodology is also illustrated empirically with a simple example widely used in the machine learning literature. We apply our optimal architecture to a ReLu DNN with the aim of predicting median house prices from the Boston Housing Dataset. The dataset originally adopted by Harrison and Rubinfeld (1978) for modeling willingness to pay for clean air in the Boston area is extensively used by the machine learning literature to validate new learning techniques; such as Al Bataineh and Kaur (2018), Papadopoulos and Haralambous (2011), Granitto et al. (2001), Nado et al. (2018), Bakker and Heskes (2003), Myshkov and Julier (2016), and Zhou et al. (2001). The optimal structure selected by our novel methodology is a DNN with three layers and nodes equal to $[52, 39, 39]$. Our results provide an improvement in prediction accuracy (MSE) in between 28.55% and 47.32% in comparison to those studies and highlight the performance gains of our optimal DNN for prediction relative to the aforementioned works.

The accuracy of the approximation function depends on the number of nodes in the neural network. Our method is flexible on the number of nodes and allows to experiment with different choices depending on the complexity of the problem and dimension of the dataset. Our optimization method suggests that the predictive accuracy achieved by the optimal architecture of the DNN improves further as we consider larger datasets which naturally call for more nodes in the neural network. Therefore, and although we have not explored it in detail here, it is presumed that savings in computational time will increase more than proportionately with bigger and more complex datasets requesting bigger sized architectures.

Our contribution is related to recent and influential literature discussing the suitability of different neural network architectures. In particular, Montufar et al. (2014) explain how each hidden layer of a deep ReLu neural network can be associated to a *folding operator*. Figure 3.1 reports an example of a recursive folding operation performed by the hidden layers of a neural network (Montufar et al., 2014).

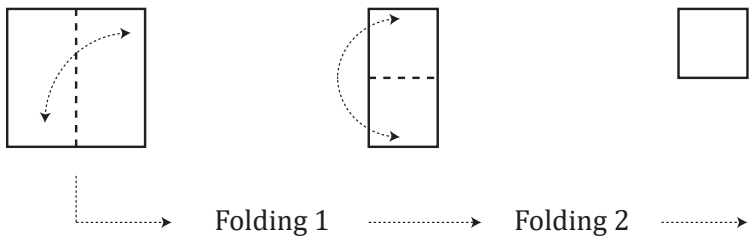


Figure 3.1: The Figure reports an example of recursive folding as in Montufar et al. (2014)

If each hidden layer folds the output of the previous hidden layer, the output layer of a ReLu neural network folds –starting from the first hidden layer– the input layer (space) recursively. Thus, the recursive partitioning implies that functions computed in the final

folded space will be effectively applied to all the collapsed subsets identified by the collections of foldings (see also Figure 3.1).

From the above argumentation, one could conclude that the composition of the folding operations –computed by each hidden layer in a feedforward neural network– ensures approximating a number of linear regions which is exponential in the depth of the network. More formally, theorem 4 in Montufar et al. (2014) shows how an increase in depth ensures an exponential increase in the number of linear regions being approximated while an increase in width a polynomial increase. Therefore, Montufar et al. (2014) suggest that an increase in the depth of a ReLu neural network leads to a gain in accuracy higher than the one resulting from an increase in width. Theoretical studies such as Montufar et al. (2014), Pascanu et al. (2013) and Goodfellow et al. (2016) claim that ReLu deep neural networks (DNN) should perform better than shallow networks. Alternatively, as suggested in Goodfellow et al. (2016), one could choose to implement a deep or shallow neural network for *statistical reasons* (e.g., tree-based architectures allow modeling nonlinearities through multiplicative relationships between features and thus have been effectively adopted in clinical and economic applications where the analyst is interested in the heterogeneity around some treatment effect). In other words, the choice of the specific machine learning algorithm (i.e., tree structures, shallow or deep networks, SVM, recurrent or feedforward networks) depends on a set of prior beliefs that the analyst has regarding the type of nonlinearity to be learned. Given Definition 3.2.1, one could infer that choosing a deep structure implies assuming that the underlying data generating process can be represented by a composition of simpler functions. Specifically, as expressed in Goodfellow et al. (2016), the unknown underlying data generating process can be approximated by discovering a subset of underlying components which can be in turn described by a different subset of simpler factors of variations. However, even if the choice of the deep neural network structure is dictated by the prior belief regarding the underlying data generating process, it is crucial to identify an optimal node allocation strategy that allows maximizing the efficiency in which a given architecture is able to capture all the regular or repeating patterns in the underlying data generating process.

A recent review article by Kraus et al. (2020) compares the performance of deep learning models to conventional models from machine learning (i.e., lasso, random forests, and SVMs) and shows the improvements over the latter models in both prediction and operational performance. These authors note that default, out-of-the-box architectures (i.e., not optimally selected/tuned) are characterized by high bias and propose a deep-embedded neural network architecture that specifically addresses the need of effectively handling categorical variables and their concatenated embedding. Although we pursue a similar objective, our strategy for optimizing the neural network is different. The aim of our study is to implement a data-free optimization procedure that delivers an optimal width and depth of a given size network architecture before training it with data. More specifically, the number of hidden nodes (size) of the neural network is dictated by the complexity of the underlying data generating process to be approximated, and the optimization proposed in the present chapter allows maximizing the efficiency in which a neural network architecture (of a given size) can partition (recursively)

the input space and thus, the efficiency in which a neural network approximates the linear regions characterizing the underlying data generating process.

Another branch of the literature on deep neural networks focuses on extending the Universal Approximation Theorem to the unbounded and non-continuously differentiable ReLu activation function (see Guilhoto, 2018). Modern neural networks use the ReLu activation function due to the ease in training shallow and deep networks, as opposed to sigmoidal activation functions that make the training arduous (Géron, 2017). However, it has been shown that shallow networks, despite being universal approximators, may fail to learn and generalize the data generating process correctly due to the unfeasibly large number of hidden nodes required to approximate the underlying function (Goodfellow et al., 2016; Barron, 1993).

The rest of the chapter is organized as follows: Section 3.2 reports the definitions and notations used in the chapter, the Universal Approximation Theorem, and function approximation by step functions. Section 3.3 discusses the novel methodology used for structure identification by reporting the numerical results of the analyzed maximization. Section 3.4 presents a Monte Carlo simulation exercise that provides empirical evidence in finite samples of the performance of our method in improving the out-of-sample goodness of fit of deep neural networks for different data generating processes and input dimensions. In Section 3.5 the novel methodology is applied to the Boston Housing dataset. Section 3.7 concludes.

3.2 Universal approximation theorem

Subsection 3.2.1 introduces relevant definitions and notations. Subsection 3.2.2 analyzes and explains the Universal Approximation Theorem and the relation that subsists between piecewise linear functions (PWL) and ReLu DNN. Subsection 3.2.3 formalizes the relation between function approximation and neural network structure. Subsection 3.2.4 reports the main theory analyzing the number of linear regions approximated by both shallow and deep neural networks.

Let y_i for $i = 1, \dots, n$ denote the outcome variable of interest and $\mathbf{x}_i = (x_{1i}, \dots, x_{di})$ a set of input variables used to predict the outcome. A general predictive model is expressed as

$$y_i = f(\mathbf{x}_i, \varepsilon_i), \tag{3.1}$$

where $f(\cdot, \cdot)$ is a real-valued function used to predict the outcome variable and ε_i denotes the set of unobserved variables. The contribution of the different variables is assumed to be additive such that we obtain

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \tag{3.2}$$

with ε_i interpreted as an error term. The choice of the functional form $f(\mathbf{x}_i)$ depends on the loss function $L(y_i, f(\mathbf{x}_i))$ penalizing the difference between the outcome variable and the prediction. For example, it is well known that if the loss function is quadratic then

the best predictive model is $f(\mathbf{x}_i) = E[y_i \mid \mathbf{x}_i]$. Similarly, if the loss function is the check function used in quantile regression then the optimal prediction is given by the conditional quantile $f(\mathbf{x}_i) = Q_\tau[y_i \mid \mathbf{x}_i]$, with $\tau \in (0,1)$. Other prominent examples in the machine learning literature include classifiers such as logistic regression, linear discriminant analysis, kernel methods as support vector machines, tree-based methods such as decision trees and their generalization to random forests, and nonparametric regression in the spirit of nearest neighbors and local kernel smoothing.

The question of interest is to approximate the unknown function $f(\mathbf{x})$. Under the assumption that the function $f(\mathbf{x})$ is linear on the observable input variables, OLS estimation provides unbiased, consistent and efficient estimators of the coefficients associated to $\mathbf{x}$. Nonlinear specifications of $f(\mathbf{x})$, the presence of unobserved heterogeneity, high-dimensional problems and complex datasets, require of alternative methods to approximate the function $f(\mathbf{x})$. Recent advances in machine learning have shown that neural network methods provide accurate predictions that do not require of specific knowledge of the data generating process and, hence, are not affected by model misspecification and related econometric issues. Neural networks approximate the function of interest by combining and composing different types of activation functions over one or more layers of nodes.

3.2.1 Definitions and Notations

We will consider throughout the family of Rectified Linear Unit (ReLu) activation functions. These functions have proved recently more suitable to deep learning problems than sigmoidal activation functions (Jarrett et al., 2009; Nair and Hinton, 2010; Goodfellow et al., 2016; and Géron, 2017). A ReLu activation function is defined as follows. Let $\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$, with

$$\theta(\mathbf{x}) = (\max\{0, x_1\}, \max\{0, x_2\}, \cdots, \max\{0, x_d\}),$$

where d is the dimension of the input variables. One prominent advantage of ReLu functions is the projection property given by $\theta \circ \theta = \theta$, where $\circ$ denotes function composition (Schmidt-Hieber, 2017).

The corresponding DNN for ReLu activation functions is defined as follows.

Definition 3.2.1. (**ReLu DNN**)
For any two natural numbers d, $n_2 \in \mathbb{N}$, which are called input and output dimension respectively, a $\mathbb{R}^d \to \mathbb{R}^{n_2}$ ReLu DNN is given by specifying a natural number $N \in \mathbb{N}$, a sequence of N natural numbers $Z_1, Z_2, \cdots, Z_N$ and a set of $N+1$ affine transformation $T_1 : \mathbb{R}^d \to \mathbb{R}^{Z_1}, T_i : \mathbb{R}^{Z_{i-1}} \to \mathbb{R}^{Z_i}$ for $i = 2, \cdots, N$ and $T_{N+1} : \mathbb{R}^{Z_N} \to \mathbb{R}^{n_2}$. Such a ReLu DNN is called a $(N+1)$-layer ReLu DNN, and is said to have N hidden layers. The function $G : \mathbb{R}^d \to \mathbb{R}^{n_2}$ represented by this ReLu DNN is:

$$G = T_{N+1} \circ \theta \circ T_N \circ \cdots \circ T_2 \circ \theta \circ T_1. \tag{3.3}$$

The *depth* of a ReLu DNN is defined as $N+1$. The width of the n^{th} hidden layer is Z_n, and the *width* of a ReLu DNN is $\max\{Z_1, \cdots, Z_N\}$. The *size* of the ReLu DNN is $Z = Z_1 + Z_2 + \cdots + Z_N$. The number of active weights (different from zero) in the n^{th} hidden layer of a fully connected ReLu DNN is $w_n = Z_n \times Z_{n-1}$. The *number of active weights* in a fully connected ReLu DNN is then $w_1 + w_2 + \cdots + w_N$. Moreover, we have that:

$$T_n = \mathbf{W}^{(n)}\mathbf{x} + \mathbf{b}^{(n)}, \tag{3.4}$$

where - for $N = 1$ - $\mathbf{W}^{(n)} \in \mathbb{R}^{Z_1 \times d}$, $\mathbf{x} \in \mathbb{R}^{d \times 1}$ is the input layer, and $\mathbf{b}^{(n)} \in \mathbb{R}^{Z_1}$. For $N \neq 1$, $\mathbf{W}^{(n)} \in \mathbb{R}^{Z_n \times Z_{n-1}}$, $\mathbf{x}$ is the value from the previous hidden layer $\mathbf{h}_{n-1} \in \mathbb{R}^{Z_{n-1}}$, and $\mathbf{b}^{(n)} \in \mathbb{R}^{Z_n}$.

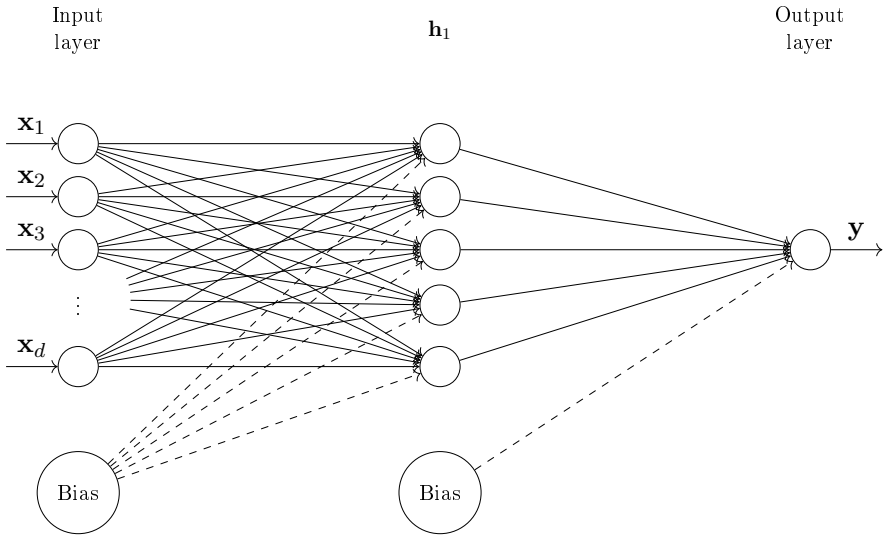


Figure 3.2: Shallow ReLu neural network

The ReLu activation function can be also formulated as $\theta(s) = \mathbb{I}(s > 0) \cdot s$, where $\mathbb{I}(s > 0)$ is an indicator function that takes a value of 1 if the argument is true and zero, otherwise. Using this characterization of the activation function, see Pascanu et al. (2013), we can define a single hidden layer ReLu neural network $G : \mathbb{R}^d \to \mathbb{R}$ as

$$G(\mathbf{x}) = \boldsymbol{\omega}_O^\top \text{diag}\left(\begin{bmatrix}\mathbb{I}(\mathbf{W}_1^{(1)}\mathbf{x} + b_1^{(1)}) \\ \vdots \\ \mathbb{I}(\mathbf{W}_{Z_1}^{(1)}\mathbf{x} + b_{Z_1}^{(1)})\end{bmatrix}\right)(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + b^O, \tag{3.5}$$

where $\mathbf{W}_j^{(1)}$ identifies row j for $j = 1, \cdots, Z_1$ of the matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{Z_1 \times d}$, $\mathbf{x} \in \mathbb{R}^{d \times 1}$ is the

input layer, $b_j^{(1)} \in \mathbb{R}$ is the element j for $j = 1, \cdots, Z_1$ of the random vector $\mathbf{b}^{(1)} \in \mathbb{R}^{Z_1}$, $\boldsymbol{\omega}_O \in \mathbb{R}^{Z_1}$, and $b^O \in \mathbb{R}$.

Definition 3.2.2. (**Piecewise linear functions (Arora et al., 2016)**)
We say a function $G : \mathbb{R}^d \to \mathbb{R}$ is *continuous piecewise linear (PWL)* if there exists a finite set of closed sets whose union is $\mathbb{R}^d$, and G is affine linear over each set (note that the definition automatically implies the continuity of the function). The number of pieces of G is the number of maximal connected subsets of $\mathbb{R}^d$ over which G is affine linear.

Note that the notion of number of pieces where G is affine linear and number of linear regions are used as synonyms in the rest of the chapter.

Based on Equation 3.5, it is possible to clarify the choice of the ReLu activation function. In particular, one could notice how ReLu activation functions have two operation nodes: they can be either constant and equal to zero or positive and equal to the linear affine transformation to which they are applied. Therefore, for each ReLu activation function, there is a hyperplane that defines the boundary between these two regions, and each hidden layer can be seen as an arrangement of hyperplanes. It follows that a shallow ReLu neural network can be regarded as an arrangement of hyperplanes and a deep Relu neural network as a composition of arrangements of hyperplanes. This result has two important implications that are analyzed more in details in the following sections: I) shallow and deep ReLu neural networks are piecewise linear functions and as such are universal approximators, II) by extending the results of Zaslavky (1975), it is possible to quantify the number of linear regions being approximated by a ReLu neural network and thus, to assess the efficiency of the recursive partition of the input space. As other widely adopted activation functions such as i) Sigmoid $\theta(x) = (1 + e^{-x})^{-1}$, ii) Softplus, $\theta(x) = \log(1 + e^x)$, or iii) Tangent, $\theta(x) = (e^{2x} - 1)/(e^{2x} + 1)$ cannot be identified by a hyperplane, the methodology proposed in the present chapter is restricted only to ReLu activation functions.

As a final remark, it is important to stress that if the ReLu activation functions are more suitable for training deep neural networks, they are still subject to the *dying ReLu* problem. That is, due to a large learning rate, the output of the ReLu activation function remains effectively zero (i.e., the gradient of the ReLu function is equal to zero when the input is negative leading to no update from the backpropagation algorithm). If this eventuality is observed, a possible solution is to adopt the leaky ReLu activation function $\theta(x) = \max\{\alpha x, x\}$ (for $\alpha = 0$, the leaky ReLu is the ReLu activation function). In this case, it is preferable to apply a small leak ($\alpha \approx 0.01$) rather than a huge leak ($\alpha \approx 2$) in order to preserve the theoretical results from Montufar et al. (2014).

3.2.2 Universal Approximation Theorem

In this section we review the literature that shows under different activation functions the existence of a universal approximation function in neural network settings.

Cybenko (1989) defines a shallow network with activation function θ:

$$G(\mathbf{x}) = \sum_{j=1}^{Z_1} \omega_j^O \theta(\mathbf{W}_j^{(1)}\mathbf{x} + b_j^{(1)}). \tag{3.6}$$

This author proves that this function approximates arbitrarily well the unknown function $f(\mathbf{x})$ over the n-dimensional unit cube[26]. In the remainder of the chapter, $f(\mathbf{x})$ identifies the unknown underlying function to be approximated, and $G(\mathbf{x})$ the shallow or deep neural network function used to approximate $f(\mathbf{x})$. This is formalized in the following lemma that is stated without proof. Interested readers are referred to Cybenko (1989).

Theorem 1 (Universal Approximation Theorem by Cybenko).

Let θ be any continuous discriminatory function and $C(I_n)$ be the space of continuous functions on I_n, with I_n an n-dimensional unit cube. Then, the finite sum of the form (3.6) is dense in $C(I_n)$. In other words, given any $f(x) \in C(I_n)$ and for any $\epsilon > 0$, there is a sum of the above form, for which

$$\sup_{\mathbf{x} \in I_n} |f(\mathbf{x}) - G(\mathbf{x})| < \epsilon. \tag{3.7}$$

The Universal Approximation Theorem formulated by Cybenko (1989) applies to all bounded Borel measurable sigmoid functions. Leshno et al. (1993) extend the above theorem to *deep* feedforward neural networks proving that a multilayer neural network can approximate any continuous function as long as the activation functions are bounded and not polynomial. Hornik (1991) broadens the literature related to the Universal Approximation Theorem proving that multilayer perceptrons can approximate arbitrarily well any function, provided that a sufficiently large enough number of hidden nodes is used. The results obtained by Hornik (1991) improve over the previous literature by proving that the capability of approximating arbitrarily well any given function depends mainly the number of hidden nodes, and not on the particular activation function used. Therefore, failing to converge to an optimal solution can be attributed to inadequate training or a low number of hidden nodes, imposing a –sometimes unfeasible– number of hidden nodes to be used when a shallow architecture is selected.

Complementary to the findings of Hornik (1991), a recent branch of the literature based on mean-field approximation (Sirignano and Spiliopoulos, 2018; Mei et al., 2018; and Tewlgarsky, 2016) that analyzes neural networks trained with Stochastic Gradient Descent (SGD) prove that the distribution of the network parameters converges to the solution of a given partial differential equation. In particular, Mei et al. (2018) complete the results of Hornik (1991) by proving that for shallow networks with $Z \gg d$ and trained with SGD, an arbitrary choice of Z ensures convergence to a solution close to the optimum. These results imply, also, that shallow networks with $Z \gg d$ do not overfit even if the number of nodes grows to infinity. Lastly, Lu et al. (2017) complement the results provided by Hornik (1991) by extending the

[26] The representation in 3.6 is equivalent to Equation 3.5 where the output of the shallow ReLu neural network is expressed using matrix multiplication, and the ReLu activation function θ as an indicator function. When reporting the Universal approximation theorem the authors express Equation 3.5 as in (3.6) to allow comparability with Cybenko (1988).

Universal Approximation Theorem to width-bounded ReLu DNN, with a minimum depth equal to $d+4$, where d is the number of input variables and, hence, of nodes in the input layer. Based on Lu et al. (2017), it is possible to conclude that both shallow and deep neural networks - notwithstanding the activation function used - are universal approximators.

It is possible to extend the proof of the Universal Approximation Theorem to ReLu neural network by proving that ReLu activation functions - for a given structure - are bounded. In particular, Guilhoto (2018) constructs the following bounded continuous activation function $\theta^*(x)$ by subtracting two ReLu activation functions:

$$\theta^*(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \in [0,1] \\ 1 & \text{if } x > 1 \end{cases} \tag{3.8}$$

Any function of the form $\theta^*(wx+b)$ can be defined as:

$$\theta^*(wx+b) = ReLu(wx+b) - ReLu(wx+b-1). \tag{3.9}$$

Another branch of the literature extends the Universal Approximation Theorem to ReLu DNN using approximation results for PWL functions. More formally,

Lemma 1 (Proposition 4.2 from Goodfellow et al. (2013b))**.**
Let C be a compact domain $C \subset \mathbb{R}^n, G : C \to \mathbb{R}$ be a continuous function, and $\epsilon > 0$ be any positive real number. Then, there exists a continuous piece-wise linear (PWL) function G, such that

$$\sup_{x \in C} |f(x) - G(x)| < \epsilon. \tag{3.10}$$

This result shows that PWL functions are universal approximators. Arora et al. (2016) prove that every $\mathbb{R}^d \to \mathbb{R}$ PWL function can be represented by a ReLu DNN (see their Theorem 2.1), by showing the equivalence between the lattice representation of PWL functions and ReLu DNNs. The results of Arora et al. (2016) coupled with Lemma 1 demonstrate that ReLu DNNs are, indeed, universal approximators[27].

Thus, regardless the data generating process considered, both architectures of the neural network are able to approximate the function $f(\mathbf{x})$ arbitrarily well. Empirically though, there are differences in the quality of the approximations between deep and shallow network structures (Larochelle et al., 2007; Novak et al., 2018). One of the main advancements on the topic is provided by Pascanu et al. (2013) and Montufar et al. (2014) with the comparison

[27] Additionally, Hanin and Sellke (2017) - based on Theorem 2.1 in Wang (2004) - prove that a ReLu DNN with a given structure is equivalent to the difference between two max functions, and thus is a universal approximator.

between deep and shallow networks in terms of linear regions approximated. More specifically, by showing that a ReLu deep neural network is within the class of PWL functions, we are able to state that the approximation power of the neural network is determined by the number of linear regions that the function embedded in the network structure can approximate.

3.2.3 Linear Regions Approximation

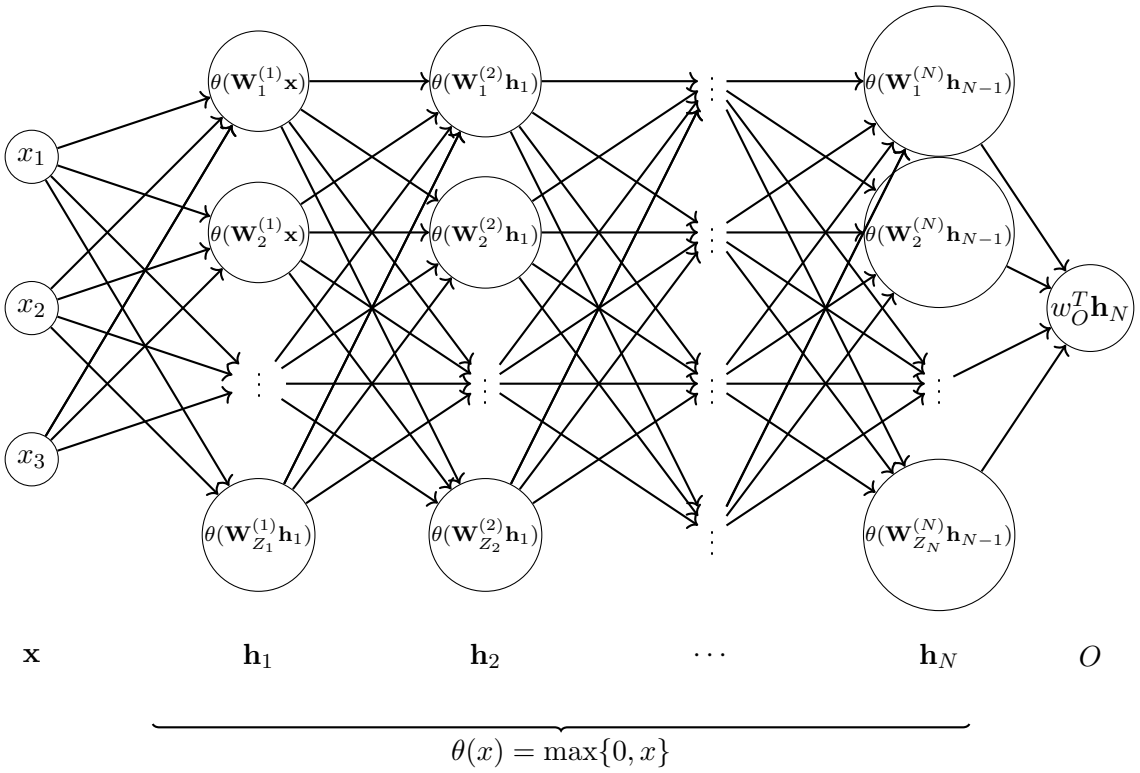


Figure 3.3: ReLu Deep Neural Network with bias terms 0.

As mentioned earlier, see also Eckle and Schmidt-Hieber (2019), the output of a ReLu deep neural network is always a PWL function of the input. To show this, we note that a ReLu deep neural network is the composition of continuous PWL functions, see expression (3.3), and thus, is another PWL function. Based on the results of Arora et al. (2016), it is also possible to conclude that the number of linear regions describing a ReLu neural network depends directly on the number of hidden nodes Z used to train the network.

Being PWL functions, ReLu deep neural networks can similarly be described by the number of linear regions that they can approximate. This concept is clarified by Farrell et al. (2019) when comparing neural networks to more classic nonparametric techniques. These authors also draw a parallelism between different nonparametric techniques to approximate unknown continuous functions and DNN structures, noting that smoothing splines are usually defined by the spline basis (smoothing parameter) and the number of knots (tuning parameters). In kernel regression, the shape of the kernel constitutes the smoothing parameter while

the bandwidth defines the tuning parameter. Similarly, in neural networks the type of connections (graph structure) and activation function identify the smoothing parameters, while the width and depth of the neural network are the tuning parameters of the sized-Z architecture under consideration.

Figure 3.3 shows that the output of a ReLu deep neural network is a weighted sum of piecewise linear functions, with the number of pieces in $\mathbb{R}^{h_{n-1}}$ equal to Z_N. One could also notice - as mentioned also in Montufar et al. (2014) - that the computation performed by $\mathbf{h}_N$ on the post activations of $\mathbf{h}_{N-1}$ is effectively carried out on the post activation regions of $\mathbf{h}_{N-2}$, and on the regions of the preceding hidden layers until the input layer is reached. Therefore, it is possible to express the output of the ReLu neural network as a weighted sum of piecewise linear functions on the input space $\mathbf{x}$.

Having defined the relation that subsists between PWL and deep neural networks, we can apply the function approximation in Aumann (1963). This author approximates any bounded real function $f(\mathbf{x})$ as a linear combination of characteristic functions $\chi_{\mathcal{S}}$ on a set $\mathcal{S}$, defined as

$$\chi_{\mathcal{S}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{S} \\ 0 & \text{if } x \notin \mathcal{S}, \end{cases} \tag{3.11}$$

such that, given the subsets $\mathcal{S}_1, \cdots, \mathcal{S}_p$ of $\mathcal{S}$:

$$f(\mathbf{x}) \sim \sum_{i=1}^{p} a_i \chi_{\mathcal{S}_i}(\mathbf{x}). \tag{3.12}$$

More formally, Aumann (1963) shows that the approximation (3.12) minimizes the sup norm:

$$\sup_{x \in \mathcal{S}} |f(x) - \sum_{i=1}^{p} a_i \chi_{\mathcal{S}_i}(\mathbf{x})|, \tag{3.13}$$

by a proper choice of a_i. Expression (3.12) can be seen as the sum of p different ReLu activation functions $\chi_{\mathcal{S}_i}$, $i = 1, \ldots, p$. The weights $\mathbf{W}^{(n)}$ of the neural network for $n = 1, \cdots, N$ correspond to the slopes of the linear regions and ω_i^O to the coefficients a_i in expression (3.12).

Montufar et al. (2014), Pascanu et al. (2013), and Raghu et al. (2017) state that the number of linear regions approximated by a ReLu network defines the complexity of the unknown target function that can be approximated by the neural network, identifying the model's flexibility. Pascanu et al. (2013) state that - in practice - the linear regions approximated by the neural network are correlated (see also Figure 3.3). This is an important characteristic that ensures good generalization performance of the neural network, and not a mere overfitting problem arising from an increasing number of linear regions. Tewlgarsky (2016) after identifying the complexity of a given function with the number of oscillations, proves that - due to the function composition defined in Definition 3.2.1 - deeper architectures will capture a

higher number of oscillations in the function being approximated than shallow networks[28]. Based on the argument that low-oscillation functions cannot approximate arbitrarily well high-oscillation functions, Tewlgarsky (2016) proves that increasing the architecture's depth increases the architecture's flexibility.

3.2.4 Number of Linear Regions

Definition 3.2.1 shows that each hidden unit has two operational nodes, one takes a value of zero and the other takes a positive value. The boundary between these two operational regions is given by the hyperplane H_j consisting of all the inputs $\mathbf{x} \in \mathbb{R}^d$ with $\mathbf{W}_j^{(n)}\mathbf{x} + b_j^{(n)}$. Therefore, the number of linear regions represented by a shallow ReLu neural network of size Z_1 is defined by the number of regions defined by the set of hyperplanes $\{H_j\}_{j\in[Z_1]}$. Zaslavsky (1975) proves that an arrangement of n hyperplanes can divide an $\mathbb{R}^d$ dimensional space in a number of regions equal to $\sum_{s=0}^{d}\binom{n}{s}$.

Being each hidden node in a shallow network a hyperplane, we can consider the hidden layer in shallow ReLu networks as an arrangement of hyperplanes, and thus by extending the result of Zaslavsky (1975) it is possible to obtain the number of linear regions approximated by a shallow ReLu Network (Pascanu et al., 2013) as

$$\sum_{s=0}^{d}\binom{Z_1}{s}. \tag{3.14}$$

Montufar et al. (2014) extend the result of Pascanu et al. (2013) and obtain the following lower bound to the maximal number of linear regions represented by a ReLu DNN (with N layers) of size $Z = \sum_{j=1}^{N} Z_j$:

$$\left(\prod_{j=1}^{N-1}\left\lfloor\frac{Z_j}{d}\right\rfloor^d\right)\sum_{s=0}^{d}\binom{Z_N}{s} \tag{3.15}$$

where $\lfloor\cdot\rfloor$ defines the "floor operator" which is the function that returns the greatest integer less than or equal to its real input argument. Based on the definition of ReLu DNN (Definition 3.2.1), Montufar et al. (2014) state that each hidden layer in a deep feedforward neural network folds the space of the previous hidden layer, with the recursive folding of the input space being determined by $\mathbf{W}$ and $\mathbf{b}$. This space folding implies that the final function computed by the last hidden layer (arrangement of Z_N hyperplanes) is applied to all the subsets identified by the succession of foldings performed by the deep structure (see also Figure 3.3). The authors, for each hidden layer, divide the set of ReLu activation functions in d non-overlapping subsets of cardinality $\lfloor Z_j/d\rfloor$.

Corollary 6 of Montufar et al. (2014) compares the expressive power of single layer ReLu NNs with ReLu DNNs of same size Z and conclude that if $d = O(1)$, the number of linear

[28] Lemma D.1 and D.2 in Arora et al. (2016) provide a similar result.

regions approximated by the latter DNN behaves as $\Omega\left(N^{-Nd}d^{-(N-1)d}Z^{Nd}\right)$[29]. In contrast, the number of linear regions approximated by its shallow NN counterpart behaves as $O\left(Z^d\right)$. Therefore, the number of regions grows exponentially in depth (N) and polynomially in width ($\frac{Z}{N}$) in ReLu DNNs, which is much faster than the polynomial growth of shallow ReLu NNs of same size Z. Thus, an increase in depth of a neural network should always lead to an exponential increase in the number of linear regions approximated by a ReLu DNN, which according to the above universal approximation theorems, results in a better fit of the DNN architecture to the input data. For completeness, we also note that an upper bound for the maximal number of linear regions of a function approximated by a network architecture with rectified linear units of size Z is of order $O\left(\left\lfloor\frac{Z}{N}\right\rfloor^{Zd}\right)$, as recently shown by Raghu et al. (2017, Theorem 1).

The above results also suggest that a deep neural network is able to represent the same number of linear regions of a shallow network with a lower number of trainable parameters (hidden units). However, empirical studies show that this is not always the case; in some empirical applications shallow networks outperform deep networks (Pasupa and Sunhem, 2016; Kim and Gofman, 2018). This is illustrated numerically in the following examples. Let us consider a ReLu DNN given by $d = 4$ and $Z = 30$. The number of linear regions approximated by a shallow network is $\sum_{s=0}^{4}\binom{30}{s} = 31931$. In contrast, if we consider a ReLu DNN with $N = 2$, we can either have $\lfloor\frac{7}{4}\rfloor^4\sum_{s=0}^{4}\binom{23}{s} = 10903$ for $Z_1 = 7$ and $Z_2 = 23$ or $\lfloor\frac{8}{4}\rfloor^4\sum_{s=0}^{4}\binom{22}{s} = 145744$, for $Z_1 = 8$ and $Z_2 = 22$. Similarly, if we increase the depth of the ReLu DNNs to $N = 3$, we can either have $\left(\lfloor\frac{8}{4}\rfloor^4\lfloor\frac{8}{4}\rfloor^4\right)\sum_{s=0}^{4}\binom{14}{s} = 376576$ for $Z_1 = 8$, $Z_2 = 8$ and $Z_3 = 14$ or $\left(\lfloor\frac{7}{4}\rfloor^4\lfloor\frac{7}{4}\rfloor^4\right)\sum_{s=0}^{4}\binom{16}{s} = 2517$, for $Z_1 = 7$, $Z_2 = 7$ and $Z_3 = 16$. Finally, if we increase the depth to $N = 4$, we can either have $\left(\lfloor\frac{8}{4}\rfloor^4\lfloor\frac{8}{4}\rfloor^4\lfloor\frac{8}{4}\rfloor^4\right)\sum_{s=0}^{4}\binom{6}{s} = 233472$ for $Z_1 = 8$, $Z_2 = 8$, $Z_3 = 8$ and $Z_4 = 6$ or $\left(\lfloor\frac{7}{4}\rfloor^4\lfloor\frac{7}{4}\rfloor^4\lfloor\frac{7}{4}\rfloor^4\right)\sum_{s=0}^{4}\binom{9}{s} = 256$, for $Z_1 = 7$, $Z_2 = 7$, $Z_3 = 7$ and $Z_4 = 9$. These examples show that under certain conditions a shallow network could provide a better approximation of the unknown function $f(\mathbf{x})$. We can formalize the results in these examples as follows.

Property 1 (Increase in Depth).
Let $Z = z$ denote the total number of hidden nodes in the ReLu DNN. Condition $(Z_1/d) < 2$ implies that

$$\left\lfloor\frac{Z_1}{d}\right\rfloor^d\sum_{s=0}^{d}\binom{z-Z_1}{s} < \sum_{s=0}^{d}\binom{z}{s}. \tag{3.16}$$

Then, the number of linear regions approximated by the ReLu DNN is smaller than the number of linear regions approximated by the shallow network counterpart.

This property can be extended to consider the condition $(Z_j/d) < 2$ for all $j = 1, \cdots, N-1$. More formally,

[29] Two real-valued functions $f(h)$ and $g(h)$ satisfy that $f(h) = \Omega(g(h))$ if there is a positive constant c such that $f(h) \geq cg(h)$, for all h sufficiently large.

Property 2 (Shallow versus Deep ReLu network).
Given a ReLu DNN comprised by $Z = z$ hidden nodes, depth N and such that $(Z_j/d) < 2$ for $j = 1, \cdots, N-1$, it holds that

$$\left(\prod_{j=1}^{N-1} \left\lfloor \frac{Z_j}{d} \right\rfloor^d\right) \sum_{s=0}^{d} \binom{z - \sum_{j=1}^{N-1} Z_j}{s} < \sum_{s=0}^{d} \binom{z}{s}. \tag{3.17}$$

Under the conditions of Property 2 we cannot guarantee that the architecture of the DNN improves over the architecture of the corresponding shallow network.

In the following section we derive an optimal architecture of the DNN based on expression (3.15). This optimal architecture is obtained before bringing the model to the data. In a second stage, we show how the optimal DNN outperforms cross-validation methods under mean square error (MSE) evaluation criteria implemented to simulated data (Section 3.4) and a real data example in Section 3.5.

3.3 Optimal Deep Neural Network Structure

This section presents the objective function that characterizes an optimal DNN architecture and discusses analytical and numerical methods for solving the optimization exercise under complex architectures of the network.

3.3.1 Maximization of the Number of Linear Regions

Our strategy for the identification of the optimal structure of a neural network is to maximize the lower bound of the maximal number of linear regions approximated by the ReLu DNN. The maximization of this quantity implies an optimal number of terms to approximate the unknown function $f(\mathbf{x})$. Under some constraints, the solution to (3.18) also yields a greater number of linear regions than the shallow network counterpart structure.

The objective function is (3.15) and the optimization problem is to choose the optimal number of hidden layers N and nodes per layer $(Z_1, \ldots, Z_N)$, for a given size z. More formally,

$$\max_{N, \{Z_l\}_{l=1}^{N}} \quad \left(\prod_{j=1}^{N-1} \left\lfloor \frac{Z_j}{d} \right\rfloor^d\right) \sum_{s=0}^{d} \binom{Z_N}{s} \tag{3.18}$$

$$\text{s.t.} \quad Z_1 + Z_2 + \cdots + Z_N = z$$
$$Z_j \geq d \quad \text{for } j = 1, 2, \cdots, N$$

The number of nodes cannot be smaller than d since the difference enters the denominator of the binomial coefficient in equation (3.15). Note that this is a standard assumption in the literature (Montufar et al., 2014; Pascanu et al., 2013; Mei et al., 2018; and Hornik, 1991),

and that - as stated by Aggarwal (2018) - the "*probabilistic regularization*" characterizing the stochastic gradient descent learning algorithm ensures a proper training of the hidden layers. The optimization procedure is complex and, in most cases, difficult to obtain analytically. The above objective function can be expressed as

$$\max_{(N,\{Z_l\}_{l=1}^{N-1},\{\mu_l\}_{l=1}^{N})} LB(N,\{Z_l\}_{l=1}^{N-1};d)+\sum\nolimits_{l=1}^{N-1}\mu_l(d-Z_l)+\mu_N(-N)$$

where $\{\mu_l\}_{l=1}^{N}\in\mathbb{R}^N$ denotes the collection of N Lagrange multipliers associated with the $N-1$ constraints, $Z_l\geq d, l=1,\ldots,N-1$, and with the constraint $N>0$, because we have incorporated the equality constraint $Z=\sum_{l=1}^{N}Z_l$ into the lower bound objective function, $LB(N,\{Z_l\}_{l=1}^{N-1};d)\equiv\left(\prod_{l=1}^{N-1}\left\lfloor\frac{Z_l}{d}\right\rfloor^d\right)\sum_{r=0}^{d}\binom{Z-\sum_{l=1}^{N-1}Z_l}{r}$. This is a combinatorial optimization problem because the decision variables $\{N,Z_1,\ldots,Z_{N-1}\}$ are integer values. Judd (1990) shows that optimizing a NN is an NP-hard problem, meaning that a polynomial time algorithm that solves it is not known but could be found[30].

One could instead try to approximate the combinatorial optimization problem by its continuous counterpart, and obtain the FOCs:

$$\begin{gathered}\partial_{Z_{l'}}LB(N^*,\{Z_l^*\}_{l=1}^{N^*-1};d)-\mu_{l'}=0,\ \text{for } l'=1,\ldots,N-1,\\ \partial_{NL}LB(N^*,\{Z_l^*\}_{l=1}^{N^*-1};d)-\mu_N=0,\\ \mu_{l'}(d-Z_{l'}^*)=0,\ \text{for } l'=1,\ldots,N-1\\ -\mu_N N^*=0.\end{gathered}\tag{3.19}$$

The FOCs of this problem represent a system of $2N$ equations in $2N$ unknowns, $(N^*,\{Z_l^*\}_{l=1}^{N^*-1},\{\mu_l^*\}_{l=1}^{N^*})\in\mathbb{R}^{2N}$. Note, however, that the number of equations and the number of unknowns is part of the solution of the problem, N^*, which is 'problematic'. Typically, this problem can be solved in two stages: a first step that - given $Z=z$ and N - identifies the optimal layerwise widths of the network $(Z_1^*,\ldots,Z_N^*)$; and a second step that given the optimal widths for different depths, identifies the optimal depth of the network N^* such that the optimal network architecture is given by the vector $(Z_1^*,\ldots,Z_{N^*}^*)$. In this case the above Lagrangian function simplifies but it is still a complex exercise.

We illustrate the latter optimization procedure with a simple exercise with $d=2$ and $Z_1+Z_2=Z$, so that $N=2>0$ and the corresponding constraint is omitted for simplicity ($\mu_3=0$). If Z_1 is an even number the optimization problem in the first step is

$$\mathcal{L}(Z_1,Z_2;\mu)=\left(\frac{Z_1}{2}\right)^2\left[1+Z_2+\frac{Z_2(Z_2-1)}{2}\right]-\mu[Z_1+Z_2-Z].$$

[30] Algorithms that require an 'intractable' exponential amount of time to find a solution are called NP.

The first order conditions of this problem with respect to Z_1, Z_2 are

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Z_1} &= \frac{Z_1}{2} \sum_{s=0}^{2} \binom{Z_2}{s} - \mu = 0 \\
\frac{\partial \mathcal{L}}{\partial Z_2} &= \left(\frac{Z_1}{2}\right)^2 \left(\frac{1}{2} + Z_2\right) - \mu = 0 \\
\frac{\partial \mathcal{L}}{\partial \mu} &= -Z_1 - Z_2 + Z = 0.
\end{aligned}$$

Solving the above system, we obtain that:

$$\begin{gathered}
\frac{Z_1}{2}\left[1 + Z_2 + \frac{Z_2(Z_2-1)}{2}\right] = \left(\frac{Z_1}{2}\right)^2 \left(\frac{1}{2} + Z_2\right) \\
Z_1 + Z_2 = Z
\end{gathered}$$

from which:

$$\left(1 + \frac{Z_2}{2} + \frac{Z_2^2}{2}\right) = \left(\frac{Z - Z_2}{2}\right)\left(\frac{1}{2} + Z_2\right). \tag{3.20}$$

Similarly, if Z_1 is an odd number the optimization problem is

$$\mathcal{L}(Z_1, Z_2; \mu) = \left(\frac{Z_1 - 1}{2}\right)^2 \left[1 + Z_2 + \frac{Z_2(Z_2-1)}{2}\right] - \mu[Z_1 + Z_2 - Z].$$

The FOCs of this problem with respect to Z_1, Z_2 are

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Z_1} &= \frac{Z_1 - 1}{2} \sum_{s=0}^{2} \binom{Z_2}{s} - \mu = 0 \\
\frac{\partial \mathcal{L}}{\partial Z_2} &= \left(\frac{Z_1 - 1}{2}\right)^2 \left(\frac{1}{2} + Z_2\right) - \mu = 0 \\
\frac{\partial \mathcal{L}}{\partial \mu} &= -Z_1 - Z_2 + Z = 0,
\end{aligned}$$

and solving the above system, we obtain that:

$$\begin{gathered}
\frac{Z_1 - 1}{2}\left[1 + Z_2 + \frac{Z_2(Z_2-1)}{2}\right] = \left(\frac{Z_1 - 1}{2}\right)^2 \left(\frac{1}{2} + Z_2\right) \\
Z_1 + Z_2 = Z
\end{gathered}$$

from which:

$$\left(1 + \frac{Z_2}{2} + \frac{Z_2^2}{2}\right) = \left(\frac{Z - Z_2 - 1}{2}\right)\left(\frac{1}{2} + Z_2\right). \tag{3.21}$$

To obtain a numerical solution, let z take a specific value for the number of total nodes Z, e.g. $z = 60$. Then, the first order conditions (3.20) and (3.21) accept the following two solutions:

$$\begin{cases}
Z_2^* = (117 + \sqrt{14585})/8 = 29.72 \approx 30 \\
Z_2^* = (117 - \sqrt{14585})/8 = -0.47 \approx 0
\end{cases} \tag{3.22}$$

In this case the only feasible solution is $Z_2^* = 30$ that entails $Z_1^* = 30$. The computation of the Hessian matrix

$$\mathbf{H}^B = \begin{bmatrix} 0 & g_1 & g_2 \\ g_1 & \mathcal{L}_{11} & \mathcal{L}_{12} \\ g_2 & \mathcal{L}_{21} & \mathcal{L}_{22}, \end{bmatrix} \tag{3.23}$$

shows that the solution $Z_1^* = Z_2^* = 30$ yields a maximum. Note that g_i indicates the derivatives of the constrains with respect to the choice variables (Z_i), and $\mathcal{L}_{ij}$ captures the second and cross-partial derivatives of the Lagrange function. Therefore, for a depth of 2 and $z = 60$, the optimal architecture of the ReLu neural network is given by two layers of equal length. To complete the solution, in the second step, we would need to repeat the maximization exercise for $N = 3$ and beyond up to $N \leq \lceil \frac{z}{d} \rceil$. The optimal architecture of the ReLu neural network is defined by the quantities $(N^*, Z_1^*, \ldots, Z_{N^*}^*)$.

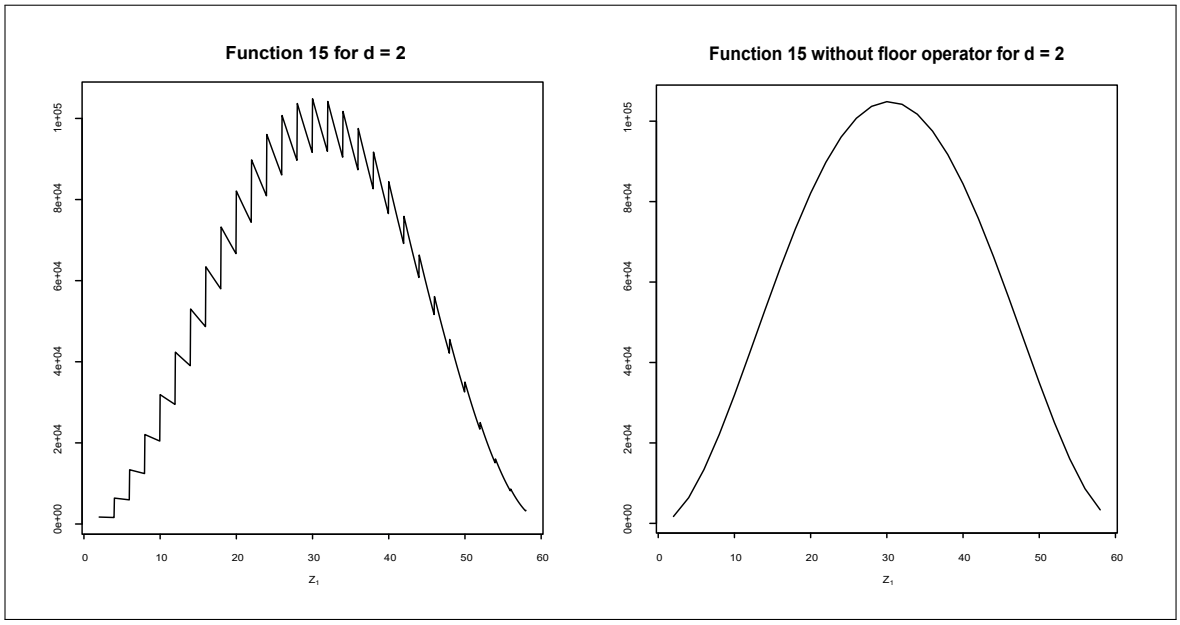


Figure 3.4: The Figure reports the lower bound to the maximal number of linear regions for the ReLu DNN considered in the above maximization problem.

The left panel of Figure 3.4 plots the function $\left\lfloor \frac{Z_1}{2} \right\rfloor^2 \left[1 + \frac{z - Z_1}{2} + \frac{(z-Z_1)^2}{2}\right]$ assuming that Z_1 is a real number defined on the set $Z_1 \in [2, z-2]$, for $z = 60$. The plot shows the saw type function implied by the floor operator. Despite this, the existence of a global maximum is clear and is found at $Z_1^* = 30$ as shown in our analytical derivation. For completeness, we also plot in the right panel the smooth version of function (3.15) that does not consider the floor operator.

The above example confirms the presence of an interior solution to the optimization problem (3.18) for low-dimensional DNNs. In practice, the difficulty of the optimization for realistic values of the number of nodes and layers implies that the solution to (3.18) is achieved through numerical methods.

3.3.2 Numerical Optimization

The multivariate optimization imposes domains on the optimal variables: each hidden layer must be defined by a number of hidden nodes which is at least equal to the input dimension, and the maximum number of hidden nodes allowed per hidden layer must guarantee that the remaining hidden layers have a number of hidden units at least equal to the input dimension.

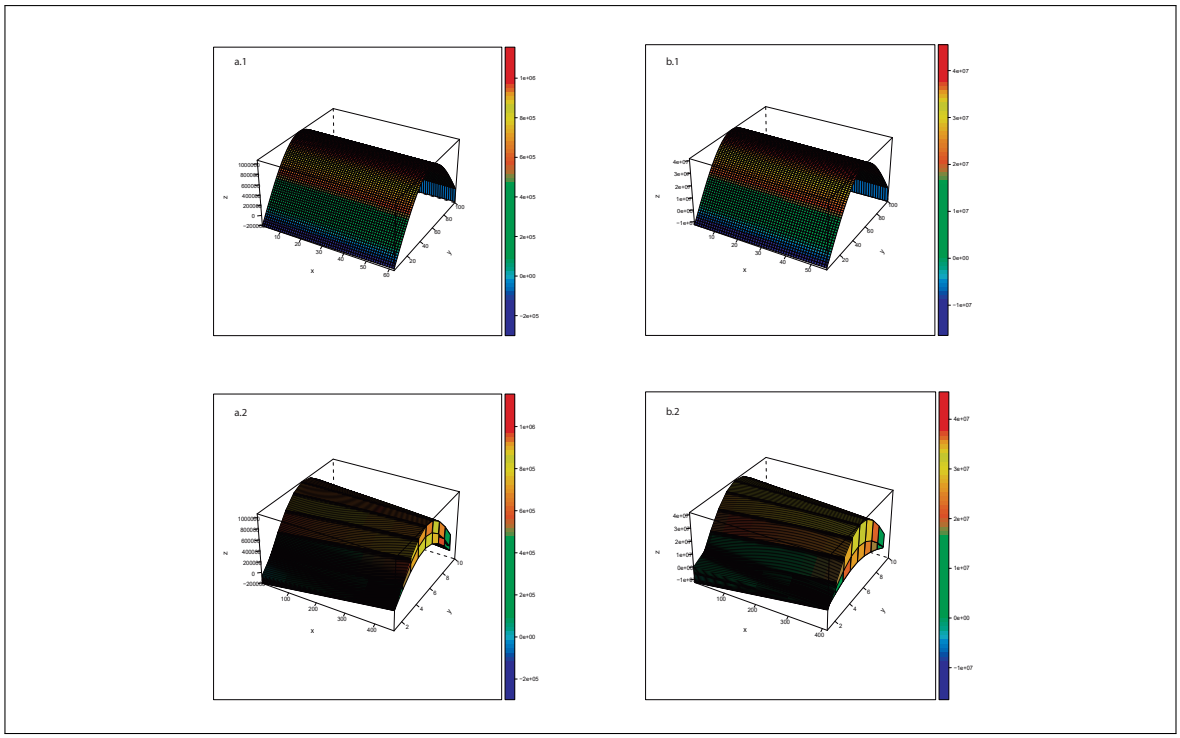


Figure 3.5: The Figure reports the number of linear regions as a function of the depth and the different combinations obtainable with the given number of hidden nodes. Sub-Figures $a.1$ and $b.1$ report the plane project of the number of linear regions as a function of the possible structures associated to $d = 2$ and $d = 3$ and three hidden layers. Sub-Figures $a.2$ and $b.2$ report the number of linear regions as a function of the depth of the neural network and the possible structures for a given number of hidden nodes.

Under these optimization constraints, a simple procedure to obtain the solution to (3.18) for a given number of nodes z is to evaluate the objective function for all possible combinations of integer numbers that satisfy the constraints $Z_1 + Z_2 + \cdots + Z_N = Z$ and $Z_j \geq d$ for $j = 1, \ldots, N$. This is the low-dimensional version of the nonlinear integer programming method discussed above. This procedure yields exact solutions to the optimization problem (3.18) for low values of N and Z. However, the problem becomes computationally intractable as N and, in particular, Z increase. In this scenario quasi newton algorithms designed to solve constrained optimization problems are more suitable. In particular, the L-BFGS-B algorithm, which is the limited-memory quasi newton algorithm designed to solve constrained optimization problems, is used to return the parameters that optimize the objective function (3.18). The obtained solutions are also compared against the SLSQP - Sequential Least Squares Programming algorithm - optimizer implemented on Pytorch. Both algorithms - in contrast to others, such as the Stochastic Gradient Descent - allow for the floor operator

to be taken care of. The comparison between the L-BFGS-B and the SLSQP algorithm is conducted solely with the intention to control for the correct convergence of the former, as they both provide satisfactory solutions.

The numerical optimization exercise is illustrated for a total number of hidden nodes equal to $Z = 60$, and different number of input variables with $d = 2, 3, 4, 5, 6$. We report the optimal widths for different depths; the optimized structure will be identified by the combination of depth and width that maximizes the minimum number of linear regions approximated by the ReLu DNN.

Figure 3.5 investigates this eventuality. In particular, Figures $a.1$ and $b.1$ provide a three-dimensional representation of a two dimensional problem, that is of the optimization of the layer wise width of the network, for a given depth and number of hidden nodes. The plots are obtained by fitting a generalized additive model between the number of linear regions associated to each structure combination obtainable when $d = 2, 3$ and $N = 3$, and the possible combinations. The fitted curve is then replicated along the x-axis in order to generate a plane for a better visualization of the convex problem solved by the optimization. The different figures confirm that the maximization problem has an interior solution and, thus, an optimal combination of widths that, for a given depth, maximizes the number of linear regions approximated by the neural network. In contrast, Figures $a.2$ and $b.2$, project the *complete* optimization problem. The optimization problem is considered not only in terms of the optimal width of the network, but also in terms of optimal depth. The y-axis represents the number of possible combinations associated to each hidden layer, while the x-axis identifies the different depths of the neural network considered, and the z-axis represents the number of linear regions associated to each combination for a given neural network depth.[31]. Figure 3.5 shows that the analyzed optimization problem (expression (3.18)) is convex if it is considered either for a given depth, or if it is considered in terms of optimal width and depth. Therefore, there exists a maximum for the lower bound of the maximal number of linear regions in terms of both width and depth and, importantly, the strategy of decomposing the optimization problem (3.18) in the two step procedure previously suggested is able to capture the global optimum.

Table 3.1 reports the results from the optimization exercise (3.18) for different input dimensions. For completeness, we report the maximization exercise from the first stage that allows us to obtain an optimal width for given depths, and then identifying the optimal combination of width and depth that maximizes the number of linear regions approximated by a neural network.

We also consider the minimization of (3.15). It is important to specify that the purpose of the numerical minimization is merely explanatory, and reported to provide a more complete

[31] Once the plane is obtained, to smooth the 3D surface, a generalized additive model is fitted. It is important to highlight that this procedure is reported only for completeness on the methodology used to construct Figure 3.5 and as such, does not change the outcome of the optimization problem

understanding of the duality between deep and shallow networks. As mentioned above, if the minimum number of linear regions approximated by a ReLu DNN is always higher than the number of linear regions approximated by the shallow network, it would be possible to conclude that - regardless the structure adopted - a deep network always outperforms the shallow network counterpart. Table 3.1 reports the results from the numerical minimization of the number of linear regions approximated by a ReLu DNN. The results reported in Table 3.1 show that the minimum number of linear regions approximated by a deep neural network (for different depths) is always lower than the number of regions approximated by the shallow counterpart (with the same number of hidden nodes). This exercise shows that not every DNN outperforms a shallow network and motivates the need of an optimization procedure to maximize the lower bound (3.15). Summing up, to be certain that the DNN provides a better approximation than the shallow network one needs to maximize the lower bound in expression (3.18) and show that the solution is larger than the number of linear regions of the shallow network counterpart.

Table 3.1: Maximum and Minimum number of linear regions for ReLu DNN. In red are highlighted the cases where a shallow network outperforms a deep one. In blue are highlighted the cases in which an increase in depth leads to a decrease in the number of linear regions approximated by the ReLu DNN. *Optimal Structure 1* is obtained by maximizing the objective function, *Optimal Structure 2* by minimizing it.

