Insights From Infinitely Wide Neural Networks

by

Mohamad Amin Mohamadi

B.Sc., Amirkabir University of Technology, 2020

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

April 2023

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Insights From Infinitely Wide Neural Networks

submitted by **Mohamad Amin Mohamadi** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Danica J. Sutherland, Assistant Professor, Computer Science, UBC
Supervisor

Frank Wood, Associate Professor, Computer Science, UBC
Supervisory Committee Member

Helge Rhodin, Assistant Professor, Computer Science, UBC
Supervisory Committee Member

Abstract

Studying neural networks in the limit of infinite-width has provided us with numerous valuable theoretical and practical insights about the initialization of NNs, their training dynamics and properties of the learnt functions. One of the theoretical tools emerged from this study is the empirical Neural Tangent Kernel (eNTK). The eNTK can provide a good understanding of a given network's representation: they are often far less expensive to compute and applicable more broadly than infinite-width NTKs. In this work, we use eNTKs to predict the local dynamics of neural networks in an active learning setup to propose a new method for approximating active learning acquisition strategies that are based on retraining with hypothetically-labeled candidate data points. Furthermore, to tackle the notorious space and computational complexity of calculating eNTKs, we propose a fast approximation of eNTK using a block-diagonal kernel resulting from eNTK with respect to only one (or average) of the output neurons of a network. We further use this approximation in our proposed "look-ahead" strategies in deep active learning. We finally present empirical evidence that our querying strategy beats other look-ahead strategies by large margins, and achieves equal or better performance compared to state-of-the-art methods on several benchmark datasets in pool-based active learning.

Lay Summary

Deep learning has drastically advanced over the past decade along with a dramatic increase in the volume of available data. It is, however, time-consuming and expensive to manually annotate large datasets. In this thesis, we alleviate this by allowing a model to “actively” request annotation of specific data points in a specific fashion, which we call ”look-ahead” active learning. To do so, we leverage the recent theoretical advances in understanding the training dynamics of infinitely wide neural networks, while proposing approximations for the tools derived from these analyses to improve computational requirements of our algorithms.

Preface

This thesis is based on my investigations of the empirical Neural Tangent Kernel under supervision of Dr. Danica J. Sutherland. The author, Mohamad Amin Mohamadi, in collaboration with his supervisor Dr. Sutherland developed both the mathematical framework/proofs presented and implemented and deployed the empirical investigations. Wonhoe Bae, Mohamad Amin Mohamadi's colleague has also been partially involved in the development of this thesis.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Danica, for her continuous support, guidance, and mentorship throughout my studies. In particular, Danica's paticence has had a profound impact on my development as a researcher, and her vision has helped me a lot in developing this thesis. I would also like to thank my other reviewers, Frank Wood and Helge Rhodin, for their time and valuable comments. I wish to acknowledge that collaborating with Wonhoe Bae, my dear colleague, was a great opportunity for me. I learnt a lot through working with him as a team and I always admired his dedication and hard work. I would also like to thank my parents, and my siblings, who have supported and encouraged me all my life.

Chapter 1

Introduction

The pursuit of a theoretical foundation for deep learning has lead researchers to uncover interesting connections between neural networks (NNs) and kernel methods. It has long been known that randomly initialized NNs in the infinite width limit are Gaussian processes with the Neural Network Gaussian Process (NNGP) kernel, and training the last layer with gradient flow under squared loss corresponds to the posterior mean [21, 35, 43, 46, 51, 73, 76]. More recently, Jacot et al. [29] built off a line of closely related prior work to show that the same is true with a different kernel, the Neural Tangent Kernel (NTK), if we train all the parameters of the network. In a gist, NTK is a dot-product kernel using which we can describe the evolution of NNs during their training by gradient descent. Yang [77], Yang and Littwin [78] showed this holds not just for fully-connected NNs but universally across architectures, including ResNets and Transformers. Lee et al. [36] also showed that the dynamics of training wide but finite-width NNs with gradient descent can be approximated by a linear model obtained from the first-order Taylor expansion of that network around its initialization. Furthermore, they experimentally showed that this approximation excellently holds even for networks that are not so wide.

