The Neural Tangent Kernel

Axel Orrhede
UC San Diego
San Diego, CA 92121
aorrhede@ucsd.edu

Abstract

This paper investigates the connection between neural networks and classical kernel methods through the Neural Tangent Kernel (NTK). Focusing on a simple neural network with one hidden layer of 10,000 units trained on a toy dataset of four one-dimensional datapoints, we derive both empirical and analytical expressions for the NTK. It is then shown that, for wide enough networks, the NTK remains effectively constant during training, allowing the network to be modeled as a kernel machine. Comparisons between the kernel-based predictions and the trained network outputs show their similarities and provide insights into the training behavior of neural networks. Finally, it is concluded that the NTK worked well for analyzing wide networks with one hidden layer, but this type of kernel analysis might not be useful for all neural networks.

1 Introduction

Deep neural networks have shown remarkable performance in tasks ranging from image recognition to scientific computing. However, understanding why these models train effectively despite the non-convexity and complexity of their optimization is still perplexing. Although empirical advances have propelled DNNs to the state-of-the-art in many machine learning tasks, their theoretical foundation remains largely undiscovered. The Neural Tangent Kernel (NTK) bridges deep learning and classical kernel methods, giving us insight into the training dynamics of overparameterized networks.

2 The neural network

This paper will analyze the curve fitting of a neural network onto $N = 4$ one-dimensional datapoints $\mathbf{x}$ with associated output $\mathbf{y}$. For this we will define a simple network with one hidden layer with $D = 10000$ units:

$$f(x, W) = \sum_{d=1}^{D} \frac{\mathbf{w}_d^1 \cdot a(\mathbf{w}_d^0 x + \mathbf{b}^0)}{\sqrt{D}} + b^1 \tag{1}$$

Where W contains the weights $\mathbf{w}^0$ and biases $\mathbf{b}^0$ of the first layer as well as the weights $\mathbf{w}^1$ and bias b^1 of the second layer. The division by $\sqrt{D}$ is used to rescale the outputs to avoid divergence in the infinite wide case while still allowing us to initialize all the weights with a $\mathcal{N}(0, 1)$ gaussian. The biases are initialized with value 0 and the activation function $a(z)$ is chosen to be the ReLU function ($max(z, 0)$).

The random initialization leaves us with the following problem.

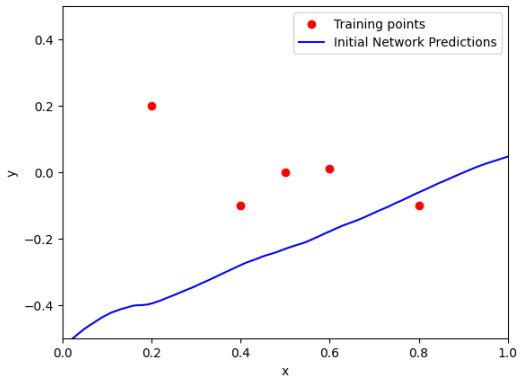

Figure 1: Data points for training the network and a initial prediction for all $x \in [0, 1]$ by the untrained network

Using a loss function over datapoints in the training set $x_i \in \mathbf{x}$ defined as:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} l_i, \quad l_i = \frac{(f(x_i, W) - y_i)^2}{2} \tag{2}$$

Allows us to iteratively subtract the negative gradient of the loss function with respect to the parameters in order to refine our model.

$$W^{i+1} = W^i - \alpha \frac{d\mathcal{L}}{dW} \tag{3}$$

Where $\alpha = 0.1$ gives us the following training characteristics.

