CS4900: UGRC - I Report

An empirical analysis of feature learning in ReLU networks

Jan-May 2023

Balakrishnan A

CS20B012

Guide: Prof. Harish Guruprasad Ramaswamy

Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Contents

Abstract

The usage and literature on neural networks grow endlessly with time, but our understanding of them has not grown proportionately. In this work, we analyze the ability of neural networks to learn useful features/representations from data, which are not adequately learned by kernel machines. We shall do so by making comparisons between the two, with the help of the neural tangent kernel, drawing inspiration from datasets involving rich symmetry and/or simplifying transforms of high-dimensional features.

1 Introduction

Neural networks account for a large class of models widely used today for several tasks involving pattern recognition, ranging from image classification to object detection to natural language processing, including the ever-growing regime of generative AI. At the heart of a neural network are just simple but clever transformations stacked on top of each other and a widely-used gradient descent optimization. What, then, makes them such shining candidates for this large variety of tasks? Why not kernel machines, such as support vector machines and linear models (linear/logistic regression), which also involve such matrix transformations and the same, albeit simpler, gradient descent?

Understanding the answers to these questions could pave the way forward to better interpretability of neural networks. Uncovering flaws and positives from their way of learning features can provide paths to develop better neural networks, exploiting their strengths and attempting to overcome their flaws.

The general belief as to why neural networks/deep learners outperform traditional machine learning models, such as kernel machines, is that neural networks do not just work with the set of features given to them as part of the data; they uncover new useful features from data using their hidden layers, enriching their ability to learn the data. In contrast, kernel machines simply use the kernel features, which are decided before fitting the data. However, we are now left unable to answer whether kernel machines, with the right choice of kernel, can perform on par with neural networks. This dilemma is reasoned with using the neural tangent kernel [4], which approximates neural network learning by kernel machine learning using a special data-dependent kernel. Its introduction in the seminal work of Jacot et al. [4] has led to a lot of efforts in understanding how neural networks learn, by looking at it as a simpler problem of the learning of kernel machines. However, like all approximations, a few unrealistic/simplifying assumptions are involved, leaving the problem only partially answered.

The vast success of transfer learning, especially in the field of computer vision, suggests that the hidden representations learned by neural networks during training account for some common features we can expect across vastly different datasets belonging to the same class of problems. However, these features may be non-trivial and unlike the human-recognizable features one tends to think of, such as the presence of certain body parts. Our main focus in this work is to analyze feature learning in neural networks along with why these features are learned - do they agree with human-identifiable features or artificially injected features left purposefully for the model to find? We perform extensive empirical analysis to check whether the learning follows a trend that we may expect. Throughout our work, we compare and contrast them to kernel machines with the help of the neural tangent kernel.

2 Related works

There is no shortage of works on neural networks, detailing all aspects from generalization to comparisons with kernel machines and similar learners to interpretability and so on. A few papers in the kernel comparison regime are seminal to our work, especially those of Atanasov et al. [1] and Jacot et al. [4].

Jacot et al. [4] introduces the neural tangent kernel (NTK), a widely used tool for comparing neural networks and SVMs, that we also use heavily. This brings the literature a step closer to one of the ideas behind neural networks' success, the belief that they learn a kernel suitable for the data rather than using a fixed kernel that SVMs use. The NTK is defined as follows:

$$\mathbf{K}(t) = [K_{i,j}(t)] = [K(\mathbf{x}_i, \mathbf{x'}_j, t)],$$

$$K(\mathbf{x}, \mathbf{x'}, t) = \frac{\delta f(\mathbf{x}, t)}{\delta \boldsymbol{\theta}(t)} \cdot \frac{\delta f(\mathbf{x'}, t)}{\delta \boldsymbol{\theta}(t)}$$

i.e.,

$$\mathbf{K}(t) = \frac{\delta f(\mathbf{X}, t)}{\delta \boldsymbol{\theta}(t)} \times \left[\frac{\delta f(\mathbf{X'}, t)}{\delta \boldsymbol{\theta}(t)}\right]^{\mathrm{T}}$$

where,

t refers to the progress of training or the current epoch,

$\mathbf{X}, \mathbf{X'}$ are data matrices of which $\mathbf{x}, \mathbf{x'}$ are data vectors,

$f(\cdot, t)$ is the network function, the output of the neural network, and

$\boldsymbol{\theta}(t)$ represents the parameters of the neural network at instant t.

There is also a random functions approximation provided by the paper following the idea introduced by Rahimi and Recht [7], but we resort to using this closed-form kernel throughout our experiments. The authors perform a robust analysis, allowing for deep neural networks without restrictions of activation functions besides Lipschitzness and twice differentiability. There is an infinite-width limit assumption, but that is only so that the NTK remains constant throughout training.

However, a shortcoming of the paper is that only vanilla neural networks are studied, allowing for only weights, biases, and activation functions in the network. All other developed layers such as Dropout, Batch Normalization, Pooling, etc. are not involved and how to handle those is not completely clear. We restrict ourselves to analyses of vanilla neural networks.

Atanasov et al. [1] build on this by showing that a very large class of neural networks and data can be modeled by static data-dependent kernels using the NTK corresponding to the final parameters learned by the neural network, rather than limiting to infinite-width networks that have a static NTK throughout training. The paper shows that the NTK aligns its eigenvectors along the optimal linear function (a proof was shown only for linear networks, although observed empirically also for networks with non-linearities) and afterward grows only in scale, which is inconsequential for kernel regression. This *silent* alignment happens early, while the loss has not decreased appreciably.

Bounds for training duration corresponding to the depths and widths of networks were established by the paper, allowing for the trained neural network to be approximated by the final NTK. This is supported by a plot depicting correlations of the predictions of the trained network with the predictions of the NTK on a subset of the (0, 1)-classes of benchmark datasets such as MNIST, CIFAR-10, modeled in a regression setting. Both initial and final NTKs are used, to illustrate an alignment that happens after some amount of training, leading to a very strong correlation with the final NTK's predictions, whereas the correlation is much weaker with the initial NTK.

Other influential and guiding works are those of Daniely and Malach [3] and Shi et al. [8], both uncovering similar forms of data where neural networks outperform kernel machines. While the class of data distributions is highly restricted in [3], it is slightly relaxed in [8], allowing for a somewhat larger class of distributions.

Both papers prove significantly better performance of neural networks over kernel machines, demonstrating it with slightly modified MNIST data.

Another work that suggests different conditions where neural networks outperform kernel machines is that of Damian et al. [2]. The restriction on the data is now relaxed extensively, allowing for polynomially distributed data. Proofs of bounds on loss and training time are provided so that neural networks can significantly outperform kernel machines. However, a lot of other restrictions are imposed on the model and the learning algorithm, studying only 2-layer networks and an unusual training algorithm. Nonetheless, the possibility of extending these results to more general cases could make a strong case for the favourability of neural networks.

3 Our contributions

We analyze the ability of neural networks to learn features from data using two datasets - each feature-rich in their own way. Specifically, we observe the capacity of neural networks to:

1. detect symmetry from data that is rich in symmetry [Section 4]
2. learn hidden/abstract representations from data [Section 5]

We look at different conditions, such as different depths of the network and different variants of backpropagation, and observe how they affect the ability of the neural network to learn these special features. We also continually contrast them with kernel machines, especially their NTKs, in attempts to answer whether neural networks do outperform kernel machines.

Through this empirical analysis, we conclude that neural networks can sometimes detect symmetry or learn hidden representations, as long as the data is not too non-linear and when the network is not heavily underparametrized or overparametrized.

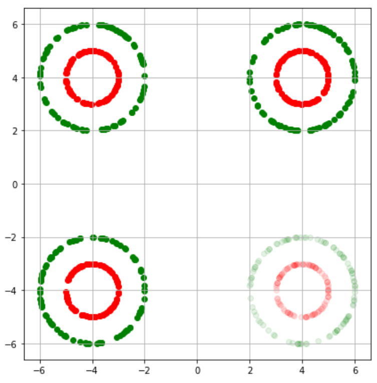
4 Learning symmetry

The next and widely-spanning experiment is an analysis of the ability of neural networks to detect symmetry in a highly-symmetrical dataset, with further horizon expansions to detect other key learning patterns and contrasts to SVMs.

