Journal of
Risk and Financial Management

MDPI

Article

Neural Network Models for Empirical Finance †

Hector F. Calvo-Pardo [1,*], Tullio Mancini [2], and Jose Olmo [3]

[1] Department of Economics, Highfield Campus, University of Southampton, SO17 1BJ, Southampton, UK; and CEPR and CPC
[2] University of Southampton; T.Mancini@soton.ac.uk
[3] University of Southampton and Universidad de Zaragoza; J.B.Olmo@soton.ac.uk
* Correspondence: calvo@soton.ac.uk
† Tullio Mancini acknowledges financial support from the University of Southampton Presidential Scholarship and Jose Olmo from 'Fundación Agencia Aragonesa para la Investigación y el Desarrollo'.

Received: date; Accepted: date; Published: date

check for updates

Abstract: This paper presents an overview of the procedures that are involved in prediction with machine learning models with special emphasis on deep learning. We study suitable objective functions for prediction in high-dimensional settings and discuss the role of regularization methods in order to alleviate the problem of overfitting. We also review other features of machine learning methods, such as the selection of hyperparameters, the role of the architecture of a deep neural network for model prediction, or the importance of using different optimization routines for model selection. The review also considers the issue of model uncertainty and presents state-of-the-art methods for constructing prediction intervals using ensemble methods, such as bootstrap and Monte Carlo dropout. These methods are illustrated in an out-of-sample empirical forecasting exercise that compares the performance of machine learning methods against conventional time series models for different financial indices. These results are confirmed in an asset allocation context.

Keywords: Machine Learning, Neural Networks, Dropout methods, LASSO techniques, Financial modeling.

1. Introduction

Statistical science has changed a great deal in the past ten years, and it is continuing to change, in response to technological advances in science and industry. The world is awash with big and complicated data, and researchers are trying to make sense out of it. While traditionally scientists fit a few statistical models by hand, they now use sophisticated computational tools in order to search through a large number of models, looking for meaningful patterns and accurate predictions. Standard statistical models have been extended in many ways. Models now allow for more predictors than observations, accommodating nonlinear relationships, interactions between the predictors, and, in particular, the presence of strong correlations (multicollinearity). One of the main advantages of these novel models based on machine learning techniques is the gain in predictive performance when compared to standard statistical models and the ease of manipulation due to the availability of toolboxes and off-the-shelf routines that make their implementation straightforward, even in large dimensions that are characterized by many covariates and increasingly complex datasets.

Nowadays, machine learning (ML) technology is widespread: from web searches to content filtering on social networks to recommendations on e-commerce websites. ML identifies objects in images, transcribes speech into text, matches news items, posts or products with users' interests, and selects the relevant results of the search, making use of a class of techniques, called deep learning. Deep learning allows for computational models that are composed of multiple processing layers to learn representations of big complex datasets, uncovering intricate structure within them. These methods

have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, and many other domains, such as drug discovery and genomics, being increasingly present in consumer products, such as cameras, smartphones, or computerized personal assistants. For example, Apple's Siri, Amazon's Alexa, Google Now, or Microsoft's Cortana employ deep neural networks to recognize, understand, and answer human questions.

A defining characteristic of machine learning models is its ability to accommodate a large set of potential predictor variables and different functional forms. The definition of machine learning is often context-specific. We use the term to describe a diverse collection of high-dimensional models for statistical prediction, combined with regularization methods for model selection that is based on a variety of penalty functions. The development of algorithms to implement the optimization procedures in an efficient manner is also an intrinsic part of this novel methodology. Machine learning was initially developed for prediction; this is particularly relevant in empirical finance, in which the object of interest is usually the prediction of an asset return or its conditional volatility. The literature on machine learning for empirical finance modeling has grown enormously in recent years, see, for example, Chinco et al (2019). These authors apply the Least Absolute Shrinkage and Selection Operator (lasso) to make rolling one-minute-ahead return forecasts using the entire cross-section of lagged returns as candidate predictors. The lasso increases both the out-of-sample fit and forecast-implied Sharpe ratios. This out-of-sample success comes from identifying predictors that are unexpected, short-lived, and sparse. Another recent influential study on empirical finance modeling is Gu et al. (2020). These authors perform a comparative analysis of machine learning methods for measuring asset risk premiums. These authors study a set of candidate models that include linear regression, generalized linear models with penalization, dimension reduction via principal components regression and partial least squares, and compare these methods against machine learning methods, such as regression trees (including boosted trees and random forests) and neural networks. This study demonstrates the presence of large economic gains to investors while using forecasts from regression trees and neural networks, in some cases doubling the performance of leading regression-based strategies from the literature. A more general treatment of the topic can be found in Friedman (1994), which provides an early unifying review across the relevant disciplines (applied mathematics, statistics, engineering, artificial intelligence, and connectionism); LeCun et al. (2015) that provides a general overview of deep learning, and Goodfellow et al. (2016), which provides a thorough textbook treatment.

Our aim in this paper is to present an overview of machine learning methods that complements the work of Gu et al. (2020). Rather than introducing the main features of the above methods for prediction in high-dimensional settings, we focus on feedforward neural networks and, in particular, in deep learning models. Our objective is to explain, in detail, the optimization problem that characterizes the prediction in machine learning models. Model overfit is an important feature of these models, due to the estimation of a large number of parameters. To correct for this, regularization methods are introduced and their properties discussed at length. We distinguish different types of penalty functions in a mean square prediction error setting and discuss the properties of methods, such as lasso, elastic net, or ridge regressions. We also pay particular emphasis to the role of the tuning parameters that determine the quality of the predictions, such as the length and depth of a neural network, the constant characterizing the contribution of the penalty function to the optimization problem, and the effect of other hyperparameters that are fine tuned through cross-validation, model dropout (e.g., Smyl 2020), and other optimization methods. In contrast to Gu et al. (2020), our overview is more focused on the understanding of the underlying mechanisms necessary to implement an artificial neural network. In this overview, we are also concerned with recent topics that have gained significant attention in the deep learning literature, such as the optimality of the architecture (e.g., Calvo-Pardo et al 2020) or the measurement of the uncertainty around model predictions. We discuss, in some detail, the choice of bootstrap methods, see Tibshirani (1996) for a simulation-based review of the topic, and the Monte Carlo dropout of Smyl (2020).

Finally, in the same spirit of Chinco et al (2019) and, more specifically, Gu et al. (2020), we also propose an application of these 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) as compared to linear regression and techniques that are based on dimension reduction, our empirical exercise performs a comparative study against conventional time series models that are widely used for empirical finance modeling. In particular, we present a forecasting exercise of the conditional mean and volatility of asset returns for three U.S. financial indices. Our objective in this section is twofold. First, we aim to assess the predictive performance of a modern deep neural network model and then 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 predictive performance is done by implementing a Diebold and Mariano (1995) test of predictive accuracy. The results of the empirical analysis provide overwhelming evidence in favor of the neural network model for the three financial indices. Second, as in Gu et al. (2020), we add economic significance to the comparison. To do this, we compare the Sharpe ratios between optimal portfolios that were constructed from a combination of the three financial indices. The optimal combination is obtained while using Markowitz 's (1952) mean-variance and minimum-variance portfolios as the investor's objective functions. Portfolio performance is done estimating an out-of-sample Sharpe ratio. The results confirm the above findings on the outperformance of machine learning models over sophisticated time series models in terms of economic performance.

The paper is structured as follows. Section 2 discusses the choice of suitable objective functions in machine learning problems. Section 3 presents recent advances in deep learning and focuses on deep neural networks. Section 4 studies the role of uncertainty in machine learning models and discusses recent advances on the analysis of uncertainty while using these novel procedures. Section 5 presents an empirical comparative study of these methods for modeling the conditional mean and volatility of financial returns for several financial indices. Section 6 summarizes the contributions of the study.

2. The Objective Function in Machine Learning Problems—Minimization Versus Regularization

This section lays out the optimization problem that is common in the machine learning literature. Machine learning describes a diverse collection of high-dimensional models for statistical prediction, combined with regularization methods for model selection and mitigation of overfitting. The high-dimensional nature of machine learning enhances the flexibility of the methodology relative to more traditional econometric prediction techniques. However, with enhanced flexibility comes a higher propensity to overfitting the data. Therefore, it is necessary to consider objective functions that penalize the excessive parametrization of the model. The final goal of machine learning methods is to achieve an approximate optimal specification with a manageable computational cost. In this section, we describe candidate optimization functions for supervised and unsupervised machine learning problems and then discuss the role of regularization.

2.1. Unsupervised Learning

ML algorithms can be broadly categorized as unsupervised or supervised. Unsupervised learning algorithms aim at uncovering useful properties of the structure of the input dataset, i.e., there is no y, and given that the true data generating process (DGP) $p_{\text{data}}(\mathbf{X})$ is unknown, the goal is to learn $p_{\text{data}}(\mathbf{X})$, or some useful properties of it, from a random sample of $i = 1...N$ realizations of input data only, $\{\mathbf{X}_i\}$, on the basis of which the empirical distribution $\widehat{p}_{\text{data}}(\mathbf{X})$ obtains. Letting $p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})$ be a parametric family of probability distributions indexed by $\boldsymbol{\theta}$ that estimates the unknown true

$p_{\text{data}}(\mathbf{X})$, unsupervised learning corresponds to finding the parameter vector $\boldsymbol{\theta}$ that minimizes the dissimilarity/distance between $p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})$ and $\widehat{p}_{\text{data}}(\mathbf{X})$**:**

$$\boldsymbol{\theta}_{ML} \in \arg\min_{\boldsymbol{\theta}} D_{KL}(\widehat{p}_{\text{data}}||p_{\text{model}}) \equiv \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X}\sim\widehat{p}_{\text{data}}}[\log \widehat{p}_{\text{data}}(\mathbf{X}) - \log p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})] \tag{1}$$

noticing that $\boldsymbol{\theta}_{ML}$ is the maximum likelihood estimaton and $D_{KL}(\widehat{p}_{\text{data}}||p_{\text{model}})$ denotes the Kullback–Leibler divergence. To obtain this, we note that

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbf{X};\boldsymbol{\theta}) \tag{2}$$

and $p_{\text{model}}(\mathbf{X};\boldsymbol{\theta}) = \prod_{i=1}^{N} p_{\text{model}}(\mathbf{X}_i;\boldsymbol{\theta})$, which, after taking logs and dividing by N, is equivalent to

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N} \log p_{\text{model}}(\mathbf{X}_i;\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{X}\sim\widehat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})],$$

by the analogy principle.

The cross-entropy in the above expression is simply $-\mathbb{E}_{\mathbf{X}\sim\widehat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})]$: since $\log \widehat{p}_{\text{data}}(\mathbf{X})$ does not depend on $\boldsymbol{\theta}$, minimizing D_{KL} is equivalent to minimizing the cross-entropy, or 'empirical risk minimization', e.g., the mean-squared error is the cross-entropy between the empirical distribution and a Gaussian model. In machine learning (ML), the cross-entropy is called 'cost function', $J(\boldsymbol{\theta})$, while, in statistics, it is called the 'loss function', $l(\boldsymbol{\theta}) \equiv L[\widehat{p}_{\text{data}}(\mathbf{X}), p_{\text{model}}(\mathbf{X};\boldsymbol{\theta})]$. Examples of popular unsupervised deep learning models, not necessarily parametric, are k-means clustering, auto-encoders, and generative adversarial networks (GANs).

