Learning sparse functions with neural networks

Alessio Cristofoletto, Vittorio Garretto

June 2024

Abstract

It is observed that neural networks outperform kernel machines on the task of learning sparse functions, i.e. functions of only k components of the d-dimensional input, such that $k \ll d$. We confirm this and show that, k being equal, a class of polynomial functions characterized by the *merged staircase property* (MSP) is significantly easier to learn than other polynomials. This happens because the MSP facilitates the feature learning process: SGD learns lower degree monomials first and then it is driven to learn the higher-degree terms. Also, the MSP drives the network to squash the weights relative to features that act as noise in the data, while non-MSP functions don't induce sufficient denoising. Finally, we explore a possible method to boost the NN's performance on non-MSP functions, exploiting the network's ability to perform feature selection.

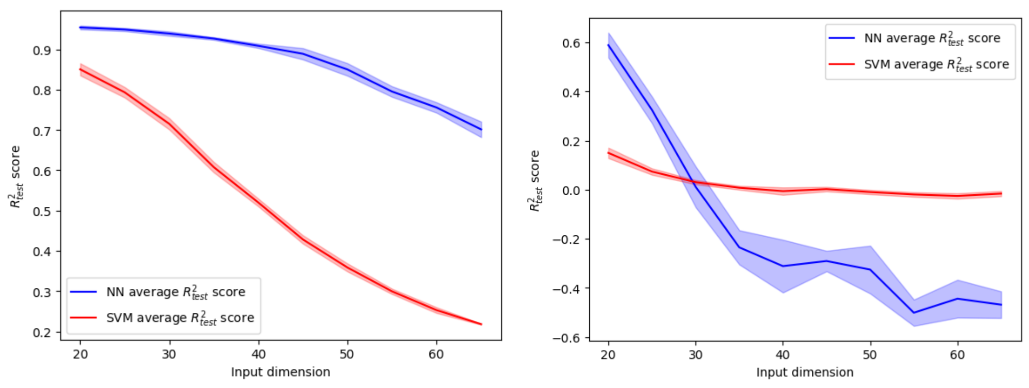
1 Introduction

The problem of learning sparse functions is of great interest to machine learning and deep learning practitioners: in many applications, the dataset is a collection of high-dimensional input vectors, but only some of the features are actually relevant, with the rest acting as noise in the data. This suggests that datapoints are actually embedded in a low-dimensional manifold in the input space and that, in order to learn from the training set and generalize well, a model must be able to build an internal representation of the data that identifies that manifold.
In Ref.[1], a comparison between a neural network (NN) and a support vector machine (SVM) shows that NNs clearly outperform SVMs on the task of learning sparse target functions from high-dimensional input data sampled from a Gaussian distribution. The author of Ref.[1] explains this difference in performance by examining the outer product of the first-layer weight matrix, $W^T W$. The diagonal elements of this matrix represent the squared norm of the weights associated with each feature in the input vector. For the neural network, the diagonal elements associated with the correct relevant features are far greater than the others, suggesting that the NN is capable of learning the right representation of the dataset.
However, it is possible to observe that some sparse polynomial functions are much harder to learn than others; for instance, a NN needs a larger sample size to learn $f = x_1x_2$, than it does to learn $f = x_1x_2 + x_1$ with equal precision. Ref.[2] characterizes with the *merged staircase property* (MSP) the class of sparse polynomials that are the easiest to learn for a neural network with 2 hidden layers. The authors prove this result for high-dimensional input vectors X with discrete features $x_i \in \{-1, 1\}$.
In this work, we confirm this result for continuous features, sampled from a Gaussian distribution $\mathcal{N}(0, 1)$, by comparing the sample sizes required to reach a fixed performance goal and their scaling with respect to input dimension d. We also try to gain insight into the different performance of the NN on MSP functions by examining the feature learning process of the neural network. In particular, we analyze the dynamics of the $W^T W$ matrix during training to try to understand if the MSP polynomials are easier to learn because the NN learns only the simpler, lower-degree terms instead of learning all of the relevant features, or if it's precisely those simpler terms that help it learn the full target function.
To conclude, we explore a method to improve the performance of the NN on a non-MSP function, with which we aim to exploit the network's feture learning ability to perform data augmentation in a way that does not require any prior knowledge of key properties of the data.

