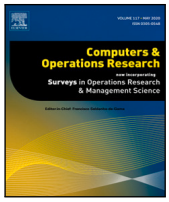
Computers & Operations Research 156 (2023) 106264

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Optimal deep neural networks by maximization of the approximation power

Hector Calvo-Pardo [a,b,c,d,1], Tullio Mancini [a,1], Jose Olmo [a,e,*,1]

[a] University of Southampton, UK
[b] CFS, Germany
[c] CPC, UK
[d] ILB, France
[e] Universidad de Zaragoza, Spain

ARTICLE INFO

Keywords:
Machine learning
Artificial intelligence
Data science
Forecasting
Feedforward neural networks

ABSTRACT

We propose an optimal architecture for deep neural networks of given size. The optimal architecture obtains from maximizing the lower bound of the maximum number of linear regions approximated by a deep neural network with a ReLu activation function. The accuracy of the approximation function relies on the neural network structure characterized by the number, dependence and hierarchy between the nodes within and across layers. We show how the accuracy of the approximation improves as we optimally choose the width and depth of the network. A Monte-Carlo simulation exercise illustrates the outperformance of the optimized architecture against cross-validation methods and gridsearch for linear and nonlinear prediction models. The application of this methodology to the Boston Housing dataset confirms empirically the outperformance of our method against state-of the-art machine learning models.

1. Introduction

Neural networks and, more specifically, deep learning models are extremely popular in high-dimensional problems such as pattern recognition, biomedical diagnosis, and others (see (Schmidhuber, 2015; LeCun et al., 2015) for overviews of the topic). These models are widely used in prediction tasks due to their unrivaled performance and flexibility in modeling complex unknown functions of the data. The success of these techniques rests in their ability to approximate complex unknown functional forms for the relationship between the outcome variable and the predictors.

The literature contemplates two types of models depending on the number of hidden layers characterizing the architecture of the network. A neural network with only one hidden layer (shallow network) can adequately approximate any given continuous function if its width is large enough (Cybenko, 1989; Hornik, 1991; Barron, 1994; Anthony and Bartlett, 1999). However, the width of such a hidden layer has to be exponentially large in order to approximate a given function to arbitrary precision, see Montufar et al. (2014). In contrast, when multiple layers are involved (deep neural networks), recent literature (Hanin and Sellke, 2017; Lu et al., 2017) has shown that to approximate any given integrable function to arbitrary precision multi-layer neural networks with width given by at most d + 1 units, with d the dimension of the input variables, are better suited. There is no consensus, though, on the depth of the network, that depends on the given function and may be very large. Furthermore, there are few hints on how to determine the suitable architectures needed to realize a given function or which architectures are more efficient.

Similarly, based on the literature of piecewise functions and function oscillation, Telgarsky (2016) shows that the function composition characterizing deeper architectures returns highly oscillatory functions and thus, it is better suited to fit an unknown target function compared to shallow structures of similar size. Deeper architectures, by capturing and learning the repeated regularities or hierarchical structures in the data, allow learning the underlying function with a lower level of model complexity (Aggarwal, 2018). For example, Telgarsky (2016) shows that, for any positive integer N, there exist some networks with depth $O(N^3)$, width $\Theta(1)$, and $\Theta(1)$ parameters, that cannot be approximated by an $O(N)$-layer network unless it has a width of $\Omega(2N)$, where N is the number of nodes in the neural network. Their results confirm further that deeper networks usually have more powerful expressivity of functions, which provides an explanation for why deeper networks outperform shallow networks with the same number of parameters in many practical tasks.[2]

* Correspondence to: Department of Economics, University of Southampton, Highfield Campus, SO17 1BJ, Southampton, UK.
E-mail addresses: J.B.Olmo@soton.ac.uk, joseolmo@unizar.es (J. Olmo).

[1] All authors have worked equally on this manuscript.

[2] We can compare the expressiveness of neural networks that use rectified linear units (ReLUs) by the number of linear regions, reflecting the number of pieces of the continuous piecewise linear functions modeled by such networks.

https://doi.org/10.1016/j.cor.2023.106264
Received 25 October 2021; Received in revised form 24 April 2023; Accepted 24 April 2023
Available online 28 April 2023

The approximation power or expressivity of neural networks not only depends on the number of hidden units but also on their allocation across layers. For example, in a rectifier network of depth N, not all configurations and in some cases none can reach the ceiling of 2^N active units, see Serra and Ramalingam (2019). On the one hand, we can construct networks where the number of active regions is exponential on network depth (Pascanu et al., 2013; Montufar et al., 2014). On the other hand, there is a bottleneck effect by which the number of active units on each layer affects how the regions are partitioned by subsequent layers, up to the point that even shallow networks define more linear regions than their deeper counterparts as the input dimension approaches N, see Serra et al. (2018). The correct specification of the width and depth of the network, together with other factors, ensures an efficient balance between bias and variance of the model predictions and thus, it ensures that the model will not over- or under-fit the underlying function.

One popular strategy in the implementation of neural networks is to determine the architecture of the network during training of the model. These methods use k-fold cross-validation to determine the width and depth of the network. In this scenario both variables are treated as model hyperparameters that are optimized while training. However, the performance of cross-validation approaches depends on (i) the number of variants of the learning models — if too few models are tuned we may miss the global optimum or if too many we may over-fit (Rao et al., 2008); (ii) the optimal division of the folds — observations must be *i.i.d.* and the distribution of the target variable must be similar across different folds; and (iii) the number of observations available — too few and it is not possible to correctly implement k-fold cross-validation.

Due to the centrality of the problem, alternative approaches have been proposed in the literature. Among many, Reiners et al. (2022) and Burke and Flanders (1995) explain how a key aspect in the learning and training of deep neural network is the capability of correctly selecting the network architecture. An optimal network structure ensures not only to avoid overparameterization and overfitting, but also to improve the optimization performance. Burke and Flanders (1995) propose to leverage a new class of networks called *ontogenic neural networks* that iteratively learn the optimal network structure during backpropagation via prototype units. Conversely, Reiners et al. (2022), suggest to consider a biobjective optimization problem. Implementing such a strategy entails differentiating – while training the neural network – between a measure used for prediction accuracy evaluation (e.g., cross-entropy in classification problems), and a penalty function used to assess the total complexity of the network parameters. This final component is coupled with a pruning algorithm that reduces the network structure while adding minimal additional computational cost.

In this paper, we show that it is possible to optimize the network architecture based on recent theoretical contributions that characterize the minimum expressive power of rectifier networks, i.e. a novel Neural Architecture Search (NAS) method for DNNs with ReLu activation functions. Similar approaches based on completely different techniques have been recently studied to automate the discovery of top-performing neural networks, e.g. TE-NAS by Chen and Gong (2021) or NASWOT by Mellor et al. (2021). The relevant question is whether we can select the best neural architectures without involving any training and, therefore, eliminating a large portion of the search cost. While Chen and Gong (2021) propose a novel framework called training-free neural architecture search (TE-NAS) based on optimally balancing the ex-ante trainability of the network and its expected accuracy, Mellor et al.'s (2021) method relies instead in identifying and minimizing the number of input space regions 'blind' to the neural network architecture. Instead, we propose an alternative methodology based on optimizing the allocation of units within and across layers so that the lower bound on the maximal number of linear regions approximated by ReLu DNNs is maximized. In doing so, we reduce the set of hyperparameters to be determined by cross-validation to the learning rate, number of epochs and drop-out rates. By optimizing width and depth prior to training architectures of a given size, our proposed method substantially saves upon the necessary time and computing power involved in tuning while training. More importantly, we reduce the approximation error and improve, in turn, the predictive ability of the DNN.

In most problems, it is not possible to obtain the number of active regions corresponding to the continuous piecewise linear functions expressed by rectifier networks. We approximate this number by the lower bound, more specifically, we focus on maximizing the lower bound of the theoretical maximal number of linear regions for a given dimension of the input variables and choice of nodes. By doing so, we obtain an architecture of the neural network that guarantees a theoretical minimum maximal number of active regions. Our study considers as benchmark the lower bound derived by Montufar et al. (2014) for deep fully-connected ReLU neural networks. Following their work, various results on the lower and upper bounds for the maximal number of linear regions of fully-connected ReLU neural networks have been obtained (Bianchini and Scarselli, 2014; Telgarsky, 2016; Poole et al., 2016; Montufar, 2017; Raghu et al., 2017; Serra et al., 2018; Croce et al., 2018; Serra et al., 2018; Hanin and Rolnik, 2019a,b). Xiong et al. (2020) obtain upper and lower bounds for the number of linear regions of multi-layer ReLU convoluted neural networks (CNNs). We focus on maximizing the lower bound because the upper bound is uninformative as the actual theoretical maximal number of activation regions may be far below the upper bound. Interestingly, though, the literature concerned with deriving upper bounds of the expressivity of a neural network is much more abundant than the literature studying lower bounds, see recent contributions by Raghu et al. (2017), Montufar (2017), Serra et al. (2018) and Hanin and Rolnik (2019a,b), as seminal contributions deriving theoretical and empirical upper bounds for the average and maximal number of linear regions approximated by rectifier networks.