	2 Hidden Layers	3 Hidden Layers	4 Hidden Layers	5 Hidden Layers	6 Hidden Layers	7 Hidden Layers	Shallow Network N. Regions
Input Dimension: 2							
Optimal Structure 1	(30, 30)	(20, 20, 20)	(16, 16, 14, 14)	(12, 12, 12, 12, 12)	(10, 10, 10, 10, 10, 10)	(10, 10, 8, 8, 8, 8, 8)	
Optimal Structure 2	(3, 57)	(3, 3, 54)	(3, 3, 3, 51)	(3, 3, 3, 3, 48)	(3, 3, 3, 3, 3, 45)	(3, 3, 3, 3, 3, 3, 42)	1831
Minimum Regions	1654	1486	1327	1177	1036	904	
Maximum Regions	104850	2110000	21274624	132689664	546875000	1515520000	
Input Dimension: 3							
Optimal Structure 1	(30, 30)	(21, 21, 18)	(15, 15, 15, 15)	(12, 12, 12, 12, 12)	(12, 12, 9, 9, 9, 9)	(9, 9, 9, 9, 9, 9, 6)	
Optimal Structure 2	(5, 55)	(5, 5, 50)	(5, 5, 5, 45)	(5, 5, 5, 5, 40)	(5, 5, 5, 5, 5, 35)	(5, 5, 5, 5, 5, 5, 30)	36051
Minimum Regions	27776	20876	15226	10701	7176	4526	
Maximum Regions	4526000	116237212	1125000000	5016387584	10480803840	16271660538	
Input Dimension: 4							
Optimal Structure 1	(28, 32)	(20, 20, 20)	(16, 12, 16, 16)	(12, 12, 12, 12, 12)	(12, 12, 12, 8, 8, 8)	(12, 8, 8, 8 , 8, 8, 8)	
Optimal Structure 2	(7, 53)	(7, 7, 46)	(7, 7, 7, 39)	(7, 7, 7, 7, 32)	(7, 7, 7, 7, 25)	(7, 7, 7, 7, 7,7, 18)	523686
Minimum Regions	317683	179447	92171	41449	15276	4048	
Maximum Regions	99519049	2420312500	13361283072	34179096474	22175970048	13844348928	
Input Dimension: 5							
Optimal Structure 1	(30, 30)	(20, 20, 20)	(15, 15, 15, 15)	(15, 10, 10, 10, 15)	(10, 10, 10, 10, 10, 10)	(10, 10, 10, 10, 10, 5, 5)	
Optimal Structure 2	(9, 51)	(9, 9, 42)	(9, 9, 9, 33)	(9, 9, 9, 9, 24)	(9, 9, 9, 9 ,9, 15)	(9, 9, 9, 9, 9, 9, 6)	5985198
Minimum Regions	2621112	974982	284274	55455	4944	63	
Maximum Regions	1356422112	22754099200	70940996208	39367213056	21407727616	1073741824	
Input Dimension: 6							
Optimal Structure 1	(30, 30)	(18, 18, 24)	(18, 12, 12, 18)	(12, 12, 12, 12, 12)	(12, 12, 12, 12, 6, 6)	(12, 12, 12, 6, 6, 6, 6)	
Optimal Structure 2	(11, 49)	(11, 11, 38)	(11, 11, 11, 27)	(11, 11, 11, 11, 16)	(11, 11, 11, 11, 10, 6)	(11, 11, 11, 9, 6, 6, 6)	56049058
Minimum Regions	16122226	3345616	397594	14893	64	64	
Maximum Regions	12003312500	101000893491	93102981120	42110812160	1073741824	16777216	

Table 3.1 also shows that when the structure that minimizes the number of linear regions is considered, an increase in depth leads to a decrease in the number of linear regions approximated. Figure 3.6 shows that the number of linear regions decreases linearly when $\mathbf{x} \in \mathbb{R}^2$. As the dimension of the input layer increases - or the ratio between input dimension and number of hidden nodes per layer decreases - the observed decrease in the number of linear regions approximated by the ReLu DNN becomes gradually exponential.

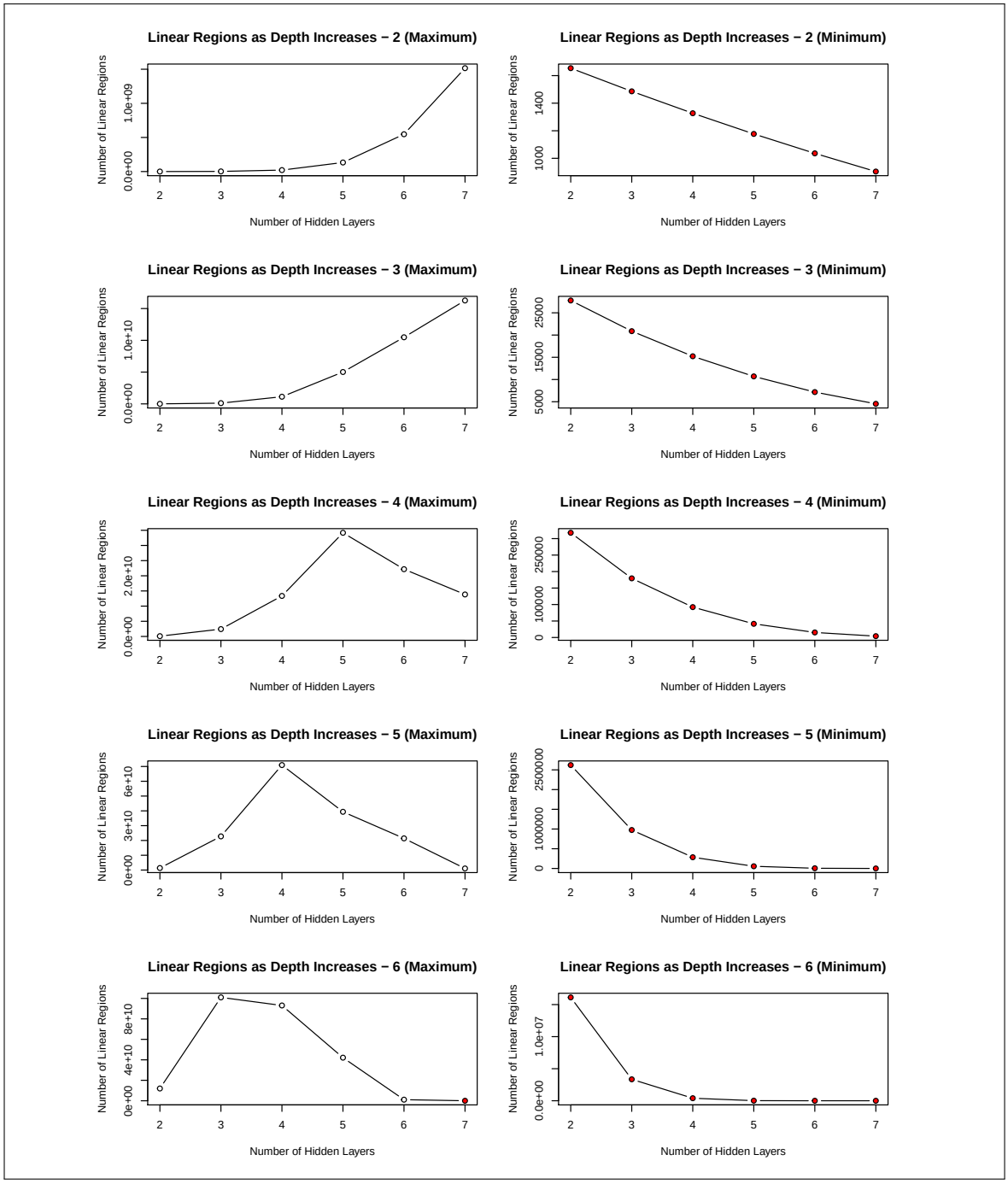


Figure 3.6: The Figure reports the relation between the number of linear regions and the depth of the neural network for different input dimensions. The color red indicated an approximated number of linear regions lower than the shallow counterpart.

This behavior can be justified by the fact that a higher input dimension will require a higher number of maximum nodes for the minimum representation. The increase of hidden nodes per layer will lead to a decrease in the number of linear regions which is exponential in the ratio between input dimension and width of the hidden layers. To summarize, the minimization exercise shows that a ReLu DNN with sub-optimal structure underperforms a shallow network with the same number of hidden nodes for low input dimension. This

result provides an explanation of the reasons behind the controversial empirical evidence regarding performance of deep and shallow architectures, and it further justifies the proposed methodology: the maximization algorithm ensures not only to obtain - for a given number of hidden nodes - the optimal width and depth of a neural network, but also to obtain a neural network structure that outperforms the shallow counterpart, ensuring better generalization (Bengio, 2009; Ciresan et al., 2012; and Goodfellow et al., 2013a) and lower computational power needed.

For input dimensions equal to 2 and 3, an increase in the depth of the ReLu DNN (see Figure 3.6) results in an exponential increase in the number of linear regions approximated. However, when $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$, an increase in depth leads to an exponential increase in the number of linear regions up to a certain cutoff point; after that, the increase in depth leads to a decrease in the number of linear regions (Property 2 is violated). The cut-off points identify the depth for which Property 2 is violated. When the minimum ratio is $\lfloor Z_j/d \rfloor = 2$, deeper structures underperform shallower ones. Ultimately, the observed decrease in the number of linear regions leads - in the case of $x \in \mathbb{R}^6$ and $N = 7$ - to the underperformance of the ReLu DNN when compared to the shallow counterpart. To summarize, recalling the fact that in the analyzed case an increase in depth implies a reduction in width, one could conclude that, when Property 2 is satisfied, depth is always better than width.

Table 3.1 shows the existence - for different input dimensions - of an optimal architecture in terms of both depth and width of the network. The optimization exercises for different number of layers show that increasing the depth, for a given number of hidden nodes, not always leads to an increase in the number of affine regions approximated by the neural network. This aspect ensures the existence of an optimal neural network depth, with a given optimal layer width. The maximum depth considered in this numerical exercise allows identifying the optimal depth - with corresponding optimal layer widths - for $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$.

The next step involves comparing the out-of-sample performance of our proposed optimal architecture against a benchmark neural network obtained from a k-fold cross-validation procedure.

3.4 Monte Carlo Simulation

This section assesses statistically differences in out-of-sample performance between competing neural network architectures. In order to do so, different data generating processes are simulated, and the relevant test statistic for the comparison of the out-of-sample performance is constructed.

The present algorithm optimizes the width and depth of a neural network of a given size $Z = z$. To ensure the robustness of our results, we repeat the optimization over different candidates $z = 40, 60, 90$. We also vary the input dimensions and consider five different DGPs, one linear and four nonlinear. When the linear DGP is considered, the out-of-sample

performance of the optimal structure is compared against a linear model (OLS). The rationale for this exercise is to assess the predictive power of deep neural networks in an unfavorable context in which OLS methods are proved to be optimal. When nonlinear generating processes are considered, the out-of-sample performance of the structure selected with the above optimal methodology is compared with the out-of-sample performance of a structure selected with k-fold cross-validation[32].

3.4.1 Data Generating Process

We consider the following linear and nonlinear DGPs:
Linear Process - Model 1:

$$y = a + \mathbf{ax} + \epsilon, \tag{3.24}$$

with different input dimensions: $\mathbf{x} \in \mathbb{R}^4$, $\mathbf{x} \in \mathbb{R}^5$, and $\mathbf{x} \in \mathbb{R}^6 \sim \mathcal{N}(\mu, 1)$. The parameters chosen for the vector of coefficients $\mathbf{a}$ are generated from a $U(-10, 10)$ and then rounded to the closest digit[33]. Similarly, the parameter for the vector of means μ are generated from $U(-5, 5)$ and then rounded to the closest digit. When $\mathbf{x} \in \mathbb{R}^4$, $\mathbf{a} = [-8, 2, 2, 2]^\top$ and $\mu = [-4, 1, 1, 1]$; When $\mathbf{x} \in \mathbb{R}^5$, $\mathbf{a} = [-8, 2, 2, 2, 7]^\top$ and $\mu = [-4, 1, 1, 1, 5]$, when $\mathbf{x} \in \mathbb{R}^6$, $\mathbf{a} = [-8, 2, 2, 2, 7, 3]^\top$ and $\mu = [-4, 1, 1, 1, 5, 1]$. The error term is $\epsilon \sim \mathcal{N}(0, 1)$ that is uncorrelated to the input variables.

In the nonlinear case we consider four different DGPs.

Model 2:

$$y = a_1 x_1 + (5e^{-6})(1 - e^{a_2 x_2 + a_3 x_3}) + a_4 x_4^2 + \epsilon. \tag{3.25}$$

Model 3:

$$y = a_1 x_1^2 + (1e^{-6})\frac{a_2 x_2}{a_3 x_3} + (1e^{-6})(a_4 x_4)(a_5 x_5) + \epsilon. \tag{3.26}$$

Model 4:

$$y = a_1 x_1^2 + (1e^{-6})\frac{a_2 x_2}{a_3 x_3} + a_4^3 x_4 + (1e^{-6})(a_5 x_5)(a_6 x_6) + \epsilon. \tag{3.27}$$

Model 5:

$$y = a + \mathbf{ax} * \tau + \epsilon, \tag{3.28}$$

where

$$\tau = 4x_1 - 0.5x_2.$$

The nonlinear DGPs expressed by expressions (3.25), (3.26), and (3.27) incorporate the nonlinearity of the process predominantly in terms of extreme variations. Models 2 to 4 are

[32] The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.
[33] Before the generation of the simulated parameter the seed was set to 1234.

multiplied by scaling factors of the order (1e-6) to reduce the contribution of the input variables; therefore, by re-scaling the marginal effects, the process y is comprised only by those observations that are "extreme" whereas those marginal effects of lower magnitude are re-scaled such that y is approximately zero in those cases. Model 5 is the nonlinear counterpart of Model 1, that considers interactions between the different input variables.

3.4.2 Accuracy Test

The main objective of this subsection is to compare the out-of-sample performance of the proposed novel methodology against a structure selected via k fold-cross validation. By performing k-fold cross-validation, the structure that returns the lowest out-of-sample mean square error (MSE) will be selected. The procedure is as follows. We simulate 1500 observations from the different DGPs; 1200 observations are used for training the model and 300 observations are used as validation set for the final comparison. A 3-folds cross-validation, over a randomized grid search[34], is performed over the 1200 observations to select the structure that returns the lowest out-of-sample MSE. Thus, being both neural networks fitted using the same number of training observations and on the same dataset, the comparison carried out for the nonlinear DGP processes analyses two comparable neural networks.

Table 3.1 shows that, for a given input dimension, it is possible to identify the optimal structure in terms of width and depth of the network. In particular, we identify the optimal structure when $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$. Prior to performing k-fold cross-validation, we consider two neural networks with the same depth, obtained from the results in Table 3.1, such that the comparison is done in terms of optimal number of nodes per layer (optimal width) for a given depth of the network. As a result, for a given input dimension and for a given optimal depth, the two structures - one obtained from the maximization proposed in the chapter, and the other obtained from k-fold cross-validation - will be compared in terms of MSE on the validation set (300 observations)[35].

The objective of the current simulation settings is to compare two different models using a pre-specified loss function, and see if the differences are statistically significant. In our case, we want to test if the novel methodology returns an accuracy (B) higher than the one returned by a model selected via k-fold cross-validation (A). The hypothesis can be written as:

$$\mathcal{H}_0 \quad : \quad E[L_A(\hat{y}, y) - L_B(\hat{y}, y)] \leq 0 \tag{3.29}$$

$$\mathcal{H}_1 \quad : \quad E[L_A(\hat{y}, y) - L_B(\hat{y}, y)] > 0, \tag{3.30}$$

with $L(\hat{y}, y)$ denoting the loss function associated with each of our two models, A and B. For the case of the MSE, we have $L(\hat{y}, y) = (\hat{y} - y)^2$, and A and B denote two different

[34] A grid search with all the possible combinations of the structure would be infeasible.

[35] Empirical Evidence shows that hyper-parameters selected via cross-validation not always return a good out-of-sample accuracy with unseen data (Kohn et al., 1991; Rao et al., 2008).

models[36] for predicting the response variable y. Under the alternative hypothesis, Model B exhibits a lower MSE than Model A. The empirical counterpart of the above expectation is $\overline{MSE}_m \equiv \hat{E}[L(\hat{y}_m, y_m)] = \frac{1}{F}\sum_{i=1}^{F}(\hat{y}_{mf} - y_{mf})^2$, with $m = A, B$ and F the out-of-sample evaluation period. A feasible test statistic is of the form

$$T_F = \sqrt{F}\frac{\overline{MSE}_A - \overline{MSE}_B}{\sqrt{\hat{V}(\overline{MSE}_A - \overline{MSE}_B)}}. \tag{3.31}$$

Under the null hypothesis, this statistic converges to a standard normal distribution such that rejection of the null hypothesis can be assessed from a one-sided test with critical value $z_{1-\alpha}$, with α the significance level of the test. Furthermore, if the observations y_f, $f = 1, \ldots, F$ are *i.i.d.*, the relevant variance of the statistic is

$$V(\overline{MSE}_A - \overline{MSE}_B) = (\sqrt{F})^2\frac{1}{F}\left[V(e_{Af}^2) + V(e_{Bf}^2) - 2Cov(e_{Af}^2, e_{Bf}^2)\right], \tag{3.32}$$

with e_{Af} and e_{Bf} the residuals of each model, constructed as $e_{mf} = y_f - \widehat{y}_{mf}$, with $m = A, B$. Then, a suitable estimator of the asymptotic variance is

$$\hat{V}(\overline{MSE}_A - \overline{MSE}_B) = \frac{1}{F}\sum_{f=1}^{F}(e_{Af}^2 - \overline{e_A^2})^2 + \frac{1}{F}\sum_{f=1}^{F}(e_{Bf}^2 - \overline{e_B^2})^2 - 2\frac{1}{F}\sum_{f=1}^{F}(e_{Af}^2 - \overline{e_A^2})(e_{Bf}^2 - \overline{e_B^2}), \tag{3.33}$$

with $\overline{e_m^2} = \frac{1}{F}\sum_{f=1}^{F} e_{mf}^2$, with $m = A, B$. If there is serial dependence then the variance of the test statistic is more complex and we need to incorporate the presence of serial correlation. Robust estimators are HAC estimators developed by Newey-West (1987). Note that the difference between the in-sample and out-of-sample exercise is how we construct the residuals e_f.

3.4.3 Simulation Results

Table 3.2 reports the results from the Monte Carlo simulation. When the linear data generating process is considered, the out-of-sample accuracy of the neural network with structure obtained via the proposed maximization (to which we will refer as optimal ReLu DNN) is compared against a linear regression and thus, a model that correctly identifies the DGP. In this case when considering the null hypothesis (3.29), the out-of-sample MSE of the linear model will estimate the loss function L_B; failing to reject the null hypothesis indicates that there is no statistically significant difference between the out-of-sample MSE of the optimal ReLu DNN and the corresponding MSE of the linear model (which by definition is BLUE). Hence, both models are indistinguishable in terms of predictive accuracy in mean. From Table 3.2, one could notice that the null hypothesis is rejected at 0.1 significance level for $\mathbf{x} \in \mathbb{R}^4$

[36] k-fold cross-validation and optimization methodology.

and $z = 90$ (possibly, due to the over-complicated architecture of the neural network) and for $\mathbf{x} \in \mathbb{R}^5$ and $z = 40$. In all other cases, the difference in accuracy is not significant.

Table 3.2: The Table reports, for each input dimension and each data generating process considered, the out-of-sample MSE of the models considered, the test statistic and p value for the difference in accuracy. *** indicates 0.01 significance level, ** 0.05, and * indicated 0.1 significance level.

	Linear	**NN_{optim}**	**NN_{cv}**	**Test Stat**	**P-value**
Linear Process - Model 1					
$\mathbf{x} \in \mathbb{R}^4$					
$\mathbf{z = 40}$	1.0207	1.0240	-	0.0896	0.4644
$\mathbf{z = 60}$	1.0207	1.0816	-	1.2180	0.1116
$\mathbf{z = 90}$	1.0207	1.0818	-	1.5138*	0.0650
$\mathbf{x} \in \mathbb{R}^5$					
$\mathbf{z = 40}$	1.0600	1.1517	-	1.4927*	0.0677
$\mathbf{z = 60}$	1.0600	1.0911	-	1.0955	0.1366
$\mathbf{z = 90}$	1.0600	1.0852	-	0.7071	0.2397
$\mathbf{x} \in \mathbb{R}^6$					
$\mathbf{z = 40}$	1.0739	1.0426	-	-1.2314	0.8909
$\mathbf{z = 60}$	1.0739	1.0890	-	0.3827	0.3510
$\mathbf{z = 90}$	1.0739	1.0744	-	0.0103	0.4959
Nonlinear Process - Model 2					
$\mathbf{x} \in \mathbb{R}^4$					
$\mathbf{z = 40}$	90.3837	1.1749	1.2883	1.4923*	0.0678
$\mathbf{z = 60}$	90.3837	1.1502	1.2981	1.4918*	0.0679
$\mathbf{z = 90}$	90.3837	1.1337	1.2685	1.4154*	0.0785
Nonlinear Process - Model 3					
$\mathbf{x} \in \mathbb{R}^5$					
$\mathbf{z = 40}$	5513.2835	1.5221	1.7633	1.3613*	0.0867
$\mathbf{z = 60}$	5513.2835	1.4396	1.7002	1.9154**	0.0277
$\mathbf{z = 90}$	5513.2835	1.2374	1.5599	2.0329**	0.0210
Nonlinear Process - Model 4					
$\mathbf{x} \in \mathbb{R}^6$					
$\mathbf{z = 40}$	4101.4600	1.7025	2.0295	1.7821**	0.0374
$\mathbf{z = 60}$	4101.4600	1.6699	2.3017	0.8602	0.1948
$\mathbf{z = 90}$	4101.4600	1.5777	1.9049	1.3362*	0.0907
Nonlinear Process - Model 5					
$\mathbf{x} \in \mathbb{R}^4$					
$\mathbf{z = 40}$	9517.0577	5.5102	6.3055	1.9203**	0.0274
$\mathbf{z = 60}$	9517.0577	4.2384	9.3619	1.5554*	0.0599
$\mathbf{z = 90}$	9517.0577	2.6755	9.1836	1.3791*	0.0839
$\mathbf{x} \in \mathbb{R}^5$					
$\mathbf{z = 40}$	24985.8370	10.0566	15.2530	1.5874*	0.0562
$\mathbf{z = 60}$	24985.8370	5.9994	9.2081	1.0345	0.1504
$\mathbf{z = 90}$	24985.8370	7.2509	13.8439	1.6874**	0.0457
$\mathbf{x} \in \mathbb{R}^6$					
$\mathbf{z = 40}$	54202.9318	53.0290	89.1285	1.1435	0.1264
$\mathbf{z = 60}$	54202.9318	7.9023	13.5481	2.5353***	0.0056
$\mathbf{z = 90}$	54202.9318	14.3952	55.7897	3.9699***	<0.0001

Table 3.3 reports the structures returned from the proposed methodology and from the

3-fold cross-validation. The out-of-sample MSE of the linear model is still reported for completeness, but as the comparison with the performance of neural networks is no longer meaningful, the test of the null hypothesis (3.29) will not be extended also to the linear model. When the null hypothesis (3.29) is tested, the out-of-sample MSE of the optimal model will be L_B; thus, rejecting the null hypothesis (3.29) will provide evidence of the outperformance of the optimal neural network over a cross-validated one. With the exception of three cases, the null hypothesis is rejected in all cases at 0.1 significance level. When $\mathbf{x} \in \mathbb{R}^6$ and $z = 40$ both models are not able to approximate adequately the fourth nonlinear data generating process; conversely, when $z = 60, 90$ the out-of-sample error of the two models decreases drastically and the null hypothesis is rejected at 0.01 significance level, suggesting the outperformance of our DNN architecture.

Overall, these results suggest that the optimal neural network architecture proposed in this chapter is comparable in terms of predictive accuracy to BLUE estimators in the most unfavorable case given by a linear DGP. When nonlinear DGPs are considered, the null hypothesis (3.29) is used to compare the out-of-sample performance of the optimal neural network against a cross-validated structure.

Table 3.3: The Table reports, for each input dimension and each data generating process considered, the neural network structures selected using the proposed methodology and the 3-folds cross-validation with a randomized grid search approach.

	Optimized Neural Network	**Cross Validated - Nonlinear 1**	**Cross Validated - Nonlinear 2**
$\mathbf{z = 40}$			
$\mathbf{x} \in \mathbb{R}^4$	[12, 12, 8, 8]	[24, 5, 4, 7]	[22, 4, 4, 10]
$\mathbf{x} \in \mathbb{R}^5$	[15, 10, 15]	[24, 8 , 8]	[27, 7, 6]
$\mathbf{x} \in \mathbb{R}^6$	[22, 18]	[16, 10, 14]	[24, 9, 7]
$\mathbf{z = 60}$			
$\mathbf{x} \in \mathbb{R}^4$	[12, 12, 12, 12, 12]	[32, 5, 4, 4, 15]	[43, 4, 5, 4, 4]
$\mathbf{x} \in \mathbb{R}^5$	[15, 15, 15, 15]	[22, 7, 5, 26]	[35, 5, 5, 15]
$\mathbf{x} \in \mathbb{R}^6$	[18, 18, 24]	[40, 6, 14]	[46, 7, 7]
$\mathbf{z = 90}$			
$\mathbf{x} \in \mathbb{R}^4$	[18, 12, 12, 12, 12, 12, 12]	[50, 15, 9, 4, 4, 4, 4]	[62, 6, 6, 4, 4, 4, 4]
$\mathbf{x} \in \mathbb{R}^5$	[15, 15, 15, 15, 15, 15]	[56, 10, 9, 5, 5, 5]	[60, 9, 6, 5, 5, 5]
$\mathbf{x} \in \mathbb{R}^6$	[18, 18, 18, 18, 18]	[38, 7, 6, 6, 33]	[43, 6, 6, 6, 29]

To summarize, the results reported in Table 3.2 show that a neural network with the structure selected via the proposed maximization outperforms a neural network whose structure is obtained via a 3-fold cross-validation with a randomized grid search. This result is true for all the different input dimensions and DGPs considered. Similarly, it is shown that the optimal neural network has an out-of-sample performance statistically equivalent to the MSE of a linear model, when the true data generating process is linear. These results are robust to the different number of hidden nodes considered, showing that the outperformance does not depend on the specific number of hidden nodes chosen.

The MSEs reported in Table 3.2 show also another important result: the existence of an optimal number of hidden nodes. Testing the null hypothesis (3.29) across different values of z ensures the robustness of the results to different choices of the initial number of hidden nodes. Our findings show that - given a specific DGP - there exists an optimal number of hidden nodes that minimizes the MSE. For example, when a linear DGP is considered $z = 40$ returns the lowest MSE, while when the nonlinear DGPs are considered, the MSE is minimized for $z = 60$ or $z = 90$ (when the true DGP is linear, a high number of hidden nodes may result in overparametrization).

3.5 Empirical Application

The above DNN prediction techniques are applied to real data on house prices. For comparability purposes, we choose a popular dataset widely used in the literature on linear and nonlinear prediction, see for example Al Batanieh and Kaur (2018). The Boston Housing dataset initially studied in Harrison and Rubinfeld (1978) consists of 506 datapoints split between 404 training and 102 test observations. This dataset is concerned with housing values in the suburbs of Boston. The dependent variable is median values of Boston house prices in thousands of dollars, that is explained by thirteen independent variables, which for simplicity reasons are not reported, and described in details by Harrison and Rubinfeld (1978).

In order to guarantee a proper training of the ReLu DNN, a feature-wise normalization consisting on transforming the observations into zero-mean and unit standard deviation random variables is performed. The mean and the standard deviation used for the feature-wise normalization are computed considering only the train dataset.

The first step of the neural network prediction exercise is to set the optimal neural network architecture. Different optimization algorithms, weights initializers, learning rates, number of epochs, drop-out rates, and total number of hidden nodes are considered. In particular, the learning rates 0.0001, 0.001, 0.01, and 0.1 for the Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$), for the Stochastic Gradient descent (SGD) with Nesterov momentum of 0.9, and the RMSProp optimizer with $\rho = 0.9$ are tuned. When the Adam optimizer is considered, we use the He normal initializer that draws samples from a truncated normal distribution with $\mu = 0$ and $\sigma = \sqrt{2/\text{Indim}}$, where "Indim" is the number of input units in the weight tensor; conversely, when the SGD is tuned, a truncated normal distribution with $\mu = [0.5, 0.1]$ and $\sigma = [0.02, 0.01]$ is considered. The number of epochs analyzed are: 500, 1000, 2000 and 5000. Hinton et al. (2012) show that dropout can be used to effectively reduce the generalization error of large neural networks fitted on a limited amount of data. Therefore, given the low number of observations in the training set, in order to improve the out-of-sample performance, the dropout training is adopted. Following Srivastava et al. (2014) and given the relatively low dimension of the fitted feedforward neural network, different dropout rates $p = 0.1, 0.2, 0.3$ are tuned for all hidden layers, and $p = 0.1$ for the input layer.

The proposed optimization procedure lets the total number of hidden nodes as a free

parameter. In this application, we consider $Z = z = 50, 100, 130, 150$, and 200. The number of input variables is $d = 13$, and we allow for a maximum depth of 10. The results of our optimization exercise in (3.18) are as follows. For $z = 50$, the optimal structure of the DNN is $[37, 13]$; for $z = 100$ is $[48, 26, 26]$; for $z = 130$ is $[52, 39, 39]$; for $z = 150$ is $[33, 39, 39, 39]$ and, finally, for $z = 200$ is $[44, 39, 39, 39, 39]$. To check the convergence of the optimization algorithm - for each z - the maximization is conducted using $z + / - 1$.

Each combination is evaluated using 4-folds cross-validation on the training set, and finally evaluated on the validation set. Thus, the predictive performance of our method is measured by averaging the four MAEs and MSEs obtained from the training samples under the 4-folds cross-validation, and the validation MSE and MAE obtained from fitting the tuned model on the validation set. It is important to remember that our proposed DNN architecture focuses on optimizing the structure of the network but is silent about the specific choice of the aforementioned hyperparameters. Therefore, the 4-folds cross-validation is used to choose the optimal combination of learning rate, optimizer, weight initializer, number of epoch, and dropout rate. The optimal combination of nodes and hyperparameters is evaluated on the validation set[37].

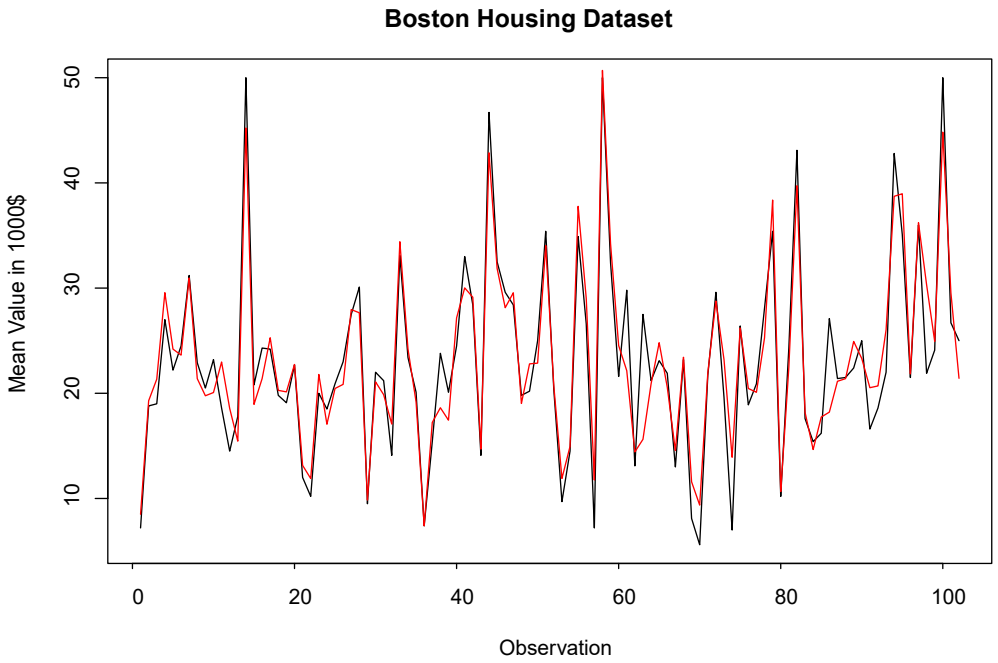


Figure 3.7: The Figure reports the observed average prices (in black) against the fitted values (in red) for the validation set.

Based on the out-of-sample accuracy, the best combination is defined by the RMSProp optimizer with learning rate 0.001, 2000 epochs, $z = 130$, a dropout rate of 0.1 across all hidden layers, and no dropout in the input layer. The cross-validated MSE and MAE are 7.67 and 2.02 respectively, and the validation MSE and MAE are 8.76 and 2.17. Figure 3.7 reports the fitted values out-of-sample of the trained ReLu DNN against the observed values.

[37] This cross-validation exercise to optimize the tuning parameters is different from the 4-folds cross-validation method used in the Monte Carlo section as benchmark model to obtain the out-of-sample MSE.

An out-of-sample MAE of 2.17 implies that the model will predict house prices with an error - on average - of 2,170$.

The empirical results show that the proposed methodology can be used to improve the predictive performance of neural networks. For example, Al Bataineh and Kaur (2018) - considering three algorithms for neural network training - find a test MSE of 13.96 for the Levenberg-Marquardt, of 12.77 for the Bayesian Regularization, and of 16.63 for the Scaled Conjugate Gradient; Granitto et al. (2001) after proposing an algorithm for the construction of ensemble neural networks, that ensures a good balance between diversity and accuracy, find a test MSE of 14.46 ± 6.89; Myshkov and Julier (2016) explore the posterior distribution obtained from Bayesian inference methods for neural networks and find a test RMSE for the Stochastic Gradient Langevin Dynamics (Welling and Teh, 2011) of 3.99 (MSE of 15.92); Papadopoulos and Haralambous (2011) propose a new methodology to extend regression neural networks by producing not only point predictions but also prediction intervals, and the test RMSE associated with the point prediction is 4.06 (MSE of 16.48); finally, Bakker and Heskes (2003) propose an algorithm for neural network ensemble (GASEN) that uses genetic algorithm to select an optimal subset of neural network for the construction of the ensemble learner, with MSE of 12.26. The differences between the MSE obtained using the novel methodology and the ones observed in the literature are statistically significant.

By taking the relative differences with the lowest and highest MSEs reported by the aforementioned literature, our optimised NN architecture represents an improvement over the average prediction in extant studies between 28.55% and 47.32%.

3.6 The CART procedure and future implementations

As previously mentioned in chapter 1, a useful approach to frame the objective of identifying the optimal neural network structure is to compare the proposed methodology with model selection in tree-based architectures. The recursive binary partition algorithm used to train the tree is designed to identify the optimal number of non-overlapping regions that ensures optimality in predicting the target variable. An increase in the depth of the tree leads to an increase in the number of non-overlapping regions being identified, leading to higher approximation power that –depending on the underlying data generating process– can be translated in either lower bias or higher variance.

Therefore, based on the previous sections, one could understand how the depth and the width of the neural networks play a similar role to the depth of regression/classification trees when recursively folding the input space. Depending on the function to be learned, an increase in the number of linear regions being approximated leads to higher approximation power, leading to either a reduction in bias or an increase in variance. Nonetheless, based on the results and the discussion in Pascanu et al. (2013), the risk of overfitting (high variance) increases at a slower rate when increasing the depth/width of neural networks as opposed to increasing the depth of regression trees.

The comparison with tree-based structures allows not only framing the identification problem in neural networks in a clearer way, but it also suggests future steps that can be undertaken in order to identify an optimal neural network structure. In particular, following the existing literature (e.g., Bertsimas and Dunn, 2017), it is possible to define the objective of the CART procedure as a formal optimization problem:

$$\begin{aligned} \min \quad & L(\mathbb{T}) + \alpha|\mathbb{T}| \\ \text{s.t.} \quad & N(\ell) \geq N_{min} \qquad \forall\ \ell \in \text{leaves}(\mathbb{T}) \end{aligned} \tag{3.34}$$

with $L(\mathbb{T})$ being the in-sample loss of the tree $\mathbb{T}$, $|\mathbb{T}|$ is the number of branch nodes in tree $\mathbb{T}$, α is the complexity parameter (to be tuned) that balances additional complexity of the tree $\mathbb{T}$ against the increase in predictive accuracy, and $N(\ell)$ is the number of training points contained in each leaf node ℓ. Solving Equation (3.34) allows to frame the so-called *optimal tree problem*.

Based on the present chapter and the proposed optimization, it would be possible to frame the *optimal neural network* problem in a similar way:

$$\begin{aligned} \min \quad & L(G) + \alpha|G| \\ \text{s.t.} \quad & Z_j \geq d \quad \text{for } j = 1, 2, \cdots, N \end{aligned} \tag{3.35}$$

with $L(G)$ being the in-sample loss of the neural network G, $|G|$ is the maximized lower bound to the number of linear regions being approximated by the neural network (which ensures an efficient allocation of hidden nodes across hidden layers), and α is the complexity parameter which, also based on the simulation results, should be defined as a function of the dimensions (number of hidden nodes) of the neural network such that $\alpha = g(\cdot, z)$.

Starting from these premises, future research will focus on the implementation of Equation (3.35) for the identification of an optimal neural network structure which relies –most importantly– on the correct identification of $\alpha = g(\cdot, z)$.

3.7 Conclusions

It is standard practice in the machine learning community of researchers and practitioners to engage into time and computational power consuming "fine tuning" of the neural network architecture while training. The width and depth of the architecture is a subset of the hyperparameters to be fine tuned. This chapter proposes an optimization method to obtain suitable values of these quantities. We do this by maximizing the lower bound on the maximum number of linear regions that a deep neural network can approximate characterized by Montufar et al (2014), granting maximum flexibility of deep architectures of given size. The optimization is done numerically using state-of-the-art methods such as L-BFGS-B and SLSQP algorithms.

The performance of the proposed optimal architecture for deep neural networks is assessed in an exhaustive Monte-Carlo exercise and also empirically. This novel procedure is shown to outperform k-fold cross-validation procedures for prediction in nolinear models. In linear settings, in which standard OLS methods are optimal, our approach is competitive and provides comparable mean square error values. We illustrate the ability of our optimal deep neural network architecture to predict median house prices from the Boston Housing dataset. This dataset is extensively used by the machine learning literature to validate new learning techniques. Our neural network architecture reduces the out-of-sample mean square prediction error between 28.55% and 47.32% compared to recent studies fitting neural networks to the Boston Housing dataset. By optimizing width and depth prior to training for a given choice of nodes, our proposed method substantially saves upon the necessary time and computing power involved in fine tuning while training. Although we have not explored it in detail in the chapter, we expect that those savings will increase more than proportionately with bigger and more complex datasets that call upon bigger sized architectures.

CHAPTER 4

Prediction intervals for deep neural networks

Chapter Abstract

The aim of this chapter is to propose a suitable method for constructing prediction intervals for the output of neural network models. To do this, we adapt the extremely randomized trees method originally developed for random forests to construct ensembles of neural networks. The extra-randomness introduced in the ensemble reduces the variance of the predictions and yields gains in out-of-sample accuracy. An extensive Monte Carlo simulation exercise shows the good performance of this novel method for constructing prediction intervals in terms of coverage probability and mean square prediction error. This approach is superior to state-of-the-art methods extant in the literature such as the widely used MC dropout and bootstrap procedures. The out-of-sample accuracy of the novel algorithm is further evaluated using experimental settings already adopted in the literature.

4.1 Introduction

Neural networks are widely used in prediction tasks due to their unrivaled performance and flexibility in modeling complex unknown functions of the data. Although these methods provide accurate predictions, the development of tools to estimate the uncertainty around their predictions is still in its infancy. As explained in Hüllermeier and Waegeman (2020) and Pearce et al. (2018), out-of-sample pointwise accuracy is not enough[38]. The predictions of deep neural network (DNN) models need to be supported by measures of uncertainty in order to provide satisfactory answers for prediction in high-dimensional regression models, pattern recognition, biomedical diagnosis, and others (see Schmidhuber (2015) and LeCun et al. (2015) for overviews of the topic). Due to the centrality of the topic, a plethora of literature in machine learning has focused on the construction of algorithms to measure the uncertainty around the predictions of neural network methods.

A pioneering contribution is provided by Hwang and Ding (1997) that construct asymptotically valid prediction intervals for neural networks. Yet, being their research focused only on single layer feedforward neural networks with sigmoidal activation function, it does not find applicability in some widely adopted neural network structures (for example, convolutional neural networks, recurrent neural networks, and deep feedforward neural networks with the ReLu activation function). This early work on prediction intervals on neural networks does not incorporate recent advances in machine learning prediction. Therefore, it is of much interest to extend such procedures to construct prediction intervals by incorporating recent state-of-the-art methods that improve the generalization power in neural network models. One of the main regularization methods to improve the predictions is dropout. This technique—proposed by Srivastava et al. (2014)—ensures better generalization for neural networks by forcing the hidden nodes not to co-adapt with the neighboring nodes.

Levasseur et al. (2017) notice that one of the main obstacles for assessing uncertainty around the outputs of neural network models is the fact that the weights characterizing the predictions are usually fixed, implying that the output is deterministic. In contrast, Bayesian neural networks (Denker and LeCun, 1991)—instead of defining deterministic weights—allow the networks' weights to be defined by a given probability distribution and can capture the posterior distribution of the output, providing a probabilistic measure of uncertainty around the model predictions. Being the approximation of the posterior distribution a difficult task, the literature focusing on deep Bayesian neural networks has proposed different alternatives for the estimation of such distribution. These alternatives center around the Bayesian interpretation of dropout methods to estimate the uncertainty in the model predictions. A noteworthy example is Gal and Ghahramani (2016a); these authors develop a Monte Carlo dropout to model both parameter and data uncertainty by fitting a deep neural network with dropout implemented not only at training but also during test phase. During test time, each

[38] A trustworthy representation of uncertainty can be considered pivotal when machine learning techniques are applied to medicine (Yang et al., 2009; Lambrou et al., 2011), or to anomaly detection, optimal resource allocation and budget planning (Zhu and Laptev, 2017), or cyber-physical systems (Varshney and Alemzadeh, 2017) defined as surgical robots, self-driving cars and the smart grid.

forward pass is multiplied by a random variable to generate a random sample of the approximated posterior distribution. Levasseur et al. (2017) analyze the coverage probability of the procedure proposed by Gal and Ghahramani (2016a) and conclude that the construction of prediction intervals with correct empirical coverage probabilities is highly dependent on the adequate tuning of the dropout rate.

Applying dropout during the test phase can also be regarded as an approach to estimate the uncertainty around the predicted outputs from deep neural networks that works outside the Bayesian framework (for example, Cortes-Ciriano and Bender, 2019). However, any suitable method that aims at constructing valid prediction intervals based on the Monte Carlo dropout must also incorporate the uncertainty due to noise in the data. It is based on this final aspect that Kendall and Gal (2017), Serpell et al. (2019), and Zhu and Laptev (2017) propose novel methodologies for the correct estimation of the prediction uncertainty for both shallow and deep networks. To do so, Kendall and Gal (2017) propose a new loss function that allows estimating the aleatoric uncertainty from the input data; Serpell et al. (2019) couple the stochastic forward passes of the Monte Carlo dropout with the Mean Variance Estimation[39]; and Zhu and Laptev (2017) propose to estimate the data uncertainty with a consistent estimator in a hold out set.

Another branch of the literature has been focusing on adopting bootstrap based approaches for the estimation of the prediction intervals of neural networks (see for example, Carney et al., 1999; and Errouissi et al., 2015). Bootstrap procedures have become increasingly popular, despite their computational requirements, as they provide a reliable solution to obtain the predictive distribution of the output variable in both shallow and deep neural networks. Recent advances in the neural network literature (Pearce et al., 2018; Lee et al., 2015; and Lakshminarayanan et al., 2017) have also shown how parameter resampling without data resampling can improve over standard bootstrap approaches not only in terms of out-of-sample accuracy but also in terms of prediction uncertainty estimation.

Our contribution focuses on the latter form of resampling. In particular, this chapter focuses on estimating the uncertainty around the predictions of neural network models. Our novel approach extends the Extra-trees algorithm (Geurts et al., 2006) to ensembles of deep neural networks using a fixed Bernoulli mask. In other words, T different sub-networks with randomized architectures (each network will have different layer-specific widths) are independently trained on the same dataset. Thus, the fixed Bernoulli mask introduces an additional randomization scheme to the prediction obtained from the ensemble of neural networks that ensures independence between the components of the ensemble reducing, in turn, the variance associated to the prediction and yielding accurate prediction intervals. Additionally, based on the findings of Lee et al. (2015) and Lakshminarayanan et al. (2017), the novel procedure is expected to outperform bootstrap based approaches in terms not only of estimation accuracy but also of uncertainty estimation. When comparing classical bootstrap approaches to the

[39] The Mean Variance Estimation method - introduced by Nix and Weigend (1994) - involves fitting a neural network with two output nodes capturing the mean and the variance, respectively, of a Normal distribution.

extra-neural network we notice that: (i) both methods guarantee conditional randomness of the predicted outputs, the extra-neural network method does it through the Bernoulli random variables with probability p and random weight initialization, whereas the bootstrap does it through the nonparametric data resampling and random weight initialization; (ii) by performing data resampling, the naive (nonparametric) bootstrap approach requires the assumption that observations are independent and identically distributed (*i.i.d.*); importantly, each single model is trained with only 63% unique observations of the original sample due to resampling with replacement; (iii) by randomizing the neural network structures, the extra-neural network approach increases the diversity (see Zhou (2012) for an analysis of diversity and ensemble methods) among the individual learners; and (iv) the extra-neural network will benefit from the generalization gains associated with dropout (one can think of the dropout approach of Srivastava et al. (2014) as an ensemble of sub-networks trained for one gradient step).

To summarize, the Monte Carlo dropout approximates the predictive distribution by fitting a deep or shallow network with dropout both at train and test time. Conversely, both extra-neural network and bootstrap based approaches approximate the target predictive distribution via ensemble methods; if the independence among predictions in the latter procedure is guaranteed by both data resampling and random weights' initialization, the extra-neural network algorithm ensures independence among the predictions of the ensemble of neural networks by random weights' initialization and by randomizing the neural network structure.

To analyze the out-of-sample performance and the empirical coverages of the proposed methodologies, we carry out an extensive Monte Carlo exercise that evaluates the Monte Carlo dropout, the bootstrap approach, and extra-neural network for both deep and shallow neural networks given different dropout rates and data generating processes. The simulation results show that all three procedures return prediction intervals approximately equal to the theoretical ones for nominal values equal to 0.01 and 0.05; for prediction intervals constructed at 0.10 significance level, the extra-neural network is shown to outperform both Monte Carlo dropout and bootstrap. Additionally, the simulation findings show that the extra-neural network approach returns prediction intervals with correct empirical coverage rates for different dropout rates (within a reasonable range) as opposed to the Monte Carlo dropout that returns correct prediction intervals for specific values of the dropout rate. The findings not only show the robustness of the extra-neural network to the choice of the dropout rate, but they also complete the results of Levasseur et al. (2017) by showing that the Monte Carlo dropout returns correct prediction intervals when the dropout rate that yields the highest out-of-sample accuracy is adopted.

Finally, the novel methodology is also evaluated on real world datasets. In order to allow for comparability with other approaches found in the literature, the experimental settings of Hernández-Lobato and Adams (2015) are adopted. The empirical results show that extra-neural network methods outperform other state-of-the-art approaches used in the literature. These results complete the conclusions drawn from the Monte Carlo simulation by showing

the generalization of the extra-neural network methodology when applied to large dimensional datasets.

The rest of the chapter is organized as follows: Section 4.2 provides the definition of a DNN used for regression purposes, the concept of dropout originally introduced by Srivastava et al. (2014), and a brief note discussing random weight initialization and uncertainty for extra-neural networks. . Section 4.3 reviews extant methodologies to construct prediction intervals that can be applied to DNNs. Section 4.4 introduces a novel methodology to construct prediction intervals based on an adaptation of Extra-trees for random forests. Section 4.5 presents the simulation setup including linear and nonlinear models along with the choice of parameters and hyperparameters for the implementation of neural network methods. Section 4.6 discusses the results of the empirical study. Section 4.7 concludes.

4.2 Dropout in DNN models

We follow the original setup of Hwang and Ding (1997) and propose the following specification for predicting the output variable y_i, for $i = 1, \ldots, n$:

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \tag{4.1}$$

with $f(\mathbf{x}_i)$ a real-valued function used to predict the outcome variable using a set of covariates $\mathbf{x}_i$. The choice of the functional form $f(\mathbf{x}_i)$ depends on the loss function penalizing the difference between the outcome variable and the prediction. For example, it is well known that if the loss function is quadratic then the best predictive model is $f(\mathbf{x}_i) = E[y_i \mid \mathbf{x}_i]$. The error term ϵ defines the noise in the output variable that cannot be explained by the covariates $\mathbf{x}$ and satisfies the conditional independence assumption $E[\epsilon_i \mid \mathbf{x}_i] = 0$.

In this chapter we consider $f(\mathbf{x}_i)$ to be modeled by a ReLu deep neural network. For any two natural numbers d, $n_1 \in \mathbb{N}$, which are called input and output dimension respectively, a $\mathbb{R}^d \to \mathbb{R}^{n_1}$ ReLu DNN is given by specifying a natural number $N \in \mathbb{N}$, a sequence of N natural numbers $Z_1, Z_2, \cdots, Z_N$, and a set of $N+1$ affine transformations $\mathbf{T}_1 : \mathbb{R}^d \to \mathbb{R}^{Z_1}, \mathbf{T}_i : \mathbb{R}^{Z_{i-1}} \to \mathbb{R}^{Z_i}$, for $i = 2, \cdots, N$, and $\mathbf{T}_{N+1} : \mathbb{R}^{Z_N} \to \mathbb{R}^{n_1}$. Such a ReLu DNN is called a $(N+1)$-layer ReLu DNN, and is said to have N hidden layers. The function $f : \mathbb{R}^d \to \mathbb{R}^{n_1}$ is the output of this ReLu DNN that is constructed as

$$f(\mathbf{x}_i; \boldsymbol{\omega}) = \mathbf{T}_{N+1} \circ \boldsymbol{\theta} \circ \mathbf{T}_N \circ \cdots \circ \mathbf{T}_2 \circ \boldsymbol{\theta} \circ \mathbf{T}_1, \tag{4.2}$$

with $\mathbf{T}_n = \mathbf{W}^n \mathbf{h}_{n-1} + \mathbf{b}_n$, where, for $N = 1$, $\mathbf{W}^n \in \mathbb{R}^{Z_1 \times d}$; $\mathbf{h}_0 \equiv \mathbf{x}$, with $\mathbf{x} \in \mathbb{R}^{d \times 1}$ the input layer, and $\mathbf{b}_n \in \mathbb{R}^{Z_1}$ is an intercept or bias vector. For $N \neq 1$, $\mathbf{W}^n \in \mathbb{R}^{Z_n \times Z_{n-1}}$ is a matrix with the deterministic weights determining the transmission of information across layers; $\mathbf{h}_{n-1} \in \mathbb{R}^{Z_{n-1}}$ is a vector defined as $\mathbf{h}_{n-1} = \boldsymbol{\theta}(\mathbf{T}_{n-1})$, and $\mathbf{b}_n \in \mathbb{R}^{Z_n}$. The function $\boldsymbol{\theta}$ is a ReLu activation function defined as $\boldsymbol{\theta}(\mathbf{T}_{n-1}) = \max\{0, \mathbf{T}_{n-1}\}$ and $\boldsymbol{\omega} = \{\boldsymbol{W}^n, \mathbf{b}_n\}_{n=1}^N$ collects the set of estimable features of the model. The *depth* of a ReLu DNN is defined as $N+1$. The

width of the n^{th} hidden layer is Z_n, and the *width* of a ReLu DNN is $\max\{Z_1, \cdots, Z_N\}$. The *size* of the ReLu DNN is $Z_{\text{tot}} = Z_1 + Z_2 + \cdots + Z_N$. The number of active weights (different from zero), in a fully connected ReLu DNN, of the n^{th} hidden layer is $w_n = (Z_n \times Z_{n-1}) + Z_n$. The *number of active weights* in a fully connected ReLu DNN is $w_1 + w_2 + \cdots + w_N$. Under these premises, universal approximation theorems developed for ReLu DNN models (Lu et al., 2017) guarantee that $f(\mathbf{x}_i; \boldsymbol{\omega})$ approximates the true function $f(\mathbf{x}_i)$ in (4.1) arbitrarily well. See also Cybenko (1989), Leshno et al. (1993), Hornik (1991), Lu et al. (2017), and Mei et al. (2018) for universal approximation theorems in similar contexts.

In practice, there is an approximation error due to replacing $f(\mathbf{x}_i)$ by $f(\mathbf{x}_i; \boldsymbol{\omega})$ in model (4.1), where $f(\mathbf{x}_i; \boldsymbol{\omega})$ denotes a feasible version of the DNN model that can be estimated from the data.[40] The model that we consider in practice is

$$y_i = f(\mathbf{x}_i; \boldsymbol{\omega}) + u_i, \tag{4.3}$$

where $u_i = \varepsilon_i + f(\mathbf{x}_i) - f(\mathbf{x}_i; \boldsymbol{\omega})$. In the related literature the effect of the approximation error is usually neglected, see Pearce et al. (2018) and Heskes (1997). The different sources of error in model (4.1) are explained in Section 4.3. Before doing that, we review the concept of dropout in DNN models.

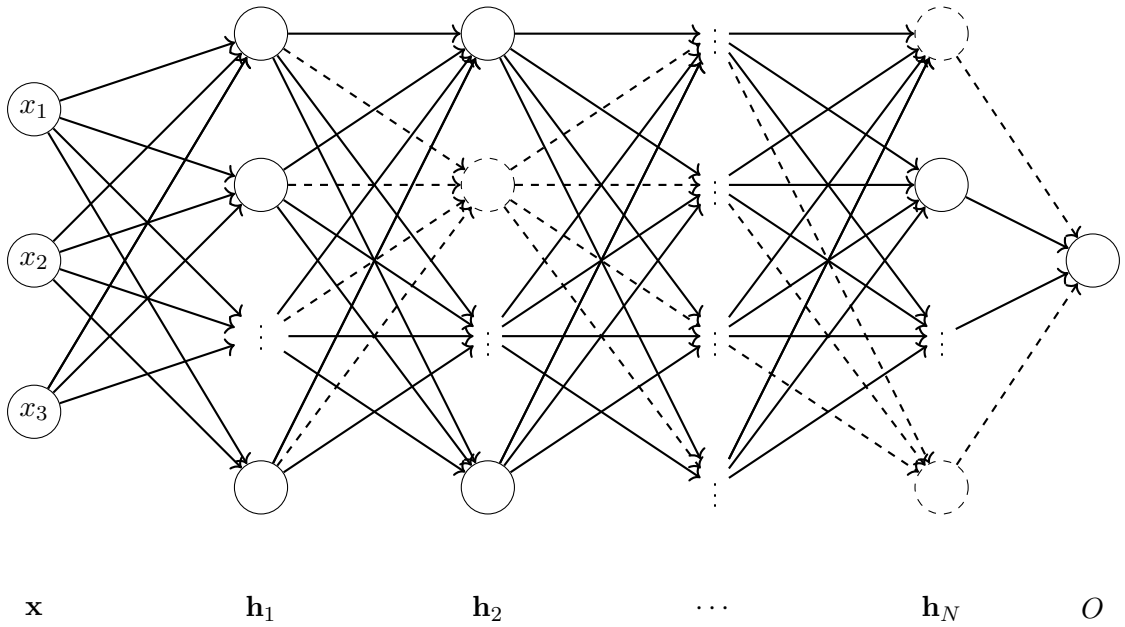


Figure 4.1: Forward pass in the backpropagation algorithm in a neural network with bias terms 0 and trained with dropout. The subset of neurons (and thus of network weights) set equal to zero is represented with dotted lines.

Training with dropout (*dropout training*) implies that for each iteration of the learning algorithm different random sub-networks (or *thinned* networks) will be trained.[41] Let h_{zn}

[40] The feasibility of the model entails that it is defined by a truncation of the true ReLu DNN model that approximates arbitrarily well the unknown function $f(\mathbf{x}_i)$.

[41] Warde-Farley et al. (2014) explain how each sub-network is usually trained for only one gradient step.

denote the elements of the vector $\mathbf{h}_n$ for a given node $z = 1, \ldots, Z_n$ and layer $n = 1, \ldots, N$. Srivastava et al. (2014) develop a dropout methodology that is applied to each function h_{zn} to obtain a transformed variable $\overline{h}_{zn}$. This variable is obtained by pre-multiplying h_{zn} by a random variable r_{zn} with distribution function $F(r_{zn})$, such that $\overline{h}_{zn} = r_{zn} \cdot h_{zn}$, for all (z, n), prior to being fed forward to the activation function of the next layer, h_{zn+1}, for all $z = 1...Z_{n+1}$. For any layer n, $\mathbf{r}_n$ is then a vector of independent random variables, $\mathbf{r}_n = [r_{1n}, ..., r_{Z_n n}] \in \mathbb{R}^{Z_n}$. In this chapter we consider only the Bernoulli probability distribution $F(r_{zn})$, where each r_{zn} has probability p of being 1 (and $q = 1 - p$ of being 0). The vector $\mathbf{r}_n$ is then sampled and multiplied element-wise with the outputs of that layer, h_{zn}, to create the thinned outputs, $\overline{h}_{zn}$, which are then used as input to the next layer, h_{zn+1}. When this process is applied at each layer $n = 1...N$, this amounts to sampling a sub-network from a larger network at each forward pass (or gradient step). At test time, the weights are scaled down as $\overline{\mathbf{W}}^n = p\mathbf{W}^n, n = 1...N$, returning a deterministic output [42] . We then identify $\mathbf{r}^\star = [\mathbf{r}_1, ..., \mathbf{r}_N]$ as the collection of independent random variables applied to a feedforward neural network of depth $N + 1$.