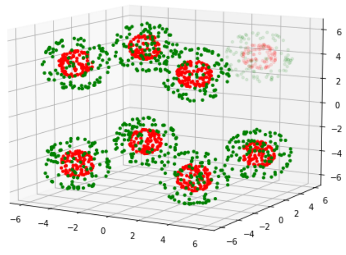
4.1 Problem setup

The data for the problem is 7-dimensional, i.e, $\mathbf{x} \in \mathbb{R}^7$. The true distribution of the data consists of 2 concentric spherical shells in each of the 128 orthants in 7-d space. For our experiment, we consider the shells to be centered at $(\pm 4, \pm 4, \pm 4, \pm 4, \pm 4, \pm 4, \pm 4)$ [with corresponding signs for each coordinate according to the orthant], with radii of the shells as 1 and 2. The shell with radius 1 corresponds to the *negative* datapoints, with class label 0, whereas the shell with radius 2 is labeled 1 (*positive*). The test data is simply all the orthants populated with 100 datapoints of each class.

For the training data, we randomly choose most of the 128 orthants to be densely populated, with the others being sparsely populated, of which some are even left empty. In an extreme case, let us consider one single randomly chosen orthant, which is left empty, with the remaining orthants



(a) Simplified data in 2 dimensions



(b) Simplified data in 3 dimensions

Figure 1: Representations of the dataset for fewer dimensions, i.e., 2 and 3. Red points belong to class 0, while green points belong to class 1. Faded points belong to the orthant (quadrant/octant respectively) absent in the training data

populated densely, i.e., with 100 datapoints of each class. An illustration of the data distribution for fewer dimensions, i.e., 2 and 3, is provided in Figure 1.

We wish to analyze the performances of neural networks and kernel machines (we consider both the NTK corresponding to the trained neural network as well as the generic RBF kernel), observing the ability of both to generalize well to the empty orthant, and whether they fail to perform well in the dense orthants in order to achieve this.

We also analyze the performance in the empty orthant, contrasting it to its neighboring orthants to observe if it is somehow utilizing their learning alone instead of learning the inherent symmetry in the data. We observe the radial decision boundary of sorts learned by the models, for a few randomly chosen orthants. We then observe the closest points for each of the datapoints in the empty test orthant, using two different feature embeddings, namely, the true data, and the final layer feature vector, which is finally used for classification.

The neural network we use has d hidden layers, and we observe performances for $d \in \{1, 2, 3, 4, 5\}$. Each hidden layer contains m neurons, which is determined by cross-validation. Optimization is done using the Adam [5] optimizer, with learning rate and weight decay determined by cross-validation, with binary cross-entropy as the loss function.

4.2 Observations

4.2.1 Generalization to the empty orthant

We observe that the prediction capacity of the neural network is quite consistent between different orthants for smaller values of d, such as $1, 2, 3$. The empty orthant achieves slightly lesser accuracy but not falling to the extent of random guessing most of the time. For larger d, i.e., $4, 5$, however, the excess parametrization in the network leads it to overfit to the training data, doing no better than random guessing in the empty orthant.

The results of the SVM, however, indicate a more drastic variation. We also perform cross-validation between different kernels, namely the linear kernel, the RBF, and the best network's NTK. The linear kernel is completely unable to learn the spherical data that is too complex for a hyperplane and is of no

Model	Accuracy	
	Empty	Remaining
Linear SVM	0.5 ± 0	0.5 ± 0
RBF SVM	0.5 ± 0	$\mathbf{1} \pm 0$
1-NN	0.741 ± 0.00384	0.9333 ± 0.002186
1-NTK	0.705 ± 0.003997	0.8757 ± 0.002891
2-NN	0.558 ± 0.004353	0.8874 ± 0.002771
2-NTK	0.54 ± 0.004368	0.8514 ± 0.003118
3-NN	$\mathbf{0.829} \pm 0.0033$	0.9773 ± 0.001304
3-NTK	0.702 ± 0.004009	0.73 ± 0.003891
4-NN	0.5 ± 0	0.8951 ± 0.002685
4-NTK	0.498 ± 0.004382	0.8487 ± 0.00314
5-NN	0.5 ± 0	0.9923 ± 0.0007651
5-NTK	0.5 ± 0	0.815 ± 0.003403

Table 1: Accuracies of different depth neural networks and different SVMs (linear, RBF, NTK for different depths) on the empty orthant and the remaining orthants. All entries denote a 95% confidence interval of the proportion of points correctly predicted. The prefix d before each NN/NTK denotes the number of hidden layers in the neural network

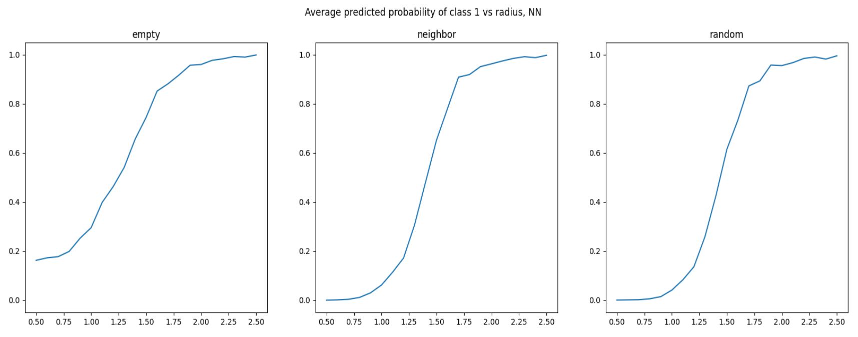
significant interest to us. We see that the RBF kernel outperforms the NTK, overfitting strongly (to 100% accuracy) to the training data (i.e., all the densely-populated orthants) and doing no better than random guessing on the empty orthant. This is also expected and not of much interest to us, so we resort to analyzing the NTK SVM instead hereafter. We see that the NTK uses the network's learning to not overfit to the training data and instead allow some learning in the empty orthant. However, its overall performance is now subpar to that of the network. In attempting to equalize performance on the empty orthant to that of the network, the score on the remaining orthants has decreased. If both classes of models do detect symmetry, the ability appears to be weaker for kernel machines as compared to neural networks. This is illustrated in Table 1.

4.2.2 Detection of symmetry in data

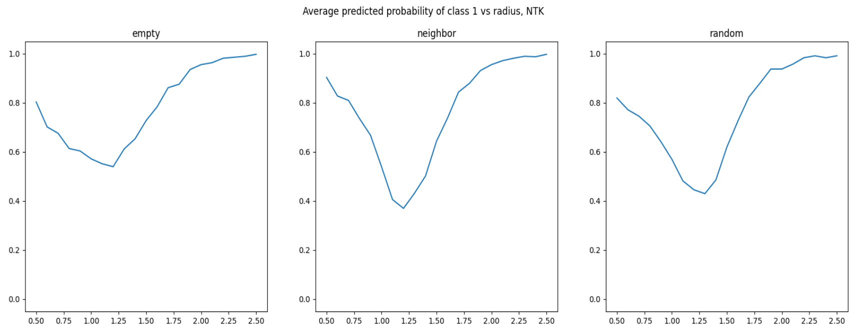
From the scores obtained in the various orthants, it seems as if the network is able to detect the symmetry in the data, perhaps by means of *folding* the data space to reduce all features to their absolute values. However, attaining such a global view is not easy, and requires strong corroboration from observations to support such a claim.