2.2. Supervised Learning

Supervised learning methods aim to develop a computational relationship (formula/algorithm) between P inputs (predictors, features, explanatory or independent variables), $\mathbf{X} = \{...x_p...\}$, and K outputs (dependent or response variables), $\mathbf{y} = \{...y_k...\}$, for determining/predicting/estimating values for $\mathbf{y}$, given only the values of $\mathbf{X}$, in the presence of unobserved/uncontrolled quantities $\mathbf{z} = \{...z_u...\}$:

$$y_k = g_k(...x_p...; ...z_u...), \forall k,$$

where $g_k(\cdot)$ denotes a functional form relating the input observed variables, the unobserved variables and the dependent variables. In order to reflect the uncertainty that is associated with the unobserved inputs **z,** the above relationship is replaced by a statistical model:

$$y_k = f_k(...x_p...) + \varepsilon_k : \varepsilon_k \sim F_\varepsilon(\varepsilon_k), \mathbb{E}[\varepsilon_k|...x_p...] = 0, \forall k.$$

By construction, this model satisfies that $\mathbb{E}_\varepsilon[y_k|...x_p...] = f_k(...x_p...)$, with $\mathbb{E}_\varepsilon[\cdot|\Im]$ denoting the conditional expectation evaluated under the distribution function of the error term conditional on the information set $\Im$. For simplicity, we drop the k subscript, which indicates that we are assuming that there are separate models for each output k, ignoring that they depend on the same set of input variables:

$$y = f(\mathbf{X}) + \varepsilon : f(\mathbf{X}) = \mathbb{E}_\varepsilon[y|\mathbf{X}] \tag{3}$$

i.e., to the extent that the error term ε is a random variable, the output variable y becomes a random variable.[1] Specifying a set of observed input values $\mathbf{X}$, specifies a distribution of output values, y, the mean of which is the target function $f(\mathbf{X})$. The input and output variables can be real or categorical, but categories can be always converted into 'indicators' or 'dummies' that are real-valued.

---

[1] In practice, strategies that treat the K outputs as a joint system often improve accuracy.

More specifically, supervised learning algorithms aim to obtain a useful approximation $\widehat{f}(\mathbf{X})$ to the true (unknown) 'target' function $f(\mathbf{X})$ in (3) by modifying (under constraints) the input/output relationship $\widehat{f}(\mathbf{X})$ that it produces, in response to differences $\{y_i - \widehat{y}_i\}$ (errors) between the predicted $\widehat{y}_i = \widehat{f}(\mathbf{X}_i)$ and real y_i system outputs:

$$\widehat{f}(\mathbf{X}) \in \arg\min_{g(\mathbf{X})} \frac{1}{N} \sum_{i=1}^{N} L[y_i, g(\mathbf{X}_i)] \tag{4}$$

where $L(\cdot,\cdot)$ is the 'loss function', or a measure of distance (error) between y_i and $\widehat{y}_i = \widehat{f}(\mathbf{X}_i)$. Common examples are $L[y_i, \widehat{y}_i] = |y_i - \widehat{y}_i|$ which plugged into (4) corresponds to selecting the median, *Med*, of the conditional distribution. More formally, $\widehat{f}(\mathbf{X}) = Med_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}}[y|\mathbf{X}]$ that minimizes the Mean Absolute Error (MAE), or $L[y_i, \widehat{y}_i] = [y_i - \widehat{y}_i]^2$, which selects the $\widehat{f}(\mathbf{X}) = \mathbb{E}_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}}[y|\mathbf{X}]$ that minimizes the Mean Squared Error (MSE) in (4). Alternatively stated, consider a random sample of $i = 1...N$ realizations, $\{y_i, \mathbf{X}_i\}$, constituting the empirical distribution $\widehat{p}_{\text{data}}(y, \mathbf{X})$, the goal of supervised learning is to learn to predict y from $\mathbf{X}$, estimating $p(y|\mathbf{X})$. Letting $p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta})$ be a parametric family of probability distributions that are indexed by $\boldsymbol{\theta}$ that estimates the unknown true $p(y|\mathbf{X})$, supervised learning corresponds to finding the parameter vector $\boldsymbol{\theta}$ that minimizes the dissimilarity/distance between $p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta})$ and $\widehat{p}_{\text{data}}(y|\mathbf{X})$**:**

$$\boldsymbol{\theta}_{ML} \in \arg\min_{\boldsymbol{\theta}} D_{KL}(\widehat{p}_{\text{data}} || p_{\text{model}}) \equiv \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}}[\log \widehat{p}_{\text{data}}(y|\mathbf{X}) - \log p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta})] \tag{5}$$

and again, solving (5) is equivalent to cross-entropy minimization,

$$\min_{\boldsymbol{\theta}} - \mathbb{E}_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}} \log p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta}).$$

As an example, notice that, if we set $p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta}) = N(g(\mathbf{X};\boldsymbol{\theta}), \sigma^2)$ in (5), with $N(\cdot,\cdot)$ a Normal distribution, we obtain:

$$\begin{aligned} \min_{\boldsymbol{\theta}} - \mathbb{E}_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}}[\log p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta})] &= \min_{\boldsymbol{\theta}} - \frac{1}{N} \sum_{i=1}^{N} \log p_{\text{model}}(y_i|\mathbf{X}_i;\boldsymbol{\theta}) \\ &= \min_{\boldsymbol{\theta}} \left\{ \log(\sigma[2\pi])^{1/2} + \left[2\sigma^2\right]^{-1} \underbrace{\frac{1}{N} \sum_{i=1}^{N} [y_i - g(\mathbf{X}_i;\boldsymbol{\theta})]^2}_{\equiv MSE(\boldsymbol{\theta})} \right\} \end{aligned}$$

and therefore, cross-entropy minimization corresponds to mean squared error (MSE) minimization when the model is hypothesized to be Gaussian with mean $g(\mathbf{X};\boldsymbol{\theta})$. In addition, this example shows that optimally choosing the parameter vector $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta}_{ML}$, which characterizes $\widehat{f}(\mathbf{X}) = g(\mathbf{X};\widehat{\boldsymbol{\theta}})$, is equivalent to solving (4) when $L[y_i, \widehat{y}_i] = [y_i - \widehat{y}_i]^2$:

$$\widehat{f}(\mathbf{X}) \in \arg\min_{g(\mathbf{X})} \mathbb{E}_{y,\mathbf{X}\sim\widehat{p}_{\text{data}}}[y - g(\mathbf{X})]^2 = \arg\min_{g(\mathbf{X})} \frac{1}{N} \sum_{i=1}^{N} [y_i - g(\mathbf{X}_i)]^2$$

Therefore, approximating/learning the unknown function $f(\mathbf{X})$ corresponds to estimating the unknown true conditional probability $p(y|\mathbf{X})$, once we conjecture a parameterization $p_{\text{model}}(y|\mathbf{X};\boldsymbol{\theta})$ for it. Popular supervised deep learning models, which are not necessarily parametric, are support vector machines (SVMs) based on kernel methods, k-nearest neighbor regression, or decision trees.

Notice that (4) is the available sample $\{y_i, \mathbf{X}_i\}$ analog to solving for the global prediction error in (3):

$$\widehat{f} \in \arg\min_{g(\mathbf{X})} \int \{\mathbb{E}_{\varepsilon} L[f(\mathbf{X}) + \varepsilon, g(\mathbf{X})]\} \, p_{\text{data}}(\mathbf{X}) d\mathbf{X} \tag{6}$$

where $p_{\text{data}}(\mathbf{X})$ is the unknown true data generating process. As an example, replace $L[y_i, \widehat{y}_i] = [y_i - \widehat{y}_i]^2$ in (6) in order to obtain the standard expressions for the bias-variance trade-off in the Mean Squared Error (MSE):

$$\begin{aligned} \widehat{f} &\in \arg\min_{g(\mathbf{X})} \int \left\{ \mathbb{E}_\varepsilon [f(\mathbf{X}) + \varepsilon - g(\mathbf{X})]^2 \right\} p_{\text{data}}(\mathbf{X}) d\mathbf{X} \\ &= \arg\min_{g(\mathbf{X})} \underbrace{\int [f(\mathbf{X}) - g(\mathbf{X})]^2 p_{\text{data}}(\mathbf{X}) d\mathbf{X}}_{\text{MSE}(\widehat{f})} \\ &\quad + \underbrace{\int \left\{ \mathbb{E}_\varepsilon [\varepsilon^2 | \mathbf{X}] \right\} p_{\text{data}}(\mathbf{X}) d\mathbf{X}}_{\text{Variance of the noise } \varepsilon} \end{aligned}$$

where the MSE$(\widehat{f})$ denotes the MSE of $\widehat{f}(\mathbf{X})$ averaged over all training samples of size N that could be realized from the system with probabilities that are governed by $p_{\text{data}}(\mathbf{X})$ and $F_\varepsilon(\varepsilon)$. It can be further decomposed as:

$$MSE(\widehat{f}) \equiv \int MSE[\widehat{f}(\mathbf{X})] p_{\text{data}}(\mathbf{X}) d\mathbf{X} = \int Var[\widehat{f}(\mathbf{X})] p_{\text{data}}(\mathbf{X}) d\mathbf{X} + \int Bias^2[\widehat{f}(\mathbf{X})] p_{\text{data}}(\mathbf{X}) d\mathbf{X}$$

where $Bias^2[\widehat{f}(\mathbf{X})] = \{f(\mathbf{X}) - \mathbb{E}_\varepsilon[\widehat{f}(\mathbf{X})]\}^2$ measures the square of the difference between the target function $f(\mathbf{X})$ and the average approximation value at a particular sample $\mathbf{X}$, $\mathbb{E}_\varepsilon[\widehat{f}(\mathbf{X})]$.

Problem (6) defines the target performance measure for prediction in supervised learning/function approximation: as future new input only observations become available, collected in a prediction or test sample '$\top$', $\{y_i, \mathbf{X}_i\}_{i=1}^{N^\top}$, we want to predict (estimate) a likely output value using $\widehat{f}(\mathbf{X}_i)$, such that $\widehat{y}_i = \widehat{f}(\mathbf{X}_i)$, where $\widehat{f}(\mathbf{X})$ was obtained from (4) exploiting the available sample, $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$. Computing then $\frac{1}{N^\top} \sum_{i=1}^{N^\top} L[y_i, \widehat{y}_i]$ allows for the researcher to evaluate the out-of-sample performance of the algorithm/function approximation $\widehat{f}(\mathbf{X})$, showing that accurate approximation and future prediction are one and the same objective.[2] As future data is unavailable, the standard practice is to divide the available sample $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$ into two disjoint parts: a training/learning sample '$\llcorner$' $\{y_i, \mathbf{X}_i\}_{i=1}^{N^\llcorner}$ in (4) where $\widehat{f}(\mathbf{X})$ obtains, and a prediction/test sample $\{y_i, \mathbf{X}_i\}_{i=1}^{N^\top}$ where the out-of-sample predictive performance of $\widehat{f}(\mathbf{X})$ is evaluated, so that $N = N^\llcorner + N^\top$.

More complex forms of the unknown target function $f(\mathbf{X})$ naturally call for bigger training samples $N^\llcorner$ in order to obtain better representations/approximations $\widehat{f}(\mathbf{X})$. However, this comes at the expense of increasing the chances of $\widehat{f}(\mathbf{X})$ 'overfitting'. Overfitting happens when a model that represents the training data very well represents very poorly unseen data $N^\top$ in the 'prediction/test phase'.[3] The reason for overfitting lies on the 'curse-of-dimensionality' that the complexity of the

---

[2] Yet another goal of supervised learning is interpretation, as opposed to prediction: there, interest lies in the structural form of the approximating function that was obtained from (4) in order to understand the mechanism that produced the data. The identification of the input variables that are most relevant to explain the variation in output, or the nature of that dependence and how it changes with changes in other inputs, are instead the primary objectives, and the aim is to understand how the system works.

[3] An intuitive way to understand why is as follows. Suppose that we have a sample of size N with which we are trying to approximate a function of N variables $f(x_1, ..., x_N)$. If Kolmogorov's conjecture was right, then we could instead approximate a degree N polynomial function of just one variable, say x_1, $f(x_1, ..., x_N) = g(x_1) = \sum_{i=1}^{N} a_i x_1^i$ and problem (4) would reduce to a parametric least squares (OLS) solution:

$$\widehat{f}(\mathbf{X}) = f(\mathbf{X}; \widehat{\mathbf{a}}) \in \arg\min_{\{a_i\}} \frac{1}{N} \sum_{i=1}^{N} [y_i - \sum_{i=1}^{N} a_i x_1^i]^2$$

Because there are N normal equations (one for each) in N unknowns (sample observations), we would obtain a unique solution $\widehat{\mathbf{a}}$, corresponding to a 'perfect fit' of the sample/training data. If then one more sample observation was collected, $N^\top = \{y^{N+1}, x_1^{N+1}\}$, and we wanted to test the predictive ability of $\widehat{f}(\mathbf{X}) = \sum_{i=1}^{N} \widehat{a}_i x_1^i$, almost with probability one $y^{N+1} \neq \widehat{y}^{N+1} = \sum_{i=1}^{N} \widehat{a}_i (x_1^{N+1})^i$, i.e., the prediction error $[y^{N+1} - \widehat{y}^{N+1}]^2$ will be very big, indicating 'overfitting'. In big

unknown target function creates: as the number of input variables P upon which $f(\mathbf{X})$ depends increases, the necessary sample size for accurately approximating $f(\mathbf{X})$ grows exponentially, i.e., at a rate $N^{1/P}$, rendering all training samples very sparsely populated. Note that this is the case, even if we set $\varepsilon = 0$ in (3), converting (4) into an interpolation problem, i.e. reducing the MSPE to an MSE-only problem still requires a large enough training sample for the approximation to be accurate.

