



Neural Tangent Kernel

$$\Theta^{(L)}(x, y) = \sum_{p=1}^P \frac{d}{d\theta_p} f_{\theta}(x) \frac{d}{d\theta_p} f_{\theta}(y)$$

Annotations:
- (L) : depth
- x, y : two samples
- $p=1$: all parameters

ES667 - Deep Learning Regularization Effects on Neural Tangent Kernels

Akash Gupta | 23110020

Luv Agarwal | 23110189

Rachit Mehta | 23110261

Problem Definition: Deep Learning Task

Task Overview

Understand how Neural Tangent Kernels evolve during training and are affected by regularization techniques.

Empirical vs Theoretical Results

We often see billions of parameters being used in deep neural networks, yet they don't overfit, whereas theoretically they should.

Our Objective

Study regularization effect on NTK

Neural Tangent Kernel (NTK)

Theory

1

Concept

When a Neural network has very wide hidden layers and large number of parameters, the change in parameters during training is very low (infinitesimal). NTK captures how a neural network function changes infinitesimally with respect to its parameters.

2

Mathematical Definition

Kernel defines network function's gradient similarity. NTK is a kernel function that tells us how two inputs are related based on the current state of the network.

3

Optimization Role

NTK alignment measures training effectiveness and convergence.

Why Do We Need NTK?

- Standard neural networks are hard to analyze due to their non-linear and overparameterized nature.
- We need **theoretical tools** to understand:
 - **Why they generalize well** despite being overparameterized.
 - How **training dynamics** behave.
 - The role of **regularization** in generalization.

Why Do We Need NTK?

NTK helps by:

- Giving a **linear approximation** of the network training dynamics.
- Allowing us to use kernel theory to **analyze deep learning**.
- Showing that in the infinite-width limit, training a neural network is like training a **kernel machine** (like ridge regression).

Neural Tangent Kernel (NTK)

Mathematical Definition

Consider a neural network $f(x; \theta)$ with:

$x \in \mathbb{R}^d$ (input), $\theta \in \mathbb{R}^P$ (parameters), $f(x; \theta) \in \mathbb{R}$ (output).

Neural Tangent Kernel (NTK) is defined as:

$$\Theta(x, x') := \nabla_{\theta} f(x; \theta)^{\top} \nabla_{\theta} f(x'; \theta)$$

Where:

$\nabla_{\theta} f(x; \theta) \in \mathbb{R}^P$ is the gradient of the output w.r.t. parameters

$\Theta(x, x') \in \mathbb{R}$ measures the inner product of the gradients at x and x'

Neural Tangent Kernel (NTK)

How to evaluate the NTK Matrix

We perform spectral analysis of the NTK matrix and focus our observations around the few largest eigenvalues of it. This is because the eigenvalues-

- **The largest eigenvalue influences the stability of gradient descent.**
- **Few large eigenvalues mean faster training of the model as they signify the dominance of a few principal components in training.**
- **When simulating infinite width layer, the NTK becomes practically constant, and thus eigenvalues define all properties of training.**
- **Sharp changes in eigenvalues coupled with anomalies in validation loss denote transitions between well-trained and over-fitting of the model**

Methodology

We analysed evolution of NTK spectrum across various parameter changes such as

- **Training Epochs**
- **Number of input samples**
- **L1 and L2 Regularization**
- **Width of hidden layer**

Data Description: Dataset

Overview

Data Source

MNIST Dataset

Memory Size

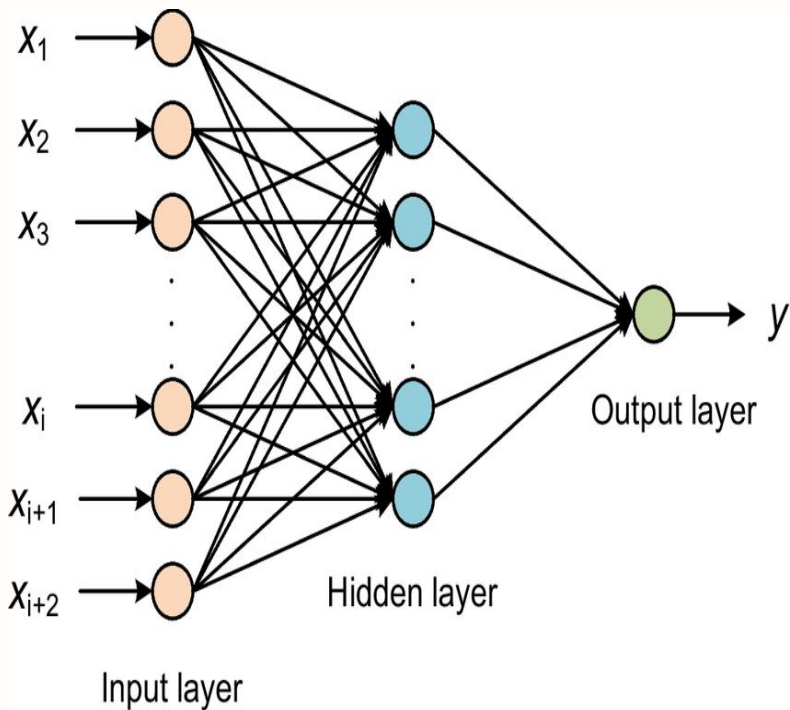
Approximately 50 MB for the entire dataset

Tensor Shapes

Inputs shaped as images of dimension 28x28.

Outputs as one-hot encoded vectors of length 10.

Model Architecture: Design and Data Flow



Architecture

A single hidden layer, fully connected, neural network of adjustable width

Activations

ReLU activates non-linear features throughout.

Libraries

Numpy, Torch, neural_tangents, Tensorflow, Jax, Scipy, Matplotlib

Empirical NTK Matrix(Definition)

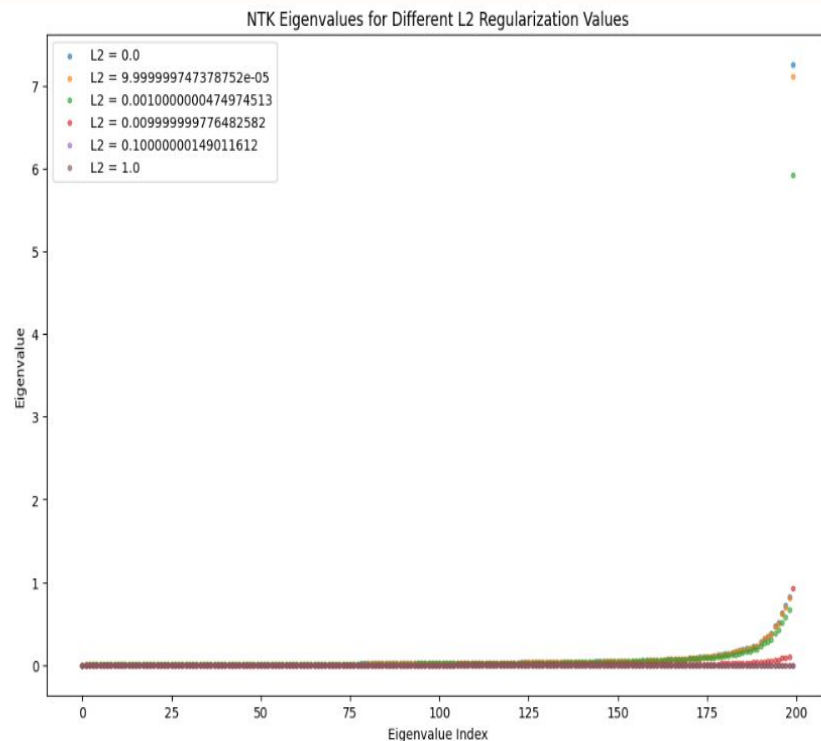
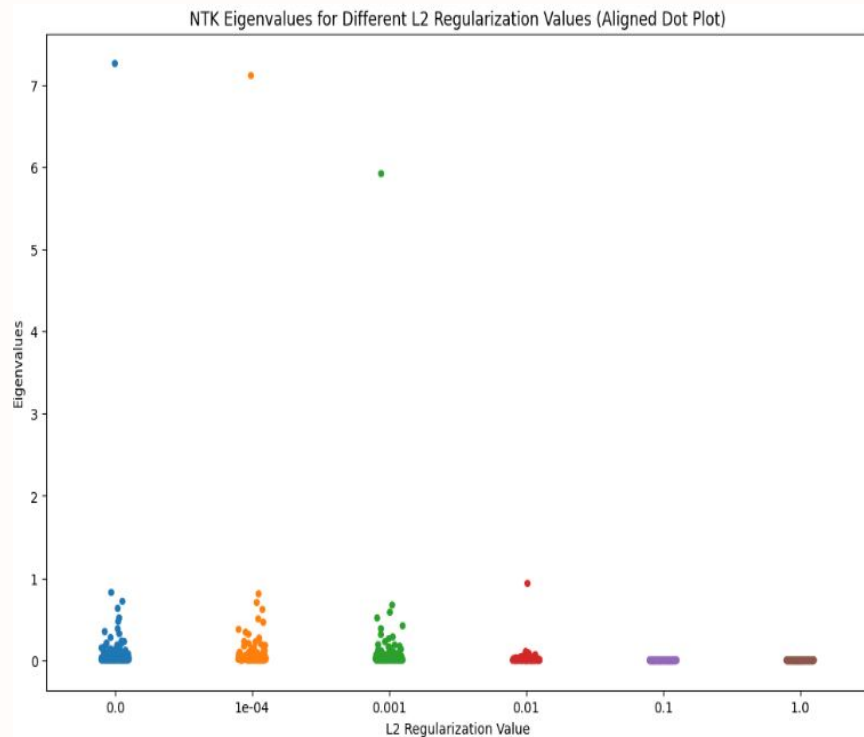
Given a neural network $f(x; \theta)$ and a batch of inputs x_1, x_2, \dots, x_n , the Empirical Neural Tangent Kernel (NTK) matrix Θ is a $n \times n$ matrix defined as:

$$\Theta_{ij} = \nabla_{\theta} f(x_i; \theta)^T \nabla_{\theta} f(x_j; \theta)$$

Each entry Θ_{ij} is the inner product of the gradients of the network output with respect to the parameters θ , evaluate data inputs x_i and x_j .

The empirical NTK is computed numerically from a finite (real – world) neural network, rather than theoretically in the infinite – width limit.

Experimental Results



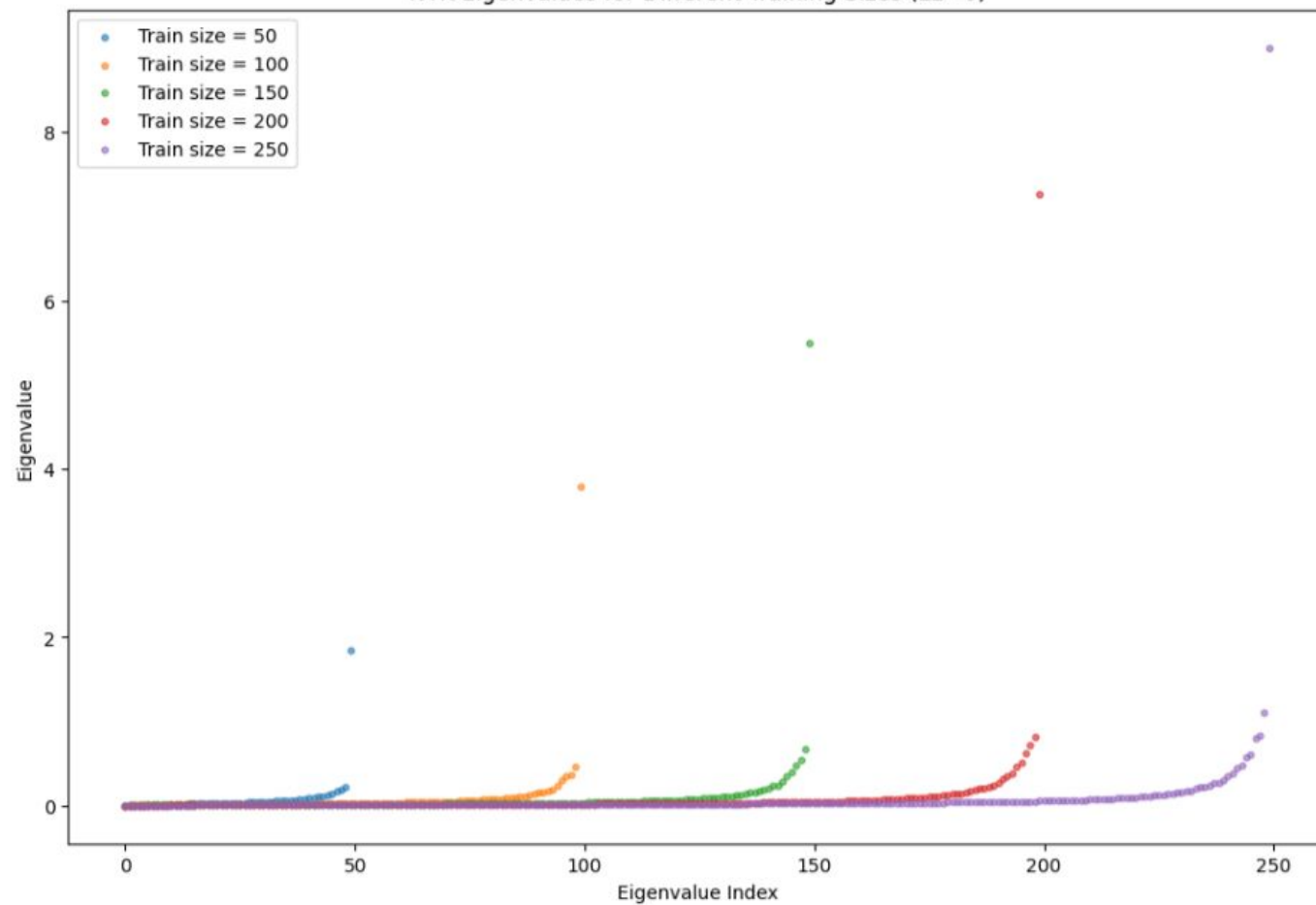
$$J = \sum_{n=1}^N (y_n - \hat{y}_n)^2 + \lambda |w|^2$$

$$|w|^2 = w^T w = w_1^2 + w_2^2 + \dots + w_D^2$$

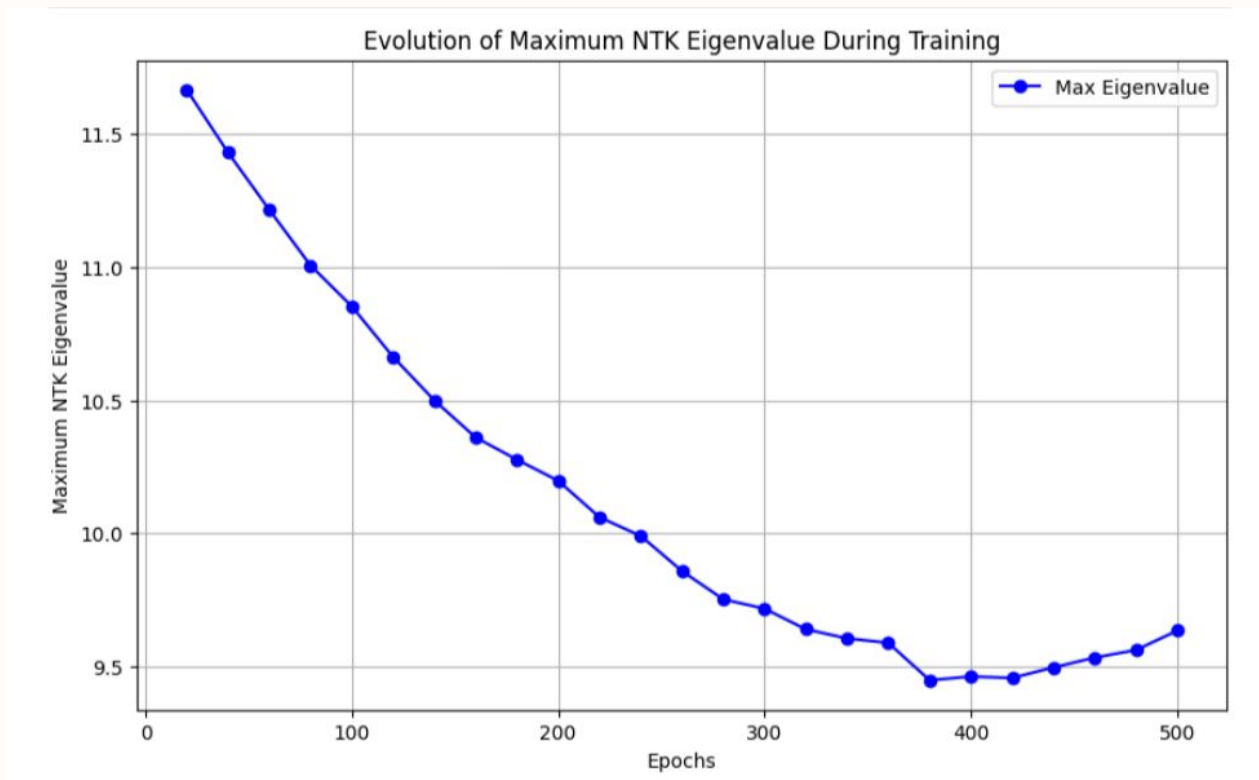
**NTK Spectrum for
varied L2 parameter**

The constant decrease in the largest eigenvalue with an increase in the L2 regularisation parameter suggests that the **expressive power of the model diminishes** - this is obvious as a larger regularisation often tends to focus more on minimizing the norm, ignoring the overall objective loss.