Our choice of objective function for searching an optimal neural network architecture is derived from Montufar et al. (2014), but can be challenged from different angles. Many expressible functions are not efficiently learnable, at least by gradient descent (Shalev-Shwartz et al., 2017). Hanin and Rolnik (2019a,b) show that to adequately explain the power of deep learning it is necessary to consider networks with parameters as they will naturally occur before, during, and after training. These authors find that the number of linear regions is far below exponential (theoretical maximum). The typical behavior of a network used in practice, or practical expressivity, lags significantly behind its theoretical expressivity. Hanin and Rolnik (2019a,b) also suggest that for certain measures of complexity, trained deep networks are remarkably similar to the same networks at initialization implying that the improvement in the expressivity of the neural network during training is small. We overcome some of these criticisms with our proposed NAS optimization for rectifier networks. Other aspects such as the difference between the practical and theoretical expressivity of the neural networks are of an empirical nature and need to be studied on a case-by-case basis.

We assess empirically the performance of our *optimal* rectifier network architecture by measuring its predictive ability in different simulation experiments and in a toy example widely used in the empirical machine learning literature. The simulation exercise evaluates the out-of-sample performance of deep neural networks equipped with our optimal architecture against different benchmarks, depending on the data generating process considered. The main competitors that we consider for nonlinear data generating processes are state-of-the-art methods such as k-fold cross-validation and randomized grid search. In this setting, the predictive ability of our approach is shown to be superior to those methods. The simulation section also considers a linear data generating process that helps us understand the limits of our optimization procedure. We do this by comparing the predictions of our optimal DNN against the predictions of ordinary least squares (OLS) methods, that are known to be best linear unbiased estimators (BLUE) in linear settings. The out-of-sample mean square prediction

error (MSPE) and mean absolute prediction error (MAPE) obtained from our optimized ReLU DNN are comparable to those obtained from OLS methods in this unfavorable setting.

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

The accuracy of the approximation function provided by our *optimal* DNN model not only depends on the disposition of the network in terms of width and depth. It also depends on the number of nodes in the neural network. Our method is flexible on the number of nodes and allows to experiment with different choices depending on the complexity of the problem and dimension of the dataset.

Our work is also strictly related to the literature in operation research that has studied the application of neural network models to different areas of knowledge. For example, the literature focusing on on-line training aims at defining training strategies that reduce the computational time of deep structure to ensure continuous model update. The optimal network structure identification strategy proposed in the present paper could be coupled with the Tabu Search approach proposed by Marti and El-Fallahi (2004) to enhance the Multi-Start procedure. Alternatively, focusing on the branch of the literature studying the network training performance, the feature engineering strategy developed by Piramuthu et al. (1998) that ensures reduction in training complexity, could also be leveraged to provide an optimal feature dimension strategy to be used by the approach proposed in the following paper.

The rest of the paper is organized as follows: Section 2 reports the definitions and notations used in the paper, briefly reviewing the relationship between function class expressivity and universal approximation theorems for rectifier networks. Section 3 presents the novel methodology used for neural architecture search (NAS), reporting numerical results for the optimization. Section 4 presents a Monte Carlo simulation exercise that provides empirical evidence in finite samples of the performance of our method in improving the out-of-sample goodness of fit of deep neural networks for different data generating processes and input dimensions. In Section 5 the novel methodology is applied to the Boston Housing dataset. Section 6 concludes.

2. Background

Section 2.1 introduces relevant definitions and notations, briefly reviewing the literature on universal approximation theorems and the relation that subsists between continuous piecewise linear functions (CPWL) and rectifier networks (i.e. ReLu DNNs). Section 2.2 reports the main theory analyzing the number of linear regions approximated by both shallow and deep neural networks, illustrating the width-depth trade-off in the allocation of hidden units that informs our neural architecture search (NAS) strategy in Section 3.

Let y_i for $i = 1, \dots, n$ denote the outcome variable of interest and $\mathbf{x}_i = (x_{1i}, \dots, x_{di})$ a set of input variables used to predict the outcome. The contribution of the different variables is assumed to be additive such that

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \tag{1}$$

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

2.1. Definitions and notations

We will consider throughout the family of Rectified Linear Unit (ReLU) activation functions. These functions have proved recently more suitable to deep learning problems than sigmoidal activation functions (Jarrett et al., 2009; Nair and Hinton, 2010; Goodfellow et al., 2016; Géron, 2019). A ReLu activation function is defined as follows. Let $\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$, with

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

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

The corresponding DNN with ReLu activation functions is represented in Fig. 1 and defined as follows.

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

$$G = T_{N+1} \circ \theta \circ T_N \circ \cdots \circ T_2 \circ \theta \circ T_1. \tag{2}$$

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

$$T_n = \mathbf{W}^{(n)}\mathbf{x} + \mathbf{b}^{(n)}, \tag{3}$$

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

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

$$G(\mathbf{x}) = \boldsymbol{\omega}_O^\top \mathrm{diag}\left(\begin{bmatrix} \mathbb{I}(\mathbf{W}_1^{(1)}\mathbf{x} + b_1^{(1)}) \\ \vdots \\ \mathbb{I}(\mathbf{W}_{Z_1}^{(1)}\mathbf{x} + b_{Z_1}^{(1)}) \end{bmatrix}\right)(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + b^O, \tag{4}$$

where $\mathbf{W}_j^{(1)}$ identifies row j for $j = 1, \dots, Z_1$ of the matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{Z_1 \times d}$, $\mathbf{x} \in \mathbb{R}^{d \times 1}$ is the input layer, $b_j^{(1)} \in \mathbb{R}$ is the element j for $j = 1, \dots, Z_1$ of the random vector $\mathbf{b}^{(1)} \in \mathbb{R}^{Z_1}$, $\boldsymbol{\omega}_O \in \mathbb{R}^{Z_1}$, and $b^O \in \mathbb{R}$.

Definition 2 (*Continuous Piecewise Linear Functions (Arora et al., 2018)*)**.** We say a function $G : \mathbb{R}^d \to \mathbb{R}$ is continuous piecewise linear (CPWL) if there exists a finite set of closed sets whose union is $\mathbb{R}^d$, and G is affine linear over each set (note that the definition automatically implies the continuity of the function). The number of pieces of G is the number of maximal connected subsets of $\mathbb{R}^d$ over which G is affine linear.[3]

So-called universal approximation theorems (Cybenko, 1989; Hornik, 1991; Anthony and Bartlett, 1999) show that even with a single hidden layer, i.e., when the depth of the architecture is the smallest possible value, one can approximate arbitrarily well any continuous function by increasing the size of the network or the number of neurons used in the neural network. While this suggests that single layer networks are good enough from a learning perspective, Hertrik et al. (2021) have recently found that for rectifier networks (ReLu DNNs), (i) the class of CPWL functions represented by a single layer is a *strict subset* of the class of CPWL functions represented by two or more hidden layers, obtaining (ii) precise bounds on the size of the network of a given depth that can represent CPWL functions. Telgarsky (2016), Eldan and Shamir (2016), or Arora et al. (2018), show that it can be advantageous theoretically to increase the depth because a substantial reduction in the size can be achieved. Being the expressivity of the unknown function to be learned by rectifier networks characterized by the number of linear regions approximated by it (e.g. Montufar et al., 2014; Pascanu et al., 2013; Raghu et al., 2017 or Hanin and Rolnik, 2019a), Hertrik et al. (2021) provide a theoretical link between universal approximation theorems and the number of linear regions that informs our neural architecture search (NAS) strategy.

2.2. Number of linear regions

Definition 1 shows that each hidden unit has two operational nodes, one takes a value of zero and the other takes a positive value. The boundary between these two operational regions is given by the hyperplane H_j consisting of all the inputs $\mathbf{x} \in \mathbb{R}^d$ with $\mathbf{W}_j^{(n)}\mathbf{x} + b_j^{(n)}$. Therefore, the number of linear regions represented by a shallow ReLu neural network of size Z_1 is defined by the number of regions defined by the set of hyperplanes $\{H_j\}_{j\in[Z_1]}$. Zaslavsky (1975) proves that an arrangement of n hyperplanes can divide an $\mathbb{R}^d$ dimensional space in a number of regions equal to $\sum_{s=0}^{d}\binom{n}{s}$.

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

$$\sum_{s=0}^{d}\binom{Z_1}{s}. \tag{5}$$

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

$$\left(\prod_{j=1}^{N-1}\left\lfloor\frac{Z_j}{d}\right\rfloor^d\right)\sum_{s=0}^{d}\binom{Z_N}{s} \tag{6}$$

where $\lfloor\cdot\rfloor$ defines the "floor operator" which is the function that returns the greatest integer less than or equal to its real input argument. Based on the definition of ReLu DNN (Definition 1), Montufar et al. (2014) state that each hidden layer in a deep feedforward neural network folds the space of the previous hidden layer, with the recursive folding of the input space being determined by $\mathbf{W}$ and $\mathbf{b}$. This space folding implies that the final function computed by the last hidden layer (arrangement of Z_N hyperplanes) is applied to all the subsets identified by the succession of foldings performed by the deep structure, which for each hidden layer, divides the set of ReLu activation functions in d non-overlapping subsets of cardinality $\lfloor Z_j/d\rfloor$.