Because $N^{\llcorner}$ is finite, problem (4) does not have a unique solution[4]. Therefore, one must restrict the set of admissible functions to a smaller set $\mathcal{G}$ than the set of all possible functions $g(\mathbf{X})$. To see the effect of restricting the class of admissible functions in (4), denote, by $f^*(\mathbf{X}) \in \arg\min_{g(\mathbf{X})} \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, g(\mathbf{X}_i)]$ and by $f^*_{\mathcal{G}}(\mathbf{X}) \in \arg\min_{g(\mathbf{X})\in\mathcal{G}} \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, g(\mathbf{X}_i)]$ the best approximation in the unrestricted and restricted classes of functions, respectively, both in terms of out-of-sample performance, $N^{\top}$. The difference in out-of-sample performance between the solution from (4) and $f^*(\mathbf{X})$ ('excess test error' $\mathcal{E}$) can then be decomposed, as follows:

$$\begin{aligned}\mathcal{E} &\equiv \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, \widehat{f}(\mathbf{X}_i)] - \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, f^*(\mathbf{X}_i)] \\ &= \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} \underbrace{\{L[y_i, \widehat{f}(\mathbf{X}_i)] - L[y_i, f^*_{\mathcal{G}}(\mathbf{X}_i)]\}}_{\text{Estimation error}} + \underbrace{\{L[y_i, f^*_{\mathcal{G}}(\mathbf{X}_i)] - L[y_i, f^*(\mathbf{X}_i)]\}}_{\text{Approximation error}}.\end{aligned}$$

The approximation error increases the more restrictive the class of functions $\mathcal{G}$ is, unless the true unknown target function $f(\mathbf{X})$ happens to belong to $\mathcal{G}$, in which case $f^*_{\mathcal{G}}(\mathbf{X}) = f^*(\mathbf{X})$. The estimation error depends on how good the algorithm/approximation $\widehat{f}(\mathbf{X})$ is (1st term) as well as on how well the selected class of functions $\mathcal{G}$ can best represent the complexity of the unknown target function $f(\mathbf{X})$ (second term).

'Universal approximators' for the class of all continuous target functions $f(\mathbf{X})$ are classes of functions $\mathcal{G} = \{g(\mathbf{X}) : g(\mathbf{X}) = \sum_{z=1}^{Z} a_z b(\mathbf{X}|\gamma_z), \gamma_z \in \mathbb{R}^q\}$ that could exactly represent $f(\mathbf{X})$ if the sample size was not finite, i.e., $f(\mathbf{X}) = \sum_{z=1}^{\infty} a^*_z b(\mathbf{X}|\gamma_z)$ for some set of expansion coefficient values $\{a^*_z\}_{z=1}^{\infty}$, and that nonetheless approximate well with a small number Z of coefficients. Therefore, universal approximators minimize the approximation error and estimation error, minimizing the out-of-sample performance difference $\mathcal{E}$ between the solution from (4) and $f^*(\mathbf{X})$, i.e., if the training sample size was infinite, $\lim_{N^{\llcorner}\to\infty} \widehat{f}(\mathbf{X}) = f(\mathbf{X};\widehat{\boldsymbol{\theta}}) = \sum_{z=1}^{\infty} \widehat{a}_z b(\mathbf{X}|\widehat{\gamma}_z) = \sum_{z=1}^{\infty} a^*_z b(\mathbf{X}|\gamma_z) = f(\mathbf{X})$ with $\widehat{\boldsymbol{\theta}} = \widehat{\boldsymbol{\theta}}_{ML} = \{\widehat{a}_z, \widehat{\gamma}_z\}_{z=1}^{\infty}$, and therefore, $\lim_{N^{\top}\to\infty} \frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, \widehat{f}(\mathbf{X}_i)] = 0$ ('Oracle property'). However, because the training sample size is finite, $Z < \infty$ and $\frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, \widehat{f}(\mathbf{X}_i)] > 0$. Choosing Z corresponds then to 'model selection': as entries $\{a_z\}_{z=1}^{Z}$ are added, the approximation is able to better fit the training data, increasing the variance component of (6), but decreasing the bias. The bias decreases because adding entries enlarges the function space spanned by the approximation $\widehat{f}(\mathbf{X})$. With a finite sample size, the goal is to choose a small Z that keeps the variance and bias small, so that (6) can be expected to remain small.

Examples of function classes that are universal approximators beyond feed-forward neural networks (described below), are radial basis functions, tensor product methods, and regression trees. Regression trees and their extension, Random Forests, are 'tree-structured' methods that are commonly used for flexibly estimating regression functions where out-of-sample performance is important. 'Tree-structured' methods have dictionaries of the form $\{\mathbf{1}_{\{\mathbf{X}\in R\}}\}_R$, where $\mathbf{1}_{\{.\}}$ is an indicator function,

---

data problems, where $P > N$ (or is close to N), overfitting means that the approximation obtained from (4) will almost surely perform poorly in unseen data, i.e., in (6).

[4] If $N^{\llcorner} = +\infty$ (and with an infinitely fast computer), then we would directly compute $f(\mathbf{X})$ from (3) predicting the mean of y for each value of $\mathbf{X}$.

and R represents subregions of the space of all possible values of $\mathbf{X} \in \mathbb{R}^P$, $R \subseteq \mathbb{R}^P$. The most common example is $\mathbf{1}_{\{\mathbf{X}\in R\}} = \prod_{p=1}^{P} \mathbf{1}_{\{u_p \leq x_p \leq v_p\}}$, with the $2P$ coefficients $\{u_p, v_p\}_{p=1}^{P}$ representing the respective lower and upper limits of the region (hyper-rectangle) on each input x_p axis. Usually, only Z disjoint regions are chosen, $\{R_z\}_{z=1}^{Z}$, so that $\mathbf{X} \in R_z \Longrightarrow \widehat{f}(\mathbf{X}) = a_z$, meaning that $\mathbf{X}$ in the same region have the same 'approximation' value a_z (with an obvious abuse of notation, but with a similar interpretation). Recursive partitioning tree-structured methods are also universal approximators, in the sense defined previously, i.e., $\widehat{f}(\mathbf{X}) = \sum_{z=1}^{\infty} a_z^* \mathbf{1}_{\{\mathbf{X}\in R_z\}} = f(\mathbf{X})$. Choosing the optimal number of regions Z is a formidable combinatorial optimization problem, but recursive partitioning is an approximate solution when employing greedy optimization strategies. This effectively results in sequentially splitting the initial sample $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$, starting with the single covariate x_p that minimizes the mean-squared error of the resulting subsamples (or leaves). When considering one different covariate at a time, the mean-squared error is therefore sequentially reduced. However, too many subsamples (a very deep tree) would correspond to a very large Z, which risks overfitting. Therefore, in practice, a very deep tree is estimated and then pruned (or regularized) to a more sparse tree, using cross-validation to select the optimal depth.[5]

2.3. Regularization Methods

In general, the choice of the set of admissible functions $\mathcal{G}$ is based on considerations outside the data and it is usually done by the choice of a learning method.[6] Choosing a learning method can be modeled as adding a penalty term $\lambda\Omega[g(\mathbf{X})]$ to restrict solutions to (4):

$$\widehat{f}(\mathbf{X};\lambda) \in \arg\min_{g(\mathbf{X})} \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} L[y_i, g(\mathbf{X}_i)] + \lambda\Omega[g(\mathbf{X})] \tag{7}$$

where λ ('regularization parameter') modulates the strength of the penalty functional $\Omega[\cdot]$ over all possible functions $g(\mathbf{X})$. The choice of a penalty functional is made on the basis of 'outside the data information' about the unknown target $f(\mathbf{X})$, e.g., on the basis of a prior over the class of models $g(\mathbf{X})$, $\Pr[g(\mathbf{X})]$. A natural choice for $\widehat{f}(\mathbf{X})$ would then be the function that is most probable given the data:

$$\widehat{f}(\mathbf{X}) \in \arg\max_{g(\mathbf{X})} \Pr[g(\mathbf{X})|\{y_i, \mathbf{X}_i\}] \tag{8}$$

which is known as maximum a posteriori probability (MAP) estimate. According to Bayes' theorem, the probability of a model given the training data is proportional to the likelihood that the training data have been generated by the model times the probability of the model:

$$\Pr[g(\mathbf{X})|\{y_i, \mathbf{X}_i\}] \sim \Pr[\{y_i, \mathbf{X}_i\}|g(\mathbf{X})] \Pr[g(\mathbf{X})], \tag{9}$$

If $\Pr[\{y_i, \mathbf{X}_i\}|g(\mathbf{X})] = N(0, \sigma^2)$ then (3) implies that

$$\Pr[\{y_i, \mathbf{X}_i\}|g(\mathbf{X})] = \Pr[\{\mathbf{X}_i\}] \prod_{i=1}^{N^{\llcorner}} (2\pi\sigma)^{-1} \exp\{-\varepsilon_i^2/2\sigma^2\}$$

with $\varepsilon_i = y_i - g(\mathbf{X}_i)$. Substituting the above expression into (9), taking logs and discarding terms not involving $g(\mathbf{X})$ yields an equivalent expression to (8):

$$\widehat{f}(\mathbf{X}) \in \arg\min_{g(\mathbf{X})} \frac{1}{\sigma^2} \sum_{i=1}^{N^{\llcorner}} [y_i - g(\mathbf{X}_i)]^2 - 2\log\Pr[g(\mathbf{X})]$$

---

5 See Friedman (1994) or Athey and Imbens (2019) for further details.

6 The class of functions $g(\mathbf{X}) = \sum_{m=1}^{M} a_m b(\mathbf{X}|\gamma_m)$, $\gamma_m \in \mathbb{R}^q$ are commonly known as 'dictionaries'. The choice of a learning method selects a particular dictionary. Examples of dictionaries that are universal approximators are feed-forward neural networks, radial basis functions, recursive partitioning tree-structured methods, and tensor product methods. See Friedman (1994) for additional details.

that coincides with (7) if $L(\cdot,\cdot)$ is the quadratic loss function and $\lambda\Omega[g(\mathbf{X})] = -2\sigma^2 \log \Pr[g(\mathbf{X})]$. The quantity $\lambda\Omega[g(\mathbf{X})]$ naturally captures that reductions in the noise variance σ^2 lead to increasing weight on the training data part $\Pr[\{y_i, \mathbf{X}_i\}|g(\mathbf{X})]$ in determining the approximation $\widehat{f}(\mathbf{X})$, relative to the prior $\Pr[g(\mathbf{X})]$. For example, restricting $g(\mathbf{X}) \in \mathcal{G}$, as above, can be achieved by setting $\Omega[g(\mathbf{X})] = H\{bias^2[g(\mathbf{X})]\}$ with $H\{h\} = 0 \cdot \mathbf{1}_{\{h=0\}} + \infty \cdot \mathbf{1}_{\{h\neq 0\}}$ (with the convention that $\infty \cdot 0 = 0$), since, when $h = 0 = bias^2[g(\mathbf{X})] \Leftrightarrow g(\mathbf{X};\widehat{\boldsymbol{\theta}}) = \sum_{z=1}^{Z} \widehat{a}_z b(\mathbf{X}|\widehat{\boldsymbol{\gamma}}_z)$, i.e., learning $\widehat{f}(\mathbf{X};\lambda)$ in (4) reduces to parameter learning, $\widehat{f}(\mathbf{X};\lambda) = g(\mathbf{X};\widehat{\boldsymbol{\theta}},\lambda)$,where $\boldsymbol{\theta} = \{a_z, \boldsymbol{\gamma}_z\}_{z=1}^{Z}$.