2 Results

NNs can outperform SVMs, but both fail on high-degree monomials

We show the difference in performance between neural networks and SVMs on a regression problem on two different sparse target functions $y = x_1x_2$ and $y = x_1x_2x_3$. The neural network outperforms the SVM on the first function and the R^2 score of both models decreases with the size d of the input vector, which is expected, since increasing d adds noise to the data. Unexpectedly, using a 3rd degree monomial worsens the performance, especially in the NN's case, which means that the network is overfitting.

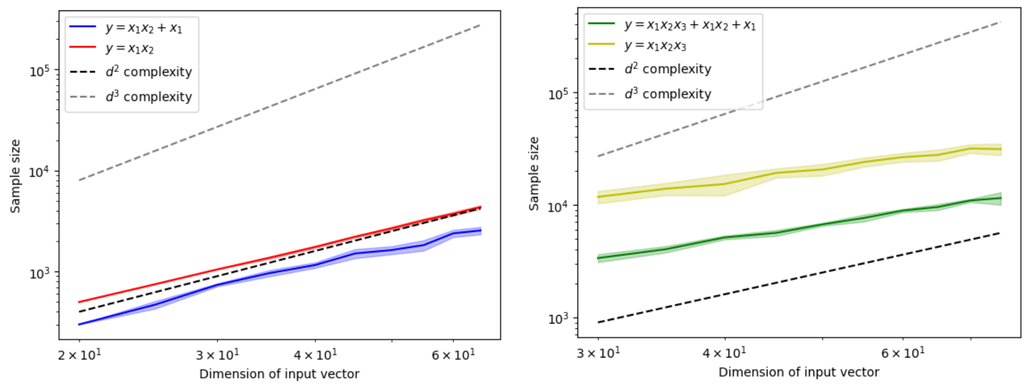


(a) (b)

Figure 1: **Neural network vs SVM.** (a) Plot of the R^2_{test} score on target function $y = x_1x_2$, with 2000 examples in training set. (b) Plot of the R^2_{test} score on target function $y = x_1x_2x_3$, with 2000 examples in training set. For both plots we show mean and standard deviation on 5 independent runs for each input dimension.

MSP functions are easier to learn

The drastic difference in performance displayed by the neural network on monomials only on degree apart can be explained by defining a class of functions, characterized by the *merged staircase property* (see Definition 3-4), which the authors of Ref.[2] have proven to be learnable with smaller sample sizes by NNs with 2 hidden layers trained with SGD. However, it is not obvious that this proof, made for boolean variables, holds true for continuous input data. To answer this question, in Fig.2 we show that, for 2nd and 3rd degree functions of Gaussian variables, in order to achieve a fixed performance target, the neural network always needs smaller sample sizes when learning a MSP target than it does learning its non-MSP counterpart. This comparison holds true independently of the input dimension, with overall sample size obviously growing with d. The plots in Fig.2 also confirm the intuition that higher-degree polynomials need larger sample sizes than lower-degree ones, to achieve the same test performance.



(a) (b)

Figure 2: **MSP functions require smaller sample sizes to be learnable.** (a) Plot of the minimum sample size required to reach a performance goal of $R^2_{test} \geq 0.90$ on 2nd degree polynomials. (b)Plot of the minimum sample size required to reach a performance goal of $R^2_{test} \geq 0.90$ on 3rd degree polynomials. For both plots we show mean and standard deviation on 5 independent runs for each input dimension in log-log scale.