NTK Eigenvalues for Different Training Sizes (L2=0)



But what about NTK evolution during training??



Maximum Eigen-value and Change in parameters

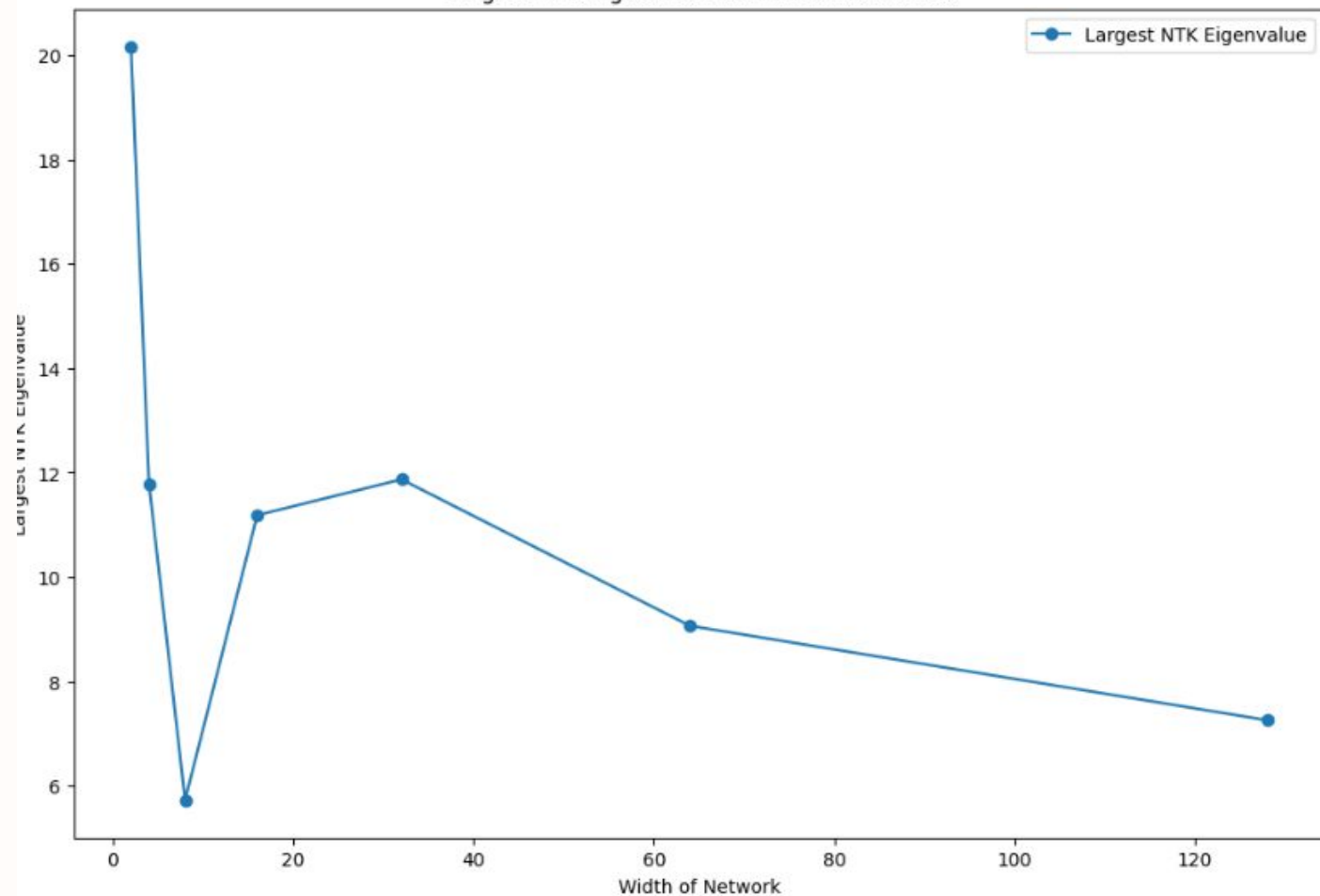
Key Intuitive Concepts

1. **NTK Definition:** The Neural Tangent Kernel (NTK) captures how the network's output changes with respect to small changes in parameters.
2. **Eigenvalues of NTK:** The eigenvalues represent the strength of sensitivity of the output to changes in parameters.
3. **Eigenvectors:** Correspond to the directions in the parameter space where the network is most or least sensitive.
4. **Maximum Eigenvalue:** The largest eigenvalue corresponds to the direction where the output changes the most.
5. **Rate of Change:** The maximum eigenvalue quantifies the rate at which the output changes with respect to parameter changes in the most sensitive direction.
6. **Conclusion:** The maximum NTK eigenvalue reflects the maximum sensitivity of the network's output to parameter changes, representing the rate of change in that direction.

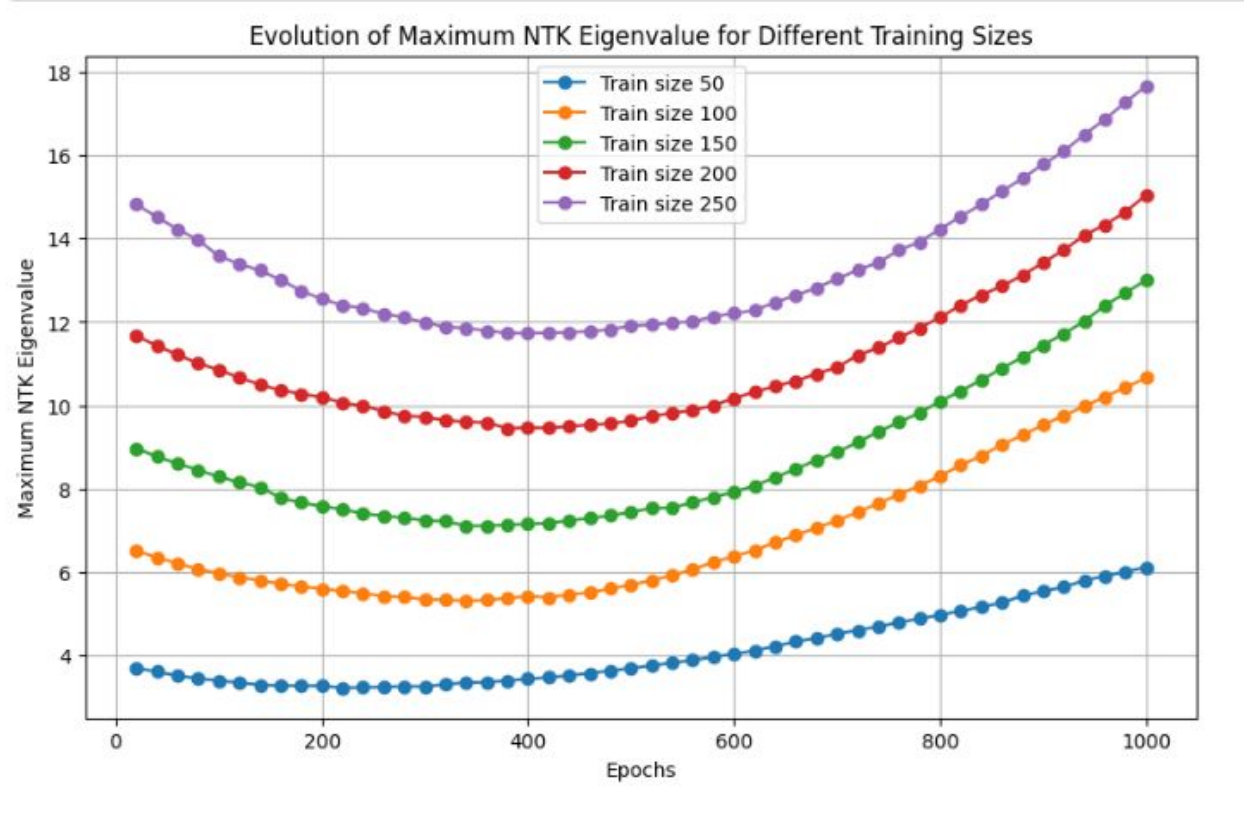
Explanation of Results

- *Maximum NTK eigenvalue* corresponds to *rate of change of parameters* in the Neural Network.
- According to the plot, *the rate of change decreases, reaches a minimum and then increases.*
- First the parameters of Neural network are random. Thus, *at initial epochs change in parameters is high.*
- As the *training epoch increase*, the NN is trained, and we observe that the *rate of change in parameters decreases.*
- After a few epochs, the parameters are trained enough and, *the rate of change of parameters is nearly zero.*
- If training is not stopped now, then *NN tries to overfit the data by learning complex functions* leading to again increase in parameter change.