Additional parametric or non-parametric penalty terms can be added to (7), with the result of further restricting the solutions in the approximation subspace of $\mathcal{G}$ that respect that particular penalty. By the addition of a penalty term (or 'regularization'), the aim is to improve the out-of-sample performance of the approximation $\widehat{f}(\mathbf{X};\lambda)$, reducing its chances to 'overfit', without affecting its training error. Non-parametric penalties can be of the form $\Omega[g(\mathbf{X})] = \int |Dg(\mathbf{X})|^2 d\mathbf{X}$, where, for example, $|Dg(\mathbf{X})|^2 = \sum_{j=1}^{n} (\frac{\partial g}{\partial x_j})^2$ is the norm of the gradient of the functions in the class, with larger values of λ penalizing functions that oscillate more (i.e., that are 'less smooth').

Parametric penalties would, instead, penalize functions $g(\mathbf{X})$ not in a particular parametric family $k(\mathbf{X}|\boldsymbol{\theta})$. That is, $g(\mathbf{X}) \notin \{k(\mathbf{X}|\boldsymbol{\theta}), \boldsymbol{\theta} \in \mathbb{R}^q\} \Longrightarrow \Omega[g(\mathbf{X})] = \infty$, transforming (4) into an equivalent parameter estimation problem:

$$\widehat{\boldsymbol{\theta}}_\lambda \in \arg\min_{\boldsymbol{\theta}} \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} L[y_i, k(\mathbf{X}_i|\boldsymbol{\theta})] + \lambda\varpi[\boldsymbol{\theta}] \tag{10}$$

where the penalty function $\varpi[\boldsymbol{\theta}]$ admits different forms that are widely used in the recent ML literature: (i) 'ridge' (L^2 regularization): $\varpi[\boldsymbol{\theta}] = \sum_{j=1}^{q} \theta_j^2$, penalizing approximations with large parameter values[7]; (ii) 'subset selection': $\varpi[\boldsymbol{\theta}] = \sum_{j=1}^{q} \mathbf{1}_{\{\theta_j\neq 0\}}$, which penalizes approximations with a large number of parameters (requiring combinatorial optimization); and, (iii) 'bridge': $\varpi_v[\boldsymbol{\theta}] = \sum_{j=1}^{q} |\theta_j|^v$, which coincides with 'ridge' when $v = 2$ and it is a continuous approximation of the subset selection penalty as $v \to 0$. When $v = 1$, L^1 regularization obtains, akin to the 'least absolute shrinkage and selection operator', popularly known as LASSO; (iv) 'weight decay': $\varpi_w[\boldsymbol{\theta}] = \sum_{j=1}^{q} \frac{(\theta_j/w)^2}{1+(\theta_j/w)^2}$ approaches 'ridge' as $w \to \infty$ and subset selection as $w \to 0$. Smaller values of v and w privilege approximations with a small number of parameters. (v) '(Stochastic) Gradient descent': $\varpi[\boldsymbol{\theta}] = \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} L[y_i, k(\mathbf{X}_i|\boldsymbol{\theta})]$,, which penalizes 'paths' that do not follow the 'steepest descent', $\nabla_{\boldsymbol{\theta}}\varpi[\boldsymbol{\theta}] = \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} \nabla_{\boldsymbol{\theta}} L[y_i, k(\mathbf{X}_i|\boldsymbol{\theta})]$, when searching for the value $\widehat{\boldsymbol{\theta}}_\lambda$ that minimizes (10) with $\widehat{f}(\mathbf{X};\lambda) = k(\mathbf{X}|\widehat{\boldsymbol{\theta}})$, i.e. a high value of λ privileges '$\tau-$paths' $\boldsymbol{\theta}_{\tau+1} = \boldsymbol{\theta}_\tau - \epsilon\nabla_{\boldsymbol{\theta}}\varpi[\boldsymbol{\theta}_\tau]$ that reach $\widehat{\boldsymbol{\theta}}_\lambda$ taking the least possible number of steps τ, each of which depends on ϵ or 'learning rate'. Because ϵ governs the strength of the gradient $\nabla_{\boldsymbol{\theta}}\varpi[\boldsymbol{\theta}_\tau]$ in the updating of $\boldsymbol{\theta}_\tau$, choosing λ is equivalent to the choice of ϵ, a free hyperparameter to be 'fine tuned' or optimized during training.

When instead of using all available $N^{\llcorner}$ observations in the training sample, we randomly subsample from $\{y_i, \mathbf{X}_i\}$ and form a 'minibatch' with $B < N^{\llcorner}$ observations, $\varpi[\boldsymbol{\theta}] = \frac{1}{B} \sum_{i=1}^{B} L[y_i, k(\mathbf{X}_i|\boldsymbol{\theta})]$ is called a 'stochastic gradient descent (SGD) penalty'. SGD can be combined with 'momentum', where the size of the updating step depends on how large an exponentially decaying moving average sequence of past gradients is, $\alpha : \boldsymbol{\theta}_{\tau+1} = \boldsymbol{\theta}_\tau - \frac{\epsilon}{1-\alpha}\nabla_{\boldsymbol{\theta}}\varpi[\boldsymbol{\theta}_\tau]$. Momentum then adds another hyperparameter α, with larger values of $\alpha \in (0,1)$ corresponding to a higher reliance on previous gradients, leading to a larger step size when updating. Current optimization methods, like AdaGrad, RMSProp, or Adam, supplement SGD (with or without 'momentum') to allow the learning rate ϵ to

---

[7] 'Early stopping' the number of training iterations ('epochs') over the learning sample once the out-of-sample performance of the approximation starts to increase, can be shown to be equivalent to L^2 regularization (Goodfellow et al. 2016). Similarly, 'dropout' when applied to neural network (NN) methods, has been shown to be equivalent to L^2 regularization with a penalty strength parameter λ that is inversely proportional to the precision of the prior of a deep Gaussian process characterizing the NN parameters (Gal and Ghahramani 2016).

'adapt', shrinking or expanding according to the entire history. For example, Adam combines RSMProp and momentum, which is directly incorporated with exponential decay rates, $\rho_1, \rho_2 \in [0,1)$, for the first two moment estimates, s_1 and s_2, of the gradient $\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]$, initialized at the origin, $s_1 = s_2 = 0$. Subsequently, the bias-corrected updates of the first and second moments, $\widehat{s}_1 = \frac{\rho_1 s_1 + (1-\rho_1)\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]}{1-\rho_1^\tau}$ and $\widehat{s}_2 = \frac{\rho_2 s_2 + (1-\rho_2)[\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]]'\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]}{1-\rho_2^\tau}$, are used in order to update the parameters: $\boldsymbol{\theta}_{\tau+1} - \boldsymbol{\theta}_\tau = -\epsilon\frac{\widehat{s}_1}{\sqrt{\widehat{s}_2+\delta}}$.

An alternative optimization method is exponentially decaying the average of the squared gradient, so that the updating can converge even faster. For example, RSMProp uses an exponentially decaying average with decay rate $\rho \in [0,1)$ that discards history from the extreme past and employs the squared gradient, initializing at the origin, $s = 0$. Subsequently, the update of s given by $\widehat{s} = \rho s + (1-\rho)\left[\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]\right]'\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]$ is used in order to update the parameters: $\boldsymbol{\theta}_{\tau+1} - \boldsymbol{\theta}_\tau = -\frac{\epsilon}{\sqrt{\widehat{s}+\delta}}\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}_\tau]$. Back-propagation is the method for computing the gradient of the cost function in (10), $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) = \frac{1}{N^\llcorner}\sum_{i=1}^{N^\llcorner}\nabla_{\boldsymbol{\theta}}L[y_i, k(\mathbf{X}_i|\boldsymbol{\theta})] + \lambda\nabla_{\boldsymbol{\theta}}\omega[\boldsymbol{\theta}]$, which itself is a function of the gradients of the loss function and penalty terms. Those gradients are computed 'backwards', as dictated by the 'chain rule of calculus', since they are compositions of functions of the parameters $\boldsymbol{\theta}$. Once those gradients are computed, SGD or other optimization algorithms are used to perform the learning/approximation exploiting them.

Finally 'bagging' ('bootstrap aggregating') is also a powerful regularization method that can combine parametric and non-parametric penalties. It involves creating B different datasets from the training sample $N^\llcorner$ by sampling with replacement $N^B = N^\llcorner$ observations, and solving (7) on each of the B different training datasets, $\widehat{f}_B(\mathbf{X};\lambda)$. The out-of-sample performance of the B-ensemble predictor is then $\frac{1}{N^\top}\sum_{i=1}^{N^\top}\frac{1}{B}L[y_i, \widehat{f}_B(\mathbf{X}_i;\lambda)]$. Because sampling is done with replacement, each dataset b for $b = 1, \cdots, B$ is missing some of the observations from the original dataset $N^\llcorner$ with high probability, which reults in different approximations $\widehat{f}_b(\mathbf{X};\lambda)$ which make different errors in the test sample $N^\top$. Those errors will tend to cancel out if sampling is random, improving the out-of-sample performance of the B-ensemble model relative to its members.

How is λ determined? Because choosing the strength of the penalty λ determines the solution approximation $\widehat{f}(\mathbf{X};\lambda)$ to (7)—and hence (10)—this is referred to as 'model selection'. Ideally, one would like to choose the λ that maximizes the out-of-sample performance of $\widehat{f}(\mathbf{X};\lambda)$:

$$\widehat{\lambda} \in \arg\min_{\lambda}\frac{1}{N^\top}\sum_{i=1}^{N^\top}L[y_i, \widehat{f}(\mathbf{X}_i;\lambda)]. \tag{11}$$

However, different 'splittings' of the available sample into complementary learning and test subsamples, $N = N^\llcorner + N^\top$, are going to provide different values of $\widehat{\lambda}$. To avoid the computational burden that are associated with computing $\widehat{\lambda}$ for all possible assignments $\binom{N}{N^\llcorner}$ and then minimizing the average over these replications, this process is instead approximated by dividing the available sample of size N into K disjoint subsamples of approximately equal size, N/K. Each of the subsamples denoted as $N^{\llcorner k}$, for $k = 1, \ldots, K$ is used as 'test sample' in (11), such that the complement sample $N - N^{\llcorner k}$ is used as training sample in (7) to fit the model. By doing so, we obtain K different approximations $\widehat{f}_K(\mathbf{X};\lambda)$, each of which is evaluated once on the test sample $N^{\llcorner k}$. Averaging the results over K in (11), we obtain $\frac{1}{K}\{\frac{1}{N^K}\sum_{i=1}^{N^K}L[y_i, \widehat{f}_K(\mathbf{X}_i;\lambda)]\}$, and solving for $\widehat{\lambda}$ returns $\widehat{\lambda}_K$, as determined by 'K-fold' cross validation.

3. Neural Networks for Prediction

This section analyzes artificial neural networks. This is, arguably, the most powerful modeling device in machine learning and the preferred approach for complex machine learning problems, such as computer vision, natural language processing, pattern recognition, biomedical diagnosis, and others (see Schmidhuber (2015) and LeCun et al. (2015) for overviews of the topic). Artificial neural networks are divided into shallow and deep networks, depending on the number of hidden layers used to predict the output. The flexibility of neural networks with several layers draws from their ability in order to incorporate nonlinear interactions between the predictors, being denominated deep

neural networks or, more generally, deep learning methods. The complexity of these methods entails, by construction, a lack of interpretation and transparency for disentangling the relationship between the predictors and the output.