In addition to theoretical insights, NTKs have had significant impact in diverse practical settings. Arora et al. [3] show very strong performance of NTK-based models on a variety of low-data classification and regression tasks. The condition number of an NN's NTK has been shown correlation directly with the trainability and generalization capabilities of the NN [74, 75]; thus, Chen et al. [11], Park

et al. [56] have used this to develop practical algorithms for neural architecture search. Bachmann et al. [7], Wei et al. [72] estimate the generalization ability of a specific network, randomly initialized or pre-trained on a different dataset, with efficient cross-validation. Developing on this, [11, 56] have proposed algorithms for neural architecture search and Wei et al. [72] has proposed simple tricks for predicting the generalization of a NN. Zhou et al. [82] use NTK regression for efficient meta-learning, and [26, 44, 71] use NTKs for active learning. Zhou et al. [82] has replaced the inner-loop update of MAML with kernel ridge regression using the NTK to perform efficient meta-learning. NTK has also been employed to develop novel active learning acquisitions and benchmarks [26, 71].

There has also been significant theoretical insight gained from empirical studies of networks' NTKs. For instance, Several recent works have also used the NTK to discover, characterize and analyze different aspects of NNs. Fort et al. [15] use NTKs to study how the loss geometry the NN evolves under gradient descent. Franceschi et al. [16] employ NTKs to analyze the behaviour of Generative Adverserial Networks (GANs). Nguyen et al. [48, 49] use NTKs for dataset distillation. Adlam et al. [1], He et al. [22] use NTKs to predict and analyze the uncertainty of a NN's predictions. Tancik et al. [67] use NTKs to analyze the behaviour of MLPs in learning high frequency functions, leading to new insights into our understanding of neural radiance fields. We thus believe NTKs will continue to be used in both theoretical and empirical deep learning.

Inspired by these works, in this work, we utilize NTKs to propose an algorithm that makes it feasible to conduct active learning with look-ahead acquisition strategies using general neural networks. Moreover, we present a simple trick for a strong approximation of the empirical NTK (eNTK) that is orders of magnitude faster in compute in comparison to the eNTK.

1.1 Contribution

Our contributions can be categorized into two main branches:

- First, we propose an algorithm that makes it feasible to conduct active learning with look-ahead acquisition strategies using general neural networks. More specifically, we approximate a network using its Neural Tangent Kernel

(NTK) [29], which makes it possible to obtain the behavior of retraining a network on a candidate data point without actually retraining the network. At each query step, we use the local approximation of the neural network as a proxy to compute a look-ahead acquisition function, from which we query a new data point. We prove that in the asymptotic regime where the width of the neural network goes to infinity, this approximation agrees with the result of actually retraining the network, even in an iterative setup such as in active learning. Our approximation decreases the wall-clock time more than 100 times compared to naïve computation of a look-ahead acquisition function, making them far more feasible in practice.

- Second, we present a simple trick for a strong approximation of the eNTK that removes the O^2 from the size of the kernel matrix, resulting in a factor of O^2 improvement in the memory and up to O^3 in computation, where O is the number of neurons in the output layer of the corresponding NN (*e.g.* number of classes in the classification task). Since for typical classification datasets O is at least 10 (*e.g.* CIFAR-10) and potentially $1{,}000$ or more [*e.g.* ImageNet, 13], this provides multiple orders of magnitude savings over the full eNTK. Without this, most empirical applications of eNTK become infeasible to compute. For instance, the full eNTK of a classification model even on the CIFAR-10 dataset, stored in double precision, takes over 1.8 terabytes in memory. We prove this approximation converges to the original eNTK at a rate of $\tilde{\mathcal{O}}(n^{-1/2})$ for a standard-initialization NN of depth L and width n in each layer, and the predictions of kernel regression with the approximate kernel do the same. We also conduct diverse experimental investigations to support our theoretical results, across a range of architectures and settings. We hope this approximation further enables researches to employ NTKs towards theoretical and empirical advances in wide networks.

1.2 Overview

The rest of this thesis is organized in three chapters. In the next two chapters, we present our proposed algorithms in details, along with the preliminary background, literature review and experimental evaluations. Finally, in Chapter 4, we conclude

the thesis and discuss the limitations and future work.

Chapter 2

Fast Look-Ahead Deep Active Learning

Deep learning has drastically advanced over the past decade along with a dramatic increase in the volume of available data. It is, however, time-consuming and expensive to manually annotate large datasets. Active learning attempts to alleviate this by allowing a model to "actively" request annotation of specific data points, with the expectation that a model trained on informative points will learn a better prediction than one on a random, "passively" labeled set of the same size.

