Delft University of Technology

Faculty Electrical Engineering, Mathematics and Computer Science

WI4450 Special Topics in Computational Science and Engineering: NTK analysis for finite width and depth

Thai Ha Bui

Yazan Mash'Al

June 18, 2025

Contents

1 Introduction

In this report, we explore the Neural Tangent Kernel (NTK), a tool to analyze ANN training dynamics Jacot et al. (2020). In the infinite-width limit, NTK shows ANN training with squared error loss acts as kernel regression with a fixed kernel, limiting learning to the initial function space. We investigate NTK behavior in finite-width and finite-depth ANNs to understand their generalization Zhang et al. (2021).

1.1 Network Architecture and Notation

This study focuses on fully-connected feedforward neural networks (FNNs). The architectural parameters and notation used throughout this work are defined in Table 1 in Appendix A.

The network computes a function $f(x;\theta)$ by recursively applying transformations:

$$\begin{aligned} x^{(0)} &:= x \\ y^{(i)} &:= W^{(i)}x^{(i-1)} + b^{(i)}, \quad \text{for } i = 1, \dots, L_h \\ x^{(i)} &:= \sigma(y^{(i)}), \qquad\qquad \text{for } i = 1, \dots, L_h \\ f(x;\theta) = y^{(L_h+1)} &:= W^{(L_h+1)}x^{(L_h)} + b^{(L_h+1)} \end{aligned}$$

1.2 Organisation of this paper

This paper is organized as follows. Section 3 presents our research questions on the NTK's stochastic behavior, scaling with depth-to-width ratio β, activation function effects, and data-dependent structure in finite regimes. Section 2 reviews NTK literature, including infinite-width regimes, finite-size corrections, and activation function roles, laying the theoretical groundwork. Section 4 describes our methodology, covering theoretical results, network architectures, training setups, NTK computation, and evaluation metrics. Section 5 analyzes empirical NTK behavior across network configurations and activation functions for standard networks and PINNs. Section 6 discusses generalization implications, NTK limitations, and future research directions. Appendices B, D, C, E, F, and G provide background theory, preliminary NTK analysis, architecture details, and additional experiments.

2 Literature Review

2.1 NTK and the Infinite-Width Regime

The NTK was first introduced by Jacot et al. (2020) as a framework for understanding the training dynamics of neural networks. In the limit where network width tends to infinity while depth remains fixed, the NTK becomes deterministic and constant throughout training, which is usually called the "lazy training" regime, where the evolution of the network function under gradient descent simplifies to kernel regression with a fixed kernel, specifically the NTK at initialisation.

Subsequent works, including Lee et al. (2020) and Arora et al. (2019), rigorously confirmed that infinitely wide networks exhibit "lazy training". In this setting, internal representations do not evolve significantly, and learning is driven by a linear model in function space governed by the NTK. These results provided important theoretical insights into convergence guarantees and generalization in over-parameterized networks.

However, while significant at the time, the infinite-width theory does not fully capture the behavior of real-world networks, which have finite depth and width and often achieve better-than-predicted performance through feature learning.

2.2 Finite Depth and Width Corrections to the NTK

Recognizing the limitations of infinite-width assumptions, Hanin and Nica (2019) investigated NTK behavior in networks where both depth d and width n are finite and large. Their analysis provides critical corrections to the infinite-width theory, focusing on how the NTK behaves when the ratio d/n is non-negligible.

Key findings from their work include:

1. **Variance at Initialization**: The normalized variance of the NTK grows exponentially with d/n. Thus, even in highly overparameterized networks, the NTK remains stochastic unless depth is much smaller than width. This is the result of Theorem 1 in section 4.1.2.
2. **NTK Evolution During Training**: Unlike in the infinite-width regime, the NTK is not static during training for deep and wide networks. The mean update of the NTK after a single SGD step also scales exponentially with d/n, indicating potential for data-dependent feature learning even in settings close to lazy training. This is the result of Theorem 2 in section 4.1.3.

This work challenges the traditional view that large neural networks merely perform kernel regression, suggesting instead that practical architectures can enter an intermediate regime where some feature learning coexists with kernel-like behavior.

2.3 Activation Functions and NTK Behavior

While much of the NTK literature focuses on ReLU activations due to their analytical tractability, Hanin and Nica's analysis leaves open the question of how different activation functions might influence finite-width NTK dynamics. Activation functions impact the recursion defining the NTK and could thus alter its scaling properties, variance, and training-time evolution. Investigating nonlinearities such as sigmoid, and others could reveal new regimes of behavior, motivating part of the present project's focus.

2.4 Relevance to This Work

The literature reviewed highlights the gap between idealized infinite-width analyses and the realities of training practical networks. This project seeks to bridge that gap by:

- Performing detailed numerical studies of NTK scaling behavior with finite d and n,
- Investigating how different activation functions affect NTK properties,
- Providing results that could aid selection of architectures that balance depth, width, and activation function choices.

3 Research Questions

As mentioned earlier, we seek to build upon the work of Hanin and Nica (2019) by conducting a detailed empirical investigation into the behavior of neural networks in the finite-width and finite-depth regime. The goal is to explore how different architectural choices influence the behavior of the NTK, and ultimately, the training dynamics and generalization capabilities of these networks.

This analysis will be carried out for two classes of networks:

- A traditional fully-connected feedforward neural network
- A physics-informed neural network (PINN) designed to solve the Poisson partial differential equations

Since both networks differ primarily in their loss functions, we hypothesise that NTK behaviour will remain largely consistent across both. However, subtle differences might emerge due to the structural differences in learning objectives.

The following research questions will guide our investigation:

1. **Does the NTK exhibit stochastic behavior at finite width and depth?**
2. **How does the NTK scale with the depth-to-width ratio $\beta = \frac{d}{n}$?**
3. **What is the influence of activation function choice on NTK behavior in finite networks?**
4. **Does the NTK reflect data-dependent structure in the finite regime?**

4 Methodology

The experiments are built using JAX by Bradbury et al. (2018) and the 'neural-tangents' library by Novak et al. (2020) for NTK-specific computations. We implemented a class `NTKAnalyzer` and use this class in our Python notebook experiments.

4.1 Theoretical results

4.1.1 Finite Width and Depth Corrections

While the infinite-width NTK theory provides profound insights, practical neural networks are always of finite size. Hanin and Nica (2019) investigated the crucial finite depth and width corrections to the NTK for randomly initialized ReLU networks. Their work revealed that the behavior of finite networks can deviate significantly from the idealized infinite-width limit. These finite-size effects are largely governed by a parameter β, which quantifies the "finiteness" of the network's hidden layers (defined in Section 1.1). Two key theoretical results from their work, which form a central part of our numerical verification, are stated in sections 4.1.2 and 4.1.3.

4.1.2 Normalized Second Moment of NTK (Theorem 1 Hanin and Nica (2019))

For a ReLU network with d layers having trainable parameters, input dimension n_0, and under specific initialization schemes, the expected value of the on-diagonal NTK for an input x is given by:

$$\mathbb{E}[K_{\mathcal{N}}(x,x)] = d\left(\frac{1}{2} + \frac{\|x\|_2^2}{n_0}\right) \tag{1}$$

Furthermore, the NTK is stochastic for finite $\beta > 0$. The normalized second moment, which quantifies this stochasticity, scales approximately as:

$$\frac{\mathbb{E}[K_{\mathcal{N}}(x,x)^2]}{(\mathbb{E}[K_{\mathcal{N}}(x,x)])^2} \approx \exp(C\cdot\beta)(1 + O(\beta/n_{\text{hid}})) \tag{2}$$

where n_{hid} is a characteristic hidden width, $C \approx 5$ for their specific ReLU setup, and $\beta = d/n$.

