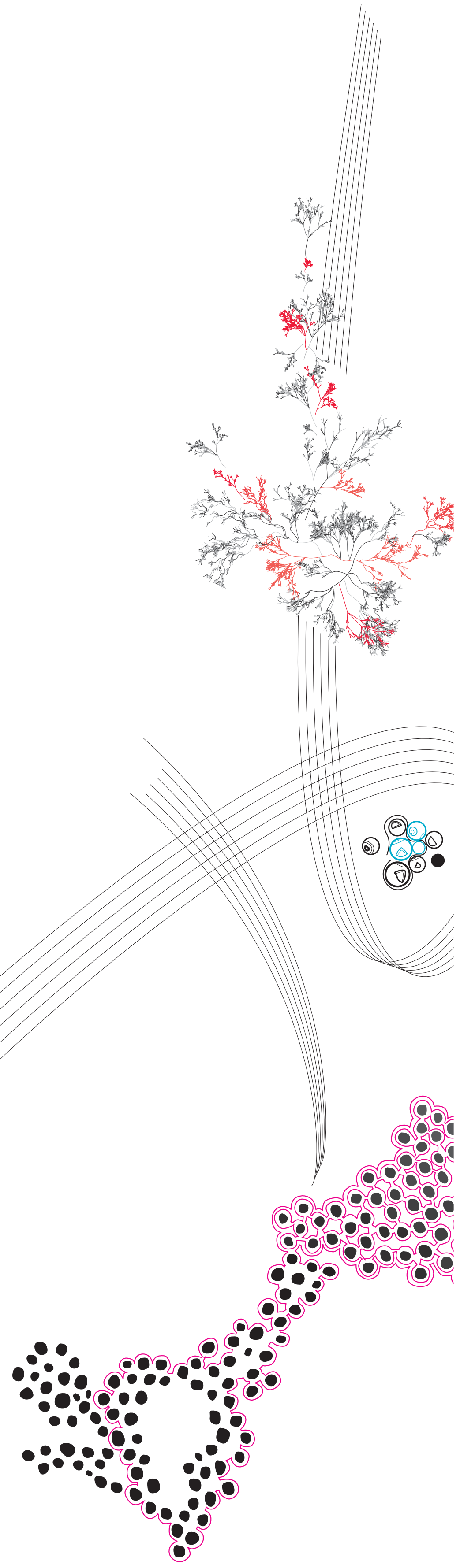


BSc Thesis Applied Mathematics

Eigenvalues of the Neural Tangent Kernel for different Network Architectures

Niels Berg

Supervisors: Christoph Brune, Tjeerd Jan Heeringa

July, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

I would like to give many thanks to Tjeerd Jan Heeringa for taking a lot of time to help me with my research for my bachelor assignment.

Eigenvalues of the Neural Tangent Kernel for different Network Architectures

Niels Berg

July, 2024

Abstract

For fully connected neural networks the width, depth, and activation functions used are usually picked using intuition of what has worked well before. It would be better if we can know what structure of a neural network works well before starting training. For this purpose we can calculate the Neural Tangent Kernel (NTK). We show that the speed of training can be bounded using the smallest eigenvalue of the NTK. We look at how the smallest eigenvalue changes between networks with different widths, depths, and activation functions, and also how this bound may not directly correlate with the speed of training.

Keywords: Neural Tangent Kernel, Activation Functions

Contents

1 Introduction

Neural networks have become a very mainstream topic in the 2020's, with more and more research in the field being done each day. One newer topic is the study of the neural tangent kernel (NTK), whose properties can tell us much about the network it is associated with. One of the properties of the NTK is its eigenvalues. This leads us to our research question which is:

How do the eigenvalues of the neural tangent kernel impact the speed of training for networks with varying widths, depths, and activation functions?

We will answer the research question by looking at the theory behind the neural tangent kernel and its eigenvalues, after which we will compute the eigenvalues for many different network architectures. Finally we will perform training on some networks to see how the theoretical results have impact on the training in practice.

2 Neural Networks & The Neural Tangent Kernel

2.1 Neural Network

A fully connected neural network is a function $f_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_D}$, where $f_\theta(x)$ is normally defined as follows:

$$\begin{aligned} x^{(0)} &= x \\ x^l &= \sigma(W^{(l)}x^{(l-1)} + b^{(l)}) \\ x^D &= W^{(D)}x^{(D-1)} + b^{(D)} \\ f_\theta(x) &= x^D. \end{aligned} \tag{1}$$

We use θ to denote the collection of all the weights $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ and biases $b^l \in \mathbb{R}^n_l$. All the elements of θ are called the learnable parameters which we will take a closer look at in the next section. The dimensions n_l are called the width of layer l. The number of layers D is referred to as the depth of the network. Finally, the function σ is called the activation function and is used to introduce some nonlinearity to the network. Many different activation functions are used for this which we will also see later.

2.2 Learning

We can try to approximate any function with a neural network, which means that we want to find the optimal parameters for the best possible approximation.

For this, a loss function is used to compare the output of a network on a specific training sample $f(x_n)$ to the target y_n. A common loss function is the Mean Squared Error loss function:

$$L(\theta) = \frac{1}{N}\sum_{n=1}^{N} ||y_n - f_\theta(x_n)||_2^2$$

for a data set $((x_1, y_1), ..., (x_N, y_N))$. We can also write it as

$$\frac{1}{N}(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T(\boldsymbol{y} - f_\theta(\boldsymbol{x})),$$

where $\boldsymbol{y} = (y_1, ..., y_N)^T$ and $f_\theta(\boldsymbol{x}) = (f_\theta(x_1), ..., f_\theta(x_N))^T$.