Largest NTK Eigenvalue vs. Width of Network

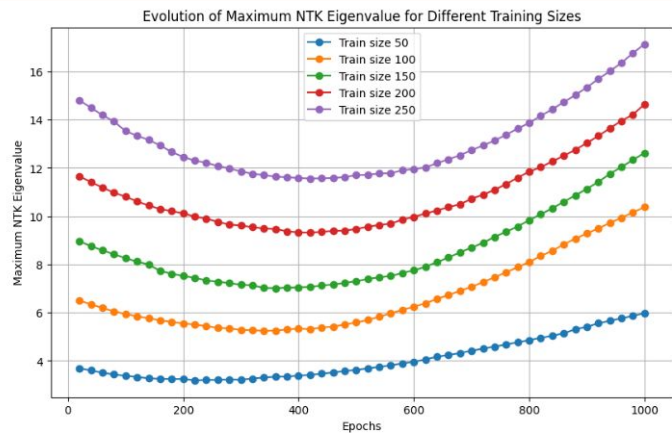


Evolution of Maximum NTK - No Regularisation

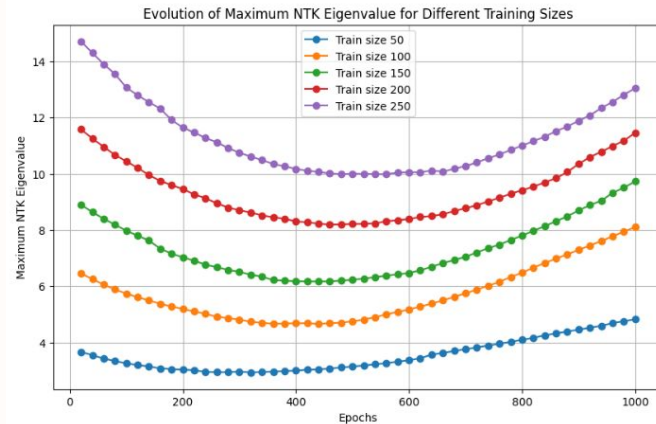


Evolution of Maximum NTK - No Regularisation

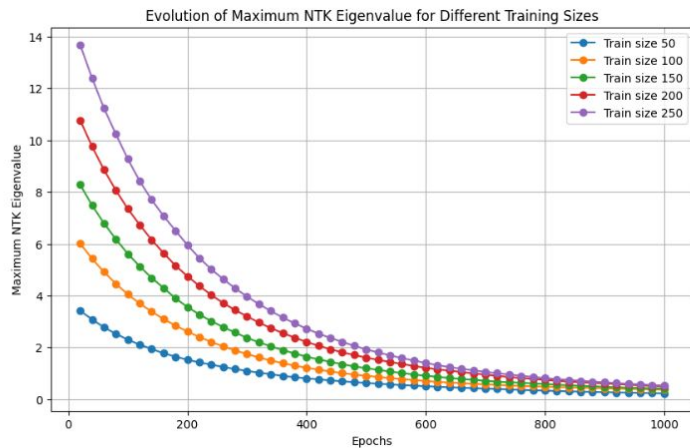
- As Training Data increases
- Number of Data points a Neural Network sees in a single Epoch increases
- Implies Greater change in Parameters
- So the magnitude maximum of Eigenvalue is also large for large training data



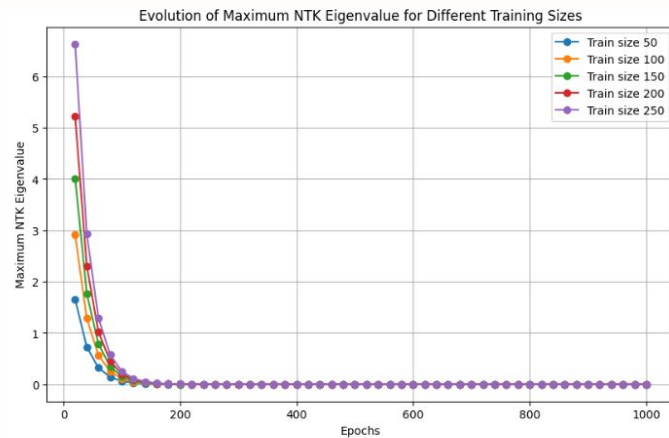
L2 - 0.0001



L2 - 0.001



L2 - 0.01



L2 - 0.1

Effect of L2 Regularisation

Reduction in Over-fitting

- L2-regularization prevents overfitting on Training Data
- As L2-Norm increases, overfitting graph after minima in the plot reduces.

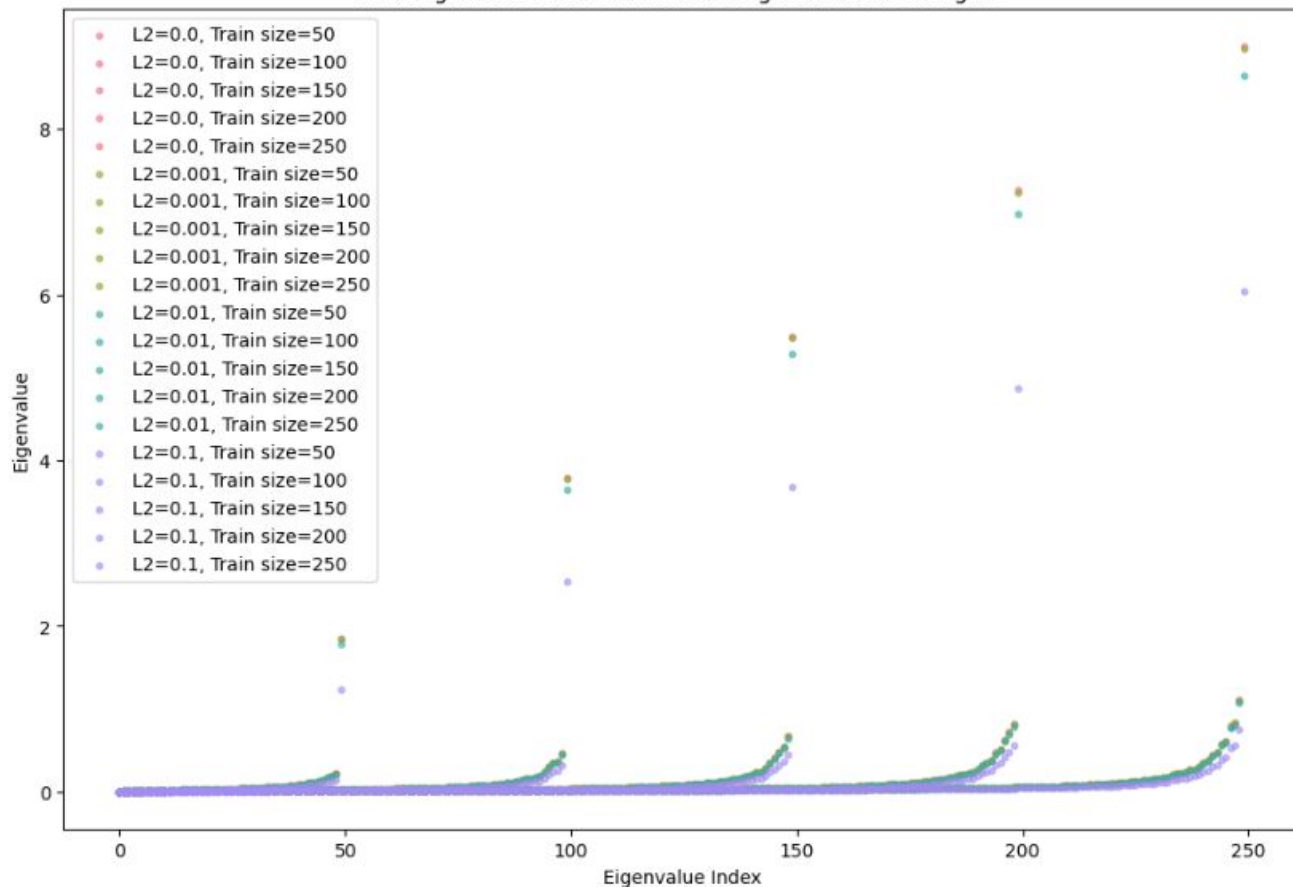
Increasing L2-Coefficient results in a sharper decline