Corollary 6 of Montufar et al. (2014) compares the expressive power of single layer ReLu NNs with ReLu DNNs of same size Z and conclude that if the number of input variables d is $O(1)$, then the number of linear regions approximated by the latter DNN behaves as $\Omega\left(N^{-Nd}d^{-(N-1)d}Z^{Nd}\right)$.[4] In contrast, the number of linear regions approximated by its shallow NN counterpart behaves as $O\left(Z^d\right)$. Therefore, the number of regions grows exponentially in depth (N) and polynomially in width ($\frac{Z}{N}$) in ReLu DNNs, which is much faster than the polynomial growth of shallow ReLu NNs of same size Z. Thus, an increase in depth of a neural network should always lead to an exponential increase in the number of linear regions approximated by a ReLu DNN, which according to Hertrik et al. (2021), results in a better fit of the DNN architecture to the input data.[5]

The above results also suggest that a deep neural network is able to represent the same number of linear regions of a shallow network with a lower number of trainable parameters (hidden units). However, empirical studies show that this is not always the case (e.g. Pasupa and Sunhem, 2016; Kim and Gofman, 2018), motivating the 'information bottleneck effect' behind the *width-depth* trade-off identified by Serra et al. (2018). The following numerical examples make use of Eqs. (5) and (6) to illustrate Serra et al.'s (2018) width-depth trade-off, upon which we build our neural architecture search (NAS) strategy.

Let us consider a ReLu DNN given by $d = 4$ and $Z = 30$. The number of linear regions approximated by a shallow network is $\sum_{s=0}^{4}\binom{30}{s} = 31931$. In contrast, if we consider a ReLu DNN with $N = 2$, we can either have $\left\lfloor\frac{7}{4}\right\rfloor^4\sum_{s=0}^{4}\binom{23}{s} = 10903$ for $Z_1 = 7$ and $Z_2 = 23$ or $\left\lfloor\frac{8}{4}\right\rfloor^4\sum_{s=0}^{4}\binom{22}{s} = 145744$, for $Z_1 = 8$ and $Z_2 = 22$. Similarly, if we increase the depth of the ReLu DNNs to $N = 3$, we can either have $\left(\left\lfloor\frac{8}{4}\right\rfloor^4\left\lfloor\frac{8}{4}\right\rfloor^4\right)\sum_{s=0}^{4}\binom{14}{s} = 376576$ for $Z_1 = 8$, $Z_2 = 8$ and $Z_3 = 14$ or $\left(\left\lfloor\frac{7}{4}\right\rfloor^4\left\lfloor\frac{7}{4}\right\rfloor^4\right)\sum_{s=0}^{4}\binom{16}{s} = 2517$, for $Z_1 = 7$, $Z_2 = 7$ and $Z_3 = 16$. Finally, if we increase the depth to $N = 4$, we can either have $\left(\left\lfloor\frac{8}{4}\right\rfloor^4\left\lfloor\frac{8}{4}\right\rfloor^4\left\lfloor\frac{8}{4}\right\rfloor^4\right)\sum_{s=0}^{4}\binom{6}{s} = 233472$ for $Z_1 = 8$, $Z_2 = 8$, $Z_3 = 8$ and $Z_4 = 6$ or $\left(\left\lfloor\frac{7}{4}\right\rfloor^4\left\lfloor\frac{7}{4}\right\rfloor^4\left\lfloor\frac{7}{4}\right\rfloor^4\right)\sum_{s=0}^{4}\binom{9}{s} = 256$, for $Z_1 = 7$, $Z_2 = 7$, $Z_3 = 7$ and $Z_4 = 9$.

These examples show that under certain conditions a shallow network can provide better expressivity of the neural network than a DNN. The following results provide sufficient conditions that guarantee this result. Let $Z = z$ denote the total number of hidden nodes in the ReLu DNN.

Lemma 1. *Let a ReLu DNN satisfy the condition $(Z_1/d) < 2$. Then, the number of linear regions approximated by the ReLu DNN is lower than the number of linear regions approximated by the shallow network counterpart.*

The proof of this result is immediate by noting that the condition $(Z_1/d) < 2$ implies the following:

$$\left\lfloor\frac{Z_1}{d}\right\rfloor^d\sum_{s=0}^{d}\binom{z-Z_1}{s} < \sum_{s=0}^{d}\binom{z}{s}, \tag{7}$$

that represent the expressivity of a DNN (left expression) and a shallow neural network (right expression). This proposition can be extended to consider the condition $(Z_j/d) < 2$ for all $j = 1, \ldots, N-1$. More formally,

[3] The number of pieces where G is affine linear and the number of linear regions are used as synonyms throughout the paper.

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

[5] For completeness, we also note that an upper bound for the maximal number of linear regions of a function approximated by a network architecture with rectified linear units of size Z is of order $O\left(\left\lfloor\frac{Z}{N}\right\rfloor^{Zd}\right)$, as recently shown by Raghu et al. (2017), Theorem 1).

Lemma 2. *Let a ReLu DNN of depth N satisfy the condition $(Z_j/d) < 2$ for $j = 1, \dots, N-1$. Then,*

$$\left(\prod_{j=1}^{N-1} \left\lfloor \frac{Z_j}{d} \right\rfloor^d\right) \sum_{s=0}^{d} \binom{z - \sum_{j=1}^{N-1} Z_j}{s} < \sum_{s=0}^{d} \binom{z}{s}. \tag{8}$$

Under the conditions in these lemmas, we cannot guarantee that the architecture of the DNN improves over the architecture of the corresponding shallow network.

3. Optimal deep neural network structure

In this section we derive an optimal architecture of the DNN based on expression (6). This optimal architecture is obtained before bringing the model to the data, in line with but very different from recent neural architecture search (NAS) contributions (e.g. TE-NAS by Chen and Gong (2021) or NASWOT by Mellor et al. (2021)). In a second stage, we show how the optimal DNN outperforms cross-validation methods under mean square error (MSE) evaluation criteria implemented to simulated data (Section 4) and a real data example in Section 5.

3.1. Maximizing the lower bound of the number of linear regions

Our strategy for the identification of the optimal structure of a neural network is to maximize the lower bound of the maximal number of linear regions approximated by the ReLu DNN. By doing so, our aim is to maximize the expressivity of the ReLu DNN. The above subsection shows that not always DNNs outperform shallow networks as some of the recent literature confirms, see Serra et al. (2018). Our aim is to provide a methodology that allows us to maximize the expressivity of the neural network. We choose the lower bound of the number of linear regions derived by Montufar et al. (2014) in expression (6) but other objective functions are also possible.[6] Our optimization problem is to choose the optimal number of hidden layers N and nodes per layer $(Z_1, \dots, Z_N)$, for a given neural network size z. More formally, our optimization problem is laid out as follows:

$$\begin{aligned} \max_{N, \{Z_l\}_{l=1}^{N}} \quad & \left(\prod_{j=1}^{N-1} \left\lfloor \frac{Z_j}{d} \right\rfloor^d\right) \sum_{s=0}^{d} \binom{Z_N}{s} \\ \text{s.t.} \quad & Z_1 + Z_2 + \cdots + Z_N = z \\ & Z_j \geq d \quad \text{for } j = 1, 2, \dots, N \end{aligned} \tag{9}$$

The number of nodes cannot be smaller than d since the difference enters the denominator of the binomial coefficient in Eq. (6). Note that this is a standard assumption in the literature (Montufar et al., 2014; Pascanu et al., 2013; Mei et al., 2018; Hornik, 1991), and that — as stated by Aggarwal (2018) - the "*probabilistic regularization*" characterizing the stochastic gradient descent learning algorithm ensures a proper training of the hidden layers. The optimization procedure is complex and, in most cases, difficult to obtain analytically. The Lagrangian of the above objective function is

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

where $\{\mu_l\}_{l=1}^{N} \in \mathbb{R}^N$ denotes the collection of N Lagrange multipliers associated with the $N-1$ constraints, $Z_l \geq d, l = 1, \dots, N-1$, and with the constraint $N > 0$, because we have incorporated the equality constraint $Z = \sum_{l=1}^{N} Z_l$ into the lower bound objective function; $LB(N, \{Z_l\}_{l=1}^{N-1}; d) \equiv \left(\prod_{l=1}^{N-1} \left\lfloor \frac{Z_l}{d} \right\rfloor^d\right) \sum_{r=0}^{d} \binom{Z - \sum_{l=1}^{N-1} Z_l}{r}$. This is a combinatorial optimization problem because the decision variables $\{N, Z_1, \dots, Z_{N-1}\}$ are integer values. Judd (1990) shows that optimizing a NN is an NP-hard problem, meaning that a polynomial time algorithm that solves it is not known but could be found.[7]

One could instead try to approximate the combinatorial optimization problem by its continuous counterpart and obtain the following first order conditions (FOCs) characterizing an approximate optimal solution:

$$\begin{aligned} \partial_{Z_{l'}} LB(N^*, \{Z_l^*\}_{l=1}^{N^*-1}; d) - \mu_{l'} = 0, \text{ for } l' = 1, \dots, N-1, \\ \partial_{N_L} LB(N^*, \{Z_l^*\}_{l=1}^{N^*-1}; d) - \mu_N = 0, \\ \mu_{l'}(d - Z_{l'}^*) = 0, \text{ for } l' = 1, \dots, N-1 \\ -\mu_N N^* = 0. \end{aligned} \tag{10}$$

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

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

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

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

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

Solving the above system, we obtain that:

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

from which:

$$\left(1 + \frac{Z_2}{2} + \frac{Z_2^2}{2}\right) = \left(\frac{Z - Z_2}{2}\right)\left(\frac{1}{2} + Z_2\right). \tag{11}$$

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

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

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

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

[6] By maximizing the lower bound of the number of linear regions we follow a minimax strategy that aims to provide an optimal architecture of the ReLu neural network under the worst case scenario. Other authors such as Montufar (2017) or Hanin and Rolnick (2019b) focus on the upper bounds or the expected number of linear regions, respectively.