To find the best parameters, we need to find the minimum of the loss function. The usual method to do this is with gradient descent. The parameters are initialised randomly to obtain $\theta^{(0)}$. Some learning rate η is chosen, and then the parameters are updated:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta L_(\theta^{(k)}). \tag{2}$$

When the learning rate is sufficiently small, this process guarantees that the loss approaches a (local) minimum.

2.3 Neural Tangent Kernel

The Neural Tangent Kernel (NTK) arises when we look at the dynamics of the network during training.
We can rearrange equation (2) into the following form:

$$-\nabla_\theta L(\theta^{(k)}) = \frac{\theta^{(k+1)} - \theta^{(k)}}{\eta}.$$

When η approaches 0, this will become the derivative of θ with respect to time t, so

$$-\nabla_\theta L(\theta^{(t)}) = \frac{d\theta^{(t)}}{dt}$$

When we now want the derivative of the network over time: $\frac{df_\theta(x)}{dt}$, we can use the chain rule to do the following derivation. We will omit the t from $\theta^{(t)}$ but keep in mind that the parameters are always dependent on the time.

$$\begin{aligned} \frac{df_\theta}{dt}(\boldsymbol{x}) &= (\nabla_\theta f_\theta(\boldsymbol{x}))\frac{d\theta}{dt} = -(\nabla_\theta f_\theta(\boldsymbol{x}))\nabla_\theta L_{MSE}(\theta) \\ &= -(\nabla_\theta f_\theta(\boldsymbol{x}))\frac{1}{N}\nabla_\theta((\mathbf{y} - f_\theta(\mathbf{x}))^T(\mathbf{y} - f_\theta(\mathbf{x}))) \\ &= -\frac{2}{N}(\nabla_\theta f_\theta(\boldsymbol{x}))(\nabla_\theta f_\theta(\boldsymbol{x}))^T(\boldsymbol{y} - f_\theta(\boldsymbol{x})). \end{aligned} \tag{3}$$

The term $(\nabla_\theta f_\theta(\boldsymbol{x}))(\nabla_\theta f_\theta(\boldsymbol{x}))^T$ is known as the empirical neural tangent kernel. The size of the NTK is $N \times N$. Its elements have size $n_D \times n_D$. For simplicity, we will only consider a network with a single output, so $f_\theta(x) \in \mathbb{R}$ i.e. $n_D = 1$.

In this case, element i, j of the NTK is

$$\boldsymbol{\Theta}_{ij} = (\nabla_\theta f_\theta(x_i))^T(\nabla_\theta f_\theta(x_j)) = \sum_\theta \frac{df(x_i)}{d\theta_k}\frac{df(x_j)}{d\theta_k}. \tag{4}$$

2.4 Initialisation

To start the learning process we need to choose the initial values for all the parameters. When analysing the neural tangent kernel, NTK parametrisation is used which slightly tweaks the structure of the network:

$$\begin{aligned} x^{(0)} &= x \\ x^{l} &= \sigma(\frac{1}{\sqrt{n_l}}W^{(l)}x^{(l-1)} + b^{(l)}) \\ x^{D} &= W^{(D)}x^{(D-1)} + b^{(D)} \\ f_\theta(x) &= x^{D}. \end{aligned} \tag{5}$$

The only difference with between the equations in (1) and those in (5) is the factor $\frac{1}{\sqrt{n_l}}$, so this new definition does not change what functions are learnable, but it prevents the gradients from getting too large when calculating the gradient using backpropagation, for networks with very large widths. for this reason this network definition and initialisation is usually used when studying the neural tangent kernel, starting with the first paper on the NTK [3].
Lastly, with this definition the weights and biases are all initialised using a normal distribution with mean 0 and standard deviation 1.

3 Properties of the Neural Tangent Kernel

3.1 Change in Time

We will now see how the network changes in time.
Recall from equation (3) that

$$\frac{df_\theta(\boldsymbol{x})}{dt} = -\frac{2}{N}\boldsymbol{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x})).$$

Since the training targets $\boldsymbol{y}$ are independent of time we can also state the following:

$$\frac{d}{dt}(\boldsymbol{y} - f_\theta(\boldsymbol{x})) = \frac{df_\theta(\boldsymbol{x})}{dt} = -\frac{2}{N}\boldsymbol{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x})), \tag{6}$$

which looks a lot like an ordinary differential equation. In fact, if $\boldsymbol{\Theta}$ would be independent of time then we can solve the differential equation like

$$\begin{aligned} &\frac{d}{dt}(\boldsymbol{y} - f_\theta(\boldsymbol{x})) = -\frac{2}{N}\boldsymbol{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x})) \\ \implies &(\boldsymbol{y} - f_\theta(\boldsymbol{x})) = e^{-\frac{2}{N}\boldsymbol{\Theta}t}(\boldsymbol{y} - f_{\theta^{(0)}}(\boldsymbol{x})) \\ \implies &f_\theta(\boldsymbol{x}) = \boldsymbol{y} - e^{-\frac{2}{N}\boldsymbol{\Theta}t}(\boldsymbol{y} - f_{\theta^{(0)}}(\boldsymbol{x})). \end{aligned} \tag{7}$$