Our analysis focuses on traditional feedforward networks. These consist of an input layer of predictor variables, one or more hidden layers that interact and nonlinearly transform the predictors, and an output layer that aggregates hidden layers into an ultimate outcome prediction. Deep learning builds on feedforward neural networks (NN) or multi-layer perceptrons (MLPs) in order to learn unknown target functions of increasing complexity. MLPs are then compositions of single-layer/shallow NNs, each hidden unit of which (or 'neuron') is fully connected to the hidden units of the subsequent layer, to capture the fact that information flows forward from the inputs $\mathbf{X}$ to the output y. Thus, artificial neural networks, or MLPs, are similar to biological neural networks: they are collections of connected units called neurons. An artificial neuron receives inputs from other neurons, computes the weighted sum of the inputs, and maps the sum via an activation function to the neurons in the next layer, and so on until it reaches the last layer or output. Accordingly, the network is free of cycles or feedback connections that pass information backward.[8]

Single-layer/shallow NNs are universal approximators (Hornik 1991; Cybenko 1989) and they have dictionaries of functions of the form $\{b(\mathbf{X}|\gamma_1) = s(\mathbf{W}_1'\mathbf{X} + \mathbf{b}_1) : \gamma_1 = (\mathbf{b}_1, \mathbf{W}_1), \mathbf{W}_1'\mathbf{X} = [...\sum_{p=1}^{P} w_{zp}x_p...]' \in \mathbb{R}^{Z_1}\}$ where $s(\cdot) : \mathbb{R}^{Z_1} \to \mathbb{R}^{Z_1}$ is a vector-valued 'activation function' (i.e., applied unit-wise), mapping the output from the single hidden layer $\mathbf{h}_1 = \mathbf{W}_1'\mathbf{X} + \mathbf{b}_1 \in \mathbb{R}^{Z_1}$ and the bias of each hidden unit $z \in \mathbb{R}^{Z_1}$ in the single hidden layer, $\mathbf{b}_1 \in \mathbb{R}^{Z_1}$, into the output, $\widehat{y} = \sum_{z=1}^{Z_1} w_{2z}s_z(\mathbf{W}_1'\mathbf{X} + \mathbf{b}_1) + b_{2z} \equiv \widehat{f}(\mathbf{X}; \boldsymbol{\theta}_1)$, with the weights $\mathbf{w}_2 \in \mathbb{R}^{Z_1}$ and bias $b_2 \in \mathbb{R}$ being the parameters $\{a_z\}_{z=1}^{Z_1}$ of the function class $\mathcal{G}$ that is defined above, i.e., $\boldsymbol{\theta}_1 = (\mathbf{w}_2, b_2; \mathbf{b}_1, \mathbf{W}_1) \equiv (\mathbf{a}; \gamma_1)$. Popular choices for the activation function include: (i) Rectified linear units (ReLu), $s(h) = \max\{0, h\}$; (ii) Softplus, $s(h) = \log(1 + e^h)$; (iii) Hard tanh, $s(h) = \max\{-1, \min\{1, h\}\}$; (iv) Sigmoid or 'logistic', $s(h) = (1 + e^{-h})^{-1}$; or, (v) Maxout, $s(h) = \max_{j \in \mathbb{G}^i} h_j$, where the number of hidden units z in layer l, Z_l, is divided into groups of k values, $\{(z_1, ..., z_k), ..., (z_{Z_l-k+1}, ..., z_{Z_l})\}$, and $\mathbb{G}^i = \{(i-1)k + 1, ..., ik\}$ is the set of indices into the inputs for group i. All of the activation functions $s(\cdot)$ have in common that a certain threshold must be overcome for information to be passed forward, much alike neurons in the human brain, which need to receive a certain amount of stimuli in order to be activated. The threshold hurdle creates a nonlinearity that allows for artificial NNs to learn nonlinear and non-convex unknown target functions $f(\mathbf{X})$.

Single-layer NNs are also known as 'three-layer' networks, where the inputs $\mathbf{X}$ form the first layer. The second or 'hidden' layer $\mathbf{h}_1$ is comprised of $(\mathbf{b}_1, \mathbf{W}_1, s(\cdot)) : \mathbf{h}_1 = s(\mathbf{W}_1'\mathbf{X} + \mathbf{b}_1)$, and the third corresponds to the output layer, $\widehat{y} = \mathbf{w}_2's(\mathbf{h}_1) + b_2 \in \mathbb{R}$. A deep NN (DNN) is constructed by adding hidden layers, with each subsequent one taking as inputs the output of the previous ones. For example, a 'four-layer' NN that adds one hidden layer to a 'three-layer' NN (or shallow/single-layer NN), rather than simply taking the linear combination of the dictionary entries of single-layered NNs, $\{b(\mathbf{X}|\gamma_1)\}$, would result in the collection of functions that are represented by the dictionary $\{b(\mathbf{X}|\gamma_2) = s(\mathbf{W}_2's(\mathbf{W}_1'\mathbf{X} + \mathbf{b}_1) + \mathbf{b}_2) : \gamma_2 = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{W}_1, \mathbf{W}_2), \mathbf{W}_1'\mathbf{X} = [...\sum_{p=1}^{P} w_{zp}x_p...]' \in \mathbb{R}^{Z_1}, \mathbf{W}_2 \in \mathbb{R}^{Z_1 \times Z_2}\}$. Adding hidden layers then results in parameter addition, increasing the variance, and reducing the bias. The overall effect on performance (i.e., on generalization/test error) will depend on how well the resulting dictionary matches the unknown target function $f(\mathbf{X})$. Additionally, although it is an open question in the deep learning literature, why do over-parameterized DNNs perform well in terms of generalization/test error, original contributions due to Pascanu et al. (2013), and Montufar et

[8] MLPs that allow information to flow backwards are called recurrent neural networks and they are discussed in Goodfellow et al. (2016) .

al. (2014) show that deeper ReLu architectures have more flexibility to express the behavior of the unknown target function, relative to equally sized single-layer/shallow architectures.

An incipient strand of the literature (e.g., Arora et al. 2019; Allen-Zhu et al. 2019) building on the Rademacher complexity of both the function class being approximated and of the dataset shows that the dictionaries of deeper architectures can better capture interactions between the units of different layers through the composition of functions that they can represent.

Generally, a DNN approximation $\widehat{f}(\cdot) : \mathbb{R}^P \to \mathbb{R}$ of size $Z = \sum_{l=1}^{L} Z_l$ with $L \in \mathbb{N}$ hidden layers and $Z_l \in \mathbb{N}$ nodes per layer l, is of the form:

$$\begin{aligned} \widehat{f}(\mathbf{X}) &\equiv f(\mathbf{X}; \mathbf{\Lambda}_L) = \mathbf{w}'_{L+1} s(\mathbf{W}'_L \mathbf{h}_{L-1} + \mathbf{b}_L) + b_{L+1} \\ &= \underset{L-\text{composition}}{f \circ f \circ ... \circ f}(\mathbf{X}; \mathbf{\Lambda}_1) \end{aligned}$$

where $s(\cdot) : \mathbb{R}^{Z_{L-1}} \to \mathbb{R}^{Z_L}$ is the vector-valued activation function that maps the output from the previous hidden layer $\mathbf{h}_{L-1} = s(\mathbf{W}'_{L-1}\mathbf{h}_{L-2} + \mathbf{b}_{L-1}) \in \mathbb{R}^{Z_{L-1}}$ and the bias of each hidden unit $z \in \mathbb{R}^{Z_L}$ in the last hidden layer L, $\mathbf{b}_L \in \mathbb{R}^{Z_L}$, into the output layer $l = L+1$, with weights $\mathbf{w}_{L+1} \in \mathbb{R}^{Z_L}$ and bias unit $b_{L+1} \in \mathbb{R}$. The matrices $\mathbf{W}_l = [\mathbf{w}_1 ... \mathbf{w}_{Z_l}] \in \mathbb{R}^{Z_{l-1} \times Z_l}$ contain the weights $\mathbf{w}_z \in \mathbb{R}^{Z_{l-1}}$ of each hidden unit $z = 1...Z_l$ for each hidden layer $l = 1...L$, with $Z_0 = P$ the dimension of the input vector $\mathbf{X} \in \mathbb{R}^P$; $\mathbf{\Lambda}_L \equiv [\boldsymbol{\theta}_L; Z, L, \{Z_l\}_{l=1}^L; \epsilon, \lambda, \alpha]$ is the collection of parameters $\boldsymbol{\theta}_L = [(\mathbf{w}_{L+1}, b_{L+1}) ... (\mathbf{W}_1, \mathbf{b}_1)]$ and hyperparameters $[Z, L, \{Z_l\}_{l=1}^L]$ and $[\epsilon, \lambda, \alpha]$ to be learned and/or 'fined tuned' by the optimization algorithm

Approximating the unknown target function $f(\mathbf{X})$ with a DNN is then equivalent to parameter estimation:

$$\widehat{\mathbf{\Lambda}}_L \in \arg\min_{\mathbf{\Lambda}_L} \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} L[y_i, f(\mathbf{X}_i; \mathbf{\Lambda}_L)] + \lambda \varpi[\boldsymbol{\theta}] \tag{12}$$

where it is standard practice to 'cross-validate' the choice of hyperparameters $[Z, L, \{Z_l\}_{l=1}^L]$ and $[\epsilon, \lambda, \alpha]$ before estimating the parameters that characterize the restricted class of functions/models that are represented by the dictionary $\{b(\mathbf{X}|\gamma_L) : \gamma_L = (\mathbf{b}_1, ..., \mathbf{b}_L, \mathbf{W}_1, ..., \mathbf{W}_L)\}$ augmented by the output layer weights and bias, $(\mathbf{w}_{L+1}, b_{L+1})$, $\boldsymbol{\theta}_L = [(\mathbf{w}_{L+1}, b_{L+1}) ... (\mathbf{W}_1, \mathbf{b}_1)]$, that solve the 'empirical risk minimization' problem (12). In deep learning, standard choices are: (i) a cross-entropy cost/loss function, $L[\cdot, \cdot]$; (ii) a ReLu activation function $s(\cdot)$, which naturally leads to sparse settings, whereby a large portion of hidden units are not activated, thus having zero output (LeCun et al. 2015); (iii) a SGD penalty $\varpi[\boldsymbol{\theta}]$, usually combined with momentum α, as optimization method; and, (iv) network architecture size, depth, and nodes per layer, $[Z, L, \{Z_l\}_{l=1}^L]$, as well as learning rate, ϵ, that depend on the characteristics of the dataset, $\{y_i, \mathbf{X}_i\}_{i=1}^N$. Performance is then assessed on the test sample, from evaluating $\frac{1}{N^{\top}} \sum_{i=1}^{N^{\top}} L[y_i, f(\mathbf{X}_i; \widehat{\mathbf{\Lambda}}_L)]$.

In practice, 'tuning' or optimizing the hyperparameters is a daunting task in terms of processing time and computational capacity, e.g., only determining the optimal depth (number of layers L) and nodes per layer ($\{Z_l\}_{l=1}^L$) for architectures of a given size Z involves solving an NP-hard combinatorial optimization problem because $L, \{Z_l\}_l \in \mathbb{N}$, i.e., are integer values (Judd 1990). Yet, in Calvo-Pardo et al (2020), we show that recent advances in combinatorial optimization software (RStudio) can be exploited to optimally allocate hidden units ($\{Z_l\}_{l=1}^L$) within ('width') and across ('depth', L) layers in deep architectures of a given size $Z = \sum_{l=1}^{L} Z_l$. Adopting the lower bound on the maximal number of linear regions that a ReLu DNN can approximate as the maximization criterion, see Montufar et al. 's (2014), we obtain

$$LB(L, \{Z_l\}_{l=1}^{L-1}; P) \equiv \left(\prod_{l=1}^{L-1} \left\lfloor \frac{Z_l}{P} \right\rfloor^P \right) \sum_{r=0}^{P} \binom{Z - \sum_{l=1}^{L-1} Z_l}{r}.$$

Similarly, upper bounds, or maximal number of linear regions of a function approximated by a network architecture with rectified linear units of size Z, have been recently characterized by Raghu et al. 's (2017) Theorem 1 to equal

$$UB(L, \{Z_l\}_{l=1}^{L}; P) = O\left(\left[\frac{Z}{L}\right]^{ZP}\right)$$

from which they conclude that the maximal number of regions approximated by a shallow ReLu NN, $UB(1, Z; P)$, is always smaller than the maximal number of regions approximated by an equally-sized deep ReLu NN, $UB(L, \{Z_l\}_{l=1}^{L}; P) : \sum_{l=1}^{L} Z_l = Z$:

$$UB(1, Z; P) < UB(2, \frac{Z}{2}; P) < ... < UB(L, \frac{Z}{L}; P)$$

We effectively solve (12) in two-stages:

$$(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}}) \in \arg \max_{(L, \{Z_l\}_{l=1}^{L-1})} LB(L, \{Z_l\}_{l=1}^{L-1}; P) \tag{13}$$

$$\widehat{\mathbf{\Lambda}}_L(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}}) \in \arg \min_{\mathbf{\Lambda}_L(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}})} \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} L[y_i, f(\mathbf{X}_i; \mathbf{\Lambda}_L)] + \lambda \varpi[\boldsymbol{\theta}] \tag{14}$$