Figure 4.1 shows how the dropout mask works. At each training step, a random subset of neurons (identified with the dotted lines in Figure 4.1) will randomly not be considered when training the network and thus will be "dropped out" (Géron, 2019). In other words, dropout training generates –at each training step– a unique random neural network. In particular, as previously mentioned, dropout prevents a phenomenon called feature co-adaptation which will now be further discussed following Aggrawal (2018). Aggrawal (2018) explains how the hidden layers in neural networks should be able to capture important features/characteristics of the observed data without having dependencies on irrelevant features. To better understand this statement, one could consider a situation where only 50% of the weights are updated during the backpropagation algorithm, while the other 50% remain constant to their random initialization values. In these circumstances, Aggrawal (2018) argues that neural networks will still provide reasonably good results –in-sample– by adapting the trained weights to the weights that have remained to a random state and thus that transmit only noise. This circumstance (which could happen in deep neural networks where the updates of the weights do not happen at the same rate across hidden layers) would lead to sub-optimal results leading to poor out-of-sample performance (e.g., learning dependencies that do not exist). Dropout prevents this phenomenon by forcing the neural network to make predictions (and thus to update while training) using only a subset of weights (focusing on the previously example, dropout would ensure that the 50% of the weights being trained are not updated considering the half remaining to a random state).

[42] In practice, an inverted dropout methodology is applied when implementing this methodology in Keras for RStudio. In this case, instead of scaling-down the weights at test time, the weights are scaled-up during train time as $\overline{\mathbf{W}}^n = (1/p)\mathbf{W}^n, n = 1...N$. At test time, a single deterministic forward pass on the unscaled weights $\mathbf{W}^n$ is performed.

4.2.1 Random weight initialization

Shallow and deep neural network are usually trained via the gradient descent (GD) algorithm that—being an iterative algorithm—requires an initial value for the parameter to be estimated. Goodfellow et al. (2016) explain how training algorithms and thier convergence depend heavily on the choice of the initialization: different initial points can determine if the algorithm converges or not, if it converges to a global or local minimum, or the speed of convergence. Consequentially, it follows that different weight initialization will lead to different parameter ($\boldsymbol{\omega}$) estimates. More formally, consider Gaussian initialization and define $\{\mathbf{W}_0^1, \ldots, \mathbf{W}_0^N\}$ as the weights generated at the beginning of the GD algorithm; by considering $e = 1, \cdots, E$ epochs, it is possible to define the GD update rule as:

$$\mathbf{W}_e^n = \mathbf{W}_{e-1}^n - \eta \nabla_{\mathbf{W}^n} L(\mathbf{W}_{e-1}^n), \qquad n = 1, \cdots, N \tag{4.4}$$

with η being the learning rate and $\nabla_{\mathbf{W}^n} L(\mathbf{W}_{e-1}^n)$ being the partial gradient of the training loss $L(\mathbf{W}_{e-1}^n)$ with repsect to $\mathbf{W}^n$ defined as:

$$L(\mathbf{W}_{e-1}^n) = \frac{1}{M} \sum_{i=1}^{M} L(f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}); y_i), \qquad n = 1, \cdots, N \tag{4.5}$$

From Equation (4.4) and (4.5), one could notice how the estimated $\{\mathbf{W}_E^n\}_{n=1}^N$ depends on $\{\mathbf{W}_0^n\}_{n=1}^N$, η, and the optimization algorithm implemented. Therefore, random weight initialization is assumed to capture parameter uncertainty. The present analysis does not consider recent advances analyzing the relation between neural networks' dimensions (Z_{tot}) and weight initialization that ensures the presevation of the initialization properties during training. As an example, Zou et al. (2018) provide the condition under which Gaussian random initialization and (stochastic) GD produce a set of iterated estimated weights that centers around $\{\mathbf{W}_0^n\}_{n=1}^N$ with a perturbation small enough to guarantee the global convergence of the algorithm, ultimately impacting on the approximation of the epistemic uncertainty via random weight initialization.

4.3 Prediction intervals for DNN models

The prediction intervals for the output of a ReLu DNN are derived from its predictive distribution. This distribution can be approximated asymptotically using a Normal distribution; by resampling methods using bootstrap procedures; and by simulation methods using Monte Carlo dropout. In this section we review the prediction intervals obtained from these procedures. Section 4.4 complements these methods by introducing a novel approach to construct prediction intervals based on Extra-neural networks [43].

[43] The Monte Carlo dropout, the percentile bootstrap and the extra-neural network approaches obtain their predictions via model averaging; therefore, to ease comparison across the different algorithms, we will use—in the following Sections—T to indicate the number of stochastic forward passes in the Monte Carlo dropout, and the number of independently trained neural networks in both bootstrap and extra-neural

4.3.1 Asymptotic prediction intervals (Delta Method)

In practice, we estimate Model (4.3) using a training sample to obtain parameter estimates $\widehat{\boldsymbol{\omega}}$, such that the relevant empirical model is

$$y_i = f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}) + e_i, \tag{4.6}$$

with $f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}})$ a function that is estimated from the data and $\widehat{\boldsymbol{\omega}}$ the parameter estimates of the matrices of weights $\mathbf{W^n}$ and bias parameters $\mathbf{b_n}$ defining the DNN; e_i is the residual of the model. Using Expressions (4.1) to (4.6), the error term in (4.1) can be decomposed as

$$\epsilon_i = \underbrace{f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}) - f(\mathbf{x}_i; \boldsymbol{\omega})}_{\text{estimation error}} + \underbrace{f(\mathbf{x}_i; \boldsymbol{\omega}) - f(\mathbf{x}_i)}_{\text{bias effect}} + \underbrace{e_i}_{\text{aleatoric error}} \tag{4.7}$$

such that the conditional variance of the output variable given the set of covariates $\mathbf{x}$, denoted as σ^2_ϵ, satisfies that $\sigma^2_\epsilon = \sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) + \sigma^2_e$, with $\sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)$ the epistemic uncertainty due to the estimation of the model parameters and hyperparameters (estimation effect) and σ^2_e the variance due to the aleatoric error. The bias term does not have an effect on the variance of the predictor but introduces an error in the model forecast. More formally, $E[f(\mathbf{x}_i; \boldsymbol{\omega})] = f(\mathbf{x}_i) + \mu_i$, with μ_i a constant that captures the approximation error (bias) due to using a truncation of the *asymptotic* true ReLu DNN model. In this chapter we concentrate on estimating the uncertainty around the predictions, given by σ^2_ϵ, however, when possible, we will also discuss the bias effect due to the approximation of the ReLu DNN model.

The distinction between epistemic and aleatoric uncertainty is extremely relevant when DNNs are considered[44]. It has been shown that deep models, notwithstanding the high confidence in their predictions, fail on specific instances due to parameter uncertainty (see Hüllermeier and Waegeman, 2020). Additionally, deep learning models are subject to drastic changes in their performance when minor changes to the dataset are engineered (well known problem of *adversarial examples* in Papernot et al., 2017) implying variability in the parameter estimates. For this reason, the literature focusing on deep learning and uncertainty quantification propose algorithms that allow capturing all sources of uncertainty (see Zhu and Laptev, 2017; Hüllermeier and Waegeman, 2020; Senge et al., 2014; Kull and Flach, 2014; and Varshney and Alemzadeh, 2017).

In a neural network setting we estimate the predictive variance σ^2_ϵ using the test sample, of size n, such that $\widehat{\sigma}^2_\epsilon = \widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) + \widehat{\sigma}^2_e$. Under the assumption of homoscedasticity of the error

network algorithms.

[44] Even if other machine learning algorithms are characterized by high variance in their predictions (e.g., orthogonal splits characterizing decision and regression trees), the distinction between aleatoric and epistemic uncertainty has been analyzed, mainly, by the literature in deep learning. A first attempt for the computation of aleatoric and epistemic uncertainty in decision trees and random forests is made by Shaker and Hüllermeier (2020). As previously mentioned, decision trees partition the input space into several non-overlapping regions, each of which is associated with a constant predictor. Shaker and Hüllermeier (2020) –by extending the results from Bayesian neural networks– provide an estimation for both epistemic and aleatoric uncertainty associated to each region.

term over the test sample, we can estimate consistently the aleatoric uncertainty such that $\widehat{\sigma}_e^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}))^2$. However, estimating the variance due to parameter estimation is cumbersome unless the specific form of the function $f(\mathbf{x}_i;\boldsymbol{\omega})$ is known to the modeler. Under this stringent assumption, the only uncertainty in the proposed model specification is in the choice of the model parameters $\boldsymbol{\omega}$ and hyperparameters. In this case the literature proposes the delta method to approximate the estimated function $f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}})$ under a first order Taylor expansion around the true parameter vector $\boldsymbol{\omega}$. More specifically, given a data point $\mathbf{x}_i$, and assuming that the number of observations M is sufficiently large to ensure that $\widehat{\boldsymbol{\omega}}$ is a local approximation of the true parameter vector $\boldsymbol{\omega}$, Ungar et al. (1996) show that it is possible to linearize the neural network around the data point as:

$$f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}) = f(\mathbf{x}_i;\boldsymbol{\omega}) + \mathbf{f}_{\boldsymbol{\omega}i}^{\intercal}(\widehat{\boldsymbol{\omega}} - \boldsymbol{\omega}) + o_P(|\widehat{\boldsymbol{\omega}} - \boldsymbol{\omega}|), \tag{4.8}$$

with $\mathbf{f}_{\boldsymbol{\omega}i}^{\intercal}$ a vector with entries $\partial f(\mathbf{x}_i;\boldsymbol{\omega})/\partial\omega_N$, with N the number of parameter in $\boldsymbol{\omega}$, defined as (see also De vieaux et al., 1998):

$$\mathbf{f}_{\boldsymbol{\omega}i}^{\intercal} = \left[\frac{\partial f(\mathbf{x}_i;\boldsymbol{\omega})}{\partial\omega_1}, \frac{\partial f(\mathbf{x}_i;\boldsymbol{\omega})}{\partial\omega_2}, \cdots, \frac{\partial f(\mathbf{x}_i;\boldsymbol{\omega})}{\partial\omega_N}\right] = \nabla_{\boldsymbol{\omega}} f(\mathbf{x}_i;\boldsymbol{\omega}) \tag{4.9}$$

Following Seber and Wild (1989), the literature focusing on the delta method (see Hwang and Ding, 1997; Ungar et al., 1996; De vieaux et al., 1998) propose the following estimator of the asymptotic variance of $f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}})$ evaluated at the true parameter vector $\boldsymbol{\omega}$:

$$\widehat{\sigma}_{\widehat{\boldsymbol{\omega}}}^2(\mathbf{x}_i) \approx \widehat{\sigma}_e^2[\mathbf{f}_{\boldsymbol{\omega}i}^{\intercal}(\mathbf{J}_{\boldsymbol{\omega}}^{\intercal}\mathbf{J}_{\boldsymbol{\omega}})^{-1}\mathbf{f}_{\boldsymbol{\omega}i}], \tag{4.10}$$

with $\mathbf{J}_{\boldsymbol{\omega}}$ the Jacobian matrix evaluated at $\boldsymbol{\omega}$. This is defined as

$$\mathbf{J}_{\boldsymbol{\omega}} = \left[\frac{\partial f(\mathbf{x}_i;\boldsymbol{\omega})}{\partial\boldsymbol{\omega}}\right]_{\boldsymbol{\omega}}. \tag{4.11}$$

Therefore, using the delta method, the corresponding asymptotic predictive variance of y_i is estimated as $\widehat{\sigma}_{\epsilon}^2 = \widehat{\sigma}_e^2(1 + S(\widehat{\boldsymbol{\omega}}))$, with $S(\widehat{\boldsymbol{\omega}}) = \mathbf{f}_{\widehat{\boldsymbol{\omega}}i}^{\intercal}(\mathbf{J}_{\widehat{\boldsymbol{\omega}}}^{\intercal}\mathbf{J}_{\widehat{\boldsymbol{\omega}}})^{-1}\mathbf{f}_{\widehat{\boldsymbol{\omega}}i}$ and under the central limit theorem, we obtain the following asymptotic prediction interval for y_i:

$$f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}) \pm z_{1-\alpha/2}\widehat{\sigma}_e\sqrt{1 + S(\widehat{\boldsymbol{\omega}})}, \tag{4.12}$$

with $z_{1-\alpha/2}$ the relevant critical value from the standard Normal distribution at an α significance level.

Hwang and Ding (1997) showed that, regardless the not identifiability of the weights in a neural network, the prediction interval in (4.12) is asymptotically valid when the feedforward neural network is trained to convergence. Despite providing asymptotically valid prediction intervals, the delta method is not widely adopted by the literature focusing on uncertainty

quantification and deep learning due to problems associated with the computation of the Jacobian matrix. In particular, due to the high number of parameters in $\boldsymbol{\omega}$, the complex calculation of $\mathbf{J}$ is prone to error (Tibshirani, 1996); additionally, the near singularities in the model due to overfitting (Tibshirani, 1996) or due to the small sample size (De vieaux et al., 1998) make the computation of the gradient $\mathbf{J}$ unreliable or unfeasible.

Thus, the literature has been focusing on bootstrapping techniques for the construction of prediction intervals for neural networks. In fact, as also highlighted by Tibshirani (1996), bootstrapping prediction intervals provide a feasible alternative that does not suffer from the matrix inversion problem and does not depend on the existence of derivatives.

4.3.2 Bootstrap predictive distribution

An alternative approach to asymptotic prediction intervals is to construct a finite-sample approximation of the prediction interval. Bootstrap procedures provide a reliable solution to obtain predictive intervals of the output variable. We proceed to explain how bootstrap works in a DNN context. The literature has developed many different forms of bootstrapping methods. One of its simplest and most popular forms is the percentile or naive bootstrap proposed by Efron (1979). Under this method observations are drawn from an independent and identically distributed sample with replacement and each observation has the same probability of being extracted.

Let $\{\mathbf{x}_i\}_{i=1}^M$ be a sample of M observations of the set of covariates, with $\mathbf{x}_i \in \mathbb{R}^d$ and M the length of the train sample. Let $\{\mathbf{y}_i\}_{i=1}^M \in \mathbb{R}$ be the output variable, and define $\mathbf{x}_i^{\llcorner} = (\mathbf{x}_i, y_i) \in \mathbb{R}^{d+1}$. Applying the naive bootstrap proposed by Efron (1979) to this multivariate dataset, we generate the bootstrapped dataset $\boldsymbol{x}^{\llcorner,\star} = \{\boldsymbol{x}_i^{\llcorner,\star}\}_{i=1}^M = \{\boldsymbol{x}_i^{\star}, y_i^{\star}\}_{i=1}^M$ by sampling with replacement from the original dataset $\mathbf{x}^{\llcorner}$ (with $^{\llcorner}$ indicating the paired sample); in this case, $\mathbf{x}^{\star}$ represents a bootstrapped replica of the dataset $\mathbf{x}$. By repeating this procedure T times, it is possible to obtain T bootstrapped samples defined as $\{\mathbf{x}^{\llcorner,\star(t)}\}_{t=1}^T$. Each bootstrap sample is fitted to a single neural network to obtain an empirical distribution of bootstrap predictions $f(\mathbf{x}^{\star(t)}; \widehat{\boldsymbol{\omega}}^{\star(t)})$, with $\widehat{\boldsymbol{\omega}}^{\star(t)} = \{\mathbf{W}^{1,\star(t)}, ..., \mathbf{W}^{N,\star(t)}, b_1^{\star(t)}, \ldots, b_N^{\star(t)}\}$, for $t = 1, \ldots, T$. In this context, a suitable bootstrap prediction interval for y_i at an α significance level is $\left[\widehat{q}_{\alpha/2}, \widehat{q}_{1-\alpha/2}\right]$, with $\widehat{q}_{\alpha}$ the empirical $\alpha-$quantile obtained from the bootstrap distribution of $f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{\star(t)})$, for $t = 1, \ldots, T$.

Alternatively, under the assumption that the error ϵ is normally distributed, we can refine the empirical predictive interval by using the critical value from the Normal distribution. A suitable prediction interval for $\mathbf{x}_i$ from the test sample, with $i = 1, \ldots, n$, is

$$f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}) \pm z_{1-\alpha/2} \widehat{\sigma}_{\epsilon}^{\star}, \tag{4.13}$$

with $f(\mathbf{x}_i; \widehat{\boldsymbol{\omega}})$ the pointwise prediction of model (4.6) and $\widehat{\sigma}_{\epsilon}^{\star 2} = \widehat{\sigma}_{\widehat{\boldsymbol{\omega}}}^{\star 2}(\mathbf{x}_i) + \widehat{\sigma}_e^2$. Under homoscedasticity of the error term ϵ_i, the aleatoric uncertainty σ_e^2 is estimated from the test

sample as $\widehat{\sigma}_e^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}))^2$, with $\widehat{\boldsymbol{\omega}}$ the set of parameter estimates obtained from the original sample $\{\mathbf{x}_i^{\llcorner}\}_{i=1}^{M}$. The epistemic uncertainty is estimated from the bootstrap samples as $\widehat{\sigma}^{\star 2}_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T}[f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{\star(t)}) - \bar{f}(\mathbf{x}_i)]^2$, with

$$\bar{f}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{*(t)}). \tag{4.14}$$

Unlike for the delta method, the use of bootstrap methods allows us to ameliorate the effect of the bias in the prediction of the ReLu DNN model. The bias in model (4.6) is defined as $E[f(\mathbf{x}_i;\boldsymbol{\omega})] - f(\mathbf{x}_i)$. Therefore, a suitable estimator of this quantity is $\bar{f}(\mathbf{x}_i) - f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}})$, with $\bar{f}(\mathbf{x}_i)$ defined in (4.14), such that the above prediction interval can be refined as

$$f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}) - \underbrace{(\bar{f}(\mathbf{x}_i) - f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}))}_{\text{bias correction}} \pm z_{1-\alpha/2}\widehat{\sigma}^{\star}_{\epsilon} = (2f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}) - \bar{f}(\mathbf{x}_i)) \pm z_{1-\alpha/2}\widehat{\sigma}^{\star}_{\epsilon}. \tag{4.15}$$

This bootstrap prediction interval can be further refined by exploiting the average prediction in (4.14). In this case the variance of the predictor is $\overline{\sigma}^{\star 2}_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) = \frac{1}{T}\widehat{\sigma}^{\star 2}_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)$ and the relevant prediction interval is [45]

$$\bar{f}(\mathbf{x}_i) \pm z_{1-\alpha/2}\widehat{\sigma}^{\star}_{\epsilon}, \tag{4.16}$$

with $\widehat{\sigma}^{\star 2}_{\epsilon} = \overline{\sigma}^{\star 2}_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) + \overline{\sigma}^2_e$, where $\overline{\sigma}^2_e = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{f}(\mathbf{x}_i)\right)^2$. This expression assumes that the covariance between the predictions from the different bootstrap samples is zero. Interestingly, in this case the bias correction is not necessary unless T is small. This is so because the bias term for the average predictor is negligible and given by $\frac{1}{T}\mu_i$.

As highlighted by Dipu Kabir et al. (2018), the variation in the outputs of the different networks will be driven by the different random initialization of the weights (parameter uncertainty) and the different bootstrap samples (data uncertainty). Being the bootstrap procedure able to capture both the aleatoric and epistemic uncertainties, it provides more accurate prediction intervals than other methods (delta method) as also shown in an extensive simulation study in Tibshirani (1996).

4.3.3 Monte Carlo Dropout (Stochastic Forward Passes)

This subsection introduces an alternative to bootstrap methods to construct prediction intervals in a ReLU DNN setting. In this case we introduce randomness into the DNN prediction by applying Monte Carlo dropout. A natural interpretation of this methodology follows from the seminal contribution of Gal and Ghahramani (2016a). These authors develop a new theoretical framework casting dropout training in DNNs as approximate Bayesian inference in deep Gaussian processes. As a byproduct of this theory, Gal and Ghahramani (2016a) provide

[45] In particular, the following prediction interval obtains by substituting $f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}})$ in Equation (4.15) with the average prediction $\bar{f}(\mathbf{x}_i)$.

the tools to model prediction uncertainty with dropout in DNNs. In this Section, we adopt this methodology to settings outside Bayesian neural networks and derive prediction intervals for the output y_i for DNNs in Bayesian and outside Bayesian DNN models.

A growing branch of the literature has been focusing on the Bayesian interpretation of dropout[46](see among others Gal and Ghahramani, 2016a and 2016b; Kingma et al., 2015). Maeda (2014) explains how dropout training can be considered an approximate learning method of the model parameters that optimizes a weighted sum of the likelihoods of all possible models. Starting from this interpretation, one could consider dropout as a tool for the estimation of the posterior of a Bayesian neural network. More specifically, let $p(\widehat{y} \mid \mathbf{x}, \mathbf{X}, \mathbf{Y})$ denote the distribution of the predictive output $\widehat{y}$ conditional on the set of observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{Y} = \{y_1, \ldots, y_n\}$. In other words, we denote by $\widehat{y}$ the observed output corresponding to the input of the neural network $\mathbf{x}$, and by $\mathbf{X}$, $\mathbf{Y}$ the input and output datasets. The predictive probability distribution of the DNN model is

$$p(\widehat{y} \mid \mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int_{\boldsymbol{\Omega}} p(\widehat{y} \mid \mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega}, \tag{4.17}$$

with $p(\widehat{y} \mid \mathbf{x}, \boldsymbol{\omega})$ the likelihood function of the observations, and $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ where $\boldsymbol{\Omega}$ denotes the parameter space. The posterior probability distribution $p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y})$ is intractable. Gal and Ghahramani (2016a) propose DNN dropout to approximate this distribution. More formally, under model dropout, we consider a distribution function $q(\boldsymbol{\omega})$ that follows a Bernoulli distribution, $\mathrm{Ber}(p)$, as explained in Section 4.2. The above predictive distribution in this Bayesian neural network setting can be approximated by

$$p(\widehat{y} \mid \mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int_{\Omega} p(\widehat{y} \mid \mathbf{x}, \boldsymbol{\omega}) q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}. \tag{4.18}$$

In practice this predictive distribution can be approximated using Monte Carlo methods. Thus, by sampling T sets of vectors from the Bernoulli distribution $\{\mathbf{r}^{\star(t)}\}_{t=1}^{T}$, one can approximate the above predictive distribution from the random sample $\widehat{y}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})$, for $i = 1, \ldots, n$, where $\widehat{\boldsymbol{\omega}}^{(t)} = \{\widehat{\mathbf{W}}^{1(t)}, \ldots, \widehat{\mathbf{W}}^{N(t)}, \widehat{b}_1^{(t)}, \ldots, \widehat{b}_N^{(t)}\}$ denotes the sequence of weights associated to the different nodes and layers of the neural network and the associated bias parameters for a given pass t for $t = 1, \ldots, T$. Using this Monte Carlo (MC) dropout technique, Gal and Ghahramani (2016a) propose the first moment from the MC predicted outputs as the model prediction:

$$\bar{f}_{MC}(\mathbf{x}_i) = \frac{1}{T} \sum_{t=1}^{T} \widehat{y}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}), \text{ for } i = 1, \ldots, n. \tag{4.19}$$

These authors show that, in practice, this is equivalent to performing T stochastic forward passes through the network and averaging the results. This result has been presented in the literature before as model averaging. Srivastava et al. (2014) have reasoned empirically that

[46] Hinton et al. (2012) in their seminal paper associate dropout training to a form of Bayesian learning.

MC dropout can be approximated by averaging the weights of the network (multiplying each weight $\mathbf{W}^n$ by p at test time, and referred to as standard dropout).

Importantly, the model parameters $\boldsymbol{\omega}$ are fixed across random samples implying that the cross-correlation between the predictions $\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})$ and $\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t')})$ for $t,t'=1,\ldots,T$ is perfect. Then, the predictive variance is defined as

$$\sigma^2_{MC}=\sigma^2_e+\frac{1}{T^2}\sum_{t=1}^{T}\sum_{t'=1}^{T}\mathbb{E}\left[\left(\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})-\mathbb{E}[\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})]\right)\left(\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t')})-\mathbb{E}[\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t')})]\right)\right], \tag{4.20}$$

The first component on the right hand side Expression of (4.20) captures the aleatoric uncertainty whereas the second term captures the epistemic uncertainty associated to parameter estimation. The second term includes the estimation of the variance and covariance terms between the different random samples obtained from using dropout. Thus, under the assumption that the approximation error is negligible, the above predictive variance can be estimated as

$$\widehat{\sigma}^2_{MC}=\widehat{\sigma}^2_e+\frac{1}{T}\sum_{t=1}^{T}\left(\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})-\bar{f}_{MC}(\mathbf{x}_i)\right)^2, \tag{4.21}$$

with $\widehat{\sigma}^2_e=\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\bar{f}_{MC}(\mathbf{x}_i)\right)^2$ a consistent estimator of σ^2_e under homoscedasticity of the error term, see also Gal and Ghahramani (2016a) and Kendall and Gal (2017). A suitable prediction interval for y_i under the assumption that $p(\widehat{y}\,|\mathbf{x},\boldsymbol{\omega})$ is normally distributed is

$$\bar{f}_{MC}(\mathbf{x}_i)\pm z_{1-\alpha/2}\widehat{\sigma}_{MC}. \tag{4.22}$$

Recent literature focusing on the approximation of the predictive distribution of DNNs has proposed several algorithms –based on the MC dropout of Gal and Ghahramani (2016a)– for the estimation of the prediction uncertainty in deep learning. As an example, Serpell et al. (2019) augment the MC dropout by implementing the MVE discussed above and stochastic forward passes. If the MVE approach allows modeling the data uncertainty, accommodating a varying e, the Monte Carlo dropout captures the uncertainty in the model parameters. The two procedures together ensure the correct estimation of σ^2_{MC}; Zhu and Laptev (2017) improve over the original Monte Carlo dropout by estimating the noise level using the residual sum of squares evaluated on a hold-out set[47] –Equation 4.21.

Finally, the present chapter highlights three important aspects related to the MC dropout originally proposed by Gal and Ghahramani (2016a). First, it is possible to extend the original approach to the approximation of the predictive distribution of deep neural networks outside a Bayesian framework (see Cortes-Ciriano and Bender, 2019). As one could notice, using dropout also at test phase allows randomizing the output of the DNN at each forward pass

[47] The authors precise that the approach of Gal and Ghahramani (2016a) relies on the implausible assumption of knowing the correct noise level a priori.

and thus, by performing T stochastic forward passes, it is possible to obtain the sample $\{\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})\}_{t=1}^{T}$. Second, if the MC dropout is implemented outside a Bayesian framework, it is pivotal to tune the dropout rate on a test sample and not on a train sample; in fact, as suggested by Lakshminarayanan et al. (2017), tuning the dropout rate on the training data implies interpreting dropout as a tool for Bayesian inference (any Bayesian posterior should be approximated starting only from the training data). Last but not least, the estimation of the σ^2_{MC} depends on the choice of p. As the epistemic uncertainty in the MC dropout is determined solely by the choice of p, if p is set equal to 1 the epistemic uncertainty measured by $\frac{1}{T}\sum_{t=1}^{T}\left(\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)}) - \bar{f}_{MC}(\mathbf{x}_i)\right)^2$ will be zero. Thus, the empirical coverage rate of the MC dropout will depend significantly upon the right choice of p (as opposed to the other analyzed methods). This final insight, which is also reported in Levasseur et al. (2017), has important implications in terms of the trade-off between out-of-sample accuracy and construction of prediction intervals. In fact, if the optimal dropout rate $1-p$ is effectively zero (no dropout should be implemented), the MC dropout approach would return narrow prediction intervals. In this circumstance, it will be necessary to reduce the out-of-sample prediction accuracy –by increasing the dropout rate– in order to return correct prediction intervals.

4.4 Extra-neural networks (Fixed Bernoulli Mask)

In this section we introduce a novel methodology to construct prediction intervals that is based upon the work of Srivastava et al. (2014). In this case, the original concept of ensemble of sub-networks—from which the dropout training is built upon—is adopted. The Bernoulli mask $\mathbf{r}^{\star}$ introduces an additional randomization scheme to the predictions obtained from the ensemble of neural networks that ensures independence and the consequential validity of the prediction interval (4.27) presented below.

For notation purposes, we will identify the fixed Bernoulli mask as $\bar{\mathbf{r}}^{\star}$ as opposed to $\mathbf{r}^{\star}$ used in dropout training. In other words, T sets of vectors $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$ are sampled from the Bernoulli distribution prior to training (instead of test time with Monte Carlo dropout) that are kept constant during both train and test phases. This approach reduces to train and independently fit T random sub-networks on the same dataset. In this setting, generating the predictive distribution is similar, in spirit, to an ensemble approach that trains different sub-neural networks on the same dataset. The proposed algorithm—being based on the extremely randomized trees proposed by Geurts et al. (2006)—is called *extra-neural networks*.

Before analyzing the prediction intervals for the extra-neural network, it is necessary to analyze the relation between the correlation among the different models that constitute an ensemble and the error of the latter (see also Brown et al., 2005; and Zhou, 2012) that defines the aleatoric uncertainty. In particular, consider T fitted sub-networks defined as $f_t(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})$ with $t = 1, \cdots, T$. We use f_t to note that each prediction belongs to a potentially different model; $\widehat{\boldsymbol{\omega}}^{(t)}$ denotes the parameter estimates obtained from fitting each sub-network independently. The prediction of the extra-neural network model is

$$\bar{f}_{EN}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}), \text{ for } i = 1, \ldots, n. \tag{4.23}$$

Given the ensemble predictor expressed above, it is possible to compute the mean square prediction error (MSPE) of the prediction conditional on the regressor vector $\mathbf{x}_i$. Then,

$$MSPE(\bar{f}_{EN}(\mathbf{x}_i)) \equiv \mathbb{E}[(\bar{f}_{EN}(\mathbf{x}_i) - y_i)^2] = \text{Bias}^2(\bar{f}_{EN}(\mathbf{x}_i)) + V(\bar{f}_{EN}(\mathbf{x}_i)). \tag{4.24}$$

We compute the conditional bias and variance of $\bar{f}_{EN}(\mathbf{x}_i)$ as

$$\text{Bias}(\bar{f}_{EN}(\mathbf{x}_i)) \equiv \mathbb{E}[\bar{f}_{EN}(\mathbf{x}_i) - y_i] = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})] - f(\mathbf{x}_i), \tag{4.25}$$

and

$$V(\bar{f}_{EN}(\mathbf{x}_i)) \equiv \mathbb{E}\left(\bar{f}_{EN}(\mathbf{x}_i) - \mathbb{E}[\bar{f}_{EN}(\mathbf{x}_i)]\right)^2 = \mathbb{E}\left[\bar{f}^2_{EN}(\mathbf{x}_i)\right] - \mathbb{E}^2[\bar{f}_{EN}(\mathbf{x}_i)],$$

such that

$$V(\bar{f}_{EN}(\mathbf{x}_i)) = \frac{1}{T^2}\sum_{t=1}^{T}\sum_{t'=1}^{T}\left(\mathbb{E}[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}) f_{t'}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t')})] - \mathbb{E}[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})]\mathbb{E}[f_{t'}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t')})]\right).$$

Furthermore, assuming that the first two statistical moments of all the individual predictors indexed by $t = 1, \ldots, T$ are equal, with $\mathbb{E}\left[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})\right] = f(\mathbf{x}_i) + \mu_i$, where μ_i is the bias term, $\mathbb{V}\left[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})\right] = \sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)$, and $\text{Cov}\left[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}) f_{t'}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t')})\right] = c_i$, we obtain

$$MSPE(\bar{f}_{EN}(\mathbf{x}_i)) = \mu_i^2 + \frac{1}{T}\sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) + \frac{T-1}{T}c_i. \tag{4.26}$$

This expression extends Zhou (2012) by showing that the MSPE of the ensembler (4.23) depends on the variance of the individual ensemblers, their covariance and the approximation bias. The smaller the covariance, the smaller the generalization error of the ensemble. In contrast, if the different predictors are perfectly correlated (as for the MC dropout) we know that $c_i = \sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)$ and thus $MSPE(\bar{f}_{EN}(\mathbf{x}_i)) = \sigma^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)$—effectively reducing to zero the effect of ensembling. Similarly, the MSPE is minimized when the errors are perfectly uncorrelated and thus when $c_i = 0$.

This result has important implications when analyzing the epistemic uncertainty of an extra-neural network. If it is assumed that the correlation among the predictions from the sub-networks is equal to zero[48], then as $T \to \infty$, the $MSPE(\bar{f}_{EN}(\mathbf{x}_i))$ converges to zero, assuming that the approximation bias is negligible. Therefore, a suitable prediction interval is

$$\bar{f}_{EN}(\mathbf{x}_i) \pm z_{1-\alpha/2}\left(\frac{\widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)}{T} + \widehat{\sigma}^2_e\right)^{1/2}, \tag{4.27}$$

[48] Monte Carlo simulation results show an average correlation of 0.0498; the results are available upon request.

with $\widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)}) - \bar{f}_{EN}(\mathbf{x}_i))^2$ and $\widehat{\sigma}^2_e = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{f}_{EN}(\mathbf{x}_i)\right)^2$, where n is the length of the test sample.[49]

As explained in Zhou (2012), the covariance term in Equation (4.26) captures the diversity existing among the T different sub-networks identifying the extra-neural network. The aim of the extra-neural network approach proposed in this chapter is to construct individual predictors that are mutually independent such that the prediction interval (4.27) is valid. The diversity in the model predictions depends on the variance of the Bernoulli masks generated by the random sample $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$. It is well known that the variance of a Bernoulli distribution is defined as $\varsigma^2 = p(1-p)$; therefore, it can be easily shown that the solution to $\partial\varsigma^2/\partial p = 0$ is $p = 1/2$ and that $\partial^2\varsigma^2/\partial p^2 = -2$ showing that ς^2 is maximized in $p = q = 0.5$. Consequentially, one could conclude that the covariance in equation 4.24 is minimized for $p = 0.5$ and maximized for $p = 0$ and $p = 1$, see also Figure 4.2:

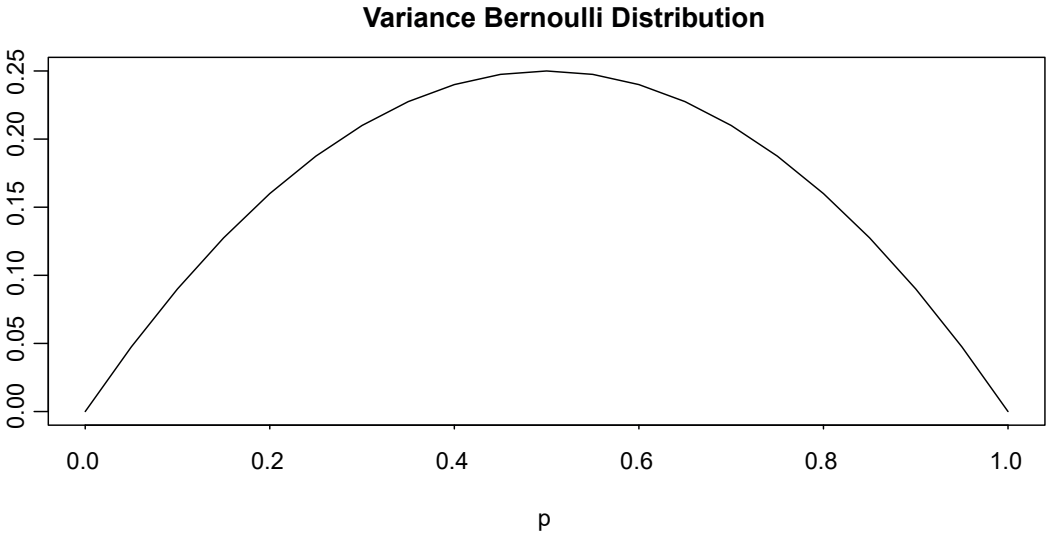


Figure 4.2: Bernoulli Variance. The variance (ς^2) is maximized for $p = 0.5$ and minimized ($= 0$) for $p = 0$ and 1.

However, a complete analysis of the covariance of an ensemble of neural networks must consider the relation existing between the number of hidden nodes and the particular data generating process analyzed. Based on the literature on approximation theory and DNNs, the number of hidden nodes defines the approximation power (or flexibility) of the neural networks (for a summary on the topic see chapter 3 above). Farrell et al. (2019), by comparing DNN structures to different nonparametric techniques for approximating unknown continuous functions, also make explicit the dependence between the number of hidden nodes in the DNN (Z_{tot}) and the approximation power. If the size of the networks is such that the *ambiguity*—measure of disagreement among the different networks on a specific input, see Krogh

[49] Note that for obtaining a consistent estimator of $\widehat{\sigma}^2_e$ we have imposed homoscedasticity of the error terms ϵ_i over the test sample.

and Vedelsby (1995) for a detailed analysis—is too low, the assumption of $c = 0$ becomes unrealistic.

Based on the above paragraph, one could conclude that the analysis of the covariance in an extra-neural network must consider not only p that determines the variance of $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$ but also the particular data generating process under study, as Z_{dropout} is also determined by p. As the two effects must be considered together when choosing the probability p, one must consider that p converging to 0.5 from above or from below may have a similar impact in terms of decrease in c but and opposite effect on the dimension of Z_{dropout}. As p converges to 0.5 from below, the dimensions of the sub-networks will increase (higher probability for each neuron to be 1 and thus to be retained in the sub-network). Conversely, p converging to 0.5 from above will ensure a reduction of the number of hidden nodes in the T sub-networks (higher probability of being "dropped out" - $q = 1 - p$).

Algorithm 3 Extra-neural networks

INPUT: Training Data $\{x_i^{\llcorner} \equiv (\mathbf{x}_i, y_i)\}_{i=1}^{M}$
OUTPUT: Prediction Interval $\widehat{f}(\mathbf{x}; \boldsymbol{\omega})$.

procedure T LEARNERS

while $(t < T)$ **do**
Generate a Bernoulli mask $\bar{\mathbf{r}}^{\star}$ prior to training.
Apply Bernoulli mask $\bar{\mathbf{r}}^{\star}$ to the *original* neural network.
Train random thinned network on $\mathbf{x}^{\llcorner}$ with random initialization of $\{\mathbf{W}_0^n\}_{n=1}^{N}$
Store $f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})$.

Compute the ensemble estimate:

$$\bar{f}_{EN}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}) \tag{4.28}$$

Compute the epistemic and aleatoric variance:

$$\begin{cases} \widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T}[f_t(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}) - \bar{f}_{EN}(\mathbf{x}_i)]^2 \\ \widehat{\sigma}^2_e = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{f}_{EN}(\mathbf{x}_i)\right)^2 \end{cases} \tag{4.29}$$

Define Prediction Interval:

$$\bar{f}_{EN}(\mathbf{x}_i) \pm z_{1-\alpha/2}\widehat{\sigma}_{\epsilon}, \tag{4.30}$$

with $\widehat{\sigma}_{\epsilon} = \left(\frac{\widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)}{T} + \widehat{\sigma}^2_e\right)^{1/2}$.

return Prediction interval (4.30)

Algorithm 3 reports the procedure to be used for implementing the extra-neural network. In order to generate $\{f_t(\mathbf{x};\widehat{\boldsymbol{\omega}}^{(t)})\}_{t=1}^{T}$, we sample T vectors $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$ prior to training. Each fixed Bernoulli mask is applied independently to the original network returning T independent sub-networks of size $Z_{\text{dropout}}^{(t)} \leq Z_{\text{tot}}$. Each sub-network is then trained independently on $\mathbf{x}^{\llcorner}$, and T deterministic forward passes are performed at test phase. Thus, even if the novel algorithm is based upon the original idea of dropout proposed by Srivastava et al. (2014) and introduces randomness by means of a random sample $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$, it is closer to classical ensemble methods than to training with dropout. This has important implications while training. In this case, performing weight scaling at test phase (or train phase if the algorithm is implemented in Keras) is not required as the Bernoulli mask is applied before training. Training T independent sub-networks identified by $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$ makes no longer necessary to ensure that the expected total input to the units of a DNN at test time is approximately the same as the expected value at training (see Goodfellow et al., 2016).

The procedure reported in Algorithm 3 shows that an extra-neural network is an ensemble of T neural networks with randomized weights and structures and no data resampling. Based on the results reported by Pearce et al. (2018), Lee et al. (2015) and Lakshminarayanan et al. (2017) regarding deep ensembles [50] it is expected that the extra-neural network algorithm will improve over a bootstrapping ensemble approach. More precisely, Lee et al. (2015) show how parameter resampling without bootstrap resampling—equivalent to training T different $f(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})$ on $\mathbf{x}^{\llcorner}$—outperforms a bootstrap approach (analyzed in Subsection 4.3.2) in terms of predictive accuracy; Lakshminarayanan et al. (2017) complement the results of Lee et al. (2015) by showing that data resampling in deep ensembles deteriorates not only the prediction accuracy but also the definition of the predictive uncertainty of the ensemble itself. Although these authors do not provide robust justification for these results, recent research from El Karoui and Purdom (2018) –focused on the construction of confidence intervals for linear regression via bootstrapping procedures in mid and high-dimensional settings– provides a useful insight. In particular, as the number of observation grows to infinity, the probability of sampling a unique observation tends to a Poisson distribution with $\lambda = 1$, and thus to 0.63. The reduction of unique observations has a negative impact not only on the prediction accuracy of the neural network (by being data hungry), but also on the construction of confidence intervals (which can also be translated to the construction of prediction intervals). When $d/M \to 0.63$ and pairs bootstrapping is implemented, the bootstrap design matrix will be singular even if the original design matrix was of full rank equal to d. El Karoui and Purdom (2018) explain how this leads to the complete unreliability of bootstrapping (pairs) methods for the computation of the variance of confidence intervals in mid-dimensional settings. In their simulation exercise, the authors show how even for a value of $d/M \approx 0.2$, pairs bootstrapping returns over-specified (higher width) confidence intervals. Therefore, even if applied to linear regressions, the results of El Karoui and Purdom (2018), provide the basis for a deeper understanding of the results found in Lakshminarayanan et al. (2017) and Lee et al. (2015).

[50] Deep ensembles and ensembles of DNNs are considered synonym for the rest of the chapter.

Therefore, the extra-neural networks by randomizing not only the weights of the T sub-networks but also their structure, and by fitting the networks on the entire training set $\{\mathbf{x}_i\}_{i=1}^{M}$, are expected to outperform the bootstrap approach in terms of both out-of-sample prediction accuracy (Lee et al., 2015) and uncertainty quantification (Lakshminarayanan et al., 2017) [51].

To summarize, the MC dropout trains a single neural network with dropout training (the Bernoulli masks are sampled prior to each gradient step, and thus are variable throughout the training of the single neural network) on the original dataset. The extra-neural network is an ensemble algorithm that trains T different random sub-networks on the same dataset. The random sub-networks are identified via Bernoulli masks that are sampled prior to training (thus, for each sub-network, the set of random Bernoulli masks is kept fixed throughout the training). The bootstrap approach trains the same neural network structure (i.e., MC dropout with $p = 1$ –no dropout) on resampled versions of the dataset.

The main drawback of the extra-neural network algorithm is associated to the computing power required. In particular, if the computational requirements of the proposed methodology are equivalent to existing bootstrapping procedures (with and without data resampling), they are significantly greater than the ones of the MC dropout methodology. However, due to the parameter sharing in the MC dropout, the extra-neural networks will ensure a lower MSPE (see Equation 4.26). Additionally, it is expected an improvement also in terms of hyperspace: the novel methodology allows reaching a good estimation performance without the pivotal fine-tuning that is required by the other procedures. As in the case of bootstrap based procedures, the independence among the different learners in the extra-neural networks allows parallel computing ensuring savings in computational time. Last but not least, the extra-neural network improves over a bootstrap based approach in terms of applicability: if the bootstrap approach relies on the assumption of $i.i.d$ observations, the extra-neural network does not.

Lastly, a comparison with random forests is also discussed to further clarify the novel methodology. Random forests (Breiman, 2001) are characterized by high generalization performance by reducing the correlation among the different trees via two different sources of randomness: a) resampling with replacement, and b) subsampling the set of features adopted in the recursive binary splitting algorithm. Conversely, the extremely randomized trees (Geurts et al., 2006) no longer perform resampling with replacement and introduce an extra source of randomness by randomizing the threshold adopted while performing the binary splitting for a given subsample of features. One could notice how the randomness introduced in the ensemble of trees increases as the dimension of the subsample of features decreases.

[51] By considering deep ensembles the equivalent of a random forest (Breiman, 2001) where the single learners are neural networks and where the parameter uncertainty is captured not by the random subset selection of features at each node (trees) but by random weight initialization, the extra randomization introduced by extra-neural networks is comparable to the extremely randomized trees in Geurts et al. (2006). In this case, randomizing also the structure is equivalent to randomizing the cut-point at each node in a tree.

We will now extend the above discussion to ensembles of neural networks. In particular, when the bootstrapping approach is performed, there are two different sources of randomness: a) resampling with replacement, and b) random weight initialization. Conversely, the extremely randomized networks do not perform resampling with replacement and introduce an extra source of randomness by randomizing the structure (in terms of dimension and in terms of allocation of nodes across layers) of the different learners. If, for example, the distribution adopted for random weight initialization is $\mathcal{N}(0, \sigma)$, one could notice how the randomness introduced in the ensemble of neural networks increases as σ increases. To generalize, the choice of the distribution for the initialization of the weights is a driving factor of the randomness among learners. Therefore, random weight initialization in ensembles of neural networks plays a role similar to feature subsampling in ensembles of trees.

Alternatively, one could introduce feature subsampling also in ensembles of neural networks. In the case of extra-neural networks, this could be achieved by allowing for a fixed dropout mask not only in the hidden layers but also in the input layer. However, this final approach may have a negative impact on the imputation of the aleatoric uncertainty and thus on the correct construction of prediction intervals. In fact, Pearce et al. (2018) explain how the aleatoric uncertainty is driven by the underlying randomness of the data generating process and also by the presence of missing variables. A possible solution would be coupling the dropout in the input layer with the extraction of feature importance from a random forest: given the feature importance based on the fitting of a random forest, instead of imposing to each input node the same probability $1 - p$ of being dropped out, it would be possible to construct weighted probabilities for the input nodes ensuring that the less important features will have a higher probability of being dropped out, while the most important ones to be retained in the network (an example of the application of the feature importance from a random forest for the implementation of dropout can be found in Santra et al., 2020).

All the results analyzed in Section 4.3 and 4.4 will be formally evaluated in an extensive simulation study focused on assessing if the reported procedures return correct prediction intervals (empirical coverage close to the nominal one) for different significance levels and data generating processes. Finally, the empirical experimental setting of Hernández-Lobato and Adams (2015) is implemented to compare the performance (in terms of RMSPE) of the different algorithms.

4.5 Monte Carlo simulation

The Monte Carlo simulation will analyze the empirical coverage rates of the prediction intervals obtained from Expression (4.16) (bootstrap approach), Expression (4.22) (MC dropout) and Expression (4.27) (extra-neural network). For each prediction interval, the empirical coverage rates ($\bar{\alpha}$) for three different significance levels (0.01, 0.05, and 0.10) are computed. This allows evaluating the correctness of the constructed prediction intervals for different significance levels. All three procedures are analyzed for increasing $T = [30, 50, 70]$, and for a sample size $M + n = 1200 + 300$. When the small-dimensional linear process is considered—in order

to evaluate the impact that different ps may have on the correct definition of the prediction intervals [52] —we will consider $p = [0.995, 0.990, 0.950, 0.900, 0.800]$. Subsection 4.5.1 reports the setting for the simulation of the small dimensional linear and nonlinear data generating processes; Subsection 4.5.2 summarizes the results.

4.5.1 Data Generating Processes

When the nonlinear data generating process (DGP) is considered, the dataset $\mathbf{x} \in \mathbb{R}^5$ is defined by $\mathbf{x}_1 \sim \mathcal{N}(-4, 1)$, $\mathbf{x}_2 \sim \mathcal{N}(2, 1)$, $\mathbf{x}_3 \sim \mathcal{N}(2, 1)$, $\mathbf{x}_4 \sim \mathcal{N}(2, 1)$, and $\mathbf{x}_5 \sim \mathcal{N}(4, 1)$, the means of which are randomly sampled with replacement from a domain defined in $[-5, 5]$. In order to introduce correlation among the variables, the Choleski decomposition is applied. The desired correlation matrix is defined as:

$$\mathbf{C} = \begin{bmatrix} 1 & 0.5 & 0.6 & 0.7 & 0.5 \\ 0.5 & 1 & 0.7 & 0.8 & 0.5 \\ 0.6 & 0.7 & 1 & 0.7 & 0.5 \\ 0.7 & 0.8 & 0.7 & 1 & 0.8 \\ 0.9 & 0.5 & 0.6 & 0.8 & 1 \end{bmatrix} \tag{4.31}$$

Before imposing the correlation structure in $\mathbf{C}$, it is necessary to make sure that the simulated variables are independent. To do so, the current correlation matrix $\mathbf{\Sigma}$ is calculated; following, the inverse of the Cholesky factorization ($\mathbf{A}^{-1}$) of $\mathbf{\Sigma}$ is computed. By matrix multiplying $\mathbf{A}^{-1}$ and $\mathbf{x}$, we will ensure that the obtained dataset will be defined by independent Normally distributed variables. Finally, the Cholesky factorization ($\mathbf{A}$) of $\mathbf{C}$ is calculated and multiplied by the simulated dataset, ensuring that $\mathbf{Z} = \mathbf{x}\mathbf{A} \approx \mathcal{N}(0, \mathbf{C})$.

The nonlinear DGP is defined by a ReLu DNN with two hidden layers of width 3 and 2 respectively, and bias equal to 1 across all hidden layers [53] :

[52] The choice of the dropout rate $q = 1 - p$ is dictated by a really small network size and also a fairly small simulated dataset.

[53] A similar DGP is also simulated in Tibshirani (1996) with $\mathbf{x} \in \mathbb{R}^4$, and a shallow network with sigmoid activation functions and two hidden nodes; the Gaussian error $\boldsymbol{\epsilon}$ follows the same distribution.

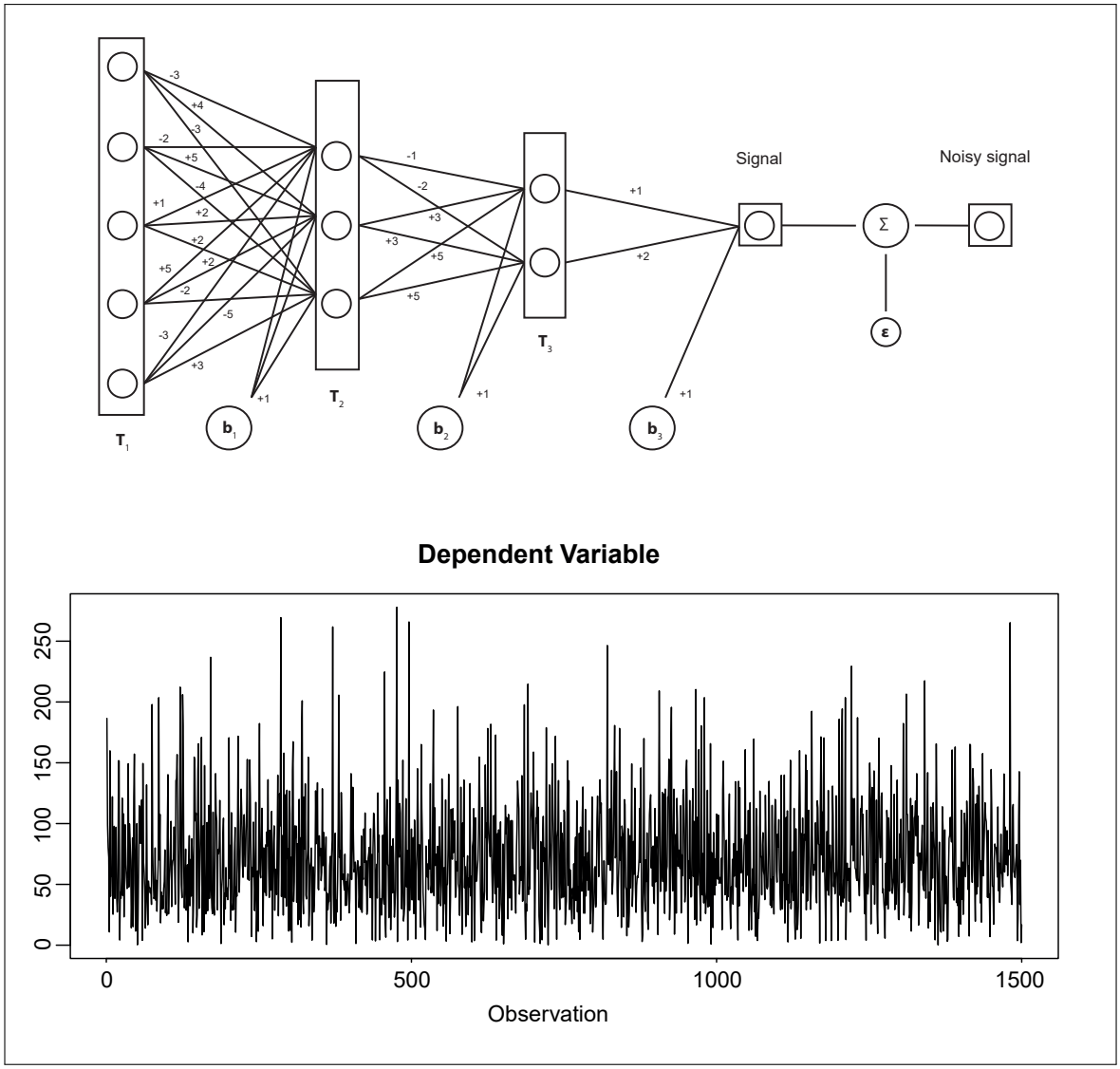


Figure 4.3: Data Generating Process. Top panel presents the underlying signal with randomly sampled coefficients and the bottom panel presents the noisy $\mathbf{y}$

which is also defined as

$$\mathbf{T}_1 = \underbrace{\theta(1 - 3\mathbf{x}_1 - 2\mathbf{x}_2 + 1\mathbf{x}_3 + 5\mathbf{x}_4 - 3\mathbf{x}_5)}_{\mathbf{h}_{11}} + \underbrace{\theta(1 + 4\mathbf{x}_1 + 5\mathbf{x}_2 + 2\mathbf{x}_3 + 2\mathbf{x}_4 - 5\mathbf{x}_5)}_{\mathbf{h}_{21}}$$
$$+ \underbrace{\theta(1 - 3\mathbf{x}_1 - 4\mathbf{x}_2 + 2\mathbf{x}_3 - 2\mathbf{x}_4 + 3\mathbf{x}_5)}_{\mathbf{h}_{31}}$$

$$\mathbf{T}_2 = \underbrace{\theta(1 - 1\mathbf{h}_{11} + 3\mathbf{h}_{21} + 5\mathbf{h}_{31})}_{\mathbf{h}_{12}} + \underbrace{\theta(1 - 2\mathbf{h}_{11} + 3\mathbf{h}_{21} + 5\mathbf{h}_{31})}_{\mathbf{h}_{22}}$$

$$\mathbf{y} = 1 + \mathbf{h}_{12} + 2\mathbf{h}_{22} + \boldsymbol{\epsilon}$$

with $\epsilon \sim \mathcal{N}(0, 0.7)$, $\theta(\mathbf{x}) = \max\{0, \mathbf{x}\}$, and the coefficients (network weights) randomly sampled with replacement from $[-5, 5]$. The standard deviation of the error term is set equal to 0.7 in order to reduce the nuisance in the system by differentiating the stochastic behavior of the regressors $\mathbf{x}$ and of the error term.

Following, a linear DGP that allows for interactions among the variables is also simulated. Also in this case $\mathbf{x} \in \mathbb{R}^5$, with $\mathbf{x}_1 \sim \mathcal{N}(-4, 1)$, $\mathbf{x}_2 \sim \mathcal{N}(1, 1)$, $\mathbf{x}_3 \sim \mathcal{N}(1, 1)$, $\mathbf{x}_4 \sim \mathcal{N}(1, 1)$, and $\mathbf{x}_5 \sim \mathcal{N}(5, 1)$. The cross-correlation matrix is defined in 4.31. The analyzed DGP is [54]

$$y = -8\mathbf{x}_1 + 2\mathbf{x}_2 + 2\mathbf{x}_3 + 2\mathbf{x}_4 + 7\mathbf{x}_5 + 3\mathbf{x}_1\mathbf{x}_2 - \mathbf{x}_3\mathbf{x}_5 + 2\mathbf{x}_1\mathbf{x}_4 + \epsilon \tag{4.32}$$

The parameters chosen for the vector of coefficients are generated from a $U[-10, 10]$ and then rounded to the closest digit; the error term is $\epsilon \sim \mathcal{N}(0, 1)$ and it is uncorrelated with the input variables.

For both linear and nonlinear DGPs, a total of 1500 observations are generated, 1200 observations are used for the training set and 300 for the test set. The datasets are normalized so that $\mathbf{x}$ has zero mean and unit variance.

When fitting the neural networks, no optimal tuning of the neural network hyper-parameters and structure is conducted[55]. The reasons for imposing the network hyper-parameters as opposed to fine-tuning them are: (I) it is ensured that the simulation results obtained are not dependent on fine-tuning; (II) it allows conducting a comparison of the empirical coverage rates across the three different methodologies analyzed, and (III) it allows analyzing the impact that different ps may have on the empirical coverage probabilities.

When the nonlinear DGP is simulated, it is assumed that the neural network structure is known ($Z_1 = 3$ and $Z_2 = 2$). Conversely, when the linear DGP is analyzed—as the true network structure is unknown, and due to the simplicity of the DGP—a shallow network with 5 hidden nodes is considered. When a nonlinear DGP is analyzed a $p = 0.995$ is applied (the true network structure is known and thus a low dropout rate is required); conversely, when a linear DGP is fitted, by imposing $p = [0.995, 0.990, 0.950, 0.900, 0.800]$, it is possible to analyze the impact that different ps may have on the empirical coverage rates of the obtained prediction intervals. A sensible choice of the network parameters for the linear process is to use the Adam optimizer with learning rate 0.1 and 10 epochs; for the nonlinear process the Adam optimizer with learning rate 0.01 and 80 epochs.

[54] The interaction terms are introduced in order to have an unknown network structure. In fact, if no interactions are assumed, the true network structure is a shallow network with one hidden node.

[55] For the correct choice of the network hyper-parameters, the analyst should ensure that the test set used for parameter tuning and for the aleatoric uncertainty computation is distinct—that is, a hold-out set should also be generated—otherwise, the consequential under-estimation of the aleatoric uncertainty could lead to narrower prediction intervals.