But as stated previously this is only valid if the neural tangent kernel is independent of time which it clearly is not since the parameters θ change over the course of training and as such the gradient with respect to θ will change, which is part of the definition of the NTK.
However, a famous result from Jacot et al. [3] introducing the NTK as a field of study proves that the neural tangent kernel does stay constant when the width of the network tends to infinity. This makes sense, because as the width increases, the amount of parameters goes up as well, so when a step of gradient descent is applied, each individual parameter is changed by a smaller amount.
This result means that we can approximate the evolution of the network over the training time with

$$f_\theta(\boldsymbol{x}) = \boldsymbol{y} - e^{-\frac{2}{N}\boldsymbol{\Theta}^{(0)}t}(\boldsymbol{y} - f_{\theta^{(0)}}(\boldsymbol{x})),$$

where $\mathbf{\Theta}^{(0)}$ is the neural tangent kernel at time 0, so before any training. This greater the width of the network, the better this approximation will be.

3.2 Smallest Eigenvalue

The eigenvalues of the neural tangent kernel tell much about the neural network it came from. This is not a novel concept; in fact the eigenvalues of the NTK have been heavily studied, such as in [5], [6], [8], which all also link the eigenvalues to training speed in some way.

When we train our model, we are trying to minimise the loss. Let us calculate the change of the loss over time.

$$\begin{aligned}\frac{dL(\theta)}{dt} &= \frac{d}{dt}((\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T(\boldsymbol{y} - f_\theta(\boldsymbol{x}))) \\ &= 2(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T\frac{d}{dt}(\boldsymbol{y} - f_\theta(\boldsymbol{x})).\end{aligned}$$

We can use equation (6) to continue with the above:

$$\frac{dL(\theta)}{dt} = 2(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T(-\frac{2}{N}\mathbf{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x}))) = -\frac{4}{N}(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T\mathbf{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x})). \tag{8}$$

There exists a theorem that links the above to the smallest eigenvalue of $\mathbf{\Theta}$.

Rayleigh's theorem [2, p. 234-235] states, among others:
For all real vectors x and symmetric real matrices A,

$$\frac{x^TAx}{x^Tx} \geq \lambda_{min}$$

Where λ_{min} is the smallest eigenvalue of A.

Recall from equation (4)

$$\begin{aligned}\mathbf{\Theta}_{ij} &= (\nabla_\theta f_\theta(x_i))^T(\nabla_\theta f_\theta(x_j)) = \sum_\theta \frac{df(x_i)}{d\theta_k}\frac{df(x_j)}{d\theta_k} \\ &= \sum_\theta \frac{df(x_j)}{d\theta_k}\frac{df(x_i)}{d\theta_k} = (\nabla_\theta f_\theta(x_j))^T(\nabla_\theta f_\theta(x_i)) = \mathbf{\Theta}_{ji},\end{aligned}$$

so the NTK is a symmetric matrix and thus we apply Rayleigh's theorem to obtain

$$\frac{(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T\mathbf{\Theta}(\boldsymbol{y} - f_\theta(\boldsymbol{x}))}{(\boldsymbol{y} - f_\theta(\boldsymbol{x}))^T(\boldsymbol{y} - f_\theta(\boldsymbol{x}))} \geq \lambda_{\min},$$

where $\lambda_{\min}$ is the smallest eigenvalue of $\mathbf{\Theta}$.
We can use the above to continue equation (8):

$$\frac{dL(\theta)}{dt} = -\frac{4}{N}(\boldsymbol{y}-f_\theta(\boldsymbol{x}))^T\mathbf{\Theta}(\boldsymbol{y}-f_\theta(\boldsymbol{x})) \leq -\frac{4}{N}\lambda_{\min}(\boldsymbol{y}-f_\theta(\boldsymbol{x}))^T(\boldsymbol{y}-f_\theta(\boldsymbol{x})) = -4\lambda_{\min}L(\theta)t. \tag{9}$$

Note the inequality sign switching because we multiplied the inequality by $\frac{-4}{N}$.
From the above we now obtain an upper bound for the loss function in continuous time:

$$\begin{aligned}&\frac{dL(\theta)}{dt} \leq -4\lambda_{\min}L(\theta) \\ \implies &L(\theta^{(t)}) \leq L(\theta^{(0)})e^{-4\lambda_{\min}}.\end{aligned} \tag{10}$$

4 Experiments

We will first explore the structure of the NTK in section 4.1, after which we will look at the eigenvalues in section 4.2. To answer our research question we will then see how the training proceeds for some of these networks in section 4.3. Finally the results will be explained in section 4.4.

4.1 Graphical Representation of the NTK

We will take use a network described as in (5), also initialised using the initialisation stated there.

The dataset we use is the Fashion-MNIST database [7], which contains 6000 grayscale images of 28 by 28 pixels for 10 different pieces of clothing. This is a very commonly used toy dataset in machine learning research.
Because we would like to have a network with a single-dimensional output, we will make a network whose purpose is to differentiate between images of a T-shirt/Top (which have 0 as target) and images of an Ankle Boot (which we will give 1 as target). These restrictions on the input and output means that in our network $n_0 = 28 * 28 = 784$ and $n_D = 1$. We will also use the same depth for all layers in between, i.e. $n_i = n_j$ for all i, j with $0 < i, j < D$.

We have 12000 data points (images) to work with, but we cannot use these all because we will be unable calculate a 12000×12000 NTK. Instead we will use a very small database for all our experiments consisting of 100 images of a T-Shirt/Top and 100 images of an Ankle Boot so that we can perform calculations with the NTK.