Hidden layers, d	Match fraction			
	Class 0		Class 1	
	Initial	Final	Initial	Final
1	0 ± 0	0 ± 0	1 ± 0	1 ± 0
2	0 ± 0	0.1348 ± 0.02969	1 ± 0	0.9386 ± 0.02069
3	0.03557 ± 0.01578	0.7957 ± 0.03513	0.9863 ± 0.009457	0.8771 ± 0.02852
4	0 ± 0	0.03358 ± 0.01532	0.9724 ± 0.01385	0.9863 ± 0.009457
5	0 ± 0	0 ± 0	1 ± 0	1 ± 0

Table 2: Match fractions for the output layer feature vector. Denotes the fraction of the test points (from the orthant absent in train data) having the closest train point (on the basis of Euclidean distance between output layer features of the datapoint) to be a point of the same class, with high values representative of folding. All entries denote a 95% confidence interval of the proportion of matching points



(a) Neural network



(b) Its corresponding NTK

Figure 2: The decision boundary for the neural network and its NTK when $d = 3$, plotted radially on 3 orthants - the empty one, a neighbor of it, and a random non-neighbor. 100 points were sampled for each radius from 0.5 to 2.5 (at increments of 0.1) and the average probability (for neural networks) or the estimated probability as the fraction of points predicted to be of class 1 (for NTK) was plotted

Support for this claim comes from the following empirical observations. The first and most significant is that the final layer features that we used to compare distances and determine the closest points seem to have evolved (more accurately, the parameters of the networks have evolved, transforming the final layer features) so that the closest points for the test datapoints are dominantly points from the same class (illustrated by Table 2). Initially, nearly all the points have the closest training point to be a point of class 1, the same result we get when we use Euclidean distance between the points as is. However, in some cases, after training, a large fraction of points have the closest point to be a point of the same class, indicating that they may have been clustered together into one single sphere, by means of folding. This is most starkly visible when $d = 3$. This effect is practically missing when d is 4 or 5, perhaps because of the increased capacity allowing it to fit each sphere individually. It is also very weak when d is 1 or 2, perhaps from a lack of capacity to achieve folding.

Another supporting observation, that neural networks are able to maintain a global view whereas kernel machines cannot, comes from the radial decision boundary (probability of predicting class 1 with radius) visualized for both cases. In the case that symmetry is detected by the model, we would expect a decision boundary closely agreeing to a sigmoid curve, as the hidden layers would be expected to determine the distance from the unsigned/folded centre, upon which a sigmoid transformation is applied to get the probability. As shown in Figure 2, there is a consistent and largely monotonic variation of the average probability of predicting class 1 with radius for neural networks, indicating that neural networks are able to identify the spheres (or a radial separation) within every orthant. On the other hand, kernel machines seem to come up with a decision boundary that separates the inner shell from the rest of the data, as the predicted probability (estimated for the NTK, as we used sklearn's LinearSVC [6], which does not provide a predict_proba method) is high except around the inner shell's radius of 1, implying a local outlook instead.

5 Learning hidden representations

5.1 Problem setup

We borrow some inspiration from Damian et al. [2], to understand the ability of neural networks to learn a hidden representation from polynomially-distributed data. The data is described as follows:

5.1.1 Details of the experiment

We choose a 100-dimensional distribution, $\mathbf{x} \in \mathbb{R}^{100}$. We use a scalar transformation of the data, $v = \mathbf{w} \cdot \mathbf{x}$. For our experiment, we choose a trivial transform, $\mathbf{w} = \mathbf{1} \Rightarrow v = \mathbf{1} \cdot \mathbf{x}$, i.e., $v = \sum_i x_i$. On top of this scalar transform, we add a polynomial transform[1]. This becomes our target to predict, i.e., $y = \text{poly}(\mathbf{w} \cdot \mathbf{x})$. We attempt different degrees of polynomials to compare the learning with increasing complexity. For the first case, we consider $\mathbf{x}$ to be drawn from a standard multivariate Gaussian distribution, i.e., $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{100})$, which is then scaled to prevent the polynomial target from becoming very large. Effectively, we have a covariance matrix of $\frac{1}{a} \cdot \mathbf{I}_{100}$, where a is determined by the maximum observed value of $\mathbf{w} \cdot \mathbf{x}$. In the second case, we choose our multivariate distribution to have a covariance proportional to $\mathbf{\Sigma} = \mathbf{w}\mathbf{w}^{\mathrm{T}} + \mathbf{I}$, i.e., $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{w}\mathbf{w}^{\mathrm{T}} + \mathbf{I}_{100})$. We term this as the data with correlation for simpler reference.

We restrict the neural network architecture to contain only 1 hidden layer with a variable number of neurons, which is tuned. This layer is followed by a ReLU non-linearity. The optimizer used is AdaDelta [9], as is used by Daniely and Malach [3], who also worked on the importance of knowing the hidden representations. The loss function used is L1 (MAE) loss, to model optimization similar to support vector regression, while the scoring metric used is L2 (MSE) loss.

We attempt multiple training modes, to observe the capacity of the network and its utilization of its first hidden layer to learn the reference vector. The training modes are described as follows:

- *reg*: Regular training procedure, where all the layers participate in backpropagation throughout training
- *stop_first*: The model is allowed to train regularly for 1 epoch, after which the hidden layer's (incoming) weights are frozen
- *first_last*: The model learns only the weights of its hidden layer for 1 epoch (the remaining being frozen), after which these weights are frozen and only the output layer's weights are learned
- *reinit*: The same as *stop_first*, but the weights of the output layer are reinitialized after the regular epoch of training

5.1.2 Expected results

We expect our neural network to learn better than SVMs, as it may be able to use the hidden layer to learn the hidden *reference vector*, i.e., $\mathbf{w}$ in its weights, converting it to a simpler problem of fitting

[1]Choosing random coefficients for the polynomial leads to almost linear polynomials with either very little or very large variations for the target. For this reason, we restrict v to be within a fixed range, say [-2, 2], and choose as many roots as the degree allows within this range, to build a polynomial with appropriate target variation and non-linearity

a polynomial now. Contrasting this to SVMs, for example, the RBF kernel SVM, which has to learn each of the polynomial coefficients individually, we expect a significant performance improvement with neural networks. This is also the result observed by Damian et al. [2], although for a more restricted dataset, weight initialization, and training algorithm.

5.2 Observations

5.2.1 Detecting hidden structure

We believe that the hidden layer of the network is key in detecting the hidden representation in the data. The weights of the hidden layer transform the input into new features, of which we would expect some to be v. This can be achieved when the hidden layer's weights evolve to become a constant scalar multiple of the reference vector, $\mathbf{1}$. In order to observe whether the hidden layer's weights are aligning with the reference vector, we plot the evolution of the cosine similarity of the hidden layer's incoming weights to $\mathbf{1}$ with training. With the help of this plot, we are able to verify that some fraction of the weights do evolve in direction to become the reference vector, implying that the hidden representation has been learned. Now, the network only needs to solve the problem of learning the polynomial, which we believe is a simpler task.

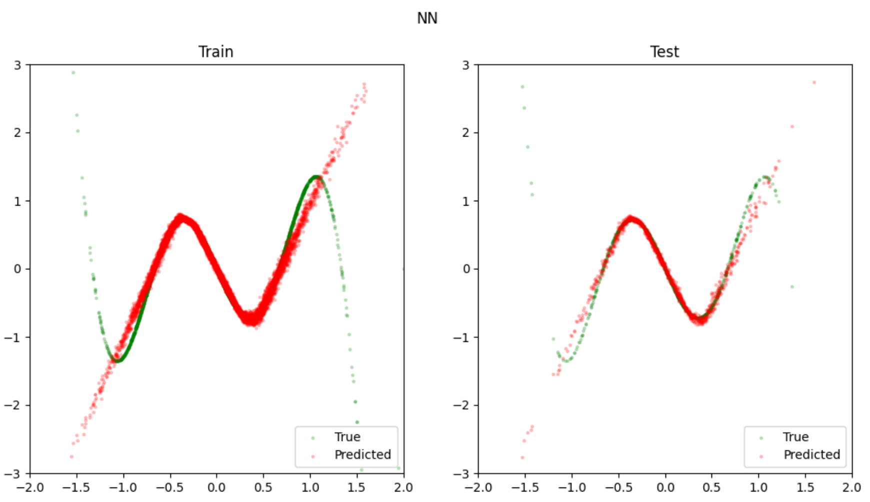
We perform the experiment for different degrees $d \in \{3, 4, 5, 6, 7, 8, 9\}$ of the polynomial. Our method of sampling the polynomial causes even degree polynomials to have an optimum at $v = 0$, leading to a slower alignment of the weights with the reference vector. Hence, while degrees 3, 5, and 7 are able to align some of their hidden layer's weights along $\mathbf{1}$ in 1-2 epochs, it typically takes 5-10 epochs to do the same for degrees 4 and 6. We observe that for degrees 8 and 9, the alignment never occurs, and the network is also unable to fit the data. The illustrations for the results overlap significantly to those for Section 5.2.3 and have been presented in Appendix B.