The first stage optimization (13) solves for the optimal depth $\widehat{L}$ and the number of hidden units per layer (or optimal width, layer-wise) $\{\widehat{Z}_l\}_{l=1}^{\widehat{L}}$ given the network architecture size, $Z = \sum_{l=1}^{L} Z_l$.[9] The outcome of the first stage is an optimal deep network architecture in the sense of maximizing the expressive power of the approximation $f(\mathbf{X}; \mathbf{\Lambda}_L)$ within the restricted class of functions that are generated by the dictionary $\{b(\mathbf{X}|\gamma_L) : \gamma_L = (\mathbf{b}_1, ..., \mathbf{b}_L, \mathbf{W}_1, ..., \mathbf{W}_L)\}$. The second stage optimization (14) proceeds, just as in (12), but takes as given the optimal values of the hyperparameters $(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{L})$ from the first stage (13), i.e., $\mathbf{\Lambda}_L(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}}) = [\boldsymbol{\theta}_L; Z, (\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}}); \epsilon, \lambda, \alpha]$. Rather than engaging into time and computer intensive 'fine tuning' of the whole set of hyperparameters $[Z, L, \{Z_l\}_{l=1}^{L}; \epsilon, \lambda, \alpha]$ while training the deep architecture to estimate/learn $\boldsymbol{\theta}_L$, as in (12), proceeding in two-stages considerably saves on runtime and memory while improving performance, as we show in the next section. Finally, notice that being the first stage conditional on the architecture size, bigger and more complex datasets $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$ will naturally summon architectures with more hidden units, Z.

Deep neural networks have become so powerful, because of (i) the availability of large datasets, necessary to 'train' them, and because of the rapid improvements in (ii) computational power[10] and in (iii) optimization algorithms and software. Deep neural networks are characterized by a large number of parameters that need to be 'optimized' during 'training'. This is called 'fine-tuning' or 'optimally fitting a neural network' to the 'training sample'. The backpropagation optimization algorithm informs the machine of how it should change the internal parameters used to compute the representation in each layer from the representation in the previous layer. Software optimization methods (e.g, Adam,

---

[9] The first stage optimization (13) is a constrained combinatorial optimization problem:

$$(\widehat{L}, \{\widehat{Z}_l\}_{l=1}^{\widehat{L}}, \{\mu_l\}_{l=1}^{\widehat{L}}) \in \arg \max_{(L, \{Z_l\}_{l=1}^{L-1}, \{\mu_l\}_{l=1}^{L})} LB(L, \{Z_l\}_{l=1}^{L-1}; P) + \sum_{l=1}^{L-1} \mu_l(P - Z_l) + \mu_L(-L)$$

where $\{\mu_l\}_{l=1}^{\widehat{L}} \in \mathbb{R}^L$ is the collection of L Lagrange multipliers that are associated with the $L-1$ constraints, $Z_l \geq P, l = 1...L-1$, and with the constraint $L > 0$, because the constraint on the architecture size $Z = \sum_{l=1}^{L} Z_l$ is incorporated into the maximand. Since $L, \{Z_l\}_{l=1}^{L} \in \mathbb{N}$, we also solve in two stages to reduce the computational burden. In the first stage of (13), the number of hidden units are optimally allocated for a given depth, $\left\{L, \{\widehat{Z}_l\}_{l=1}^{L}\right\}$, while, in the second stage of (13), the optimal depth is sought after for a given allocation of hidden units, $\left\{\widehat{L}, \{Z_l\}_{l=1}^{\widehat{L}}\right\}$.

[10] Particularly, of graphics processing units (GPUs), suited to perform the linear algebra operations at the root of 'fitting' neural networks, e.g. Google DeepMind optimized a deep neural network while using 176 GPUs for 40 days to beat the best human players in the game Go.

Adagrad, RMSprop) that implement SGD or any of its variants, allow for substantial gains in the necessary time and computational power when training models with millions of parameters, and it is nowadays often paired with step size 'adaptive regularization'. It is also now standard practice to do regularization while optimizing (e.g., via 'weight decay', 'dropout', or 'batch normalization') to prevent overfitting and improve the performance of DNNs 'out-of-sample'.

'Batch normalization' (Ioffe and Szegedy 2015; not to be mistaken with 'minibatch regularization') is a method of adaptive reparameterization that is best suited for training very deep models that involve the composition of several functions or layers. By normalizing the output of each layer before forwarding it as input to the next layer, the unexpected effect of many functions being composed together changing simultaneously is removed, allowing for the gradient to update the parameters under the assumption that the other layers do not change. As a result, it allows the use of higher learning rates, ϵ, which are less sensible to the initialization of parameters. Concretely, the normalization involves computing:

$$\overline{h}_{zl} = \frac{1}{\sigma}(h_{zl} - \mu), z \in B : \mu = \frac{1}{|B|}\sum_{z\in B} h_{zl}, \sigma = \sqrt{\delta + \frac{1}{|B|}\sum_{z\in B}(h_{zl} - \mu)^2}$$

with $\delta \approx 10^{-8}$ being set to prevent the undefined value $\sqrt{0}$, and B denoting a minibatch of output units h_{zl} in layer $l = 1...L$.

Another recent methodology introducing randomness into deep neural networks is 'Dropout'. This method discards a small, but random, portion of the neurons during each iteration of training to prevent neurons from co-adapting, providing a powerful regularization method (Srivastava et al. 2014). The intuition is that, since several neurons are likely to model the same nonlinear relationship simultaneously, discarding a random fraction of them forces them to perform well, regardless of which other hidden units are in the model.

With dropout, each input and hidden unit z in layer $l = 1...L$, h_{zl}, is pre-multiplied by a random variable $r_{zl} \sim F(r_{zl})$, $\overline{h}_{zl} = r_{zl} \cdot h_{zl}, \forall(z, l)$, prior to being fed forward to the activation function of the next layer, $h_{zl+1} = s_z(\sum_{z=1}^{Z_l} w_{zl+1}\overline{h}_{zl} + b_{zl+1}), \forall z = 1...Z_{l+1}$. For any layer, l, $\mathbf{r}_l$ is then a vector of independent random variables, $\mathbf{r}_l = [r_{1l}, ..., r_{Z_l l}] \in \mathbb{R}^{Z_l}$. Standard choices for the probability distribution $F(\mathbf{r}_l)$ are (i) the Normal, i.e., $F(\mathbf{r}_l) = N(\mathbf{1}, \mathbf{I})$, or (ii) the Bernoulli, in which case each r_{zl} has probability p of being 1 (and $1 - p$ of being 0). The vector $\mathbf{r}_l$ is then sampled and multiplied element-wise with the outputs of that layer, h_{zl}, in order to create the thinned outputs, $\overline{h}_{zl}$, which are then used as input to the next layer, h_{zl+1}. When this process is applied at each layer $l = 1...L$, this amounts to sampling a sub-network from a larger network. In the ML literature, common choices for p are 0.8 for the input layer, $l = 1$, and 0.5 for the units in hidden layers, in $l = 2...L$.

During learning, the derivatives of the loss function are backpropagated through the sub-network. At test time, the weights are scaled down as $\overline{\mathbf{W}}_l = p\mathbf{W}_l, l = 1...L$, resulting in a DNN (without dropout) that allows for the conduct of approximate inference. It is actually exact for many classes of models that do not have nonlinear hidden units, like the softmax regression classifier, regression networks with conditionally normal outputs, or deep networks with hidden layers without nonlinearities. This efficient test time procedure is an approximate model combination that (i) scales down the weights of the trained neural network, (ii) works well with other distributed representation models, e.g. restricted Boltzmann machines, and (iii) acts as a regularizer. Beyond the MLPs discussed, an array of alternative architectures have been proposed, including convolutional and recurrent NNs, which target specific data structures, like vision tasks and sequential data handling, respectively. See Goodfellow et al. (2016) for a detailed textbook treatment.

4. Uncertainty and Deep Learning

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 for estimating 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. The predictions of deep neural network models need to be supported by measures of uncertainty that shed light on the reliability of the predictions. Recent literature in machine learning has focused on the construction of algorithms in order to measure the uncertainty around the predictions of neural network methods. The first subsection reviews methods for assessing the uncertainty regarding the model predictions.

4.1. Uncertainty in Model Prediction

Despite their unrivaled success in prediction and forecasting tasks, deep learning models struggle in conveying the uncertainty or degree of statistical confidence/reliability associated with those forecasts. Some recent contributions in the ML literature have made progress in the provision of prediction intervals for the point forecasts that are provided by deep learning models trained with dropout. For example, Gal and Ghahramani (2016) show that a NN with arbitrary depth and nonlinearities, with dropout being applied before every hidden layer and a parametric L^2 penalty $\varpi[\boldsymbol{\theta}] = \sum_{l=1}^{L}\left\{\|\mathbf{W}_l\|_2^2 + \|\mathbf{b}_l\|_2^2\right\}$, minimizes the Kullback–Leibler divergence between an approximate (variational) distribution, $q(\boldsymbol{\theta})$—over matrices $\boldsymbol{\theta} = (\mathbf{W}_1, ..., \mathbf{W}_L)$ with columns randomly set to zero, $\mathbf{W}_l = \mathbf{M}_l diag[r_{zl}]_{z=1}^{Z_l}, r_{zl} \sim Bernoulli(p_l), l = 1, ..., L, z = 1, ..., Z_l$—and the posterior of a deep Gaussian process, $p(\boldsymbol{\theta}|y; \mathbf{X})$,, which is intractable:

$$\begin{aligned} & -\sum_{i=1}^{N}\int q(\boldsymbol{\theta})\log p(y_i|\mathbf{X}_i;\boldsymbol{\theta})d\boldsymbol{\theta} + D_{KL}(q(\boldsymbol{\theta})||p(\boldsymbol{\theta})) \\ \propto \; & -\sum_{i=1}^{N}\frac{\log p(y_i|\mathbf{X}_i;\widehat{\boldsymbol{\theta}})}{\tau N} + \sum_{l=1}^{L}\left\{\frac{p_l l^2}{2\tau N}\|\mathbf{M}_l\|_2^2 + \frac{l^2}{2\tau N}\|\mathbf{b}_l\|_2^2\right\} \end{aligned}$$

where the first and second terms in the sum are approximated. In the first term, each term in the sum over N is approximated by Monte Carlo integration with a single sample $\widehat{\boldsymbol{\theta}}^b \sim q(\boldsymbol{\theta})$ to obtain an unbiased estimate of $\log p(y_i|\mathbf{X}_i;\widehat{\boldsymbol{\theta}})$. In the second, l denotes prior length-scale, and τ model precision, i.e., $p(y|\mathbf{X};\boldsymbol{\theta}) = N(\widehat{y}(\mathbf{X};\boldsymbol{\theta}), \frac{1}{\tau}\mathbf{I}) : \widehat{y}(\mathbf{X};\boldsymbol{\theta}) = \sqrt[-2]{Z_L}\mathbf{W}_L s(... \sqrt[-2]{Z_1}\mathbf{W}_2 s(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1)...)$ and variance-covariance matrix $\frac{1}{\tau}\mathbf{I}$. The sampled $\widehat{\boldsymbol{\theta}}^b$ result in realizations from the Bernoulli distribution $[\mathbf{r}_l^b]$ equivalent to the binary variables in the dropout case, i.e., sampling B sets of vectors of realizations from the Bernoulli distribution $\{[\mathbf{r}_l^b]\}_{b=1}^{B}$ with $[\mathbf{r}_l^b] = [r_{zl}^b]_{z=1}^{Z_l}$, giving $\{\mathbf{W}_1^b, ..., \mathbf{W}_L^b\}_{b=1}^{B}$, with which the first two moments of the predictive distribution $p(y_i|\mathbf{X}_i;\widehat{\boldsymbol{\theta}})$ are estimated (by moment-matching). The first moment, $\frac{1}{B}\sum_{b=1}^{B}\widehat{y}(\mathbf{X};\mathbf{W}_1^b, ..., \mathbf{W}_L^b)$, is known as Monte Carlo (MC) dropout and, in practice, it corresponds to performing B stochastic forward passes through the NN and averaging the results (model averaging). The second moment, $\frac{1}{\tau}\mathbf{I} + \frac{1}{B}\sum_{b=1}^{B}\widehat{y}(\mathbf{X};\mathbf{W}_1^b, ..., \mathbf{W}_L^b)'\widehat{y}(\mathbf{X};\mathbf{W}_1^b, ..., \mathbf{W}_L^b)$, equals the sample variance of B stochastic forward passes through the NN plus the inverse model precision, providing a measure of the uncertainty that is attached to the deep NN point forecast.