This ratio quantifies the stochasticity of $K_{\mathcal{N}}(x,x)$ relative to its mean, which can be easily seen from:

$$\text{Var}[K_{\mathcal{N}}(x,x)] = \mathbb{E}[K_{\mathcal{N}}(x,x)^2] - (\mathbb{E}[K_{\mathcal{N}}(x,x)])^2$$
$$\implies \frac{\text{Var}[K_{\mathcal{N}}(x,x)]}{(\mathbb{E}[K_{\mathcal{N}}(x,x)])^2} = \frac{\mathbb{E}[K_{\mathcal{N}}(x,x)^2]}{(\mathbb{E}[K_{\mathcal{N}}(x,x)])^2} - 1$$

Thus a value greater than 1 indicates $\text{Var}[K_{\mathcal{N}}(x,x)] > 0$, while values close to 1 indicates a nearly deterministic NTK.

4.1.3 NTK Evolution During Training (Theorem 2 Hanin and Nica (2019))

For finite networks ($\beta > 0$), the NTK $K_{\mathcal{N}}(x,x';\theta(t))$ generally evolves during training. Hanin and Nica (2019) showed that the expected change in the on-diagonal NTK after one step of Stochastic Gradient Descent (SGD). The scaling of this expected relative change is approximately:

$$\frac{\mathbb{E}[\Delta K_{\mathcal{N}}(x,x)]}{\mathbb{E}[K_{\mathcal{N}}(x,x)]} \propto \frac{d\cdot\beta}{n_0}\cdot\exp(C\cdot\beta)(1 + O(\beta/n_{hid})) \tag{3}$$

where $\Delta K_{\mathcal{N}}(x,x) = K_{\mathcal{N}}(x,x;\theta_{t_1}) - K_{\mathcal{N}}(x,x;\theta_{t_0})$. This non-zero expected change signifies that the kernel is not "frozen" and that the network can learn data-dependent features.

4.2 Preliminary Experiment with Infinite NTK

To initiate our investigation into the behavior of the NTK for finite-width neural networks, we conducted a preliminary experiment at infinite width, as defined by Jacot et al. (2020). This experiment serves as a foundational step to establish a baseline understanding of NTK behavior before delving into the more complex analysis of finite architectures. Results are in Appendix D.

4.3 Theoretical verification

To verify Theorems 1 and 2 from Hanin and Nica (2019), we compute the on-diagonal NTK $K_{\mathcal{N}}(x,x)$ for a arbitrary input $x \in \mathbb{R}^{n_0}$. The NTK is calculated using (6) and JAX's automatic differentiation and `neural_tangents` for efficient kernel evaluation. We perform Monte Carlo simulations (100 runs) to estimate:

- The normalized second moment $\frac{\mathbb{E}[K_{\mathcal{N}}(x,x)^2]}{\mathbb{E}[K_{\mathcal{N}}(x,x)]^2}$, compared against $\exp(5\beta)(1 + O(\beta/n_{\text{hid}}))$.
- The relative NTK change after SGD step $\frac{\mathbb{E}[\Delta K_{\mathcal{N}}(x,x)]}{\mathbb{E}[K_{\mathcal{N}}(x,x)]}$ compared against $\frac{d\cdot\beta}{n_0} \cdot \exp(5\beta)(1 + O(\beta/n_{hid}))$.

To verify Theorem 2 (stated in 4.1.3), we simulate a single step of stochastic gradient descent (SGD) with a batch of size 1, using the square loss $\mathcal{L}(x) = \frac{1}{2}(\mathcal{N}(x) - \mathcal{N}_*(x))^2$. The NTK update $\Delta K_{\mathcal{N}}(x,x)$ is computed, and its expected value $\mathbb{E}[\Delta K_{\mathcal{N}}(x,x)]$ is normalized by $\mathbb{E}[K_{\mathcal{N}}(x,x)]$ to compare with the theoretical prediction for ReLU networks [1].

In 4.1.3 the learning rate is implicitly absorbed into the derivation, and the result is specific to ReLU networks with square loss. In contrast, our experimental computation adapts this to practical training settings, resulting in:

$$\frac{\mathbb{E}[\Delta K_{\mathcal{N}}(x,x)]}{\mathbb{E}[K_{\mathcal{N}}(x,x)]} \propto \frac{d\beta}{n_0} \exp(C\beta)\lambda_{\text{lr}} \tag{4}$$

where C depends on the activation function (e.g., $C = 5$ for ReLU), $\lambda_{\text{lr}} = 0.01$ is our explicit learning rate. This generalization introduces explicit scaling with the learning rate and additional factors not emphasized in the paper, aligning with the theoretical result for ReLU and square loss when adjusted accordingly.

We explore a range of network configurations, varying d and n_{hid}, to test different $\beta = d/n_{\text{hid}}$ regimes, Appendix C describes all the values for d and n_{hid} conducted in this study. For Theorem 1, a log-linear plot was used to visualize the normalized second moment $\mathbb{E}[K(x,x)^2]/(\mathbb{E}[K(x,x)])^2$ against β, with empirical data points plotted alongside a theoretical leading-order prediction curve $\exp(C\beta)$. For Theorem 2, a log-log parity plot was employed to compare the empirical relative NTK change $|\mathbb{E}[\Delta K]/\mathbb{E}[K]|$ against the theoretical scaling term, with a $y = x$ line indicating perfect agreement and outliers highlighted for large β values.

4.4 Exploring different activation functions

To assess how our NTK-based metrics change under different nonlinearities, we will compare three widely used activations: the Rectified Linear Unit (ReLU), the Gaussian Error Linear Unit (GeLU), and a sigmoid-like [2] approximation ("Sigmoid_like"), which we will refer to as *sigmoid* in this report. We denote by $\phi : \mathbb{R} \to \mathbb{R}$ the activation applied elementwise, we define

$$\begin{aligned}
\phi_{\text{ReLU}}(x) &= \max(0, x), \\
\phi_{\text{GELU}}(x) &= x\,\Phi(x) = x\,\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right], \\
\phi_{\text{Sigmoid_like}}(x) &= \frac{1}{2}\operatorname{erf}\left(\frac{x}{2.4020563531719796}\right) + \frac{1}{2}, \\
&\text{where } \Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\,dt \text{ and } \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt.
\end{aligned}$$

[1] In addition to the one-step SGD, we conducted a *supplementary experiment* to observe the kernel's long term evolution throughout a complete training cycle. The code for this analysis is provided in the supplementary notebook, `full_training_analysis.ipynb`.

[2] Stax (the API underlying Neural Tangents) does not include a built-in logistic sigmoid activation, but it does expose an erf primitive via the 'Erf(a,b,c)' layer and the `sigmoid_like` is approximated by a scaled and shifted error function $\frac{1}{2}\,\texttt{Erf}\left(\frac{x}{\alpha}\right) + \frac{1}{2}$, where the constant $\alpha \approx 2.4020563531719796$ is chosen to minimize the squared loss between this function and the ground truth sigmoid is minimized on the interval $[-5, 5]$. See more: https://github.com/google/neural-tangents/blob/main/neural_tangents/_src/stax/elementwise.py.

For comparing different activation functions, log-linear plots were used to display the normalized second moment and the magnitude of relative NTK change against β, with separate curves for ReLU, GeLU, and Sigmoid.