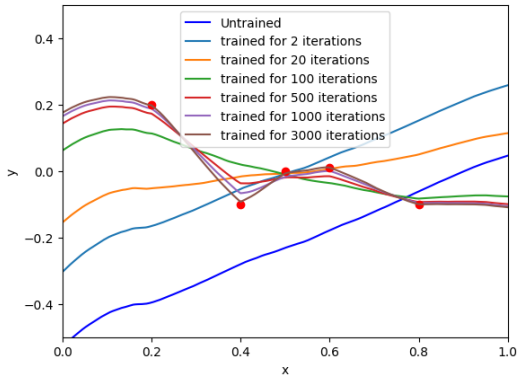

Figure 2: Training the neural network from equation 1
for 3000 iterations with gradient descent over all the datapoints and evaluating the network outputs for all $x \in [0, 1]$ after 2, 20, 100, 500, 1000 and 3000 iterations.

This model seemed to work just fine for the task at hand, the model appears to fit the datapoints nearly perfectly while outputting reasonable values when the input assumes values outside of the training dataset. In fact, the lines connecting the datapoints appear to get straighter as training progresses, suggesting that the network starts "interpolating" for inputs in this region. This is the interpolating regime described in Belkin [2021].

3 The neural tangent kernel

In order to analyze training in this regime, we must perform some more mathematical derivations. We begin by assuming that the step size α is small enough so that we take near-infinitesimal steps,

we can then "derive" the update rule from equation 3 with respect to time, giving us:

$$\frac{dW}{dt} = -\frac{d\mathcal{L}}{dW} = -\frac{1}{N}\sum_{i=1}^{N}\frac{dl_i}{df_i}\frac{df_i}{dW} \tag{4}$$

When $f_i = f(x_i, W)$. This allows us to study how the outputs change over time for datapoint j.

$$\frac{df_j}{dW}\frac{dW}{dt}^T = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial f_i}\frac{df_i}{dW}\frac{df_j}{dW}^T \tag{5}$$

The neural tangent kernel, introduced by Jacot et al. [2020] is found in the expression above, and the entries of the kernel matrix are defined by:

$$\mathbf{K}_{i,j} = \sum_{p=1}^{P}\frac{\partial f_i}{\partial W_p}\frac{\partial f_j}{\partial W_p}^T \tag{6}$$

Where we are summing over all the parameters $\mathbf{W}_p \in \mathbf{W}$. This is the same calculation as multiplying the Jacobian of f with respect to $\mathbf{W}$ with its transpose. This kernel can be used to find weights $\mathbf{v}$ for a kernel machine and make predictions over testing data $f_K(x)$.

$$\mathbf{v} = (\mathbf{K}(\mathbf{x}, \mathbf{x}))^{-1}\mathbf{y} \tag{7}$$

$$f_K(x) = \mathbf{K}(x, \mathbf{x})\mathbf{v} \tag{8}$$

Where $\mathbf{x}$ and $\mathbf{y}$ denotes the values in the training set while $x \in [0, 1]$ are the test points. $\mathbf{K}$ is the matrix computed according to equation 6.

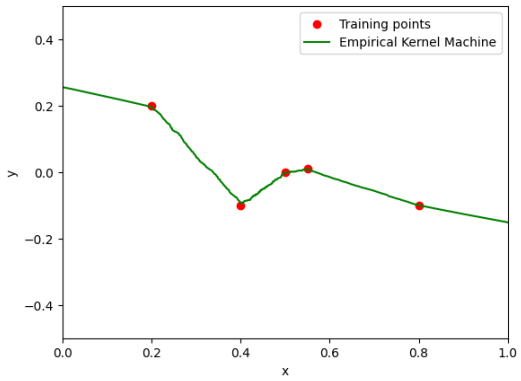

Figure 3: Training datapoints for a neural tangent kernel machine with kernel from equation 6 evaluated at all $x \in [0, 1]$

3.1 Analytical neural tangent kernel

On its own, this kernel does not help us analyze the training, since it has to be recalculated for every step. We shall therefor try to find a situation where the kernel is constant, and analyze its properties from there. We begin by expanding the calculation of the kernel matrix for the different parameters.