The networks that we will compare will have widths between 50 and 1000 in order have a somewhat wide network in order to be able to use the theoretical results for infinite width networks, but not too wide as to keep the calculations in a reasonable time. This is supported by figure 1 in [4] shows us that the NTK is fairly consistent for a width of 50, and nearly constant for a width of 1000.
We will consider a depth of 2 (1 hidden layer) as the smallest, since if we recall equation (5), the activation function is not used in the last layer which means it will not appear in a network of depth 1. We will consider a maximum depth of 10 to see how the eigenvalues will change for some deeper networks.

The activation functions that we will compare are some of the most widely used activation functions and include the following:

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mathrm{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases}$$

$$\mathrm{LReLU}_{0.01} = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$$

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

With this in place, we can take a look at what the NTK looks like for some different widths, heights, and activation functions.

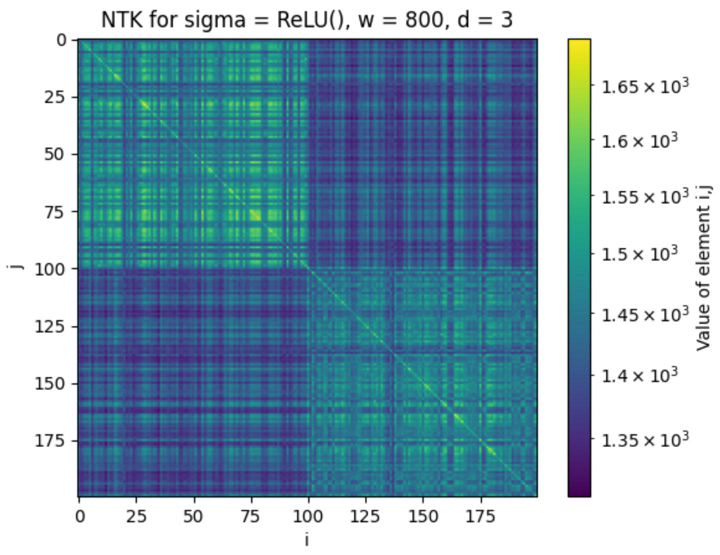


FIGURE 1: Visual representation of the NTK for a network using ReLU as activation function, a width of 800 and depth 3. A brighter color indicates a higher value for $(\nabla_\theta f_\theta(x_i))^T(\nabla_\theta f_\theta(x_j))$.

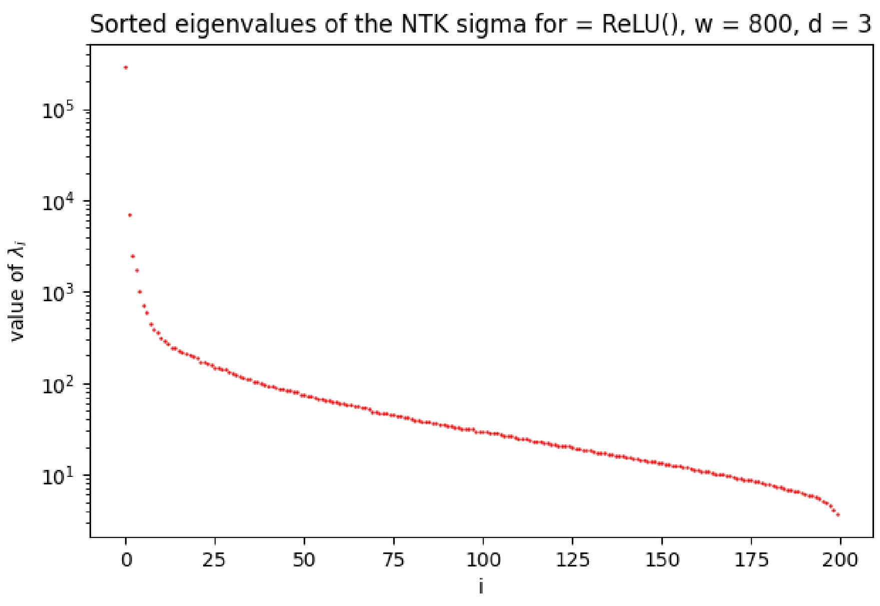
There are some observations to be made from figure 1. We can see that i ranges from 0 up to 200 on the x-axis, and the same for j on the y-axis. The brightness of one of the pixels in the image corresponds to a high value for

$$\mathbf{\Theta}_{ij} = (\nabla_\theta f_\theta(x_i))^T(\nabla_\theta f_\theta(x_j)) = \sum_\theta \frac{df(x_i)}{d\theta_k}\frac{df(x_j)}{d\theta_k}.$$

This is the case when for many parameters θ_k, the value of $\frac{df(x_i)}{d\theta_k}$ is similar to that of $\frac{df(x_j)}{d\theta_k}$, or in other words the training images contain similar patterns.

The visuals in the plot are not surprising. The diagonal is very bright since these are the gradients of f on a training example compared with itself. The squares we see are also expected because we have sorted the data. This means that figure 1 tells us the first 100 images are similar to each other and that the last 100 images are also similar to each other since these quadrants of the image are bright. We already know this is the case since the first 100 images are all of T-shirts and the last 100 are images of ankle boots.

The spectrum of this NTK can be seen in figure 2.



```
The smallest eigenvalue is 3.663857529962649
```

FIGURE 2: All eigenvalues sorted from highest to lowest of the NTK of a network with activation function ReLU, width 800 and depth 3.