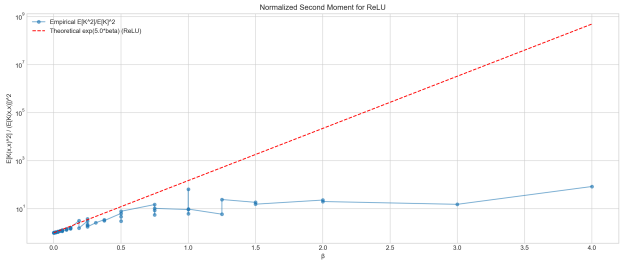
5 Results

5.1 Finite NTK Analysis

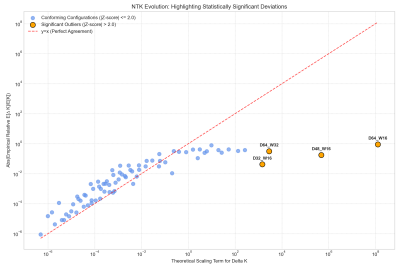
5.1.1 ReLU

Theorem 1

Figure 1a illustrates the stochasticity of the initial on-diagonal NTK for various ReLU network configurations, quantified by the normalized second moment $E[K(x,x)^2]/(E[K(x,x)])^2$ described in 4.1.2. For smaller values of β, the empirical data points generally exhibit close alignment with the theoretical leading-order prediction $\exp(C\beta)$ [3]. This agreement in the small-β regime empirically confirms the dominant exponential increase in NTK stochasticity as finite-size effects begin to manifest.



(a) Normalized second moment for ReLU networks plotted against β.



(b) Parity log log plot of empirical relative NTK change $\left|\frac{\mathbb{E}[\Delta K]}{\mathbb{E}[K]}\right|$ vs theoretical scaling for ReLU.

Figure 1: Comparison of NTK relative change and normalized second moment for ReLU networks.

As β increases further, a divergence between the empirical and theoretical results and is observed. This deviation is anticipated, as the more complete theoretical expression for the normalized second moment includes higher-order correction terms, scaling approximately as $\exp(C\beta)(1 + O(\beta/n_{\text{hid}}))$. Since the plot is log-linear in y-axis, we have that:

$$\log\left[\exp(C\beta)(1 + O(\beta/n_{\text{hid}}))\right] = C\beta + \log(1 + O(\beta/n_{\text{hid}}))$$

So, the $C\beta$ term corresponds to the linear trend expected from the leading-order exponential prediction $\exp(C\beta)$, while the $\log(1 + O(\beta/n_{\text{hid}}))$ term represents the deviation from this straight line due to higher-order corrections. As β increases, this logarithmic correction term becomes more influential, accounting for any observed divergence of the empirical data.

Theorem 2

The scatter plot 1b compares the theoretical scaling term for the ΔK, with the empirical $\left|\frac{\mathbb{E}[\Delta K]}{\mathbb{E}[K]}\right|$. This supports the theoretical scaling's accuracy for NTK evolution.

A closer inspection reveals that for large β the empirical data points systematically diverge from the $y = x$ line. The theoretical scaling term in (4) is a practical, leading-order approximation of the kernel's evolution. A more complete, albeit more complex, theoretical description would include higher-order correction terms that are not captured in this simplified scaling law, we refer to Hanin and Nica (2019) for more details.

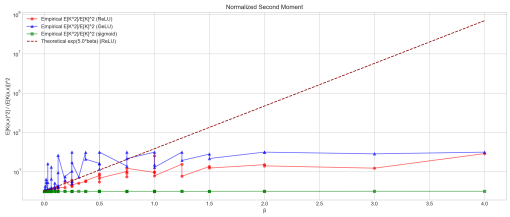
[3] with $C \approx 5.0$ for ReLU networks proven by Hanin and Nica (2019)

Similarly for the observations from theorem 1, for small β, the error term is negligible, causing $\log(1 + O(\beta/n_{\text{hid}})) \approx 0$ and forcing the points to lie close to the $y = x$ diagonal. But when β becomes larger. This term accounts for the observed vertical displacement of these outliers from the line of perfect agreement.

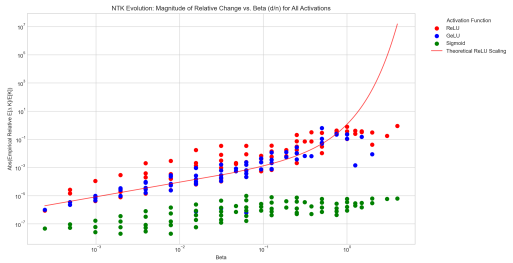
5.1.2 Other activation functions

Theorem 1

Figure 2a shows how different activation functions (ReLU, GeLU, and Sigmoid) affect the initial randomness of the NTK. The results reveal clear differences between these activations. For networks using ReLU and GeLU, the NTK becomes much more stochastic as β increases, which indicates stronger finite-size effects. For example, across the tested β range, this randomness measure for ReLU networks increased from approximately 1.00 to 83.99, and for GeLU networks from 1.00 to 100.0. As explained in Section 5.1.1, deviations from the theoretical line at larger β can be due to to higher-order correction terms. In sharp contrast, the Sigmoid activation function resulted in very stable NTK behavior: its randomness measure stayed extremely close to 1.00 (with a maximum of only 1.03) for all tested β values. This suggests that, unlike ReLU and GeLU, the NTK for these Sigmoid networks remains nearly deterministic, and its level of randomness is largely unaffected by changes in β.



(a) Normalized second moment plotted against β.



(b) The effect of network geometry β on the magnitude of NTK evolution

Figure 2: Comparison of second moment and NTK evolution for ReLU, GeLU, Sigmoid in red, blue and green, respectively.

Theorem 2

Figure 2b shows all three activation functions exhibit a qualitatively similar trend: $|\mathbb{E}[\Delta K]/\mathbb{E}[K]|$ increases monotonically with β. This observation suggests that the role of β as a parameter controlling the degree of feature learning (i.e., kernel evolution) is not merely an artifact of ReLU networks but a more fundamental property of deep, finite-width architectures. For all tested functions, as networks become deeper or narrower (increasing β), they move further away from the fixed-kernel lazy regime, and their internal representations adapt more significantly during training. The behaviors of ReLU and GeLU are quantitatively very similar, which is expected as GeLU is a smooth approximation of the ReLU function.

The Sigmoid activation

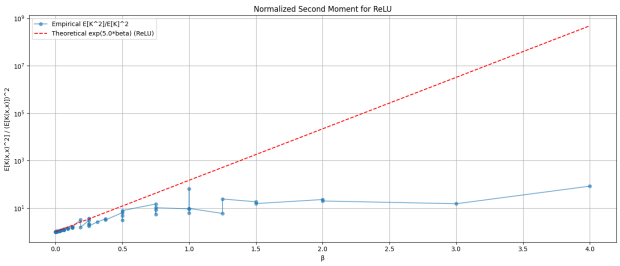
There is an obvious quantitative difference in the stability of the kernel for the Sigmoid activation. For any given value of β, the NTK evolves by several orders of magnitude less than for ReLU and GeLU networks, this is visible from the results of both Theorem 1 and 2. This provides strong empirical evidence that the Sigmoid NTK is significantly more stable during training. The reason for this stability lies in the well known saturating nature of the Sigmoid function. For inputs with large magnitudes, the derivative of the Sigmoid function approaches zero, leading to the well-known vanishing gradient problem. Since the kernel's evolution is driven by these gradients, their diminished magnitude in Sigmoid networks directly results in a smaller ΔK, and thus a more static kernel compared to non-saturating functions like ReLU.