A similar result is obtained when the experiment is performed with $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{100} + \mathbf{1}_{100\times100})$. In this case, the neural network is able to fit polynomials of degree up to 11, and fails to do so for degrees 12 and above. However, we observe here that the RBF kernel is able to handle the simplified data better and consistently outperforms the NTK.

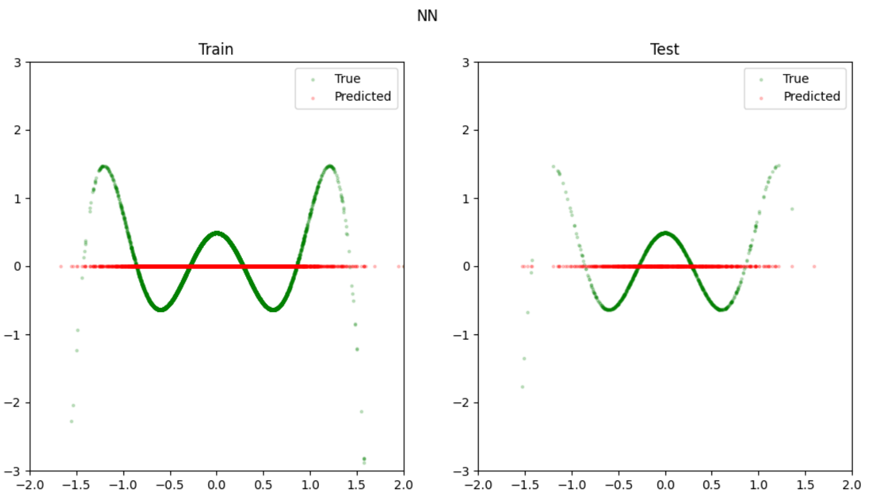
This assessment of the speed of alignment becomes crucial when we experiment with the other training modes mentioned. For odd degrees, as a partial alignment occurs within the first epoch, the network is able to make do with its output layer and achieve a good fit. However, as this alignment is very slow for even degrees, the weights of the hidden layer remain random and the output layer is unable to manage the complexity of the data, failing to fit the data well. We describe this in more detail for degrees 3 and 6 in Appendix A.

5.2.2 Benefit of knowing the hidden representation

Now that we have evidence that the neural network is able to detect the hidden representation in the data, we wish to explore whether knowing the hidden representation is of any use to the network or if it can only achieve the same results even when the transformed data is directly available to the network. To verify this, we work with a different set of degrees $d \in \{9, 10, \cdots, 19, 20\}$. Here, the input to the models is only 1-dimensional, i.e., $v \in \mathbb{R}$. The same dataset as above is used, only that instead of providing $\mathbf{x}$ to the networks, the transformed scalar representation, v is provided. We now observe that the model is able to fit (at least partially as a lower degree approximation) up to $d = 18$, beyond which it fails, only predicting a straight line for $d \in \{19, 20\}$. Hence, we establish the idea that



(a) Neural network fit when $d = 7$



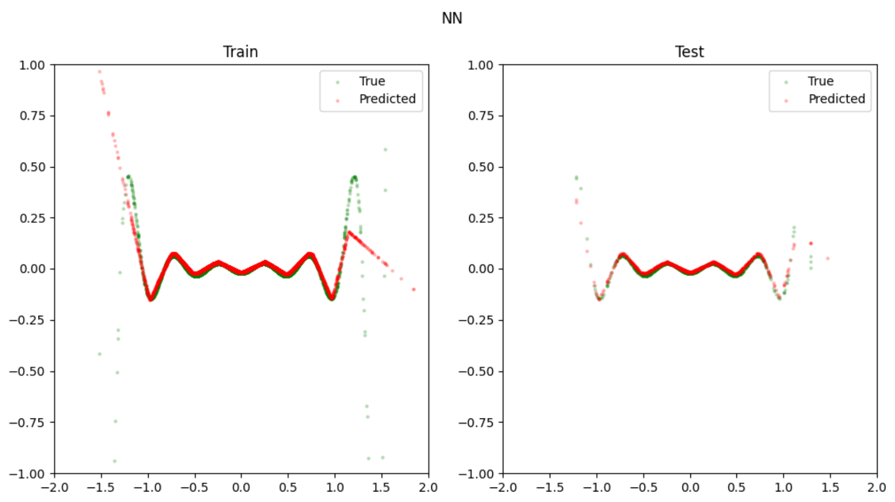
(b) Neural network fit when $d = 8$

Figure 3: Scatter plots of true values and predictions of the neural network when $d = 7$ and $d = 8$ on training and test data

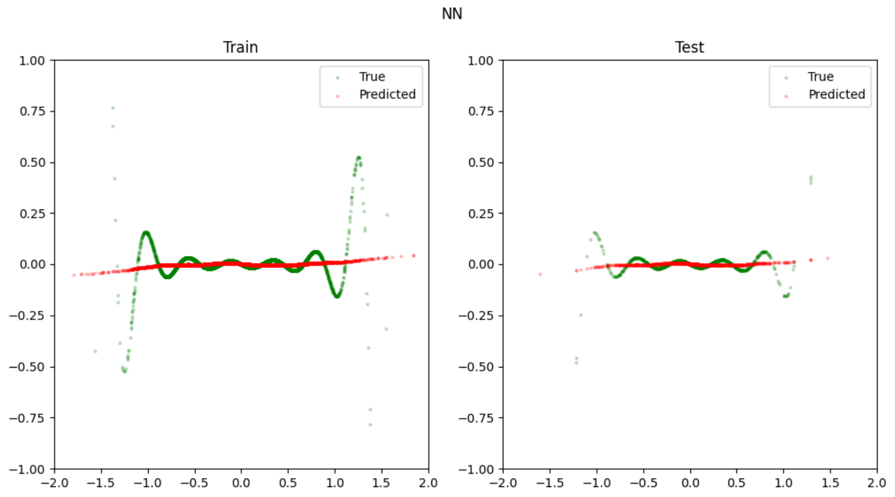
the hidden representation improves the ability of the network to learn the data. This is illustrated in Figure 4.

5.2.3 Illustrating the alignment

Since the idea of learning the hidden representation is so pervasive throughout this experiment, we attempt to link this learning of the representation (that we have been terming as *alignment* of the hidden layer weights to the reference vector) to the silent alignment effect described by Atanasov et al. [1]. We consider the cases when $d = 7, 8$ without correlation and $d = 11, 12$ with correlation, and analyze the cosine similarities, the correlation of predictions made by the network and the NTK, as well as the fitting capacity of the NTK for certain early epochs of training. Here, we deviate from our usual usage of the final NTK, and instead use NTKs corresponding to intermediate partially-trained neural networks.