$$K_{i,j} = \sum_{d=1}^{D}\left(\frac{\partial f_i}{\partial \mathbf{w}_d^0}\frac{\partial f_j}{\partial \mathbf{w}_d^0}^T + \frac{\partial f_i}{\partial \mathbf{w}_d^1}\frac{\partial f_j}{\partial \mathbf{w}_d^1}^T + \frac{\partial f_i}{\partial \mathbf{b}_d^0}\frac{\partial f_j}{\partial \mathbf{b}_d^0}^T\right) + \frac{\partial f_i}{\partial b_1}\frac{\partial f_j}{\partial b_1}^T \tag{9}$$

We then calculate the derivatives from equation 1 using the chain rule.

$$\frac{\partial f_i}{\partial \mathbf{w}^0} = \frac{\mathbf{w}^1}{\sqrt{D}}a'(\mathbf{w}^0 x_i + \mathbf{b}^0)x_i \tag{10}$$

where:

$$a'(z) = \begin{cases} 0 & \text{if } z \leq 0, \\ 1 & \text{if } z > 0 \end{cases} \tag{11}$$

And:

$$\frac{\partial f_i}{\partial \mathbf{w}^1} = \frac{a(\mathbf{w}^0 x_i)}{\sqrt{D}} \tag{12}$$

$$\frac{\partial f_i}{\partial \mathbf{b}^0} = \frac{\mathbf{w}^1}{\sqrt{D}} a'(\mathbf{w}^0 x_i + \mathbf{b}^0) \tag{13}$$

$$\frac{\partial f_i}{\partial b_1} = 1 \tag{14}$$

Giving us:

$$\mathbf{K}_{i,j} = \sum_{d=1}^{D} \frac{\mathbf{w}^1 \mathbf{w}^{1T}}{D} a'(\mathbf{w}^0 x_i + \mathbf{b}^0) a'(\mathbf{w}^0 x_j + \mathbf{b}^0) x_i x_j + \frac{a(\mathbf{w}^0 x_i) a(\mathbf{w}^0 x_j)}{D} + \frac{\mathbf{w}^1 \mathbf{w}^{1T}}{D} a'(\mathbf{w}^0 x_i + \mathbf{b}^0) a'(\mathbf{w}^0 x_j + \mathbf{b}^0) + \frac{1}{D} \tag{15}$$

When $D \to \infty$, this sum can be calculated as an expectation.

$$\mathbf{K}_{i,j} = \mathbb{E}(\mathbf{w}^1 \mathbf{w}^{1T} a'(\mathbf{w}^0 x_i + \mathbf{b}^0) a'(\mathbf{w}^0 x_j + \mathbf{b}^0) x_i x_j + a(\mathbf{w}^0 x_i) a(\mathbf{w}^0 x_j) + \mathbf{w}^1 \mathbf{w}^{1T} a'(\mathbf{w}^0 x_i + \mathbf{b}^0) a'(\mathbf{w}^0 x_j + \mathbf{b}^0) + 1) \tag{16}$$

From here, we assume that the outputs of the first layers ($[\mathbf{w}^0 x_i + \mathbf{b}^0, \mathbf{w}^0 x_j + \mathbf{b}^0]$) are mean 0 and with the following covariance matrix to each other:

$$\Sigma = \begin{bmatrix} x_i^2 & x_i x_j \\ x_j x_i & x_j^2 \end{bmatrix} \tag{17}$$

Because of how the weights and biases are initialized. Prince [2022] Then states that it can be shown from here that:

$$E(a(\mathbf{w}^0 x_i + \mathbf{b}^0) \cdot a(\mathbf{w}^0 x_j + \mathbf{b}^0)) = |x_i||x_j| \cdot \frac{\cos(\theta) \cdot (\pi - \theta) + \sin(\theta)}{2\pi} \tag{18}$$

$$E(a'(\mathbf{w}^0 x_i + \mathbf{b}^0) \cdot a'(\mathbf{w}^0 x_j + \mathbf{b}^0)) = \frac{\pi - \theta}{2\pi} \tag{19}$$

Where:

$$\theta = \arccos\left(\frac{x_j x_i}{x_i \cdot x_j}\right). \tag{20}$$

This gives us an analytical expression for the kernel.

$$K_{i,j} = \frac{x_i x_j + 1}{2\pi}(\pi - \theta) + \frac{|x_i||x_j|}{2\pi}((\pi - \theta)cos(\theta) + sin(\theta)) + 1 \tag{21}$$

This analytical expression for the kernel will yield us a kernel machine with equations 7 and 8.