5.2 Finite NTK Analysis for PINNs

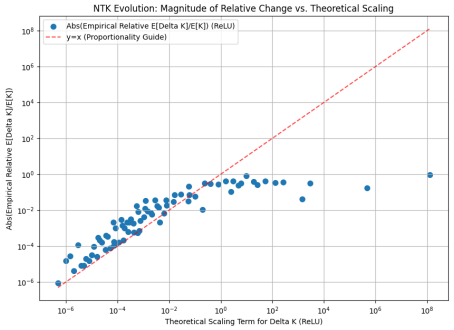
5.2.1 The Effect of β on NTK - ReLU

Theorem 1

We analysed 78 network configurations, given in Appendix C. The ReLU-specific results are shown in Figure 3a, where we plot the normalized second moment $E[K(x,x)^2]/(E[K(x,x)])^2$ against the depth-to-width ratio β for all PINN configurations that used ReLU activation. We focus on ReLU right now in order to compare against the theoretical limits proposed in Hanin and Nica (2019).



(a) Normalized second moment vs. β for ReLU-activated PINNs.



(b) Log-log plot of relative NTK deviation $\left|\frac{\Delta K}{K}\right|$ vs. theoretical scaling for ReLU-activated PINNs.

Figure 3: Comparison of normalized second moment and relative NTK deviation for ReLU-activated PINN configurations across β.

As with standard fully-connected networks, we observe that ReLU-activated PINNs exhibit increasing NTK stochasticity as β grows. For relatively small values of β, the second moment closely follows the leading-order exponential growth predicted by $\exp(C\beta)$, in line with the theory for finite-width ReLU networks Hanin and Nica (2019). This alignment supports the idea that even in the presence of PDE constraints, the core dynamics of NTK stochasticity remain governed by architecture-related parameters. However, as β increases, some divergence from the pure exponential trend emerges. This deviation likely arises from the higher-order corrections described earlier.

Importantly, while ReLU networks demonstrate the theoretically predicted exponential trend in the small-β regime, their utility in PINNs remains limited. As discussed earlier (see Section D.1.2), ReLU networks struggled to learn PDE solutions effectively in our experiments. This is due to the vanishing second derivative of ReLU, which makes it unsuitable for enforcing second-order differential constraints. As a result, despite the appealing theoretical scaling behavior, ReLU may not be a practical choice for PDE-driven architectures such as PINNs.

In summary, the ReLU activation provides a useful benchmark for understanding the relationship between NTK stochasticity and architectural scaling, but its practical shortcomings in solving differential equations motivate the need for smoother activations like Erf or GELU in physics-informed learning settings.

Theorem 2

We now move ahead to Theorem 2. Figure 3b illustrates the empirical relationship between the relative change in the Neural Tangent Kernel (NTK) and the theoretically predicted scaling for ReLU-activated PINNs. The red dashed line corresponds to the identity $y = x$, which serves as a proportionality guide: if the theoretical scaling perfectly matched the empirical magnitude of relative NTK change, all points would lie on this line.

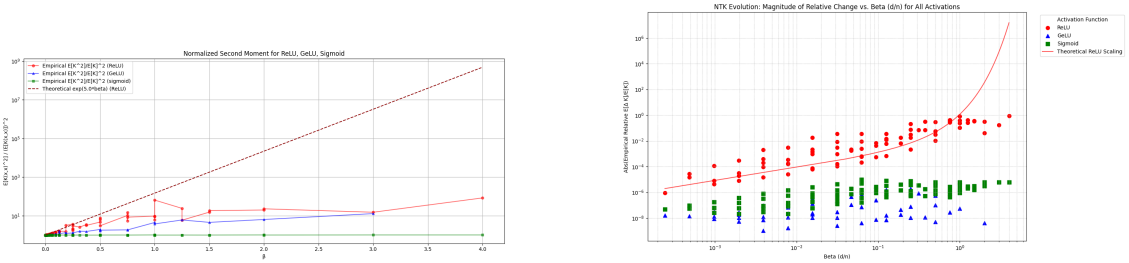
The empirical values closely track the theoretical prediction over several orders of magnitude. This provides strong evidence that the theoretical scaling law derived in Theorem 2 captures the leading-order behavior of NTK changes across different PINN configurations. Systematic deviations from the identity line begin to appear around scaling values of order 10^0 and above. These deviations likely

stem from higher-order corrections or nonlinear effects in NTK evolution that become relevant when the relative change is no longer small.

5.2.2 Other Activation Functions

Theorem 1

The results for the other activations functions, GeLU, and Sigmoid, can be seen in Figure 4a. While the qualitative trends resemble those observed in standard feedforward networks (as discussed in Figure 2a), the magnitude of the observed stochasticity is significantly reduced in the PINN setting.



(a) Normalized second moment vs. β for ReLU, GeLU, and Sigmoidactivated PINNs.

(b) Relative $\Delta K/K$ vs. β for the same PINN configurations.

Figure 4: Comparison of second-moment scaling and NTK evolution across β for ReLU (red), GeLU (blue), and Sigmoid (green) PINNs.

For ReLU-based PINNs, the normalized second moment increased from approximately 1.00 to a maximum of 83.99 across the tested β values. Similarly, GeLU networks exhibited an increase from 1.00 to a peak of 12.95. These results indicate that ReLU and GeLU activations still induce substantial stochastic behavior in finite-width PINNs, particularly as β increases, though to a noticeably lesser extent in the case of GeLU. In contrast, Sigmoid-like activations yielded remarkably stable behavior: their NTK randomness measure remained tightly clustered around 1.00 throughout the entire range of β, with a maximum value of only 1.027. This confirms that Sigmoid-like networks in the PINN setting retain near-deterministic NTK behavior, and are largely immune to finite-size variability.

The overall reduced stochasticity across all activations in PINNs, compared to standard networks, might be attributed to the structure imposed by the physics-informed loss function. Unlike traditional supervised losses that rely solely on data-point supervision, PINNs incorporate differential operators and boundary constraints into their loss, leading to smoother gradients and stronger regularization. As a result, the parameter gradients (and thus the NTK) exhibit less variability across random initializations, effectively suppressing the stochastic amplification typically observed at higher values of β.

Theorem 2

Figure 4b shows the empirical evaluation of Theorem 2 by plotting the relative change against the depth-to-width scaling parameter β for PINNs using ReLU, GeLU, and Sigmoid activations.

As β increases, ReLU networks exhibit a sharp rise in NTK variability, with the relative NTK change increasing by several orders of magnitude. The empirical values grow closely in line with the theoretical scaling, verifying the predictive accuracy of Theorem 2 in this regime. This behavior reflects the sensitivity of ReLU-activated networks to finite-size effects, especially in deeper or narrower architectures.

In contrast, Sigmoid-activated PINNs show significantly reduced NTK variation. While there is still a moderate increase in relative NTK change with β, the growth is slower and more tightly clustered. The empirical values remain substantially below the theoretical upper bound, suggesting that Sigmoid introduces smoothing effects that dampen the sensitivity of the NTK to architecture scaling.

GeLU-activated networks demonstrate the most stable NTK behavior across all β values. The relative NTK change remains consistently low, with no significant increase across the full tested range.

6 Discussion and Conclusion

This paper looked at NTKs, and how they can be used to analyze training dynamics. For a comparison between the FINN and PINN, the analysis can be found in appendix H.

6.1 Research questions

6.1.1 Does the NTK exhibit stochastic behavior at finite width and depth?

Yes, absolutely. In finite-width networks, the NTK is a **random variable** whose value depends on the specific random initialization of the network's weights. Our experiments explicitly measured the variance of the NTK diagonal over 100 different initializations and found this variance to be non-zero.

6.1.2 How does the NTK scale with the depth-to-width ratio β?

The NTK's behavior scales **exponentially** with β. We observed two primary effects. Firest, the **variance** of the kernel at initialization grows exponentially with β, meaning deeper, narrower networks have a much more random and unpredictable kernel. Secondly, the **evolution** of the kernel during training (ΔK) also scales exponentially with β. This confirms that β is the critical parameter governing the network's departure from the stable, "lazy" infinite-width limit. A larger β pushes the network into a more "chaotic" regime where the kernel is both more random and more dynamic.