Learning lower-degree monomials drives SGD to learn higher-degree monomials

To understand why MSP functions are easier to learn than their non-MSP counterparts, we examine the dynamics of the first-layer W^TW matrix during training on $y = x_1 + x_1x_2 + x_1x_2x_3$ (MSP) and $y = x_1x_2x_3$ on the same training set, recording the diagonal elements relative to the features x_1, x_2, x_3 for each training epoch. We observe that for the non-MSP function (Fig.3(b)) the network learns the key features contemporarily, increasing the weights associated to them during training in a uniform way. On the other hand, it appears that for the MSP function, the features relative to the lower-degree monomials are learned first and only then the network is able to identify the features that appear in the higher-degree terms. It is reasonable to assume that this is also due to the fact that lower degree monomials have larger magnitude than higher degree one, when centered and rescaled. On top of this, further studies (see Appendix, Fig.5) show that MSP also squashes the weights associated with the other features, thus inducing some denoising. This doesn't happen for the non-MSP function, even though in this case the network is still able to assign larger weights to the relevant features.

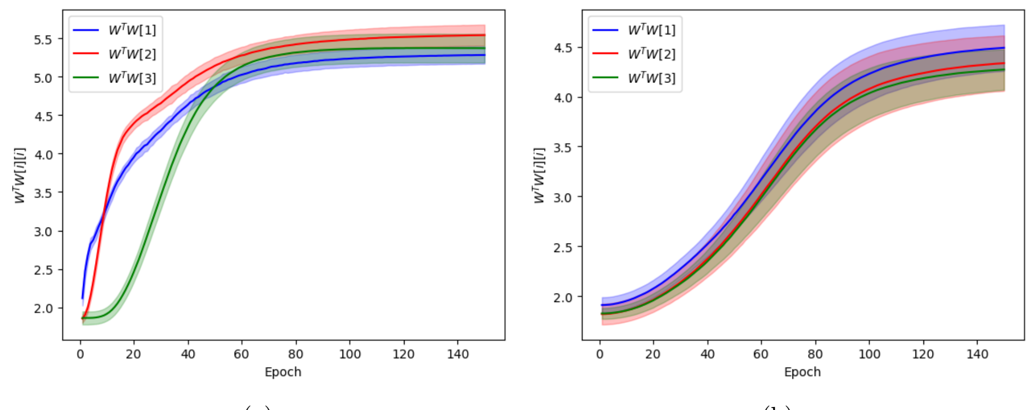


Figure 3: **Learning dynamics: MSP vs non-MSP.** (a) Plotting the evolution of the 3 diagonal elements of W^TW relative to the features x_1, x_2, x_3 during training on the MSP function $y = x_1 + x_1x_2 + x_1x_2x_3$. (b) Plotting the evolution of the 3 diagonal elements of W^TW relative to the features x_1, x_2, x_3 during training on the non-MSP function $y = x_1x_2x_3$. For both plots we show mean and standard deviation on 10 independent runs.

Data augmentation based on feature learning

In the previous section and in the Appendix, we presented evidence that, while the different learning dynamics between MSP and non-MSP functions induce more denoising in the former case than in the latter, in both instances the network is able to identify the correct features as the most relevant. This also means that the diagonal elements of the W^TW matrix with the smallest values after training are very likely to be associated with irrelevant features. We try to leverage this to perform data augmentation and boost the performance of the NN on non-MSP functions: after training the network on a random 2nd degree non-MSP polynomial, we augment the training set by adding a copy of the original dataset, with the features that the network has deemed irrelevant resampled. The results of this approach can be seen in Fig.4.
As expected from the results of previous sections, the test performance depends strongly on the sample size, so that even with an augmented dataset made of highly correlated examples, we are able to boost the performance significantly by using larger training sets. Obviously, this is not as effective as having a larger dataset of true data, but it is still remarkable that it is possible to improve the R^2_{test} in such an agnostic way, without having to access new data. In fact, this approach differs from other traditional data augmentation techniques because it doesn't require human knowledge of some key properties of the data.