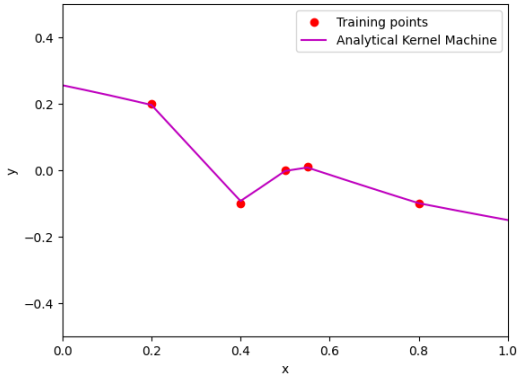

Figure 4: Training datapoints for a neural tangent kernel machine with kernel from equation 21 evaluated at all $x \in [0, 1]$

Plotting the kernel machines against the trained neural network gives us figure 5.

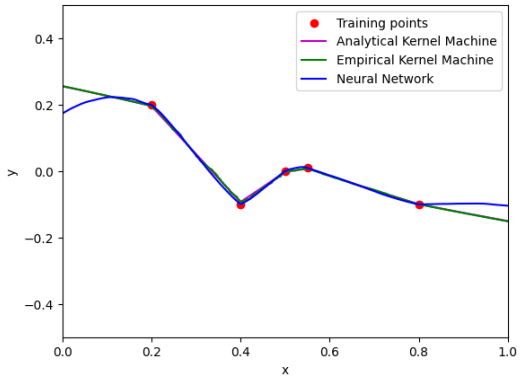

Figure 5: The trained network from figure 2 and the two kernel machines from figure 3 and 4 plotted against each other

We can compare the outputs of the models and see that a lot of resemblance can be seen in the interpolating region, but the network does not match the kernel methods outside the range of the training data.

4 Modelling training with neural tangent kernels

We can see that the analytical expression for the NTK (equation 21) is in fact constant, if the network is wide enough, it will be linear in the parameters (Belkin [2021]). We are going to assume that is the case and exploit these two properties to study the training of a neural network with the kernel machines. We start by denoting a time dependence of the parameters $\mathbf{W}(t)$. Looking back at equation 5, we can rearrange it and sum over all the datapoints to find the kernel matrix $\mathbf{K}$.

$$\frac{df(\mathbf{x}, \mathbf{W}(t))}{dt} = -\frac{1}{N}\mathbf{K}(f(\mathbf{x}, \mathbf{W}(t)) - \mathbf{y}) \tag{22}$$

We find that we can easily subtract $d\mathbf{y}/dt = 0$ to recognize an ordinary differential equation.

$$\frac{d}{dt}(f(\mathbf{x}, \mathbf{W}(t)) - \mathbf{y}) = -\frac{1}{N}\mathbf{K}(f(\mathbf{x}, \mathbf{W}(t)) - \mathbf{y} \tag{23}$$

With solution:

$$f(\mathbf{x}, \mathbf{W}(t)) - \mathbf{y} = e^{-K\frac{t}{N}}(f_0 - \mathbf{y}) \tag{24}$$

Where f_0 is the predictions with untrained weights. We will then create a kernel machine using equations 7 and 8 to predict the errors of a training network on $x \in [0, 1]$.

$$f(x, \mathbf{W}(t)) - \mathbf{y} = \mathbf{K}(x, \mathbf{x})\mathbf{K}^{-1}(\mathbb{I} - e^{-K\frac{t}{N}})(f_0 - \mathbf{y}) \tag{25}$$