(a) Neural network fit when $d = 18$ and v is directly given



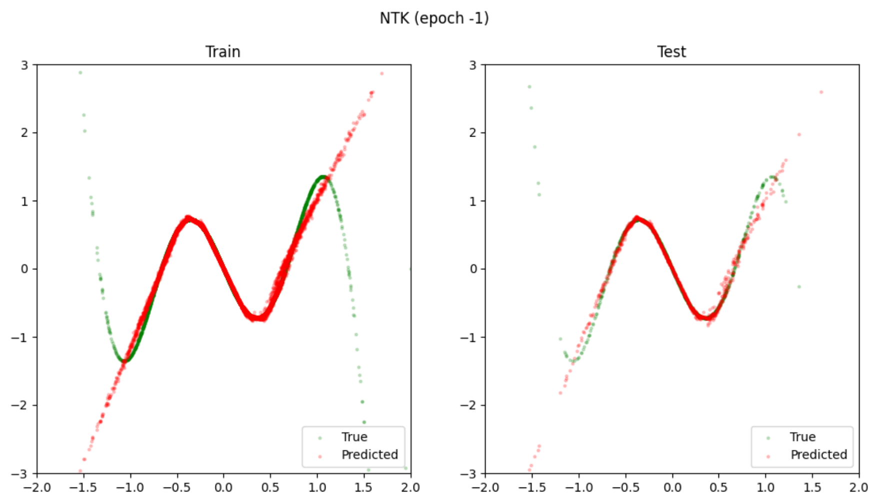
(b) Neural network fit when $d = 19$ and v is directly given

Figure 4: Scatter plots of true values and predictions of the neural network when $d = 18$ and $d = 19$ on training and test data when the hidden representation, v, is directly available

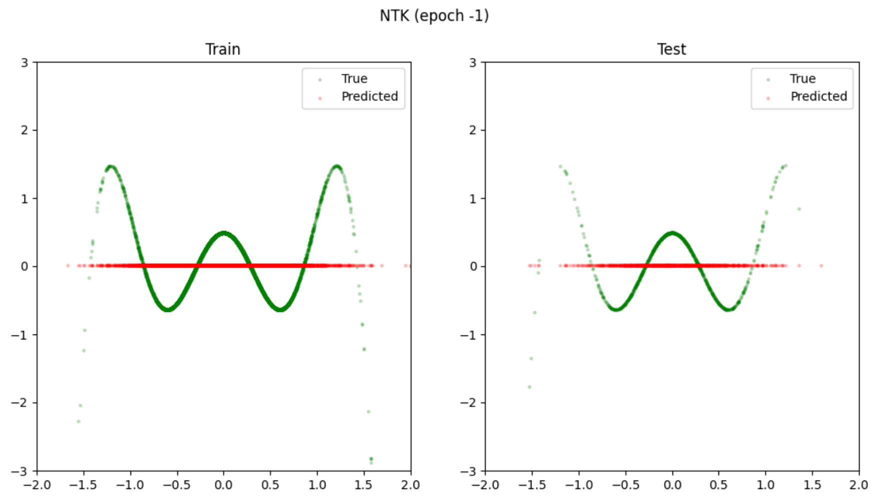
We try to associate jumps in cosine similarity to the silent alignment effect, when the NTK aligns its eigenvectors along the optimal function, for which we believe aligning the hidden layer weights along $\mathbf{1}$ is vital. Our reasoning holds water for $d = 7$, with a jump in correlation between the predictions, increase in fit closeness, and a jump in cosine similarities according at the same time during the 2nd epoch of training. $d = 11$ takes much longer for the alignment to occur, only during roughly the 16th epoch of training. There is a mostly consistent improvement in all 3 factors in this time and it is not very clear whether our alignment characterizes the silent alignment. More details on the experiment have been deferred to Appendix B as they require substantiation with several plots.

5.2.4 Do we need to learn the hidden representation?

We also observe the ability of the RBF kernel to learn the data, and we observe that though it usually performs well in fitting non-linear data, it is unable to handle so many dimensions, and does



(a) NTK fit when $d = 7$



(b) NTK fit when $d = 8$

Figure 5: Scatter plots of true values and predictions of the neural network's NTK when $d = 7$ and $d = 8$ on training and test data

no better than the NTK for most degrees. $d = 3$ appears to be a small enough degree that the RBF is able to learn it quite well, achieving an MSE of < 0.1, whereas for all values of $d \in \{4, 5, \cdots, 9\}$, it never achieves an MSE of < 0.1. Further, whenever the neural network is able to fit the data, the RBF is significantly outperformed by the NTK, i.e., for $d \in \{4, 5, 6, 7\}$. For $d \in \{8, 9\}$, both the RBF and the NTK perform very similarly in terms of validation MSE. However, a key difference between the two is that the RBF tends to overfit to the training data, achieving near 0 loss on the train data but only guessing a straight line on the validation and test data. On the other hand, the NTK underfits, guessing a constant value.

Now that we have established that the NTK performs superior to generic kernels, namely the RBF kernel, we seek to compare the performances of the neural network and its NTK. We observe that they both perform well when they are able to fit the data. However, our expectation that the neural network will outperform its NTK seems to be in vain, as the NTK happens to achieve marginally better losses than the network for all degrees. This is visible in some of the plots as the network performs a somewhat *fuzzier* fit (Figure 3) compared to a smooth fit by the NTK (Figure 5). Hence, our results do not agree with those presented by Damian et al. [2]. This may be due to the restrictions imposed by their work, such as a very uncommon label noise, a highly unconventional learning algorithm, along with a symmetric initialization. We attempt to replicate the paper by following their assumptions, but we observe no significant change. Our experiment seems to show, hence, that neural networks are able to detect hidden representations from data. However, whenever it is able to do so, it is not completely needed, as there is a kernel machine that is able to fit the data by itself (the NTK). Neural networks are able to exploit the hidden representation to perform better, as we show that knowing the hidden representations leads to more capacity to fit higher degrees of polynomials. However, learning the hidden representation in the process of fitting the data does not provide an edge to the network, since even kernel machines can fit the data without the hidden representation.

6 Reproducibility statement

All the experiments were run on a local machine with 16 GB of RAM and an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz with 6 cores. All code written to perform the experiments is publicly available at `https://github.com/Bala-A87/UGRC`

7 Acknowledgements

I would like to thank my guide, prof. Harish, for his guidance and support throughout the course of this work, and the faculty organizer, prof. Narayanaswamy, for the opportunity to perform and present my work.

References

[1] Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural networks as kernel learners: The silent alignment effect. *arXiv preprint arXiv:2111.00034*, 2021.

[2] Alexandru Damian, Jason Lee, and Mahdi Soltanolkotabi. Neural networks can learn representations with gradient descent. In *Conference on Learning Theory*, pages 5413–5452. PMLR, 2022.

[3] Amit Daniely and Eran Malach. Learning parities with neural networks. *Advances in Neural Information Processing Systems*, 33:20356–20365, 2020.

[4] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

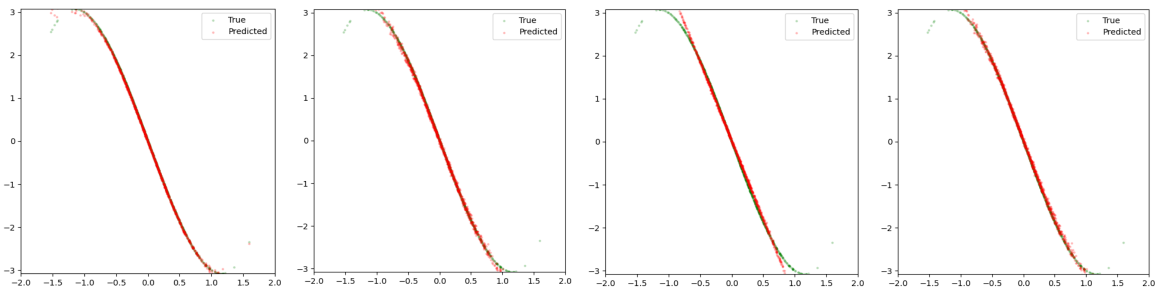
[8] Zhenmei Shi, Junyi Wei, and Yingyu Liang. A theoretical analysis on feature learning in neural networks: Emergence from inputs and advantage over fixed features. *arXiv preprint arXiv:2206.01717*, 2022.