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

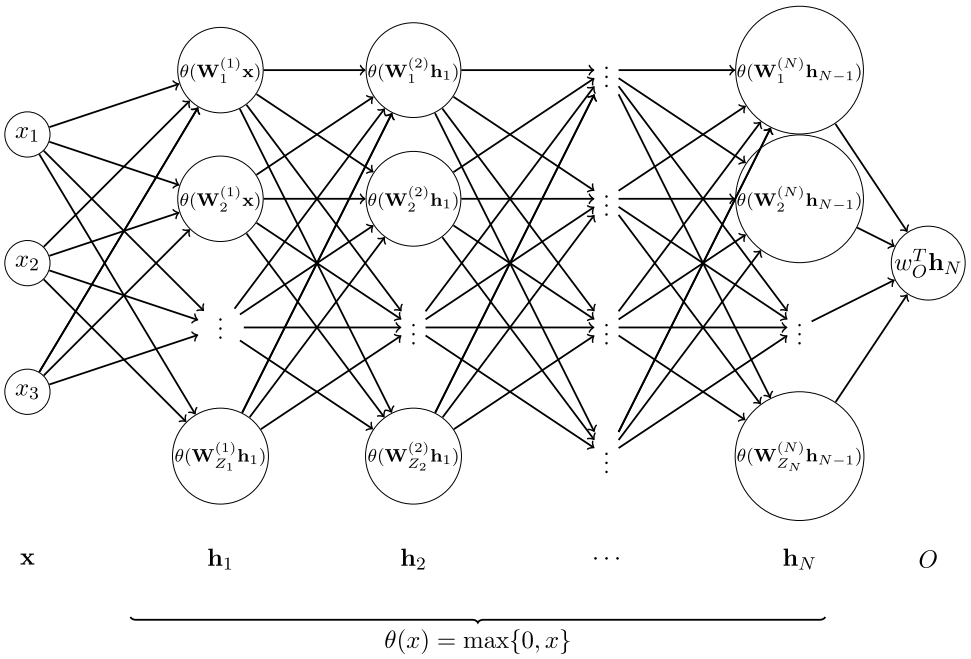


Fig. 1. ReLu Deep Neural Network with bias terms 0.

and solving the above system, we obtain that:

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

from which:

$$\left(1+\frac{Z_2}{2}+\frac{Z_2^2}{2}\right)=\left(\frac{Z-Z_2-1}{2}\right)\left(\frac{1}{2}+Z_2\right). \tag{12}$$

To obtain a numerical solution, let z take a specific value for the number of total nodes Z, e.g. $z = 60$. Then, the first order conditions (11) and (12) accept the following two solutions:

$$\begin{cases} Z_2^* = (117+\sqrt{14585})/8 = 29.72 \approx 30 \\ Z_2^* = (117-\sqrt{14585})/8 = -0.47 \approx 0 \end{cases} \tag{13}$$

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

$$\mathbf{H}^B = \begin{bmatrix} 0 & g_1 & g_2 \\ g_1 & \mathcal{L}_{11} & \mathcal{L}_{12} \\ g_2 & \mathcal{L}_{21} & \mathcal{L}_{22}, \end{bmatrix} \tag{14}$$

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

The left panel of Fig. 2 plots the function $\left\lfloor \frac{Z_1}{2} \right\rfloor^2 \left[1+\frac{z-Z_1}{2}+\frac{(z-Z_1)^2}{2}\right]$ assuming that Z_1 is a real number defined on the set $Z_1 \in [2, z-2]$, for $z = 60$. The plot shows the saw type function implied by the floor operator. Despite this, the existence of a global maximum is clear and is found at $Z_1^* = 30$ as shown in our analytical derivation. For completeness, we also plot in the right panel the smooth version of function (6) that does not consider the floor operator.

The above example confirms the presence of an interior solution to the optimization problem (9) for low-dimensional DNNs. In practice, the difficulty of the optimization for realistic values of the number of nodes and layers implies that the solution to (9) needs to be achieved through numerical methods.

3.2. Numerical optimization

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

Under these optimization constraints, a simple procedure to obtain the solution to (9) for a given number of nodes z is to evaluate the objective function for all possible combinations of integer numbers that satisfy the constraints $Z_1 + Z_2 + \cdots + Z_N = Z$ and $Z_j \geq d$ for $j = 1, \ldots, N$. This is the low-dimensional version of the nonlinear integer programming method discussed above. This procedure yields exact solutions to the optimization problem (9) for low values of N and Z. However, the problem becomes computationally intractable as N and, in particular, Z increase. In this scenario quasi newton algorithms designed to solve constrained optimization problems are more suitable. In particular, the L-BFGS-B algorithm, which is the limited-memory quasi newton algorithm designed to solve constrained optimization problems, is used to return the parameters that optimize the objective function (9). The obtained solutions are also compared against the SLSQP – Sequential Least Squares Programming algorithm – optimizer implemented on Pytorch. Both algorithms – in contrast to others, such as the Stochastic Gradient Descent – allow for the floor operator to be taken care of. The comparison between the L-BFGS-B and the SLSQP algorithm is conducted solely with the intention to control for the correct convergence of the former, as they both provide satisfactory solutions.

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

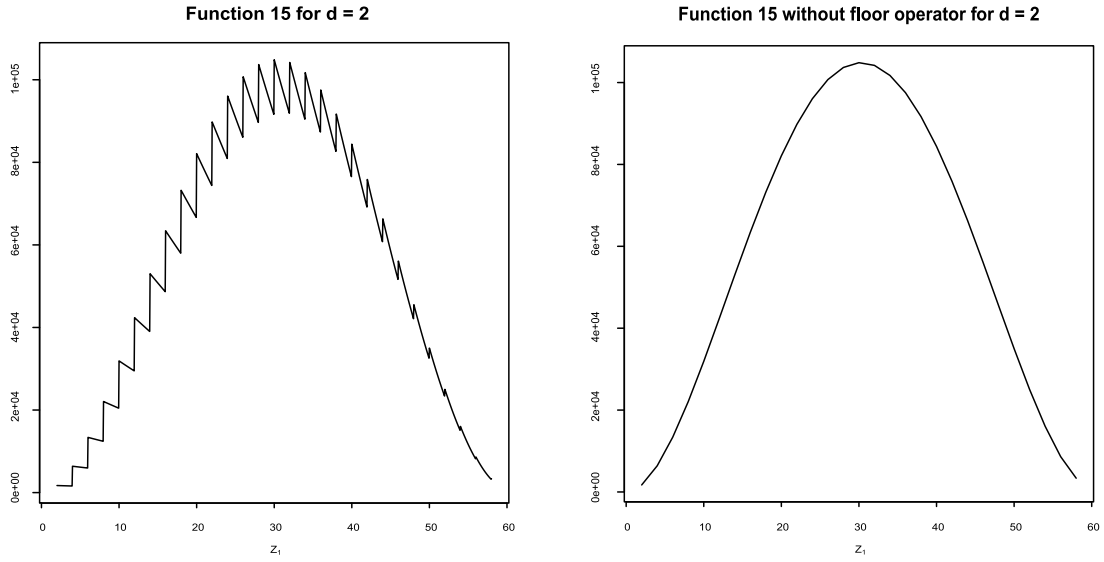


Fig. 2. The Figure reports the lower bound to the maximal number of linear regions for the ReLu DNN considered in the above maximization problem.