6.1.3 What is the influence of activation function choice on NTK behavior in finite networks?

Due to time constraints, we were able to experiment with only three types of activation functions (ReLU, GeLU, Sigmoid) and found that the choice of activation function has a quantitative impact. While all tested activations showed the same qualitative trend of increasing instability with β, their magnitudes were different. **Sigmoid**, a saturating activation, produces a significantly more stable NTK. This is because its vanishing gradients suppress the magnitude of kernel evolution, keeping the network closer to the lazy training regime. On the other hand, **ReLU and GeLU**, being non-saturating, allow for much larger gradients and thus a more dynamic kernel, indicating a stronger disposition towards feature learning.

6.1.4 Does the NTK reflect data-dependent structure in the finite regime?

Yes. The NTK is always data-dependent in the sense that the kernel value $K(x, x')$ is a function of the input points x and x'. More importantly, our results show that the **evolution of the kernel is also data-dependent**. The change ΔK is driven by gradients computed on a specific training data point (x, y_*). This means that in the finite regime, the kernel doesn't just measure similarity based on a fixed set of features; it actively adapts its features and structure based on the training data it sees.

6.2 Future Research

In future work, we propose to further leverage the NTK as a tool to further understand the training dynamics of neural networks. Furthermore, we see a clear path to extending this analysis beyond multi-layer perceptrons. One could investigate the NTK in architectures with strong structural priors, such as Convolutional Neural Networks and Transformers, to understand the effect of operations like convolution and self-attention. Finally, though FFNNs and PINNs showed qualitatively similar trends, quantitative differences emerged, which was likely due to differences in both the loss formulation and the target function. Future work could explore unifying the modeling task across architectures to isolate these effects more precisely. This could come about through normalizing the losses, and NTK matrices in order to facilitate comparison.

6.3 Conclusion

In this report, we analyzed the evolution and variability of the NTK in both FFNNs and PINNs, with a focus on the influence of network width, depth, and activation function. Our empirical results aligned

with the results in Hanin and Nica (2019) and confirmed that the NTK, thus neural network dynamics, do indeed change and does not stay the same as the initialization as $\beta \to \infty$. This suggests that deeper and narrower networks exhibit more data dependency by adapting their internal representations to the training data, emphasizing a fundamental shift from the lazy kernel regression regime towards a rich feature learning paradigm. While deeper, narrower architectures are beneficial for promoting essential feature learning, there is a point where increasing the β is not desirable due to instability. The optimal architecture is likely one that balances this trade-off, operating at the "edge of chaos" to maximize learning capacity without sacrificing trainability.

We also found that while ReLU and GeLU networks exhibit strong finite-size effects and high NTK variability, nonsaturating functions such as sigmoid is much more stable.

Overall, our findings shed light on how architectural and training choices affect NTK dynamics. Hence NTK can be a powerful tool to researching neural networks behaviour and offer guidance for designing stable and efficient networks.

References

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. and Wang, R. (2019), 'On Exact Computation with an Infinitely Wide Neural Net'. arXiv:1904.11955 [cs].
URL: *http://arxiv.org/abs/1904.11955*

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S. and Zhang, Q. (2018), 'JAX: composable transformations of Python+NumPy programs'.
URL: *http://github.com/jax-ml/jax*

Hanin, B. and Nica, M. (2019), 'Finite Depth and Width Corrections to the Neural Tangent Kernel'. arXiv:1909.05989 [cs].
URL: *http://arxiv.org/abs/1909.05989*

Jacot, A., Gabriel, F. and Hongler, C. (2020), 'Neural Tangent Kernel: Convergence and Generalization in Neural Networks'. arXiv:1806.07572 [cs].
URL: *http://arxiv.org/abs/1806.07572*

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J. and Pennington, J. (2020), 'Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent', *Journal of Statistical Mechanics: Theory and Experiment* **2020**(12), 124002. arXiv:1902.06720 [stat].
URL: *http://arxiv.org/abs/1902.06720*

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J. and Schoenholz, S. S. (2020), Neural tangents: Fast and easy infinite neural networks in python, *in* 'International Conference on Learning Representations'.
URL: *https://github.com/google/neural-tangents*

Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O. (2021), 'Understanding deep learning (still) requires rethinking generalization', *Communications of the ACM* **64**(3), 107–115.
URL: *https://arxiv.org/abs/1611.03530*

A Notations

Table 1: Table of notations

Parameter	Symbol	Description
Input vector	x	$x \in \mathbb{R}^{n_0}$
Input Dimension	n_0	Number of input features
Number of Hidden Layers	d	Number of hidden layers (=depth)
Width of the j-th layer	n_j	For many experiments, we will assume a uniform hidden width, $n_j = n_{\text{hid}}$ for all j.
Hidden Layer Width	n_{hid}	Number of neurons per hidden layer (uniform across layers)
Output Dimension	n_{out}	Number of output units
Activation Function	$\sigma(\cdot)$	The activation function applied element-wise after each hidden layer. While we primarily focus on ReLU ($\sigma(z) = \max(0, z)$) for direct comparison with Hanin and Nica (2019), other activations like Gelu and Sigmoid will also be explored. The output layer is linear.
Weight Initialization Std	W_{std}	Standard deviation for weights: $(2/n_{i-1})^{1/2}$ for hidden layers, $(1/n_{i-1})^{1/2}$ for output layer
Bias Initialization Std	B_{std}	Biases initialized to zero but trainable
Finiteness Parameter	$\beta = \sum_{j=1}^{d} 1/n_j$	Inverse temperature, simplifies to $\beta = L_h/n_{\text{hid}}$ for uniform widths
Laplacian Operator	$\Delta u = \frac{d^2u}{dx^2}$	The Laplacian operator is the sum of second partial derivatives of u with respect to spatial variables. This is going to be used in the PINN neural network.

B NTK Background

Let $f(x;\theta)$ denote the output of a neural network for an input $x \in \mathbb{R}^{n_0}$ (where n_0 is the input dimension), parameterized by $\theta \in \mathbb{R}^P$ (where P is the total number of trainable parameters). The empirical NTK, for a given set of parameters θ and inputs x, x', is defined as the Gram matrix of the network's output gradients with respect to its parameters:

$$\Theta(x, x'; \theta) = J(x;\theta)J(x';\theta)^T = \sum_{p=1}^{P} \frac{\partial f(x;\theta)}{\partial \theta_p} \frac{\partial f(x';\theta)}{\partial \theta_p} \tag{5}$$

If the network has D_{out} outputs, then $f(x;\theta) \in \mathbb{R}^{D_{out}}$, $\frac{\partial f(x;\theta)}{\partial \theta_p} \in \mathbb{R}^{D_{out}}$, and $\Theta(x, x';\theta)$ becomes a $D_{out} \times D_{out}$ matrix. For a scalar output network ($D_{out} = 1$), $\Theta(x, x';\theta)$ is a scalar. Throughout this work, we often refer to the NTK as $K_{\mathcal{N}}(x, x')$ and its on-diagonal value for a single input x as $K_{\mathcal{N}}(x, x)$, its computation can be written out as:

$$K_{\mathcal{N}}(x, x) = \sum_{j=1}^{d} \sum_{\alpha=1}^{n_{j-1}} \sum_{\beta=1}^{n_j} \left(\frac{\partial \mathcal{N}}{\partial W_{\alpha,\beta}^{(j)}}(x) \right)^2 + \sum_{j=1}^{d} \sum_{\beta=1}^{n_j} \left(\frac{\partial \mathcal{N}}{\partial b_{\beta}^{(j)}}(x) \right)^2, \tag{6}$$

for a network with d layers, n_j neurons in layer j, $W_{\alpha,\beta}^{(j)}$ is the weight from neuron α in layer $j-1$ to neuron β in layer j, and $b_{\beta}^{(j)}$ is the respective bias. In the infinite-width limit, Jacot et al. (2020) showed that this kernel converges to a deterministic limiting kernel, $\Theta_\infty(x, x')$, which remains constant throughout training. This lazy training regime implies that the network effectively linearizes around its initialization, and its predictions are equivalent to those of a kernel machine using Θ_∞.