4.5.2 Simulation Results

Table 4.1 reports the out-of-sample performance and the empirical coverages of the three procedures analyzed. When the nonlinear DGP is considered, one could notice that the three methodologies return—for the three different significance levels—prediction intervals with empirical coverage probabilities approximately equal to the theoretical ones. Focusing on the linear DGP, one could notice that the bootstrap approach returns prediction intervals with empirical coverages approximately equal to the significance level at which they are constructed; when the extra-neural network is considered, all prediction intervals, for the different ps considered, have an empirical coverage probability approximately equal to the nominal one; conversely, the MC dropout returns correct prediction intervals only for given values of p.

As explained in the previous sections, the epistemic uncertainty in the MC dropout is captured exclusively by dropout at test time (and thus by the dropout rate $q = 1 - p$). Conversely, when the extra-neural network is analyzed, the epistemic uncertainty depends not only on the dropout rate considered, but also on the random weight initialization used for fitting the T sub-networks. As a result, the correct construction of the prediction intervals using the MC dropout approach requires to identify the optimal dropout rate as opposed to the extra-neural network algorithm proposed in the present chapter.

The present research extends the results of Levasseur et al. (2017). Similarly, these authors state that the construction of prediction intervals with empirical coverage rates approximately equal to the theoretical ones—using the MC dropout approach—depends on the correct choice of the dropout rate. Consequentially, these authors suggest that the dropout rate should be tuned to return the correct prediction intervals. The theoretical analysis in Section 4.3 coupled with the results in Table 4.1 clearly show that the prediction intervals computed from the MC dropout rely significantly on the correct choice of the dropout rate. These results also suggest that choosing the dropout rate that maximizes the out-of-sample accuracy guarantees prediction intervals with the correct $\bar{\alpha}$ (the out-of-sample error is minimized for the p that returns correct prediction intervals).

Although the results in Table 4.1 show that all three procedures return prediction intervals with empirical coverage probabilities close to the theoretical ones for both linear and nonlinear DGPs, the performance of the extra-neural network approach is clearly superior in terms of coverage probabilities and also MAPE and MSPE errors. This is particularly the case for $\alpha = 0.10$. This outperformance is especially remarkable for the linear process for which we do not impose or know a priori the true structure of the network. Finally, focusing on the out-of-sample performance, one could notice that: (i) the out-of-sample errors decrease as T increases, and (ii) for given dropout rates, the ensemble of neural networks outperforms the bootstrap approach.

Table 4.1: The table reports the out-of-sample mean average prediction error (MAPE) and mean squared prediction error (MSPE) for the analyzed procedures. EN_1 refers to the extra-neural network fitted for a nonlinear DGP, EN_2 for a linear DGP. MC_1 refers to the MC dropout for a nonlinear DGP, MC_2 for a linear DGP. Finally, $BOOT_1$ reports the results for the bootstrap approach a nonlinear DGP, $BOOT_2$ for a linear process.

	Nonlinear			**Linear**										
	$\mathbf{EN_1}$	$\mathbf{MC_1}$	$\mathbf{BOOT_1}$	$\mathbf{EN_2}$					$\mathbf{MC_2}$					$\mathbf{BOOT_2}$
p	0.995	0.995	-	0.995	0.990	0.950	0.900	0.800	0.995	0.990	0.950	0.900	0.800	-
T = 30														
MAPE	1.4979	3.5218	1.8476	1.0322	1.0493	1.1113	1.1196	1.2383	1.2993	1.3152	1.5834	1.6290	2.0478	1.0451
MSPE	3.8232	19.8904	5.4190	1.7208	1.7544	2.0327	2.0315	2.6099	2.9998	2.9527	4.1508	4.0322	7.0050	1.7037
$\mathbf{Cov_{99}}$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.00	0.01	0.00	0.01
$\mathbf{Cov_{95}}$	0.05	0.04	0.04	0.03	0.03	0.05	0.04	0.06	0.04	0.04	0.01	0.01	0.00	0.03
$\mathbf{Cov_{90}}$	0.07	0.08	0.06	0.09	0.09	0.08	0.10	0.08	0.07	0.07	0.02	0.02	0.00	0.09
T = 50														
MAPE	1.5068	3.5404	1.4480	1.0419	1.0808	1.0668	1.0940	1.2034	1.3044	1.3124	1.5337	1.5842	2.0583	1.0671
MSPE	3.6592	20.0717	3.4133	1.7332	1.8930	1.7991	1.9732	2.4329	3.0670	2.8893	3.8274	3.9688	6.7150	1.8043
$\mathbf{Cov_{99}}$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.01	0.00	0.01
$\mathbf{Cov_{95}}$	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.04	0.01	0.01	0.00	0.04
$\mathbf{Cov_{90}}$	0.08	0.08	0.09	0.09	0.09	0.09	0.09	0.10	0.08	0.08	0.01	0.02	0.00	0.08
T = 70														
MAPE	1.4756	3.5200	1.6426	1.0423	1.0467	1.1051	1.1611	1.1980	1.3026	1.3042	1.5277	1.5603	2.0060	1.0522
MSPE	3.5096	20.1656	4.3616	1.7131	1.7315	1.9444	2.2202	2.4559	3.0330	2.9104	3.8339	3.8921	6.6385	1.7290
$\mathbf{Cov_{99}}$	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.01
$\mathbf{Cov_{95}}$	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.00	0.01	0.00	0.04
$\mathbf{Cov_{90}}$	0.09	0.08	0.08	0.10	0.10	0.09	0.09	0.10	0.07	0.08	0.01	0.02	0.00	0.08

To summarize, the simulation results show that the proposed extra-neural network methodology not only returns correct prediction intervals but it also improves the forecast accuracy for both deep and shallow ensembles. Based on Equation 4.26, one could also notice that the coverage probability of a prediction interval improves not only by correctly estimating the variance but also by providing more accurate pointwise predictions of the true observations. Therefore, the following Section, by using the experimental settings of Hernández-Lobato and Adams (2015), evaluates the out-of-sample accuracy in terms of root mean square prediction error (RMSPE) of the novel approach for real world datasets.

4.6 Empirical Analysis

Hernández-Lobato and Adams (2015) after proposing a novel scalable method for learning Bayesian neural networks—called probabilistic backpropagation (PBP)—evaluate the performance of their novel methodology on real world datasets. The experimental settings used in their evaluation are widely adopted by the literature focusing on deep learning (see for example Gal and Ghahramani 2016a; and Lakshminarayanan et al., 2017) when evaluating novel algorithms. Therefore, using their experimental setup ensures comparability of the results with the variational inference method by Graves (2011), the probabilistic backpropagation of Hernández-Lobato and Adams (2015), the MC dropout in Gal and Ghahramani (2016a), and the deep ensemble approach developed by Lakshminarayanan et al. (2017).

The original experiment of Hernández-Lobato and Adams (2015) evaluates the models not only in terms of RMSPE but also in terms of predictive log-likelihood (the latter being extremely relevant in Bayesian learning). Being the present chapter focused on evaluating the accuracy of different procedures in constructing prediction intervals for regression tasks, only the former performance metrics will be considered. In fact, if the simulation (reported in the previous Section) analyzes the correctness of the prediction intervals obtained from state-of-the-art methodologies designed not only for conditional mean but also variance estimation for both shallow and deep networks, it does not assess the performance of the extra-neural network approach for large datasets. Therefore, the present empirical application (focused on shallow structures in order to allow for cross-comparability) complements the results reported in Table 4.1 by analyzing the RMSPE of the extra-neural network in large dimensional settings. The obtained RMSPEs (see also Equation 4.26), by capturing both bias and variance of the predictions, provide an additional indication regarding the accurateness of the prediction intervals obtained from the extra-neural network algorithm.

The experimental setup is as follows: 10 datasets are analyzed. Each dataset is split into random training (0.90 of the observations) and test (0.10 of the observations) sets 20 times and the average test set performance (RMSPE) and relative standard error are reported. As an exception, the protein and Year Prediction MSD datasets are split only 5 and 1 times into train and test sets. The datasets are normalized to guarantee that the regressors have zero mean and unit standard deviation. The same network architecture is considered: 1-hidden layer ReLu neural network with $Z_1 = 50$ for the small datasets and $Z_1 = 100$ for the larger protein and Year Prediction MSD datasets. Each neural network is trained for 40 epochs. Following Gal and Ghahramani (2016a), we use a dropout rate of 0.05, Adam optimizer and a batch size of 32. We decide to use the same dropout rate as in Gal and Ghahramani (2016a) for comparability reasons. We refer to Gal and Ghahramani (2016a), Hernández-Lobato and Adams (2015), and Lakshminarayanan et al. (2017) for additional details on the implementation of their algorithms. Lakshminarayanan et al. (2017) use 5 networks in their ensemble, and Gal and Ghahramani (2016a) perform 10000 stochastic forward passes[56]. In order to allow for a fair comparison between the deep ensemble of Lakshminarayanan et al. (2017) and the novel algorithm proposed in the present chapter, we will fit, at first, an extra-neural network with 5 sub-networks; following, in order to compare the predictive performance of Algorithm 3 with the MC dropout of Gal and Ghahramani (2016a), an extra-neural network with 70 sub-networks will also be considered.

Table 4.2 reports the average RMSPE and relative standard errors[57]; in bold are reported the lowest average RMSPEs. The authors—as opposed to what could be inferred from the related literature—indicate that it is not possible to ascertain the outperformance of one procedure over the competitors by relying solely on the average (over the resampled train and test sets) RMSPE; it is necessary to consider also the reported standard errors. Thus,

[56] This is not directly reported by the authors and it is inferred from the code reported in their Github page (Gal and Ghahramani, 2016c).

[57] For the last dataset, it is not possible to compute the SE as only 1 split is performed.

the extra-network ($T = 70$) is shown to outperform the competing algorithms in four cases (excluding the Year Protection MSD dataset [58]); both MC dropout and deep ensemble models are shown to outperform the other procedures in one case. When comparing the deep ensemble of Lakshminarayanan et al. (2017) and the extra-neural network ($T = 5$), the extra-neural network is shown to outperform five times, the deep ensemble three times[59].

Table 4.2: The table reports the average test RMSPE and relative standard error (SE) for the variational inference method (VI) of Graves (2011); the probabilistic backpropagation (PBP) of Hernández-Lobato and Adams (2015); the MC dropout of Gal and Ghahramani (2016a); and the deep ensemble proposed by Lakshminarayanan et al. (2017). Extra-net$_1$ uses $T = 70$, while Extra-net$_2$ uses $T = 5$. The number of observations used for the split is reported as $M + n$, and the dimension of the input as d. In bold the lowest average RMSPE is highlighted.

Dataset	**(M+n)**	**d**	**VI**	**PBP**	**MC-Dropout**	**Deep Ens.**	**Extra-net$_1$**	**Extra-net$_2$**
Boston Housing	506	13	4.32±0.29	3.01±0.18	2.97±0.19	3.28±1.00	**2.80±0.15**	3.22±0.21
Concrete Strength	1030	8	7.19±0.12	5.67±0.09	5.23±0.12	6.03±0.58	5.26±0.15	**5.09±0.10**
Energy Efficiency	768	8	2.65±0.08	1.80±0.05	1.66±0.04	2.09±0.29	**0.59±0.01**	0.72±0.02
Kin8nm	8192	8	0.10±0.00	0.10+0.00	0.10±0.00	0.09±0.00	**0.08±0.00**	**0.08±0.00**
Naval Propulsion	11934	16	0.01±0.00	0.01±0.00	0.01±0.00	**0.00±0.00**	0.01±0.00	0.03±0.00
Power Plant	9568	4	4.33±0.04	4.12±0.03	**4.02±0.04**	4.11±0.17	4.12±0.05	4.24±0.04
Protein Structure	45730	9	4.84±0.03	4.73±0.01	4.36±0.01	4.71±0.06	**4.32±0.01**	4.36±0.02
Wine Quality Red	1599	11	0.65±0.01	0.64±0.01	**0.62±0.01**	0.64±0.04	0.63±0.01	0.64±0.01
Yacht Hydrodynamics	308	6	6.89±0.67	1.02±0.05	1.11±0.09	1.58±0.48	**0.72±0.06**	0.97±0.06
Year Protection MSD	515345	90	9.03±NA	8.88±NA	8.85±NA	8.89±NA	**8.84±NA**	8.97±NA

4.7 Conclusions

The definition of a robust methodology for the correct construction of prediction intervals for both deep and shallow neural networks is currently an open question that an increasing community of researchers and practitioners is focusing on. This chapter proposes a novel approach based on an ensemble of neural networks and compares its performance against state-of-the-art competitors found in the literature such as bootstrap methods and Monte Carlo dropout.

Our novel algorithm builds upon the work of Geurts et al. (2006) by extending the extremely randomized trees approach to ensembles of neural networks. The introduction of a Bernoulli mask allows for an additional randomization scheme in the prediction of the individual learners that ensures not only the correct construction of the prediction intervals, but also training the neural networks on the entire training set, better generalization performance due to randomized architecture structures, and accuracy gains due to an increase in the diversity among the members of the ensemble. The randomization across individual learners guarantees mutual independence across individual prediction models reducing the variance of the ensemble predictor by $1/T$, with T the number of models comprising the ensemble prediction.

The performance of the proposed algorithms is assessed in a comprehensive Monte Carlo

[58] If the predictions are rounded to the closest digit, or the floor operator is used, the obtained RMSE is 8.85.
[59] The deep ensemble proposed by Lakshminarayanan et al. (2017) is a novel algorithm that it is shown to consistently outperform classic bootstrap based approaches.

exercise. The simulation results show that the MC dropout, bootstrap approach, and extra-neural network returns prediction intervals with empirical coverage rates close to the significance level at which the intervals are constructed. Nevertheless, the extra-neural network is shown to outperform the competing models in most cases but more significantly for $\bar{\alpha} = 0.10$. The simulation results also show the robustness of the extra-neural network approach to the choice of the dropout rate, as opposed to the MC dropout approach. In fact, in order to return correct prediction intervals with the latter algorithm, it is necessary to fine-tune the dropout rate that minimizes the out-of-sample error.

Additionally, the experimental settings of Hernández-Lobato and Adams (2015) are used to further evaluate the proposed approach on real world datasets. The results suggest that the extra-neural network approach outperforms state-of-the-art deep learning algorithms in terms of out-of-sample RMSPE.

CHAPTER 5

Machine Learning the Carbon Footprint of Bitcoin Mining

Chapter Abstract

Building on an economic model of rational Bitcoin mining, we measure the carbon footprint of Bitcoin mining power consumption using feedforward neural networks. We find associated carbon footprints of 2.77, 16.08, and 14.99 $MtCO_2$ for 2017, 2018, and 2019 based on a novel bottom-up approach, which conform with recent estimates, lie within the economic model bounds while delivering much narrower prediction intervals, and yet raise alarming concerns, given recent evidence (e.g., from climate-weather integrated models). We demonstrate how machine learning methods can contribute to non-for-profit pressing societal issues, like global warming, where data complexity and availability can be overcome.

5.1 Introduction

Does Bitcoin mining contribute to climate change? Participation in the Bitcoin blockchain validation process[60] requires specialized hardware and vast amounts of electricity, translating into a significant carbon footprint. Mora et al. (2018) estimate that the 2017 carbon footprint of Bitcoin reached 69 million metric tons of CO_2-equivalent ($MtCO_2e$), forecasting a violation of the Paris COP21 UNFCCC Agreement[61] by 2040 due to Bitcoin's cumulative emissions alone. At the heart of the controversy sparked, with various contributions revising downward Mora et al.'s (2018) projections (e.g., Houy, 2019; Masanet et al., 2019 or Stoll et al., 2019), is the difficulty in measuring the power consumption of the Bitcoin mining network and the associated carbon emissions (De Vries, 2018, 2019, 2020). Bitcoin miners are globally geo-located, facing very different energy costs, and employ hardware with unknown energy intensities. To overcome the significant constraints in estimating the carbon emissions of daily power consumption associated with Bitcoin's blockchain, here we use machine learning (ML) methods, demonstrating their usefulness for pressing societal issues, like climate change.

A subset of ML methods, feedforward neural networks, are becoming increasingly popular due to their unrivaled performance in prediction tasks (LeCun et al., 2015). Feedforward neural networks, also called multilayer perceptrons (MLPs), have been developed since the mid-twentieth century, relying on joint advances from computer science, applied mathematics and information and probability theory. Their recent success stems from their theoretical ability to approximate unknown data generating processes (*Universal Approximation Theorem* and its variants), while handling large and complex datasets. They approximate or learn some unknown function of the data (or inputs) that generates an output, like the CO_2 emissions of Bitcoin network energy consumption, assuming that information 'feeds forward' from the input, through the unknown function, to the output.[62] They are called neural networks (NN) because they are composed of many functions connected in a chain, where each link is called a layer, each of which consists of an array of nodes (or units). By adding layers and nodes within each layer, feedforward NNs (or deep neural networks, DNN) can approximate functions of increasing complexity. CO_2 emissions are complex to forecast, but having a reliable general-purpose method to do so in a timely manner can inform progress towards keeping global temperatures from rising above 2°C, in addition to net-zero carbon emissions. Our main contribution is to provide a robust measure of the carbon footprint associated

[60] The revolutionary element of Bitcoin is the underlying 'blockchain' technology. Instead of a trusted third party, incentivized network participants validate transactions and ensure the integrity of the network via the decentralized administration of a data protocol (also called 'proof-of-work'). The distributed ledger protocol created has since then been called the 'first blockchain'.

[61] The Paris Agreement is an agreement within the United Nations Framework Convention on Climate Change (UNFCCC), dealing with greenhouse gas (GHG) emissions mitigation, adaptation, and finance, signed in 2016. It sets out a global framework to avoid dangerous climate change by limiting global warming to well below 2°C and pursuing efforts to limit it to 1.5°C. It also aims to strengthen countries' ability to deal with the impacts of climate change and support them in their efforts. The Paris Agreement is the first-ever universal, legally binding global climate change agreement, adopted at the Paris climate conference (COP21) in December 2015.

[62] This is in contrast to recurrent neural networks, where information is allowed to feed-back from the output to the model itself.

with producing increasingly popular cryptocurrencies, like Bitcoin (BTC), as well as of the uncertainty associated with that measure, conveying the likelihood of potentially alarming scenarios.

The carbon footprint of daily Bitcoin network electricity consumption obtains from multiplying the carbon intensity of the geo-located operating Bitcoin miners times their daily power consumption, which is then added across regions/countries (our novel *bottom-up* approach). To gauge the sensitivity of our *bottom-up* greenhouse gas emissions to uncertainty in carbon intensities, we report the emissions that obtain from adopting instead a *top-down* approach, the current standard in the literature. To estimate a realistic level of daily electricity consumption to produce Bitcoins, we first calculate a lower and an upper limit based on Hayes' (2017) economic model of rational Bitcoin mining decisions. The lower limit corresponds to the lowest marginal cost for mining Bitcoins, as defined by a scenario in which all miners use the most efficient available hardware. The upper limit obtains when the least efficient technology for mining Bitcoins is employed instead. Based on IPO filings of major hardware manufacturers, insights on mining facility operations, and mining pool compositions, our DNN adopts as target output the carbon footprint of the market share weighted average of the daily energy efficiency deployed by operating miners, identified by their IP addresses. Our estimated level of electricity consumption is thus a conservative one, closely tracking Hayes'(2017) lower limit. As inputs, our DNN admits a comprehensive range of factors previously found to drive Bitcoin prices in different currencies, like (i) fundamental factors advocated by monetary economics (e.g., its usage in trade, money supply, or price level); (ii) factors driving investors' interest in/attention to the crypto-currency (e.g., speculation or Bitcoin's role as safe haven); (iii) exchange rate hedging motives (see Kristoufek, 2015; Liu and Tsivinsky, 2018; McNally et al., 2018, or Jang and Lee, 2017), together with (iv) novel supply-side factors for both Bitcoin and ASIC mining chips producers, related to for-profit mining decisions, but excluding those employed in the construction of the upper and lower limits. Aggregated at the yearly frequency, we find Bitcoin mining energy consumption, ranging between 5.2 and 56.8 TWh in 2017, between 25.1 and 93.3 TWh in 2018, and between 27.1 and 91.1 TWh in 2019 according to Hayes' (2017) upper and lower bounds. Obtaining mean point estimates of daily power consumption within those economically meaningful limits provides substantial gains in accuracy relative to recent contributions in the literature, while externally validating our ML approach.[63]

Crucially, our novel approach also enables the construction of prediction intervals (PIs) around the estimated carbon footprint of Bitcoin mining, substantially narrowing down the associated uncertainty –currently measured by the difference between the carbon footprint of

[63] For daily estimates of the electricity consumption by the Bitcoin network, the University of Cambridge recently added a new source, the Cambridge Bitcoin Electricity Consumption Index (CBECI, https://www.cbeci.org), which is an alternative to the already existing Bitcoin Energy Consumption Index (BECI). De Vries (2020) reports that 'As per September 30, 2019, these two [respectively] estimated the network was consuming 73.1 to 78.3 terawatt-hours (TWh) of electrical energy annually. For a single Bitcoin transaction this translates to an electrical energy footprint roughly equal to the electrical energy consumption of a British household in two months.'

Hayes' (2017) upper and lower bounds, capturing the difference between the expected marginal revenue and the marginal cost of Bitcoin network operating miners. When aggregated at a yearly frequency, the corresponding CO_2 estimates [and associated 0.95 PIs] are, for the year 2017, 2.77 [1.98, 3.56] $MtCO_2e$; for 2018, 16.08 [14.19, 17.97] $MtCO_2e$; and for 2019, 14.99 [13.25, 16.73] $MtCO_2e$. To provide an order of magnitude, the Bitcoin mining estimated fossil fuels emissions for the year 2018 are higher than the annual levels of fossil fuel emissions of (i) the US states of Maine (15.6 $MtCO_2e$), New Hampshire (13.6 $MtCO_2e$), Rhode Island (10.1 $MtCO_2e$) or South Dakota (14.6 $MtCO_2e$), or of (ii) those of smaller countries, like Bolivia, Sudan or Lebanon, (*Global Carbon Atlas*).[64]

Relative to the aforementioned literature, the reported point estimates (and PIs) also represent a downward revision of the results reported by Mora et al. (2018), and are broadly in line with figures from Foteinis (2018), reporting global emissions for Bitcoin and Ethereum for 2017 of 43.9 $MtCO_2$ or from Stoll et al. (2019), reporting annual carbon emissions for Bitcoin mining in 2018 in the range 22.0 to 22.9 $MtCO_2$. Our estimates further revise downward the 2017 estimates provided by Houy (2019) or Dittmar and Praktiknjo (2019), reporting 15.5 $MtCO_2e$ for 2017, or those from Masanet et al. (2019), who report for 2017 an estimate of 15.7 $MtCO_2e$. What makes them nevertheless worrying is recent evidence, e.g. from integrated weather-climate models (CMIP6), feeding into the Sixth Assessment Report of the Intergovernmental Panel on Climate Change (IPCC) 2021, reported in Williams et al. (2020). According to them, global temperatures may rise as much as 5°C, prompting the recent global call to urgent policy measures by IMF's Chief Economist Gita Gopinath in Davos (Switzerland, 2020).

The topic is controversial, considering the growing interest of national governments on cryptocurrencies (e.g., China) and the possibility of issuing financial instruments solely on blockchain technologies (e.g., Bank of Australia and World Bank *bond-i*), while respecting the Paris Agreement. Before incentivizing the wide-scale adoption of blockchain technologies, the SCC associated with proof-of-work protocols, and their effect on rising global temperatures, need to be ascertained through better carbon intensity measurements. Besides the gains in accuracy, here we argue that ML methods present additional significant advantages for enabling timeless public decision making regarding pressing complex social issues, just as they do in private sector for-profit decisions, e.g. business analytics, new technology design, improvement or product adaptation and/or marketing. Being able to process bigger and increasingly complex data in raw form, ML techniques return tailored solutions in an automated manner. The significant 'entry cost' in terms of conceptual difficulty and computational time has significantly decreased over the last ten years, thanks to advancements in computational capacity, user-friendly software and increasing resources devoted to training and technology adoption, rendering their use commonplace.

[64] A measure of the magnitude of the economic problem can be obtained from adopting the *social cost of carbon* (SCC) estimate of 62 USD per metric ton of CO_2 equivalent (Interagency Working Group, IWG, 2016) in 2007 USD: yearly, the Bitcoin mining SCC reliably lies between [122, 760, 000; 220, 720, 000] USD for 2017; in [923, 800, 000; 1, 114, 140, 000] USD for 2018; and in [821, 500, 000; 1, 037, 260, 000] USD for 2019.

The rest of the chapter is organized as follows: Section 5.2 reports the novel methodology used in this chapter based on a bottom-up approach and the implementation of ML methods for measuring CO_2 emissions, briefly discussing the data used. Section 5.3 demonstrates the usefulness for predicting the carbon footprint associated with Bitcoin mining of our deep learning approach ('optimized ReLu DNN'), delivering substantially narrower bounds that increase the reliability of the provided estimates. We also show that our approach outperforms when benchmarked against state-of-the-art ML methods. Section 5.4 concludes.

5.2 CO_2 Emissions from Bitcoin Mining

There are three primary ways one can obtain BTCs, the most popular and widely accepted of the so-called cryptocurrencies: buy them outright, accept them in exchange, or produce them by 'mining'. Mining for Bitcoins requires computer hardware and software specifically designed to solve the cryptographic algorithm underlying the Bitcoin protocol. Such computational effort mainly consumes electricity.[65] Since at any point in time different miners operate hardware and software with varying levels of energy efficiency, measuring the overall network power consumption involved in Bitcoin production remains a challenge to date.[66] To overcome it, we propose a novel ML approach.

To estimate a realistic level of daily electricity consumption to produce Bitcoins based on a feedforward neural network, we first calculate a lower and an upper limit –based on Hayes (2017)– within which our mean predicted electricity consumption must 'travel' between the 01/01/2017 and the 01/01/ 2020, the considered period. The lower limit corresponds to the lowest marginal cost for mining Bitcoins, and is defined by a scenario in which all miners use the most efficient available hardware. The upper limit obtains when instead the least efficient technology for mining Bitcoins is employed, i.e. the break-even point of mining revenues and electricity costs. Obtaining mean point estimates of daily power consumption within those economically meaningful limits provides substantial gains in accuracy relative to recent contributions in the literature, while externally validating our ML approach.

Our feedforward deep neural network (DNN) is a supervised ML algorithm that adopts as target output the carbon emissions associated with the market share weighted average[67] of the daily energy efficiency deployed by operating miners[68]. Our daily level of electricity

[65] Each unit of mining effort has a fixed sunk cost involved in the purchase, transportation and installation of the mining hardware. De Vries (2018) reports different prices of available models of mining hardware, such as the Antminer S9. Mining effort also has a variable cost which is the direct expense of electricity consumption. See Hayes (2017) for further details.

[66] As an example, De Vries (2018) notes that 'A hashrate of 14 terahashes per second (TH/s) can either come from a single Antminer S9 running on just 1,372 W, or more than half a million PlayStation-3 devices running on 40 MW (as a single PlayStation-3 device has a hashrate of 21 megahashesper second and a power use of 60 W).'

[67] The market shares are computed in terms of either computational power or revenues, and are available from IPO filings disclosed in 2018 by Bitmain, and in 2019 by Canaan, retrieved from Bloomberg terminals.

[68] We obtain the computational power (usually provided in terahashes per second, TH/s) and the electricity consumed (in Watts per second, W/s) by ASIC chips used for Bitcoin mining from *AsicIndex*. Only mining chips that perform the *SHA-256* algorithm are considered.

consumption is a conservative one in that it follows the approach of the lower limit, and is based on the anticipated energy efficiency of the network, on hardware sales and on auxiliary losses[69]. As inputs, our DNN admits a comprehensive range of factors previously found to drive Bitcoin prices in different currencies, like (i) standard fundamental factors advocated by monetary economics and the quantity theory of money (e.g., its usage in trade, money supply or price level); (ii) factors driving investors' interest in/attention to the cryptocurrency, like speculation or the role of Bitcoin as a safe haven; (iii) exchange rate hedging motives, like the tight connection between the USD and the CNY markets (see Kristoufek, 2015; Liu and Tsivinsky, 2018; McNally et al., 2018, or Jang and Lee, 2017)[70], together with (iv) novel supply-side factors for both Bitcoin and ASIC mining chips producers, related to for-profit mining decisions, but excluding those employed in the construction of the upper and lower limits.[71] The carbon intensity associated with Bitcoin mining obtains then from multiplying the estimated electricity consumption by the average emission factor of power generation. More specifically, the present chapter improves over the current literature focusing on CO_2 emission and Bitcoin mining as it proposes not only the standard top-down approach to define the target carbon footprint of the Bitcoin network, but also a bottom-up approach that uses the market information disclosed in the IPO filings of Bitmain, Canaan, and Ebang that allows us to determine the country-specific CO_2 emissions from which the aggregate level is computed. Crucially, our novel approach also enables the construction of prediction intervals around the estimated carbon footprint of Bitcoin mining, substantially narrowing down the associated uncertainty, as currently measured in the literature by the difference between the carbon footprint of the upper and lower bounds. The latter broadly represent, respectively, the expected marginal revenue and marginal cost of Bitcoin network operating miners (Hayes, 2017; Stoll et al., 2019).

5.2.1 Power Bounds in Bitcoin Production

Hayes (2017) argues that Bitcoin production resembles a competitive market, where risk-neutral rational miners will produce until their marginal costs equal the value of their expected marginal products. To produce Bitcoins, a miner directs computational effort at solving a difficult cryptologic 'puzzle' in competition with other miners in the network, to confirm and validate transactions. And computational effort mainly consumes electrical power, measured in Watts, W.[72] The marginal cost (MC) of producing Bitcoins per day (in USD/day) depends on the cost of electricity (price p_e in USD per kWh, or $10^{-3} \times p_e$ in USD per Wh) and the energy efficiency of mining (denoted by e and measured in W per unit of 'mining effort', or

[69] These are energy losses associated with cooling and investment in new IT equipment. They are computed on the basis of the methodology employed to construct the Cambridge Bitcoin Electricity Consumption Index (CBECI) with data reported by Stoll et al. (2019).

[70] These are collected at/or converted into daily frequencies from Bloomberg, from the Federal Reserve Bank of St. Luis, from Blockchain.com, from asic-dex.com, and from the IPO filings of Canaan and Bitmain.

[71] The Bitcoin exchange rates with other cryptocurrencies such as Ethereum, Ripple, and Litecoin are not studied as these 'altcoins' were issued from 2018 onward.

[72] Recall that one Watt equals one Joule per second, i.e., 1 W=1 J/s.

'hashing power' ρ)[73]:

$$\underset{\text{[USD/day per } \rho\text{=1,000GH/s]}}{MC} = (10^{-3} \times p_e \cdot 24 \cdot e) \cdot \left(\frac{1000GH}{1000}\right) \tag{5.1}$$

In return for their work of validating the blockchain, miners are rewarded with a block of 'coins', or 'block reward' (measured in BTC per block, β)[74]. Per day, miners can then expect to earn an amount of Bitcoins (BTC/day), or expected marginal product ($\mathbb{E}$MP), the value of which depends on the market price of bitcoin (p_b in USD per BTC), the block reward β, the transaction fees f, the hashing power ρ employed by a miner (normalized at ρ =1,000 GH/s =1 TH/s, for conformity with the MC units), and the 'difficulty' of mining (denoted by δ) which captures how much aggregate effort other operating miners are putting[75]:

$$\underbrace{\underbrace{p_b}_{\text{[USD/BTC]}} \cdot \underbrace{\mathbb{E}MP}_{\text{[BTC/day per } \rho\text{=1TH/s]}}}_{\text{Value of Expected MP}} = p_b \cdot \left[\underbrace{\left(\frac{1}{\delta \cdot 2^{32}}\right)}_{\text{[Reward probability]}} \underbrace{(\beta + f) \cdot \rho \cdot (24 \cdot 3,600)}_{\text{[Daily reward per unit of effort } \rho\text{]}}\right] \tag{5.2}$$

where $s = 3,600$ is the number of seconds in one hour, $h = 24$ is the number of hours in a day, and 2^{-32} is the normalized probability of a single hash solving a block, given that the mining algorithm is the *SHA-256* algorithm.

Given the market price of Bitcoin p_b, a rational miner would produce Bitcoins until when $MC = p_b \cdot \mathbb{E}MP$ if mining for Bitcoins is competitive. Since the actual energy efficiency e of the Bitcoin network miners is unknown, the theoretical relationship $p_b = MC/\mathbb{E}MP$ can be used to obtain the break-even level of energy efficiency $\underline{e}$ below which the marginal cost of mining is above the market value of the marginal product, $e \leq \underline{e} \Longrightarrow MC(e) \geq MC(\underline{e}) = p_b \cdot \mathbb{E}MP$,

[73] Mining effort or 'hashing power' or 'hashrate' is measured in gigahashes per second (GH/s), and refers to the computational effort applied by miners to obtain bitcoins over a given time interval, typically one day. The hashrate, or number of hashes per second can be thought of as somewhat analogous to the cycles per second (hertz) of computer processors: the higher the hashrate, the more likely it is to successfully mine bitcoins per day. See Hayes (2017).

[74] When analyzing the reward obtained from mining, it is important to consider the phenomenon of *halvening* (Bitcoin halving) where the reward from mining Bitcoins is halved. *Halvening* occurs every 210,000 blocks (every four years). Within our sample, the last *halvening* happened in 09/07/2016 with the mining revenue halved from 2,396,656 USD to 1,208,034 USD. The *halvening* is an important event not only for determining the Bitcoin price (reduction of Bitcoin supply, with unchanged demand) and the break-even energy efficiency level of mining production, but also because it produces a *jump* or discontinuity in the historical observations at hand. The time interval considered 2017 - 2019 ensures that there are no observed *halvening*s. Starting from 09/07/2016, the *block reward* is 12.5 Bitcoin per block.

[75] The 'difficulty' of mining refers to the difficulty level of the algorithm when mining is undertaken. It specifies how hard in terms of computational effort is to find a bitcoin during a given time interval, and is therefore measured in gigahashes per 'block' of bitcoins, GH/block. The bitcoin network automatically adjusts the difficulty variable so that one block of bitcoins is found, on average, every ten minutes. As more aggregate computational effort is added to mining bitcoins, the time between blocks will tend to decrease below ten minutes, and the network will automatically adjust the difficulty upwards to maintain the ten minutes interval. And conversely, is less aggregate computational effort is added, adjusting the difficulty downwards.

driving rational miners out of business. Hence:

$$\underset{\text{[J/GH per } \rho=1{,}000\text{GH/s]}}{\underline{e}} = p_b \cdot \left(\frac{(\beta + f) \cdot \rho}{\delta \cdot 2^{32}} \right) (24 \cdot 3{,}600) \left[(10^{-3} \times p_e \cdot 24) \right]^{-1} \tag{5.3}$$

denotes the break-even daily energy efficiency production of Bitcoins, which characterizes the *upper limit of daily electricity consumption* $\overline{E}$ of the Bitcoin network when multiplied by the overall network hash rate H (measured in hashes per second, H/s, corresponding to 10^{-12} per 1TH/s) and the power usage effectiveness (PUE) of mining hardware, capturing the auxiliary energy efficiency losses due, for example, to cooling systems:

$$\underset{\text{[W per day, per TH/s]}}{\overline{E}} = \underline{e} \cdot H \times 10^{-12} \times \overline{PUE} \tag{5.4}$$

Instead of an average PUE of 1.05 as in Stoll et al. (2019), we consider a value of $1.10 \equiv \overline{PUE}$, e.g. just as the Cambridge Bitcoin Electricity Consumption Index does when computing the upper limit.

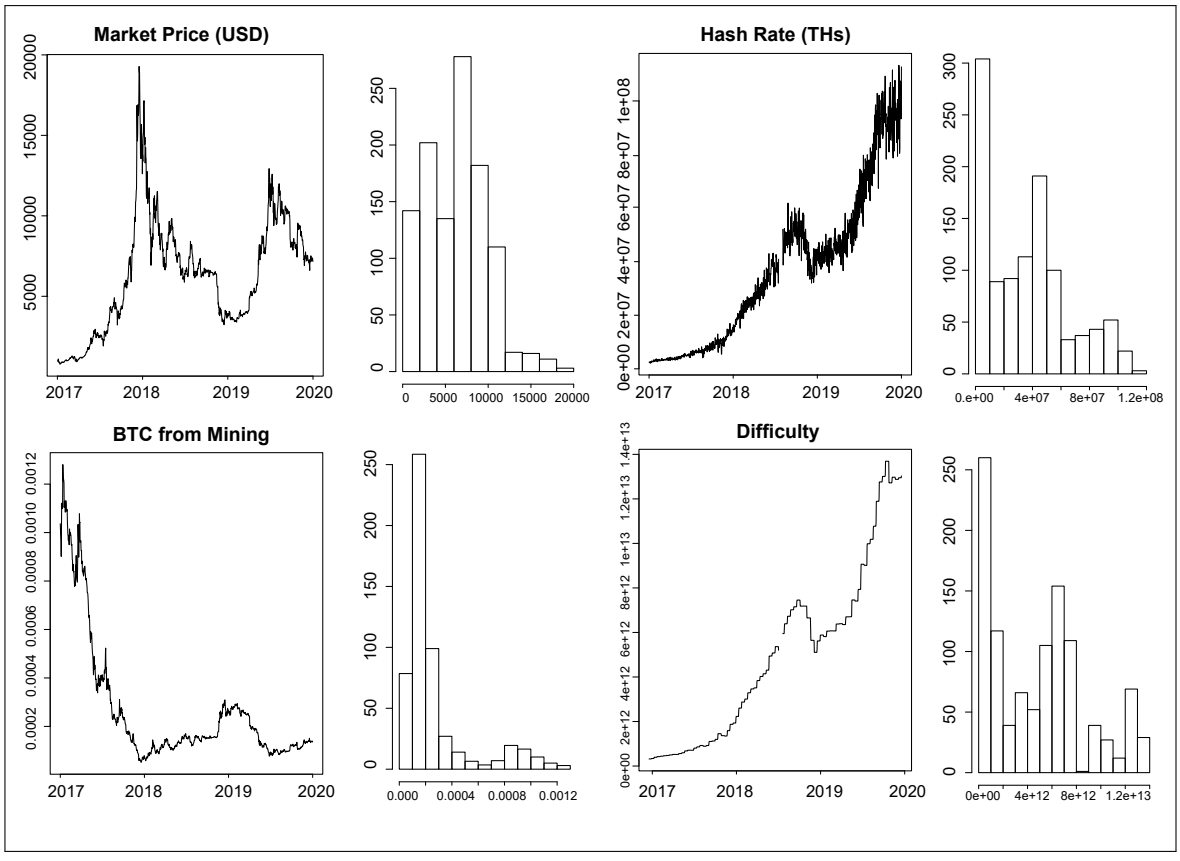


Figure 5.1: The Figure reports the difficulty in terms of hashing power employed by the network miners, the hash rate in terms of estimated number of tera hashes per second the Bitcoin network is performing, the average USD market price across major bitcoin exchanges, and the mining reward in terms of Bitcoin

Daily data for the Bitcoin network *difficulty* δ and network *hash rate* H are retrieved

using the publicly available API from *blockchain.com,*[76] and reported together with their distributions, in Figure 5.1, as well as for the daily Bitcoin price p_b and the daily value in USD of the number of Bitcoins obtained by the overall network from mining (BTC/USD), as defined in equation (5.2).[77] Notice that although the network *hash rate* and the network *difficulty* are strongly positively correlated, they nevertheless correspond to two different variables relevant to Bitcoin mining.

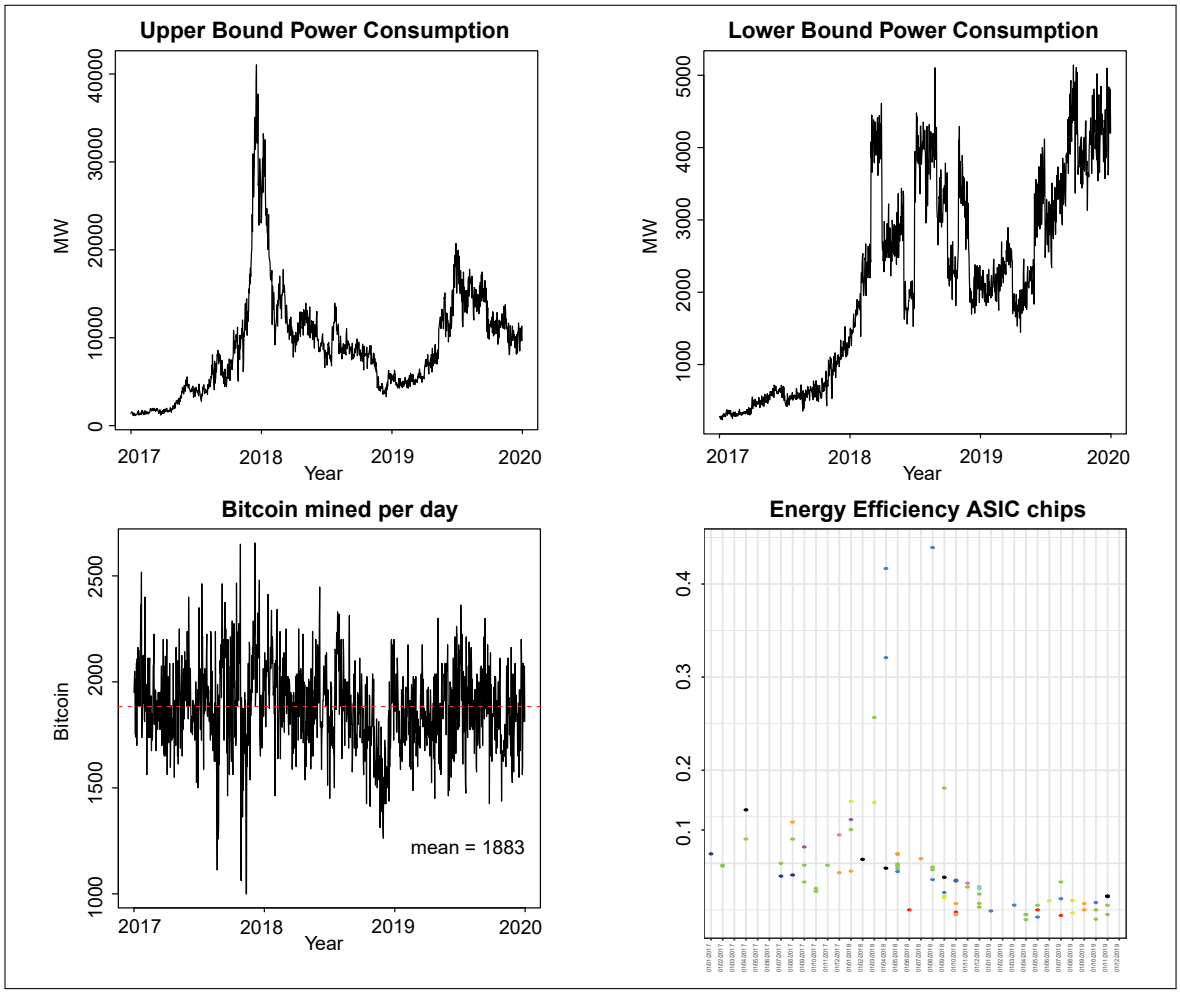


Figure 5.2: The Figure reports the number of Bitcoins mined per day, the upper and lower bounds of the energy consumption associated with Bitcoin mining, and the energy efficiency in terms of J/Gh of the ASIC mining chips that use the SHA-256 Algorithm

Similarly, it is possible to define the *lower limit of daily electricity consumption* $\underline{E}$ of the

[76] Since for the time interval 18/07/2018 - 03/08/2018 those network statistics are missing, they are imputed using the MissForest algorithm (Stekhoven, 2013), with a maximum number of trees to be grown in each forest equal to 500, a maximum number of nodes per tree equal to 100, and a maximum number of iterations of 50. The MissForest algorithm is agnostic about the distribution of the variables, estimating the missing values by fitting a random forest trained on the observed values. The Out-Of-Bag (OOB) estimates of the imputation error in terms of Normalized Root Mean Squared Error (NRMSE) is 0.04831 and convergence is achieved.

[77] For example by the end of 2019, the lower left panel of Figure 5.1 reports the USD value of the number of bitcoins one can expect per day (in BTC/USD) to be approximately 0.0003. We can obtain the actual number from equation (5.2), when employing ρ =1,000 GH/s of mining effort with a difficulty $\delta = 4 \times 10^{12}$ (lower right panel of Figure 5.1) at a price $p_b = 4,890$ (upper left panel of Figure 5.1): $4890 \cdot (\frac{12.5 \cdot 1000}{2^{32} \cdot 4 \times 10^{12}}) \cdot 24 \cdot 3600 = 0.0003074$ BTC/USD per day.

Bitcoin network, assuming that all miners operate instead with the most energy efficient $\overline{e}$ hardware with no auxiliary energy efficiency losses, $\underline{PUE} = 1$:

$$\underset{\text{[W per day, per TH/s]}}{\underline{E}} = \overline{e} \cdot H \times 10^{-12} \times 24 \times \underline{PUE} \tag{5.5}$$

To date, the most energy efficient dedicated computer hardware embeds application specific integrated circuit (ASIC) chips. Monthly data about the mining chips' daily efficiency, measured (in J/GH) as the ratio between the energy used by the ASIC chip (in Joules, J) and the number of iterations performed by the *SHA-256* algorithm (in gigahashes per second, GH/s), for different mining rigs is displayed in Figure 5.2 lower right panel, between 01/01/2017 and 01/01/2020.[78] $\overline{e}$ then corresponds to the lowest monthly energy efficiency of ASIC chips, which as time passes tends to decrease –except for a few outliers– due to an increase in the network hash rate and thus in the difficulty in producing new Bitcoins.

Figure 5.2 reports the number of Bitcoins mined per day by the network (i.e. the average $\mathbb{E}MP$ in equation (5.2), excluding the Bitcoin price p_b)[79], and the associated upper $\overline{E}$ and lower $\underline{E}$ limits of daily electricity consumption obtained from equations (5.4) and (5.5) after multiplying them by 10^{-6} (to convert them into mega Watts, MW), respectively. Although the upper limit of daily power consumption is more volatile as it follows the market price of Bitcoin, the lower limit is more stable, being defined by hardware efficiency and network hash rate. The difference between the upper and lower limits provides a sense of the uncertainty associated with the actual daily hardware efficiency in electricity consumption deployed by the Bitcoin production network of miners. The annual electricity consumption corresponding to the lower and upper bounds $\underline{E}$ and $\overline{E}$ is obtained by summing the daily electricity consumption over the year of interest: for 2017, it ranges between 5.2 and 56.8 TWh, for 2018 between 25.1 and 93.3 TWh, and for 2019, between 27.1 and 91.1 TWh.

Notice from Figure 5.2 the decreasing gap between $\overline{E}$ and $\underline{E}$, converging to a point of almost equality in 2019: miners with less efficient ASIC chips were then mining at a loss as a result of the significant decrease in Bitcoin prices that can be observed in the upper left panel of Figure 5.1. One would expect the same narrowing in the difference between the two daily limits as we get closer to 05/2020 (outside of our data window), when the *halvening* of the 'block reward' happened. By then, miners will have had to run twice the number of computations to mine the same amount of Bitcoins, doubling their electricity usage. This will reduce the break-even level of energy efficiency $\underline{e}$, reducing $\overline{E}$, until when new and more efficient ASIC chips are introduced.

In contrast to Stoll et al. (2019), we compute electricity prices, p_e, as a weighted average of the annual electricity prices in the countries were Bitcoin miners are located, using as weights

[78] The data can be retrieved online from https://asic-dex.com.

[79] Notice that the reported average number of Bitcoins mined per day over the period is similar to the one that obtains instead from the supply side: dividing equation (5.1) by the Bitcoin price p_b, we get $\frac{MC}{p_b} \simeq$ 1,800 BTC/day.

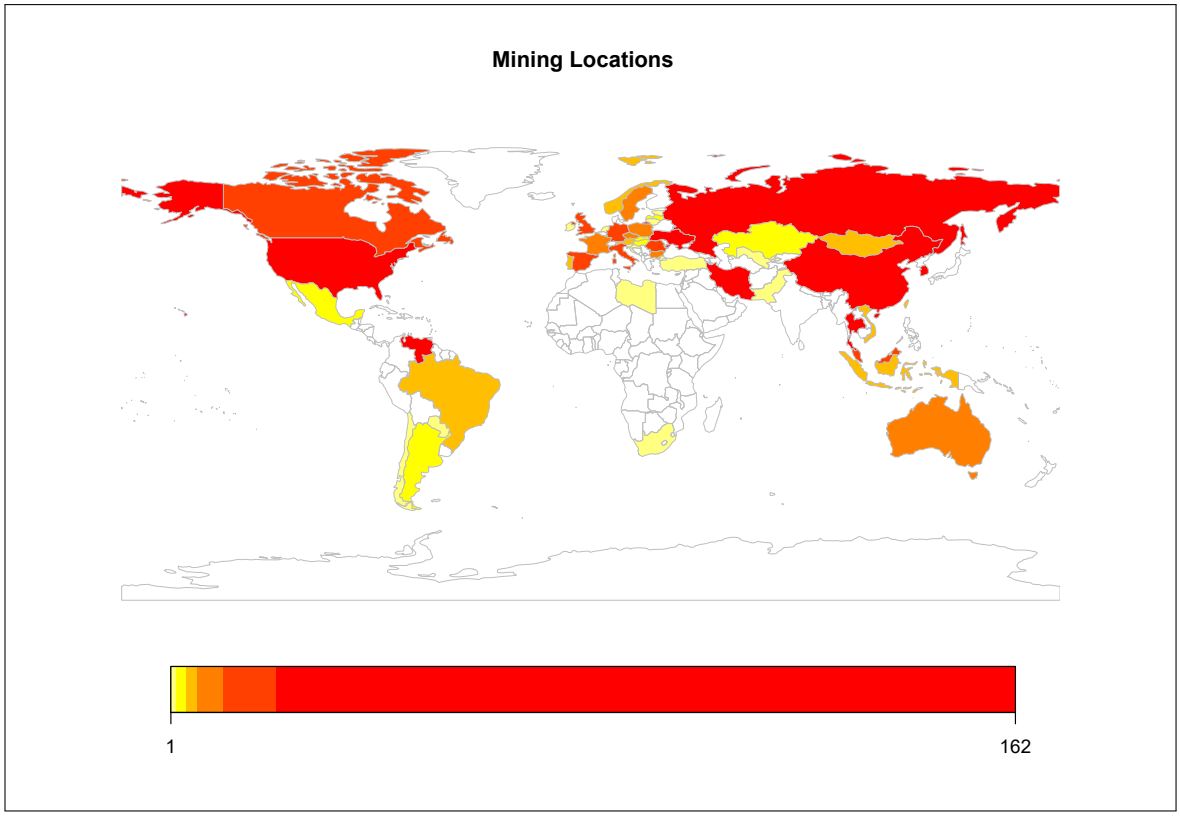


Figure 5.3: Location Bitcoin Miners 31/01/2020

the share of miners located in each country. We exploit the Internet of Things (IoT)-search engine *Shodan* to locate the geographic area of the Bitcoin miners IP addresses over the period examined[80].

Figure 5.3 reports the countries with the highest number of miners: Venezuela (91), China (162), Russia (158), Iran (122), USA (75). Venezuela, Iran, Russia and (some regions of) China are the countries with the lowest electricity prices in the World (in USD per kWh). We collect historical data on electricity prices for the USA, China, and Russia from Bloomberg Terminal up to 2018, and the electricity prices for 2019 from *GlobalPetrolPrices.com*. Figure 5.4 reports the evolution of the yearly electricity prices for different usages (residential, industrial and other) in China, the United States, and Russia[81].

For Venezuela and Iran, it was not possible to collect historical prices: since electricity prices (approximated to two digits) are generally constant over a three-year horizon, we apply the 2019 electricity price over the three-year time window examined. The household electricity price in Iran is 0.008 USD/kWh; for Venezuela, the Business electricity price is 0.128 USD/kWh (1.283 VEF/kWh). Figure 5.4 reports the employed electricity price p_e, computed as a weighted average of the electricity prices in the United States, China, Russia, Venezuela,

[80] Being *antminer* the primary tool for Bitcoin mining, by mapping the instances *Digest real="antMiner Configuration"* we were able to map the IP addresses of the Bitcoin miners.

[81] When available and clearly indicated, we only consider the residential electricity price. When unavailable, or unclear (e.g., China), we compute the average of the electricity prices corresponding to the different levels of usage.

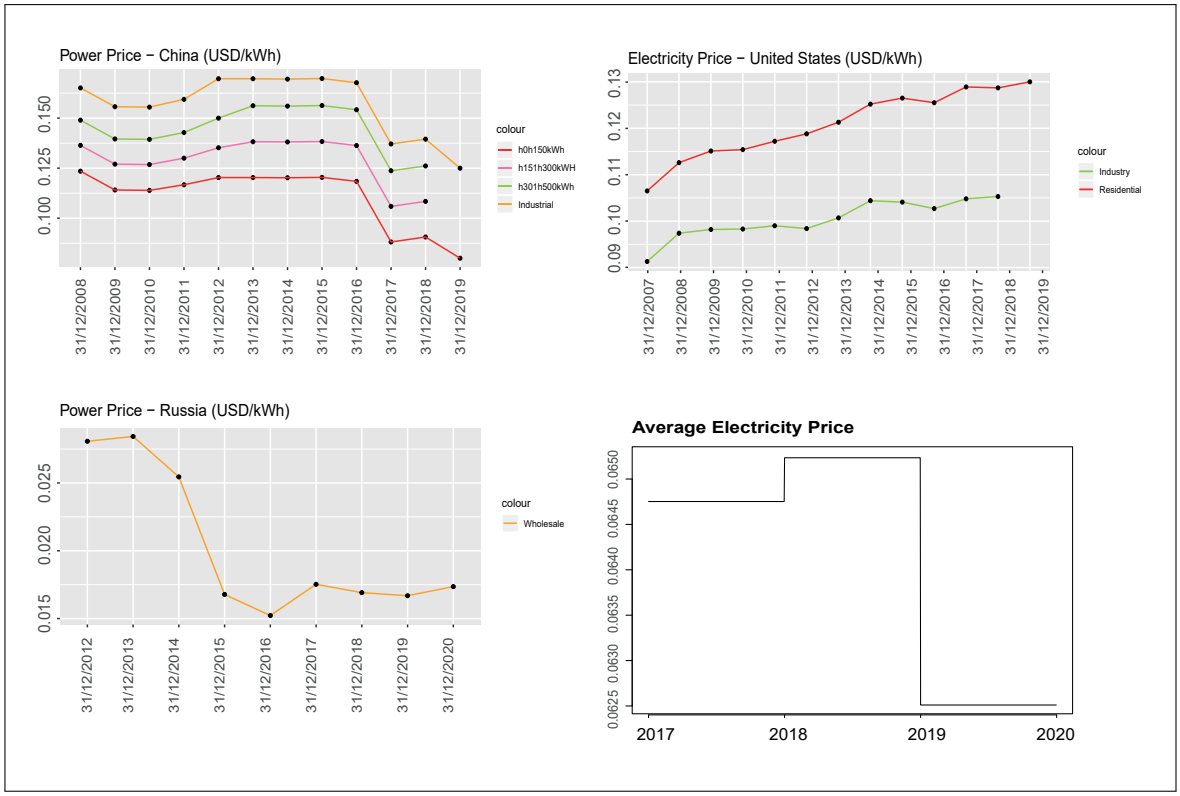


Figure 5.4: The Figure reports the energy prices (USD/kWh) for the countries United States, China, and Russia, and the weighted average of the energy prices (USD/kWh) across the countries United States, China, Russia, Venezuela, and Iran.

and Iran, where the weights are determined by the proportion of Antminer IP addresses of Bitcoin miners located in those countries.[82]

5.2.2 The Carbon Footprint of Power Bounds in Bitcoin Production

We compute the CO_2 upper ($\overline{CO_2}$) and lower ($\underline{CO_2}$) limits of the Bitcoin network daily emissions (measured in ktCO_2e), associated with the daily electricity consumption upper and lower limits, $\overline{E}$ and $\underline{E}$, from equations (5.4) and (5.5) respectively, as follows:

$$\overline{CO_2} = \overline{E} \times 10^{-3} \cdot I \times 10^{-6} + CO_2^{rw} \tag{5.6}$$

$$\underline{CO_2} = \underline{E} \times 10^{-3} \cdot I \times 10^{-6} + CO_2^{rw} \tag{5.7}$$

where I is the average emission factor, or carbon intensity, of power generation (measured in kgCO_2 per kWh)[83], which obtains from weighting the C country-specific emission factors, I_c, by the computing power share, s_c, of Bitcoin miners' IP addresses located in each country

[82] 39% of the IP addresses operating in the Bitcoin network are attributed to the remaining 44 countries.

[83] Notice that the expressions in (5.6) and (5.7) are in ktCO_2 units per day, while $\overline{E}$ and $\underline{E}$ are in Watts per day (per unit of mining effort, 1GH/s) and I is in kgCO_2 per kWh. To conform, we need to multiply $\overline{E}$ and $\underline{E}$ by 10^{-3} (KWh per Watts) per day, and I by 24 (hours per day), resulting in a product that will then be in units of kgCO_2 per day. Multiplying then by 10^{-6} we obtain ktCO_2 per day. After simplifying, expressions (5.6) and (5.7) obtain as reported.

c, $I = \sum_{c=1}^{C} s_c I_c$. CO_2^{rw} captures the approximate emissions associated with the annual Bitcoin network overall disposal of hardware employed in mining Bitcoins.