The most common form of active learning is based on using an *acquisition function* to measure the informativeness of potential data points or batches from some unlabeled pool. Many successful acquisition functions are based on the current model's uncertainty, such as maximum entropy [70] and Bayesian Active Learning by Disagreement (BALD) [27]. These functions are based on the expectation that training on points about which the current model is uncertain will be effective at decreasing the future model's uncertainty about similar points. This assumption, however, often does not hold: uncertain points might be inherently difficult to predict (aleatoric uncertainty) or simply too difficult for the model to learn right now.

It would be helpful, then, to see how a new data point would change a model, potentially with respect to other unseen data. One such strategy is known as Expected Model Change, which approximates how much a model's parameters

will change when observing a new hypothetically-labeled data point [6, 65]. This approach, though, only considers the magnitude of change in parameters, not the model's actual outputs on the data distribution, and it only looks at the size of the first stochastic gradient (SGD) step with the new data point rather than considering the full trajectory of training.

We will refer to acquisition strategies which consider full retraining of a model based on hypothetical observations as *look-ahead* criteria. Previous attempts to use look-ahead criteria based on the change in model output include Expected Error Reduction [59] and Expected Model Output Change [17]. These approaches have been used only with specialized models such as Naïve Bayes and Gaussian processes, where retraining on every new candidate data point is feasible. These classes of models, however, tend to not work as well as modern neural networks, meaning these approaches are outperformed by simpler acquisition functions on stronger models.

Our proposed method based on approximating retraining of NNs using their first-order Taylor expansion outperforms existing look-ahead strategies by large margins, and achieves equal or better performance compared to the state-of-the-art methods on several benchmark datasets in pool-based active learning, including MNIST [8], SVHN [9], CIFAR10 [10], and CIFAR100 [10]. The NTK approximation also makes it possible to decouple receiving new data labels and SGD training, unlike previous active learning methods, where knowing true labels does not add any information without SGD training. This is useful when annotation is fast (yet expensive) but model training is slow, since adding data to an existing NTK approximation is far faster than retraining with SGD. Sequentially adding true labels of new data gives performance substantially better than more common batch setups.

2.1 Preliminaries

Active learning. In pool-based active learning, we have a labeled set $\mathcal{L} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{|\mathcal{L}|}$ and unlabeled set $\mathcal{U} = \{x^{(i)}\}_{i=1}^{|\mathcal{U}|}$, where $x^{(i)}$s are inputs and $y^{(i)}$s are corresponding labels. At each time step t, a model f parameterized by θ is trained on the labeled set $\mathcal{L}$ which we denoted as $f_{\mathcal{L}}$, and then a data point from the

unlabeled set $\mathcal{U}$ is selected to be labeled, according to an acquisition function A measuring the "information gain" of a potential data point x

$$x^* = \arg\max_{x \in \mathcal{U}} A(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) \tag{2.1}$$

where x^* is the selected datapoint.

One of the most common choices of the acquisition function is *maximum entropy*, through which one computes the estimated entropy of the unknown label of x, denoted as Y: $A_{\textit{entropy}}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := \mathcal{H}(Y \mid x;\, f_{\mathcal{L}})$. Although the maximum entropy acquisition function often works well in practice, it does not consider how the model would change with a new queried data point, and so can overly prioritize inherently difficult, uninformative points (*e.g.* outliers).

Expected gradient length [64] approximates the *expected model change* by how large the gradient of the loss becomes when adding that data point, $\mathbb{E}_{y \sim p_\theta(\cdot|x)} \|\nabla \ell_\theta(\mathcal{L} \cup (x, y))\|$. BADGE [6], a more recent variant, lower-bounds the gradient norm of the last layer induced by any possible label. These approaches do not consider how the change in a model actually interacts with the data distribution: large parameter changes might give relatively small changes in predictions on most data points. They also consider a single SGD update, only roughly approximating total change over training.

One approach to alleviate this limitation is *expected error reduction* [EER; 59], given by

$$A_{\textit{EER}}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := - \mathop{\mathbb{E}}_{y \sim p_\theta(\cdot|x)} \sum_{i=1}^{|\mathcal{U}|} \mathcal{H}(Y^{(i)} \mid x^{(i)};\, f_{\mathcal{L}^+}). \tag{2.2}$$

Here $f_{\mathcal{L}^+}$ refers to the model trained on $\mathcal{L} \cup \{(x, y)\}$, and $\mathcal{H}(Y^{(i)}|x^{(i)};\, f_{\mathcal{L}^+})$ is the (estimated) entropy of the unknown label of $x^{(i)}$ in $\mathcal{U}$ using that model. Maximizing $A_{\textit{EER}}$ chooses the candidate point whose label has maximal mutual information to the labels of unlabeled data, $\mathcal{I}(Y; y|\, f_{\mathcal{L}})$. If $f_{\mathcal{L}}$ is a fairly good model and the problem is fairly difficult, the numerical value of $A_{\textit{EER}}$ is likely dominated by the sum of irreducible (aleatoric) uncertanties over the data points; it thus might be prone to noise in estimation based on those large terms.