We see that there are some extremely high eigenvalues, and that most other eigenvalues sit on a line (the y-axis is logarithmic, so this line is an exponential pattern). The lowest eigenvalue is approximately 3.664.

The patterns from the visual representation of the NTK in figure 1 are expected, but they turn out not to show up for some other network structures.

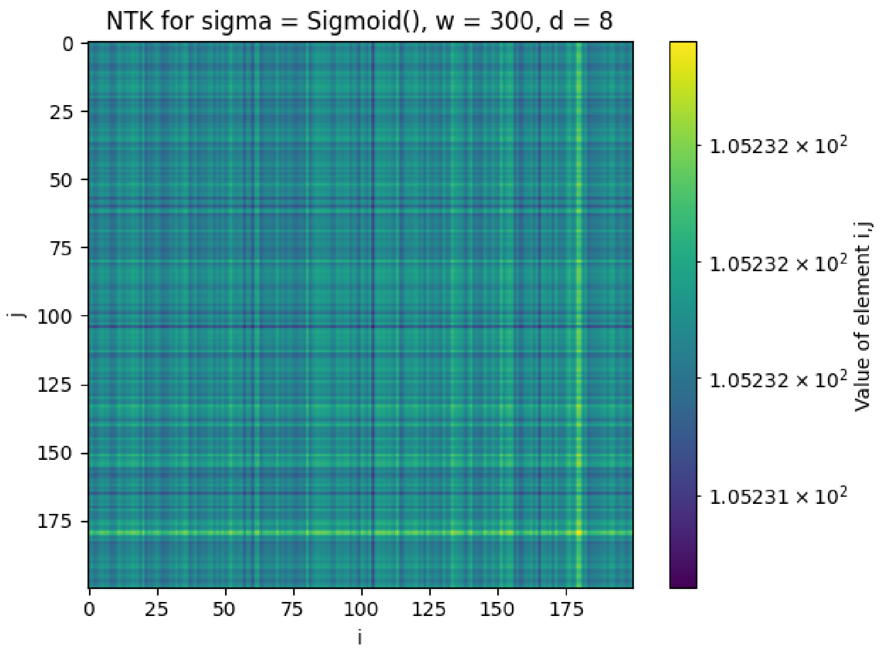


FIGURE 3: Similar visual representation of the NTK as before, but with a different network structure, namely Sigmoid as activation function, width 300, and depth 8.

In figure 3 there do not seem to be any patterns, not even on the diagonal which implies this type of network does find that any type of images correlate to each other. When we look at the spectrum in figure 4,

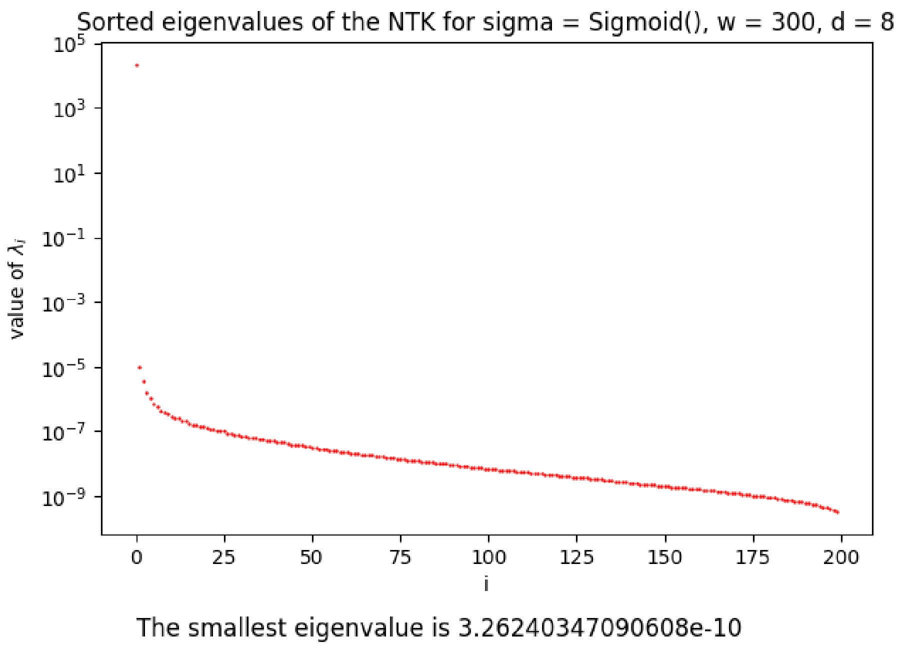


FIGURE 4: All eigenvalues sorted from highest to lowest of the NTK of a network with activation function Sigmoid, width 300 and depth 8.

We see that it looks qualitatively very similar to the spectrum in figure 2, but most of the eigenvalues are significantly lower, with the lowest being approximately $3.935 \cdot 10^{-10}$. This implies that this network will likely take much longer to train than the network in the example above this one.

4.2 Smallest Eigenvalue

Now that we have seen what the NTK and its eigenvalues look like for some network structures we can now look at the smallest eigenvalues for all network architectures that we are interested in.