[9] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

A Different training modes

As mentioned in Section 5.2.1, we attempt the experiment with different training modes to determine the effect of the hidden layer weights being partially aligned along **1**. There is a stark difference between degrees 3 and 6, due to the different speeds of alignment. Degree 3 successfully achieves a significant partial alignment in 1 epoch of training, after which the hidden layer weights are frozen. Degree 6 fails to do so, as it typically requires more epochs to learn the hidden representation. Hence, even though the network would learn the degree 6 polynomial under regular conditions, it fails to do when it can only use the hidden layer for a very restricted duration of training.

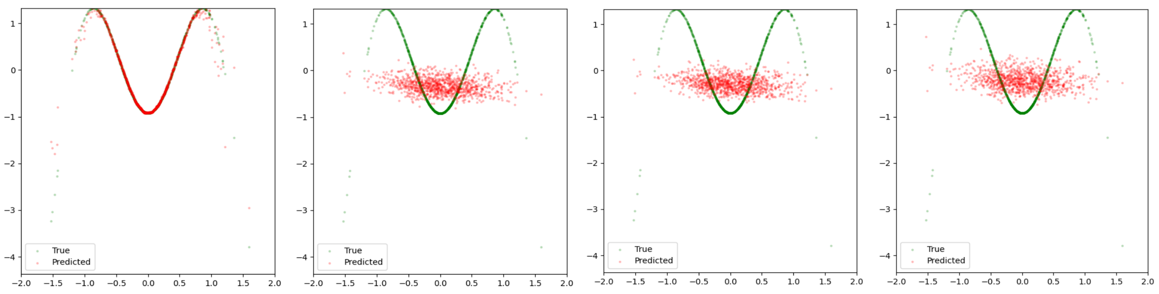
For degree 3 (Figure 6), *reg* mode of training the neural network outperforms the other modes slightly, but the overall results are not significantly affected for the different modes. The results of the network and its NTK are much the same again, as both have reduced capacity due to the freezing of the hidden layer. The NTK is still able to outperform the RBF.



(a) $d = 3$, *reg* (b) $d = 3$, *stop_first* (c) $d = 3$, *first_last* (d) $d = 3$, *reinit*

Figure 6: Scatter plots of true values and predictions of the neural network when $d = 3$ on test data for different training modes

For degree 6 (Figure 7), *reg* training vastly outperforms the rest, which struggle to even determine a vague shape of the polynomial. The RBF now outperforms the network and its NTK, but it only learns a rough direction of the polynomial and doesn't fit the data well.



(a) $d = 6$, *reg* (b) $d = 6$, *stop_first* (c) $d = 6$, *first_last* (d) $d = 6$, *reinit*

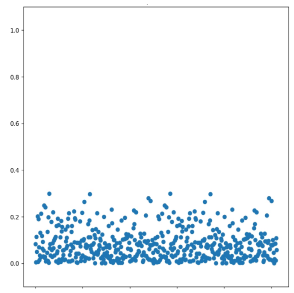
Figure 7: Scatter plots of true values and predictions of the neural network when $d = 6$ on test data for different training modes

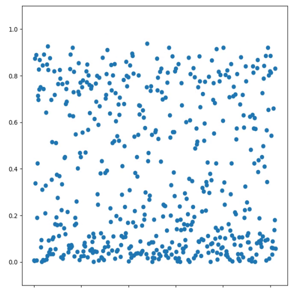
The usage of different training modes highlight the importance of knowing the hidden representations in a different way. The partial alignment achieved for $d = 3$ provides it the ability to use its output layer to fit the data, even though the network is now significantly underparametrized. The failure to align when $d = 6$ prevents the same from happening.

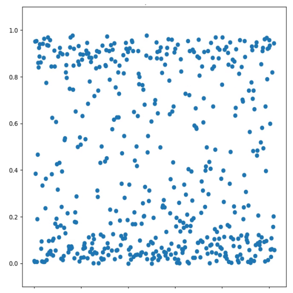
B Alignments

This section describes all the details corresponding to the alignment of the hidden layer weights along **1** (Section 5.2.1) and how we wish to compare it to the silent alignment affect [1] (Section 5.2.3).

Figures 8 and 9 illustrate the different speeds of alignment of the network's weights when $d = 3$ and $d = 6$ respectively. A large number of weights attain a cosine similarity of close to 1 within one epoch when $d = 3$, but there is hardly any alignment within one epoch when $d = 6$.


(a) Before training


(b) After 1 epoch


(c) After 5 epochs

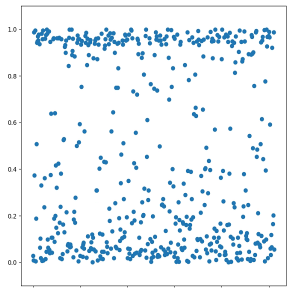
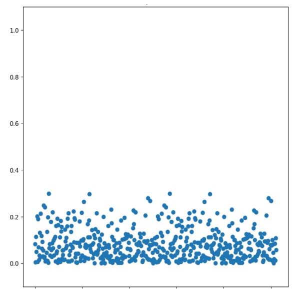
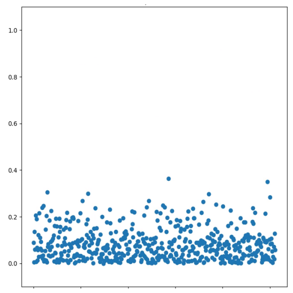
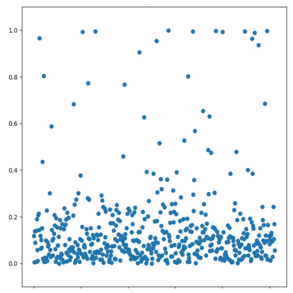

(d) After training

Figure 8: Scatter plots of cosine similarities of hidden layer weights and **1** at different epochs of training, when $d = 3$


(a) Before training


(b) After 1 epoch


(c) After 5 epochs

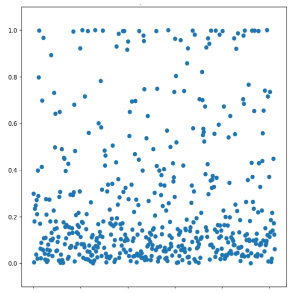

(d) After training

Figure 9: Scatter plots of cosine similarities of hidden layer weights and **1** at different epochs of training, when $d = 6$

Next, we analyze for multiple epochs the cosine similarities as well as the correlations between the predictions of a neural network (final predictions, obtained at the end of training) and its NTK at that epoch. We also plot the predictions of the NTK alongside the scatter plot of true values to observe the goodness of fit. We perform this experiment for $d = 7, 8$ without correlation, for the highest degree the network learns and the least degree it doesn't, and for $d = 11, 12$ with correlation. We wish to observe whether alignments of the weights along **1** indicates a shift in the NTK regime, characterized as the *silent alignment effect* by Atanasov et al. [1].