Which allows us to plot outputs during different "timesteps" of training the network without actually training it.

$$f(x, \mathbf{W}(t)) = \mathbf{y} + \mathbf{K}(x, \mathbf{x})\mathbf{K}^{-1}(\mathbb{I} - e^{-K\frac{t}{N}})(f_0 - \mathbf{y}) \tag{26}$$

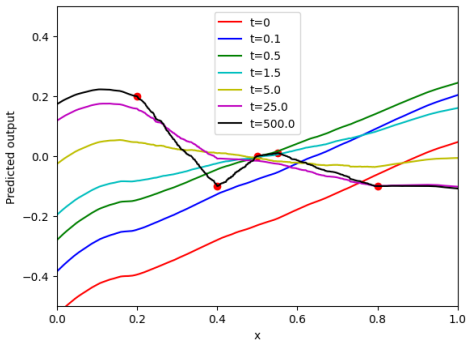

Figure 6: Training progression with the empirical kernel defined by equation 6

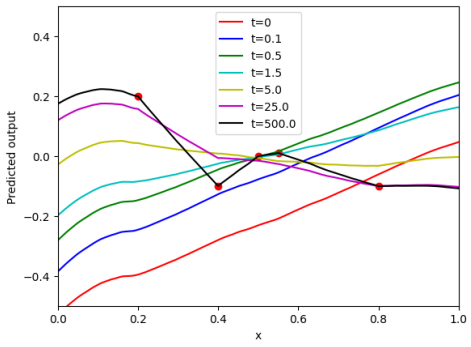

Figure 7: Training progression with the analytical kernel defined by equation 21

We can see that both of these figures are very similar to the actual training we did in figure 2. The predictions at $t = 500$ also resembles the actual training.

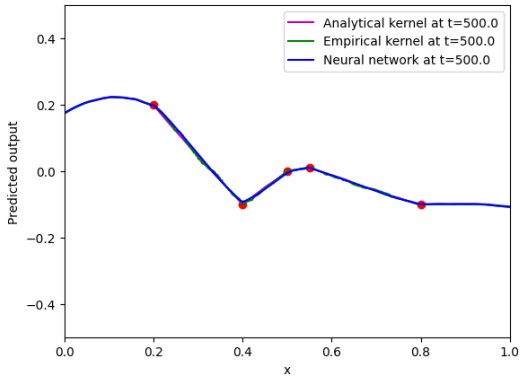

Figure 8: The trained network from figure 2 and the two training kernel machines at $t = 500$ from figure 6 and 7 plotted against each other

Our analysis showed that letting $t \to \infty$ did not make the predictions in figure 7 converge to the predictions of the kernel machine from figure 4 outside of the interpolating regime. Instead, the predictions with formula 26 actually resemble the network predictions better than the kernel machine (as shown in figure 8), because they take the initialization f_0 into account.

5 Conclusion

This project has played around with neural tangent kernels for a neural network with 1 hidden layer of 10 000 units and a toy dataset with 4 datapoints. We have successfully derived the NTK and showed that a kernel machine with this kernel will predict similarly to a trained version of the neural network. We then proceeded to show that even the training progression can be modeled by this NTK, and its predictions got even closer to the ones from the neural network.

The findings of this paper highlights multiple connections between the neural network defined in the beginning and its neural tangent kernel. However, it is hard to say if these relationships stands true for other network architectures. Belkin [2021] argues that the relationship between these two models is not yet clear to the scientific community, but a lot of researchers believe that neural networks cannot be fully characterized as an approximation of kernel machines. The conclusion of this paper is that the NTK serves as a useful tool for analyzing wide neural networks with one hidden layer.

References

Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation, 2021. URL https://arxiv.org/abs/2105.14368.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020. URL https://arxiv.org/abs/1806.07572.

Simon Prince. The neural tangent kernel. https://rbcborealis.com/research-blogs/the-neural-tangent-kernel/, 2022. Accessed: 2025-03-17.