- Without regularization, the network can freely adjust its parameters, leading to a slower, more smooth evolution of the NTK eigenvalues over training.
- With higher regularization, the network is more heavily constrained on size of parameters, causing it to make more quick adjustments in the parameters where it can still change (those with large eigenvalues).
- The regularization pushes the network towards solutions that are less flexible, which leads to sharper, more rapid changes in the maximum eigenvalue over epochs.

The figure is a scatter plot titled "NTK Eigenvalues for Different L2 Regularization Strengths". The x-axis is labeled "Eigenvalue Index" and ranges from 0 to 250. The y-axis is labeled "Eigenvalue" and ranges from 0 to 8. The plot displays data points for different combinations of L2 regularization strength and training size. The legend indicates the following series:

- L2=0.0, Train size=50 (pink)
- L2=0.0, Train size=100 (light pink)
- L2=0.0, Train size=150 (light green)
- L2=0.0, Train size=200 (green)
- L2=0.0, Train size=250 (dark green)
- L2=0.001, Train size=50 (light yellow)
- L2=0.001, Train size=100 (yellow)
- L2=0.001, Train size=150 (light green)
- L2=0.001, Train size=200 (green)
- L2=0.001, Train size=250 (dark green)
- L2=0.01, Train size=50 (teal)
- L2=0.01, Train size=100 (light blue)
- L2=0.01, Train size=150 (blue)
- L2=0.01, Train size=200 (dark blue)
- L2=0.01, Train size=250 (purple)
- L2=0.1, Train size=50 (light purple)
- L2=0.1, Train size=100 (purple)
- L2=0.1, Train size=150 (dark purple)
- L2=0.1, Train size=200 (very dark purple)
- L2=0.1, Train size=250 (black)

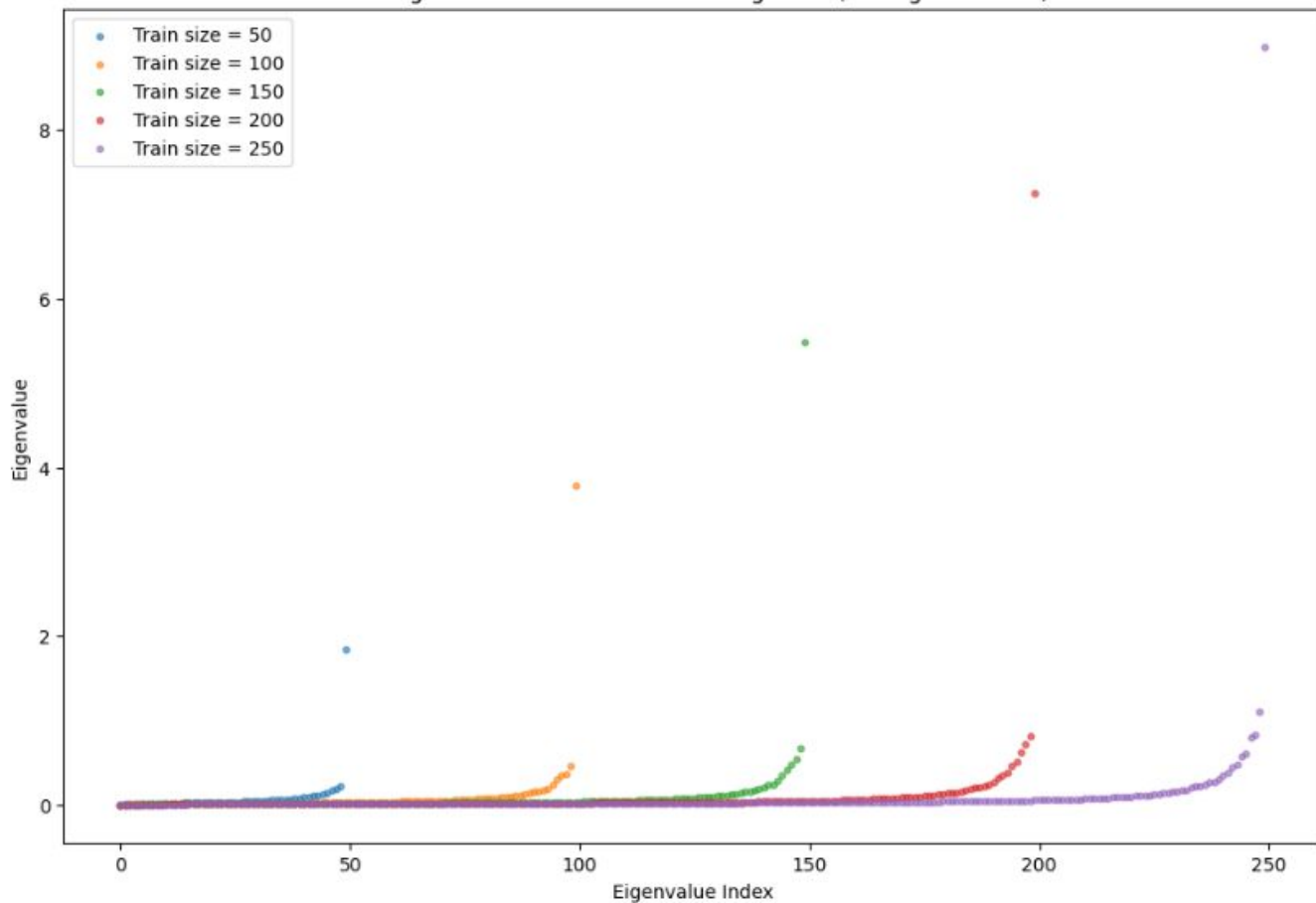
The plot shows that for L2=0.0, the eigenvalues are very low (near 0) for all training sizes. As L2 increases to 0.001, 0.01, and 0.1, the eigenvalues for smaller training sizes (50, 100, 150) increase significantly, while the eigenvalues for larger training sizes (200, 250) remain relatively low. The eigenvalues for L2=0.1 are the highest, reaching values up to 8 for the smallest training size (50) at index 250.

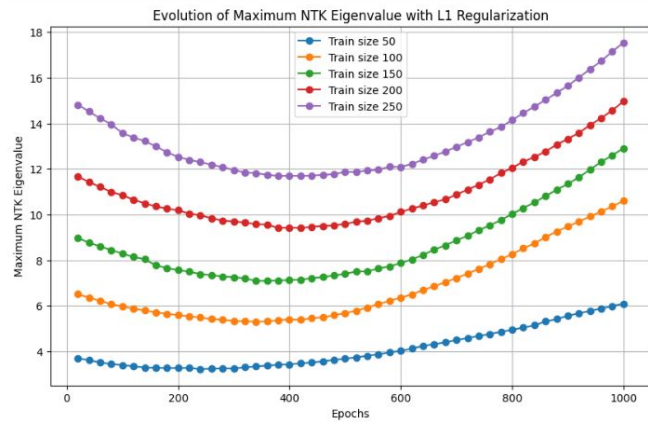


Intuitive explanation of collinearity

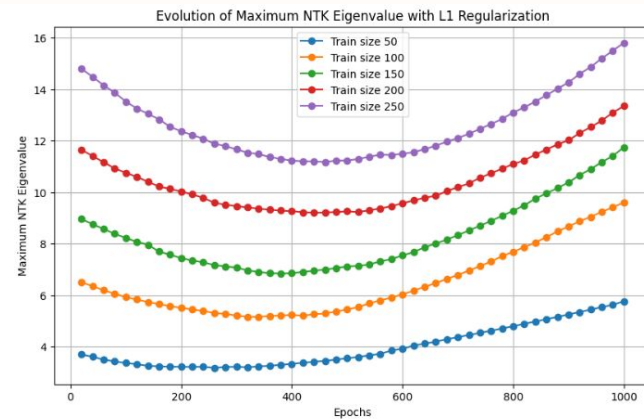
- The NTK matrix grows with training size: $n \times n$.
- Its Trace of NTK Matrix= sum of eigenvalues and it scales roughly linearly with n .
- Most of that SUM is concentrated in top eigen directions.
- Hence, the maximum eigenvalue increases approximately linearly with training size.