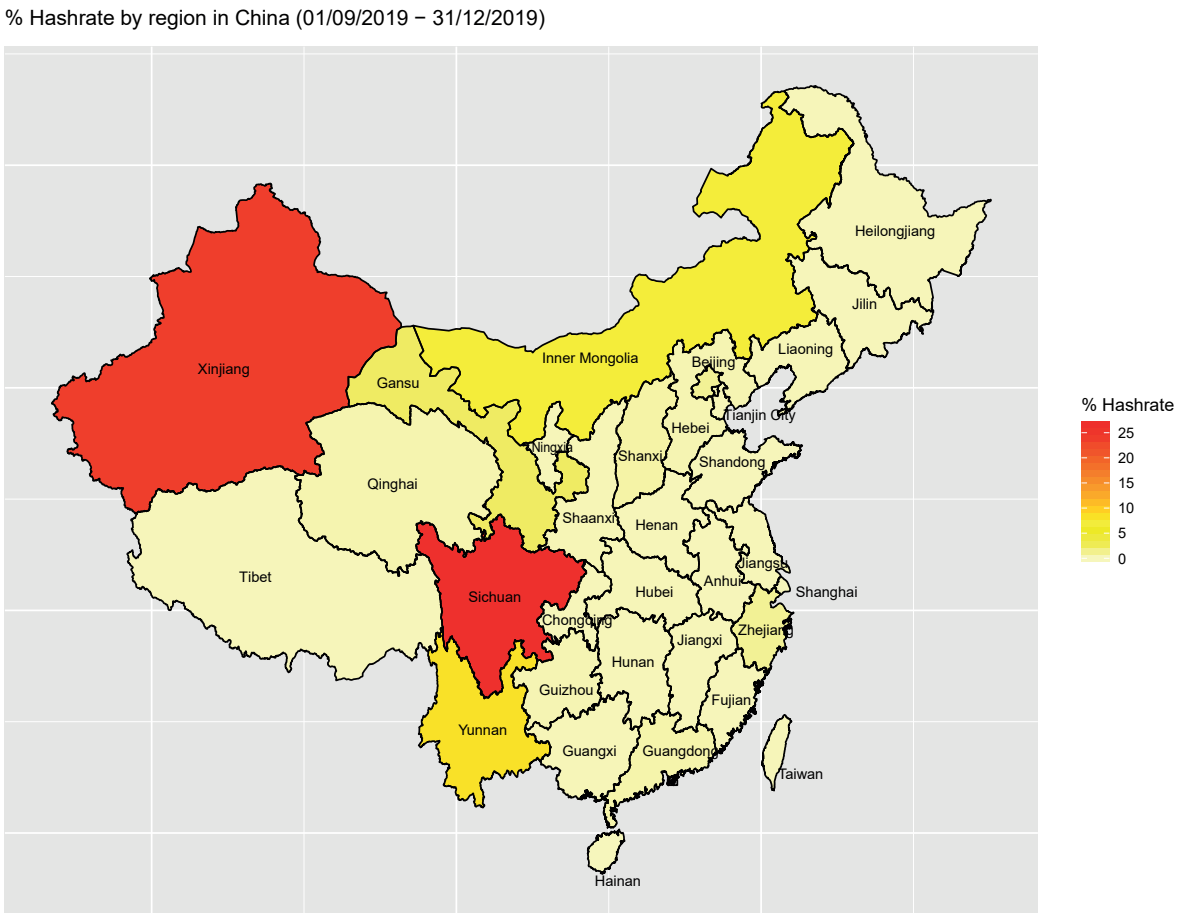


Figure 5.5: Distribution of Bitcoin miners within the Chinese borders: %.Hashrate by province in China.

Following De Vries (2019), a daily value of $CO_2^{rw} = 0.0087$ ktCO$_2$ obtains[84]. In the reminder of the chapter, we refer to equations (5.6) and (5.7) as implementing a *top-down* approach, the current standard in the literature. According to the methodology reported in Volume 2 of the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, when computing the emission of greenhouse gas from stationary sources (Electricity and Power consumption), the source consumption must be multiplied by the corresponding emission factor (IPCC, 2006). Since Bitcoin network mining spans many different countries, the contribution of the miners located in each country to the overall network hash rate is needed to

[84] As of October 2018, 3.91M antminer S9 machines were needed to produce the overall Bitcoin network total of 54.7 exahashes per second (at its peak), with each antminer producing an output of 14TH/s. Since each antminer S9 machine weights 4.2Kg and lasts of average for 1.5 years, after which it needs to be replaced/disposed of, a total of 16,442 metric tons of weight of mining displays will be disposed of every 1.5 years, or 10,948 metric tons per year. To convert these into CO_2 emissions, the Climate Institute reports that for every ton of cathode-ray tube (CRT) display products manufactured, 2.9 metric tons of carbon were released. When properly recycled, only 10 percent of greenhouse gas emissions are released. Therefore, a total of 10,948 metric tons of Bitcoin e-waste times 2.9 metric tons of CO_2 per ton of weight, yields 31,749.2 metric tons of greenhouse gas emissions per year, which when properly recycled, results in only 10%, or 3,174.9 metric tons of CO_2 released per year. Dividing by 365 days, we obtain a daily figure of $CO_2^{rw} = 0.0087$ ktCO$_2$ per day, per TH/s.

construct country-specific upper and lower limits of electricity consumption that can then be aggregated into a world total, i.e. a *bottom-up* approach. But because miners are particularly secretive about their locations, a country-specific break-even upper bound has been difficult to obtain. For comparison with results reported in the literature, emission levels corresponding to Hayes' (2017) upper and lower bounds computed from a top-down approach are reported (in black) in Figure 5.7.

Because one of the biggest sources of uncertainty in computing Bitcoin network mining CO_2 emissions is the translation of the overall network energy consumption into carbon emissions, we exploit the information provided by the Cambridge Bitcoin Electricity Consumption Index (CBECI) and the IoT-search engine *Shodan.io* to obtain 'clean energy' country-specific emission factors, I_c^e.[85] Exploiting data on the distribution of the overall network hashrate within countries[86], we were able to identify (to some extent) the heterogeneous sources of electricity employed to mine Bitcoins when and where regional emission factors are available[87]. Figures 5.5 and 5.6 report the distribution of Bitcoin miners within China by province, and within the US, by state, respectively.

Focusing on China, the Economist Intelligence Group (2018) notes that as of 2016, provinces in the eastern and northern parts of China essentially employ coal-based energy sources, due to the absence of precipitation (making hydro-power unprofitable) and the difficulty of installing wind-power generation in these mountainous regions. Shanghai and Tianjin provinces produced almost 100% of their electricity from non-renewable thermal power, while Inner Mongolia and Xinjiang almost 90%. At the other extreme, Yunnan and Sichuan provinces produced 83% and 87% of their electricity from hydro-power sources, respectively, having a surplus of hydro-power during the wet season; Tibet generated 97% of its electricity from clean energy sources and Quinghai province is the biggest producer of solar energy.[88]

[85] The US Energy Information Administration (EIA) considers biomass, hydro, solar and wind based electricity sources to be carbon neutral, i.e. associated with a zero carbon intensity.

[86] Mapping the instances Digetreal="antMiner Configuration" in the IoT-search engine Shodan, we obtained the Bitcoin network hashrate distribution for China and the US as of 20/08/2020, reported in Figures 5.5 and 5.6.

[87] For example, they are not available for Russia, Venezuela or Iran, for which we assume that a homogeneous source of electricity is available, and well captured by their reported country-specific emission factors, $I_c^e = I_c = \{I_{RU}, I_{VE}, I_{IR}\}$.

[88] Although Bendiksen and Gibbons (2019) observe that Chinese miners relocate during the rain season (May to September) to hydro-power surplus provinces such as Sichuan, Yunnan, and Guizhou, from low-cost coal based energy provinces such as Xinjiang and Inner Mongolia, we ignore such seasonal relocations for two reasons. Firstly, it is not yet fully understood how relocation costs influence miners' seasonal migration. Secondly, reliable measures of such relocation costs are needed to compute Hayes' (2017) economic upper bound.

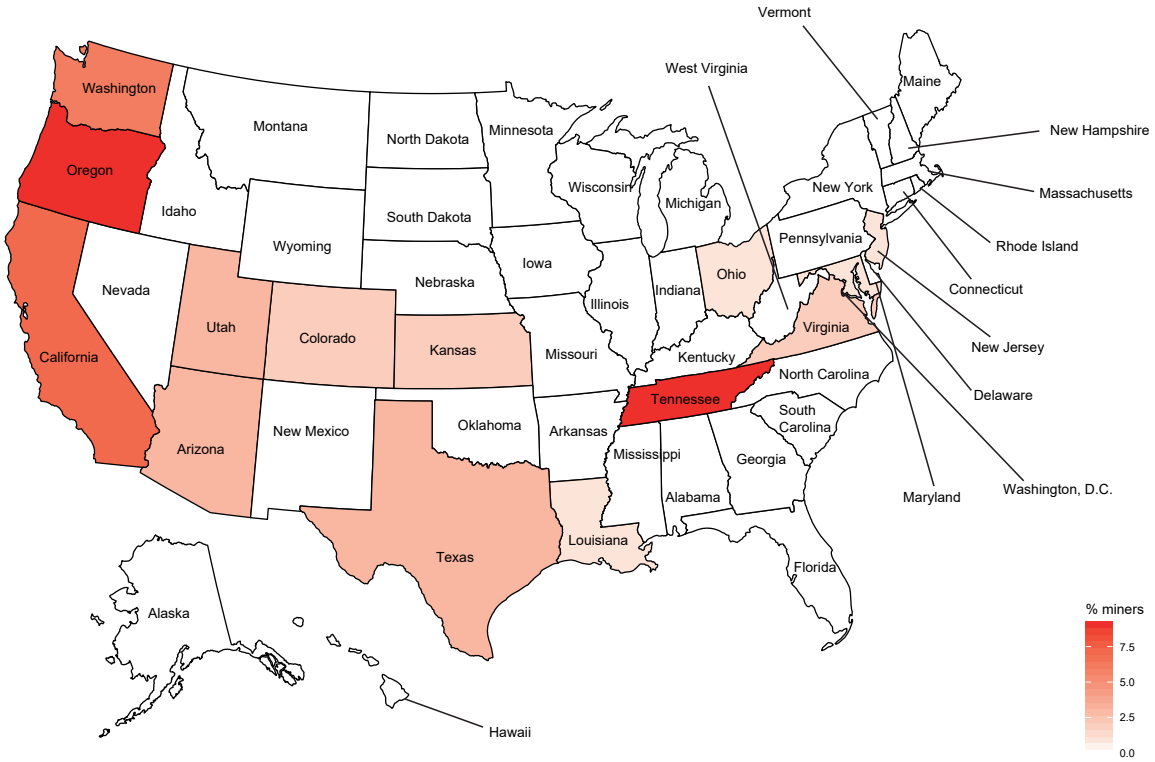


Figure 5.6: Distribution of Bitcoin miners within the US borders: % Hashrate by U.S. states.

Turning now to the U.S., Figure 5.6 reports Tenneesse, with 0.18, California, with 0.14, Oregon, with 0.18, and Washington state, with 0.12, as those states with the highest concentration of the overall US mining activity. Coupled with the report by Willms (2019) in Bitcoin Magazine, the exact location of mining centers can be identified to better understand the source of electricity used for mining, e.g. focusing on Washington state, the Shodan IoT-search engine locates Bitcoin miners in the cities of East Wentchee and Everett, where Willms (2019) reports that Salcido Enterprise has three mining centers that use inexpensive hydroelectric power from dams in the Columbia River. Similarly, Willms (2019) reports that Bitmain invested 20 million USD for the construction of five mining buildings equipped with 1620 antMiners. Focusing on California, we locate Bitcoin miners close to the city of Los Angeles and thus close to the California's Mojave District where Willms (2019) reports that Plouton Mining is investing in mining using solar power. Focusing on Oregon, we locate a high concentration of Bitcoin miners in proximity to Portland, close to the Columbia River. We assume that also in this state most of the mining activity is hydro-power based. Finally, a high concentration of Bitcoin miners in the cities of Knoxville and Chattanooga, where there are the biggest dams in the state of Tennessee, the Norris and Chickamauga dams, leads our presumption that also in the state of Tennessee bitcoin miners use clean energy sources.

With the collected data, we can examine the robustness of Hayes' (2017) greenhouse emission upper and lower bounds to Bitcoin miners usage of 'clean energy' sources, by computing them under the two scenarios.[89] In the first case we do not account for 'clean en-

[89] We follow Stoll et al. (2019), adding the US to China for the within-country analysis of energy sources. But we impose a stronger assumption than they do because when a state/province has access to renewable

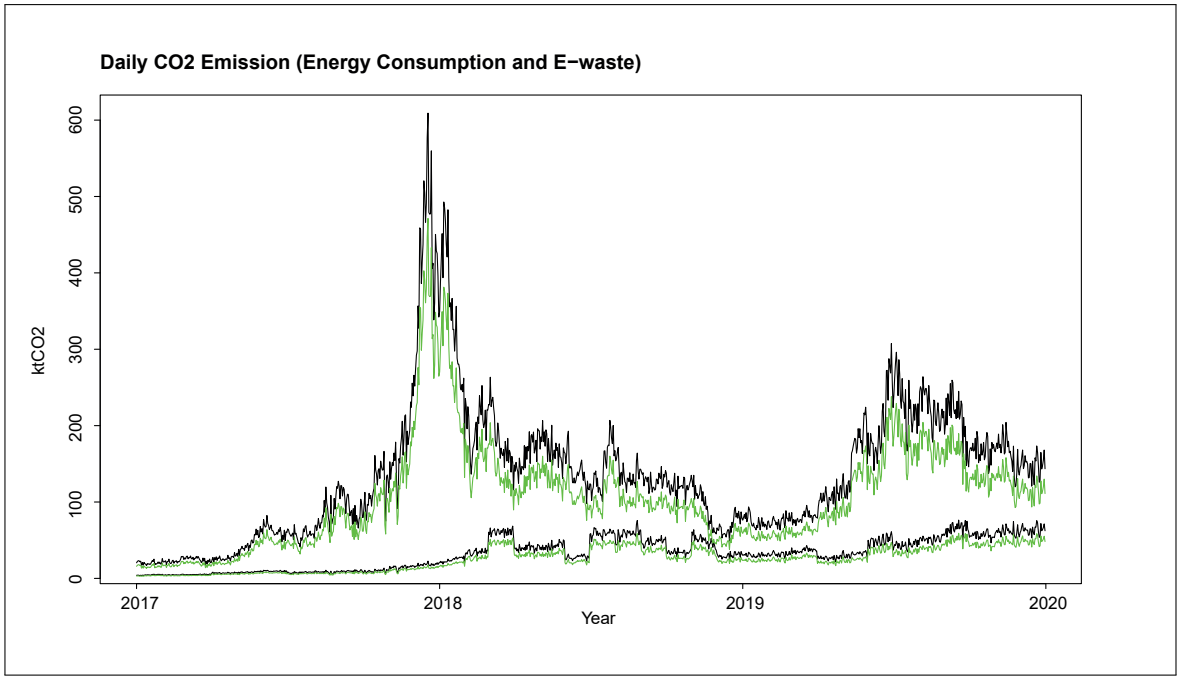


Figure 5.7: The Figure reports the lower and upper bound for the daily CO_2 estimates. In green the daily estimate when I^e is considered, in black when I applies.

ergy' mining, obtaining a weighted average carbon intensity of $I = 0.6183$, from country-level electricity emission factors of 0.97463 for China, 0.63111 for Iran, 0.5132 for Russia, 0.5471 for the United States, and of 0.2081 for Venezuela (Brander et al., 2011). In the second case, we employ a more conservative conversion factor, $I^e = \sum_{c=1}^{C} s_c[\sum_{d=1}^{D_c} s_d I_d]$, which obtains from decomposing the C country-specific emission factors, $I_c = \sum_{d=1}^{D_c} s_d I_d$, by region d where Bitcoin miners' IP addresses are located within each country c. We separate regions within each country c into two groups: one of $\{D_c - d'\}$ regions where 100% clean surplus electricity is available, with a carbon intensity of $I_d = 0$; and a complementary one of $\{d'\}$ regions where electricity generation is assumed to be equal to country c emission factor, $I_d = I_c$. Using the weights provided by CBECI we obtain $I^e_{China} = \{[100-(8.34+26.5+2.53+0.4)]/100\} \times 0.97463 + [(8.34+26.5+2.53+0.4)/100] \times 0 = 0.60651$ as the 'clean energy' carbon intensity for China, computed as a weighted mean of the Chinese emission factor of $I_{China} = 0.97463$ for the polluting provinces $\{d'\}$, and the carbon intensity of $I_d = 0$ for the non-polluting provinces $d \in \{D_{China} - d'\}$ ={Yunnan, Sichuan, Gansu, Qinghai} with weights of 0.0834, 0.265, 0.0253 and 0.004, respectively. Similarly, exploiting the information provided by Willms (2019), the 'clean energy' carbon intensity for the US obtains from $I^e_{US} = \{[100-(18+14+18+12)]/100\} \times 0.5471 + [(18+14+18+12)/100] \times 0 = 0.20790$, where the non-polluting US states $d \in \{D_{US} - d'\}$ ={Tenneesse, California, Oregon, Washington} with weights of 0.18, 0.14, 0.18 and 0.12, respectively. Combining both, we obtain a new overall average carbon intensity of $I^e = \sum_{c=1}^{C} s_c[\sum_{d=1}^{D_c} s_d I_d] = \sum_{c=1}^{C} s_c[\sum_{d=1}^{D_c - d'} s_d 0 + \sum_{d=1}^{d'} s_d I_d] = \sum_{c=1}^{C} s_c[\sum_{d=1}^{d'} s_d I_d] = 0.4784$.

Figure 5.7 displays the evolution of the upper and lower limits of the Bitcoin network daily

energy sources, we assume that the Bitcoin mining activity will only use renewable energy sources with a corresponding emission factor of zero.

carbon footprint (measured in $ktCO_2$) under both scenarios, I (in black) and I^e (in green), over the 2017-2019 period. Based on Equation 5.3, one could notice how the large fluctuation in the interval –observed in 2018 and 2019– is due to the dependence of the upper bound on the Bitcoin price (from Figure 5.1, it is possible to see how around the same period it is observed a spike in Bitcoin prices).

The annual Bitcoin network carbon footprint lower $\underline{CO_2}$ and upper $\overline{CO_2}$ limits obtain from adding the corresponding daily CO_2 emissions over the year, for each year considered, reported in million tons of CO_2, $MtCO_2$. Under scenario I (in black), annual Bitcoin mining emissions lie between 3.2 and 35.1 $MtCO_2$ for 2017, between 15.5 and 57.7 $MtCO_2$ for 2018, and between 16.7 and 56.3 $MtCO_2$ for 2019. Instead, under a 'clean energy' scenario I^e (in green), estimated annual emission bounds are: 2.5 and 27.2 $MtCO_2$ for 2017; between 12 and 44.6 $MtCO_2$ for 2018; and between 12.9 and 43.6 $MtCO_2$ for 2019.

5.3 Machine Learning the Carbon Footprint of Bitcoin Mining

The most salient fact about Figure 5.7, displaying the upper and lower limits for the carbon footprint associated with Bitcoin network mining, is the uncertainty surrounding the actual CO_2 emissions generated by Bitcoin production. This uncertainty stems from the difficulty in (i) determining the carbon intensity of the source of energy employed, as well as in (ii) estimating the actual power consumption involved in Bitcoin mining. In this section we exploit supervised machine learning (ML) methods to narrow down that uncertainty and provide a more accurate quantitative point prediction.

To understand the contribution of each difficulty, and for comparison with recent results in the literature, we adopt a realistic target level of electricity consumption and energy efficiency when mining Bitcoins, presenting results based on a 'top-down approach' to compute the associated (i) reported and (ii) 'clean energy' carbon intensities (e.g., Stoll et al., 2019), to then present our (iii) results based on a (partial) 'bottom-up' novel approach. Deploying a deep neural network with rectified linear unit activation functions (ReLu DNN) based on a comprehensive set of inputs, enables the construction of prediction intervals around point estimates of greenhouse Bitcoin mining emissions which convey substantially narrower data-based uncertainty levels relative to the difference between Hayes' (2017) upper and lower bounds.

5.3.1 Top-down Approach

Following Stoll et al. (2019), our ReLu DNN adopts as target output y, a 'realistic' level of CO_2 emissions, CO_2^r, from the Bitcoin network daily electricity consumption E^r associated with a 'realistic' energy efficiency use of hardware, e^r. As such, it will closely 'track' the lower emission limit, $\underline{CO_2}$, in (5.7):

$$\underset{[ktCO_2 \text{ per day, per TH/s}]}{CO_2^r} = E^r \cdot I + CO_2^{rw} = PUE \cdot e^r \cdot H \cdot I \times 10^{-9} + CO_2^{rw} \qquad (5.8)$$

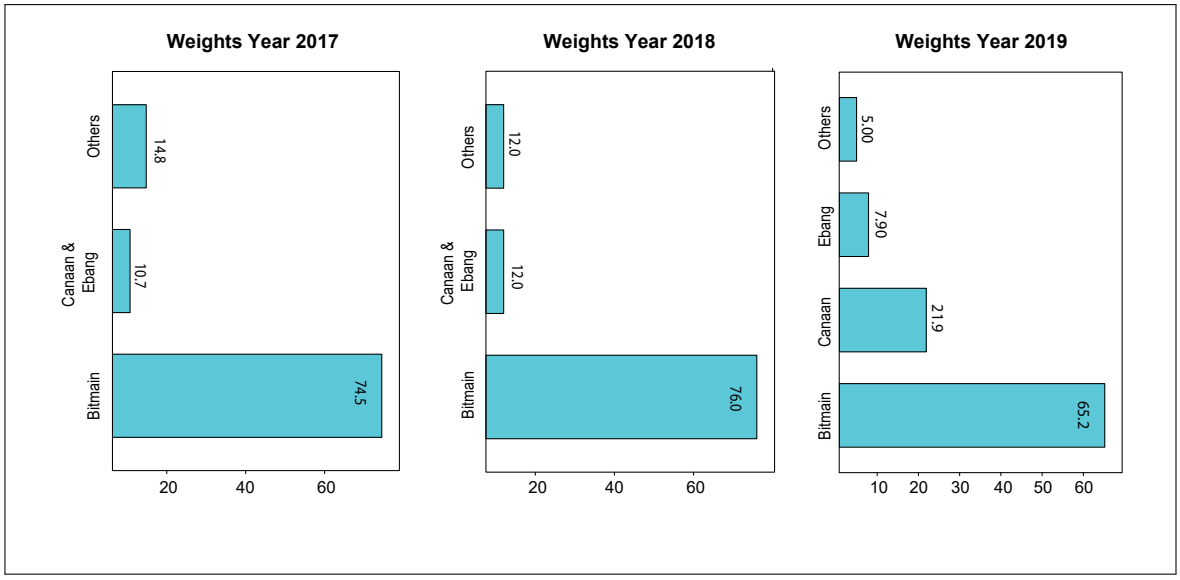


Figure 5.8: The Figure reports the weights assigned from 2017 until 2020 to the weighted mean of the energy efficiency of the different ASIC mining hardware.

where $PUE = \sum_{j=\{S,M,L\}} s_j \cdot PUE_j$ is the power usage of electricity, with s_j being the share of facility of type j, which can be small S, medium M or large L, and PUE_j is the corresponding power usage effectiveness of type j facility, with $PUE_S = 1.00$, $PUE_M = 1.10$, and $PUE_L = 1.05$. Throughout, and for comparison with CBECI, we will assume a value of $PUE = 1.05$, but we do examine the robustness of our main results to alternative values. The 'realistic' energy efficiency $e^r = \sum_{m=1}^{M} s_m^{ASIC} \cdot e_m^r$ obtains as a weighted mean of the *average* energy efficiency of all the reported ASIC mining chips at a given date, e_m^r, as displayed in Figure 5.2.[90] The weights associated with each ASIC mining chip producer, s_m^{ASIC}, are identified by the market share in terms of either computing power or revenue, and are obtained from the IPO filings disclosed in 2018 by Bitmain, and in 2019 by Canaan. For 2017, Frost & Sullivan report that Bitmain accounted for 74.5% of the revenue of the global ASIC mining hardware, Company E for 6.2%, and Company F for 4.5% (E and F's companies names were undisclosed). Based on these estimates, Figure 5.8 reports the actual weights, s_m^{ASIC}, between 2017 and 2020, assuming that they are constant during a given calendar year.[91]

An even more conservative realistic target obtains when instead of I, a 'clean energy' weighted carbon intensity I^e is considered in (5.8):

$$\underset{[\text{ktCO}_2 \text{ per day, per TH/s}]}{CO_2^{er}} = E^r \cdot I^e + CO_2^{rw} = 1.05 \cdot e^r \cdot H \cdot I^e \times 10^{-9} + CO_2^{rw} \tag{5.9}$$

[90] Considering M rational miners operating in the network, it is assumed that when a new mining chip is available, miner m will invest in updating the hardware. Therefore, the computational power of a particular mining chip at a given date is considered indicative of the energy efficiency of the ASIC producer m, until the release of a new chip.

[91] As of November 2018, Stoll et al. (2019) report that Bitmain accounts for 76% of the network computing power, and Canaan and Ebang account for 12%. Finally, looking at the IPO filings disclosed in November 2019 by Canaan, Frost & Sullivan report that as of July 2019, Bitmain accounts for 65.2% of the computing power of the market, Canaan for 21.9%, and Ebang 7.9%.

5.3.2 Bottom-up Approach

Our novel approach adopts as realistic target output y, a (partial) bottom-up (BU) approach to CO_2 emissions:

$$\underset{[\text{ktCO}_2 \text{ per day, per TH/s}]}{CO_2^{BU}} = 1.05 \cdot \sum_{m=1}^{M} e_m^r \cdot \sum_{c=1}^{C} s_{mc}^{ASIC} \cdot I_c^e \cdot H_c \times 10^{-9} + CO_2^{rw} \qquad (5.10)$$

where H_c is the daily Bitcoin network hash rate decomposed by country/regions, I_c^e is the 'clean energy' carbon intensity of region/country c, and s_{mc}^{ASIC} are the weights associated with each ASIC mining chip producer m in region/country c identified by the market share in terms of either computing power or revenue. They are obtained from the IPO filings disclosed in 2017 and 2018 by Bitmain, and in 2019 by Canaan, and from authors' imputation. Based on the incomplete information collected from the 2019 IPO filings, it is possible to obtain the geographical distribution by region $c = \{$America(US), Asia (excl. China), Europe, China$\}$ of the computing power shares of the main Bitcoin network mining operators $m = \{$BITMAIN, EBANG, CANAAN, Other$\}$, imputing the missing shares as if uniformly distributed across the remaining regions (marked with an '*'):

$c \backslash m$	$BITMAIN$	$EBANG$	$CANAAN$	$Other$	% By region c
America (US)	12	2.4	0.3*	1.25*	15.9
Asia (excl.China)	14.7	6.5*	0.3*	1.25*	22.7
Europe	7	6.5*	0	1.25*	14.8
China	31.5	6.5*	7.3	1.25*	46.6
2019 % By operator m	65.2	21.9	7.9	5	100

$c \backslash m$	$BITMAIN$	$EBANG$	$CANAAN$	$Other$	% By region c
America (US)	14	1.2	0.5*	0*	15.7
Asia (excl.China)	17	3.6*	0.5*	0*	21.1
Europe	8.1	3.6*	0	0*	11.7
China	36.9	3.6*	11	0*	51.5
2018 % By operator m	76	12	12	0	100

$c\backslash m$	$BITMAIN$	E	F	$Other$	% By region c
America (US)	13.8	1.55*	1.125*	3.7*	20.2
Asia (excl.China)	16.7	1.55*	1.125*	3.7*	23
Europe	8	1.55*	1.125*	3.7*	14.4
China	36	1.55*	1.125*	3.7*	42.4
2017 % By operator m	74.5	6.2	4.5	14.8*	100

Because our countries/regions are aggregated into $c = \{$America, Asia (excl. China), Europe, China$\}$, the country/region specific factor $I^e_c \cdot H_c = I^e_c \cdot s_c H$, where s_c is the share of the Bitcoin overall network hashrate H that is employed in region/country c, can be further decomposed within each country/region c. For example, considering $c = A(merica)$, since most Bitcoin miners are concentrated in the US and Venezuela, $I^e_A \cdot s_A H = \sum_{r \in A} I^e_r \cdot s_r H = I^e_{US} \cdot s_{US} H + I^e_V \cdot s_V H + \sum_{r \in A \backslash \{US,V\}} I^e_r \cdot s_r H$. And similarly for the other regions/countries c. When considering 'clean energy' carbon intensities I^e_c, since we only have data for the US and China, $I^e_c = I_c$ when $c = \{$Asia (excl. China), Europe$\}$, while $I^e_A \cdot s_A H = I^e_{US} \cdot s_{US} H + I_V \cdot s_V H + \sum_{r \in A \backslash \{US,V\}} I_r \cdot s_r H$, because we do not have information on 'clean energy' power sources for other countries in the America region other than the US, i.e. $I^e_r = I_r$, $r \in A \backslash \{US, V\}$.

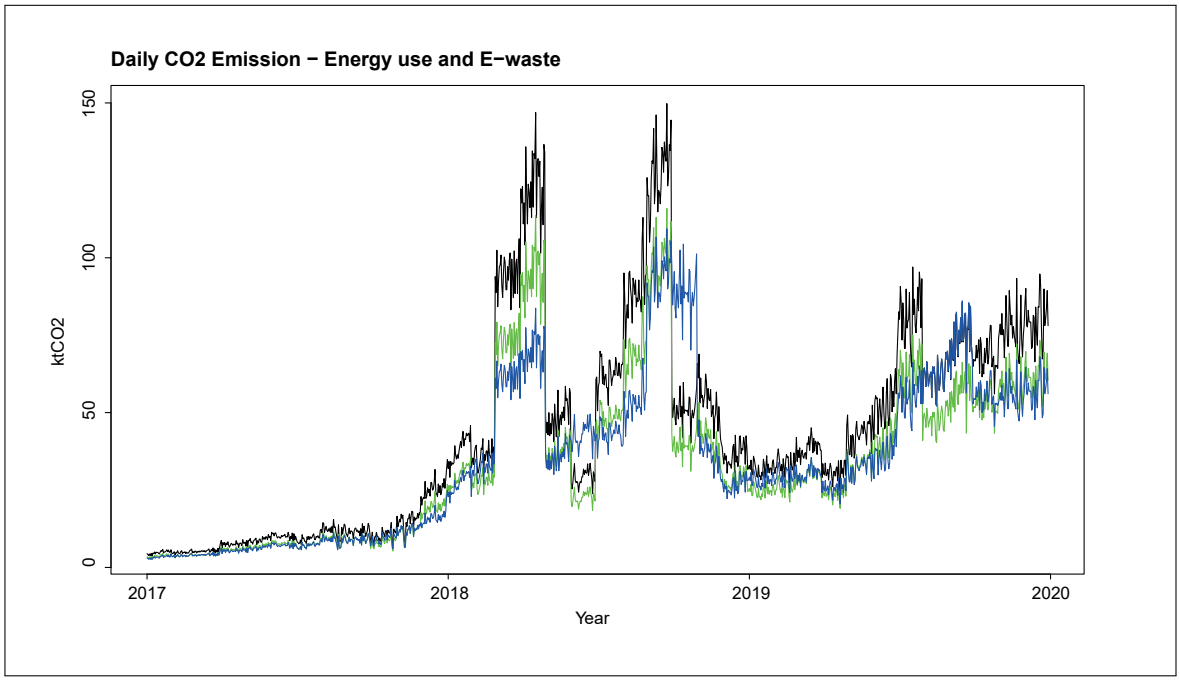


Figure 5.9: The Figure reports the realistic daily CO_2 emission in $ktCO_2$. The black and green lines report the realistic CO_2 emission obtained from a top-down approach that consider either I or I^e; the blue line obtains from the bottom-up approach.

For comparison, Figure 5.9 displays the daily evolution of the three different 'realistic' levels of CO_2 emissions, CO^r_2 in black, CO^{er}_2 in green, and CO^{BU}_2 in blue, from Bitcoin miners operating in the network over the period. The three of them are adopted separately as target outputs y to be learned by our supervised ML ReLu DNN on the basis of the collected

input data $\mathbf{X}$.

Because Bitcoin is a cryptocurrency based on a fundamentally new technology not fully understood –'blockchain'– while performing similar functions as other, more traditional assets, one key advantage of our ML-based approach is that it can handle big and complex input data in raw form, $\mathbf{X} = \{...\mathbf{x}_p...\}$. Our ReLu DNN will admit a very comprehensive set of $p = 1...P$ factors, excluding those necessary to compute the carbon footprint lower $\underline{CO_2}$ and upper $\overline{CO_2}$ limits derived in the previous section. The rationale behind this exclusion is to test whether our ML-based CO_2 emission mean predictions lie within the bounds that obtain from basic economic principles, externally validating our ML approach. In addition, we construct (95%) prediction intervals around our ML-CO_2 point estimates that are substantially narrower than the economics-based bounds, contributing methodologically to the ML and climate change literatures.

5.3.3 Input Data

The factors considered as input data range from (i) standard fundamental factors advocated by monetary economics and the quantity theory of money, like predictors of the Bitcoin price level; (ii) factors driving investors' interest in/attention to the cryptocurrency, like speculation or the role of Bitcoin as a safe haven; (iii) exchange rates with other currencies, to capture investors' hedging motives, e.g., the tight connection between the USD and the CNY markets; or (iv) supply-side factors for the costs incurred by Bitcoin and ASIC mining chips producers, related to rational for-profit mining decisions. Factors associated with the Blockchain network operation, like the network hash rate, difficulty or block reward, are excluded as they enter the definition of either the Bitcoin carbon footprint upper and lower bounds, or of our ReLu DNN target output. The resulting novel input dataset for the period 01/01/2017 - 31/12/2019 covers a comprehensive set of factors as reported in Kristoufek (2015), Liu and Tsivinsky (2018), McNally et al. (2018) and Jang and Lee (2017), adding some novel ones. Data are collected at different frequencies, and converted into daily ones using either simple imputations or Random Forest algorithms.

Because Bitcoin prices determine the upper limit of CO_2 emissions generated by the break-even electricity consumption of rational Bitcoin network miners, we start with the predictors of Bitcoin prices identified in the literature:

1. Commodity prices of Gold, platinum and crude oil are included because of the common traits shared with cryptocurrencies such as limited supply and high price volatility, but also because it is believed that Bitcoin could serve as an alternative to these commodities either as a store of value or as a hedging instrument (Dyhrberg, 2016). The daily future price of crude oil, and the spot prices of platinum (USD/ounce) and gold (USD/ounce) are obtained from Bloomberg;

2. Macroeconomic factors in different markets, such as consumption, production, and personal income growth (in USD), measure the extent to which Bitcoin is perceived as a

traditional financial asset, like the stock market. The $CAIPMOM$, $UKIPIMOM$, $IPCHNG$, $JNIPMOM$, and $SIIPMOM$ indices, measuring the volume of output in the industries of mining and quarrying, manufacturing and public utilities (electricity, gas, and water supply) for the USA, the UK, China, Japan and Singapore, as well as the indices $PITL$ and $PITLCHNG$, measuring the income received by households including wages and salaries, investment income, rental income, and transfer payments in the USA and China, are included. Finally, the $PCEMOM$ index quantifying the price changes for goods and services purchased by consumers in the USA is also considered;

3. Relative asset market performance measures capture the extent to which Bitcoin is similarly exposed to factors driving the returns of traditional assets. Based on Figure 5.3, we include the major stock market indices of the countries most relevant for Bitcoin mining: the USA, China, Venezuela, and Europe. For this reason, the indices S&P 500, Dow Jones, Nasdaq, Euro Stoxx 50, Shanghai Stock Exchange (SSE), Nikkei 225, FTSE 100, Caracas Stock Exchange (IBVC), and SHASHR will be considered as predictors;

4. Investor attention, measured by "Bitcoin" word Google searches. Liu and Tsyvinki (2018), Garcia et al. (2014) or Bouoiyour and Selmi (2017) empirically show that only cryptocurrency market specific factors –momentum and the proxies for investor attention– consistently explain the variations of cryptocurrency returns, suggesting that investors do not perceive them as traditional assets. Figure 5.10 reports the geographic location of daily data returned from Google Trends search queries for the word "Bitcoin", which quantifies the interest in the form of an index between 0 and 100. A value of 100 corresponds to peak popularity, and of 0 to insufficient data for Google to quantify any interest in the term "Bitcoin". With the exception of Nigeria, the country that receives the highest interest index, one could notice the similarity with Figure 5.3, where the geographical location of Bitcoin miners' IP addresses from the IoT search engine *Shodan* can be visualized, suggesting that a high value of the interest index is associated with Bitcoin mining activities.

5. Exchange Rates are included because of the popular belief that Bitcoin, if sufficiently adopted, may replace existing fiat currencies as a medium of exchange. Exposure of the cryptocurrency returns to major currencies is captured by the inclusion of the spot exchange rates between the USD and units of foreign currency, for the Australian Dollar, the Euro, the British Pound, the Canadian Dollar, the Singapore Dollar, the Swiss Franc, the Japanese Yen, the Chinese Yuan Renminbi (CNH), and the Chinese Yuan (CNY), all collected from Bloomberg. Being the Bitcoin price denominated in USD, Ciaian et al. (2016) notice that an appreciation of the USD against the above currencies could result in an appreciation against the Bitcoin and thereby, affect mining decisions through the reduction in the price of the cryptocurrency.[92]

6. The FED financial stress index (FSI) is a popular measure of financial uncertainty. Its

[92] We exclude the exchange rates of Bitcoin against other cryptocurrencies, like Ethereum or Ripple, because they are less popular, were introduced later and there is little evidence of significant arbitrage activity with respect to Bitcoin.

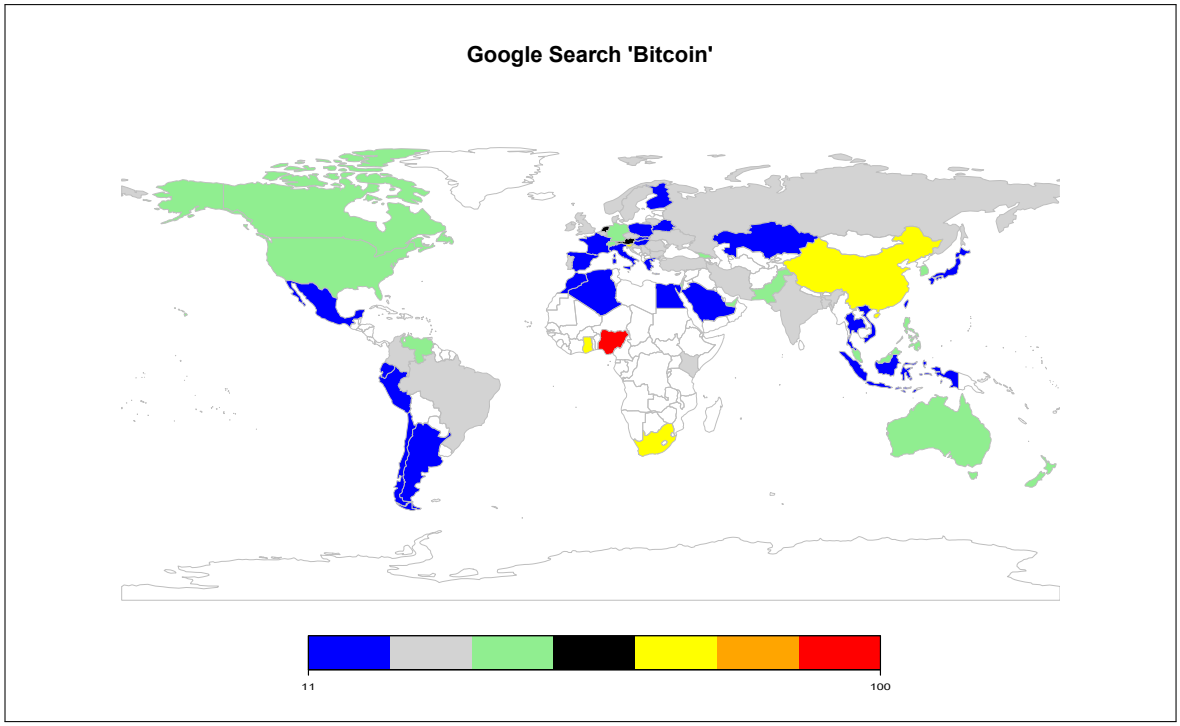


Figure 5.10: The Figure reports the Google search "Bitcoin" using 100 as reference for the maximum interest.

inclusion is intended to capture the possibility that Bitcoin is perceived as a safe haven, following Kristoufek (2015). The weekly series is provided by the Federal Reserve Bank of St. Luis (2016), and it is built from 18 different series of data at a weekly frequency: seven interest rate series, six yield spreads and five other indicators, each of which capturing a different aspect of 'financial stress'. The FSI is centered around 0 ('normal financial stress'), with negative values indicating unusual calmness and positive ones 'abnormally high' levels of financial uncertainty.

Finally, supply factors that proxy for the costs of Bitcoin mining and ASIC mining chips producers are also included:

7. ASIC mining chips producers offer mining hardwares (e.g. Antminers) the profitability of which is directly related to the marginal costs that can be expected from Bitcoin mining. Being electricity the most important input in mining for Bitcoins, we follow Liu and Tsyvinski (2018) and include the weighted average of the daily stock returns of 25 electricity companies in the USA and of 65 electricity companies in China, and the daily stock returns of Sinopec (SNP).[93]

8. To proxy for the cost of inputs relevant for manufacturing Antminers, we include the aluminum ($/25 Mt) and copper ($/25 Mt) prices –from Bloomberg– and predictors of

[93] The Sinopec has 4.02% missing at random values at a daily frequency, which are inputted using the MissForest algorithm (Stekhoven, 2013). The maximum number of trees to be grown in each forest is set equal to 500, the maximum number of nodes for each tree is equal to 500, and the maximum number of iterations is 20. The Out-Of-Bag (OOB) estimates of the imputation error in terms of Normalized Root Mean Squared Error (NRMSE) is: 2.160×10^{-9}.

the supply of coltan by its largest producers: the CDMNCLT index measuring the value (USD) of the mining and oil production in the Democratic Republic of Congo; and the RWEXCLVA and RWEXCLVO indices measuring the value and the volume (USD) of trade of coltan from Rwanda.[94]

After computing the log returns for the exchange rates, market indices, commodities prices, Sinopec prices and weighted averages of the electricity prices in the USA and China, a 'feature-wise normalization' or standardization –i.e features are centered around zero with unit standard deviation– is performed considering only the training dataset.[95]

5.3.4 ReLu DNN-CO_2 Estimation

We predict the carbon footprint associated with Bitcoin network mining by a ReLu feedforward deep NN, proceeding in two steps. First, we obtain the optimal structure of the ReLu DNN by applying the optimization proposed in chapter 2 for a given architecture size Z_{tot} and set of hyperparameters. Second, we optimize/fine-tune the DNN hyperparameters. Due to computational limitations[96], we allow for a maximum depth of $N = 15$. Because ML methods automate model selection and estimation, we first need to validate our method (optimal ReLu DNN) and show that it predicts greenhouse gas emissions better than competing methods 'out-of-sample'. We then present the results of the estimation and their reliability for the validated method.

Validation Methods

To validate the results obtained, we benchmark the (most correct) novel bottom-up target CO_2^{BU} results, from (5.10), against a DNN cross-validated architecture (cv)[97], and against a Random Forest (rf)[98], both state-of-the-art in the deep learning literature. The test data consists of daily observations for (each of three) target output(s) and the $P = 42$ input variables between 01/11/2019 and 31/12/2019. A ReLu DNN is fitted for each of the three different targets, corresponding to equations (5.8), (5.9) and (5.10). Due to the high dimensionality of the combinatorial problem, it is not feasible in (cv) to cross-validate all combinations and a 4-fold cross-validation over a randomized gridsearch is implemented instead. For each case, we report the out-of-sample mean-absolute error (MAE), mean-squared error (MSE) and square

[94] Copper is largely used for the production of electrical wires due to its high conductivity, heat resistance, and low cost. Aluminum wires are used for power transmission and distributions (generally not used in households). Coltan is employed in the production of tantalum capacitors, which are essential to manufacture mining hardware and computers.

[95] Being the levels and variances of the 42 input daily series considered significantly different, 'feature-wise normalization' is done to guarantee a proper training of the ReLu DNN.

[96] The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

[97] We perform cross-validation only on the optimal node allocation, for given DNN depth and size. Cross-validating both depth and hidden unit allocation across layers in network architectures of different sizes, is extremely computational and time consuming, and is therefore left for future work.

[98] The interested reader is referred to Breiman (2001).

root of the MSE (RMSE):

Method	Target Output	MAE	MSE	RMSE
Optimal ReLu DNN	CO_2^r	8.29	123.97	11.13
Optimal ReLu DNN	CO_2^{re}	6.17	58.76	7.67
Optimal ReLu DNN	CO_2^{BU}	**4.50**	**33.59**	**5.80**
Cross-validated ReLu DNN	CO_2^{BU}	5.35	48.48	6.96
Random Forest	CO_2^{BU}	7.17	82.62	9.09

Different architecture sizes Z, optimization algorithms (Adam, RMSProp), weight initialization values (s, s_1, s_2), learning rates ϵ, dropout rates q, and training epochs are considered during training.[99] The default 'minibatch' size of $B = 32$ is adopted and not tuned. The Random Forest hyperspace in (rf) is defined by the following parameters: (a) the number of variables to be randomly sampled at each sample split is defined in the interval $[20, 40]$, by intervals of 2; (b) the minimum size of the terminal nodes in $[2, 20]$, by intervals of 2; and (c) the number of trees to grow in the interval $[50, 500]$, by intervals of 50.

When the target is CO_2^r as defined by equation (5.8), the cross-validated NN architecture size that minimizes the out-of-sample MSE is found to be $Z = 1674$, with an optimal depth of $L = 15$ and optimal allocation of hidden units $[162, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126, 126]$.[100] The out-of-sample performance of the optimal ReLu DNN, as measured by the MAE, MSE, and RMSE, are 8.29, 123.97 and 11.13, respectively. The same optimal hyperparameters are selected when considering instead CO_2^{re}, defined by equation (5.9), returning out-of-sample MAE, MSE, and RMSE of 6.17, 58.76 and 7.67, respectively. Finally, when the bottom-up target is adopted, CO_2^{BU} as per equation (5.10), the out-of-sample MAE, MSE, and RMSE are 4.50, 33.59 and 5.80.[101] When benchmarked against (cv), the equally-sized 4-fold cross-validated network architecture of $[151, 125, 158, 91, 106, 74, 198, 131, 86, 71, 162, 132, 189]$ performs worse out-of-sample. The values for the MAE, MSE, and RMSE are 5.35, 48.48 and 6.96. Figure 5.11 reports the in- and out-of-sample MAEs of the four ReLu DNNs fitted.

[99] In particular, the different architecture sizes considered are $Z = \{200, 500, 800, 1674, 1800\}$,.the learning rates $\epsilon = \{0.0001, 0.001, 0.005, 0.003, 0.002, 0.01\}$ for the Adam optimizer ($\rho_1 = 0.9, \rho_2 = 0.999$), for the Stochastic Gradient descent (SGD) with Nesterov momentum of $\alpha = 0.9$, and for the RMSProp optimizer with $\rho = 0.9$ are tuned. When the Adam optimizer is considered the He normal initializer that draws samples from a truncated normal distribution with $\mu = 0$ and $\sigma = \sqrt{2/\text{Indim}}$ where "Indim" is the number of input units in the weight tensor (Keras documentation, 2020); when instead the SGD is tuned, a truncated normal distribution with $\mu = [0.5, 0.1]$ and $\sigma = [0.02, 0.01]$ is considered. The maximum number of training epochs analyzed are: 500, 1000, 2000, 5000 and 8000, and early stopping is applied. Different $q = \{0.05, 0.1, 0.2, 0.3\}$ are tuned for all hidden layers

[100] The cross-validated hyperparameters are: RMSProp optimiser with $\rho = 0.9$; learning rate, $\epsilon = 0.005$, dropout rate, $p = 0.1$ for all hidden layers, and number of epochs, 5000.

[101] The optimal hyperparameters are: RMSProp optimizer with $\rho = 0.9$; learning rate, $\epsilon = 0.003$, dropout rate, $p = 0.1$ for all hidden layers, and number of epochs, 5000.

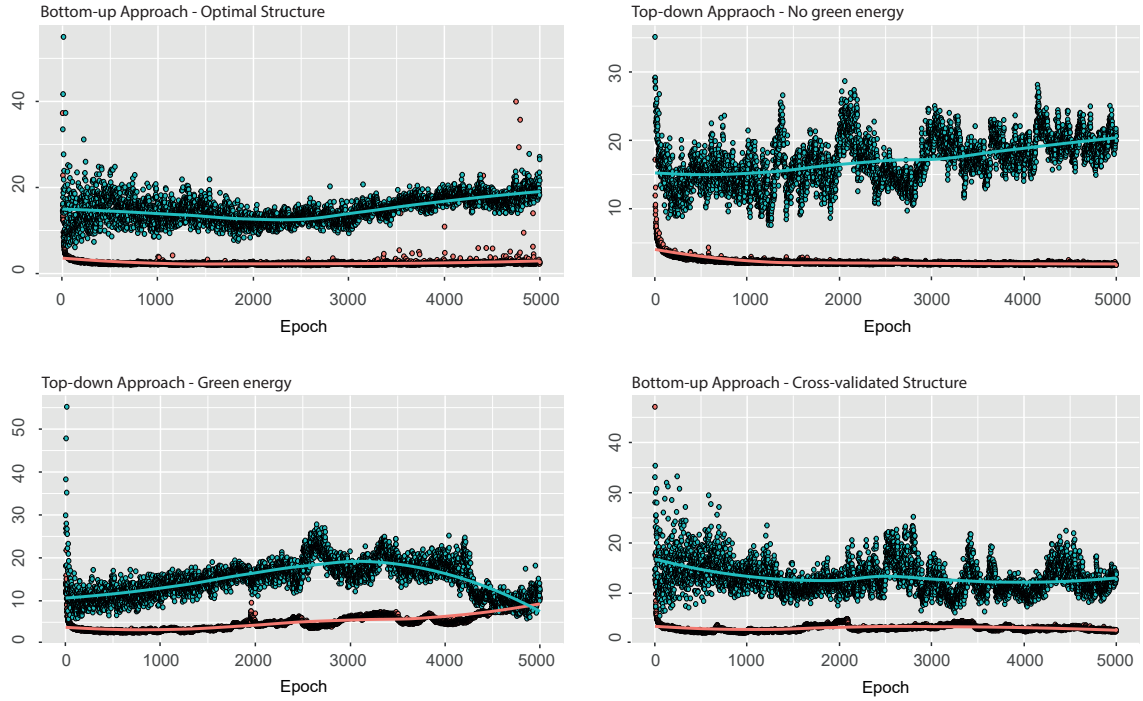


Figure 5.11: The Figure reports the training (in red) and validation (in blue) Loss, MAE, and MSE for the fitted optimal neural network with realistic targets (in panel): CO_2^{BU} (top-left), CO_2^{re} (top-right) and CO_2^{r} (down-left), while the down-right panel reports instead a cross-validated architecture for target CO_2^{BU}.

Finally, benchmarking against (rf), a Random Forest with node size of $\{6, 50\}$ trees, and 26 variables randomly sampled at each split, has associated out-of-sample MAE, MSE, and RMSE of 7.17, 82.62 and 9.09, respectively. A pairwise model comparison test statistic of the difference in out-of-sample MSE proposed in chapter 3, delivers a value of 3.77 (with associated p-value < 0.0001) for our optimal ReLu DNN against the (rf) random forest, and of 1.93 (with associated p-value of 0.0269) against the (cv) equally-sized cross-validated ReLu DNN, with levels of statistical confidence above five percent.

Therefore, our optimal ReLu DNN method better measures the carbon footprint associated with Bitcoin mining, relative to a Random Forest (rf) and a cross-validated ReLu DNN architecture (cv), both state-of-the-art ML methods. To further contribute to the literature on the carbon content of economic activity, we now report the levels and statistical reliability of the ML-measured CO_2 emissions associated with Bitcoin network mining.

CO_2-Emission Levels and Prediction Intervals

We obtain the following point estimates and associated 0.95 prediction intervals [PIs] for the yearly Bitcoin mining CO_2 emissions:

Optimal ReLu DNN Target/Year	2017	2018	2019
CO_2^{BU} ($MtCO_2e$)	2.77	16.08	14.99
[.95 PI]	[1.98,3.56]	[14.19,17.97]	[13.25,16.73]
CO_2^{re} ($MtCO_2e$)	2.98	18.11	17.45
[.95 PI]	[0.42,6.70]	[16.34,19.88]	[15.76,19.14]
CO_2^{r} ($MtCO_2e$)	3.72	23.98	20.06
[.95 PI]	[2.90,4.54]	[22.46,25.51]	[18.53,21.59]

The PIs are obtained from implementing the MC-dropout approach described above (Gal and Ghahramani, 2016a). Notice that irrespective of the approach adopted (bottom-up 'BU', or top-down), mean Bitcoin mining carbon emissions tend to increase over time. Comparing the three different targets adopted, the (partial) bottom-up approach delivers the most conservative greenhouse gas emissions.[102] Differences in the estimated yearly carbon footprints reported convey: (i) carbon intensity uncertainty (keeping the approach constant, e.g. the difference between CO_2^{re} and CO_2^{r} is solely due to adopting a 'clean' energy source carbon intensity, since both are *top-down* approaches) as well as (ii) uncertainty due to the change in approach, from adopting a novel *bottom-up* approach (CO_2^{BU}, relative to a *top-down* one that also assumes a 'clean' energy source carbon intensity , CO_2^{re}).

Figure 5.12 reports the estimated CO_2 emission values from our bottom-up target (upper panel), and associated 95 percent PIs (lower panel), for the overall period, contrasting them with Hayes' (2017) economic upper and lower bounds at a daily frequency.[103]

Three main aspects stand out: first, the ML-measured daily CO_2 emissions always remain within Hayes' (2017) rational Bitcoin mining upper and lower bounds, despite of not using that information directly as inputs, $\mathbf{X}$. Second, the 95 percent PIs displayed in the lower panel of Figure 5.12, provide a quantitative measure of the uncertainty associated with the ML-based Bitcoin mining carbon footprint, which is substantially narrower than the one captured by the difference between Hayes' (2017) upper and lower bounds, displayed in the upper panel. That difference,which corresponds to the expected daily operating margin of rational Bitcoin miners' decisions, is the current measure of uncertainty in the literature. Third, the estimates (and PIs) are in line with recent literature downward revisions of the original estimate of 69 $MtCO_2e$ provided by Mora et al. (2018) for 2017, e.g. 15.5 $MtCO_2e$ by Houy (2019), excluding unprofitable mining rigs; or 15.7 $MtCO_2e$ by Masanet et al.(2019); as well as with those for 2018, e.g. 43.9 $MtCO_2e$ (for Bitcoin and Ethereum) estimated by Foteinis (2018), or the lower and upper bounds of 22.0 (device IP method) to 22.9 (pool IP

[102] In a sense they are too conservative, because for 2017, all three targets return lower statistical lower bounds than Hayes' (2017) economic lower bound, suggesting that Bitcoin miners were deploying even more efficient ASIC chips than assumed, consuming even less energy overall. But all three point estimates remain above Hayes' (2017) economic lower bound.

[103] To get a sense from the Global Carbon Atlas, the maximum level of CO_2 that can be produced by Bitcoin mining (according to Hayes' 2017 upper bound) is higher than the emissions as of 2018 of (i) countries such as Norway (44 $MtCO_2$), Sweden (41 $MtCO_2$), Finland (47 $MtCO_2$) or New Zealand (35 $MtCO_2$); (ii) US states like Connecticut, Maryland, Nebraska, New Mexico or Oregon, or than (iii) those of all Earth's 91 subaerial volcanoes (i.e. not under water), with average yearly emissions of 38.7± 2.9 $MtCO_2$ between 2005 and 2015 (Aiuppa et al., 2019).

method) $MtCO_2e$ estimated by Stoll et al. (2019) for Bitcoin mining activity. To provide an order of magnitude, the estimates for the years 2018 and 2019 are comparable to the CO_2 yearly emissions of countries such as Bolivia, the Dominican Republic, or Croatia.

Figure 5.12: The top panel reports the economic upper and lower bounds for daily CO_2 emissions (in black), and within them, the (partial) bottom-up ReLu DNN-based daily CO_2 emissions point estimates, CO_2^{BU}, in red. The bottom panel reports the 95% prediction intervals (in black).

Recalling that the greenhouse emissions estimates reported here are the result of adopting a conservative target, one could conclude that the economic social cost associated with the proof-of-work algorithm is significant. This is an important aspect that must be considered by policy makers or financial institutions that are adopting blockchain technologies for national cryptocurrency production (e.g. China), or for the emission of financial instruments (e.g. *bond-i*), because of the Paris agreement that requires to all parties to put forward policy measures intended to keep the rise in temperature well below +2°C (e.g. zero net carbon emissions). Mora et al.'s (2018) projections, notwithstanding the aforementioned downward revisions, were based on the Paris agreement climate sensitivity of +1.5°C. But the latest evidence from a global effort comprising dozens of climate-change models (in an ensemble called the Coupled Model Intercomparison Project, CMIP6)[104], feeding into the Sixth Assessment Report of the Intergovernmental Panel on Climate Change (IPCC) due next year, now indicates climate sensitivities exceeding 5°C.[105] If one considers the exponential trend char-

[104]See go.nature.com/3garyzc.

[105]For example, Williams et al. (2020) have just confirmed on the basis of the CMIP6 Met Office Unified weather-climate model, the crucial role that cloud microphysics play in the upwards revision of the climate sensitivity figures: rising temperatures affect the size and relative concentration of water and ice droplets

acterizing the network hashrate (see for example Cocco and Marchesi, 2016), the empirical results presented in the present chapter suggest an alarming upward projection of the Bitcoin mining greenhouse gas emissions.

5.4 Conclusions

There is a growing concern about climate change. Recent evidence (e.g., from integrated weather-climate models) magnifies the contribution of greenhouse emissions, making a compelling urgent call to cut on those. By focusing on the CO_2 emissions associated with Bitcoin mining, here we show that its measurement is controversial and subject to significant uncertainty. The main reason being the complexity of the underlying object of study: how much electricity is actually consumed, and what are the actual carbon intensities of the energy sourced by a globally Geo-located network mining for Bitcoins. In a novel application of deep learning to this pressing societal issue, we were able to provide a quantitative measure of Bitcoin mining daily electricity consumption and associated CO_2 emissions, as well as of their (statistical) reliability, improving on the current methods employed in the literature. Although our estimates are in line with recent downward revisions of those provided by Mora et al. (2018), our novel (partial) bottom-up approach takes into account not only the location of miners within China (e.g. CBECI) but also within the US, to better capture the carbon intensity of the actual sources of energy employed. In addition, we incorporated miners' income fees, hardware e-waste (De Vries, 2019) and power usage effectiveness (Stoll et al., 2019), as well as allowed for different electricity prices across miners' countries, demonstrating that ML methods provide substantially narrower bounds than those currently employed in the literature, based on Hayes' (2017). Yet, our conclusions point towards a significant and substantial contribution towards rising world temperatures in view of the recent evidence of climate sensitivities exceeding 5°C (relative to the Paris agreement target level of +1.5°C).