C Network architectures

Table 2: Network configurations for Group 1 with fixed n_{hid}, varying d, and $\beta = \frac{d}{n_{hid}}$.

Width	Depth	β
16	1	0.0625
16	2	0.1250
16	4	0.2500
16	8	0.5000
16	16	1.0000
16	32	2.0000
16	48	3.0000
16	64	4.0000
32	1	0.0313
32	2	0.0625
32	3	0.0938
32	4	0.1250
32	6	0.1875
32	8	0.2500
32	10	0.3125
32	16	0.5000
32	24	0.7500
32	32	1.0000
32	40	1.2500
32	48	1.5000
32	64	2.0000
64	1	0.0156
64	2	0.0313
64	3	0.0469
64	4	0.0625
64	6	0.0938
64	8	0.1250
64	12	0.1875
64	16	0.2500
64	24	0.3750
64	32	0.5000
64	48	0.7500
64	64	1.0000
64	80	1.2500
64	96	1.5000
128	1	0.0078
128	2	0.0156
128	4	0.0313
128	6	0.0469
128	8	0.0625
128	12	0.0938
128	16	0.1250
128	32	0.2500
128	64	0.5000
128	96	0.7500

Continued on next page

Table 2: Network configurations for Group 1 with fixed n_{hid}, varying d, and $\beta = \frac{d}{n_{hid}}$.

Width	Depth	β
128	128	1.0000
256	1	0.0039
256	2	0.0078
256	4	0.0156
256	8	0.0313
256	16	0.0625
256	32	0.1250
256	64	0.2500
256	96	0.3750
256	192	0.7500

Table 3: Network configurations for Group 2 with varying n_{hid}, fixed d, and $\beta = \frac{d}{n_{hid}}$.

Width	Depth	β
512	1	0.0020
1024	1	0.0010
2048	1	0.0005
4096	1	0.0002
512	2	0.0039
1024	2	0.0020
2048	2	0.0010
4096	2	0.0005
512	4	0.0078
1024	4	0.0039
2048	4	0.0020
4096	4	0.0010
512	8	0.0156
1024	8	0.0078
2048	8	0.0039
4096	8	0.0020
512	16	0.0313
1024	16	0.0156
2048	16	0.0078
4096	16	0.0039
512	32	0.0625
1024	32	0.0313
2048	32	0.0156

D Preliminary NTK Analysis

D.1 Experimental Setup

D.1.1 Feedforward Neural Network

In this preliminary study, we implemented a feedforward neural network. The network was trained to fit a noisy sine function defined as:

$$f(x) = \sin(3\pi x) + 0.8\sin(11\pi x), \quad x \in [-1, 1]$$

The loss function employed was the Mean Squared Error (MSE), defined as:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2$$

where $f_\theta(x_i)$ represents the neural network's prediction for input x_i, y_i is the corresponding ground truth, and n is the number of training samples. The training process utilized the Optax library Bradbury et al. (2018) for optimization, with network parameters initialized randomly using a JAX PRNG key.

The NTK was computed both at initialization (initial NTK) and after training (trained NTK) using `NTKAnalyzer`, which facilitates the calculation of the NTK matrix, its eigenvalues, condition number, and diagonal properties. Visualizations included heatmaps of the NTK matrix and plots of the eigenvalue spectra, generated using Matplotlib with a high-resolution configuration (`retina` format and Seaborn styling).

D.1.2 PINN Network Initialization

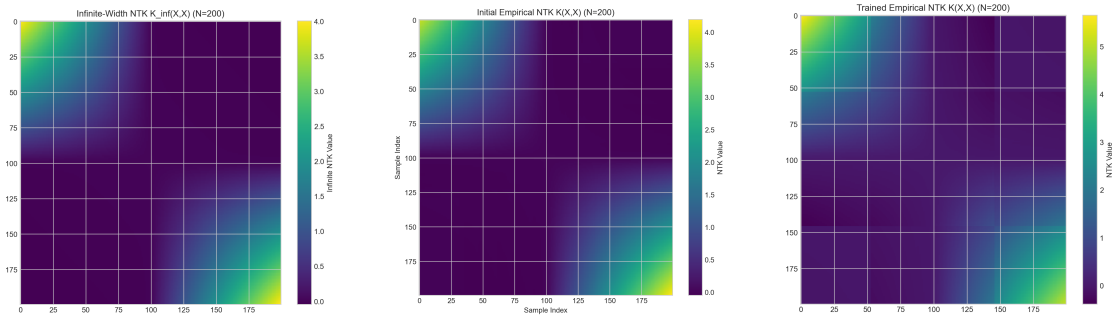
We proceed to train a PINN with a fully-connected MLP (variable depth/width, `erf` activation) to solve the 2D Poisson equation with known solution $u(x, y) = \sin(\pi x)\sin(\pi y)$, using a loss composed of the PDE residual and a boundary term weighted by $\lambda = 100$. The model is trained with the Adam optimizer (learning rate 10^{-3}) on 1000 interior and 2000 boundary points to assess whether the NTK findings generalize to this setting. It is interesting to note that the Relu activation function was used initially, as that is one of the more popular functions. However, the network did not train successfully. We believe this was because the network we are training is based on a second-order equation, and the second derivative of Relu is 0. Therefore, we opted for the Erf activ

D.2 Results

Analyzing the infinite-width NTK for a ReLU network with $N = 200$ samples reveals intriguing patterns in the kernel's structure. The heatmap of the infinite-width NTK (Figure 5a) exhibits bright regions at the corners, corresponding to sample indices at the beginning and end of the samples. This pattern suggests that the sorted input data creates clusters of similar gradient structures at the extremes, this can be explained by the ReLU nonlinearity, $\sigma(z) = \max(0, z)$, inputs from an "extreme" cluster (e.g., those with very small or very large feature values) may consistently result in many pre-activations $f(x_i)$ being either predominantly negative (leading to $\sigma(f(x_i)) \approx 0$ for many neurons) or predominantly positive ($\sigma(f(x_i)) \approx 1$). This consistency in the behavior of σ within each extreme cluster generates highly similar gradient structures $\nabla_\theta f_\theta(x_i)$ for samples within that cluster. Since the NTK, $\Theta_\infty(x_i, x_j)$, measures the similarity of these gradient structures and its recursive definition involves terms like $\mathbb{E}[\sigma(f(x_i))\sigma(f(x_j))]$, the pronounced intra-cluster similarity at the data extremes manifests as higher NTK values, thereby creating the observed bright regions in the heatmap's corners.

Contrastingly, the diagonal of the heatmap does not exhibit a uniformly bright line, which is unexpected given that $K(x_i, x_i)$ should represent the maximum self-similarity. This subdued diagonal likely results from the oscillatory nature of the target function and ReLU's non-linear gradient effects.