Under the assumption that the approximation error is negligible, the predictive variance can be estimated as

$$\widehat{\sigma}_{MC}^2 = \widehat{\sigma}_e^2 + \frac{1}{B}\sum_{b=1}^{B}\widehat{y}(\mathbf{X};\mathbf{W}_1^b, ..., \mathbf{W}_L^b)'\widehat{y}(\mathbf{X};\mathbf{W}_1^b, ..., \mathbf{W}_L^b), \tag{15}$$

with $\widehat{\sigma}_e^2 = \frac{1}{N^\top}\sum_{i=1}^{N^\top}\left(y_i - \bar{f}_{MC}(\mathbf{X}_i)\right)^2$ a consistent estimator of σ_e^2 under homoscedasticity of the error term, also see Smyl (2020) and Kendall and Gal (2017). A suitable prediction interval for y_i under the assumption that $p(\widehat{y}\,|\mathbf{X},\boldsymbol{\theta})$ is normally distributed is

$$\bar{f}_{MC}(\mathbf{X}_i) \pm z_{1-\alpha/2}\widehat{\sigma}_{MC}. \tag{16}$$

An alternative approach to MC dropout for estimating the uncertainty about the predictions is to use bootstrap methods, see Tibshirani (1996). Bootstrap procedures provide a reliable solution in order to obtain predictive intervals of the output variable. We proceed to explain how bootstrap works in a DNN context. Let $\{\mathbf{X}_i\}_{i=1}^{N^\llcorner}$ be a sample of $N^\llcorner$ observations of the set of covariates, with

$\mathbf{X}_i \in \mathbb{R}^{N^{\llcorner} \times P}$. Let $\{\mathbf{y}\}_{i=1}^{N^{\llcorner}} \in \mathbb{R}$ be the output variable and define $\mathbf{X}_i^{\neg} = (\mathbf{X}_i, y_i) \in \mathbb{R}^{N^{\llcorner} \times (P+1)}$. Applying the naive bootstrap that was proposed by Efron (1979) to this multivariate dataset, we generate the bootstrapped dataset $\mathbf{X}^{\neg,\star} = \{\mathbf{X}_i^{\neg,\star}\}_{i=1}^{N^{\llcorner}} = \{y_i^{\star}, \mathbf{X}_i^{\star}\}_{i=1}^{N^{\llcorner}}$ by sampling with replacement from the original dataset $\mathbf{X}^{\neg}$. By repeating this procedure B times, it is possible to obtain B bootstrapped samples defined as $\{\mathbf{X}^{\neg,\star(b)}\}_{b=1}^{B}$. Each bootstrap sample is fitted to a single neural network in order to obtain an empirical distribution of bootstrap predictions $f(\mathbf{X}^{\star(b)}; \widehat{\boldsymbol{\theta}}^{\star(b)})$; with $\widehat{\boldsymbol{\theta}}^{\star(b)}$ the set of bootstrap parameters for $b = 1, \ldots, B$. In this context, a suitable bootstrap prediction interval for y_i at an α significance level is $[\widehat{q}_{\alpha/2}, \widehat{q}_{1-\alpha/2}]$, with $\widehat{q}_{\alpha}$ the empirical $\alpha-$quantile obtained from the bootstrap distribution of $f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}}^{\star(b)})$, for $b = 1, \ldots, B$.

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

$$f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}}^{\star(b)}) \pm z_{1-\alpha/2} \widehat{\sigma}_{\epsilon}^{\star}, \tag{17}$$

with $f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}}^{\star(b)})$ the pointwise prediction of the model and $z_{1-\alpha/2}$ the critical value of a $N(0,1)$ distribution at an α significance level; $\widehat{\sigma}_{\epsilon}^{\star 2} = \widehat{\sigma}_{\widehat{\boldsymbol{\theta}}}^{\star 2}(\mathbf{X}_i) + \widehat{\sigma}_{e}^{2}$. Under homoscedasticity of the error term ϵ_i, the aleatoric uncertainty σ_e^2 is estimated from the test sample as $\widehat{\sigma}_e^2 = \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} \left(y_i - f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}})\right)^2$, with $\widehat{\boldsymbol{\theta}}$ the set of parameter estimates that were obtained from the original sample $\mathbf{X}^{\neg}$. The epistemic uncertainty is estimated from the bootstrap samples as $\widehat{\sigma}_{\widehat{\boldsymbol{\theta}}}^{\star 2}(\mathbf{X}_i) = \frac{1}{B} \sum_{b=1}^{B} [f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}}^{\star(b)}) - \bar{f}(\mathbf{X}_i)]^2$, with

$$\bar{f}(\mathbf{X}_i) = \frac{1}{B} \sum_{b=1}^{B} f(\mathbf{X}_i; \widehat{\boldsymbol{\theta}}^{\star(b)}). \tag{18}$$

This bootstrap prediction interval can be further refined by exploiting the average prediction in (18). In this case, the variance of the predictor is $\overline{\sigma}_{\widehat{\boldsymbol{\theta}}}^{\star 2}(\mathbf{X}_i) = \frac{1}{B} \widehat{\sigma}_{\widehat{\boldsymbol{\theta}}}^{\star 2}(\mathbf{X}_i)$ and the relevant prediction interval is

$$\bar{f}(\mathbf{X}_i) \pm z_{1-\alpha/2} \widehat{\sigma}_{\epsilon}^{\star}, \tag{19}$$

with $\widehat{\sigma}_{\epsilon}^{\star 2} = \overline{\sigma}_{\widehat{\boldsymbol{\theta}}}^{\star 2}(\mathbf{X}_i) + \overline{\sigma}_{e}^{2}$, where $\overline{\sigma}_e^2 = \frac{1}{N^{\llcorner}} \sum_{i=1}^{N^{\llcorner}} \left(y_i - \bar{f}(\mathbf{X}_i)\right)^2$. This expression assumes that the covariance between the predictions from the different bootstrap samples is zero.

4.2. Causal Inference and Interpretability

A recent area of interest in machine learning methods is the development of methods allowing to add interpretability to the outputs of these models. A typical example is to assess the causal relationship between the input and output variables. Recent progress in this direction has been done by Belloni et al (2014); Farrell (2015); Athey and Imbens (2019); and, Farrell et al (2019).

The goal of interpretation tasks is to use the structural form of the approximating function $\widehat{f}(\mathbf{X})$ to try to understand the mechanism that produced the data $\{y_i, \mathbf{X}_i\}_{i=1}^{N}$. Interest lies then in the identification of those input variables that are the most relevant to the variation in the output, the nature of the dependence of the output on the most relevant inputs, or how that dependence changes with changes in the values of other inputs. Conducting valid inference rests on the amount of correct information learned about the system (i.e., minimizing the bias at the expense of increasing the variance), rather than just prediction accuracy (where some bias is optimally traded-off against the resulting reduction in the variance). Although both are often in conflict, which limits the inferential abilities of ML methods, it is not always the case.

Athey and Imbens (2019) note that one way to perform valid (causal) inference would be to adapt the 'out-of-sample' performance objective in ML cost/loss functions to control for confounders or for discovering treatment effect heterogeneity, as is standard in the model-based statistics and econometrics literatures. Allen-Zhu et al. (2019) within the ML literature, and Farrell (2015) within

the econometrics literature, obtain nonasymptotic bounds. Based on Farrell (2015), the latter obtains conditions for valid two-step causal inference after first-step deep learning estimation. A survey regarding the differences between the two literatures and recent progress made along integrating both are provided in Athey and Imbens (2019).

5. Empirical Application

The aim of this section is to illustrate the suitability of machine learning methods for prediction in empirical finance modeling. We follow a similar structure to Gu et al. (2020). These authors perform a comparative study of machine learning methods for the canonical problem of empirical asset pricing: measuring asset risk premiums. Gu et al. (2020) consider neural network models and regression trees with the aim of identifying the best performing methods against more conventional methods that are based on linear regression models and ordinary least squares. In a similar spirit, we conduct a prediction exercise to compare the suitability of feedforward neural networks against conventional time series models for the conditional mean and volatility of asset returns.

We consider the monthly prices of the S&P, the Dow Jones, and the Nasdaq indices, starting from 30-02-1972 until 30-07-2020. The out-of-sample forecasting accuracy is compared against a GARCH(1, 1) benchmark; the comparison is conducted in terms of out-of-sample mean squared prediction error (MSPE) and in terms of optimal portfolio allocation while using out-of-sample Sharpe ratios. In order to obtain the out-of-sample forecasts, a fixed rolling window approach with 50 steps is applied. Thus, the period following 30-06-2016 (included) is used for out-of-sample evaluation.

First, the 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.

Following the recent literature on deep learning and time series forecasting focused 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 others), the present paper couples the MC-dropout approach of Gal and Ghahramani (2016) 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 $\mathbf{X}_t = \mathbf{\Xi}_t + \mathbf{\Psi}_t + \mathbf{T}_t$. Figure 1 reports an additive decomposition of the analyzed time series[11].

Based on the algorithm of Smyl (2020), the present paper will fit and forecast the trend component while using an exponential smoothing model, and the random component using either a DNN or a GARCH(1,1) model [12]. 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 are performed in order to forecast the random component $\mathbf{\Xi}_{t+1}$; 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 first moment estimates (β_1) equal to 0.900, and an exponential decay rate for the second 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 the multivariate time series with relative lagged values (up to $k = 10$). Additionally, in order to ensure the proper training of the network, the input data $\mathbf{\Xi}_{t-k}$ for $k = 1, \cdots, 10$ are normalized in order to guarantee that the regressors have zero mean and unit standard deviation.

---

[11] The seasonal component is not reported, as the magnitude was approximately 0 with the highest value observed $3e^{-04}$.
[12] As robustness exercise, we also consider a GARCH(1,1) fitted on the time series $\mathbf{X}_t$.

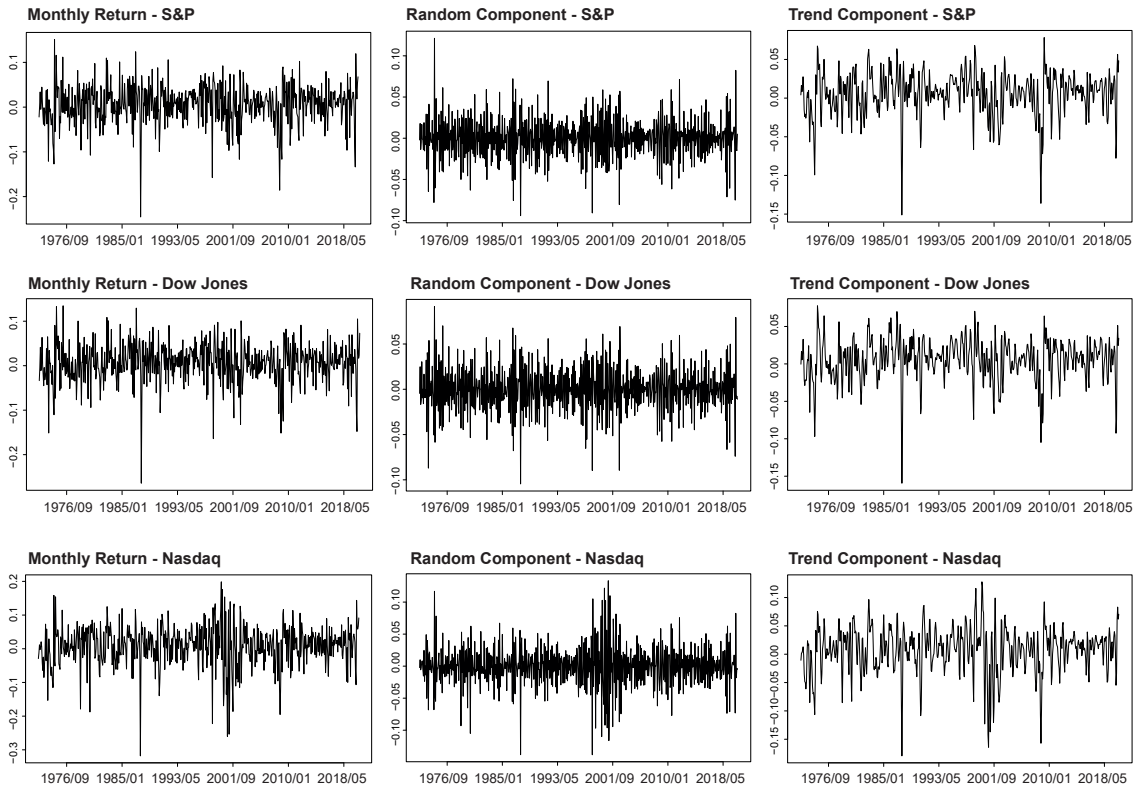

Figure 1. Monthly returns and trend and random component decompositions.