We demonstrate how ML methods can be successfully exploited to contribute to the ongoing debate. Starting from a model of rational Bitcoin mining by Hayes (2017), and based on a comprehensive set of factors reported in Kristoufek (2015), Liu and Tsivinsky (2018), McNally et al. (2018) or Jang and Lee (2017), and some novel ones, we were able to measure the carbon footprint associated with Bitcoin mining from fitting an optimized deep neural network architecture that improves upon state-of-the-art DNN methods. ML methods help in establishing how the carbon footprint of the proof-of-work algorithm is higher than those of (i) US states of Maine, New Hampshire, Rhode Island or South Dakota, of (ii) economies of the size of Bolivia, the Dominican Republic, or Croatia, or of (iii) more than half the cumulative CO_2 flux from the Earth's 91 most actively degassing subaerial volcanoes (Aiuppa et al., 2019). Next to discussions about incorporating a mining tax into electricity prices, policy decision makers could also consider alternative ones like effective carbon pricing, in their efforts to curtail greenhouse emissions associated with cryptocurrencies mining and smart contracts

in a cloud ('cloud-feedback problem'), leading to more supercooled water droplets and less ice droplets. If confirmed, clouds contribute less than previously thought (e.g. CMIP5) to temperature 'cooling'. Cloud adjustment to climate change means that we need to redouble our efforts to cut emissions.

implemented with blockchain. Finally, ML methods could be fruitfully exploited to solve the 'measurement' problem that plagues for-profit financial efforts to decarbonize the economy.

CHAPTER 6

Environmental Engel Curves: A Deep Learning Approach

Chapter Abstract

Environmental Engel curves describe how households' income relates to the pollution associated with the services and goods consumed. This chapter estimates Environmental Engel curves using deep learning techniques and the novel dataset constructed in Levinson and O'Brien (2019). This approach is model-free and, in contrast to non-parametric kernel regression methods, can accommodate a large number of covariates. Additionally, we apply recent methods to measure uncertainty around predictions of deep learning models such as Monte Carlo dropout and extra-neural networks. We construct prediction intervals for five different pollutants that allow us to confirm statistically that Environmental Engel curves are upward sloping, have income elasticities of less than one, and are concave.

6.1 Introduction

Concerns about climate change and the effect of human intervention on global warming have prompted interest in the relationship between household consumption and pollution in recent years. Much of the early work on environmental economics was aimed to study suitable policies to tax pollution. For example, work by Metcalf (1999) combines the Consumer Expenditure Survey (CEX) with pollution data to study the incidence of a proposed pollution tax. In a related study, Hassett et al. (2009) combine CEX data from several years with pollution data from different industries to show that a carbon tax would be increasingly regressive. Grainger and Kolstad (2010) and Burtraw et al. (2009) use CEX data to show that a carbon tax would be regressive if not offset by lumpsum transfers or reductions in other regressive taxes. More recently, Levinson and O'Brien (2019) explore the concept of Environmental Engel Curves (EEC) and propose a structural approach to estimate the relationship between household income and pollution, holding prices constant. This idea extends the concept of Engel curves, see Engel (1895), which study the relationship between households' consumption of particular goods (or services) and households' income, holding prices constant.

EECs are related to Environmental Kuznets curves, that measure the relationship between pollutants and national income, see Grossman and Krueger (1995). However, EECs are structural, representing income expansion paths holding prices constant. Movements along EECs represent changes in preferences as income grows, holding prices, technologies and regulations fixed. Shifts in EECs represent changes in all of those other characteristics over time. Copeland and Taylor (2005) state that the relationship between economic growth and pollution can be described by three separate components: a) technique (capturing the technologies used for the production and manufacture of goods and services), b) composition (representing the basket of goods produces by the economy), c) and scale (quantifies the relation between economic activity and pollution – an increase in economic growth leads to a proportional increase in pollution).

Levinson and O'Brien (2019) compare pollution, income, and consumption across U.S. households with annual data over the period 1984 to 2012. These authors construct EECs separately for indirect emissions from each of the five major air pollutants: particulates smaller than 10 microns (PM10), volatile organic compounds (VOCs), nitrogen oxides (NO), sulfur dioxide (SO_2), and carbon monoxide (CO), and estimate two versions of each EEC: one based solely on income and one that controls for 18 household characteristics correlated with income, such as education and age. To calculate the pollution emitted by producing the goods and services associated with household expenditures, these authors pair the CEX with emissions intensities calculated from the National Emissions Inventory (NEI).[106] Levinson and O'Brien (2019) find that EECs display three key characteristics. First, EECs are upward sloping, meaning that richer households are responsible for more overall pollution. Second, EECs have income elasticities smaller than one, indicating that although pollution increases with

[106]These authors calculate the per dollar emissions intensity of each industry by aggregating industry-level emissions in the 2002 NEI and dividing by the total sales from the 2002 economic and agricultural censuses.

income, higher income households' consumption pollution content is smaller than for lower income households. And third, EECs shift down and become more concave over time, meaning that for any level of real household income, households in more recent years consume a less polluting mix of goods, the pollution content of which increases with income at a decreasing rate.

One of the main difficulties associated with the correct study of EECs is the absence of a theoretical framework that describes such relationship. As a consequence, EECs should be constructed with as few restrictions as possible. For this reason, Levinson and O'Brien (2019) use linear and nonlinear specifications (e.g., cubic polynomials and logarithms) between households' pollution and household-specific information finding similar results across specifications. These authors also consider nonparametric estimates of EECs in their analysis; however, these nonparametric specifications are only considered for the simplest case given by the relationship between household pollution and income. It is well known that nonparametric regression models are not able to accommodate the presence of many covariates, by construction, due to the well-known curse of dimensionality, see Stone (1980).

The aim of the current chapter is to analyze if the results in Levinson and O'Brien (2019) are robust to the choice of the functional form describing the ECCs. To do this, we apply deep learning methods to model the relationship between the pollution content in consumption and household income. This approach based on deep neural networks (DNN) allows us to predict nonparametrically the pollution content of household consumption as a function of household income and a large set of covariates. In addition, we apply recent methods developed in the machine learning literature such as Monte Carlo dropout, proposed by Gal and Ghahramani (2016a), and extra-neural networks in chapter 4 to construct intervals about the model predictions. These methods are shown to outperform parametric methods such as Hwang and Ding (1997) and nonparametric bootstrap methods, see Tibshirani (1996), in deep learning environments. By doing so, we are able to attach statistical measures of uncertainty to the pointwise predictions of the model and, hence, add statistical rigor to the empirical findings of Levinson and O'Brien (2019) on the relationship between pollution and income. We construct intervals for the predictions of household pollution along the EEC curve for the reduced model that only considers household income and also the extended model that incorporates eighteen covariates capturing households' characteristics. As in Levinson and O'Brien (2019), we consider five different pollutants that capture different dimensions of environmental pollution. The deep neural network models that we fit exhibit low mean square prediction errors and, very importantly, accurate coverage probabilities for the associated prediction intervals. These intervals confirm the empirical findings in Levinson and O'Brien (2019) for the five pollutants and the years 1984 and 2012: EECs are upward sloping, have income elasticities of less than one, and are concave. However, in contrast to Levinson and O'Brien (2019), the concave relationship between household income and household pollution is observed not only for the simple model but also for the neural network that considers household characteristics. In other words, Levinson and O'Brien (2019) reveal a concave relationship between household income and pollution only for the model that consider income and income squared (as it is

imposed by the quadratic nature of the regression model). Conversely, the present chapter, without imposing any structure on the relationship between variables, uncovers concave ECCs when also household specific information are included. Therefore, the present chapter adds to the research of Levinson and O'Brien (2019) by modeling a decreasing relationship between household income and pollution for top earners (i.e., the pollution content of consumption for households in the top of the income distribution is lower than for the individuals in lower deciles).

Neural networks and, more specifically, deep learning models are widely adopted in high-dimensional and nonlinear problems such as pattern recognition, biomedical diagnosis, and others (see Schmidhuber, 2015; LeCun et al., 2015 for review of the topic). The success of deep learning methods relies on their ability to approximate complex and unknown functional forms for the relation between a given outcome variable and a set of predictors. The Universal approximation theorem by Cybenko (1989) states that a shallow feedforward neural network (only one hidden layer) with a sigmoidal activation function can approximate any Borel measurable function with arbitrarily small error. Hornik (1991) extends Cybenko's (1989) results by proving that the ability to approximate arbitrarily well any given function depends on the number of hidden nodes and not on the particular activation function used. Lu et al. (2017) extend the Universal approximation theorem of Cybenko (1989) to width-bounded deep feedforward neural networks with rectified linear unit (ReLu) activation functions. The implications of these results are noteworthy: notwithstanding the unknown functional form of the underlying data generating process, a sufficiently large shallow or deep ReLu feedforward neural network will be able to approximate, accurately, the underlying function (Goodfellow et al., 2016). Therefore, it is no longer necessary to construct an ad hoc model for the specific nonlinearity to be learned, making both shallow and deep neural networks the least restricted nonlinear function class to be considered.

The rest of the chapter is organized as follows: Section 6.2 reports the definitions and notations used in the chapter. Section 3 analyzes the sources of uncertainty that must be modeled when constructing prediction intervals and reviews two novel deep learning methodologies adopted for the prediction of household pollution and the construction of prediction intervals. Section 6.4 reports and discusses the empirical results. Section 6.5 concludes.

6.2 DNN basics

The present section provides the notation and definitions used in the remainder of the chapter. It will first define Rectified Linear Unit (ReLu) activation functions and deep (sequential) feedforward neural networks with emphasis on univariate regression tasks and the definition of dropout.

6.2.1 Definition and Notations

Let y_i for $i = 1, \cdots, n$ denote the output (target) variable and $\mathbf{x}_i = (x_{1i}, \cdots, x_{di})$ a set of input variables (covariates) used to predict y_i. A general predictive model can be expressed as:

$$y_i = f(\mathbf{x}_i) + \epsilon_i \tag{6.1}$$

with $f(\mathbf{x}_i)$ a real-valued function used to predict the outcome variable; ϵ_i is the error term that satisfies $\mathbb{E}[\epsilon_i|\mathbf{x}_i] = 0$. The choice of the function $f(\mathbf{x}_i)$ depends on the loss function $L[y_i, f(\mathbf{x}_i)]$. For example, the best predictive model for a quadratic loss function is $f(\mathbf{x}_i) = \mathbb{E}[y_i|\mathbf{x}_i]$.

In standard regression settings, the question of interest is to approximate the unknown function $f(\mathbf{x}_i)$. It is well known that if the function $f(\mathbf{x}_i)$ is linear on $\mathbf{x}_i$, under standard regularity conditions, ordinary least square (OLS) regression methods provide unbiased, consistent, and efficient estimators of the model coefficients. However, many empirical problems are characterized by nonlinear relationships between the variables of interest. A large number of covariates in many regression settings also compromises the good theoretical properties of OLS methods. In this particular case, Levinson and O'Brien (2019) explain how there is no theory that defines the form of the income-pollution relationship and thus, the construction and analysis of the EECs should be conducted with as few restrictions as possible. Nonparametric kernel regression models are a possible solution for the simplified model that only considers the relationship between household income and pollution, but it is not a feasible option when we also consider the presence of additional covariates capturing household characteristics.

Recent advances in machine learning have shown that neural network methods provide accurate predictions without requiring specific knowledge of the underlying data generating process and thus, they are not affected by model misspecification and associated econometric issues. The universal approximation theorem (Cybenko, 1989) and recent contributions (e.g., Lu et al., 2017) show that a sufficiently large neural network is able to accurately approximate the underlying function notwithstanding the type of nonlinearity to be learned. In this setting, shallow and deep neural networks provide a powerful tool for the construction of EECs and thus, the present chapter considers $f(\mathbf{x}_i)$ to be in the class of fully connected feedforward neural networks (or multi-layer perceptrons, MLP) with ReLu activation functions[107].

A ReLu activation function can be defined as follows. Let $\boldsymbol{\theta}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$, with

$$\boldsymbol{\theta}(\mathbf{x}) = (\max\{0, x_1\}, \max\{0, x_2\}, \cdots, \max\{0, x_d\}) \tag{6.2}$$

where d denotes the number of covariates (input dimension). Alternatively, the ReLu activation function can be expressed as $\boldsymbol{\theta}(\mathbf{x}) = \mathbb{I}(\mathbf{x} > 0) \cdot \mathbf{x}$, with $\mathbb{I}(\mathbf{x} > 0)$ the indicator function.

Having defined the ReLu activation function, it is now possible to provide a definition of a

[107] Other prominent examples in the machine learning literature include support vector machines (SVMs), boosting algorithms (e.g., Adaboost), decision trees (and their generalization to random forests and extremely randomized trees), and nonparametric regressions in the spirit of nearest neighbors and local kernel smoothing.

ReLu deep neural network (DNN). For any two natural numbers $d,\ n_1 \in \mathbb{N}$, which are called input and output dimension respectively, a $\mathbb{R}^d \to \mathbb{R}^{n_1}$ ReLu DNN is given by specifying a natural number $N \in \mathbb{N}$, a sequence of N natural numbers $Z_1, Z_2, \cdots, Z_N$, and a set of $N+1$ affine transformations $\mathbf{T}_1 : \mathbb{R}^d \to \mathbb{R}^{Z_1}, \mathbf{T}_i : \mathbb{R}^{Z_{i-1}} \to \mathbb{R}^{Z_i}$, for $i = 2, \cdots, N$, and $\mathbf{T}_{N+1} : \mathbb{R}^{Z_N} \to \mathbb{R}^{n_1}$. Such a ReLu DNN is called a $(N+1)$-layer ReLu DNN, and is said to have N hidden layers. The function $f : \mathbb{R}^d \to \mathbb{R}^{n_1}$ is the output of this ReLu DNN that is constructed as

$$f(\mathbf{x}_i; \boldsymbol{\omega}) = \mathbf{T}_{N+1} \circ \boldsymbol{\theta} \circ \mathbf{T}_N \circ \cdots \circ \mathbf{T}_2 \circ \boldsymbol{\theta} \circ \mathbf{T}_1, \tag{6.3}$$

with $\mathbf{T}_n = \mathbf{W}^n \mathbf{h}_{n-1} + \mathbf{b}_n$, where - for $N = 1$ - $\mathbf{W}^n \in \mathbb{R}^{Z_1 \times d}$; $\mathbf{h}_0 \equiv \mathbf{x}$, with $\mathbf{x} \in \mathbb{R}^{d \times 1}$ the input layer, and $\mathbf{b}_n \in \mathbb{R}^{Z_1}$ is an intercept or bias vector. For $N \neq 1$, $\mathbf{W}^n \in \mathbb{R}^{Z_n \times Z_{n-1}}$ is a matrix with the deterministic weights determining the transmission of information across layers; $\mathbf{h}_{n-1} \in \mathbb{R}^{Z_{n-1}}$ is a vector defined as $\mathbf{h}_{n-1} = \boldsymbol{\theta}(\mathbf{T}_{n-1})$, and $\mathbf{b}_n \in \mathbb{R}^{Z_n}$. The function $\boldsymbol{\theta}$ is a ReLu activation function defined as $\boldsymbol{\theta}(\mathbf{T}_{n-1}) = \max\{0, \mathbf{T}_{n-1}\}$ and $\boldsymbol{\omega} = \{\boldsymbol{W}^n, \mathbf{b}_n\}_{n=1}^N$ collects the set of estimable features of the model. The *depth* of a ReLu DNN is defined as $N+1$. The width of the n^{th} hidden layer is Z_n, and the *width* of a ReLu DNN is $\max\{Z_1, \cdots, Z_N\}$. The *size* of the ReLu DNN is $Z_{\text{tot}} = Z_1 + Z_2 + \cdots + Z_N$, that corresponds to the total number of nodes in the neural network architecture. The number of active weights (different from zero) in the n^{th} hidden layer is $w_n = (Z_n \times Z_{n-1}) + Z_n$. The *number of active weights* in a fully connected ReLu DNN is $w_1 + w_2 + \cdots + w_N$. The same definition applies to shallow/single-layered networks by imposing $N = n = 1$.

Universal approximation theorems developed for ReLu DNN models (Lu et al., 2017) guarantee then that $f(\mathbf{x}_i; \boldsymbol{\omega})$ approximates the true unknown function $f(\mathbf{x}_i)$ in (6.1) arbitrarily well. In practice, there is an approximation error due to replacing $f(\mathbf{x}_i)$ by $f(\mathbf{x}_i; \boldsymbol{\omega})$ in model (6.1), where $f(\mathbf{x}_i; \boldsymbol{\omega})$ denotes a feasible version of the DNN model that can be estimated from the data.[108] The model that we consider in practice is

$$y_i = f(\mathbf{x}_i; \boldsymbol{\omega}) + u_i, \tag{6.4}$$

where $u_i = \varepsilon_i + f(\mathbf{x}_i) - f(\mathbf{x}_i; \boldsymbol{\omega})$, and $f(\mathbf{x}_i) - f(\mathbf{x}_i; \boldsymbol{\omega})$ reflects the approximation error of the model.

The construction of prediction intervals around the pointwise predictions of DNN models has most recently been object of important research in machine learning applications. The possibility of constructing prediction intervals allows us to measure the uncertainty around the model predictions. The concept of Monte Carlo dropout is central to this novel literature on prediction intervals for neural network models, see Srivastava et al. (2014). Before discussing the construction of prediction intervals, we elaborate on the concept of dropout in DNN models.

Training with dropout (*dropout training*) implies that for each iteration of the learning algorithm different random sub-networks (or *thinned* networks) are trained. Let h_{zn} denote

[108] The feasibility of the model entails that it is defined by a truncation of the true ReLu DNN model that approximates arbitrarily well the unknown function $f(\mathbf{x}_i)$.

the elements of the vector $\mathbf{h}_n$ for a given node $z = 1, \ldots, Z_n$ in layer $n = 1, \ldots, N$. Srivastava et al. (2014) develop a dropout methodology that is applied to each function h_{zn} to obtain a transformed variable $\overline{h}_{zn}$. This variable is obtained by pre-multiplying h_{zn} by a random variable r_{zn} with distribution function $F(r_{zn})$, such that $\overline{h}_{zn} = r_{zn} \cdot h_{zn}$, for all (z, n), prior to being fed forward to the activation function in the next layer, h_{zn+1}, for all $z = 1...Z_{n+1}$. For any layer n, $\mathbf{r}_n = [r_{1n}, \ldots, r_{Z_n n}] \in \mathbb{R}^{Z_n}$ denotes a vector of independent random variables. In this chapter we consider only the Bernoulli probability distribution $F(r_{zn})$, where each r_{zn} has probability p of being 1 (and $q = 1 - p$ of being 0). The vector $\mathbf{r}_n$ is then sampled and multiplied element-wise with the outputs of that layer, h_{zn}, to create the thinned outputs, $\overline{h}_{zn}$, which are then used as input to the next layer, h_{zn+1}. When this process is applied at each layer $n = 1, \ldots, N$, this amounts to sampling a sub-network from a larger network at each forward pass (or gradient step). At test time, the weights are scaled down as $\overline{\mathbf{W}}^n = p\mathbf{W}^n$, $n = 1, \ldots, N$, returning a deterministic output. We then identify $\mathbf{r}^\star = [\mathbf{r}_1, ..., \mathbf{r}_N]$ as the collection of independent random variables applied to a feedforward neural network of depth $N + 1$.

6.3 Prediction Intervals for DNN models

Prediction intervals for both shallow and deep neural networks are estimated from the predictive distribution of the model output. The seminal contribution by Hwang and Ding (1997) is the first study to propose an asymptotic prediction interval in shallow neural networks. Despite the theoretical appeal of this approach its implementation in large dimensions –and under the absence of a parametric setting– has important limitations associated with the correct computation of the Jacobian matrix of the model specification, see, for example, Tibshirani (1996) and Devieaux et al. (1998).

Recent work on neural network models introduces uncertainty through bootstrap resampling techniques and Monte Carlo simulation methods enabling the construction of prediction intervals for the outputs of DNN models. When focusing on the first sub-group, pairs and residual bootstrapping can be regarded as the methodologies most adopted by practitioners (see Dipu Kabir et al., 2018; Tibshirani, 1996; and Heskes, 1997 for reviews on the topic). Despite its importance in recent empirical work, the relevant literature identifies several limitations associated with bootstrapping methods: (i) Lee et al. (2015) show how resampling with replacement reduces the number of unique observations used to train the model by 37%; (ii) Lee et al. (2015) and Lanshminarayanan et al. (2017) show, empirically, how data resampling in ensembles of neural networks deteriorates not only the prediction accuracy but also the definition of the predictive uncertainty of the ensemble itself; (iii) El Karoui and Purdom (2018) show that both pairs and residual bootstrapping suffer from several problems when applied in high-dimensional linear regression problems. In particular, the residual bootstrapping tends to give under-conservative estimates of the uncertainty, while the pairs bootstrapping provides over-conservative estimates.

Gal and Ghahramani (2016a) propose an alternative approach for the approximation of

the predictive distribution of neural networks called MC dropout. This methodology was originally developed for Bayesian neural networks (Denker and LeCun, 1991) and subsequently extended beyond the Bayesian framework by Cortes-Cirano and Bender (2019), among other authors. Similarly, in chapter 4, we propose a novel methodology for constructing prediction intervals for deep neural networks designed to overcome the aforementioned bootstrap limitations. Both methodologies are analyzed more formally in the following subsections.

6.3.1 Monte Carlo Dropout

The MC dropout approach introduces randomness into the DNN prediction by implementing dropout not only during training but also during testing. Gal and Ghahramani (2016a) propose a new theoretical framework which uses dropout in DNNs as approximate Bayesian inference for deep Gaussian processes. In this subsection we adopt this methodology outside Bayesian neural networks and illustrate how to construct prediction intervals for the output y_i. The literature focusing on Bayesian deep neural networks concentrates on correctly approximating the posterior probability distribution of the output of the DNNs, which is often intractable. In particular, let $p(\widehat{y}\,|\mathbf{x}, \mathbf{X}, \mathbf{Y})$ denote the distribution of the predictive output $\widehat{y}$ conditional on the set of observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{Y} = \{y_1, \ldots, y_n\}$. The predictive probability distribution of the DNN model is

$$p(\widehat{y}\,|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int_{\boldsymbol{\Omega}} p(\widehat{y}\,|\mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y}) \mathrm{d}\boldsymbol{\omega}, \tag{6.5}$$

with $p(\widehat{y}\,|\mathbf{x}, \boldsymbol{\omega})$ the likelihood function of the observations, and $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ where $\boldsymbol{\Omega}$ denotes the parameter space.

Gal and Ghahramani (2016a) propose MC dropout to approximate the posterior probability distribution $p(\boldsymbol{\omega} \mid \mathbf{X}, \mathbf{Y})$, which is intractable, by a distribution function $q(\boldsymbol{\omega})$ that follows a Bernoulli distribution, Ber(p). The above predictive distribution can be approximated by

$$p(\widehat{y}\,|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int_{\Omega} p(\widehat{y}\,|\mathbf{x}, \boldsymbol{\omega}) q(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega}. \tag{6.6}$$

The predictive distribution (6.6) can be approximated using Monte Carlo methods. Thus, by sampling T sets of vectors from the Bernoulli distribution $\{\mathbf{r}^{\star(t)}\}_{t=1}^{T}$, one can approximate the above predictive distribution from the random sample $\widehat{y}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)})$, for $i = 1, \ldots, n$, where $\widehat{\boldsymbol{\omega}}^{(t)} = \{\widehat{\mathbf{W}}^{1(t)}, \ldots, \widehat{\mathbf{W}}^{N(t)}, \widehat{b}_1^{(t)}, \ldots, \widehat{b}_N^{(t)}\}$ denotes the sequence of weights associated to the different nodes and layers of the neural network and the associated bias parameters. Using this MC dropout technique, Gal and Ghahramani (2016a) propose the first moment from the MC predicted outputs as the model prediction:

$$\bar{f}_{MC}(\mathbf{x}_i) = \frac{1}{T} \sum_{t=1}^{T} \widehat{y}(\mathbf{x}_i; \widehat{\boldsymbol{\omega}}^{(t)}), \text{ for } i = 1, \ldots, n. \tag{6.7}$$

These authors show that, in practice, this is equivalent to performing T stochastic forward

passes through the network and averaging the results. The predictive variance is then:

$$\widehat{\sigma}^2_{MC} = \widehat{\sigma}^2_e + \frac{1}{T}\sum_{t=1}^{T}\left(\widehat{y}(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)}) - \bar{f}_{MC}(\mathbf{x}_i)\right)^2, \tag{6.8}$$

with $\widehat{\sigma}^2_e = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{f}_{MC}(\mathbf{x}_i)\right)^2$ a consistent estimator of σ^2_e under homoscedasticity of the error term, see also Kendall and Gal (2017). The first component on the right hand side of expression (6.8) captures the aleatoric uncertainty (noise due to the error term) whereas the second term captures the epistemic uncertainty associated to parameter estimation. A suitable prediction interval for y_i under the assumption that $p(\widehat{y}\,|\mathbf{x},\boldsymbol{\omega})$ is normally distributed is

$$\bar{f}_{MC}(\mathbf{x}_i) \pm z_{1-\alpha/2}\widehat{\sigma}_{MC}. \tag{6.9}$$

6.3.2 Extra-neural network

This novel methodology extends the extremely randomized trees proposed by Geurts et al. (2006) to ensembles of neural networks. In this case, the original concept of an ensemble of sub-networks – from which the dropout training is built upon – is adopted. In the remainder of the subsection, $\bar{\mathbf{r}}^{\star}$ is used to identify a fixed Bernoulli mask, as opposed to $\mathbf{r}^{\star}$ used in dropout training. When the extra-neural network methodology is applied, T sets of vectors $\{\bar{\mathbf{r}}^{\star(t)}\}_{t=1}^{T}$ are sampled from the Bernoulli distribution prior to training and kept constant throughout the fitting and prediction phases. Extra-neural network thus effectively trains and independently fits T random sub-networks on the same dataset. Therefore, the predictive distribution is approximated using an ensemble approach comprised of different sub-networks fitted on the same dataset. The fixed Bernoulli mask (and thus, the consequential randomization of the structure of the different T sub-networks) introduces an additional randomization scheme to the ensemble's predictions that reduces the statistical dependence between the T predictions.

In particular, consider T fitted sub-networks defined as $f_t(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)})$ with $t = 1,\cdots,T$. We use f_t to note that each prediction belongs to a potentially different model; $\widehat{\boldsymbol{\omega}}^{(t)}$ denotes the sets of trainable parameters associated to each f_t. The ensemble's prediction is defined as:

$$\bar{f}_{EN}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)}), \text{ for } i = 1,\ldots,n. \tag{6.10}$$

Given the ensemble predictor expressed above, and assuming that the approximation bias is negligible, a suitable prediction interval is

$$\bar{f}_{EN}(\mathbf{x}_i) \pm z_{1-\alpha/2}\left(\frac{\widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i)}{T} + \widehat{\sigma}^2_e\right)^{1/2}, \tag{6.11}$$

with $\widehat{\sigma}^2_{\widehat{\boldsymbol{\omega}}}(\mathbf{x}_i) = \frac{1}{T}\sum_{t=1}^{T}(f_t(\mathbf{x}_i;\widehat{\boldsymbol{\omega}}^{(t)}) - \bar{f}_{EN}(\mathbf{x}_i))^2$ and $\widehat{\sigma}^2_e = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{f}_{EN}(\mathbf{x}_i)\right)^2$, where n is the length of the test sample.

In this chapter, we apply the MC dropout and the extra-neural network approaches to estimate the uncertainty about the pointwise predictions of a model that assesses the relation-

ship between household income and pollution using the rich dataset in Levinson and O'Brien (2019) for studying EECs for the U.S. for each year between 1984 and 2012.

6.4 Empirical Results

Levinson and O'Brien (2019) find that EECs display three key characteristics: increasing and concave relationship between the pollution content of household consumption and income, and an income elasticity smaller than one. These features of the relationship between the variables can be quantified and assessed through the construction of prediction intervals for a given coverage probability. Thus the hypothesis that the relationship is increasing could be tested by assessing if the first derivative of the predicted function modelling the relationship between pollution and household income is strictly positive. Similarly, the concavity of the relationship could be tested by assessing if the second derivative of the predicted functional form is negative. Once the concavity of the function is not rejected, the third hypothesis given by an income elasticity smaller than one can be tested by assessing if the slope of the functional form relating household pollution and income is less than one uniformly over the relevant domain.[109]

Following Levinson and O'Brien (2019), we consider five different pollutants: particulates smaller than 10 microns (PM10), volatile organic compounds (VOCs), nitrogen oxides (NO), sulfur dioxide (SO_2), and carbon monoxide (CO). Due to space constraints, we only report the results for the years 1984 and 2012 but results for the years 1985 to 2011 are available from the authors upon request. We implement the two novel methodologies reviewed above for the construction of prediction intervals in a DNN framework. The optimal architecture and set of hyper-parameters used to identify the ReLu DNN for the MC dropout is tuned, and the same optimal combination is applied for the implementation of the extra-neural networks algorithm. Levinson and O'Brien (2019) construct two types of EECs: one using only income as a covariate and, alternatively, a multivariate model in which the set of covariates is expanded to incorporate other households' specific information. In the latter case, Levinson and O'Brien (2019) consider $d = 18$ regressors to predict pollution.[110] Due to differences in dimensionality across problems (using income only or adding household covariates), in the first case, a set of candidate shallow networks with $Z_{\text{tot}} = [5, 10, 20]$ is considered, where Z_{tot} denotes the total number of nodes in the neural network. In the second case, we consider a DNN with optimal architecture obtained from the novel procedure in chapter 3.[111]

The hyper-space defined by the different optimization algorithms, weights initializers, learning rates, number of epochs, and drop-out rates are defined with no distinction between shallow and deep networks. In particular, the learning rates 0.001, 0.003, 0.005, and 0.01 for

[109] The reader should note that this definition of elasticity is different from the general definition of the elasticity of Y with respect to X that is given by $E_X^Y = \frac{\%\text{ change in Y}}{\%\text{ change in X}}$, which reduces to $E_X^Y = \frac{dY}{dX}\frac{X}{Y}$ for infinitesimal changes and differentiable variables.

[110] See Table 2 in Levinson and O'Brien (2019) for a detailed description of the variables.

[111] The neural network size depends on the complexity of the functional form to be approximated. Therefore, DNNs with different number of total nodes $Z_{\text{tot}} = [76, 90, 150, 200, 250, 392, 446, 500]$ are tuned.

the Adam optimizer ($\beta_1 = 0.9$; $\beta_2 = 0.999$), and for the RMSProp optimizer with $\rho = 0.9$ are tuned. When the Adam and RMSProp optimizer are analyzed, the He normal initializer and the Xavier uniform (default in Keras) initializer are implemented. The former draws samples from a truncated normal distribution with $\mu = 0$ and $\sigma = \sqrt{2/\text{Indim}}$ where "Indim" is the number of input units in the weight tensor; the latter draws samples from a uniform distribution within $[-\text{bound}, +\text{bound}]$, where bound $= \sqrt{(6/(\text{Indim} + \text{Outdim}))}$. Additionally, the stochastic gradient descent optimization algorithm with learning rate 0.0001 is also considered. The number of epochs (with early-stopping) analyzed are: 100 (for the shallow network), and 600 (for the deep neural networks). We also consider the following dropout rates (q): 0.01, 0.05, and 0.1.

Table 6.1: The table reports the optimal neural networks' parameters with relative out-of-sample accuracy measures defined by MSE, MAE, and Cov_{95} (empirical coverage probabilities at 0.95).

	Optimizer	Learning Rate	Epochs	Initializer	Structure	q	T	MAE	MSE	Cov_{95}
PM10										
Year 1984(d = 18)										
MC Dropout	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	2.8594	18.3263	0.05
Extra-network	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	2.9897	19.6456	0.04
Year 1984(d = 2)										
MC Dropout	Adam	0.003	100	He Normal	[5]	0.05	70	3.6276	27.3797	0.04
Extra-network	Adam	0.003	100	He Normal	[5]	0.05	70	3.6723	27.5817	0.04
Year 2012(d = 18)										
MC Dropout	Adam	0.003	600	He Normal	[54, 36]	0.05	70	2.5912	13.2871	0.04
Extra-network	Adam	0.003	600	He Normal	[54, 36]	0.05	70	2.6437	13.8489	0.04
Year 2012(d = 2)										
MC Dropout	Adam	0.003	100	He Normal	[5]	0.05	70	2.9997	16.6166	0.04
Extra-network	Adam	0.003	100	He Normal	[5]	0.05	70	3.0418	17.1902	0.04
CO										
Year 1984(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[40, 36]	0.05	70	12.4441	388.4378	0.05
Extra-network	Adam	0.005	600	He Normal	[40, 36]	0.05	70	12.3746	411.5797	0.05
Year 1984(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	14.4730	479.8068	0.06
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	14.4079	494.3869	0.06
Year 2012(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[78, 36, 36]	0.05	70	9.0952	236.0679	0.04
Extra-network	Adam	0.005	600	He Normal	[78, 36, 36]	0.05	70	9.2170	234.8622	0.03
Year 2012(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	9.8253	260.4555	0.04
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	9.5545	281.7539	0.04
SO_2										
Year 1984(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[54, 36]	0.01	70	30.6778	1844.7867	0.04
Extra-network	Adam	0.005	600	He Normal	[54, 36]	0.01	70	32.0010	2042.0742	0.05
Year 1984(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	37.2184	2810.3498	0.04
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	38.1013	2901.2416	0.05
Year 2012(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[40, 36]	0.05	70	32.3183	1931.8550	0.04
Extra-network	Adam	0.005	600	He Normal	[40, 36]	0.05	70	32.5744	1945.6825	0.05
Year 2012(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	35.7883	2291.4632	0.03
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	35.8810	2318.8336	0.03

	Optimizer	Learning Rate	Epochs	Initializer	Structure	q	T	MAE	MSE	Cov_{95}
NO										
Year 1984(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[54, 36]	0.05	70	18.8255	643.9020	0.05
Extra-network	Adam	0.005	600	He Normal	[54, 36]	0.05	70	19.2223	707.9086	0.06
Year 1984(d = 2)										
MC Dropout	Adam	0.003	100	He Normal	[5]	0.01	70	23.0578	985.8477	0.04
Extra-network	Adam	0.003	100	He Normal	[5]	0.01	70	23.1836	1029.5816	0.05
Year 2012(d = 18)										
MC Dropout	Adam	0.005	600	He Normal	[54, 36]	0.1	70	17.5517	541.5600	0.03
Extra-network	Adam	0.005	600	He Normal	[54, 36]	0.1	70	17.7210	581.2127	0.04
Year 2012(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	19.7508	695.7539	0.04
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	19.9146	699.3635	0.04
VOC										
Year 1984(d = 18)										
MC Dropout	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	5.4403	76.1747	0.04
Extra-network	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	5.2475	76.1247	0.05
Year 1984(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	6.5387	96.0701	0.05
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	6.4422	97.8181	0.05
Year 2012(d = 18)										
MC Dropout	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	3.5214	33.7558	0.05
Extra-network	Adam	0.003	600	He Normal	[78, 36, 36]	0.05	70	3.7183	33.5824	0.04
Year 2012(d = 2)										
MC Dropout	Adam	0.005	100	He Normal	[5]	0.05	70	4.0099	40.1979	0.04
Extra-network	Adam	0.005	100	He Normal	[5]	0.05	70	4.0851	41.0851	0.04

By looking at Table 2 in Levinson and O'Brien (2019), one could notice that when the EECs are constructed considering control variables for age, household size, marital status and indicators for race, education and regional location both numerical and categorical variables are included. Thus, in order to guarantee a proper training of the ReLu DNN, a feature-wise normalization for the numerical variables, consisting on transforming the observations into zero-mean and unit standard deviation random variables, is performed. When looking at the categorical variables, Levinson and O'Brien (2019) define a separate indicator for each category; being this approach equivalent to the one-hot-encoding procedure (see James et al., 2013), we have applied the same transformation in treating the categorical variables for the correct fitting of the DNN. Finally, 85% of the data are used for training the network and the remaining 15% as test set. We focus on the years 1984 and 2012 to assess the evolution of the EECs over time. The optimal combination of structure and hyper-parameters of both deep and shallow networks used for the year 1984 and 2012 are reported in Table 6.1 with the relative out-of-sample accuracy measures defined by MAE, MSE, and empirical coverages. The out-of-sample empirical coverage is approximately equal or equal to the nominal level at which the prediction intervals are constructed, implying the suitability of the intervals out-of-sample. The reported results in Table 1 thus convey that the data-based uncovered relationship between the pollution content of U.S. household consumption and income obtained by deep learning is statistically robust. We now turn to examine whether the increasing and

concave relationship uncovered by Levinson and O'Brien (2019) also obtains when estimated non-parametrically.

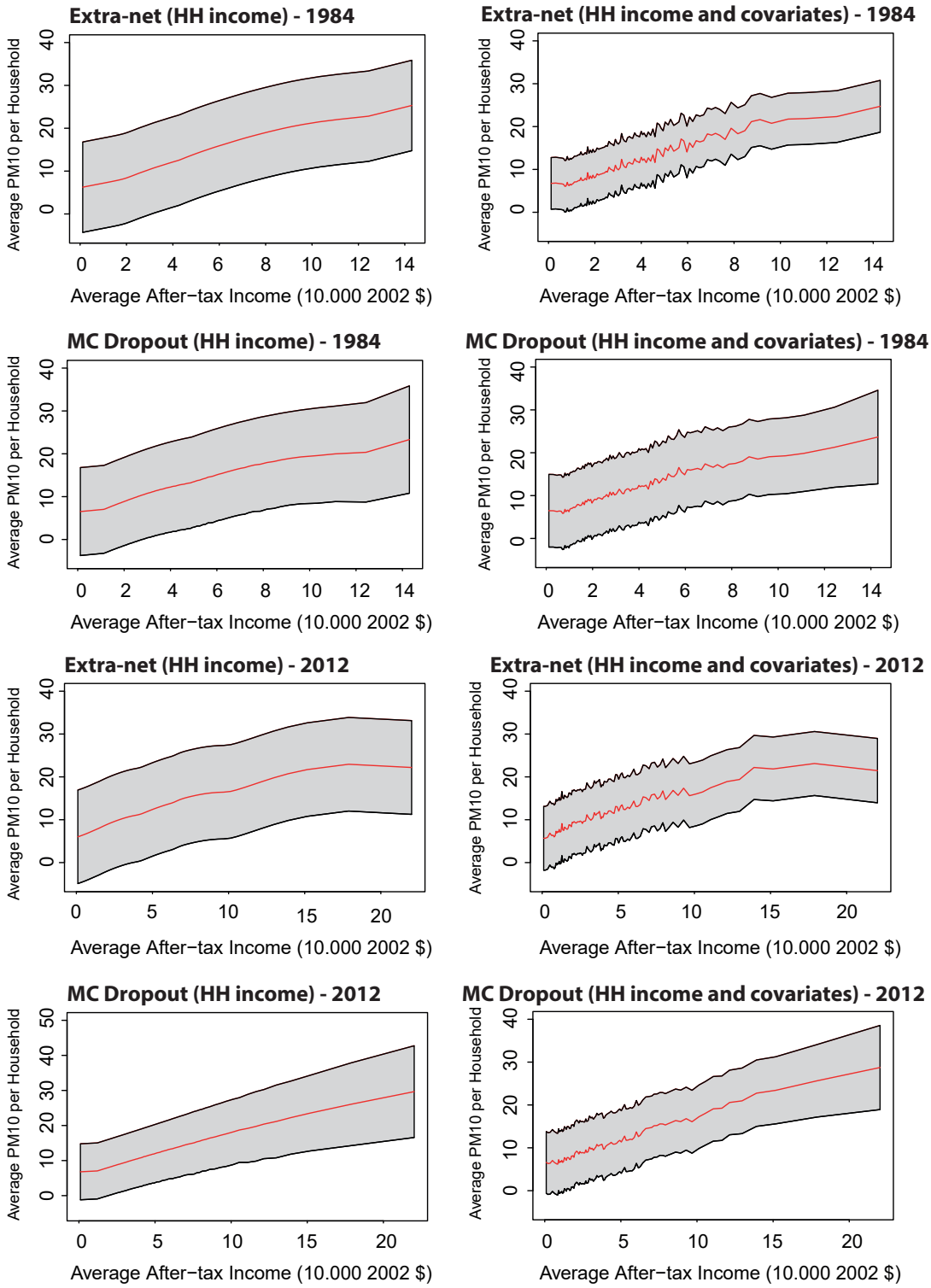


Figure 6.1: Point estimates and 0.95 prediction intervals of PM10.

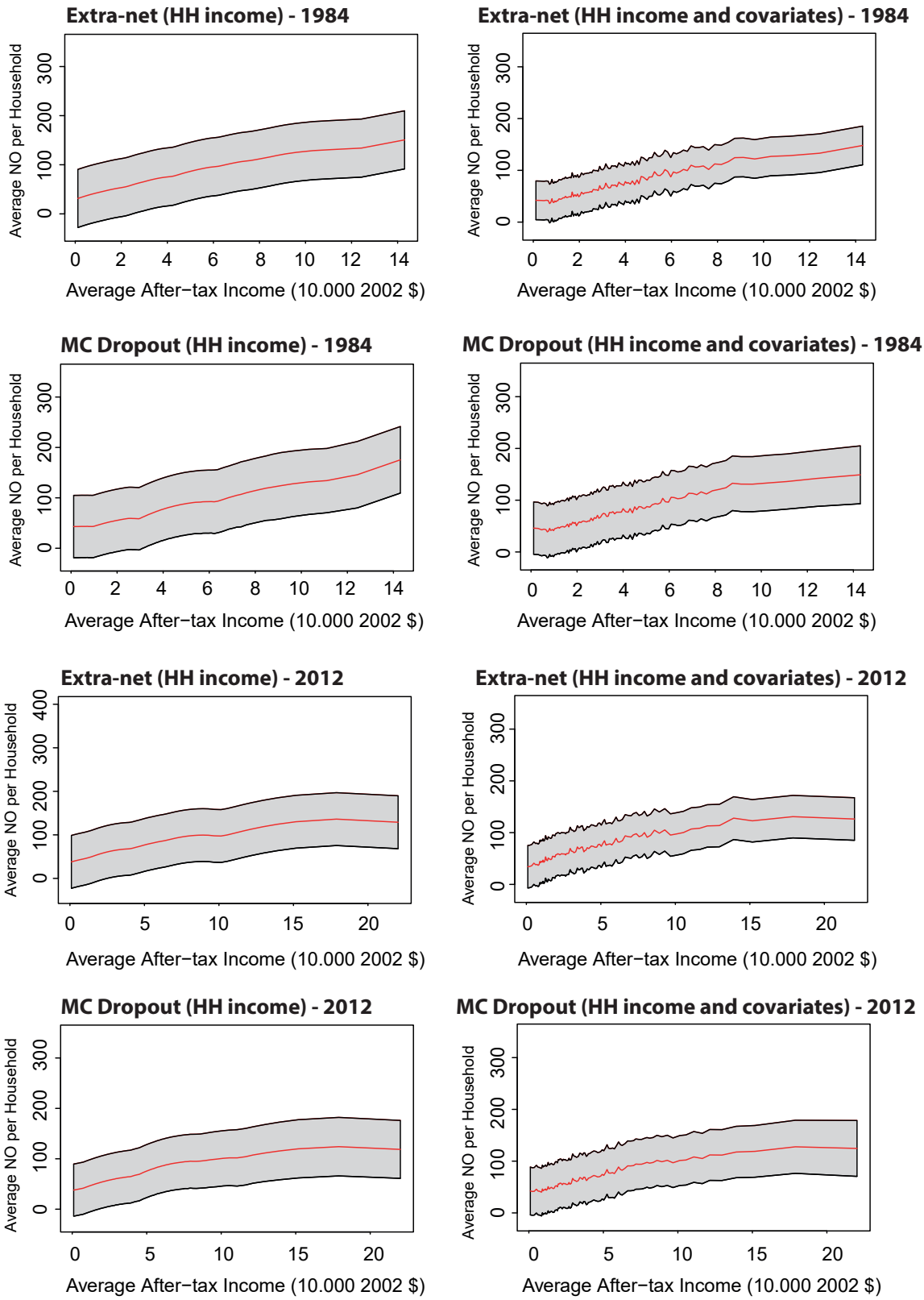


Figure 6.2: Point estimates and 0.95 prediction intervals of NO.

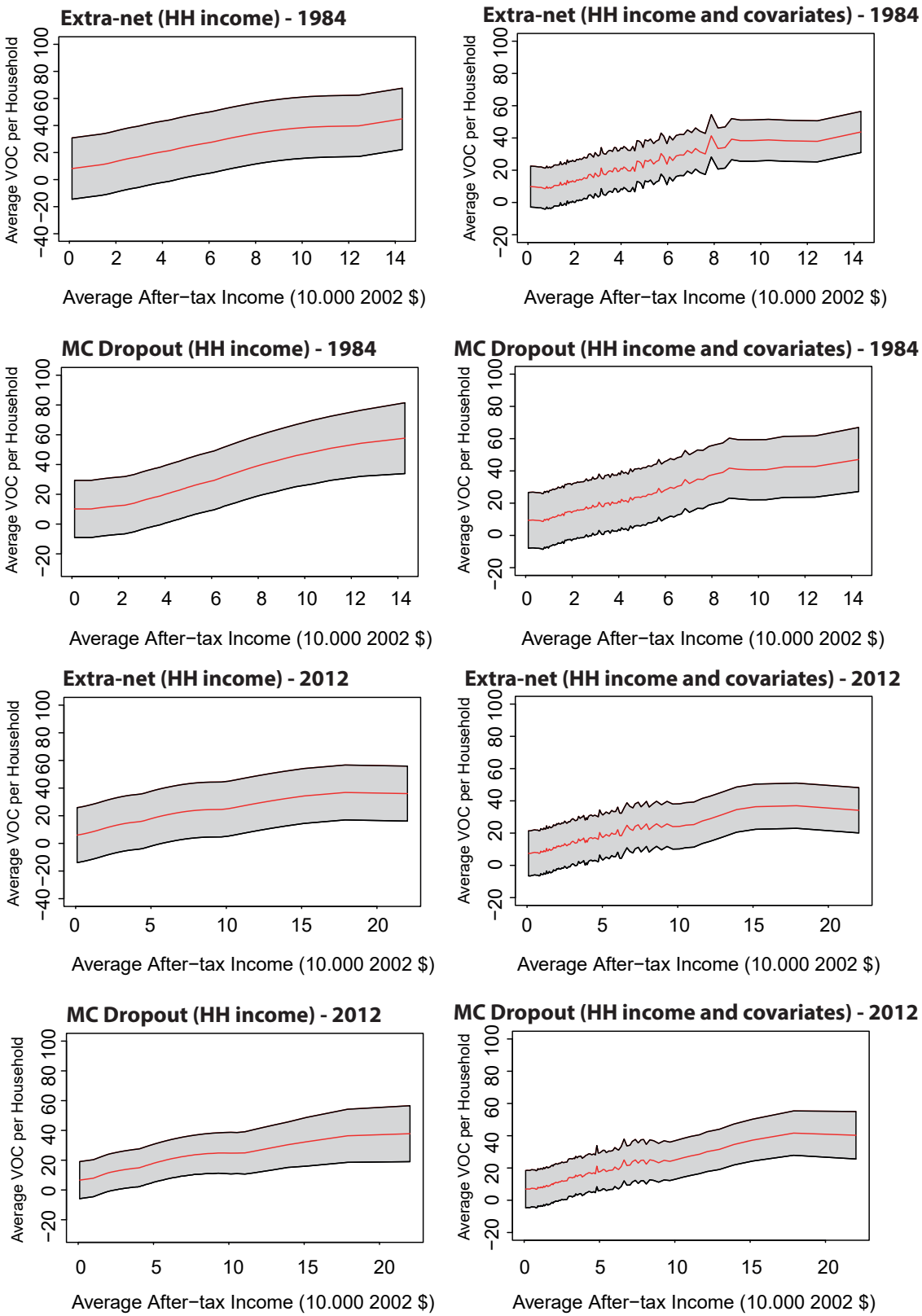


Figure 6.3: Point estimates and 0.95 prediction intervals of VOC.

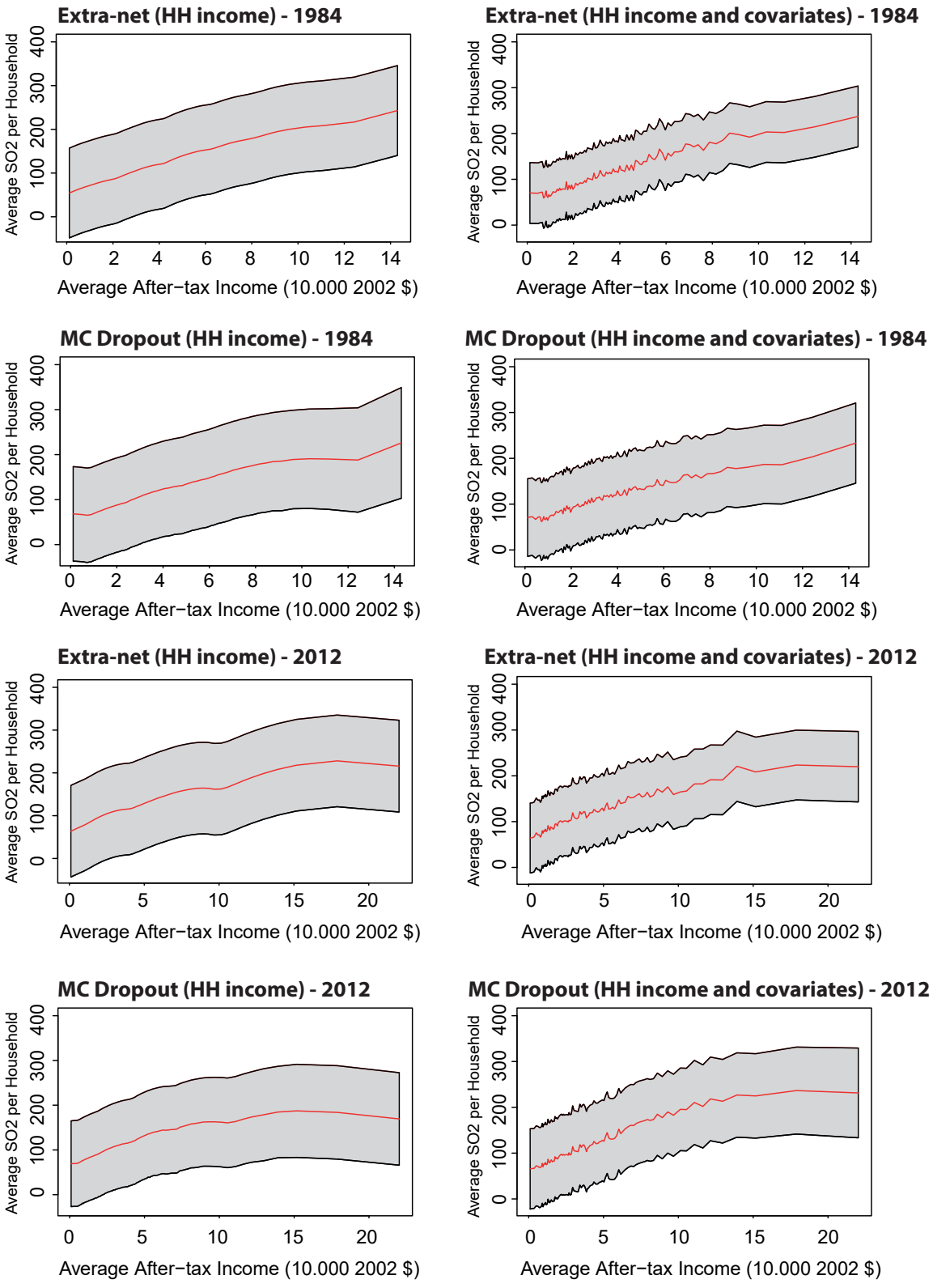


Figure 6.4: Point estimates and 0.95 prediction intervals of SO_2.

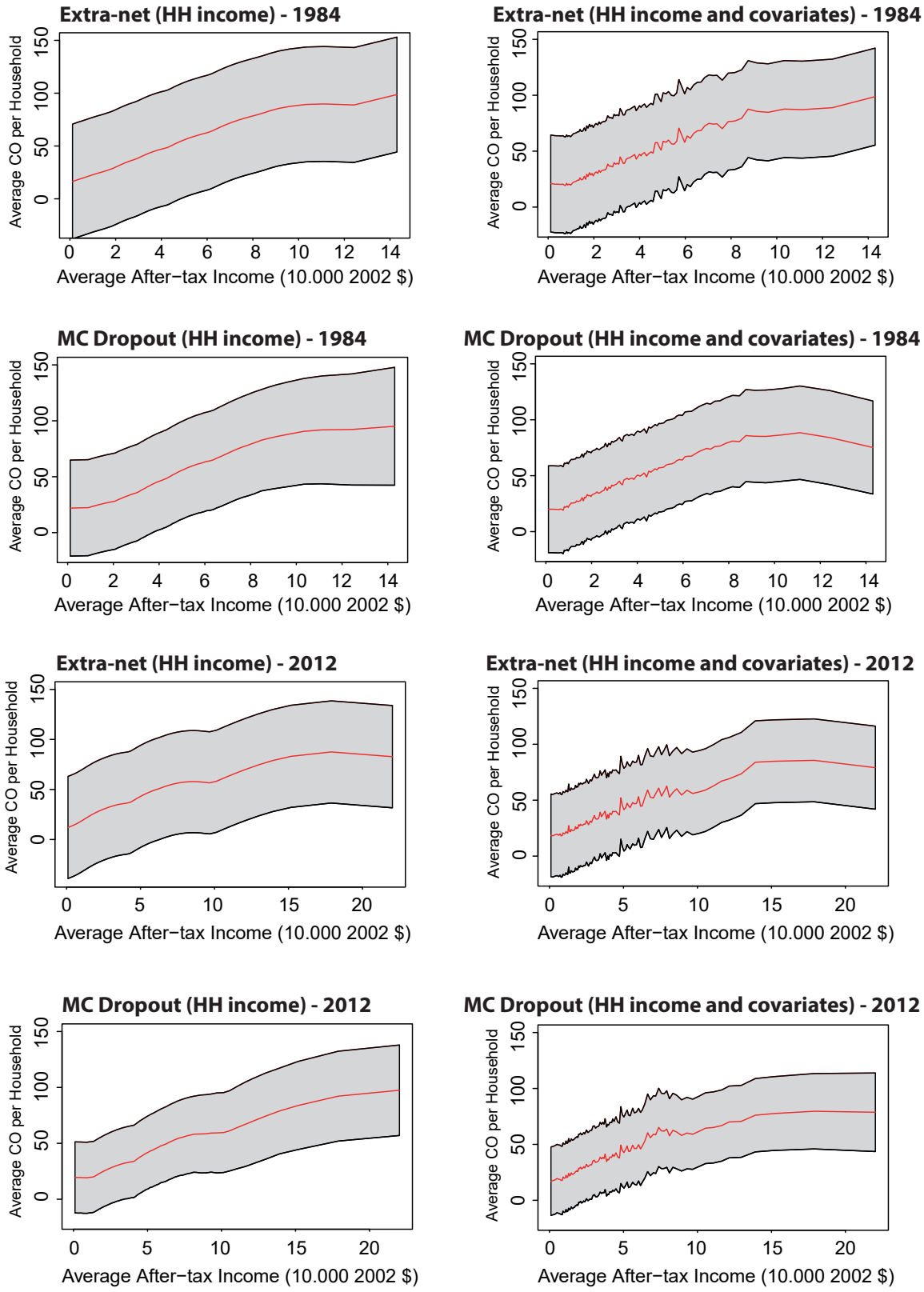


Figure 6.5: Point estimates and 0.95 prediction intervals of CO.

Figures 6.1-6.5 report the EECs constructed with both MC dropout and extra-neural networks with $d = 2$ or 18 for the five major air pollutants.

The panels in each figure are constructed as follows: the observed income is divided into 100 groups of the same size. We compute the mean of the predicted pollution and the upper and lower bounds of the prediction intervals for each group and pollutant. The mean values are then plotted. Our estimates of the EECs constructed with both MC dropout and extra-neural network provide further empirical support to the results reported in Figure 3 of Levinson and O'Brien (2019) for the PM10. In particular, the shape of the predicted curves and the associated intervals suggest an increasing and concave relationship between the variables under study. This relationship is uniform across values of household income. More formally, the upper bound of the 95% prediction interval can be used as cut-off point to determine whether the slope of the curve is greater than one or its second derivative is negative. The analysis of the prediction intervals obtained from the extra-neural network approach shows that the income elasticity is smaller than one across all values of household income. This is particularly the case for the year 2012 and holds across the five pollutants. Although the prediction intervals obtained by MC dropout are not as conclusive as those from the extra-neural nets methodology, overall, there is clear empirical evidence in support of a concave relationship between the pollution content in consumption and household income for the different pollutants considered.

These results show that – without imposing any functional form between pollution and household income – richer households pollute more, the pollution content of consumption increases at a lower rate than income, and that the pollution content of consumption grows at a decreasing rate. Interestingly, the inclusion of household characteristics in the DNN model also reveals a decreasing pattern in the relationship between household after-tax income and pollution for high income individuals for 2012. In Levinson and O'Brien (2019) this finding is only hinted for 2012 using the simple model with income and income squared but not in the model with multiple covariates. In their case the quadratic nature of the regression model determines to a large extent the overall shape of the relationship between income and pollution; however, in our setting, the model does not impose any structure on the relationship between the variables, letting the data speak. These findings suggest a decrease in the relationship between household income and pollution for top earners with respect to households in moderate to high percentiles of the distribution of income. This finding is robust across the five pollutants being particularly relevant for some pollutants such as nitrogen oxides (NO) and carbon monoxide (CO).

6.5 Conclusions

Empirical researchers have shown that, despite the economic growth that has characterized the U.S. in the past 30 years, there has been a reduction in the overall pollution. Levinson and O'Brien (2015, 2019) –by estimating EECs– are able to analyze this relationship. In particular, they show how the overall pollution in the U.S. has not increased proportionally with economic growth due to changes in the composition of U.S. households' consumption baskets towards less pollutant goods and services.