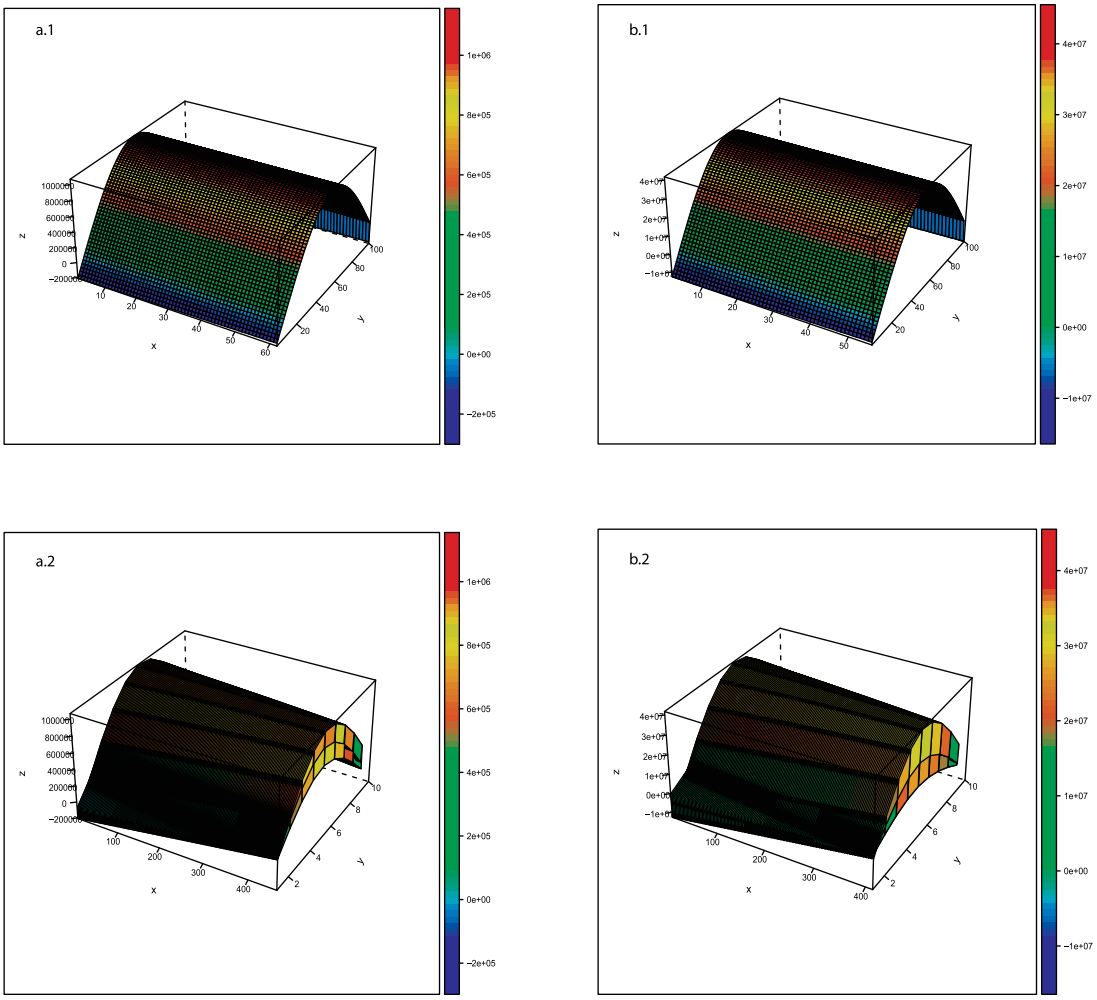


Fig. 3. The Figure reports the number of linear regions as a function of the depth and the different combinations obtainable with the given number of hidden nodes. Sub-Figures $a.1$ and $b.1$ report the plane project of the number of linear regions as a function of the possible structures associated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

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

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

Fig. 3 investigates this eventuality. In particular, Figures *a*.1 and *b*.1 provide a three-dimensional representation of a two dimensional problem, that is of the optimization of the layer wise width of the network, for a given depth and number of hidden nodes. The plots are obtained by fitting a generalized additive model between the number of linear regions associated to each structure combination obtainable when $d = 2, 3$ and $N = 3$, and the possible combinations. The fitted curve is then replicated along the x-axis in order to generate a plane for a better visualization of the convex problem solved by the optimization. The different figures confirm that the maximization problem has an interior solution and, thus, an optimal combination of widths that, for a given depth, maximizes the number of linear regions approximated by the neural network. In contrast, Figures *a*.2 and *b*.2, project the *complete* optimization problem. The optimization problem is considered not only in terms of the optimal width of the network, but also in terms of optimal depth. The y-axis represents the number of possible combinations associated to each hidden layer, while the x-axis identifies the different depths of the neural network considered, and the z-axis represents the number of linear regions associated to each combination for a given neural network depth.[8] Fig. 3 shows that the analyzed optimization problem (expression (9)) is convex if it is considered either for a given depth, or if it is considered in terms of optimal width and depth. Therefore, there exists a maximum for the lower bound of the maximal number of linear regions in terms of both width and depth and, importantly, the strategy of decomposing the optimization problem (9) in the two step procedure previously suggested is able to capture the global optimum.

Table 1 reports the results from the optimization exercise (9) for different input dimensions. For completeness, we report the maximization exercise from the first stage that allows us to obtain an optimal width for given depths, and then identifying the optimal combination of width and depth that maximizes the number of linear regions approximated by a neural network.

We also consider the minimization of (6). It is important to specify that the purpose of the numerical minimization is merely illustrative and reported to provide a more complete understanding of the duality between deep and shallow networks. As mentioned above, if the minimum number of linear regions approximated by a ReLu DNN is always higher than the number of linear regions approximated by the shallow network, it would be possible to conclude that – regardless the structure adopted – a deep network always outperforms the shallow network counterpart. Table 1 reports the results from the numerical minimization of the number of linear regions approximated by a ReLu DNN. The results reported in Table 1 show that the minimum number of linear regions approximated by a deep neural network (for different depths) is always lower than the number of regions approximated by the shallow counterpart (with the same number of hidden nodes). This exercise shows that not every DNN outperforms a shallow network and motivates the need of an optimization procedure to maximize the lower bound (6). Summing up, to be certain that the DNN provides a better approximation than the shallow network one needs to maximize the lower bound in expression (9) and show that the solution is larger than the number of linear regions of the shallow network counterpart.

Table 1 also shows that when the structure that minimizes the number of linear regions is considered, an increase in depth leads to a decrease in the number of linear regions approximated. Fig. 4 shows that the number of linear regions decreases linearly when $\mathbf{x} \in \mathbb{R}^2$. As the dimension of the input layer increases – or the ratio between input dimension and number of hidden nodes per layer decreases – the observed decrease in the number of linear regions approximated by the ReLu DNN becomes gradually exponential. This behavior can be justified by the fact that a higher input dimension will require a higher number of maximum nodes for the minimum representation. The increase of hidden nodes per layer will lead to a decrease in the number of linear regions which is exponential in the ratio between input dimension and width of the hidden layers.

To summarize, the minimization exercise shows that a ReLu DNN with sub-optimal structure underperforms a shallow network with the

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

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

For input dimensions equal to 2 and 3, an increase in the depth of the ReLu DNN (see Fig. 4) results in an exponential increase in the number of linear regions approximated. However, when $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$, an increase in depth leads to an exponential increase in the number of linear regions up to a certain value beyond which the increase in depth leads to a decrease in the number of linear regions (Lemma 2 is violated). The maximum value in Fig. 4 can be interpreted as the depth of the DNN beyond which Lemma 2 is violated. When the minimum ratio is $\lfloor Z_j/d \rfloor = 2$, deeper structures underperform shallower ones. Ultimately, the observed decrease in the number of linear regions leads – in the case of $x \in \mathbb{R}^6$ and $N = 7$ – to the underperformance of the ReLu DNN when compared to the shallow counterpart. These examples show that when Lemma 2 is satisfied depth is always better than width.

The results in Table 1 illustrate the existence – for different input dimensions – of an optimal architecture in terms of both depth and width of the network. The optimization exercises for different number of layers show that increasing the depth, for a given number of hidden nodes, not always leads to an increase in the number of affine regions approximated by the neural network. This aspect ensures the existence of an optimal neural network depth, with a given optimal layer width. The maximum depth considered in this numerical exercise allows identifying the optimal depth – with corresponding optimal layer widths – for $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$.

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

4. Monte Carlo simulation

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

The present algorithm optimizes the width and depth of a neural network of a given size $Z = z$. To ensure the robustness of our results, we repeat the optimization over different candidates $z = 40, 60, 90$. We also vary the input dimensions and consider five different DGPs, one linear and four nonlinear. When the linear DGP is considered, the out-of-sample performance of the optimal neural network is compared against the OLS estimator. The rationale for this exercise is to assess the predictive power of deep neural networks in an unfavorable context in which OLS methods are proved to be optimal. When nonlinear generating processes are considered, the out-of-sample performance of the structure selected with the above optimal methodology is compared with the out-of-sample performance of a structure selected with k -fold cross-validation.

4.1. Data generating process

We consider the following linear and nonlinear DGPs.

Model 1 (Linear Process):

$$y = a + \mathbf{a}\mathbf{x} + \epsilon, \tag{15}$$

with different input dimensions: $\mathbf{x} \in \mathbb{R}^4$, $\mathbf{x} \in \mathbb{R}^5$, and $\mathbf{x} \in \mathbb{R}^6 \sim \mathcal{N}(\mu, 1)$. The parameters chosen for the vector of coefficients $\mathbf{a}$ are generated from a $U(-10, 10)$ and then rounded to the closest digit.[9] Similarly, the parameter for the vector of means μ are generated from $U(-5, 5)$ and then rounded to the closest digit. When $\mathbf{x} \in \mathbb{R}^4$, $\mathbf{a} = [-8, 2, 2, 2]^\top$ and $\mu = [-4, 1, 1, 1]$; When $\mathbf{x} \in \mathbb{R}^5$, $\mathbf{a} = [-8, 2, 2, 2, 7]^\top$ and $\mu = [-4, 1, 1, 1, 5]$, when $\mathbf{x} \in \mathbb{R}^6$, $\mathbf{a} = [-8, 2, 2, 2, 7, 3]^\top$ and $\mu = [-4, 1, 1, 1, 5, 1]$. The error term is $\epsilon \sim \mathcal{N}(0, 1)$ that is uncorrelated to the input variables.