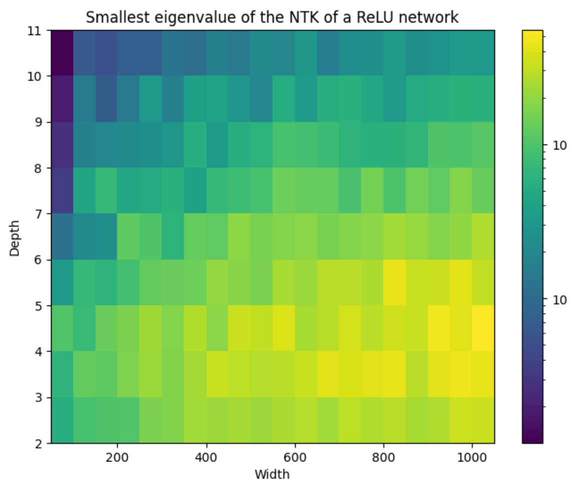


FIGURE 5: Smallest Eigenvalue of the NTK for networks using ReLU.

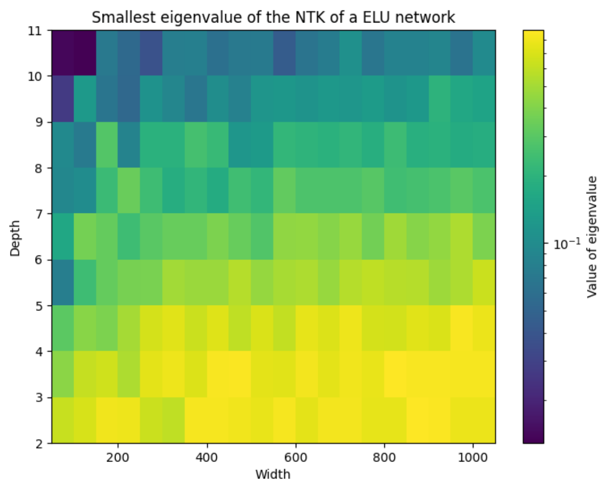


FIGURE 6: Smallest Eigenvalue of the NTK of networks using ELU.

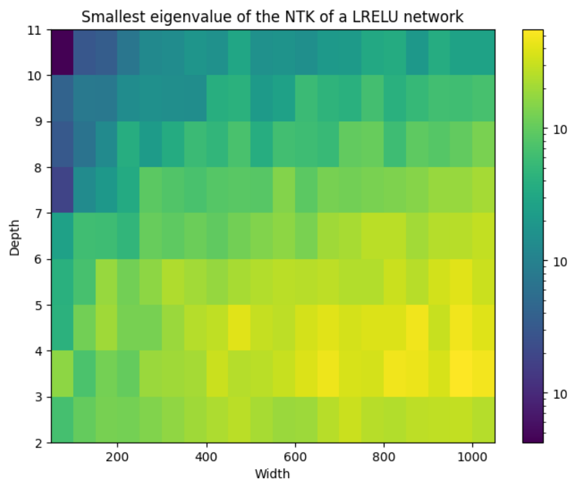


FIGURE 7: Smallest Eigenvalue of the NTK of networks using Leaky ReLU.

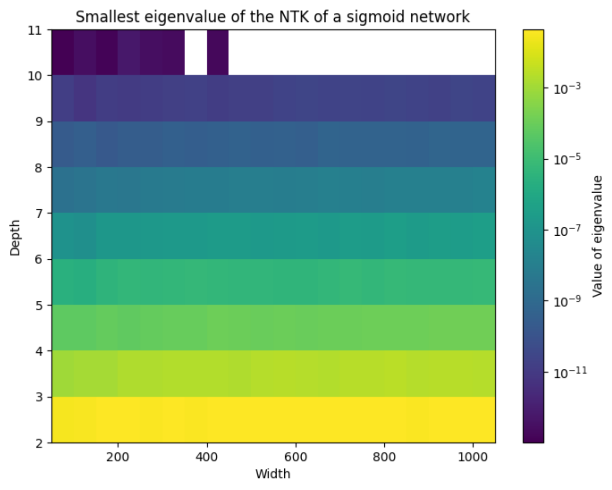


FIGURE 8: Smallest Eigenvalue of the NTK of networks using sigmoid.

The figures show the smallest eigenvalue of the NTK for different widths and depths for a network using a certain activation function. The white spaces in the bottom right figure are where the smallest eigenvalue was calculated to be below 0. For the best performance in the training of a network we would like the smallest eigenvalue to be as large as possible. Table 1 contains the greatest eigenvalue per activation function, and for what width and depth this greatest eigenvalue is found at.

TABLE 1: The greatest of the smallest eigenvalues per activation function.

Activation Function	Greatest Smallest Eigenvalue	Width	Depth
ReLU	5.490	1000	4
ELU	0.889	850	2
$\text{LReLU}_{0.01}$	5.548	950	3
Sigmoid	0.044	450	2

There are again some observations to be made.

Firstly, in general a larger depth decreases the smallest eigenvalue of the NTK. In fact for the networks using the ELU and Sigmoid activation functions the greatest smallest eigenvalue is found at the minimum depth of 2. The greatest smallest eigenvalue for the other two activation functions are not found at a much higher depth, namely at a depth of 3 and 4.

Furthermore, for the ReLU, LReLU, and ELU there is also a curve where the smallest eigenvalue increases as the width increases and also falls off more slowly at higher depth, for instance at a width of 50 the smallest eigenvalue decreases more rapidly as the depth increases than for a width of 1000.

As for the networks using the Sigmoid activation function we see that the smallest eigenvalue is mostly independent of the depth and immediately decreases as the depth increases, and that the smallest eigenvalue even becomes negative at very high width and depth due to numerical error.