The NTK remains largely same for $\beta = 0.001953$ before and after training (Figure 5b, 5c). This behaviour is characteristic of networks where the β is very small, and it operates in the so called *lazy training regime*. In this regime, the NTK remains approximately constant throughout the training process, which aligns with the theoretical predictions for infinite-width networks where the limiting NTK is deterministic and fixed during training by Jacot et al. (2020). Consequently, the lack of significant



(a) Infinite-Width NTK (b) Initial Empirical NTK (c) Trained Empirical NTK

Figure 5: Heatmaps of NTK for $N = 200$ with ReLU, d=1, n=512, $\beta = 0.001953125$

change in the NTK implies that *minimal* feature learning has occurred, the network has primarily learned by fitting the data using the set of features implicitly defined by its kernel at initialization. Indeed, in Appendix F, the narrower network (higher β) adapted to data much better.

For large finiteness β, the heatmaps in Appendix E shows that NTK becomes more chaotic and is not same as the infinite-width NTK.

D.2.1 Preliminary NTK Analysis for PINNs

We begin our analysis by comparing the NTK matrices of the PINN at three stages: initialisation, infinite-width limit, and after training. This comparison helps reveal how training modifies the network and how closely the empirical behaviour adheres to the theoretical infinite-width kernel described in Jacot et al. (2020). Figure 6 visualizes these three NTK matrices as heatmaps, computed for a network with $N = 200$ samples, hidden layer width $d = 512$, input dimension $n = 1$, and depth-to-width ratio $\beta = 0.001953125$. Interestingly, all three NTK matrices are visually almost indistinguishable. This high degree of similarity indicates that the NTK remains nearly constant throughout training and is already close to its infinite-width theoretical limit at initialisation.

Such behavior is characteristic of the lazy training regime, where the network parameters do not change significantly during optimization and the function learned is essentially a linear combination of the features defined at initialization. In this regime, the network acts like kernel regression, with the NTK fixed and determining the network. The closeness of the trained NTK to both the initial and infinite-width NTKs implies that feature learning is minimal and the network generalizes primarily through the static feature representation induced by the NTK.

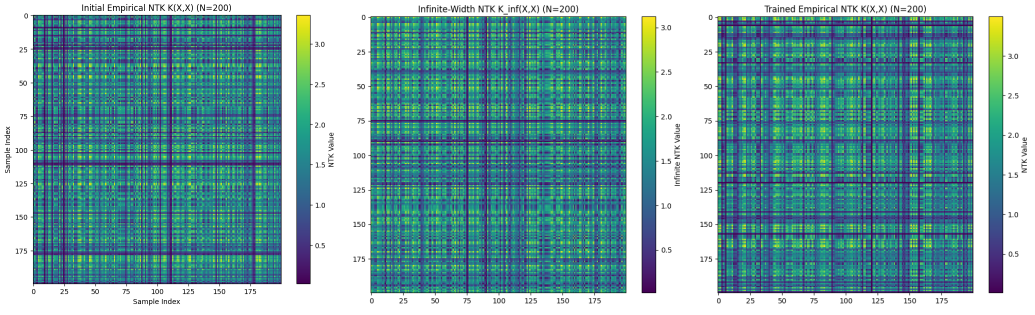


Figure 6: Heatmap with $N = 200$ samples, and $d = 512$, $n = 1$, with a $\beta = 0.00390625$. Left: Empirical NTK $K(X, X)$ at initialization. Middle: Infinite-width NTK $K_\infty(X, X)$ computed analytically. Right: Empirical NTK after training.

E Testing of Poisson PINN with Error

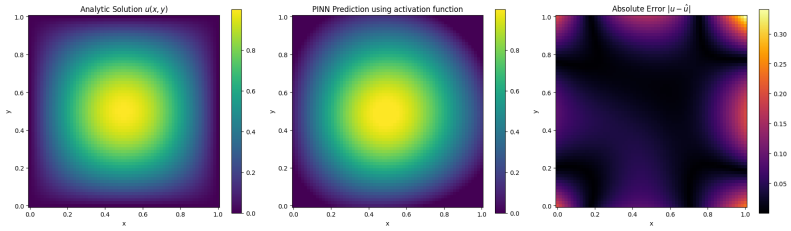


Figure 7: Left: Analytic solution of Poisson, Middle: PINN prediction on test set. Right: Heatmap of training loss, showing effective learning of the PDE solution.

F NTK at initialization as β scales for ReLU.

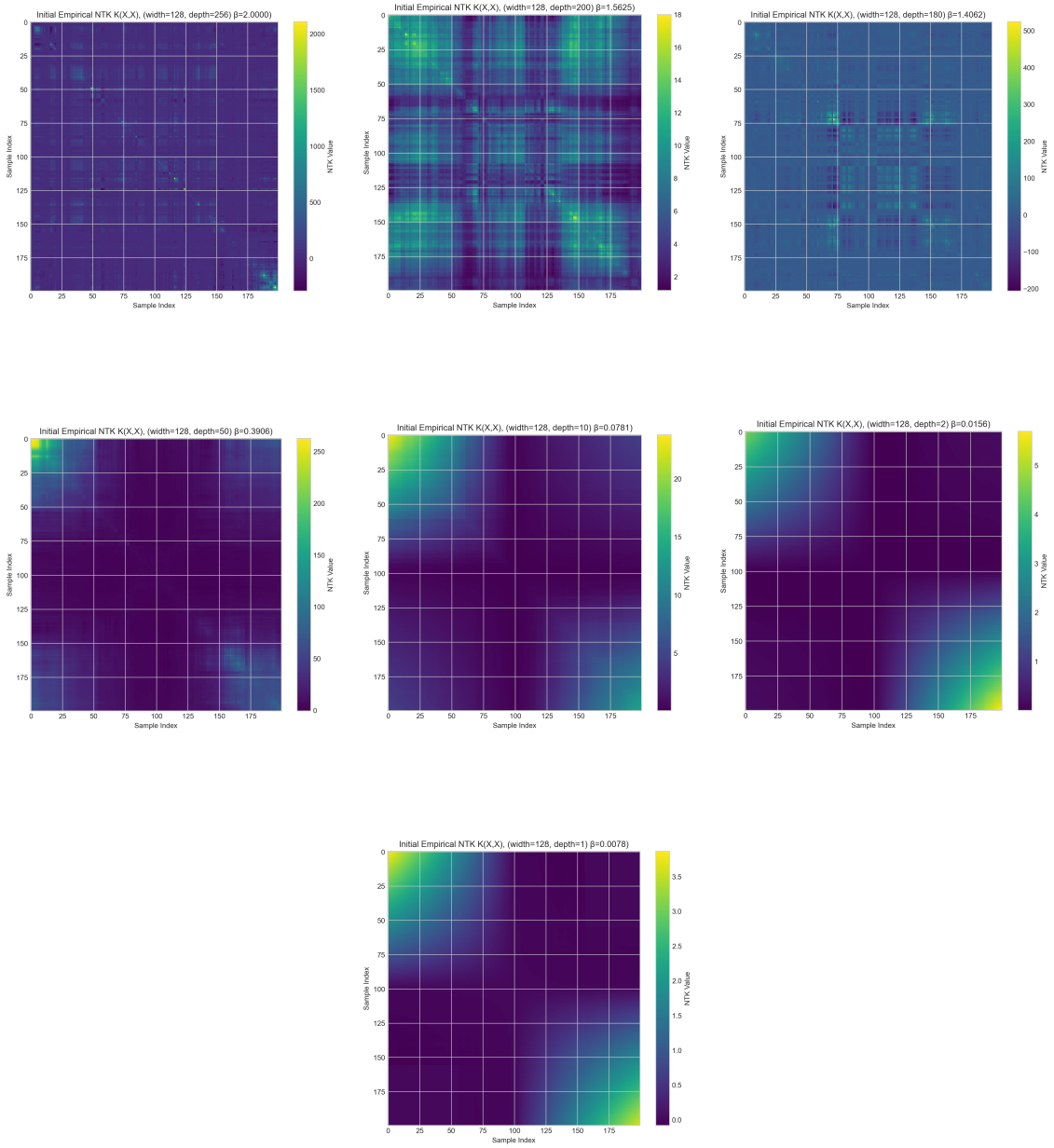
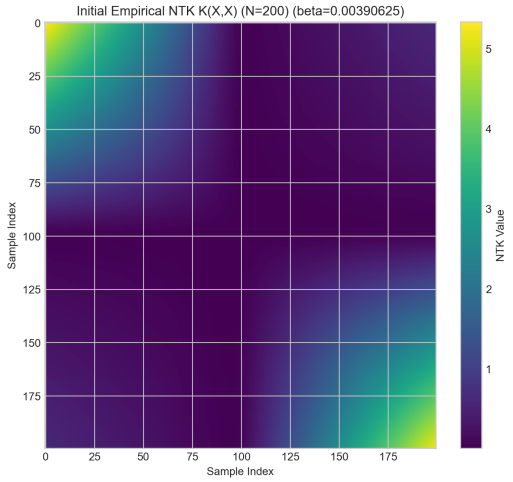
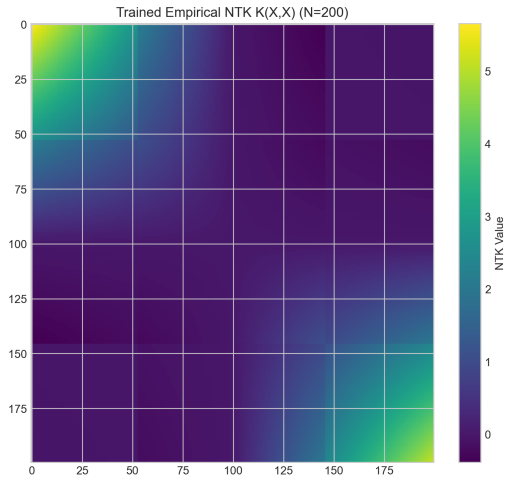