The results are summarized as follows, substantiated by several plots:

- For $d = 7$ (Figures 10, 11, 12), we observe a strong alignment during the 2nd epoch. A large fraction of the hidden layer weights align along **1**, and it is also marked by a significant rise in the correlation between the mentioned predictions and an improvement in the fit goodness

- For $d = 8$ (Figure 13), the weights of the hidden layer are hardly updated throughout training and there is no alignment to be observed

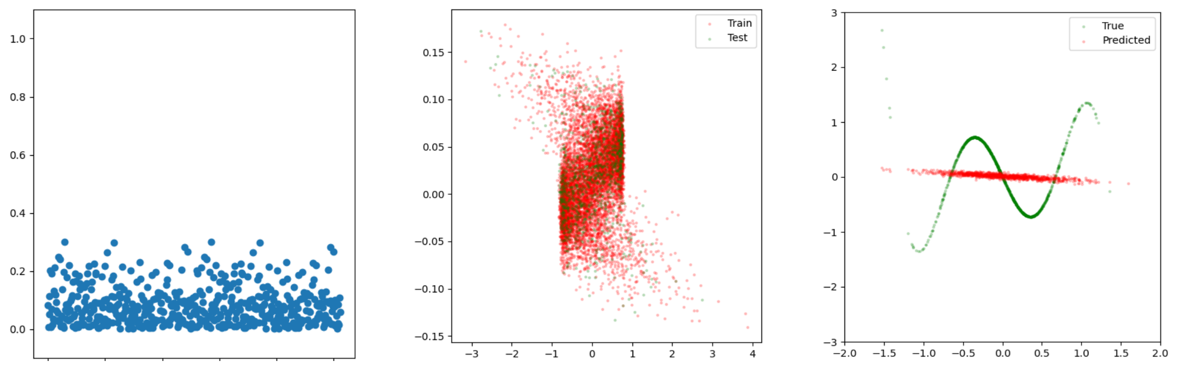
- For $d = 11$ with correlation (Figures 14, 15, 16), the effect is not as clear. The rise in cosine similarities is quite slow and mostly consistent. There seems to be a jump at the 16th epoch, with some weights attaining cosine similarities of greater than 0.5, but there is no marked improvement in the correlation or the fit goodness

- For $d = 12$ with correlation (Figures 17, 18, 19), there is an immediate rise in cosine similarities in the first epoch, followed by mostly stationary weights. This epoch sees an improvement in correlation (now both the network and the NTK predict a constant value). Over time, however, this correlation weakens and the NTK attempts to fit the polynomial, without much success

From these results, it appears as if our comparison of the alignment of the weights and the silent alignment holds water when the data $\mathbf{x}$ has its features drawn independently, and it seems to break down or at least becomes inconclusive when they are correlated in the way we have chosen.

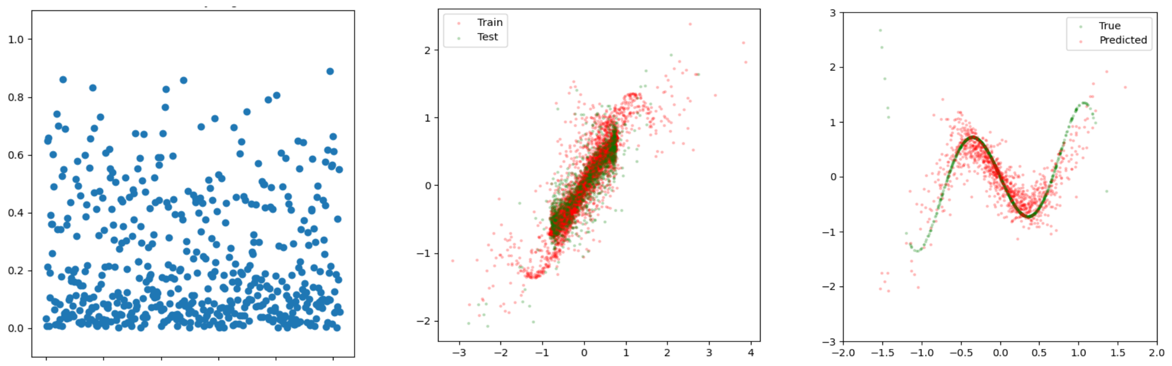
B.1 Illustrations

All of the following figures (each having 3 subfigures (a), (b), and (c)) describe the alignment characteristics for different degrees at different points of training. Subfigure (a) contains a scatter plot of cosine similarities between the hidden layer weights and $\mathbf{1}$. Subfigure (b) plots the predictions of the trained neural network on the horizontal axis and the predictions of the NTK corresponding to the current state of training of the neural network along the vertical axis. Subfigure (c) plots the predictions made by the NTK on the test data alongside the true values of the polynomial to illustrate the goodness of fit.



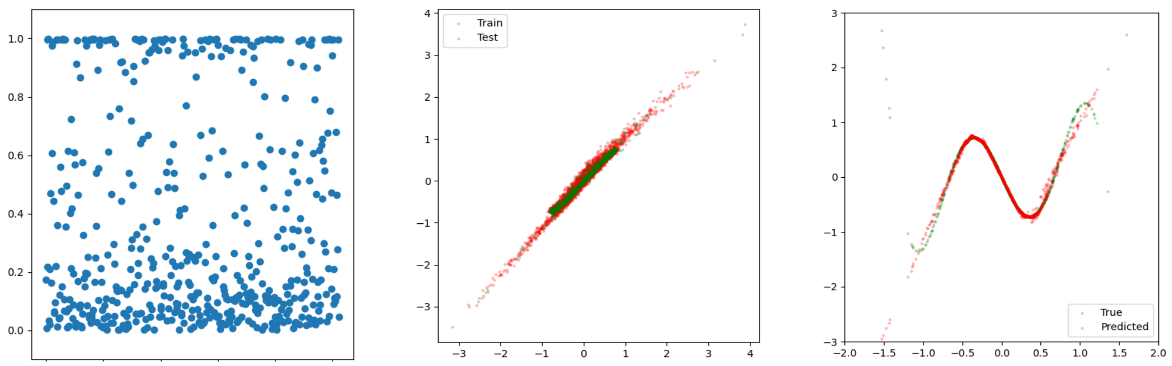
(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 10: Alignment characteristics for $d = 7$ before training



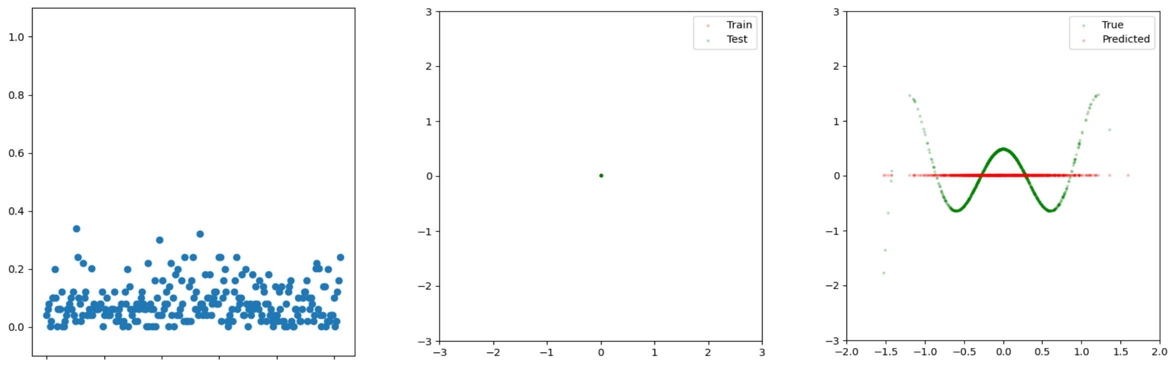
(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 11: Alignment characteristics for $d = 7$ after 2 epochs of training



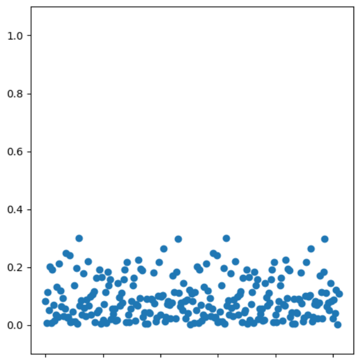
(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 12: Alignment characteristics for $d = 7$ at the end of training

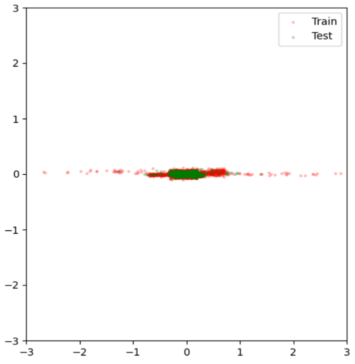


(a) Cosine similarities (b) Correlations (c) Goodness of fit

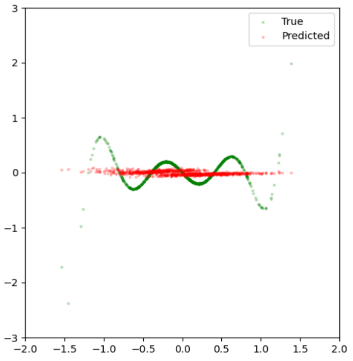
Figure 13: Alignment characteristics for $d = 8$ at the end of training