4.3 Verification

We will now execute the training process of some of the networks in order to see if the previous results hold up in practice.
For all training processes, we will use the same small dataset ($N = 200$) that we used to calculate the neural tangent kernel. The precise method that we will use is stochastic gradient descent with a learning rate of 10^{-3}, a batch size of 1 (since the number of training examples is so low) over 100 epochs.

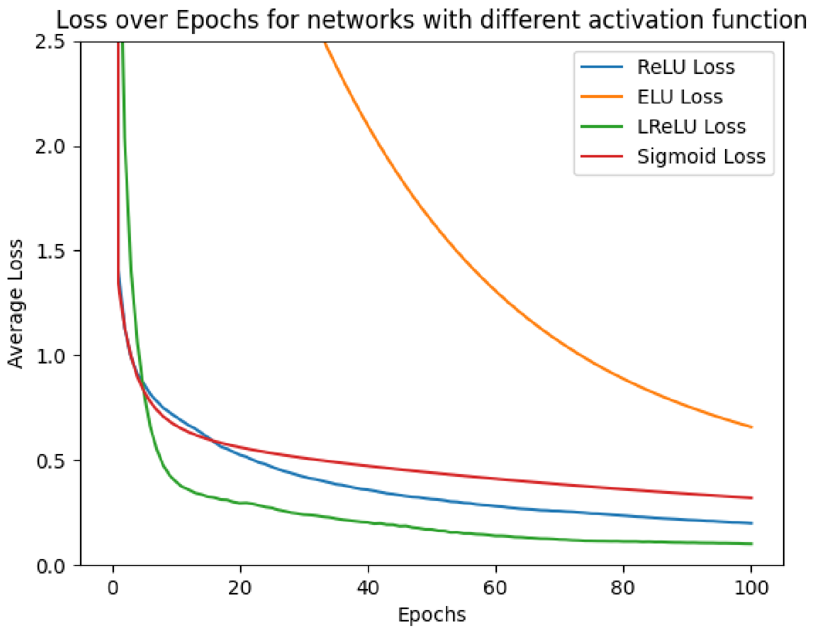


FIGURE 9: Loss over epochs for networks with the activation function, width, and depth from table 1.

Note that for these results, there is a small bit of variance because of the initialisation, but the general conclusion from figure 9 is that the networks with ReLU, Leaky ReLU, and Sigmoid as activation function train about equally well, while the network with ELU lags behind in training.

We will also take a look at how the training will go in for the networks that generated the visual NTK representations from figures 1, 3. This loss can be seen in figure 10.

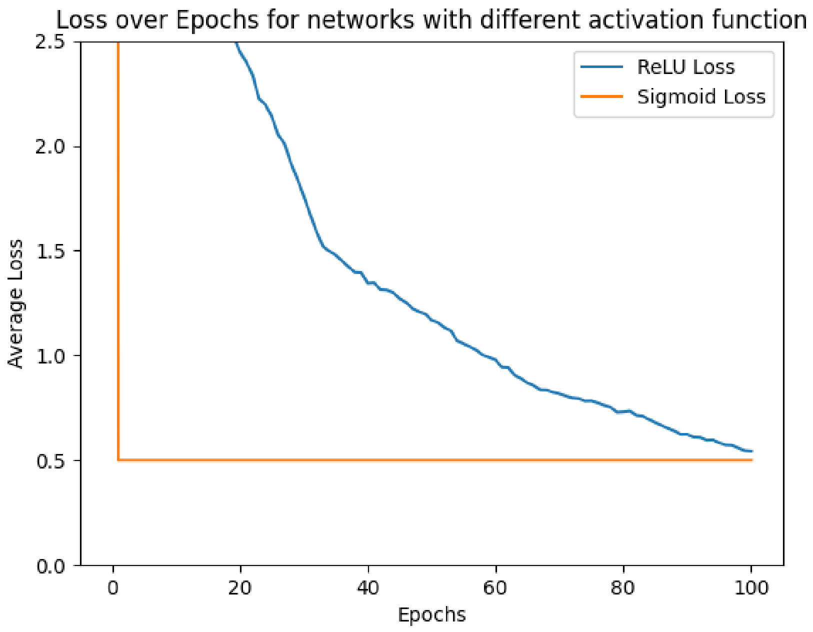


FIGURE 10: Loss compared between the networks for which we saw the NTK in figures 1, 3

.

4.4 Explanation

From the results from table 1 we would expect that the networks with the 'optimal' width and depth with the ReLU and Leaky ReLU activation function would train about equally as well. The 'optimal' network with the ELU activation function would train worse than these two, and the network with Sigmoid would train even worse.
When we look at what actually happens during training this hypothesis is not really true. The explanation is likely simple.
Let us look at the bound we obtained for the loss over the course of training from equation (10). We cannot use this bound exactly since it relies on a continuous training. Simon S. Du et al. [1] showed that there also exists a bound for the discrete training that is normally used.
For this bound we need a width at least in the order $\frac{N^6}{\lambda_{\text{min}}}$ and a learning rate in the order of $\frac{\lambda_{\text{min}}}{N^2}$, which imply that the loss is bounded like

$$L(\theta^t) \leq (1 - \frac{\eta\lambda_{\text{min}}}{2})^t L(\theta^{(0)}).$$

If we compare this to the loss of the ReLU network from figure 9. Our width is not even close to the order of magnitude to $\frac{200^6}{5.490}$. Our learning rate is a little closer to the required order $\frac{5.490}{200^2} \approx 1.37\dot{3}10^{-4}$. Because our parameters are not the sufficient size, this upper bound might not hold. But if we plot this,