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 while using the one-sided Diebold–Mariano (DM) test (1995), with hypothesis:

$$\mathcal{H}_0 : MSPE^i_{nn} \geq MSPE^{i,j}_{GARCH} \tag{20}$$

and the alternative is

$$\mathcal{H}_1 : MSPE^i_{nn} < MSPE^{i,j}_{GARCH} \tag{21}$$

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 outperformance of the DNN approach against a GARCH(1,1) benchmark.

In order to further validate the out-of-sample performance of the proposed approach, the present paper compares the MC-dropout against a GARCH(1,1) benchmark in terms of portfolio returns for a given optimal strategy. In particular, we will consider a mean-variance portfolio, with the weights defined as:

$$\begin{aligned} \min_{\omega} \quad & \omega' \hat{\Sigma} \omega - \omega' \hat{\mathbf{x}} \\ \text{s.t.} \quad & \omega' \mathbf{1} = 1 \end{aligned} \tag{22}$$

with $\omega \in \mathbb{R}^3$ is the vector of the portfolio weights invested in the three indices considered, $\hat{\mathbf{\Sigma}} \in \mathbb{R}^{3\times 3}$ is the estimated covariance matrix, $\hat{\mathbf{x}} \in \mathbb{R}^3$ is the vector of the expected returns, and $\mathbf{1} \in \mathbb{R}^3$ is a vector of ones. The covariance matrix is defined as:

$$\hat{\mathbf{\Sigma}} = \text{diag}(\hat{\sigma}) \hat{\mathbf{P}} \text{diag}(\hat{\sigma}) \tag{23}$$

with $\text{diag}(\hat{\sigma})$ being the diagonal matrix with estimated standard deviations, and $\hat{\mathbf{P}}$ the correlation matrix[13]. The present paper considers two portfolio strategies: the mean-variance and minimum-variance portfolios (the latter obtains by imposing $\hat{\mathbf{x}} = 0$ in the constrained minimization in (22)). Knowing that holding the portfolio $\omega_t^{\text{strategy}}$ 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 in order 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 while using either a GARCH(1, 1) or a DNN model. We construct the dynamic covariance matrix $\hat{\mathbf{\Sigma}}_{t+1}$ from estimates of the conditional variances and covariances over rolling windows. Based on the forecasted $\hat{\mathbf{X}}_{t+1}$ and $\hat{\mathbf{\Sigma}}_{t+1}$, the constrained minimization in (22) is solved and weights $\omega_t^{\text{strategy}}$ 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 $\omega_t^{\text{strategy}}$: $\mathrm{Y}_{t+1} = \omega_t^{\text{strategy}\,\prime} \mathbf{x}_{t+1}$.

By implementing a fixed rolling window forecasting exercise, the above procedure is repeated 50 times to obtain 50 out-of-sample Y_{t+1} from either the GARCH(1, 1) or the DNN model. This allows for us to estimate the out-of-sample Sharpe ratios as:

$$\text{Sharpe ratio}_i = \frac{\hat{\mathrm{Y}}_p - \mathrm{Y}_{rf}}{\hat{\sigma}_p} \tag{24}$$

with $\hat{\mathrm{Y}}_p$ being the mean return of the portfolio, Y_{rf} the risk-free rate (assumed equal to 0), and $\hat{\sigma}_p$ the portfolio standard deviation.

Figure 2 reports the cumulative returns of the four different strategies considered. One could notice how a portfolio strategy (either mean-variance or minimum variance) that is 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 that are obtained from a GARCH(1,1) forecasts are 0.2686 for the mean-variance and 0.3175 for the minimum-variance portfolio.

The above results extend some of the empirical findings in Gu et al. (2020). These authors 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 that are 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

[13] The constrained minimization in 22 allows for short selling but not for leverage effect.

present paper—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.

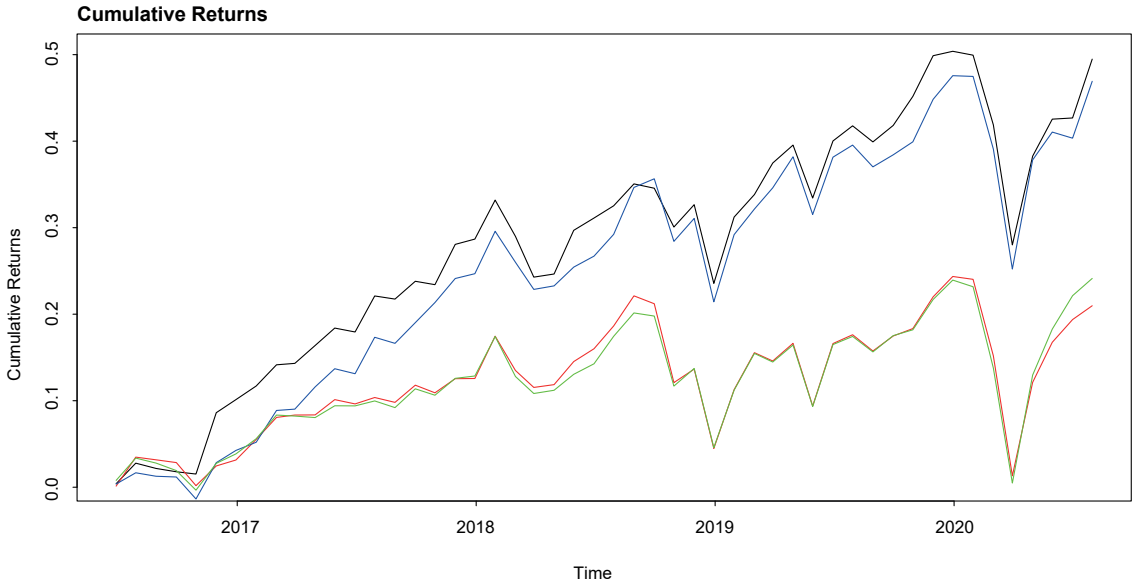

Figure 2. Out-of-sample cumulative returns of the four portfolio strategies analyzed: in black the minimum-variance portfolio from the deep neural network (DNN), in green the minimum-variance portfolio obtained from GARCH forecasts, in blue the mean-variance portfolio obtained from a DNN, in red the mean-variance constructed from GARCH forecasts.

6. Conclusions

We frame our paper in a recent literature on machine learning for empirical finance such as Chinco et al (2019) and Gu et al. (2020). In contrast to these studies, we present an overview of the procedures that are involved in prediction with machine learning models and pay special emphasis on deep learning. We study suitable loss functions for classification and prediction, regularization methods, learning algorithms for model selection, and optimal architectures of deep neural networks. The paper also analyzes modern methods for constructing prediction intervals in deep neural networks and providing a gentle introduction to causal inference.

Empirically, we illustrate the relevance of machine learning methods for financial forecasting and portfolio allocation and assess its performance as compared to traditional time series models while using statistical and economic performance measures. In line with the empirical findings of Gu et al. (2020), we find overwhelming evidence in favor of machine learning techniques, in particular, deep learning methods.

Author Contributions: The authors have contributed equally in both theory and empirical sections of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

Allen-Zhu, Zeyuan, Yuanzhi Li, and Yingyu Liang. 2019. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems*. pp. 6158–69. Vancouver, Canada.

Arora, Sanjeev, Rong Ge, Behnam Neyshabur, and Yi Zhang. 2019. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint*. arXiv:1802.05296.

Athey, Susan, and Guido W. Imbens. 2019. Machine learning methods that economists should know about. *Annual Review of Economics* 11.

Belloni, Alexandre, Victor Chernozhukov, and Christian Hansen. 2014. High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives* 28: 29–50.

Calvo-Pardo, Hector F., Tullio Mancini and Jose Olmo. 2020. Optimal Deep Neural Networks by Maximization of the Approximation Power. Available online: https://ssrn.com/abstract=3578850 (accessed on 10/09/2020)

Chinco, Alex, Adam D. Clark-Joseph, and Mao Ye. 2019. Sparse signals in the crossâ-section of returns. *The Journal of Finance* 74: 449–92.

Cybenko, George. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems* 2: 303–14.

Diebold, Francis X., and Robert S. Mariano. 1995. Comparing Predictive Accuracy. *Journal of Business & Economic Statistics* 13: 253–63.

Efron, Bradley. 1979. Bootstrap methods: another look at the jackknife. *Annals of Statistics* . Vol. 7, pp. 1–26

Farrell, Max H. 2015. Robust inference on average treatment effects with possibly more covariates than observations. *Journal of Econometrics* 189: 1–23.

Farrell, Max H., Tengyuan Liang, and Sanjog Misra. 2019. Deep Neural Networks for Estimation and Inference. *arXiv preprint*, arXiv:1809.09953.

Friedman, Jerome H. 1994. An overview of predictive learning and function approximation. In *From Statistics to Neural Networks*. Berlin and Heidelberg: Springer, pp. 1–61.

Gal, Yarin, and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*. pp. 1050–59. New York, NY, USA.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT press.Cambridge, MA, USA. .

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. 2020. Empirical asset pricing via machine learning. *The Review of Financial Studies* 33: 2223–73.

Hansen, James V. and Ray D. Nelson. 2003. Forecasting and recombining time-series components by using neural networks. *Journal of the Operational Research Society* 54: 307–17.

Hornik, Kurt. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4: 251–57.

Hüllermeier, Eyke, and Willem Waegeman. 2020. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *arXiv preprint*. arXiv:1910.09457.

Ioffe, Sergey, and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint*. arXiv:1502.03167

Judd, J. Stephen. 1990. *Neural Network Design and The Complexity of Learning*. MIT press. Cambridge, MA, USA.

Kendall, Alex, and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision?. In *Advances in Neural Information Processing Systems*; pp. 5574–84. Long Beach, CA, USA. .

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521: 436–44.

Markowitz, Harry. 1952. Portfolio Selection. *The Journal of Finance*7: 77–91.

Méndez-Jiménez, Iván and Miguel Cárdenas-Montes. 2018. Time series decomposition for improving the forecasting performance of convolutional neural networks. In *Conference of the Spanish Association for Artificial Intelligence*. Cham: Springer, pp. 87–97.

Montufar, Guido F., Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*; pp. 2924–32. Montreal, Canada.

Pascanu, Razvan, Guido Montufar, and Yoshua Bengio. 2013. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint*, arXiv:1312.6098

Pearce, Tim, Alexandra Brintrup, Mohamed Zaki, and Andy Neely. 2018. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *International Conference on Machine Learning*. pp. 4075–84. Stockholm, Sweden.

Raghu, Maithra, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. 2017. On the expressive power of deep neural networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, August 6–11, Vol. 70, pp. 2847–54.

Schmidhuber, Jürgen. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61: 85–117.

Smyl, Slawek. 2020. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting* 36: 75–85.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*15: 1929–1958.

Tibshirani, Robert. 1996. A comparison of some error estimates for neural network model. *Neural Computation* 8: 152–63.