NTK Eigenvalues for Different Training Sizes (L1 Regularization)

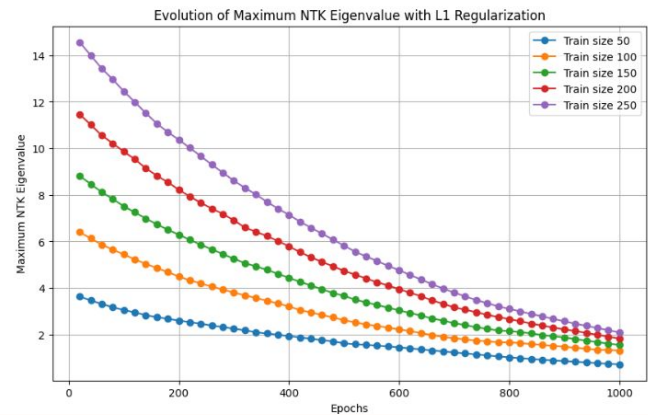




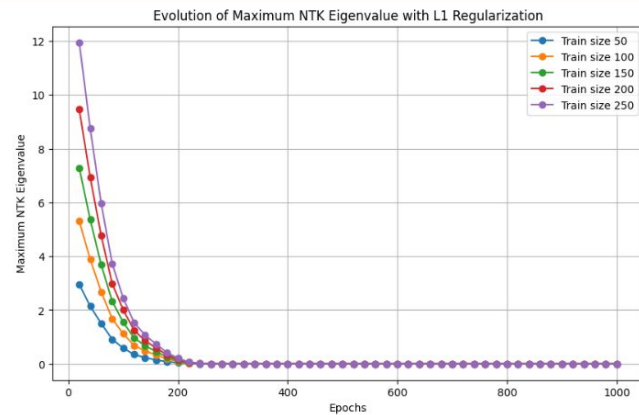
L1 - 0.0001



L1 - 0.001

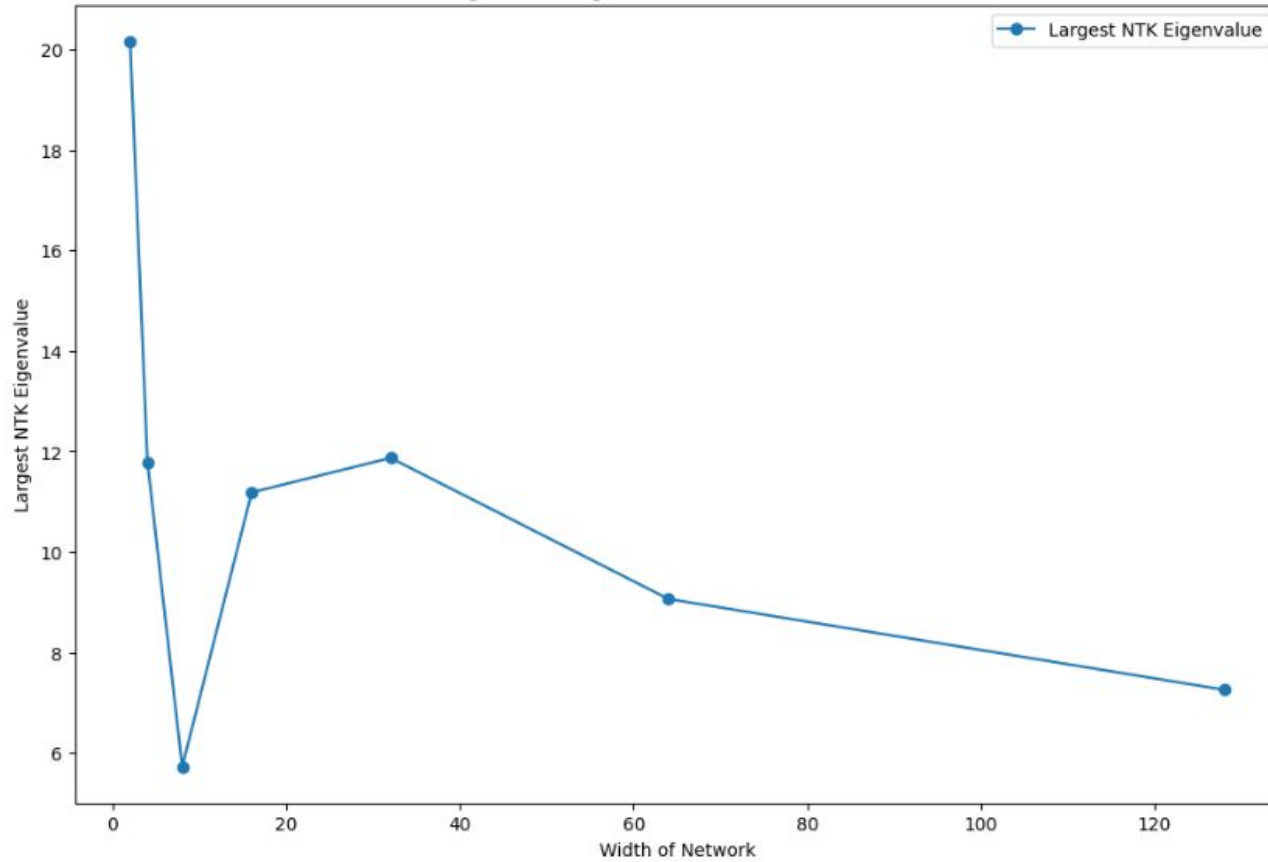


L1 - 0.01



L1 - 0.1

Largest NTK Eigenvalue vs. Width of Network



- **Width = 1:** Rank-1 NTK \rightarrow huge top eigenvalue, one dominant direction, highly ill-conditioned.
- **Width ≈ 8 :** When you only have a handful of neurons (say ≈ 8), each neuron's feature map is basically a random vector in function-space. When you sum up just a few of these random vectors to build your NTK, they don't all line up in one "strong" direction—instead their contributions point off in different directions and partially cancel or spread out.
- **Width 16–32:** Law of large numbers kicks in \rightarrow gradients align, top eigenvalue rises again.

The **Law of Large Numbers (LLN)** is a statistical principle that says:

When you average a large number of random variables, their average gets very close to the expected (true) value.

In context of neural networks:

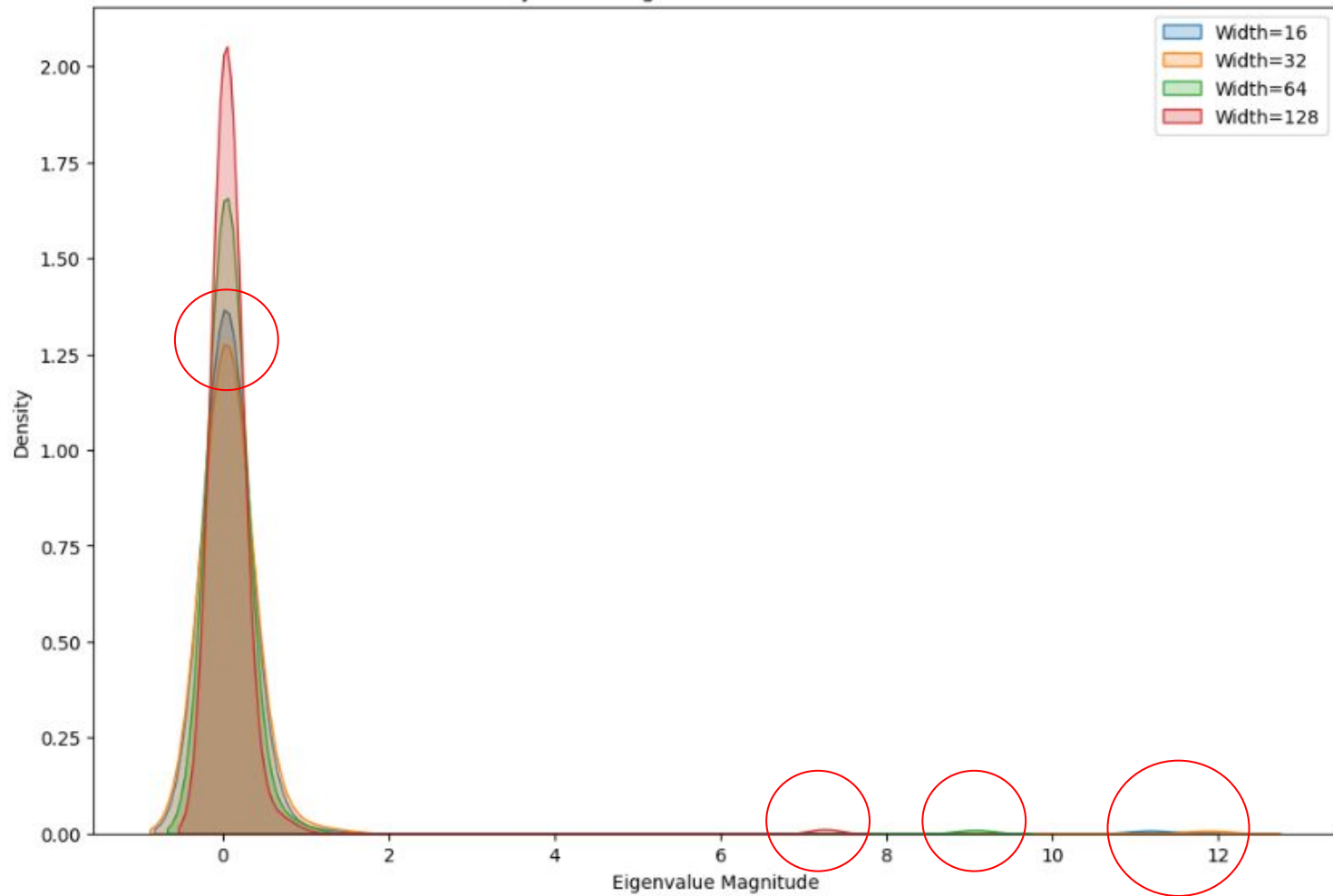
When we have dozens of neurons, each one still contributes a random “direction” in function-space, but now we’re summing so many of them that their average settles very close to the true mean direction (by the law of large numbers).

In short:

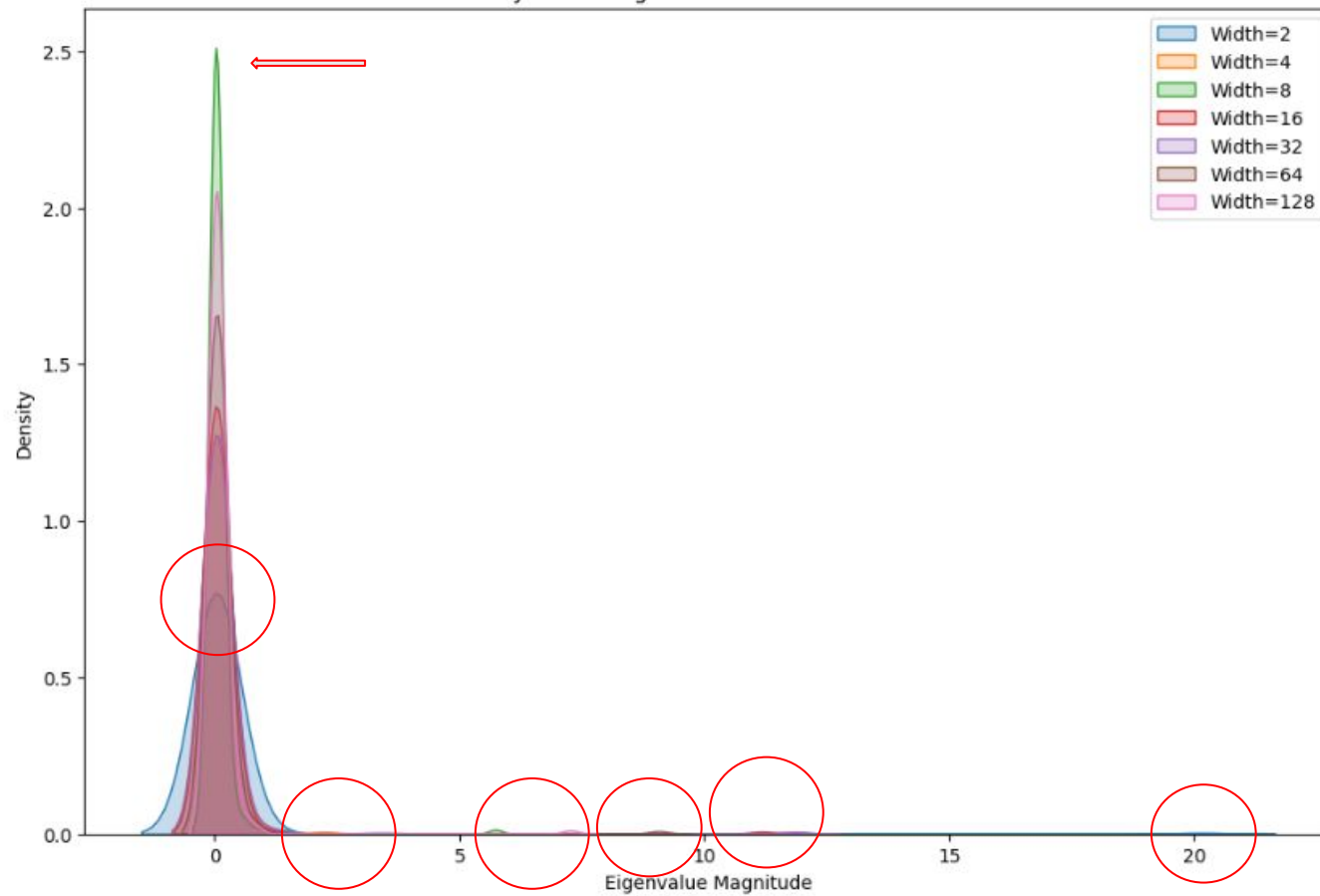
More neurons → more averaging → more alignment → stronger dominant eigenvalue.

- **Beyond Width ≈ 32 :**
 - a. NTK stabilizes — barely changes during training.
 - b. Largest eigenvalue flattens (~ 7.3 at width 128) → stable, predictable learning.
 - c. Smooth, balanced updates across directions → better generalization.

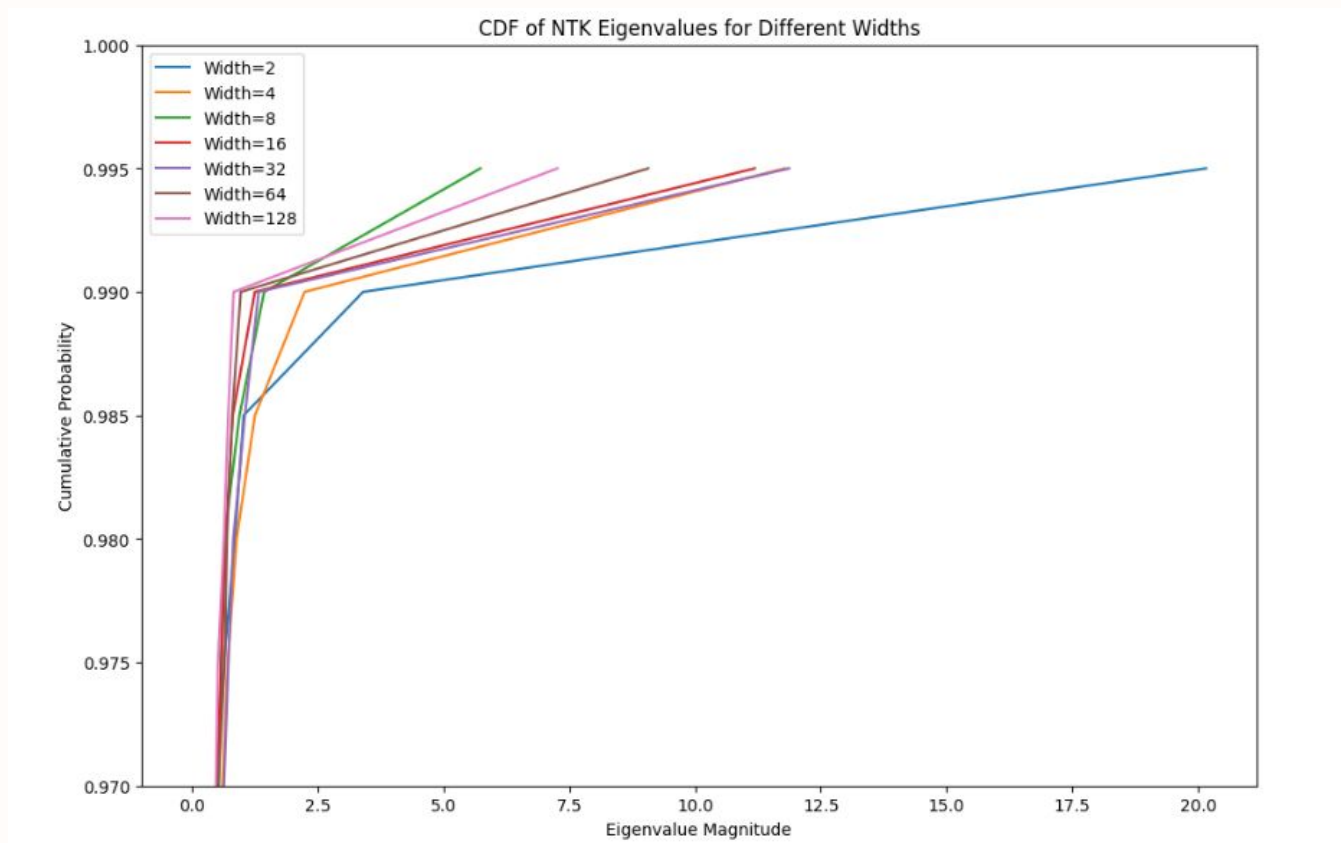
Density of NTK Eigenvalues for Different Widths