The present chapter further validates the aforementioned empirical findings by adopting deep learning techniques to estimate the EECs and associated prediction intervals. The different EECs are constructed using the MC dropout proposed by Gal and Ghahramani (2016a) and the extra-neural network algorithm implemented in chapter 4. When only income-related information is considered, a shallow neural network with 5 hidden nodes is fitted; conversely, when the wider household-specific information set is taken into account, a deep neural network with optimal architecture selected via the maximization derived in chapter 3 is considered instead. By considering both shallow and deep ReLu neural networks, the present chapter confirms the empirical findings of Levinson and O'Brien (2019): an increasing and concave relationship between after-tax household income and pollution, an elasticity lower than one, and a downward shift in the relationship when comparing 1984 and 2012. In contrast to these authors' findings, the inclusion of household characteristics in our DNN model reveals that the pollution content of consumption for households in the top of the income distribution is lower than for individuals in lower deciles of the 2012 income distribution. This finding is robust across the analysis of the five pollutants and particularly relevant for nitrogen oxides (NO) and carbon monoxide (CO).

CHAPTER 7

Conclusions

Chapter Abstract

This chapter summarizes the main findings –outlined in the present thesis– focused on Granger causality detection, optimal structure identification, and uncertainty estimation. Finally, it also provides an outlook on possible improvements of the proposed methodologies and on the possibility of extending the theoretical contributions to both convolutional and recurrent neural networks.

The present thesis, focusing on deep feedforward neural networks, contributes to the literature on deep learning and econometrics by proposing suitable methods for Granger causality detection, neural network structure identification, and uncertainty estimation in high-dimensional nonlinear systems. Following, the novel methodologies are effectively employed to study pressing environmental economics problems: the quantification of the carbon footprint associated with Bitcoin mining (or more generally, with proof-of-work validation protocols), and the analysis of the functional relationship between household income and pollution.

Chapter 2 defines a new methodology for Granger causality detection using feedforward neural networks. The proposed approach involves a two-step algorithm: at first, the optimal neural network structure, that maximizes the mutual information transfer between input and output nodes, is defined. Following, using the obtained optimal architecture, a novel objective function –that introduces sparsity in the weights connecting the input layer to the first hidden layer– is proposed to detect Granger causality between input and output variables and for the identification of the optimal lag length. The simulation study shows the importance of using the correct neural network structure for the reduction of type I and type II errors. Additionally, the simulation suggests model selection consistency for increasing sample size. The empirical application reveals that the introduction of the new blockchain platform leads to a change in the topology of the network of the analyzed energy companies. The application of the Diebold-Mariano test (1995) shows that the Granger causal network constructed with the proposed algorithm outperforms several linear VAR(K)s in terms of out-of-sample forecasting accuracy.

Chapter 3 develops an optimization method to obtain the optimal width and depth of a ReLu neural network of a given size. The proposed constrained optimization maximizes the lower bound on the maximum number of linear regions (Montufar et al., 2014) that a deep ReLu neural network can approximate. The maximization is done numerically using state-of-the-art methods such as L-BFGS-B and SLSQP algorithms. The simulation study shows how the novel procedure outperforms a k-folds cross-validation approach when the underlying data generating process is nonlinear. When the underlying data generating process is linear, a neural network with architecture obtained with the proposed optimization provides comparable mean squared error values to the ones obtained from an OLS estimator. Finally, the optimal neural network architecture is compared against state-of-the-art models using the Boston Housing dataset. Using the constrained optimization for the identification of the optimal neural network structure ensures a reduction of the out-of-sample prediction error between 28.55% and 47.32% when compared to competing machine learning algorithms.

Chapter 4 develops a suitable method for the construction of the prediction intervals for both shallow and deep neural networks. The novel methodology builds upon the work of Geurts et al. (2006) by extending the extremely randomized trees approach to ensembles of neural networks. The novel approach –by means of a fixed Bernoulli mask applied prior to training– ensures not only the correct construction of the prediction intervals but also better (compared to the naive bootstrap approach) out-of-sample accuracy by training the network

on the whole training set. The simulation results show that the MC dropout, naive bootstrap, and extra-neural network return prediction intervals with empirical coverages approximately equal to the nominal levels at which they are constructed. Additionally, the simulation results show the robustness of the extra-neural network to the choice of the dropout rate, as opposed to the MC dropout. Finally, the experimental settings of Hernández-Lobato and Adams (2015) are used to validate further the out-of-sample accuracy of the novel approach on real-world datasets. The empirical results suggest that the novel methodology outperforms competing deep learning algorithms in terms of out-of-sample RMSE.

Chapter 5 studies the carbon footprint associated with Bitcoin mining using deep learning methods. In particular, the obtained estimates show how the CO_2 emission of the proof-of-work algorithm is higher than those of (i) some U.S. states (e.g., Maine, New Hampshire, or Rhode Island), or of (ii) economies of the size of Bolivia, the Dominican Republic, or Croatia. Chapter 6 adopts both shallow and deep neural networks to estimate the ECCs and associated prediction intervals. The ECCs are constructed using not only the extra-neural network approach proposed in chapter 4 but also the MC dropout proposed by Gal and Ghahramani (2016a). The obtained ECCs show an increasing and concave relationship between after-tax household income and pollution, and a downward shift of the relationship when comparing 1984 to 2012. Thus, these results suggest that richer households pollute more, the pollution content of consumption increases at a lower rate than income, and that the pollution content of consumption decreases over time.

Future research will focus on the possibility of adopting the theoretical insights from chapter 3 to overcome some of the limitations in the methodology proposed in chapter 2. In particular, the novel methodology for Granger causality detection may be subject to a double estimation problem (two-step algorithm), it uses the tangent activation function (which can create problems in training extremely deep neural networks), and does not allow for depth selection (only width). A possible solution would be replacing the Montgomery and Eledath (1995) pruning algorithm with the constrained optimization proposed in chapter 3, and developing an iterative procedure that returns, simultaneously, the optimal neural network structure and Grager causal relations. It is important to notice that a necessary step for the implementation of the aforementioned enhancement is to change the width constraint in the optimization in chapter 3 to a depth one. Additionally, the MC dropout proposed by Gal and Ghahramani (2016a) may also be implemented to investigate the marginal effects in neural network settings.

Additionally, it is important to notice that persistence is extremely relevant in high dimensional environments and that it tends to lead to spurious outcomes (in other words, one may detect predictability when there is none). For this reason, future work will analyze – via Monte Carlo simulations– how the persistence of the data affects not only the predictive power of neural networks methods but also the probability of correcting detecting no Granger causality when there is none with the novel methodology proposed in chapter 2. This future analysis would provide also a robustness exercise to the empirical findings in chapter 5 as the

CO_2 emission data are known to be highly persistent.

Finally, one could notice that the present thesis focuses exclusively on feedforward neural networks. However, knowing that both recurrent and convolutional neural networks can be considered special cases of the more general specification of the feedforward neural network, it is possible –under appropriate adjustments– to extend the novel methodologies proposed in chapter 2, 3, and 4 to other neural network specifications. In fact, the theoretical results of chapter 2 can be extended to recurrent neural networks (mainly used for forecasting purposes); a similar procedure to the one proposed in chapter 3 could be extended to recurrent and convolutional neural networks (for the latter, Xiong et al. (2020) propose an upper bound for the number of linear regions approximated by a shallow convolutional neural network), and the extra-neural network algorithm can be applied to both recurrent and convolutional neural networks. However, it is important to make an important remark. The novel methodology for the detection of Granger causality is based on the possibility to untangle the intricate connections that define a feedforward neural network structure. Due to the presence of backward connections and of the LSTM cells in deep recurrent neural networks, it is significantly more arduous (potentially not feasible) to achieve a similar theoretical insight when focusing on recurrent neural networks.

BIBLIOGRAPHY

[1] C.C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] A. Aiuppa, T.P. Fischer, T. Plank, and P. Bani. "CO_2 flux emissions from the Earth's most actively degassing volcanoes, 2005–2015". In: *Scientific reports* 9.1 (2019), pp. 1–7.

[3] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. "Understanding deep neural networks with rectified linear units". In: *arXiv preprint arXiv:1611.01491* (2016).

[4] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. "Stronger generalization bounds for deep nets via a compression approach". In: *arXiv preprint arXiv:1802.05296* (2018).

[5] S. Athey and G.W. Imbens. "Machine learning methods that economists should know about". In: *Annual Review of Economics* 11 (2019), pp. 685–725.

[6] Global Carbon Atlas. *Global Carbon Atlas*. 2020. URL: `http://www.globalcarbonatlas.org/en/CO2-emissions` (visited on 02/01/2020).

[7] G. Aumann. "Approximation by step functions". In: *Proceedings of the American Mathematical Society* 14.3 (1963), pp. 477–482.

[8] A. Babikir and H. Mwambi. "Evaluating the combined forecasts of the dynamic factor model and the artificial neural network model using linear and nonlinear combining methods". In: *Empirical Economics* 51.4 (2016), pp. 1541–1556.

[9] B. Bakker and T. Heskes. "Clustering ensembles of neural network models". In: *Neural networks* 16.2 (2003), pp. 261–269.

[10] M. Bańbura, D. Giannone, and L. Reichlin. "Large Bayesian vector auto regressions". In: *Journal of applied Econometrics* 25.1 (2010), pp. 71–92.

[11] World Bank. "Distributed Ledger Technology (DLT) and Blockchain". In: *FinTech Note* 1 (2018), pp. 1–60.

[12] A.R. Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information theory* 39.3 (1993), pp. 930–945.

[13] A. Al Bataineh and D. Kaur. "A Comparative Study of Different Curve Fitting Algorithms in Artificial Neural Network using Housing Dataset". In: *NAECON 2018-IEEE National Aerospace and Electronics Conference*. 2018, pp. 174–178.

[14] A.J. Bell and T.J. Sejnowski. "An information-maximization approach to blind separation and blind deconvolution". In: *Neural computation* 7.6 (1995), pp. 1129–1159.

[15] A. Belloni, V. Chernozhukov, and C. Hansen. "Inference on treatment effects after selection among high-dimensional controls". In: *The Review of Economic Studies* 81.2 (2014), pp. 608–650.

[16] C. Bendiksen and S. Gibbons. "The Bitcoin mining network: Trends, Average Creation Costs, Electricity Consumption & Sources". In: *CoinShare* (2019).

[17] Y. Bengio. "Learning deep architectures for AI". In: *Foundations and trends in Machine Learning* 2.1 (2009), pp. 1–127.

[18] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[19] Y. Benjamini and Y. Hochberg. "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the Royal statistical society: series B (Methodological)* 57.1 (1995), pp. 289–300.

[20] B.S. Bernanke, J. Boivin, and P. Eliasz. "Measuring the effects of monetary policy: a factor-augmented vector autoregressive (FAVAR) approach". In: *The Quarterly journal of economics* 120.1 (2005), pp. 387–422.

[21] D. Bertsimas and J. Dunn. "Optimal classification trees". In: *Machine Learning* 106.7 (2017), pp. 1039–1082.

[22] F. Bianchi, H. Mumtaz, and P. Surico. "The great moderation of the term structure of UK interest rates". In: *Journal of Monetary Economics* 56.6 (2009), pp. 856–871.

[23] M. Billio, M. Getmansky, A.W. Lo, and L. Pelizzon. "Econometric measures of connectedness and systemic risk in the finance and insurance sectors". In: *Journal of financial economics* 104.3 (2012), pp. 535–459.

[24] Bitmain. *Application Proof of Bitmain*. 2018. URL: `https://templatelab.com/bitmain-ipo-prospectus/` (visited on 02/01/2020).

[25] F. Bloch, M.O. Jackson, and P. Tebaldi. *Centrality measures in networks*. 2017. URL: `AvailableatSSRN2749124` (visited on 06/01/2019).

[26] J. Boivin, M.T. Kiley, and F.S. Mishkin. "How has the monetary transmission mechanism evolved over time?" In: *Handbook of monetary economics*. Elsevier. 2010, pp. 369–422.

[27] P. Bonancich. "Power and centrality: A family of measures". In: *American journal of sociology* 92.5 (1987), pp. 1170–1182.

[28] J. Bouoiyour and R. Selmi. "The Bitcoin price formation: Beyond the fundamental sources". In: *arXiv preprint arXiv:1707.01284* (2017).

[29] G.E. Box and G.C. Tiao. "A canonical analysis of multiple time series". In: *Biometrika* 64.2 (1977), pp. 355–365.

[30] M. Brander, A. Sood, C. Wylie, A. Haughton, and J. Lovell. "Technical Paper| Electricity-specific emission factors for grid electricity". In: *Ecometrica, Emissionfactors.com.* (2011).

[31] L. Breiman. "Random Forest". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[32] D. Brezak, T. Bacek, D. Majetic, J. Kasac, and B. Novakovic. "A comparison of feed-forward and recurrent neural networks in time series forecasting". In: *2012 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*. IEEE. 2012, pp. 1–6.

[33] H. Brezis. *Functional analysis, Sobolev spaces and partial differential equations*. Springer Science & Business Media, 2010.

[34] G. Brown, J.L. Wyatt, and P. Tiňo. "Managing diversity in regression ensembles". In: *Journal of machine learning research* 6 (2005), pp. 1621–1650.

[35] D. Burtraw, R. Sweeney, and M. Walls. "The incidence of US climate policy: alternative uses of revenues from a cap-and-trade auction". In: *National Tax Journal* (2009), pp. 497–518.

[36] L.A. Callot and A.B. Kock. "Oracle efficient estimation and forecasting with the adaptive lasso and the adaptive group lasso in vector autoregressions". In: *Essays in Nonlinear Time Series Econometrics* (2014), pp. 238–268.

[37] Cambridge. *Cambridge Bitcoin Electricity Consumption Index*. 2020. URL: `https://www.cbeci.org/` (visited on 07/01/2020).

[38] Canaan. *Form F-1 Registration Statement*. 2019. URL: `https://www.sec.gov/Archives/edgar/data/1780652/000119312519276263/d773846df1.htm\#rom773846_14` (visited on 02/01/2020).

[39] J.G. Carney, P. Cunningham, and U. Bhagwan. "Confidence and prediction intervals for neural network ensembles". In: *IJCNN'99 International Joint Conference on Neural Networks. Proceedings*. 1999, pp. 1215–1218.

[40] K. Chakraborty, K. Mehrotra, C.K. Mohan, and S. Ranka. "Forecasting the behavior of multivariate time series using neural networks". In: *Neural networks* 5.6 (1992), pp. 961–970.

[41] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. "Recurrent neural networks for multivariate time series with missing values". In: *Scientific reports* 8.1 (2018), pp. 1–12.

[42] X. Chen and H. White. "Improved rates and asymptotic normality for nonparametric neural network estimators". In: *IEEE Transactions on Information Theory* 45.2 (1999), pp. 682–691.

[43] A. Chinco, A.D. Clark-Joseph, and M. Ye. "Sparse signals in the cross-section of returns". In: *The Journal of Finance* 74.1 (2019), pp. 449–492.

[44] L.J. Christiano, M. Eichenbaum, and C.L. Evans. "Monetary policy shocks: What have we learned and to what end?" In: *Handbook of macroeconomics* 1 (1999), pp. 65–148.

[45] P. Ciaian, M. Rajcaniova, and D.A. Kancs. "The economics of BitCoin price formation". In: *Applied Economics* 48.19 (2016), pp. 1799–1851.

[46] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. "Multi-column deep neural network for traffic sign classification". In: *Neural networks* 32 (2012), pp. 333–338.

[47] L. Cocco and M. Marchesi. "Modeling and Simulation of the Economics of Mining in the Bitcoin Market". In: *PloS one* 11.10 (2016), e.0164603.

[48] J.T. Connor, R.D. Martin, and L.E. Atlas. "Recurrent neural networks and robust time series prediction". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 240–254.

[49] B. Copeland and M.S. Taylor. *Trade and the Environment: Theory and Evidence*. Princeton (NJ): Princeton University Press, 2005.

[50] I. Cortes-Ciriano and A. Bender. "Reliable prediction errors for deep neural networks using test-time dropout". In: *Journal of Chemical Information and Modeling* 59.7 (2019), pp. 3330–3339.

[51] T.M. Cover and J.A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[52] K. Cuthbertson and D. Nitzsche. *Financial engineering: derivatives and risk management*. Chichester, UK: John Wiley & Sons, 2014.

[53] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[54] R.D. De-vieaux, J. Schumi, J. Schweinsberg, and L.H. Ungar. "Prediction intervals for neural networks via nonlinear regression". In: *Technometrics* 40.4 (1998), pp. 273–282.

[55] M. Denil, B. Shakibi, L. Dinh, M.A. Ranzato, and N. De Freitas. "Predicting parameters in deep learning". In: *Advances in neural information processing systems*. 2013, pp. 2148–2156.

[56] J.S. Denker and Y. LeCun. "Transforming neural-net output levels to probability distributions". In: *Advances in neural information processing systems*. 1991, pp. 853–859.

[57] G. DeVeciana and A. Zakhor. "Neural net based continuous phase modulation receivers". In: *IEEE transactions on communications* 40.8 (1992), pp. 1396–1408.

[58] A. DeVries. "Bitcoin's energy consumption is underestimated: A market dynamics approach". In: *Energy Research & Social Science* 70 (2020), p. 101721.

[59] A. DeVries. "Bitcoin's growing energy problem". In: *Joule* 2.5 (2018), pp. 801–805.

[60] A. DeVries. "Renewable Energy Will Not Solve Bitcoin's Sustainability Problem". In: *Joule* 3.4 (2019), pp. 893–898.

[61] F.X. Diebold. *Elements of forecasting*. South-Western College Pub, 1998.

[62] F.X. Diebold and R.S. Mariano. "Comparing Predictive Accuracy". In: *Journal of Business & Economic Statistics* 13 (1995), pp. 253–263.

[63] H.D. Dipu-Kabir, A. Khosravi, M.A. Hosen, and S. Nahavandi. "Neural network-based uncertainty quantification: A survey of methodologies and applications". In: *IEEE access* 6 (2018), pp. 36218–36234.

[64] L. Dittmar and A. Praktiknjo. "Could Bitcoin emissions push global warming above 2°C?" In: *Nature Climate Change* 9.9 (2019), pp. 656–657.

[65] J.M. Dufour and A. Taamouti. "Short and long run causality measures: Theory and inference". In: *Journal of Econometrics* 145.1 (2010), pp. 42–58.

[66] A.H. Dyhrberg. "Hedging capabilities of bitcoin. Is it the virtual gold?" In: *Finance Research Letters* 16 (2016), pp. 139–144.

[67] Ebang. *Form F-1 Registration Statement*. 2020. URL: `https://www.sec.gov/Archives/edgar/data/1799290/000121390020010071/ea121021-f1_ebanginter.htm\#a_013` (visited on 02/01/2020).

[68] K. Eckle and J. Schmidt-Hieber. "A comparison of deep networks with ReLU activation function and linear spline-type methods". In: *Neural Networks* 110 (2019), pp. 232–242.

[69] B. Efron. "Bootstrap methods: another look at the jackknife". In: *Annals of Statistics* 7.1 (1979), pp. 1–26.

[70] S. Eggleston, L. Buendia, K. Miwa, T. Ngara, and K. Tanabe. *2006 IPCC guidelines for national greenhouse gas inventories (Vol. 2)*. Hayama, Japan: Institute for Global Environmental Strategies, 2006.

[71] M. Eichler. "Granger causality and path diagrams for multivariate time series". In: *Journal of Econometrics* 137.2 (2007), pp. 334–353.

[72] M. Eichler and V. Didelez. "Causal reasoning in graphical time series models". In: *arXiv preprint arXiv:1206.5246* (2012).

[73] S. El-Hihi and Y. Bengio. "Hierarchical recurrent neural networks for long-term dependencies". In: *Nips (Vol. 409)*. 1995.

[74] E. Engel. "Das Lebenskosten belgischer Arbeiterfamilien fruher und jetzt". In: *Bulletin de Institut International de Statistique* 9 (1895), pp. 1–124.

[75] R. Errouissi, J. Cardenas-Barrera, J. Meng, E. Castillo-Guerra, X. Gong, and L. Chang. "Bootstrap prediction interval estimation for wind speed forecasting". In: *2015 IEEE Energy Conversion Congress and Exposition (ECCE)*. 2015, pp. 1919–1924.

[76] J. Fan and R. Li. "Variable selection via nonconcave penalized likelihood and its oracle properties". In: *Journal of the American statistical Association* 96.456 (2001), pp. 1348–1360.

[77] M.H. Farrell, T. Liang, and S. Misra. "Deep neural networks for estimation and inference". In: *arXiv preprint arXiv:1809.09953* (2019).

[78] M.H. Farrell, T. Liang, and S. Misra. "Deep neural networks for estimation and inference: Application to causal effects and other semiparametric estimands". In: *arXiv preprint arXiv:1809.09953* (2018).

[79] M. Forni and L. Gambetti. "The dynamic effects of monetary policy: A structural factor model approach". In: *Journal of Monetary Economics* 57.2 (2010), pp. 203–216.

[80] M. Forni, M. Hallin, M. Lippi, and L. Reichlin. "The generalized dynamic factor model consistency and rates". In: *Journal of Econometrics* 119.2 (2004), pp. 231–255.

[81] M. Forni, M. Hallin, M. Lippi, and L. Reichlin. "The generalized dynamic-factor model: Identification and estimation". In: *Review of Economics and statistics* 82.4 (2000), pp. 540–554.

[82] S. Foteinis. "Bitcoin's alarming carbon footprint". In: *Nature* 554.7691 (2018), pp. 169–169.

[83] Energy Web Foundation. *The Energy Web Chain: Accelerating the Energy Transition with an Open-Source, Decentralized Blockchain Platform*. 2018. URL: `http://www.energyweb.org/papers/the-energy-web-chain` (visited on 02/01/2019).

[84] J.H. Friedman. "An overview of predictive learning and function approximation". In: *From statistics to neural networks*. 1994, pp. 1–61.

[85] Glaucoma Research Fundation. *Five Common Glaucoma Tests*. 2020. URL: `https://www.glaucoma.org/glaucoma/diagnostic-tests.php` (visited on 10/01/2020).

[86] Y. Gal and Z. Ghahramani. "Bayesian convolutional neural networks with Bernoulli approximate variational inference". In: *arXiv preprint arXiv:1506.02158* (2016b).

[87] Y. Gal and Z. Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *International conference on machine learning*. 2016a, pp. 1050–1059.

[88] Y. Gal and Z. Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *Github repository: https://github.com/yaringal/DropoutUncert* Accessed [Online]: 02/10/2020 (2016c).

[89] D. Garcia, C.J. Tessone, P. Mavrodiev, and N. Perony. "The digital traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin economy". In: *Journal of the Royal Society Interface* 11.99 (2014), p. 20140623.

[90] A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2017.

[91] P. Geurts, D. Ernst, and L. Wehenkel. "Extremely randomized trees". In: *Machine learning* 63.1 (2006), pp. 3–42.

[92] J. Geweke. "Inference and causality in economic time series models". In: *Handbook of econometrics* 2 (1984), pp. 1101–1144.

[93] J. Geweke. "Measurement of linear dependence and feedback between multiple time series". In: *Journal of the American statistical association* 77.378 (1982), pp. 304–313.

[94] J. Geweke. "The dynamic factor analysis of economic time series". In: *Latent variables in socio-economic models*. 1977.

[95] O.W. Gilley and R.K. Pace. "On the Harrison and Rubinfeld data". In: *Journal of Environmental and Economic Managemen* 31 (1996), pp. 403–405.

[96] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteen international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[97] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. Cambridge: MIT press, 2016.

[98] I.J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. "Multi-digit number recognition from street view imagery using deep convolutional neural networks". In: *arXiv preprint arXiv:1312.6082* (2013a).

[99] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[100] I.J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. "Maxout networks". In: *arXiv preprint arXiv:1302.4389* (2013b).

[101] C.A. Grainger and C.D. Kolstad. "Who Pays a Price on Carbon?" In: *Environmental and Resource Economics* 46 (2010), pp. 359–376.

[102] C.W. Granger. "Investigating causal relations by econometric models and cross-spectral methods". In: *Econometrica: Journal of the Econometric Society* 37.3 (1969), pp. 424–438.

[103] P.M. Granitto, P.F. Verdes, H.D. Navone, and H.A. Ceccatto. "A late-stopping method for optimal aggregation of neural networks". In: *International journal of neural systems* 11.3 (2001), pp. 305–310.

[104] A. Graves. "Practical variational inference for neural networks". In: *Advances in neural information processing systems*. 2011, pp. 2348–2356.

[105] G. Grossman and A. Krueger. "Economic Growth and the Environment". In: *Quarterly Journal of Economics* 110 (1995), pp. 353–377.

[106] S. Gu, B. Kelly, and D. Xiu. "Empirical asset pricing via machine learning". In: *The Review of Financial Studies* 33.5 (2020), pp. 2223–2273.

[107] L.F. Guilhoto. *An Overview Of Artificial Neural Networks for Mathematicians*. 2018. URL: `http://math.uchicago.edu/~may/REU2018/REUPapers/Guilhoto.pdf` (visited on 02/01/2019).

[108] Y. Hagiwara, J.E.W. Koh, J.H. Tan, S.V. Bhandary, A. Laude, E.J. Ciaccio, L. Tong, and U.R. Acharya. "Computer-aided diagnosis of glaucoma using fundus images: A review". In: *Computer methods and programs in biomedicine* 165 (2018), pp. 1–12.

[109] B. Hanin and M. Sellke. "Approximating continuous functions by relu nets of minimal width". In: *arXiv preprint arXiv:1710.11278* (2017).

[110] J.V. Hansen and R.D. Nelson. "Forecasting and recombining time-series components by using neural networks". In: *Journal of the Operational Research Society* 54 (2003), pp. 307–317.

[111] M. Harding and J. Hersh. "Big Data in economics". In: *IZA World of Labor* (2018).

[112] D. Harrison and D.L. Rubinfeld. "Hedonic housing prices and the demand forclean air". In: *Journal of Environmental Economics and Management* 5 (1978), pp. 81–102.

[113] K. Hassett, A. Mathur, and G. Metcalf. "The Incidence of a U.S. Carbon Tax: A Lifetime and Regional Analysis". In: *Energy Journal* 30 (2009), pp. 155–177.

[114] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Berlin, Germany: Springer, 2005.

[115] A.S Hayes. "Cryptocurrency value formation: An empirical study leading to a cost of production model for valuing bitcoin". In: *Telematics and Informatics* 34.7 (2017), pp. 1308–1321.

[116] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE. 2016, pp. 770–778.

[117] K. He, X. Zhang, S. Ren, and J. Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[118] A. Hecq, J.P Jacobs, and M.P. Stamatogiannis. "Testing for news and noise in non-stationary time series subject to multiple historical revisions". In: *Journal of Macroeconomics* 60 (2019), pp. 369–407.

[119] J.M. Hernández-Lobato and R. Adams. "Probabilistic backpropagation for scalable learning of bayesian neural networks". In: *International Conference on Machine Learning*. 2015, pp. 1861–1869.

[120] S. Herzog, C. Tetzlaff, and F. Wörgötter. "Transfer entropy-based feedback improves performance in artificial neural networks". In: *arXiv preprint arXiv:1706.04265* (2017).

[121] T. Heskes. "Practical confidence and prediction intervals". In: *Advances in neural information processing systems*. 1997, pp. 176–182.

[122] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[123] S.L. Ho, M. Xie, and T.N. Goh. "A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction". In: *Computers & Industrial Engineering* 42.2–4 (2002), pp. 371–375.

[124] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[125] C. Hope, J. Anderson, and P. Wenman. "Policy analysis of the greenhouse effect: an application of the PAGE mode". In: *Energy Policy* 21.3 (1993), pp. 327–338.

[126] K. Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[127] N. Houy. "Rational mining limits Bitcoin emissions". In: *Nature Climate Change* 9.9 (2019), pp. 655–655.

[128] H. Hruschka. "Determining market response functions by neural network modeling: a comparison to econometric techniques". In: *European Journal of Operational Research* 66.1 (1993), pp. 27–35.

[129] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun. "Deep learning for image based cancer detection and diagnosis: a survey. Pattern Recognition". In: *Pattern Recognition* 83 (2018), pp. 134–149.

[130] D.H. Hubel. "Single unit activity in striate cortex of unrestrained cats". In: *The Journal of physiology* 147.2 (1959), p. 226.

[131] E. Hüllermeier and W. Waegeman. "Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction". In: *arXiv preprint arXiv:1910.09457* (2020).

[132] J.G. Hwang and A.A. Ding. "Prediction intervals for artificial neural networks". In: *Journal of the American Statistical Association* 92.438 (1997), pp. 748–757.

[133] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning: with application in R*. New York: Springer, 2013.

[134] H. Jang and J. Lee. "An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information". In: *IEEE Access* 6 (2017), pp. 5427–5437.

[135] K. Jarrett, K. Kavukcuoglu, M.A. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 2146–2153.

[136] J.S. Judd. *'Neural network design and the complexity of learning*. MIT press, 1990.

[137] K. Jurado, S.C. Ludvigson, and S. Ng. "Measuring uncertainty". In: *American Economic Review* 105.3 (2015), pp. 1177–1216.

[138] I. Kaastra and M. Boyd. "Designing a neural network for forecasting financial and economic time series". In: *Neurocomputing* 10.3 (1996), pp. 215–236.

[139] T. Kaji, E. Manresa, and G. Pouliot. *Deep Inference: Artificial Intelligence for Structural Estimation*. 2018. URL: `https://events.barcelonagse.eu/live/files/2773-elenamanresa66433pdf` (visited on 02/01/2019).

[140] N. El Karoui and E. Purdom. "Can we trust the bootstrap in high-dimensions? the case of linear models". In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 170–235.

[141] A. Kendall and Y. Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Advances in neural information processing systems*. 2017, pp. 5574–5584.

[142] D.E. Kim and M. Gofman. "Comparison of shallow and deep neural networks for network intrusion detection". In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2018, pp. 204–208.

[143] D.P. Kingma, T. Salimans, and M. Welling. "Variational dropout and the local reparameterization trick". In: *Advances in neural information processing systems*. 2015, pp. 2575–2583.

[144] J. Kleinberg, J. Ludwig, S. Mullainathan, and Z. Obermeyer. "Prediction policy problems". In: *American Economic Review* 105.5 (2015), pp. 491–495.

[145] R. Kohn, C.F. Ansley, and D. Tharm. "The performance of cross-validation and maximum likelihood estimators of spline smoothing parameters". In: *Journal of the american statistical association* 86.416 (1991), pp. 1042–1050.

[146] G.M. Koop. "Forecasting with medium and large Bayesian VARs". In: *Journal of Applied Econometrics* 28.2 (2013), pp. 177–203.

[147] P. Kotzanikolaou, M. Theoharidou, and D. Gritzalis. "Assessing n-order dependencies between critical infrastructures". In: *International Journal of Critical Infrastructures* 2,9.1-2 (2013), pp. 93–110.

[148] A. Kraskov, H. Stögbauer, and P. Grassberger. "Estimating mutual information". In: *Physical review E* 69.6 (2004), pp. 1–16.

[149] M. Kraus, S. Feuerriegel, and A. Oztekin. "Deep learning in business analytics and operations research: Models, applications and managerial implications". In: *European Journal of Operational Research* 281.3 (2020), pp. 628–641.

[150] L. Kristoufek. "What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis". In: *PloS one* 10.4 (2015), pp. 1–15.

[151] A. Krizhevsky, I. Sutskever, and G.E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[152] A. Krogh and J. Vedelsby. "Neural network ensembles, cross validation, and active learning". In: *Advances in neural information processing systems*. 1995, pp. 231–238.

[153] M. Kull and P. Flach. "Reliability maps: A tool to enhance probability estimates and improve classification accuracy". In: *Proc. ECML/PKDD, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2014, pp. 18–33.

[154] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in neural information processing systems*. 2017, pp. 6402–6413.

[155] A. Lambrou, H. Papadopoulos, and A. Gammerman. "Reliable confidence measures for medical diagnosis with evolutionary algorithms". In: *IEEE Transactions on Information Technology in Biomedicine* 15.1 (2011), pp. 93–99.

[156] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. "An empirical evaluation of deep architectures on problems with many factors of variation". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 473–480.

[157] S. Lawrence, C.L. Giles, and A.C. Tsoi. "Convolutional neural networks for face recognition". In: *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 1996, pp. 217–222.

[158] S. Lawrence, C.L. Giles, A.C. Tsoi, and A.D. Back. "Face recognition: A convolutional neural-network approach". In: *IEEE transactions on neural networks* 8.1 (1997), pp. 98–113.

[159] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[160] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[161] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra. "Why M heads are better than one: Training a diverse ensemble of deep networks". In: *arXiv preprint arXiv:1511.06314* (2015).

[162] T.H. Lee, H. White, and C.W. Granger. "Testing for neglected nonlinearity in time series models: A comparison of neural network methods and alternative tests". In: *Journal of Econometrics* 56.3 (1993), pp. 269–290.

[163] H. Leeb and B.M. Pötscher. "Model selection and inference: Facts and fiction". In: *Econometric Theory* 21.1 (2005), pp. 21–59.

[164] E.M. Leeper, C.A. Sims, T. Zha, R.E. Hall, and B.S. Bernanke. "What does monetary policy do?" In: *Brookings papers on economic activity* 2 (1996), pp. 1–78.

[165] M. Leshno, V.Y. Lin, A. Pinkus, and S. Schocken. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks* 6.6 (1993), pp. 861–867.

[166] L.P. Levasseur, Y.D. Hezaveh, and R.H. Wechsler. "Uncertainties in parameters estimated with neural networks: Application to strong gravitational lensing". In: *The Astrophysical Journal Letters* 850.1 (2017), pp. 1–7.

[167] A. Levinson and J. O'Brien. "Environmental engel curves". In: *National Bureau of Economic Research* w20914 (2015).

[168] A. Levinson and J. O'Brien. "Environmental Engel curves: Indirect emissions of common air pollutants". In: *Review of Economics and Statistics* 101.1 (2019), pp. 121–133.

[169] E.P. Lim, H. Chen, and G. Chen. "Business intelligence and analytics: Research directions". In: *ACM Transactions on Management Information Systems (TMIS)* 3.4 (2013), pp. 1–10.

[170] P. Liu, X. Qiu, and X. Huang. "Recurrent neural network for text classification with multi-task learning". In: *arXiv preprint arXiv:1605.05101* (2017).

[171] Y. Liu and A. Tsyvinski. "Risks and returns of cryptocurrency". In: *National Bureau of Economic Research* w24877 (2018).

[172] A.C. Lozano, N. Abe, Y. Liu, and S. Rosset. "Grouped graphical Granger modeling for gene expression regulatory networks discovery". In: *Bioinformatics* 25.12 (2009), pp. 110–118.

[173] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. "The expressive power of neural networks: A view from the width". In: *Advances in neural information processing systems*. 2017, pp. 6231–6239.

[174] H. Lütkepohl. "Comparison of criteria for estimating the order of a vector autoregressive process". In: *Journal of time series analysis* 6.1 (1985), pp. 35–52.

[175] S.I. Maeda. "A Bayesian encourages dropout". In: *arXiv preprint arXiv:1412.7003* (2014).

[176] H. Markowitz. "Portfolio Selection". In: *The Journal of Finance* 7 (1952), pp. 77–91.

[177] E. Masanet, A. Shehabi, N. Lei, H. Vranken, J. Koomey, and J. Malmodin. "Implausible projections overestimate near-term Bitcoin CO_2 emissions". In: *Nature Climate Change* 9.9 (2019), pp. 653–654.

[178] W.S. McCulloch. "Why the mind is in the head". In: *Embodiments of mind* (1965), pp. 72–141.

[179] S. McNally, J. Roche, and S. Caton. "Predicting the price of bitcoin using machine learning". In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE. 2018, pp. 339–343.

[180] S. Mei, A. Montanari, and P.M. Nguyen. "A mean field view of the landscape of two-layer neural networks". In: *Proceedings of the National Academy of Sciences* 115.33 (2018), pp. 7665–7671.

[181] I. Méndez-Jiménez and M. Cárdenas-Montes. "Time series decomposition for improving the forecasting performance of convolutional neural networks". In: *Conference of the Spanish Association for Artificial Intelligence*. Springer. 2018, pp. 87–97.

[182] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, and P. Vincent. "Unsupervised and transfer learning challenge: a deep learning approach". In: *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop*. 27. 2011, pp. 97–111.

[183] G. Metcalf. "A Distributional Analysis of Green Tax Reforms". In: *National Tax Journal* 52 (1999), pp. 655–682.

[184] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur. "Extensions of recurrent neural network language model". In: *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2011, pp. 5528–5531.

[185] Asic Miner. *Asic Miner Index*. 2020. URL: `https://asic-dex.com/` (visited on 02/01/2020).

[186] M.C. Montgomery and J.K. Eledath. *Maximum Information Transfer in Feedforward Neural Networks*. 1995. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.330\&rep=rep1\&type=pdf` (visited on 11/01/2018).

[187] G.F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. "On the number of linear regions of deep neural networks". In: *Advances in neural information processing systems*. 2014, pp. 2924–2932.

[188] C. Mora, R.L. Rollins, K. Taladay, M.B. Kantar, M.K. Chock, M. Shimada, and E.C. Franklin. "Bitcoin emissions alone could push global warming above 2 C". In: *Nature Climate Change* 8.11 (2018), pp. 931–933.

[189] J.P. Morgan. "Blockchain and the decentralization revolution". In: *J.P. Morgan white paper* (2018), pp. 1–21.

[190] M.J. Mortenson, N.F. Doherty, and S. Robinson. "Operational research from Taylorism to Terabytes: A research agenda for the analytics age". In: *European Journal of Operational Research* 241.3 (2015), pp. 583–595.

[191] S. Mullainathan and J. Spiess. "Machine learning: an applied econometric approach". In: *Journal of Economic Perspectives* 31.2 (2017), pp. 87–106.

[192] P. Myshkov and S. Julier. "Posterior distribution analysis for bayesian inference in neural networks". In: *Workshop on Bayesian Deep Learning*. 2016.

[193] Z. Nado, J. Snoek, R. Grosse, D. Duvenaud, B. Xu, and J. Martens. "Stochastic gradient langevin dynamics that exploit neural network structure". In: *International Conference on Learning Representations 2018 (Workshop)*. 2018.

[194] V. Nair and G.E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Icml*. 2010.

[195] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: `https://bitcoin.org/bitcoin.pdf` (visited on 11/01/2018).

[196] W.K. Newey and K.D. West. "A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix". In: *Econometrica* 55 (1987), pp. 703–708.

[197] W.B. Nicholson, I. Wilms, J. Bien, and D.S. Matteson. "High Dimensional Forecasting via Interpretable Vector Autoregression". In: *arXiv preprint arXiv:1412.5250* (2014).

[198] D.A. Nix and A.S. Weigend. "Estimating the mean and variance of the target probability distribution". In: *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*. 1994, pp. 55–60.

[199] R. Novak, Y. Bahri, D.A. Abolafia, J. Pennington, and J. Sohl-Dickstein. "Sensitivity and generalization in neural networks: an empirical study". In: *arXiv preprint arXiv:1802.08760* (2018).

[200] B. Oancea and S.C. Ciucu. "Time series forecasting using neural networks". In: *arXiv preprint arXiv:1401.1333* (2014).

[201] D.R. Pant, P. Neupane, A. Poudel, A.K. Pokhrel, and B.K. Lama. "Recurrent neural network based bitcoin price prediction by twitter sentiment analysis". In: *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. IEEE. 2018, pp. 128–132.

[202] H. Papadopoulos and H. Haralambous. "Reliable prediction intervals with regression neural networks". In: *Neural Networks* 24.8 (2011), pp. 842–851.

[203] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z.B. Celik, and A. Swami. "Practical black-box attacks against machine learning". In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.

[204] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. "How to construct deep recurrent neural networks". In: *arXiv preprint arXiv:1312.6026* (2014).

[205] R. Pascanu, G. Montufar, and Y. Bengio. "On the number of response regions of deep feed forward networks with piece-wise linear activations". In: *arXiv preprint arXiv:1312.6098* (2013).

[206] K. Pasupa and W. Sunhem. "A comparison between shallow and deep architecture classifiers on small dataset". In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE. 2016, pp. 1–6.

[207] R. Paul, S.H. Hawkins, L.O. Hall, D.B. Goldgof, and R.J. Gillies. "Combining deep neural network and traditional image features to improve survival prediction accuracy for lung cancer patients from diagnostic CT". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016, pp. 2570–2575.

[208] T. Pearce, A. Brintrup, M. Zaki, and A. Neely. "High-quality prediction intervals for deep learning: A distribution-free, ensembled approach". In: *International Conference on Machine Learning*. 2018, pp. 4075–4084.

[209] D. Pena and G.E. Box. "Identifying a simplifying structure in time series". In: *Journal of the American statistical Association* 82.399 (1987), pp. 836–843.

[210] M. Plagborg-Møller and C.K. C.K. Wolf. "Local projections and VARs estimate the same impulse responses". In: *Unpublished paper: Department of Economics, Princeton University* (2019).

[211] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review". In: *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519.

[212] Global Petrol Prices. *Global Petrol Prices*. 2020. URL: `https://www.globalpetrolprices.com/` (visited on 02/01/2020).

[213] F. Qiang, H. Shang-Xu, and Z. Sheng-Ying. "Clustering-based selective neural network ensemble". In: *Journal of Zhejiang University-Science A* 6.5 (2005), pp. 387–392.

[214] U. Raghavendra, H. Fujita, S.V. Bhandary, A. Gudigar, J.H. Tan, and U.R. Acharya. "Deep convolution neural network for accurate diagnosis of glaucoma using digital fundus images". In: *Information Sciences* 441 (2018), pp. 41–49.

[215] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. "On the expressive power of deep neural networks". In: *international conference on machine learning*. 2017, pp. 2847–2854.

[216] J.C. Ranyard, R. Fildes, and T.I. Hu. "Reassessing the scope of OR practice: The influences of problem structuring methods and the analytics movement". In: *European Journal of Operational Research* 245.1 (2015), pp. 1–13.

[217] R.B. Rao, G. Fung, and R. Rosales. "On the dangers of cross-validation. An experimental evaluation". In: *Proceedings of the 2008 SIAM international conference on data mining*. Society for Industrial and Applied Mathematics. 2008, pp. 588–596.

[218] R. Reed, R.J. Marks, and S. Oh. "Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter". In: *IEEE Transactions on Neural Networks* 6.3 (1995), pp. 529–538.

[219] R. Reed, S. Oh, and R.J. Marks. "Regularization using jittered training data". In: *In [Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. IEEE. 1992, pp. 147–152.

[220] F.A. Rodrigues. "Network centrality: an introduction". In: *arXiv preprint arXiv:1901.07901v1* (2019).

[221] B. Santra, A. Paul, and D.P. Mukherjee. "Deterministic dropout for deep neural networks using composite random forest". In: *Pattern Recognition Letters* 131 (2020), pp. 205–212.

[222] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. "Group sparse regularization for deep neural networks". In: *Neurocomputing* 241 (2017), pp. 81–89.

[223] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[224] J. Schmidt-Hieber. "Nonparametric regression using deep neural networks with ReLU activation function". In: *arXiv preprint arXiv:1708.06633* (2017).

[225] T. Schreiber. "Measuring information transfer". In: *Physical review letters* 85.2 (2000), pp. 461–465.

[226] Medicine National Academies of Sciences Engineering. *Valuing climate damages: updating estimation of the social cost of carbon dioxide*. National Academies Press, 2017.

[227] G.A.F. Seber and C.J. Wild. *Nonlinear regression*. New York: Wiley, 1989.

[228] R. Senge, S. Bösner, K. Dembczynski, J. Haasenritter, O. Hirsch, N. Donner-Banzhohh, and E. Hüllermeier. "Reliable classification that distinguish aleatoric and epsitemic uncertainty". In: *Information Sciences* 255 (2014), pp. 16–19.

[229] C. Serpell, I. Araya, C. Valle, and H. Allende. "Probabilistic Forecasting Using Monte Carlo Dropout Neural Networks". In: *Iberoamerican Congress on Pattern Recognition*. 2019, pp. 387–397.

[230] M.H. Shaker and E. Hüllermeier. "Aleatoric and epistemic uncertainty with random forests". In: *International Symposium on Intelligent Data Analysis*. Springer, Cham. 2020.

[231] S.S. Shapiro and M.B. Wilk. "An analysis of variance test for normality (complete samples)". In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[232] L. Shen, L.R. Margolies, J.H. Rothstein, E. Fluder, R. McBride, and W. Sieh. "Deep learning to improve breast cancer detection on screening mammography". In: *Scientific reports* 9.1 (2019), pp. 1–12.

[233] Shodan.io. *IoT-search engine*. 2020. URL: `https://www.shodan.io/` (visited on 02/01/2020).

[234] R. Shwartz-Ziv and N. Tishby. "Opening the black box of deep neural networks via information". In: *arXiv preprint arXiv:1703.00810* (2017).

[235] C.P. Simon and L. Blume. *Mathematics for economists*. New York: Norton, 1994.

[236] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. "A sparse-group Lasso". In: *Journal of Computational and Graphical Statistics* 22.2 (2013), pp. 231–245.

[237] C.A. Sims. "Comment on Glenn Rudebusch's" Do measures of monetary policy in a VAR make sense?"". In: *International Economic Review* 39.4 (1998), pp. 933–941.

[238] C.A. Sims. "Macroeconomics and reality". In: *Econometrica: journal of the Econometric Society* 48.1 (1980), pp. 1–48.

[239] J. Sirignano and K. Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of computational physics* 375 (2018), pp. 1339–1364.

[240] A. Skripnikov and G. Michailidis. "Regularized joint estimation of related vector autoregressive models". In: *Computational statistics & data analysis* 139 (2019), pp. 164–177.

[241] A. Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. 1776.

[242] S. Smyl. "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting". In: *International Journal of Forecasting* 36 (2020), pp. 75–85.

[243] S. Song and P.J. Bickel. "Large vector auto regressions". In: *arXiv preprint arXiv:1106.3915* (2011).

[244] X. Song and A. Taamouti. "Measuring nonlinear granger causality in mean". In: *Journal of Business & Economic Statistics* 36.2 (2018), pp. 321–333.

[245] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[246] Federal Reserve Bank of St. Louis. *St. Louis Fed Financial Stress Index [STLFSI]*. 2020. URL: `https://fred.stlouisfed.org/series/STLFSI` (visited on 03/01/2020).

[247] D.J. Stekhoven. *missForest: Nonparametric Missing Value Imputation using Random Forest*. 2013. URL: `Rpackageversion1.4.0`.

[248] J.H. Stock and M.W. Watson. "Forecasting using principal components from a large number of predictors". In: *Journal of the American Statistical Association* 97.460 (2002), pp. 1167–1179.

[249] J.H. Stock and M.W. Watson. "Handbook of macroeconomics". In: Elsevier, 2016. Chap. Dynamic factor models, factor-augmented vector autoregressions, and structural vector autoregressions in macroeconomics.

[250] J.H. Stock and M.W. Watson. "Implications of dynamic factor models for VAR analysis". In: *National Bureau of Economic Research* (2005).

[251] C. Stoll, L. Klaaben, and U. Gallersdorfer. "The carbon footprint of bitcoin". In: *Joule* 3.7 (2019), pp. 1647–1661.

[252] C.J. Stone. "Optimal rates of convergence for nonparametric estimators". In: *The annals of Statistics* (1980), pp. 1348–1360.

[253] J.D. Storey and R. Tibshirani. "Statistical significance for genomewide studies". In: *Proceedings of the National Academy of Sciences* 100.6 (2003), pp. 9440–9445.

[254] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE. 2015, pp. 1–9.

[255] A. Taamouti, T. Bouezmarni, and A. El Ghouch. "Nonparametric estimation and inference for conditional density based Granger causality measures". In: *Journal of Econometrics* 180.2 (2014), pp. 251–264.

[256] D. Tang, B. Qin, and T. T. Liu. "Document modeling with gated recurrent neural network for sentiment classification". In: *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2015, pp. 1422–1432.

[257] A. Tank, I. Covert, N. Foti, A. Shojaie, and E. Fox. "Neural Granger Causality for Time Series". In: *arXiv preprint arXiv:1802.05842* (2018).

[258] M. Tewlgarsky. "Benefits of depth in neural networks". In: *arXiv preprint arXiv:1602.04485* (2016).

[259] R. Tibshirani. "A comparison of some error estimates for neural network models". In: *Neural Computation* 8.1 (1996), pp. 152–163.

[260] R. Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[261] N. Tishby and N. Zaslavsky. "Deep learning and the information bottleneck principle". In: *2015 IEEE Information Theory Workshop (ITW)*. IEEE. 2015, pp. 1–5.

[262] L.H. Ungar, R.D. De-veaux, and E. Rosengarten. "Estimating prediction intervals for artificial neural networks". In: *Proc. of the 9th Yale Workshop on Adaptive and Learning Systems*. 1996.

[263] S. Urban. *Neural Network Architectures and Activation Functions: A Gaussian Process Approach*. Doctoral Dissertation, Technische Universität München, 2017.

[264] R. Vaillant, C. Monrocq, and Y. Le Cun. "Original approach for the localisation of objects in images". In: *IEE Proceedings-Vision, Image and Signal Processing* 141.4 (1994), pp. 245–250.

[265] K.R. Varshney and H. Alemzadeh. "On the safety of machine learning: Cyber-physical systems, decision sciences, and data products". In: *Big data* 5.3 (2017), pp. 246–255.

[266] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. "Phoneme recognition using time-delay neural networks". In: *IEEE transactions on acoustics, speech, and signal processing* 37.3 (1994), pp. 328–339.

[267] H. Wang and C. Leng. "Unified LASSO Estimation Via Least Square Approximation". In: *Journal of American Statistical Association* 102.479 (2007), pp. 1039–1048.

[268] P. Wang, X. Deng, H. Zhou, and S. Yu. "Estimates of the social cost of carbon: A review based on meta-analysis". In: *Journal of cleaner production* 209 (2019), pp. 1494–1507.

[269] S. Wang. "General constructive representations for continuous piecewise-linear functions". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 51.9 (2004), pp. 1889–1896.

[270] D. Warde-Farley, I. Goodfellow, A. Courville, and Y. Bengio. "An empirical analysis of dropout in piecewise linear networks". In: *arXiv preprint arXiv:1312.6197* (2014).

[271] M. Welling and Y.W. Teh. "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the 28th international conference on machine learning*. 2011, pp. 681–688.

[272] N. Wiener. "Modern Mathematics for Engineers". In: New York: McGraw-Hill, 1956. Chap. The Theory of Prediction.

[273] T.N. Wiesel and K.T. Brown. "Analysis of receptive fields in the cat's retina". In: *Annals of the New York Academy of Sciences* 74.2 (1959), p. 405.

[274] K.D. Williams, A.J. Hewitt, and A. Bodas-Salcedo. "Use of Short-Range Forecasts to Evaluate Fast Physics Processes Relevant for Climate Sensitivity". In: *Journal of Advances in Modeling Earth Systems* 12.4 (2020), p.e2019MS001986.

[275] R.J. Williams and D. Zipser. "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: *Backpropagation: Theory, architectures, and applications* 433 (1992), p. 17.

[276] J. Willms. "Bitcoin Mining In North America: A New Gold Rush In The New World". In: *Bitcoin Magazine* (2019).

[277] I. Wilms, S. Gelper, and C. Croux. "The predictive power of the business and bank sentiment of firms: A high-dimensional Granger Causality approach". In: *European Journal of Operational Research* 254.1 (2016), pp. 138–147.

[278] F.Z. Xing, E. Cambria, and R.E. Welsch. "Natural language based financial forecasting: a survey". In: *Artificial Intelligence Review* 50.1 (2018), pp. 49–73.

[279] H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao. "On the Number of Linear Regions of Convolutional Neural Networks". In: *arXiv preprint arXiv:2006.00978* (2020).

[280] F. Yang, H.Z. Wang, H. Mi, and W.W. Cai. "Using random forest for reliable classification and cost-sensitive learning for medical diagnosis". In: *BMC bioinformatics* 10.S1 (2009), S22.

[281] W.H. Young. "On the Conditions for the Reversibility of the Order of Partial Differentiation". In: *Proceedings of the Royal Society of Edinburgh* 29 (1909), pp. 136–164.

[282] M. Yuan and Y. Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[283] T. Zaslavsky. "Facing up to arrangements: face-count formulas for partitions of space by hyperplanes". In: *Memoirs of American Mathematical Society* 154 (1975), pp. 1–95.

[284] N. Zhou and J. Zhu. "Group variable selection via a hierarchical lasso and its oracle property". In: *arXiv preprint arXiv:1006.2871* (2010).

[285] Z.H. Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[286] Z.H. Zhou, J.X. Wu, Y. Jiang, and S.F. Chen. "Genetic algorithm based selective neural network ensemble". In: *IJCAI-01: proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. 2001.

[287] L. Zhu and N. Laptev. "Deep and confident prediction for time series at uber". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2017, pp. 103–110.

[288] D. Zou, Y. Cao, D. Zhou, and Q. Gu. "Stochastic Gradient Descent Optimizes Over-parameterized Deep ReLU Networks". In: *arXiv preprint arXiv:1811.08888* (2018).

[289] H. Zou. "The adaptive lasso and its oracle properties". In: *Journal of the American statistical association* 101.476 (2006), pp. 1418–1429.

APPENDIX A

Feedforward Neural Networks

Chapter Abstract

The aim of this chapter is to provide a brief introduction to deep feedforward (sequential) neural networks for both regression and classification purposes. In particular, after providing a brief definition, the different activation functions, output functions, and widely adopted penalty functions are discussed. Following, the MC dropout discussed in chapter 4 is applied for optimal portfolio allocation.

A.1 Introduction to feedforward neural networks

A deep feedforward neural network (DNN) is composed by a collection of connected units called neurons. Each neuron receives inputs from the preceding neurons, it computes the weighted sum, and maps the weighted sum via an activation function to the neurons in the next layers. This operation is repeated until the final layer, or output layer, is reached. Such a neural network does not have feedback connections that pass information backwards and thus, it is defined *feedforward* neural network due to the unidirectional weights. More formally, for any two natural numbers $d,\ n_2 \in \mathbb{N}$, which are called input and output dimension respectively, a natural number $N \in \mathbb{N}$ called *depth*, and a *width vector* $(Z_1, \cdots, Z_{N+1}) \in \mathbb{N}^{N+1}$, a $\mathbb{R}^d \to \mathbb{R}^{n_2}$ $N+1$-layers DNN, $\widehat{f} : \mathbb{R}^d \to \mathbb{R}^{n_2}$, is given by

$$f(\mathbf{x}_i; \boldsymbol{\omega}) = \theta_O(\mathbf{W}_{N+1}\theta(\cdots\theta(\mathbf{W}_2\theta(\mathbf{W}_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2)\cdots) + \mathbf{b}_{N+1}) \tag{A.1}$$

where $\theta(x)$ is the chosen activation function that introduces the non-linearity into the system, $\mathbf{W}_n \in \mathbb{R}^{Z_n \times Z_{n-1}}$ for $N \neq 1$, and $\mathbf{W}_n \in \mathbb{R}^{Z_1 \times d}$ for $N = 1$, $\mathbf{b}_n \in \mathbb{R}^{Z_n}$ is an intercept or bias vector, $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$, and $\boldsymbol{\omega} = (\mathbf{W}^n, \mathbf{b}_n)$. We distinguish between the activation function of the intermediate layers θ, and the activation function of the output layer θ_O. The choice of the former is usually a hyper-parameter to be tuned and typical choices are (i) Rectified linear units (ReLu), $\theta(x) = \max\{0, x\}$; (ii) Softplus, $\theta(x) = \log(1+e^x)$; (iii) Hard-tanh, $\theta(x) = \max\{-1, \min\{1, x\}\}$; or (iv) Sigmoid, $\theta(x) = (1+e^{-x})^{-1}$. Conversely, the choice of the output activation function θ_O is dictated by the problem at hand; regression problems will require linear activation function, while classification (binary or multilabel) will require either the sigmoid or the softmax activation functions, with the latter defined as $\theta(\mathbf{x}) = (e^{x_i} / \sum_{j=1}^{K} e^{x_j})$ and K the number of classes in the multi-classifier. Figure A.1 reports a visualization of a DNN.

Training a deep learning process usually involves a multi-step numerical optimization of a given loss function $L[\cdot|\cdot]$. One of the most popular algorithm adopted is the *gradient descent*, the intuition of which can be summarized as follows: given a generic function $y = g(x)$, the derivative of the function $\dot{g}(x)$ can provide information on how to scale a change in the input in order to obtain an equivalent change in the output $g(x + \eta) \approx g(x) + \eta\dot{g}(x)$. For example, being $g(x - \eta\text{sign}(\dot{g}(x)))$ lower than $g(x)$ for small enough η, it is possible to reduce $g(x)$ by small changes in x with the opposite sign of the derivative (see also Goodfellow et al., 2016). By extending this argumentation to a DNN setting, the gradient descent algorithm (and its variants) aims to minimize the generic loss function $L[\mathbf{Y}|\mathbf{X}; \mathbf{W}, \mathbf{b}]$ by computing - for each step E (also called *epoch*) - the derivative of the loss function with respect to $\mathbf{W}$ and $\mathbf{b}$ (the procedure adopted for the computation is called *backpropagation*) and by updating the network weights and biases in a direction that ensures a reduction of $L[\mathbf{Y}|\mathbf{X}; \mathbf{W}, \mathbf{b}]$.