In the nonlinear case we consider four different DGPs.

Model 2:

$$y = a_1 x_1 + (5e^{-6})(1 - e^{a_2 x_2 + a_3 x_3}) + a_4 x_4^2 + \epsilon. \tag{16}$$

Model 3:

$$y = a_1 x_1^2 + e^{-6}\frac{a_2 x_2}{a_3 x_3} + e^{-6}(a_4 x_4)(a_5 x_5) + \epsilon. \tag{17}$$

Model 4:

$$y = a_1 x_1^2 + e^{-6}\frac{a_2 x_2}{a_3 x_3} + a_4^3 x_4 + e^{-6}(a_5 x_5)(a_6 x_6) + \epsilon. \tag{18}$$

Model 5:

$$y = a + \mathbf{a}\mathbf{x}(4x_1 - 0.5x_2) + \epsilon. \tag{19}$$

The nonlinear DGPs introduced by expressions (16), (17) and (18) incorporate the nonlinearity of the process predominantly in terms of extreme variations. Models 2 to 4 are multiplied by scaling factors of the order e^{-6} to reduce the contribution of the input variables; therefore, by re-scaling the marginal effects, the process y is comprised only by those observations that are "extreme" whereas those marginal effects of lower magnitude are re-scaled such that y is approximately zero in those cases. Model 5 is the nonlinear counterpart of Model 1, that considers interactions between the different input variables.

4.2. Accuracy test

The main objective of this subsection is to compare the out-of-sample performance of the proposed novel methodology against a structure selected via k-fold cross-validation. By performing this methodology, the structure that returns the lowest out-of-sample mean square error (MSE) will be selected. The procedure is as follows. We simulate 1500 observations from the different DGPs; 1200 observations are used for training the model and 300 observations are used as validation set for the final comparison. A 3 -fold cross-validation over a randomized grid search[10] is performed over the 1200 observations to select the structure that returns the lowest out-of-sample MSE. Thus, being both neural networks fitted using the same number of training observations and on the same dataset, the comparison carried out for the nonlinear DGP processes analyses two comparable neural networks.

Table 1 shows that, for a given input dimension, it is possible to identify the optimal structure in terms of width and depth of the network. In particular, we identify the optimal structure when $x \in \mathbb{R}^4, \mathbb{R}^5, \mathbb{R}^6$. Prior to performing k -fold cross-validation, we consider two neural networks with the same depth, obtained from the results in Table 1, such that the comparison is done in terms of optimal number of nodes per layer (optimal width) for a given depth of the network. As a result, for a given input dimension and for a given optimal depth, the two structures – one obtained from the maximization proposed in the paper, and the other obtained from k-fold cross-validation – will be compared in terms of MSE on the validation set (300 observations).[11]

The objective of the current simulation settings is to compare two different models using a pre-specified loss function, and see if the

[9] Before the generation of the simulated parameter the seed was set to 1234.

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

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

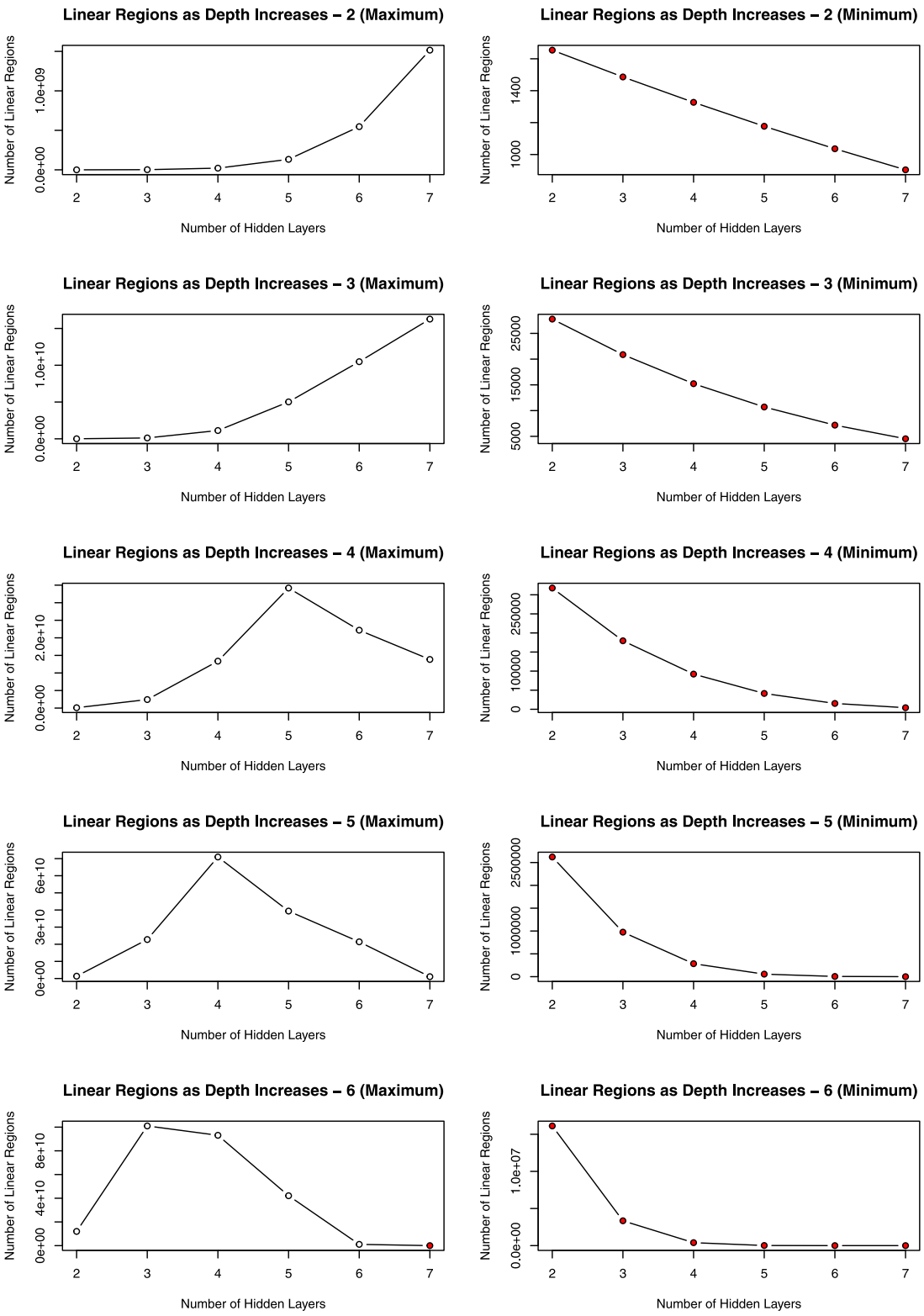


Fig. 4. The Figure reports the relation between the number of linear regions and the depth of the neural network for different input dimensions. The color red indicated an approximated number of linear regions lower than the shallow counterpart.

differences are statistically significant. In our case, we want to test if the novel methodology returns an accuracy (B) higher than the one returned by a model selected via k-fold cross-validation (A). The hypothesis can be written as:

$$\mathcal{H}_0 \ : \ E[L_A(\hat{y}, y) - L_B(\hat{y}, y)] \leq 0 \tag{20}$$

$$\mathcal{H}_1 \ : \ E[L_A(\hat{y}, y) - L_B(\hat{y}, y)] > 0, \tag{21}$$

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

and A and B denote two different models[12] for predicting the response variable y. Under the alternative hypothesis, Model B exhibits a lower MSE than Model A. The empirical counterpart of the above expectation is $\overline{MSE}_m \equiv \hat{E}[L(\hat{y}_m, y_m)] = \frac{1}{F}\sum_{f=1}^{F}(\hat{y}_{mf} - y_{mf})^2$, with $m = A, B$ and F the out-of-sample evaluation period. A feasible test statistic is of the form

$$T_F = \sqrt{F}\frac{\overline{MSE}_A - \overline{MSE}_B}{\sqrt{\hat{V}(\overline{MSE}_A - \overline{MSE}_B)}}. \tag{22}$$

Under the null hypothesis, this statistic converges to a standard normal distribution such that rejection of the null hypothesis can be assessed from a one-sided test with critical value $z_{1-\alpha}$, with α the significance level of the test. Furthermore, if the residuals, defined as $e_{mf} = y_f - \hat{y}_{mf}$, with $m = A, B$ and $f = 1, \dots, F$ are serially uncorrelated, the relevant variance of the statistic is

$$V\left(\sqrt{F}(\overline{MSE}_A - \overline{MSE}_B)\right) = V(e_{Af}^2) + V(e_{Bf}^2) - 2Cov(e_{Af}^2, e_{Bf}^2), \tag{23}$$

with e_{Af} and e_{Bf} the residuals of each model. Then, a suitable estimator of the asymptotic variance is

$$\begin{aligned}\hat{V}(\overline{MSE}_A - \overline{MSE}_B) = &\frac{1}{F}\sum_{f=1}^{F}(e_{Af}^2 - \overline{e_A^2})^2 + \frac{1}{F}\sum_{f=1}^{F}(e_{Bf}^2 - \overline{e_B^2})^2 \\ &- 2\frac{1}{F}\sum_{f=1}^{F}(e_{Af}^2 - \overline{e_A^2})(e_{Bf}^2 - \overline{e_B^2}),\end{aligned} \tag{24}$$

with $\overline{e_m^2} = \frac{1}{F}\sum_{f=1}^{F} e_{mf}^2$, with $m = A, B$.