Freytag et al. [17] propose instead *expected model output change* (EMOC), based on the difference in model predictions as measured by a distance $\mathcal{D}$ (*e.g.* the Euclidean distance between probability vectors). The acquisition function is defined as

$$A_{EMOC}(x, f_{\mathcal{L}}, \mathcal{L}, \mathcal{U}) := \mathop{\mathbb{E}}_{y \sim p_\theta(\cdot|x)} \sum_{i=1}^{|\mathcal{U}|} \mathcal{D}(f_{\mathcal{L}}(x^{(i)}), f_{\mathcal{L}^+}(x^{(i)})). \tag{2.3}$$

Intuitively, we might hope that the change in outputs roughly cancels out the irreducible (aleatoric) uncertainty, leaving us to focus primarily on the decrease in model (epistemic) uncertainty [50]. Our techniques also apply to EER, but EMOC-like criteria performed better in our initial exploration.

Despite their advantages, look-ahead acquisition functions have been in practical reach only with certain types of models: EER has been applied to Naïve Bayes [59] and Gaussian random fields [83], and EMOC only to Gaussian processess [30, 31]. Neural networks generally outperform those models, but obtaining $f_{\mathcal{L}^+}$ with SGD training for every candidate example in $\mathcal{U}$ is impractically expensive. In this work, we propose to approximate $f_{\mathcal{L}^+}$ with a more computationally efficient proxy based on neural tangent kernels, making look-ahead acquisition functions feasible on neural networks.

Neural tangent kernels. Over the past few years, connections between infinitely wide neural networks trained by gradient descent and kernel methods have become increasingly clear. The NNGP approximation of Matthews et al. [42], building off a line of work stemming from Neal [45], shows that if a network's parameters are initialized with an appropriate Gaussian distribution and only the last layer is trained, the resulting function agrees with a Gaussian process with the kernel $\mathcal{K}(x, x') = \mathbb{E}_\theta[f_\theta(x) f_\theta(x')]$, where the expectation is over initializations. Jacot et al. [29], building off several immediately preceding works on optimization of wide neural networks, show that infinitely wide fully-connected neural networks also follow Gaussian process behavior, with the NTK kernel

$$\mathcal{K}(x, x') = \mathop{\mathbb{E}}_{\theta} \left[\left\langle \frac{\partial f_\theta(x)}{\partial \theta}, \frac{\partial f_\theta(x')}{\partial \theta} \right\rangle \right] \tag{2.4}$$

where the derivatives denote Jacobians with respect to the vector of all parameters θ. They also show that the NTK remains constant over the course of training, so that these networks evolve like a linear model under kernel gradient descent. Among many important follow-ups, Arora et al. [2] expand the results to convolutional networks with finite but large widths, and Novak et al. [53] provide an implementation for nearly-arbitrary network structures. Yang [77], Yang and Littwin [78] later showed that Jacot et al. [29]'s findings are architecturally universal, extending the domain from fully-connected networks to a large class of architectures including ResNets and Transformers.

Unfortunately, these infinite-width versions of networks tend not to generalize as well as finite networks trained by SGD. Hence, we still want to conduct active learning for finite neural networks, rather than for pure NTK models. The infinite-width NTK also tends to be a mediocre proxy for the behavior of training a finite neural network, and so using it as a proxy for $f_{\mathcal{L}^+}$ does not tend to work as well as we might hope (*e.g.* Figure 2.1b). We will thus instead use a local linearized approximation of the neural network, based on the empirical neural tangent kernel, as introduced next.

2.2 Active Learning using NTKs

In this section, we first define a neural network, and the linear model which in the infinite-width limit agrees with training that network. We will prove that in this setting that sequentially retraining on increasing datasets – as in the active learning process – is the same as if we had trained from scratch on the final dataset. In practical regimes, however, the infinite-width NTK does not tend to agree with finite-width SGD very closely. We thus use a local linear approximation of the network, based on the empirical NTK, to approximate retraining in our look-ahead criteria.