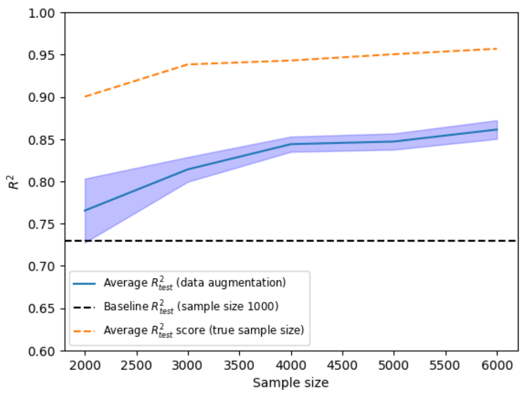


Figure 4: **Feature-learning-based data augmentation.** Comparing the NN's performance on $y = x_jx_k$ in three cases: on a training set of size 1000 (dashed black line), on the same dataset augmented to varying sizes (blue line) and on true datasets matching the size of the augmented ones (dashed orange line). For the augmented dataset, we plot mean and standard deviation on 5 independent runs.

3 Methods

Preliminary definitions

Here we provide formal definitions for the objects of our study, sparse functions and MSP functions, as defined in Ref.[2] .

Definition 1 (Sparse function)**.** Given an input vector $X \in \mathbb{R}^d$, a function $f : \mathbb{R}^k \to \mathbb{R}$ is a sparse function of X if its argument is a set of k components x_i of X, so that $k \ll d$.

Definition 2 (Fourier-Walsh decomposition)**.** Any function $f : \{-1, 1\}^k \to \mathbb{R}$ can be decomposed as $f(\mathbf{z}) = \sum_{S \subseteq [k]} \hat{f}(S)\chi_S(\mathbf{z})$, where $\chi_S(\mathbf{z}) := \prod_{i \in S} z_i$ is the basis and $\hat{f}(S) :=< f, \chi_S >$ are the Fourier coefficients.

Definition 3 (MSP set structure)**.** $\mathcal{S} := S_1, ..., S_m \subseteq 2^{[k]}$ is a *merged staircase property* set structure if the set can be ordered so that for each $i \in [m]$, $|S_i \backslash (S_i \cap (\cup_{i'<i} S_{i'})| \leq 1$

Definition 4 (MSP functions)**.** Let $\mathcal{S} \subset 2^{[k]}$ be the non-zero Fourier coefficients of f, i.e. $\hat{f}(S) \neq 0, iff S \in \mathcal{S}$. The function f satisfies the *merged staircase property if $\mathcal{S}$ is a MSP set structure*

Datasets, models and training

In this work, our datasets contain inputs $X^{(i)} \in \mathbb{R}^d$, with each of the d components is $x_i \sim \mathcal{N}(0, 1)$ and synthetic labels $y \in \mathbb{R}$. The data is pre-processed with a standard scaling procedure, as it is common practice. The process that generates the labels varies, as it is part of the subject of our study, as well as the size of the samples.
Two models are used in our work: support vector machines and neural networks. For the SVM, we use the "RBF" kernel, as in Ref.[1], and default parameters while, in order to align with the results of Ref.[2], we study neural networks with 2 hidden layers in the overparemeterized regime, with 500 units per layer. The choice to conduct our analysis in the overparameterized regime is also justified by the fact that this is the usual setting for most practical applications. With the intent to keep our analysis realistic, we depart from Ref.[2] in the choice of the activation function and decide to use the well-known and commonly used ReLU activation. Since the network is trained on a regression problem, we use a mean-squared-error loss function, with a L1 regularization added to prevent overfitting, since the network is overparameterized and the input data is mostly made of noise by design. Finally, we train the NN with stochastic gradient descent (SGD), to align with Ref.[2] and also because it is a very common choice among practitioners. We use a default learning rate $\eta = 0.01$.