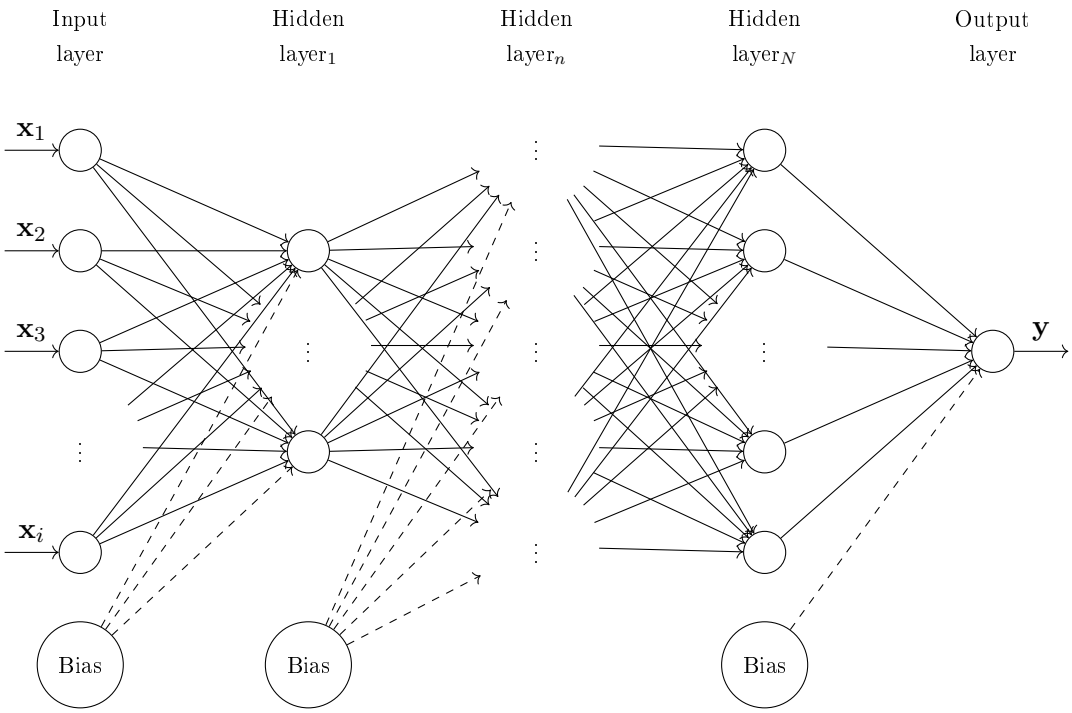


Figure A.1: Deep Feedforward neural network for regression with $n_2 = 1$. For multi-dimensional outputs, a number of output nodes equal to the output dimension must be used. Two output nodes are required for binary classification, K nodes for multi-label classification.

The choice of the loss function is imposed by the output of the network. To train the DNN for regression purposes, a common loss function is the Euclidean distance defined as:

$$\frac{1}{2n_2}||\mathbf{y}_i - \hat{f}(\mathbf{X}; \mathbf{W}, \mathbf{b})_i||_2^2 \tag{A.2}$$

Conversely, when the model is used for multi-label classification, the cross-entropy loss function is usually considered:

$$-\sum_{j=1}^{K} \mathbb{1}_{j,i} \log(p_{j,i}) \tag{A.3}$$

where $\mathbb{1}_{j,i}$ is a binary indicator that is equal to 1 is label j is correct for observation i, and $p_{j,i}$ is the probability that observation i is of class j.

As previously stated, the main goal for DNNs is to minimize the loss functions A.2 or A.3 and obtain a model that performs well also on unseen data or *test* data. Thus, in order to reduce the tendency of *overfitting* - low loss function in the training set but high loss function in the test set - a regularization term is usually added to the loss function (see Scardapane et al., 2017). A typical regularization scheme is of the form[112]:

[112]The same equation can be obtained for the loss function in A.3.

$$\frac{1}{2n_2}||\mathbf{y}_i - \hat{f}(\mathbf{X}; \mathbf{W}, \mathbf{b})_i||_2^2 + \lambda\varpi[\boldsymbol{\omega}] \tag{A.4}$$

where λ is the regularization term that defines the level of smoothness introduced by the penalty function $\varpi[\boldsymbol{\omega}]$. Typical choices of the penalty functions are: (i) Ridge, (ii) Least absolute shrinkage and selection operator (Lasso), and (iii) Elastic net.

A.2 Optimal portfolio allocation

In the same spirit of Chinco et al. (2019) and, more specifically, Gu et al. (2020), the present subsection proposes an empirical application of deep learning methods that illustrates its relevance in empirical finance. Whereas these authors highlight the advantages of using regression trees and neural networks for asset pricing (measuring the risk premium on risky assets) compared to linear regression and techniques based on dimension reduction, the proposed empirical exercise performs a comparative study against conventional time series models widely used for empirical finance modeling. In particular, we present a forecasting exercise of the conditional mean and volatility of asset returns of the S&P, the Dow Jones, and the Nasdaq indices starting from 1972/07/30 until 2020/07/30. The objective is twofold. First, we aim to assess the predictive performance of a modern deep neural network model and compare it against a traditional time series model that carries out a transitory-permanent decomposition of the asset price. The permanent component captures the trend of the log-price and the transitory component models the log-returns on the financial indices. The transitory component also accommodates the presence of conditional heteroscedasticity by fitting a GARCH(1,1) model. The statistical comparison in terms of predictive performance –mean squared prediction error (MSPE)– is done by implementing a one-sided Diebold-Mariano (1995) test of predictive accuracy. Second, as in Gu et al. (2020), economic significance is added to the comparison. To do this, the out-of-sample Sharpe ratios of optimal portfolios constructed from a combination of the three financial indices are compared. The optimal combination is obtained using Markowitz's (1952) mean-variance and minimum-variance portfolios as the investor's objective functions. In order to obtain the out-of-sample conditional mean and volatility forecasts, a fixed rolling window approach with 50 steps is applied. Thus, the period following 2016/06/30 (included) is used for out-of-sample evaluation.

First, asset prices are transformed into log returns, and apply standard stationarity tests of the analyzed series. We conduct the Dickey-Fuller test allowing for a maximum of 10 lags. The unit root null hypothesis is rejected at 0.01 significance level in all cases; additionally, we also perform the KPSS test and fail to reject the null hypothesis of stationarity in all cases at 0.1 significance level.

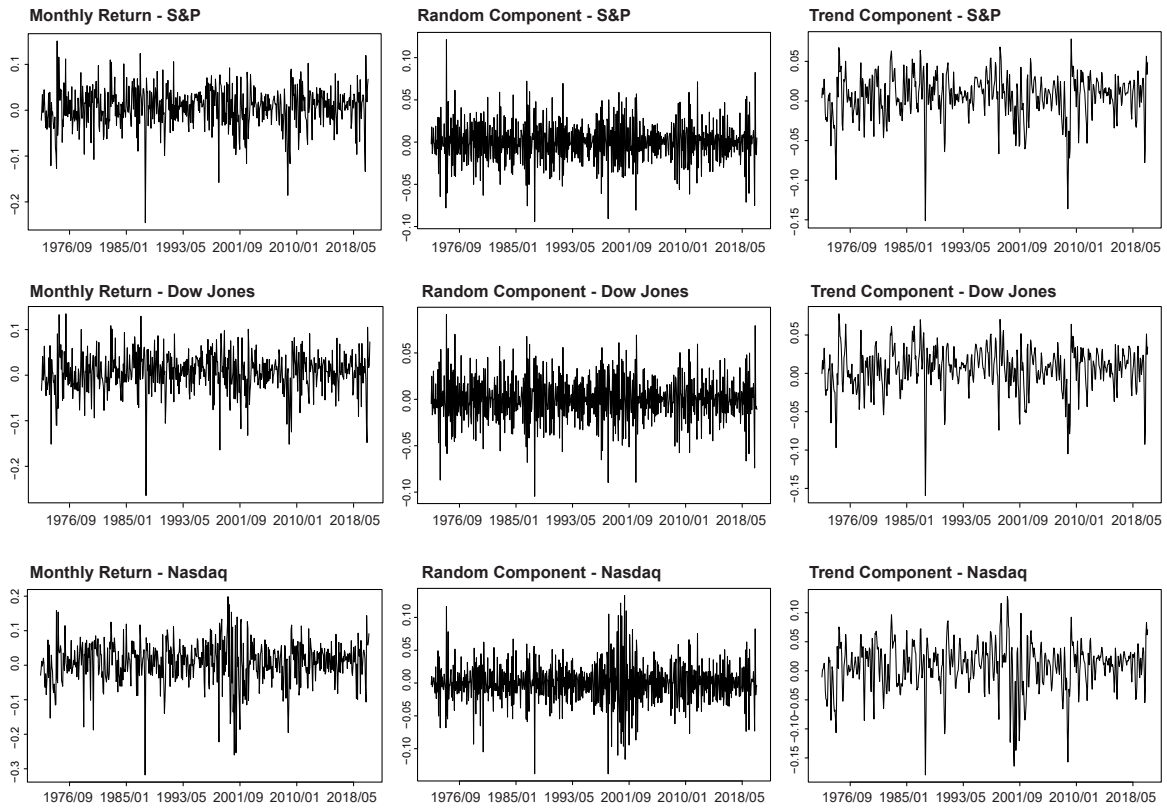


Figure A.2: Decomposed Time Series

Following the recent literature on deep learning and time series forecasting focusing on enhancing the forecasting accuracy of DNNs by using time series decomposition (see Smyl, 2020; Hansen and Nelson, 2003; Méndez-Jiménez and Cárdenas-Montes, 2018 among the others), the present chapter couples the MC dropout approach of Gal and Ghahramani (2016a) with time series decomposition. In this framework, we usually identify a trend component $\mathbf{T}_t$, a seasonal component $\mathbf{\Psi}_t$, and a random component $\mathbf{\Xi}_t$. Assuming additive decomposition, the time series can be modeled as $\mathbf{X}_t = \mathbf{\Xi}_t + \mathbf{\Psi}_t + \mathbf{T}_t$. See Figure A.2 above[113].

Based on the algorithm of Smyl (2020), the present sub-section fits and forecasts the trend component using an exponential smoothing model, and the random component using either a DNN or a GARCH(1, 1) model [114]. When a GARCH(1,1) is fitted, the final forecast will be the sum of the individual forecasts $\hat{\mathbf{\Xi}}_{t+1} + \hat{\mathbf{T}}_{t+1}$. When the DNN model is considered, B stochastic forward passes (see Gal and Ghahramani (2016a), and chapter 4 for a detailed analysis) are performed to forecast the random component $\hat{\mathbf{\Xi}}_{t+1}$; to each of these random stochastic forward passes the forecasted trend $\hat{\mathbf{T}}_{t+1}$ is added, and B point forecasts of $\{\hat{\mathbf{X}}^b_{t+1}\}_{b=1}^{B}$ are obtained. The point forecast of the log prices is the mean $\bar{\mathbf{X}}_{t+1}$ over the B forward passes.

For each time series analyzed, a neural network with three hidden layers of 50 nodes each, trained with Adam optimizer with learning rate 0.001, an exponential decay rate for the 1$^{\text{st}}$

[113]The seasonal component is not reported as the magnitude was approximately 0 with the highest value observed $3e^{-04}$.

[114]As robustness exercise, we also consider a GARCH(1,1) fitted on the time series $\mathbf{X}_t$.

moment estimates (β_1) equal to 0.900, and an exponential decay rate for the 2^{st} moment estimates (β_2) equal to 0.999 is fitted. We also consider a dropout rate of 0.1 across all layers, and 300 epochs. The input layer comprises of the multivariate time series with relative lagged values (up to $k = 10$). Additionally, in order to ensure proper training of the network, the input data $\boldsymbol{\Xi}_{t-k}$ for $k = 1, \cdots, 10$ is normalized to guarantee that the regressors have zero mean and unit standard deviation.

As mentioned earlier, a fixed rolling window approach is implemented in order to obtain 50 one-step-ahead forecasts. We first evaluate the performance of the proposed approach against a GARCH(1, 1) model in terms of MSPE using the one-sided Diebold-Mariano (DM) test (1995), with hypothesis:

$$\mathcal{H}_0 : MSPE^i_{nn} \geq MSPE^{i,j}_{GARCH} \tag{A.5}$$

and the alternative is

$$\mathcal{H}_1 : MSPE^i_{nn} < MSPE^{i,j}_{GARCH} \tag{A.6}$$

with $i = 1, 2, 3$ indicating the three time series analyzed, and $j = 1, 2$ defining the two alternative methodologies used to predict with a GARCH(1,1) model - a first methodology that decomposes the analyzed time series and combines the point forecast from a GARCH(1,1) with an exponential smoothing, and a second methodology that does not decompose the time series and directly forecast with a GARCH(1,1).

The results of the predictive ability test are as follows. For the S&P index, the test statistic of the DM_1 is 2.3970 with a p-value of 0.0102 and the test statistic of DM_2 is 2.5262 with p-value of 0.0074. For the Dow Jones index, the test statistic of the DM_1 is 2.4729 with a p-value of 0.0084 and the test statistic of DM_2 is 2.0435, with p-value of 0.0232. For the Nasdaq index, the test statistic of the DM_1 is 2.7578 with a p-value of 0.0041 and the test statistic of DM_2 is 3.9139 with p-value of 0.0001. The reported p-values show the out-performance of the DNN approach against a GARCH(1,1) benchmark.

To further validate the out-of-sample performance of the proposed approach, the MC-dropout is compared against a GARCH(1,1) benchmark in terms of portfolio returns for a given optimal strategy. Considering a mean-variance portfolio, the weights are defined as:

$$\begin{aligned} \min_{\boldsymbol{\gamma}} \quad & \boldsymbol{\gamma}^\intercal \hat{\boldsymbol{\Sigma}} \boldsymbol{\gamma} - \boldsymbol{\gamma}^\intercal \hat{\mathbf{x}} \\ \text{s.t.} \quad & \boldsymbol{\gamma}^\intercal \mathbf{1} = 1 \end{aligned} \tag{A.7}$$

with $\boldsymbol{\gamma} \in \mathbb{R}^3$ the vector of the portfolio weights invested in the three indices considered, $\hat{\boldsymbol{\Sigma}} \in \mathbb{R}^{3\times 3}$ the estimated covariance matrix, $\hat{\mathbf{x}} \in \mathbb{R}^3$ the vector of the expected returns, and $\mathbf{1} \in \mathbb{R}^3$ a vector of ones. The covariance matrix is defined as:

$$\hat{\boldsymbol{\Sigma}} = \text{diag}(\hat{\boldsymbol{\sigma}}) \hat{\mathbf{P}} \text{diag}(\hat{\boldsymbol{\sigma}}) \tag{A.8}$$

with $\text{diag}(\hat{\boldsymbol{\sigma}})$ being the diagonal matrix with estimated standard deviations, and $\hat{\mathbf{P}}$ the correla-

tion matrix[115]. The present sub-section considers two portfolio strategies: the mean-variance and the minimum-variance portfolios (the latter obtains by imposing $\hat{\mathbf{x}} = 0$ in the constrained minimization in (A.7)). Knowing that holding the portfolio $\boldsymbol{\gamma}_t^{\text{strategy}}$ for a time Δt gives the out-of-sample return for $t + \Delta t$ and by imposing $\Delta t = 1$, the rolling window approach used to evaluate the out-of-sample performance of a given strategy is as follows: at time t the one-step-ahead conditional mean and volatility of the three stocks are forecasted using either a GARCH(1, 1) or a DNN model. We construct the dynamic covariance matrix $\hat{\boldsymbol{\Sigma}}_{t+1}$ from estimates of the conditional variances and covariances over rolling windows.

Based on the forecasted $\hat{\mathbf{X}}_{t+1}$ and $\hat{\boldsymbol{\Sigma}}_{t+1}$, the constrained minimization in (A.7) is solved and the weights $\boldsymbol{\gamma}_t^{\text{strategy}}$ computed. The return of the portfolio in $t + 1$ will be the weighted mean of the observed returns of the three stocks in $t + 1$, with weights $\boldsymbol{\gamma}_t^{\text{strategy}}$: $\Upsilon_{t+1} = \boldsymbol{\gamma}_t^{\text{strategy}\ \intercal}\mathbf{x}_{t+1}$. By implementing a fixed rolling window forecasting exercise, the above procedure is repeated 50 times to obtain 50 out-of-sample Υ_{t+1} from either the GARCH(1, 1) or the DNN model. This allows us to estimate the out-of-sample Sharpe ratios as:

$$\text{Sharpe ratio}_i = \frac{\hat{\Upsilon}_p - \Upsilon_{rf}}{\hat{\sigma}_p} \tag{A.9}$$

with $\hat{\Upsilon}_p$ being the mean return of the portfolio, Υ_{rf} the risk-free rate (assumed equal to 0), and $\hat{\sigma}_p$ the portfolio standard deviation.

Figure A.3 reports the cumulative returns of the four different strategies considered. One could notice how a portfolio strategy (either mean-variance or minimum variance) based on DNN forecasts outperforms a strategy based on GARCH(1,1) forecasts. In particular, the annualized sharpe ratios of the mean-variance and minimum-variance portfolios obtained from the forecasted return and volatility from a DNN are: 0.6777 and 0.7562 respectively; the annualized sharpe ratios obtained from a GARCH(1,1) forecasts are 0.2686 for the mean-variance and 0.3175 for the minimum-variance portfolio.

[115]The constrained minimization inA.7 allows for short selling but not for leverage effect.

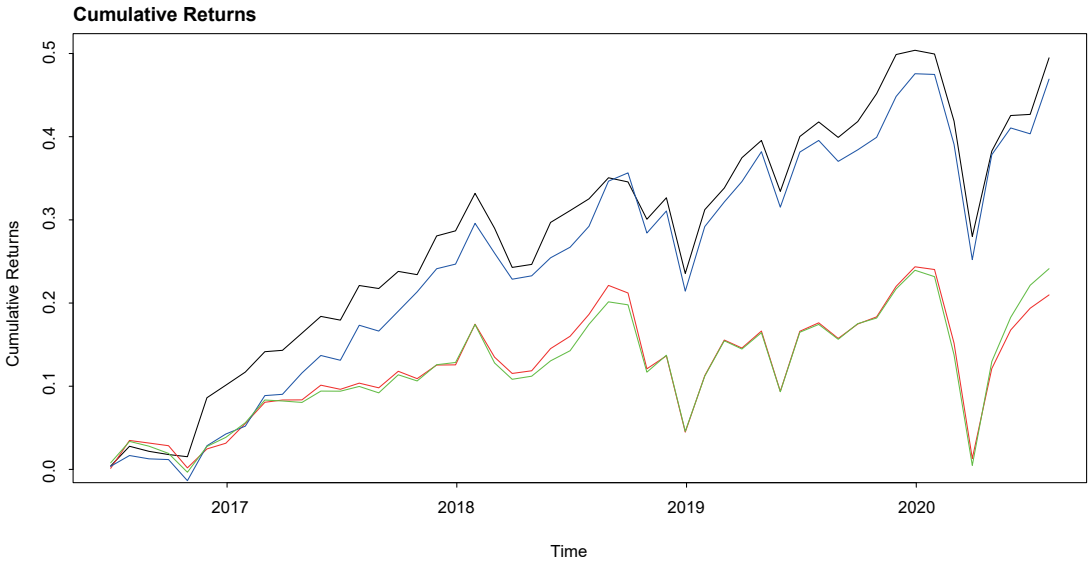


Figure A.3: Out-of-sample cumulative returns of the four portoflio strategies analyzed: in black the minim variance portfolio from the DNN, in green the minimum variance portfolio obtained from GARCH forecasts, in blue the mean variance portfolio obtained from a DNN, in red the mean variance constructed from GARCH forecasts.

The above results extend some of the empirical findings in Gu et al. (2020). These authors, after a thorough review of the literature on machine learning, compare the forecasting performance of ReLu DNNs against linear models and tree-based approaches also in terms of out-of-sample portfolio returns. Gu et al. (2020), based on the out-of-sample forecasts of the individual stock returns, construct a zero-net investment portfolio - that buys and sells the highest and lowest expected returns stocks respectively - and a value weight portfolio. By comparing the out-of-sample returns of the portfolio strategies exploiting the forecasts of the competing models, they show that portfolio strategies based on NN forecasts dominate those based on forecasts of both linear models and tree-based algorithms. If Gu et al. (2020) show that ReLu DNNs can be used to define portfolio strategies based only on the forecasted conditional means of the asset returns, the proposed empirical investigation - by considering the minimum-variance and mean-variance portfolios - improves upon their results, showing that optimal portfolio allocation strategies can also be constructed on ReLu DNNs' forecasted conditional volatilities, or on a combination of conditional mean and conditional volatilities of stock returns.

APPENDIX B

Convolutional Neural Networks

Chapter Abstract

The aim of this chapter is to provide a brief introduction to convolutional neural networks by describing the convolutional operations, typical structures, and by characterizing the different types of hidden layers that comprise the structure. Finally, an empirical investigation focused on Glaucoma detection starting from fundus images is also implemented. In particular, by reporting the hierarchical features learned by the hidden layers of the convolutional neural network, the neural network learning process is compared with a specialist in the field.

B.1 Introduction to convolutional neural networks

In order to understand how a convolutional neural network works, it is useful to summarize the pioneering work of Hubel (1958) and Wiesel and Brown (1959) focused on the functioning of the cats' visual cortex. The authors showed how the visual cortex comprises different neurons and each of these neurons has a *receptive field* that reacts only to the sub-image located within the receptive field itself. Thus, the *visual field* obtains from overlapping and combining all the receptive fields. Additionally, the authors also noticed a hierarchical structure in the receptive fields of the visual cortex, as some receptive fields react to combination of lower level receptive fields (see also Géron, 2017). The convolutional neural network mimics the functioning of the visual cortex.

The convolutional neural network (CNN) was first introduced by LeCun (1989) in its seminal paper and used to recognize hand written digits. CNNs are a special case of the DNNs discussed in the previous sub-section that use a mathematical operation called *convolution* in at least one of their hidden layers (Goodfellow et al., 2016). Nowadays, CNNs have become extremely popular in a high number of applications (LeCun et al., 1998; Waibel et al., 1994; Vaillant et al., 1994; and Lawrence et al., 1997). In particular, they are widely adopted for processing grid-like data, either in one dimension (time series or language processing) or in two dimensions (image data where the image can be seen as a 2D grid of pixels).

A CNN is a deep neural network that usually comprises three type of layers: (a) *convolutional*, (b) *pooling*, and (c) ReLu. Having defined the ReLu activation function in the previous sub-section, the following paragraphs will focus on defining the other two types of layers and operation distinctive of CNNs' architectures[116]. In particular, convolutional and pooling layers in CNNs are usually described by a 3-dimensional grid structure identified by the dimensions *height*, *width*, and *depth*. The term "depth" - in this case - must not be confused with the depth of the network, as it refers to the number of *channels* - or features map - in each layer. As an example, a given input image is transformed in the input layer as a 3 dimensional array with the first 2 dimensions capturing the spatial dimensions of the image (the grid of *pixels*) and the third dimension capturing the color of the pixel (feature map). Thus, the 3D structure of the input layer captures the spatial attributes of the image and the independent attributes of each channel (i.e., colors); while for the hidden layers, the third dimension captures the different receptive fields (types of shapes) extracted from sub-regions of the image (the hierarchical structure of the CNN entails focusing on low-level features in the first hidden layers, and in higher-level features in deeper layers). In the reminder of the sub-section we will refer to the size of the n^{th} layer as $L_n \times B_n \times d_n$, with L_n the height, B_n the width, and d_n the depth of the layer.

When the input layer is analyzed, the values are usually defined by the input data being processed (pixels grid), and $d_1 = 1$ if the image is black and white (for example the MNSIT

[116]The interested reader is referred to Aggarwal (2018) and Goodfellow et al. (2016) for a textbook argumentation.

dataset) or $d_1 = 3$ if it is a color image (RGB color channels). For $n > 1$, $L_n \times B_n$ no longer identifies the grid of raw pixels of an image but a grid of values - *feature map* - that performs the same function as the hidden layers previously discussed in DNNs. The parameters of a CNN (neuron's weights) - defined *filters* or *kernels* - are 3-dimensional shapes with a height and width smaller that the layer to which they are applied (similar to receptive fields) and a depth equal to the given layer. The dimensions of the kernel in the n^{th} layer are $F_n \times F_n \times d_n$.

The convolution operation mimics the functioning of the visual cortex reported by Hubel (1958) and Wiesel (1958). In particular, it places the kernels at each possible position of the input or hidden layer (ensuring a full overlap) and it performs a dot product between the $F_n \times F_n \times d_n$ parameters in the kernel and the matching grid from the n^{th} layer. Thus, the number of dot products defines the dimension of the gird of the following layer $n + 1$. When performing the convolution operation at layer n, given a filter dimension $F_n \times F_n$, $L_{n+1} = (L_n - F_n + 1)$ overlapping filters can be aligned vertically, and $B_{n+1} = (B_n - F_n + 1)$ horizontally (see also Aggarwal, 2018). Thus, the number of dot products will be $L_{n+1} \times B_{n+1}$. Finally, the depth of the following layer depends on the number of unique filters used for the convolution; the arrangement of the output of the d_{n+1} filters in the third dimension constitutes the aforementioned feature map.

It is a standard pattern in CNNs to increase the depth and to decrease the size (height and width) of the feature map as n increase. This standard resembles the hierarchical function of the visual cortex discussed previously: lower level feature map will focus on lower level information (wider height and width), while deeper levels feature map will focus on higher level information processed by the previous layers (shorter width and depth) and to process lower level information is required an increasing capacity of the CNN and thus a higher number of parameters.

It is now possible to describe the convolutional operation described above in a more formal way. In particular, we define the p^{th} weight (kernel or filter) in the n^{th} hidden layer with a 3D tensor defined as $\mathbf{W}^{(p,n)} = [w_{ijk}^{(p,n)}]$ with i indicating the height, j the width, and k the depth of the filter ($F_n \times F_n \times d_n$). The feature map identifying the n^{th} hidden layer is also defined by a 3D tensor $\mathbf{H}_n = [h_{ijk}^n]$. Having defined filters and features maps, the convolutional operation can be formalized as:

$$h_{ijp}^{n+1} = \sum_{r=1}^{F_n} \sum_{s=1}^{F_n} \sum_{k=1}^{d_n} w_{rsk}^{p,n} h_{i+(r-1),j+(s-1),k}^{n} \quad \begin{array}{l} \forall\ i \in \{1, \cdots, L_n - F_n + 1\} \\ \forall\ j \in \{1 \cdots B_n - F_n + 1\} \\ \forall\ p \in \{1 \cdots d_{n+1}\} \end{array} \tag{B.1}$$

As one could notice, Equation B.1 represents the dot product between the filter (kernel) - indexed by p - and the matching grid from the n^{th} layer over all its spatial position (i, j). In order to further understand how the convolutional operation in B.1 works, a two dimensional

representation of the operation is provided:

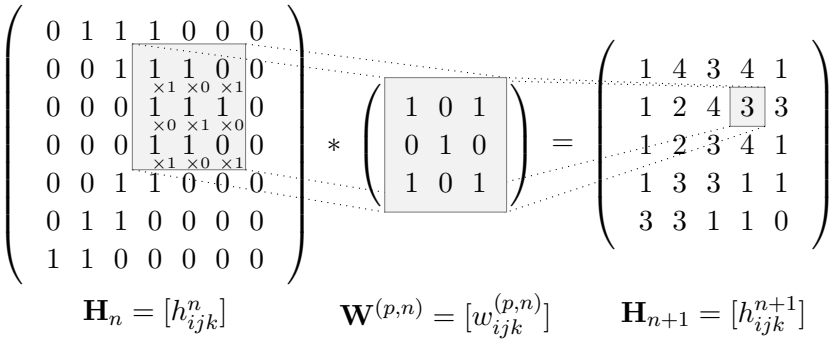


Figure B.1: Convolution operation with n^{th} layer of dimension $7 \times 7 \times 1$, filter of dimension $3 \times 3 \times 1$, and $n^{th} + 1$ layer of dimensions $5 \times 5 \times 1$.

Figure B.1 reports the convolution operation where $L_n = 7$, $B_n = 7$, $d_n = 1$, and $F_n = 3$. The dimensions of the $n^{th} + 1$ layer are defined as $L_{n+1} = (7 - 3 + 1) = 5$, and $B_{n+1} = (7 - 3 + 1) = 5$, and $d_{n+1} = 1$. The Figure reports also a visual representation of the dot product between the filter and one of the 25 matching grids in the n^{th} layer. The reader can notice that each entry in the $n^{th} + 1$ layer is obtained by replicating the dot product for all overlapping grids of dimensions 3×3 over the n^{th} hidden layer.

As one could notice the convolution operation described above reduces the size of the n^{th} layer when compared to the $n^{th} + 1$ layer. Knowing that a reduction in size is equivalent to a reduction in the information being processed by the layer (an example could be the pixels at the border of the image for the input layer), and by assuming that this loss of information is not always desirable, the operation known as *zero padding* is sometimes implemented. Zero padding ensures that the $n^{th} + 1$ layer will have the same dimensions as the n^{th} layer by adding $(F_n - 1)/2$ zeros around either the pixel grid (for the input layer) or the feature map (for hidden layers). Alternatively, due to the dimensions of the problem analyzed, it may be desirable to further reduce the dimensions of the n^{th} hidden layer. This additional reduction in the spacial footprint is often achieved by reducing the applicability of the kernels using *strides* (S_n). In this case, the convolution is not applied to each overlapping spatial position in the n^{th} hidden layer, but at the locations $S_n + 1$ along both width and height of the hidden layer, resulting in the reduced dimensions of the $n^{th} + 1$ layer equal to $(L_n - F_n)/S_n + 1$, and $(B_n - F_n)/S_n + 1$[117].

It is now necessary to analyze the pooling layers. Pooling layers work similarly to convolutional layers: by defining the dimension of the pooling operation as $P_n \times P_n$, the *max pooling* takes the maximum value of the matching grid ($P_n \times P_n$) in the n^{th} hidden layer. Thus, if a $S_n = 1$ is used, the size of the $n^{th} + 1$ layer will be $(L_n - P_n) + 1$ and $(B_n - P_n) + 1$. In other words, a max pooling layer does not have weights, it simply aggregates the inputs from the preceding layer using the max operator. Additionally, the pooling operation is performed at

[117]Thus, it is possible to notice that the above discussion implicitly assumes a stride equal to 1.

the level of each feature map (Aggarwal, 2018) implying that the pooling layer - as opposed to the convolutional one - does not change the number of feature maps d_{n+1}. The main goal of pooling layers is to shrink the image (without loss of information) in order to reduce the computational load. It is based on this final aspect that it is possible to understand why, a $S_n = 1$ is frequently used when performing max pooling.

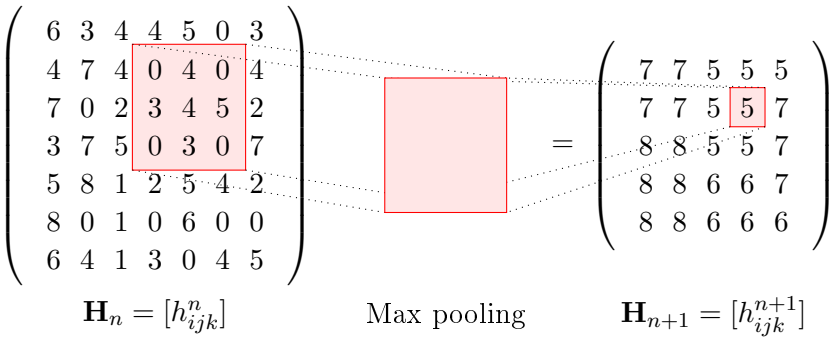


Figure B.2: Max pooling operation with n^{th} layer of dimension $7 \times 7 \times 1$, stride $= 1$, and $n^{th} + 1$ layer of dimensions $5 \times 5 \times 1$.

Finally, a fully connected hidden layer is added in order to connect each hidden state across different feature maps. In the majority of the cases, more than one fully connected hidden layer is added in order to increase the representation power of the CNN. As in the case of feedforward neural networks, the output layer is defined by the task analyzed. As an example, if the CNN is used for image classification, being the problem traceable to a binary (distinguish between two sets of images – cat vs dog competition in Kaggle) or to a mutliclass (MNSIT dataset) classification problem, typical choices of activation functions for the output layer are logistic or softmax activation functions.

Thus, based on the above definitions, it is also possible to understand how CNNs are usually organized. In particular, typical CNNs' structures tend to reduce the gird size of the feature maps and to increase the number of feature maps (d_n) as the depth increases. This common trait is achieved by reducing the dimension of the n^{th} layer using a max pooling operation, and by increasing the number of feature maps from layer n^{th} to layer $n^{th} + 1$ using a convolutional operation with zero padding and with a $d_{n+1} > d_n$ (this structure ultimately ensures also no loss of information). A graphical representation of the described structure of CNNs is reported in Figure B.2. The concatenation of max pooling and convolutional layers will depend on the particular problem at hand (famous structures are the *LeNet-5* by LeCun et al., 1998; the *AlexNet* by Krizhevsky et al., 2017; the *GoogleNet* by Szegedy et al., 2015; and the *ResNet* by He et al., 2016).

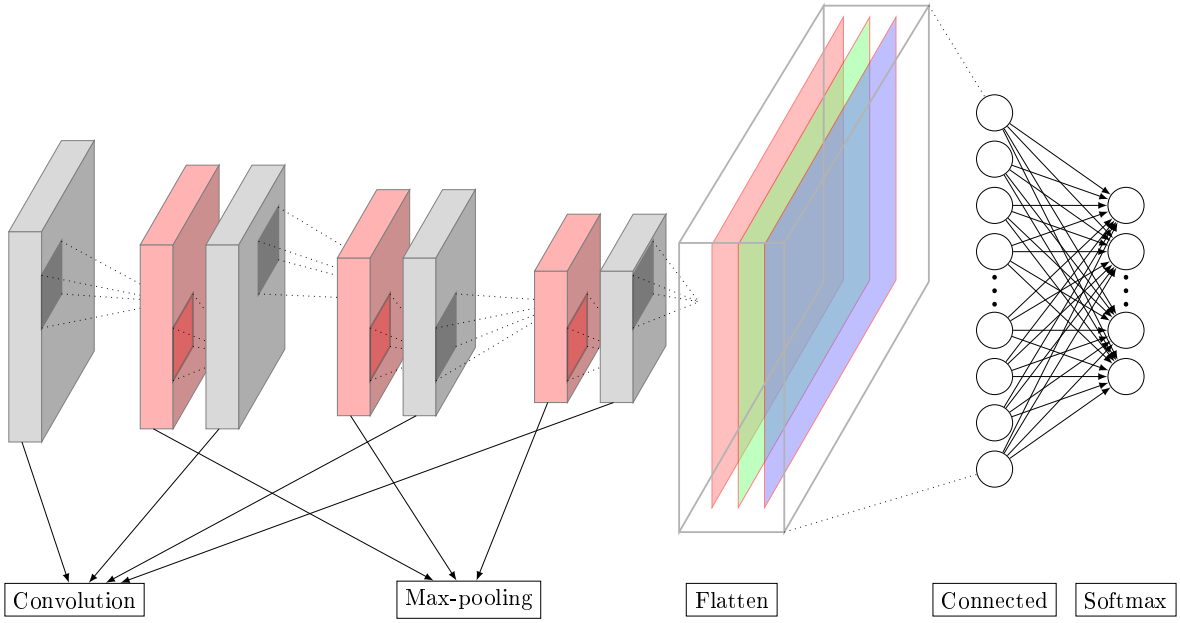


Figure B.3: Deep convolutional neural network for image processing. With four convolutional, and three max-pooling layers. Following the flattening layer, a fully connected hidden layer with a given activation function, and an output layer with softmax activation function.

B.2 Glaucoma detection via fundus images

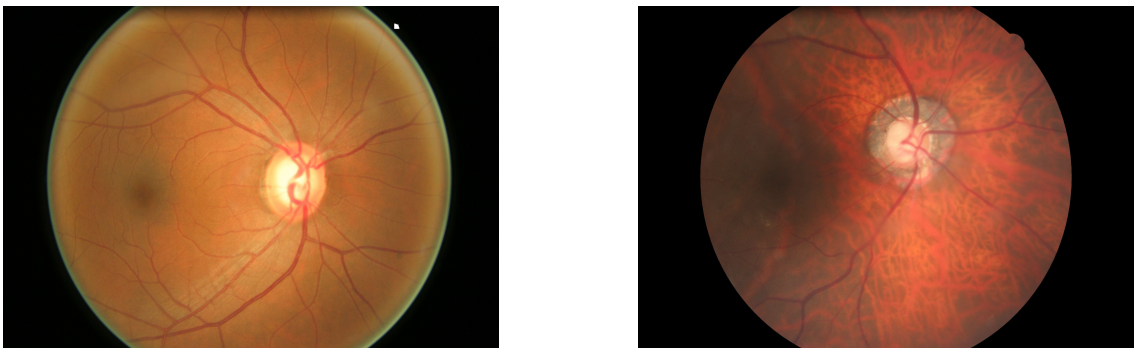
In order to investigate the power of CNN for image classification, the present sub-section fits a deep CNNs for the detection of glaucoma starting from *fundus* images (see also Raghavendra et al., 2018). Glaucoma refers to the progressive visual loss caused by a raised intravascular pressure that irreversibly damages the optic nerve. Based on the data from the World Health Organization (WHO), glaucoma is recognized as the second (after cataracts) leading cause of blindness world wide. In particular, Raghavendra et al. (2018) provide an insightful background necessary to understand the disease: The *neuro retinal rim* is defined as the annular region between *optic disc* and *cup boundary* where the retinal nerve fibers are usually found. The *intraocular pressure* (IOP) quantifies the fluid pressure in the inner portion of the eye. An increase in the levels of IOP will will damage the optic nerve by blocking the outflow of aqueous humor. The consequential corrosion of the optic nerve fibers leads to the thickening of the *retinal nerve fiber layer* (RNFL) causing the progression of glaucoma.

It is well known that the diagnosis of glaucoma is not always easy. For this reason, glaucoma specialists usually conduct numerous different tests in order to correctly diagnose or treat the specific condition. To be accurate, five factors are usually checked on an annual basis: the inner eye pressure via *Tonometry*, the shape and color of the optic nerve via *Ophtalmoscopy*, the complete field vision via *Perimetry*, the angle in the eye where the iris meets the cornea via *Gonioscopy*, and the thickness of the cornea via *Pachimetry*[118].

Figure B.4 below reports two examples of fundus images: sub-figure (a) provides the image

[118]See Glaucoma Research foundation (2020) for a detailed description of the different tests.

of a normal eye, sub-figure (b) the sub-image of an eye affected by glaucoma:



a) Fundus Image - Normal Eye

b) Fundus Image - Eye affected by Glaucoma

Figure B.4: The Figure reports two examples of fundus images: sub-figure (a) report the fundus image of a normal eye; sub-figure (b) the funuds image of an eye affected by Glaucoma.

Since the early 1980s, computer-aided diagnosis (CAD) systems have been widely adopted by the medical community in order to assist doctors in diagnosis a given disease. Focusing on glaucoma detection –in addition to the aforementioned clinical tests– the CAD systems widely adopted are the *confocal scanning laser ophtalmoscopy* (CSLO), the *optical coherence tomography* (OCT), and the *scanning laser polarimetry* (see Hagiwara et al. (2018) for a detailed summary on the topic). Recent advances in deep learning have been increasing the medical community's interest in applying CNNs as an alternative CAD system to further assist doctors; as an example, Shen et al. (2019) develop a deep CNN for breast cancer detection via mammography; Paul et al. (2016) propose a CNN feature extractor to train classifiers to detect cancer from lung computerized tomography (CT) images; and Ragavendra et al. (2018) propose a CNN to detect glaucoma from fundus images.

If the existing literature shows how CNNs can be effectively used as an aid for disease diagnosis due to the high out-of-sample accuracy with associated low false positives and negatives, little empirical investigation –to be the best of our knowledge– has been conducted on analyzing the learning process of CNNs for glaucoma detection. In particular, it is of great interest to understand what are the low and high level features in fundus images used by CNNs in order to produce an outcome. Do CNNs focus on the same features as a doctor specialized in glaucoma treatment and detection? Does the hierarchical structure of the trained CNN mirror the diagnostic mechanism followed by doctors? It is believed that answering to these questions would further improve over the current literature by helping specialists in understanding why a given CNN outcome is returned. To do this, a CNN is trained using available fundus images; following, using the fundus image of an eye affected by glaucoma, a forward pass on the trained CNN is performed and the feature maps of each convolutional layer are stored. The reported feature maps enable understanding the distinctive characteristics analyzed by

the trained CNN and the learning process is assessed using the aid of an expert opinion[119].

The dataset consists of 520 fundus images (386 of normal eyes and 134 of eyes affected by glaucoma) in the train data, and 130 images in the test set (96 images of a normal eye and 34 images of eyes affected by glaucoma). The input images are transformed into 3-dimensional tensors of shape $150 \times 150 \times 3$ –the RGB channel must considered as they are colored images. The structure of the fitted CNN will now be reported. A convolutional layer of dimensions 142×142 with depth 32 and kernel size 9×9, a max-pooling layer with pool size 2×2, a convolutional layer[120] with depth of 64 and kernel size of 7×7, a max-pooling layer with pool size 2×2, a convolutional layer with depth of 128 and kernel size of 5×5, a max-pooling layer with pool size 2×2, a convolutional layer with depth of 128 and kernel size of 3×3, a max-pooling layer with pool size 2×2, a ReLu fully connected layer with 512 hidden nodes and a sigmoid output layer are fitted. One could notice that as the depth of the network increases the depth of the feature maps increases (from 32 to 128), the size of the feature map decreases (from 142×142 to 6×6), and the kernel size decrease (from 9×9 to 3×3).

The loss function is the binary cross entropy, with learning rate 0.0001, and RMSProp optimizer. The pixel values (between 0 and 255) are rescaled into the $[0, 1]$ interval required to properly train the neural network. A fit generator is fitted starting from the train data, and for each epoch (maximum number of epoch is 3) 100 samples are drawn from the fitting process (given a batch size of 20 we would be over-sampling in order to generate more training instances). The out-of-sample accuracy is 0.7409.

Figure B.5 reports the feature maps of the CNN when the sub-figure (b) in Figure B.4 is considered. One could notice that as the depth increases the number of features (depth of the layer) increases, and how the kernel dimension decreases (the first convolutional layer processes lower level information while the deeper layers focus on higher level information) confirming the hierarchical processing of the information contained in the input figure. Additionally, the sparsity of the information increases as the depth of the CNN increases: in sub-figure (d) one could notice how some of the filters are blank (reported in green) meaning that the pattern encoded by the given filter is not recognized in the analyzed fundus image.

The results reported in Figure B.5 were discussed with an expert in Glaucoma detection. It is possible to notice how the DNN correctly focuses on the parapillary atrophy located in the inferior disc region, on the optic disc haemorrhages located on the left of the inferior disc region, and in (sub-figure d) on the neuroretinal rim –position and shape of the optic cup. However, the convolutational neural network commits a common mistake when analyzing fundus images: it focuses on the choroidal veins outside the optic nerve, and on the fovea. Therefore, future work will study the impact of reducing the fundus images to consider only the area around the optic nerve on the classification accuracy of CNNs when applied for

[119] It is important to stress that the focus of the present sub-section it is not to find the optimal CNN structure for glaucoma detection but to open the *black-box* learning process to improve the understanding of CAD results based on deep learning models.

[120] All convolutional layers preserve the grid dimension of the previous layer and no padding is applied.

glaucoma detection.

Figure B.5: The Figure reports the features learned by the CNN when classifying the fundus image reported in sub-figure (b) of Figure B.4. Sub-figure (a) reports the first convolutional layer; sub-figure (b) the second; sub-figure (c) the third and sub-figure (d) the fourth convolutional layer.

APPENDIX C

Recurrent Neural Networks

Chapter Abstract

The aim of this chapter is to provide a brief introduction to recurrent neural networks by describing the typical structures and the solutions proposed to the literature for the construction of deep architectures. Finally, an empirical application focused on text generation from "An inquiry into the Nature and Causes of the Wealth of Nations" by Adam Smith (1776) is proposed.

C.1 Introduction to recurrent neural networks

A recurrent neural network (RNN) is a neural network specialized in learning a sequence of values $x^{(1)}, \cdots, x^{(t)}$. RNNs are widely adopted for processing sequences of arbitrary length such as time series data, sentences, documents, or audio samples. As an example, they are employed in automatic translation, speech-to-text, sentiment analysis (Tang et al., 2015; Pant et al., 2018; and Liu et al., 2017), and time -series forecasting (Oancea and Ciucu, 2014; Ho et al., 2002; and Brezak et al., 2012); interestingly, the Google's Magenta project has also developed a RNN that composes melodies.

A RNN is a special case of the feedforward neural network with a specific structure that –based on the notion of *time layering* and *parameter sharing* across different parts of the model– can work on sequences of arbitrary length. Thus, a recurrent neural network is no longer *feedforward* due to the presence of backward pointing connections. Goodfellow et al. (2016) provide an intuitive example of the importance of parameter sharing and time layering when processing sequences of data: if we consider the two sentences "I went to Nepal in 2009" and "In 2009, I went to Nepal", a deep neural network –when asked to extract the year in which we went to Nepal– should be able to recognize the year as 2009 for both sentences. The DNN analyzed in Appendix A would have a set of parameters that will process all the grammar rules, learned during training, for each word in the sentence. Conversely, a RNN would be sharing the same weights across different time steps (in this case the time variable refers to the positioning of the word within the sentence). The structure of the recurrent neural network will now be analyzed in details.

Following Aggrawal (2018) and Goodfellow et al. (2016), the hidden state at time t will be a function of both input vector and the hidden state at the previous time step $t-1$:

$$\mathbf{h}_t = \theta(\mathbf{h}_{t-1}, \mathbf{x}_{it}) \tag{C.1}$$

In particular, given $\mathbf{x}_{it} \in \mathbb{R}^{d\times 1}$, a set of weights for the input $\mathbf{x}_{it}$ defined as $\mathbf{W}_x \in \mathbb{R}^{Z_1\times d}$, a separate set of weights for the output at the previous step $\mathbf{W}_h \in \mathbb{R}^{Z_1\times Z_1}$, a set of output weights $\mathbf{W}_y \in \mathbb{R}^{d\times Z_1}$, a generic activation function $\theta(\cdot)$, a bias $\mathbf{b} \in \mathbb{R}^{Z_1}$ and $\mathbf{b}_O \in \mathbb{R}^d$ the output of an RNN can be defined as:

$$\mathbf{y}_t = \theta_O(\mathbf{W}_y\theta(\mathbf{W}_x\mathbf{x}_{it} + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}) + \mathbf{b}_O) \tag{C.2}$$

Based on Equation C.1 and C.2, one could notice that $\mathbf{y}_t$ is a function of $\mathbf{x}_{it}$ and $\mathbf{h}_{t-1}$ which in turns is a function of $\mathbf{x}_{it-1}$ and $\mathbf{h}_{t-2}$ which, in turn, is a function of $\mathbf{x}_{it-3}$ and $\mathbf{h}_{t-3}$ and so on until $t = 0$. In the first time stamp ($t = 0$), h_{t-1} is usually assumed equal to 0 (Géron, 2017; and Aggrawal, 2018). Additionally, the two sets of equations formally define the concept of time layering and parameter sharing: although both input $\mathbf{x}_{it}$ and the hidden layer $\mathbf{h}_t$ are a function of time –showing that they will vary as the time stamp changes– the

weight matrices and baiases remain fixed for all t (Aggrawal, 2018). Finally, θ_O is usually the softmax activation functions returning a vector of probabilities. A typical representation of a RNN is reported below (see also Goodfellow et al., 2016):

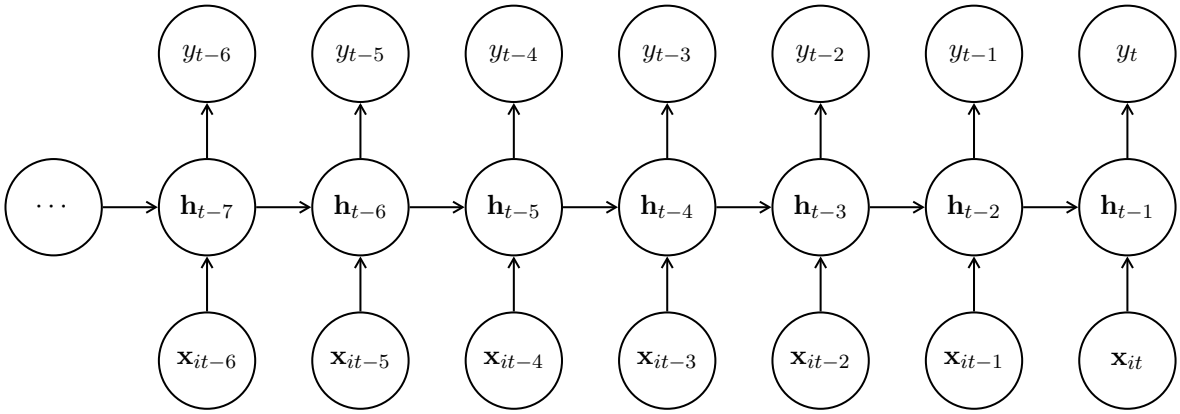


Figure C.1: Recurrent neural network with no bias term

As highlighted by Goodfellow et al. (2016), if the above RNN is unfolded it would correspond to a shallow neural network (that is, only one hidden layer). However, early work on sequential data processing, such as Schmidhuber (1992), El Hihi and Bengio (1995), and Goodfellow et al. (2016), extensively adopt deep RNNs[121]. Before adapting Equation C.1 to a $N+1$ deep RNN, it is important to specify that the weights are shared across different time steps but they are not shared across hidden layers. Therefore, for $n > 1$ (as the first recurrent operation is defined in Equation C.2), Equation C.1 is formulated as follows:

$$\mathbf{h}_t^{(n)} = \theta(\mathbf{W}^n \mathbf{h}_t^{(n-1)} + \mathbf{W}^n \mathbf{h}_{t-1}^{(n)} + \mathbf{b}) \tag{C.3}$$

For ease in notation we define $Z_1 = p$ as the dimensions of the hidden layers comprising the deep RNN. Based on Equation C.3, Figure C.2 reports the representation of a $N+1$ Deep RNN. As one could notice from Figure C.2, the characteristic time-layer structure of deep RNNs make them extremely deep neural networks (Goodfellow et al., 2016: Aggrawal, 2018) making the training of the *log-term* dependencies (e.g., high number of lags in a time series settings) extremely arduous. Goodfellow et al. (2016) explain how –even if the exploding/vanishing gradient problem is properly taken care of (see Glorot and Bengio, 2010)– the backpropagation algorithm (when training RNNs we usually refer to *backpropagation thorough time* as in Williams and Zipser, 1992) used to train the deep RNN assigns smaller weights to long-term dependencies when compared to short-term ones.

[121]See Pascanu et al. (2014) for the advantages of depth over width for deep RNNs.

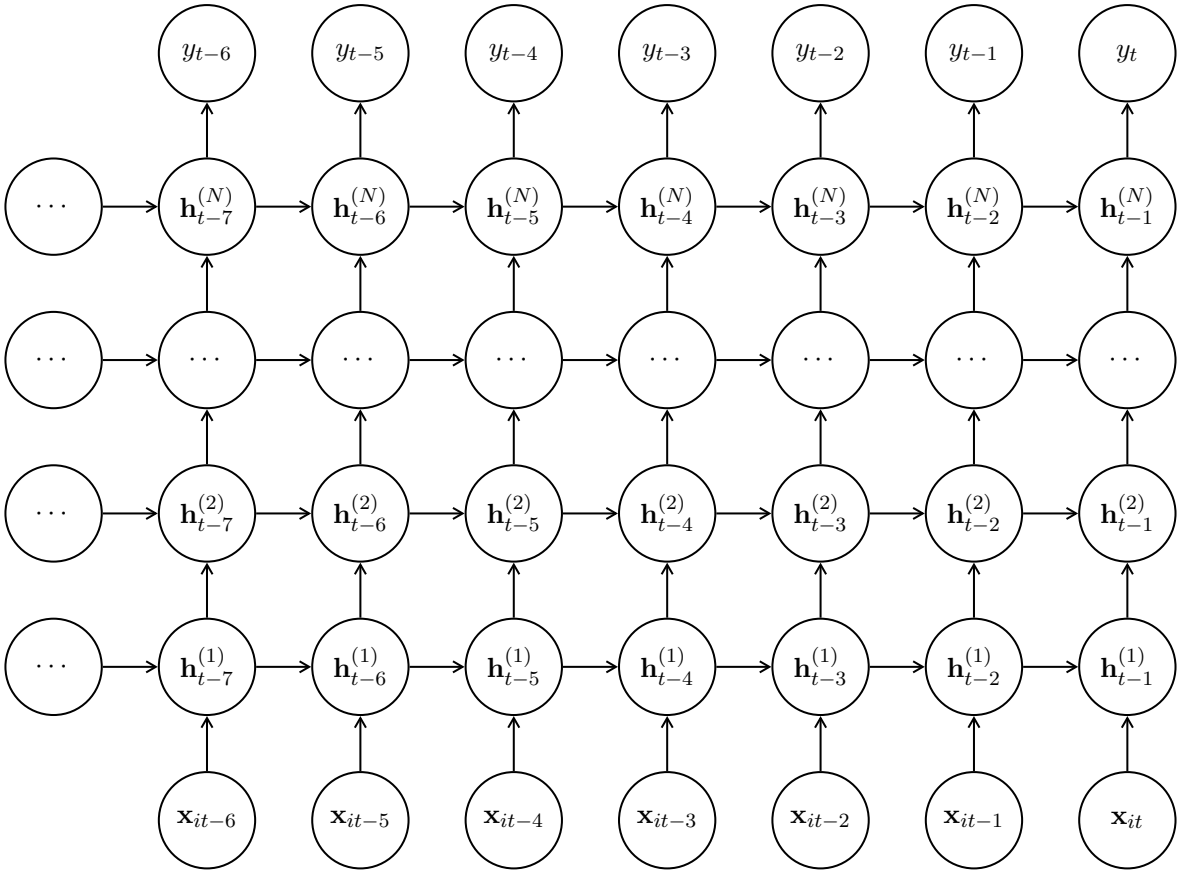


Figure C.2: Deep Recurrent neural network with no bias term

As a consequence, the deep RNN analyzed above can be considered effective in learning only short sequences (Aggrawal, 2018). The most notorious solution was firstly proposed by Hochreiter and Schmidhuber (1997) with the long short-term memory (LSTM) model. The underlying idea behind LSTM is the enforcing a constant error flow via special units to which we will refer as LSTM cells; each of these LSTM cells will be characterize by an internal recurrence. In particular, the cells states are updated during the backpropagation algorithm in order to guarantee greater information storage by decreasing the level of similarity (long-term memory) across different temporal layers. In particular, Aggrawal (2018) explains how the hidden vectors $\mathbf{h}_t^{(n)}$ and the cell state vectors $\bar{\mathbf{c}}_t^n$ are defined via a multi-step process that first determines a $4p$-dimensional intermediate vector comprising of *input* ($\bar{\mathbf{i}}$), *forget* ($\bar{\mathbf{f}}$), *output* ($\bar{\mathbf{o}}$), and c ($\bar{\mathbf{c}}$) gates (vectors of dimension p) which are then used to compute the hidden cells. Following Aggrawal (2018) we defined the intermediate vector as:

$$\begin{bmatrix} \bar{\mathbf{i}} \\ \bar{\mathbf{f}} \\ \bar{\mathbf{o}} \\ \bar{\mathbf{c}} \end{bmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \circ (\mathbf{W}^n \mathbf{h}_t^{(n-1)} + \mathbf{W}^n \mathbf{h}_{t-1}^{(n)}) \tag{C.4}$$

based on the above intermediate states, the value of the hidden layer $\mathbf{h}_t^{(n)}$ can be defined as:

$$\begin{aligned}\bar{\mathbf{c}}_t^{(n)} &= \bar{\mathbf{f}} \odot \bar{\mathbf{c}}_{t-1}^{(n)} + \bar{\mathbf{i}} \odot \bar{\mathbf{c}} \\ \mathbf{h}_t^{(n)} &= \bar{\mathbf{o}} \odot \tanh(\bar{\mathbf{c}}_t^{(n)})\end{aligned} \tag{C.5}$$

with $\odot$ capturing the Hadamard product (component-wise multiplication). The subsequent analysis of Equations C.4 and C.5 will be conducted following Aggrawal (2018). The input gate can be regarded as a boolean operator (although it is a continuous vector with values between 0 and 1) that is used to decide whether to create a cell state; the forget gate is used to decide whether to forget a cell state or not; and the output gate allows *leakage* from the hidden state to the cell state. Therefore, the input and forget gates –by regulating the amount of information that can be changed from the previous (time step) hidden node– control the flow of long-memory throughout the deep RNN. When focusing Equation C.5, we must distinguish between the update to the cell state and the update to the hidden state. In the first case, the relevant equation can be divided into two parts: I) the forget gate is used to decide which of the p cell states from $t-1$ are to be set equal to 0 (forgetting the past/long-memory), II) the input gate decides whether to add or not the elements of $\bar{\mathbf{c}}$ (to what extent incorporate new information into the past/long-memory). In the second case, the value of the hidden layer is defined using leakages from the cell state (the output gate defines how much information retain/leak from the cell state). As highlighted by Aggrawal (2018), it is not necessary to use the tanh activation function; it is possible to use any other activation function described in Appendix A.

C.2 Text generation

Having described the most important characteristics of both shallow and deep RNNs, the present section investigates the power of RNNS in text generation. In particular, starting from "An inquiry into the Nature and Causes of the Wealth of Nations" by Adam Smith (1776), a RNN with one LSTM hidden layer (with 128 hidden nodes) and an output layer with softmax activation function is trained and used to generate text (equivalent to forecasting the most likely letters) starting from a fixed-length sentence. To show the gradual training of the RNN, the forecasted texts obtained after 1 and 14 epochs –and a temperature of 0.5– are reported.

When the generating seed is "*ence a-day; tenpence, sometimes a shilling, about edinburgh*" the RNN –trained for 1 epoch– generates the following text:

"*and of the the beew so state of the expented of the stature, but the was can for the have been a contrary which the produce of a commence another a great price of the commerce of stock of the secless the commodity of the labour in a grands of their price of corn a the seems be diminually the this parts of the difficently the and to the world of the great part of this commerce their offect of can*"

Conversely, when the generating seed is "*hich they had made upon the prices either of corn, or of oth*" the RNN –trained for 14 epochs– generates the following text:

"*er farmer to the competition degrees of scotland, the price of the workmen from which the price of the different price of the with their silver, which supply the profits of the labourers and other the salen, it is the price of silver and extended into the profits of the trade of the real price of the countries. the quantity of labour, and worth the rent of the rude produce of the country with*"

One could notice that, as the number of epochs increases, the number of misspelled words reduces from seven to one, and the meaning of the forecasted sentence becomes clearer. It is expected that by increasing the number of epochs and/or increasing the network complexity, the performance of the trained RNN will also increase.