Notation. We mostly use the same notation as Jacot et al. [29]. We define f as a fully-connected neural network with $L+1$ hidden layers numbered from 0 (input) to L (output), where each layer has $n_0, \ldots, n_L$ neurons. This network has number of parameters $P = \sum_{l=0}^{L-1} (n_l + 1)n_{l+1}$, each layer has a weight matrix

$W^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and a bias vector $b^{(l)} \in \mathbb{R}^{n_{l+1}}$. The network is defined as f_θ, where $\theta = \cup_{l=0}^{L} \theta^l$ collects all of the parameters and $\theta^l = \mathrm{vec}(W^{(l)}, b^{(l)})$. We denote a labeled set $\mathcal{L}$ as defined in Section 2.1 and use $\mathcal{X} = \{x \,|\, (x, y) \in \mathcal{L}\}$ and $\mathcal{Y} = \{y \,|\, (x, y) \in \mathcal{L}\}$ to denote its inputs and labels, respectively. We write $f_{\mathcal{D}_1} \xrightarrow[t=\infty]{\mathcal{D}_1 \cup \mathcal{D}_2} f_{\mathcal{D}_1 \cup \mathcal{D}_2}$ to mean that $f_{\mathcal{D}_1 \cup \mathcal{D}_2}$ is trained using gradient descent until convergence on the dataset $\mathcal{D}_2$, starting from $f_{\mathcal{D}_1}$.

Unless otherwise specified, the loss function associated with gradient descent training is assumed to be the squared loss, $\ell(\mathcal{Y}, f(\mathcal{X})) := \frac{1}{2}\|\mathcal{Y} - f(\mathcal{X}))\|_2^2$. This allows for far more efficient usage of NTKs, but Hui and Belkin [28] demonstrate that, with the right learning rate, L_2 loss is just as effective as cross-entropy for many vision and natural language processing tasks.

Neural networks in the infinite width limit. Jacot et al. [29] show that in the infinite width limit, when the network f is trained using gradient descent with the squared loss on a labeled set $\mathcal{L}$, the outputs of the network on any arbitrary point x evolve as

$$f_t(x) = f_0(x) + \mathcal{K}(x, \mathcal{X})\, \mathcal{K}(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{I} - e^{-t\mathcal{K}(\mathcal{X}, \mathcal{X})})(\mathcal{Y} - f_0(\mathcal{X})), \qquad (2.5)$$

where $\mathcal{K}$ is the NTK of (2.4), which stays constant during training. Here we treat inputs $\mathcal{X}$ as a matrix and labels $\mathcal{Y}$ as a vector in corresponding order.

Consider sequentially training a network on increasing datasets $S_1 \subset S_2 \subset \cdots \subset S_C$, warm-starting each time from the result of training on the previous dataset. We build on the results of Jacot et al. [29] to show that the final network from this process is asymptotically equivalent, in the infinite-width limit, to training the same network from scratch (cold-start) on the last, biggest dataset S_C. This justifies our efficient approximation of the retrained network, even after we have gone through many iterations of active learning. See Appendix A.1 for a formal statement and proof.

Theorem 2.2.1 (Informal)**.** *Let $S_1, S_2, \ldots, S_C$ be C datasets such that for all $i > j$, $S_j \subset S_i$. Let f_0 be a randomly initialized network. Let $f_{S_{1,2,\ldots,C}}$ be the network resulting from starting at f_0 and training f using gradient descent on datasets*

$S_1, \ldots, S_C$ sequentially until convergence:

$$f_0 \xrightarrow[t=\infty]{S_1} f_{S_1} \xrightarrow[t=\infty]{S_2} f_{S_{1,2}} \cdots \xrightarrow[t=\infty]{S_C} f_{S_{1,2,\ldots,C}}.$$

Also, let f_{S_C} be $f_0 \xrightarrow[t=\infty]{S_C} f_{S_C}$. Assuming that f has $L+1$ layers such that taking $n_0, n_1, \ldots, n_L \to \infty$ sequentially, we have for any arbitrary data point x that $f_{S_C}(x) = f_{S_{1,2,\ldots,C}}(x)$.