(a) Cosine similarities

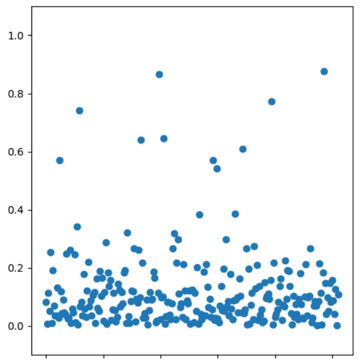


(b) Correlations

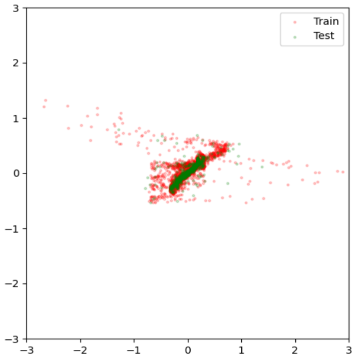


(c) Goodness of fit

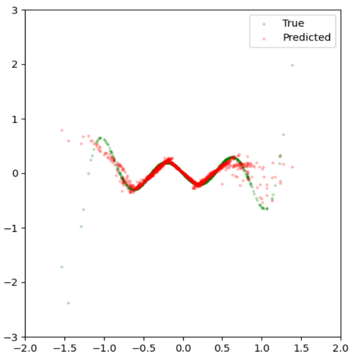
Figure 14: Alignment characteristics for $d = 11$ with correlation, before training



(a) Cosine similarities

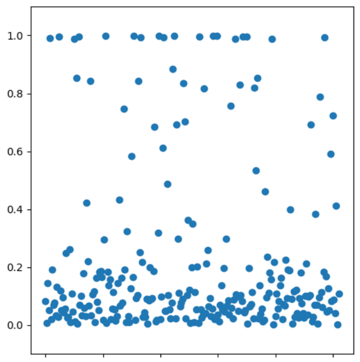


(b) Correlations

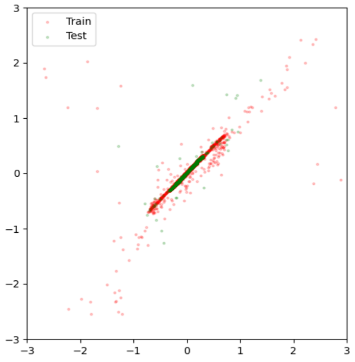


(c) Goodness of fit

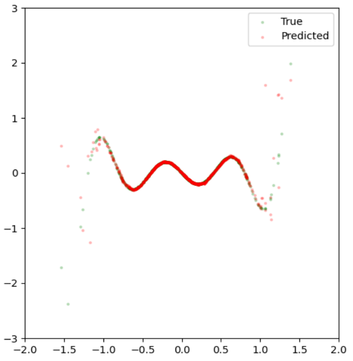
Figure 15: Alignment characteristics for $d = 11$ with correlation, after 16 epochs of training



(a) Cosine similarities

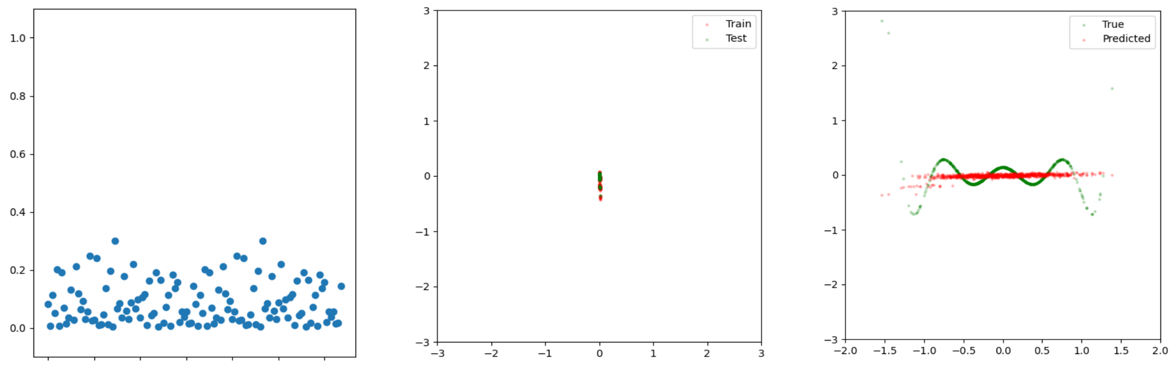


(b) Correlations



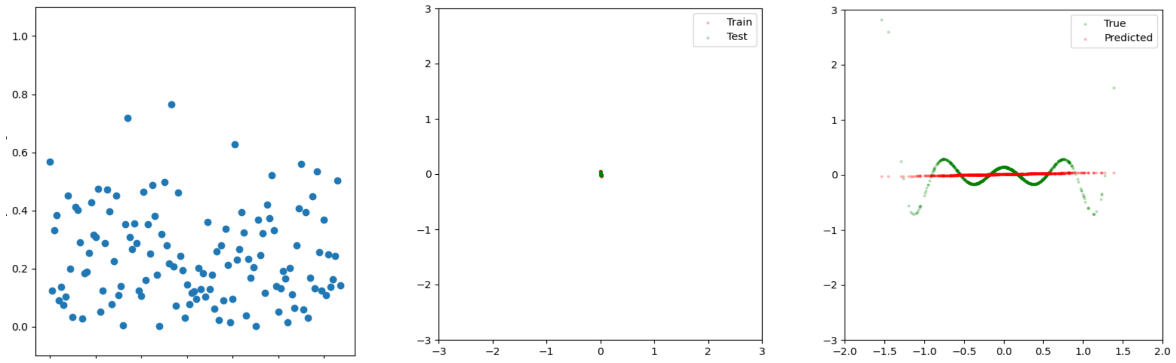
(c) Goodness of fit

Figure 16: Alignment characteristics for $d = 11$ with correlation, at the end of training



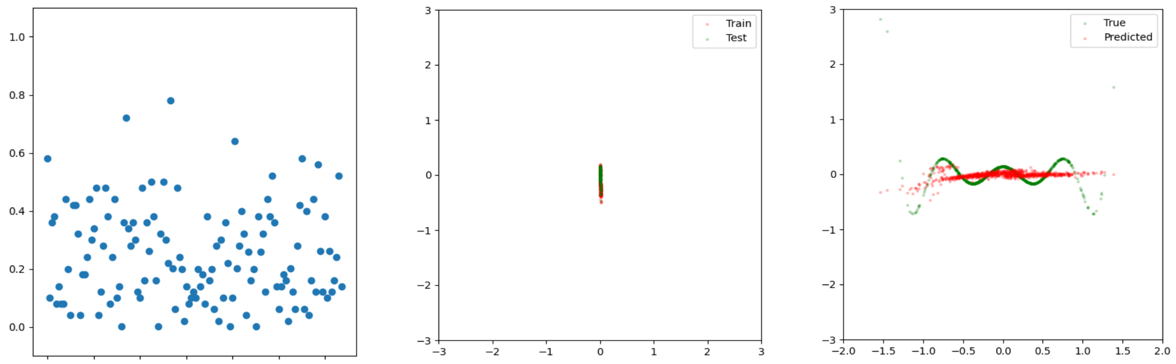
(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 17: Alignment characteristics for $d = 12$ with correlation, before training



(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 18: Alignment characteristics for $d = 12$ with correlation, after 1 epoch of training



(a) Cosine similarities (b) Correlations (c) Goodness of fit

Figure 19: Alignment characteristics for $d = 12$ with correlation, at the end of training