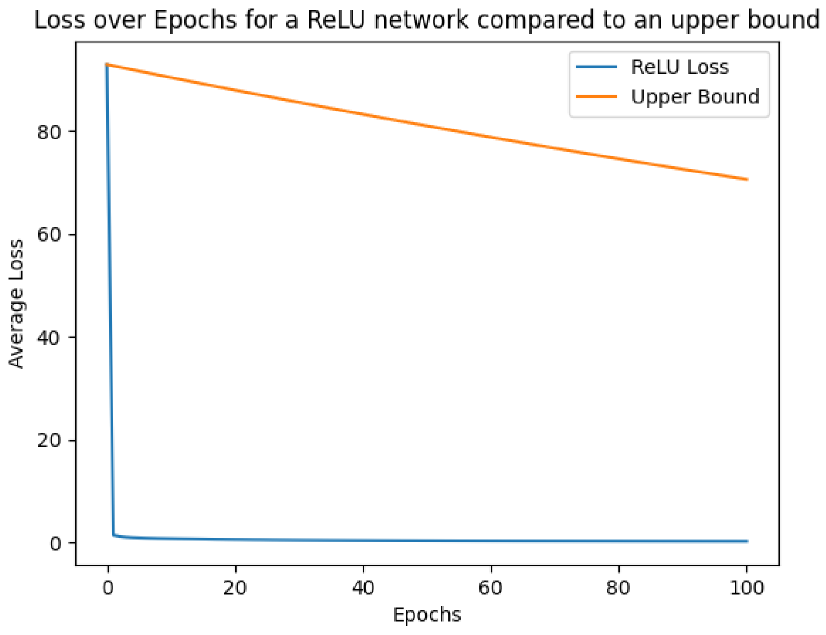


FIGURE 11: Loss of a ReLU network compared to its supposed upper bound.

We see in figure 11 that our loss fits very well under the upper bound for discrete gradient descent.
The reason why the eigenvalues from table 1 do not seem to have much of an impact on the actual training is likely due to the fact that the upper bounds induced by these eigenvalues are too high to be meaningful for our simpler task.

On the other hand, we do see something that we expect in figure 10. We see that the network with the ReLU activation function learns the dataset, albeit more slowly than the Rely network in figure 9. As for the network with the sigmoid activation function, recall that all targets are 0 or 1, which means that the steady average loss of 0.5 means the network is merely guessing the correct outcome.
This corresponds to the fact that the NTK of the ReLU network seemed very organised while the NTK of the Sigmoid network was very chaotic. The impact that an 'organised' neural tangent kernel has on the performance and training of a network is an interesting point for further study.

5 Conclusion

We have seen that in theory, the dynamics of the loss can be bounded using the smallest eigenvalue of the neural tangent kernel.

For our task of recognising clothing articles in a dataset of size 200, the networks with ReLU and Leaky ReLU as activation function have the greatest value for the smallest eigenvalue for many different widths and depths. The network with Sigmoid as activation function has lower eigenvalues and would be expected to train slower.

In practice, our task is likely too simple to see the impact of these eigenvalues, since when doing the training the networks with different activation functions performed very similarly, with only the network using ELU as activation function doing a bit less well than the other three networks.

To answer our research question: The eigenvalues of the NTK give a bound to the speed of training but do not directly correlate to how fast the loss decreases.

6 Discussion

As far as the conclusion goes, we do not prove that the reason the eigenvalues do not correspond to training speed in our experiments is due to the task being too simple, but we can be sure since the mathematical theory behind the bound is correct and the approximate bound from [1] is clearly much higher than the actual loss.

We have not concerned ourselves with the actual performance of the network, only the dynamics of the loss (i.e. the training speed). As such all losses plotted are training losses. This is because a network which quickly decreases in training loss over the course of its training will likely also decrease quickly in test loss, at least for a certain number of epochs at the start.

Finally, we know that the neural tangent kernel is constant in the limit of infinite width, but it would have been a good idea to verify how much the NTK changes for the widths that we work with, in order to see how close the NTK is to constant for those widths.

References

[1] Simon S. Du et al. *Gradient Descent Provably Optimizes Over-parameterized Neural Networks.* 2019. arXiv: `1810.02054 [cs.LG]`. URL: `https://arxiv.org/abs/1810.02054`.

[2] C. A. Horn R. A.; Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

[3] Arthur Jacot, Franck Gabriel, and Clément Hongler. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks.* 2020. arXiv: `1806.07572 [cs.LG]`. URL: `https://arxiv.org/abs/1806.07572`.

[4] S. Prince. *The Neural Tangent Kernel. Borealis AI.* URL: `The%20Neural%20Tangent%20Kernel.%20Borealis%20AI.`.

[5] Nasim Rahaman et al. *On the Spectral Bias of Neural Networks.* 2019. arXiv: `1806.08734 [stat.ML]`. URL: `https://arxiv.org/abs/1806.08734`.

[6] Sifan Wang, Xinling Yu, and Paris Perdikaris. *When and why PINNs fail to train: A neural tangent kernel perspective.* 2020. arXiv: `2007.14527 [cs.LG]`. URL: `https://arxiv.org/abs/2007.14527`.

[7] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* Aug. 28, 2017. arXiv: `cs.LG/1708.07747 [cs.LG]`.

[8] Greg Yang and Hadi Salman. *A Fine-Grained Spectral Perspective on Neural Networks.* 2020. arXiv: `1907.10599 [cs.LG]`. URL: `https://arxiv.org/abs/1907.10599`.