Performance metrics and goals

In alignment with Ref.[1], we choose the R^2 score as a performance metric and when a fixed performance target is needed, we set it at $R^2_{test} = 0.9$.

Data augmentation based on feature learning

To perform data augmentation based on the neural network's ability to perform feature learning, we start with a set of 1000 examples $X^{(i)} \in \mathbb{R}^{40}$, with labels generated by a 2nd degree monomial, which is the simplest non-MSP function. The target monomial is defined as $y = x_j x_k$, with j, k drawn from a uniform discrete distribution $\mathcal{U}\{1, 40\}$ at the beginning of the experiment, to show that no prior knowledge of the relevant features is required. At first, we train a neural network (as defined in the previous sections) on this training set for 150 epochs, a value we choose empirically to allow the NN to achieve a negligible training error. After training, we get the diagonal elements of the first-layer $W^T W$ matrix and choose the 3 elements with the lowest values and the corresponding features in the input vectors $X^{(i)}$, which the NN has "labeled" as noise-like. We proceed to double the size of the training set by making a copy of it, resampling the 3 chosen features from a Gaussian distribution $\mathcal{N}(0, 1)$ for every example in the copy and appending it to the original dataset. Note that the number of features resampled/modified can be subject to optimization and the choice of using only 3 in our experiment is made to keep this number small relative to the input dimension, for the sake of generality. We then retrain the network on this augmented dataset and evaluate its performance on a test set of 500 examples, generated in the same way described for the training set. We repeat this experiment varying the size of the augmented dataset, to study its effectiveness in relation to the augmented sample size.

4 Conclusions

In this work we have shown that neural networks outperform SVMs on the task of learning sparse functions and this lead us to a more detailed study of the nature of these sparse function. In particular, we have confirmed that MSP functions are easier to learn than non-MSP ones, as they require smaller sample sizes to reach the same performance goal. The reason behind this is that SGD is driven to learn the higher degree terms in MSP functions by learning the lower ones first, which are lacking in non-MSP ones. However, in both cases, the neural network is capable of performing a certain degree of feature selection, which we exploit to perform data augmentation to boost the performance on non-MSP functions.

A possible line for future work is to apply this method for data augmentation on a real dataset, where there are no obvious properties to exploit to create new data, and to optimize it.

References

[1] M. Belkin. "Toward a Practical Theory of Deep Learning: Feature Learning in Deep Neural Networks and Backpropagation-free Algorithms that Learn Features". CLIMB Seminars, Berkeley, 2023

[2] E. Abbe, E. Boix-Adserà, T. Misiakiewicz. "The merged-staircase property: a necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks". *arXiv:2202.08658 [cs.LG]*, 2022

Appendix

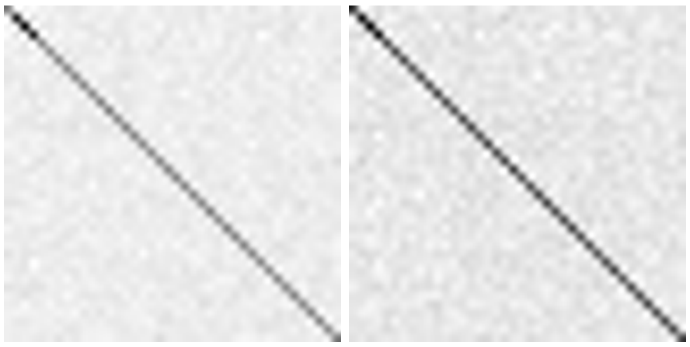


Figure 5: W^TW**: MSP vs non-MSP.** (a) W^TW after training on $y = x_1 + x_1x_2 + x_1x_2x_3$. The diagonal elements corresponding to the relevant features are clearly larger than the others. (b) W^TW after training on $y = x_1x_2x_3$. While the elements associated with the relevant features are still the largest, both diagonal and off-diagonal elements are way larger than in the MSP case, showing that in this case the network is not able to perform sufficient denoising. In both cases, the network the is trained on 3000 examples