Figure 8: NTK heatmaps as $\beta \to 0$.

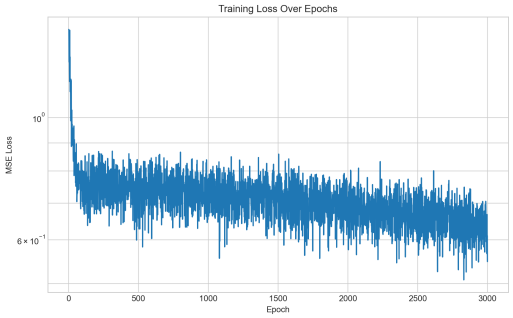
G Comparison between trained NTK.



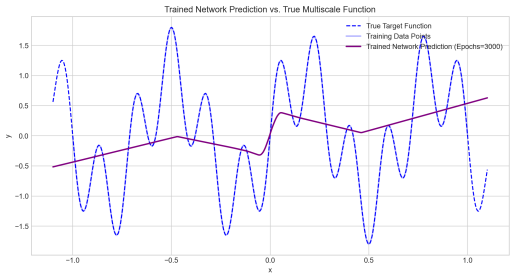
(a) Initial NTK



(b) NTK after training

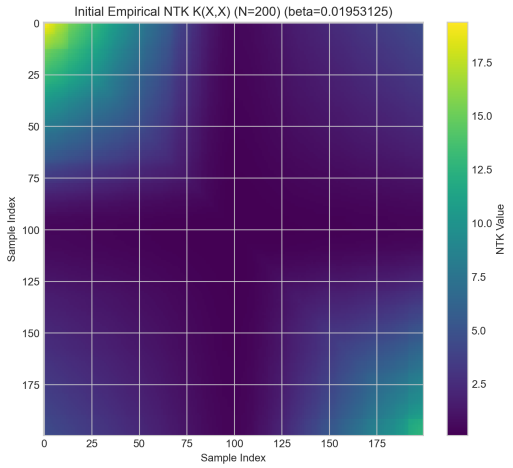


(c) Loss after 3000 epochs

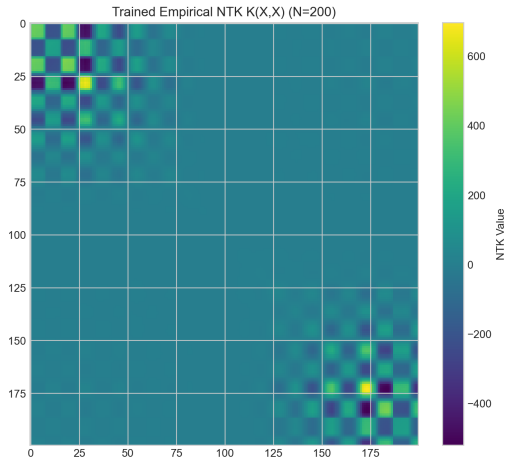


(d) Fitted values.

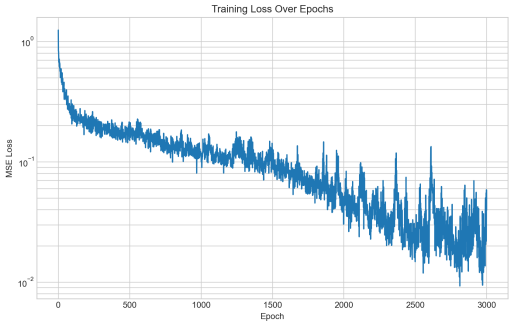
Figure 9: FNN with width $= 512$ and depth $= 1 \implies \beta \approx 0.0019$. Network was trained over 3000 epochs using Adam optimizer with learning rate 0.001. 9a closely resembles the infinite width NTK in 5a and does not change after training as seen in 9b. In 9d, the prediction on the trained interval $[-1, 1]$ is not fitted well, our network has not learned.



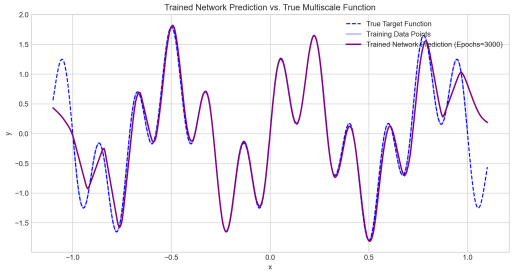
(a) Initial NTK



(b) NTK after training



(c) Loss after 3000 epochs



(d) Fitted values.

Figure 10: FNN with width $= 512$ and depth $= 10 \implies \beta \approx 0.0195$. Network was trained over 3000 epochs using Adam optimizer with learning rate 0.001. 10a closely (but not as close as 9a, see the corners are not as smooth) resembles the infinite width NTK in 5a but this changes after network is trained as seen in 10b. In 10d, the prediction on the trained interval $[-1, 1]$ is fitted well.

H FINN vs PINN

While both FFNNs and PINNs share similar architectural components, their training objectives and the functions they model differ in important ways. In our experiments, the FFNNs were trained using standard supervised learning with pointwise loss, while the PINNs were trained using a physics-informed loss that incorporates differential operators and boundary conditions. Moreover, the target functions used for each architecture were not identical, which complicates direct one-to-one comparisons. Despite these differences, we observed qualitatively similar trends in NTK behavior across both network types, especially in how different activation functions influence NTK stability as β increases. However, subtle quantitative discrepancies emerged, particularly in the magnitude and sensitivity of NTK variation for different activation functions, which may be attributed to the difference in the modeled function and loss structure. A possible approach to migitate this problem is to employ standardizing techniques on the loss functions, for instance, by Fourier or Laplace transforms.