Density of NTK Eigenvalues for Different Widths



- A smaller largest eigenvalue indicates that the spectrum is more densely concentrated in smaller eigenvalues, resulting in a cumulative distribution function (CDF) that saturates earlier.
- As the total sum of eigen values remain same (trace of the NTK matrix), so if the largest eigen (λ_1) is very large, then $\lambda_2, \lambda_3, \dots \lambda_N$ must be small to compensate.
- Similarly, if the largest eigen (λ_1) value is small, the rest of the spectrum— λ_2, λ_3 , and so on becomes denser and more evenly distributed.



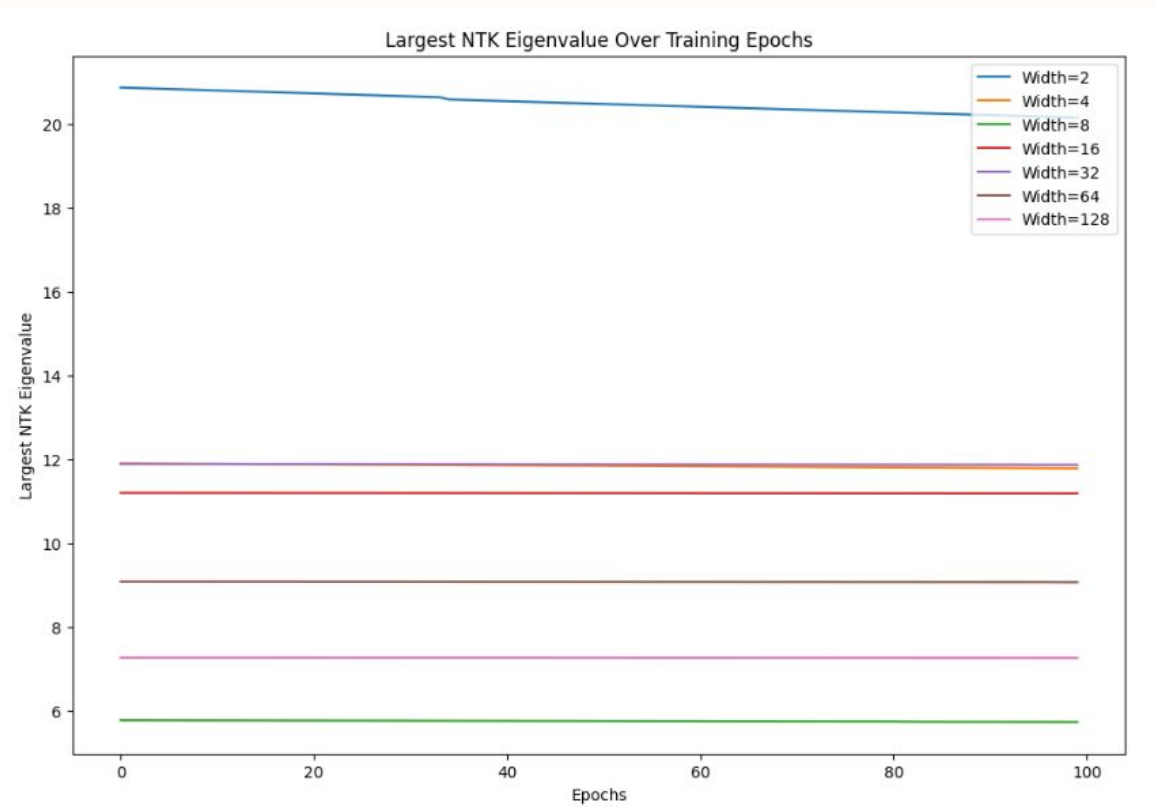
**The eigenvalue distributions that are
more spread end last**

When we have a relatively larger maximum eigenvalue

- The CDF **rises slowly** at first, because you're summing many small eigenvalues.
- Then it **jumps sharply near λ_1** , because λ_1 holds a big chunk of the total.
- So the **CDF saturates late** — it doesn't reach 1 until much further along the x-axis (eigenvalue magnitude).

In contrast, when we have a relatively smaller eigenvalue

- Since all eigenvalues are relatively small and similar, each contributes **more evenly** to the total.
- As you increase the eigenvalue threshold, the CDF **rises quickly** — because you're **accumulating meaningful mass at lower eigenvalues**.
- So the **CDF saturates early** — it approaches 1 at a much smaller eigenvalue magnitude.



The trend continues over training epoch

Key Findings

Maximum Eigenvalue of the NTK matrix decreases as we apply regularization.

As the training progresses, the Maximum Eigenvalue of the NTK matrix decreases, attains a minima and then increases, showing overfitting on training data.

As Training Data Increases, Maximum eigenvalue increases signifying greater change of parameters.

Larger first eigenvalue \rightarrow less dense for small eigenvalues \rightarrow cdf saturates late.

Increasing L2-Norm results in faster decline of maximum NTK eigenvalue over training.

Limitations of our Project

Computation Complexity of NTK for a given Training data and Model parameters-

Due to this we were unable to calculate NTK matrix for large Neural Networks.

Size of empirical- NTK varies as square of Training Data. So even for results for simple neural networks over large training data requires large Memory.

NTK Assumes Infinite-Width Networks :Most NTK theory holds strictly in the **infinite-width limit**. In practice, we work with **finite-width** networks, so theoretical predictions may not align perfectly with empirical results.

Computing the NTK eigenvalues involves **eigendecomposition**, which is $O(n^3)$ in time and $O(n^2)$ in space. This severely limits scaling to larger datasets.

Where can this be used




Use NTK scaling to **estimate behavior of wide models without training** them.



Compare NTKs to pick architectures with better learning dynamics.



Top eigenvalues tell which patterns the model learns fastest.



NTK-based results are helpful for **researchers and scholars** to study neural network learning, generalization, and optimization analytically.

Scope for Future Work



Explore how NTK behavior scales with **real-world, high-dimensional data** and on large datasets.



Study how techniques like **dropout, weight decay, or batch norm** affect the NTK spectrum.



Analyze **NTK dynamics in CNNs, Transformers, and RNNs** for broader applicability.



Improve computational methods for NTK estimation in large models.

References

Golikov et al., Neural Tangent Kernel: A Survey, 2022

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J., & Schoenholz, S. S. (2020). **Neural Tangents: Fast and Easy Infinite Neural Networks in Python.** In Proceedings of the International Conference on Learning Representations (ICLR).

Huang, J., & Yau, H.-T. (2019). **Dynamics of Deep Neural Networks and Neural Tangent Hierarchy.** arXiv:1909.08156 [cs.LG]. Available at: <https://arxiv.org/abs/1909.08156>.

Vakili, S., Bromberg, M., Garcia, J., Shiu, D., & Bernacchia, A. (2021). Uniform Generalization Bounds for Overparameterized Neural Networks. arXiv:2109.06099 [cs.LG]. Available at: <https://arxiv.org/abs/2109.06099>.