If there is serial dependence then the variance of the test statistic is more complex and we need to incorporate the presence of serial correlation. Robust estimators are HAC estimators developed by Newey and West (1987). Note that the difference between the in-sample and out-of-sample exercise is how we construct the residuals e_{mf}.[13]

4.3. Simulation results

Table 2 reports the results from the Monte Carlo simulation. When the linear data generating process is considered the out-of-sample accuracy of the neural network with structure obtained via the proposed maximization (to which we will refer as optimal ReLu DNN) is compared against a linear regression and thus, a model that correctly identifies the DGP. In this case when considering the null hypothesis (20), the out-of-sample MSE of the linear model will estimate the loss function L_B; failing to reject the null hypothesis indicates that there is no statistically significant difference between the out-of-sample MSE of the optimal ReLu DNN and the corresponding MSE of the OLS estimator. Both models are indistinguishable in terms of predictive accuracy in mean. From Table 2, one could notice that the null hypothesis is rejected at 0.1 significance level for $\mathbf{x} \in \mathbb{R}^4$ and $z = 90$ and for $\mathbf{x} \in \mathbb{R}^5$ and $z = 40$. In all other cases, the difference in accuracy is not significant.

Overall, these results suggest that the optimal neural network architecture proposed in this paper is comparable in terms of predictive accuracy to the OLS estimator for the linear DGP. When nonlinear DGPs are considered, the null hypothesis (20) is used to compare the out-of-sample performance of the optimal neural network against a cross-validated structure. Table 3 reports the structures returned from the proposed methodology and from the 3-fold cross-validation. The out-of-sample MSE of the linear model is still reported for completeness, but as the comparison with the performance of neural networks is no longer meaningful, the test of the null hypothesis (20) is only reported for the comparison between the optimal DNN model and the 3-fold cross-validated model. The out-of-sample MSE of the optimal model is denoted as L_B; thus, rejecting the null hypothesis (20) will provide evidence of the outperformance of the optimal neural network over a cross-validated one. With the exception of three cases, the null hypothesis is rejected in all cases at 0.1 significance level. When $\mathbf{x} \in \mathbb{R}^6$ and $z = 40$ both models are not able to approximate adequately the fourth nonlinear data generating process; conversely, when $z = 60, 90$ the out-of-sample error of the two models decreases drastically and the null hypothesis is rejected at 0.01 significance level, suggesting the outperformance of our DNN architecture.

To summarize, the results reported in Table 2 show that the optimal neural network has an out-of-sample performance statistically equivalent to the MSE of the OLS BLUE estimator when the true data generating process is linear. These results are robust to the different number of hidden nodes considered, showing that the outperformance does not depend on the specific number of hidden nodes chosen. For nonlinear processes, the DNN model with the structure selected via the proposed maximization outperforms a neural network whose structure is obtained via a 3-fold cross-validation with a randomized grid search. This result is true for all the different input dimensions and nonlinear DGPs considered. Our findings also show that – given a specific DGP – there exists an optimal number of hidden nodes that minimizes the MSE. For example, when a linear DGP is considered $z = 40$ returns the lowest MSE, while when the nonlinear DGPs are considered, the MSE is minimized for $z = 60$ or $z = 90$ (when the true DGP is linear, a high number of hidden nodes may result in overparametrization).

5. Empirical application

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

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

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

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

[13] In the in-sample exercise these residuals are obtained from the fitted model, and in the out-of-sample exercise the residuals are computed from the out-of-sample observations, that is, the model is given.

Table 2
The Table reports, for each input dimension and each data generating process considered, the out-of-sample MSE of the models considered, the test statistic and p value for the difference in accuracy.

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

*Indicated 0.1 significance level.
**Indicated 0.05 significance level.
***Indicates 0.01 significance level.

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

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

feedforward neural network, different dropout rates $p = 0.1, 0.2, 0.3$ are tuned for all hidden layers, and $p = 0.1$ for the input layer.

The proposed optimization procedure lets the total number of hidden nodes as a free parameter. In this application, we consider $Z = z =$

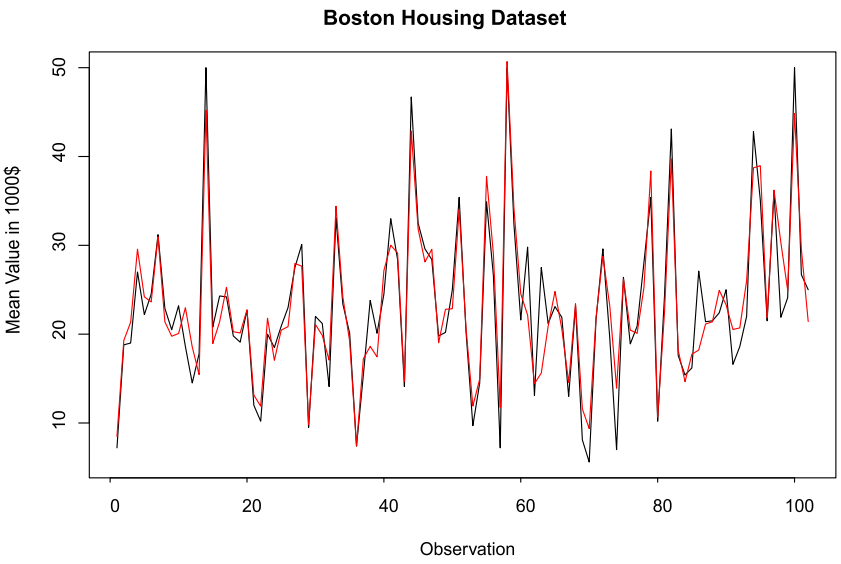


Fig. 5. The Figure reports the observed average prices (in black) against the fitted values (in red) for the validation set.

50, 100, 130, 150, and 200. The number of input variables is $d = 13$, and we allow for a maximum depth of 10. The results of our optimization exercise in (9) are as follows. For $z = 50$, the optimal structure of the DNN is $[37, 13]$; for $z = 100$ is $[48, 26, 26]$; for $z = 130$ is $[52, 39, 39]$; for $z = 150$ is $[33, 39, 39, 39]$ and, finally, for $z = 200$ is $[44, 39, 39, 39, 39]$. To check the convergence of the optimization algorithm – for each z – the maximization is conducted using $z + / - 1$.

Each combination is evaluated using 4-fold cross-validation on the training set, and finally evaluated on the validation set.[14] Thus, the predictive performance of our method is measured by averaging the four MAEs and MSEs obtained from the training samples under the 4-fold cross-validation, and the validation MSE and MAE obtained from fitting the tuned model on the validation set. It is important to remember that our proposed DNN architecture focuses on optimizing the structure of the network but is silent about the specific choice of the aforementioned hyperparameters. Therefore, the 4-folds cross-validation is used to choose the optimal combination of learning rate, optimizer, weight initializer, number of epoch, and dropout rate. The optimal combination of nodes and hyperparameters is evaluated on the validation set.[15]

Based on the out-of-sample accuracy, the best combination is defined by the RMSProp optimizer with learning rate 0.001, 2000 epochs, $z = 130$, a dropout rate of 0.1 across all hidden layers, and no dropout in the input layer. The cross-validated MSE and MAE are 7.67 and 2.02 respectively, and the validation MSE and MAE are 8.76 and 2.17. Fig. 5 reports the fitted values out-of-sample of the trained ReLu DNN against the observed values. An out-of-sample MAE of 2.17 implies that the model will predict house prices with an error – on average – of 2170$.

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

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

6. Conclusions

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

The performance of the proposed optimal architecture for deep neural networks is assessed in an exhaustive Monte-Carlo exercise and also empirically. This novel procedure is shown to outperform k-fold cross-validation procedures for prediction in nonlinear models. In linear settings, in which standard OLS methods are optimal, our approach is competitive and provides comparable mean square error values. We illustrate the ability of our optimal deep neural network architecture to predict median house prices from the Boston Housing dataset. This dataset is extensively used by the machine learning literature to validate new learning techniques. Our neural network architecture reduces

[14] The 4-fold cross-validation is implemented to replicate the procedure adopted by the majority of the referenced literature.

[15] This cross-validation exercise to optimize the tuning parameters is different from the 3-fold cross-validation method used in the Monte Carlo section as benchmark model to obtain the out-of-sample MSE.

the out-of-sample mean square prediction error between 28.55% and 47.32% compared to recent studies fitting neural networks to this dataset. By optimizing width and depth prior to training for a given choice of nodes, our proposed method substantially saves upon the necessary time and computing power involved in fine tuning while training.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

Jose Olmo acknowledges financial support from project PID2019-104326GB-I00 from Ministerio de Ciencia e Innovación and from Fundación Agencia Aragonesa para la Investigación y el Desarrollo (ARAID).

References

Aggarwal, C.C., 2018. Neural Networks and Deep Learning, No. 10. Springer, 978–3.

Al Bataineh, A., Kaur, D., 2018. A comparative study of different curve fitting algorithms in artificial neural network using housing dataset. In: NAECON 2018-IEEE National Aerospace and Electronics Conference. pp. 174–178.

Anthony, M., Bartlett, P.L., 1999. Neural Network Learning: Theoretical Foundations. Cambridge University Press.

Arora, R., Basu, A., Mianjy, P., Mukherjee, A., 2018. Understanding deep neural networks with rectified linear units. In: International Conference on Learning Representations.

Bakker, B., Heskes, T., 2003. Clustering ensembles of neural network models. Neural Netw. 16 (2), 261–269.

Barron, A.R., 1994. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. Inform. Theory 39 (3), 930–945.

Bengio, Y., 2009. Learning deep architectures for AI. Found. Trends Mach. Learn. 2 (1), 1–127.

Bianchini, M., Scarselli, F., 2014. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. IEEE Trans. Neural Netw. Learn. Syst. 25 (8), 1553–1565.

Burke, L., Flanders, S., 1995. Using ontogenic classification networks in a smart structures application. Comput. Oper. Res. 22 (9), 871–881.

Chen, W., Gong, X., 2021. Neural architecture search on IMAGENET in four GPU hours: A theoretically inspired perspective. In: ICLR 2021 Conference Paper. pp. 1–15.

Ciresan, D., Meier, U., Masci, J., Schmidhuber, J., 2012. Multi-column deep neural network for traffic sign classification. Neural Netw. 32, 333–338.

Croce, P., Zappasodi, F., Marzetti, L., Merla, A., Pizzella, V., Chiarelli, A.M., 2018. Deep convolutional neural networks for feature-less automatic classification of independent components in multi-channel electrophysiological brain recordings. IEEE Trans. Biomed. Eng. 66 (8), 2372–2380.

Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function Mathematics of control. Signals Syst. 2 (4), 303–314.

Eldan, R., Shamir, O., 2016. The power of depth for feedforward neural networks. In: Conference on Learning Theory. PMLR, pp. 907–940.

Géron, A., 2019. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

Goodfellow, I.J., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.

Goodfellow, I.J., Bulatov, Y., Ibarz, J., Arnoud, S., Shet, V., 2013. Multi-digit number recognition from street view imagery using deep convolutional neural networks. arXiv preprint arXiv:1312.6082.

Granitto, P.M., Verdes, P.F., Navone, H.D., Ceccatto, H.A., 2001. A late-stopping method for optimal aggregation of neural networks. Int. J. Neural Syst. 11 (03), 305–310.

Hanin, B., Rolnik, D., 2019a. Complexity of linear regions in deep networks. In: ICML.

Hanin, B., Rolnik, D., 2019b. Deep ReLU networks have surprisingly few activation patterns. In: NeurIPS.

Hanin, B., Sellke, M., 2017. Approximating continuous functions by ReLU nets of minimal width. arXiv preprint arXiv:1710.11278.

Harrison, D., Rubinfeld, D.L., 1978. Hedonic housing prices and the demand forclean air. J. Environ. Econ. Manage. 5, 81–102.

Hertrik, C., Basu, A., Di Summa, M., Skutella, M., 2021. Towards lower bounds on the depth of ReLU neural networks. available at https://arxiv.org/abs/2105.14835.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Netw. 4 (2), 251–257.

Jarrett, K., Kavukcuoglu, K., Ranzato, M.A., LeCun, Y., 2009. What is the best multi-stage architecture for object recognition? In: 2009 IEEE 12th International Conference on Computer Vision. pp. 2146–2153.

Judd, J.S., 1990. Neural Network Design and the Complexity of Learning. MIT Press.

Kim, D.E., Gofman, M., 2018. Comparison of shallow and deep neural networks for network intrusion detection. In: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference. CCWC, IEEE, pp. 204–208.

Kohn, R., Ansley, C.F., Tharm, D., 1991. The performance of cross-validation and maximum likelihood estimators of spline smoothing parameters. J. Amer. Statist. Assoc. 86 (416), 1042–1050.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444.

Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L., 2017. The expressive power of neural networks: A view from the width. In: Advances in Neural Information Processing Systems. pp. 6231–6239.

Marti, R., El-Fallahi, A., 2004. Multilayer neural networks: an experimental evaluation of on-line training methods. Comput. Oper. Res. 31 (9), 1491–1513.

Mei, S., Montanari, A., Nguyen, P.M., 2018. A mean field view of the landscape of two-layer neural networks. Proc. Natl. Acad. Sci. 115 (33), 7665–7671.

Mellor, H., Turner, J., Storkey, A., Crowley, E.J., 2021. Neural architecture search without training. available at https://arxiv.org/pdf/2006.04647.pdf.

Montufar, G.G., 2017. Notes on the number of linear regions of deep neural networks. Available online at https://www.researchgate.net/profile/Guido-Montufar/publication/322539221_Notes_on_the_number_of_linear_regions_of_deep_neural_networks/links/5a5f0dd7458515c03ee1bdb6/Notes-on-the-number-of-linear-regions-of-deep-neural-networks.pdf.

Montufar, G.F., Pascanu, R., Cho, K., Bengio, Y., 2014. On the number of linear regions of deep neural networks. In: Advances in Neural Information Processing Systems. pp. 2924–2932.

Myshkov, P., Julier, S., 2016. Posterior distribution analysis for Bayesian inference in neural networks. In: Workshop on Bayesian Deep Learning.

Nado, Z., Snoek, J., Grosse, R., Duvenaud, D., Xu, B., Martens, J., 2018. Stochastic gradient langevin dynamics that exploit neural network structure. In: International Conference on Learning Representations 2018 (Workshop).

Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning. pp. 807–814.

Newey, W.K., West, K.D., 1987. A simple positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix. Econometrica 55, 703–708.

Papadopoulos, H., Haralambous, H., 2011. Reliable prediction intervals with regression neural networks. Neural Netw. 24 (8), 842–851.

Pascanu, R., Montufar, G., Bengio, Y., 2013. On the number of response regions of deep feed forward networks with piece-wise linear activations. arXiv preprint arXiv:1312.6098.

Pasupa, K., Sunhem, W., 2016. A comparison between shallow and deep architecture classifiers on small dataset. In: 2016 8th International Conference on Information Technology and Electrical Engineering. ICITEE, IEEE, pp. 1–6.

Piramuthu, S., Ragavan, H., Shaw, M.J., 1998. Using feature construction to improve the performance of neural networks. Manage. Sci. 44 (3), 416–430.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., Ganguli, S., 2016. Exponential expressivity in deep neural networks through transient chaos. Adv. Neural Inf. Process. Syst. 29, 3360–3368.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Dickstein, J.S., 2017. On the expressive power of deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning, Vol. 70. pp. 2847–2854.

Rao, R.B., Fung, G., Rosales, R., 2008. On the dangers of cross-validation. An experimental evaluation. In: Proceedings of the 2008 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, pp. 588–596.

Reiners, M., Klamroth, K., Heldmann, F., Stiglmayr, M., 2022. Efficient and sparse neural networks by pruning weights in a multiobjective learning approach. Comput. Oper. Res. 141, 105676.

Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural Netw. 61, 85–117.

Schmidt-Hieber, J., 2017. Nonparametric regression using deep neural networks with ReLU activation function. arXiv preprint arXiv:1708.06633.

Serra, T., Ramalingam, S., 2019. Empirical bounds on linear regions of deep rectifier networks. Available in https://arxiv.org/abs/1810.03370.

Serra, T., Tjandraatmadja, C., Ramalingam, S., 2018. Bounding and counting linear regions of deep neural networks. In: ICML.

Shalev-Shwartz, S., Shamir, O., Shammah, S., 2017. Failures of gradient-based deep learning. In: International Conference on Machine Learning. PMLR, pp. 3067–3075.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.

Telgarsky, M., 2016. Benefits of depth in neural networks. arXiv preprint arXiv: 1602.04485.

Welling, M., Teh, Y.W., 2011. Bayesian learning via stochastic gradient Langevin dynamics. In: Proceedings of the 28th International Conference on Machine Learning. pp. 681–688.

Xiong, H., Huang, L., Yu, M., Liu, L., Zhu, F., Shao, L., 2020. On the number of linear regions of convolutional neural networks. In: International Conference on Machine Learning. PMLR, pp. 10514–10523.

Zaslavsky, T., 1975. Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. Mem. Amer. Math. Soc. 154, 1–95.

Zhou, Z.H., Wu, J.X., Jiang, Y., Chen, S.F., 2001. Genetic algorithm based selective neural network ensemble. In: IJCAI-01: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence. Seattle, Washington.