NTK approximations of retrained networks. Armed with Theorem 2.2.1, we can now approximate the outputs of a neural network from retraining. Suppose we have a labeled set $\mathcal{L}$ and unlabeled set $\mathcal{U}$ as defined in Section 2.1. A neural network $f_{\mathcal{L}}$ (whose layers are assumed to be infinitely wide) has been trained on $\mathcal{L}$ as $f_0 \xrightarrow[t=\infty]{\mathcal{L}} f_{\mathcal{L}}$. We are interested in characterizing the outcome after retraining this neural network using each of the data points in $\mathcal{U}$. In other words, for each $x' \in \mathcal{U}$ with a hypothetical label y' and $\mathcal{L}^+ := \mathcal{L} \cup (x', y')$, we would like to know what $f_{\mathcal{L}} \xrightarrow[t=\infty]{\mathcal{L}^+} f_{\mathcal{L}^+}$ looks like.

According to (2.5), the outputs of $f_{\mathcal{L}}$ for an arbitrary data point x can be formulated as

$$f_{\mathcal{L}}(x) = f_0(x) + \mathcal{K}(x, \mathcal{X})\, \mathcal{K}(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{Y} - f_0(\mathcal{X})). \tag{2.6}$$

Let $\mathcal{X}^+$ and $\mathcal{Y}^+$ be inputs and labels of $\mathcal{L}^+$. Then, based on Theorem 2.2.1, we can conclude that

$$f_{\mathcal{L}^+}(x) = f_0(x) + \mathcal{K}(x, \mathcal{X}^+)\mathcal{K}(\mathcal{X}^+, \mathcal{X}^+)^{-1}(\mathcal{Y}^+ - f_0(\mathcal{X}^+)). \tag{2.7}$$

As $\mathcal{K}$ remains constant in the infinite-width regime, most of the relevant quantities can be reused

$$\mathcal{K}(x, \mathcal{X}^+) = \begin{bmatrix} \mathcal{K}(x, \mathcal{X}) & \mathcal{K}(x, x') \end{bmatrix}, \qquad \mathcal{K}(\mathcal{X}^+, \mathcal{X}^+) = \begin{bmatrix} \mathcal{K}(\mathcal{X}, \mathcal{X}) & \mathcal{K}(\mathcal{X}, x') \\ \mathcal{K}(x', \mathcal{X}) & \mathcal{K}(x', x') \end{bmatrix}. \tag{2.8}$$

Remark 2.2.2. In the infinite width limit, we can analytically obtain the predictions

of a neural network after retraining on additional data points by augmenting its corresponding NTK using (2.8).

Several works have shown that while being an inspiring theoretical motivation, neural networks in the infinite width limit do not work as well as their finite-width counterparts [2, 40, 41]. Although Arora et al. [2] prove non-asymptotic bounds between these infinite width neural networks and their corresponding finite-width networks, empirical studies in [36, 77] have shown that this approximation may not be effective for practical network widths. This is further explained in many theoretical studies which show that infinite-width networks in the NTK-regime are incapable of learning representations, a crucial element in training deep NNs. Thus, we would like to be able to efficiently characterize the outputs of a finite neural network after retraining.

Lee et al. [36] showed the linearization of a neural network around its initialization has training dynamics converging to that of the neural network as the width grows. That is, let f_t denote the network which has been trained with gradient descent for t steps, and f_t^{lin} the result of training the linearized network for t steps. They prove that, under some regularity conditions and when the learning rate η is less than a certain threshold,

$$\sup_{t\geq 0} \|f_t(x) - f_t^{lin}(x)\|_2 = \sup_{t\geq 0} \|\Theta_t - \Theta_0\|_F = \mathcal{O}\left(\frac{1}{\sqrt{\text{width}}}\right). \tag{2.9}$$

Informally, when f is wide enough, its predictions can be approximated by those of the linearized network f^{lin}. This is attractive since f^{lin} has simple training dynamics, converging to

$$f_{\mathcal{L}}^{lin}(x) = f_0(x) + \Theta_0(x, \mathcal{X})\,\Theta_0(\mathcal{X}, \mathcal{X})^{-1}(\mathcal{Y} - f_0(\mathcal{X})), \tag{2.10}$$

where Θ_0 is the *empirical* tangent kernel of f_0, $\Theta_0(x, y) := \nabla_\theta f_0(x)\,\nabla_\theta f_0(y)^\top$.

In the infinite-width limit, the empirical NTK Θ_t converges to the NTK $\mathcal{K}$ almost surely, leading (3.4) to become equivalent to (2.6). In finite-width regimes, however, it is intuitive to expect that the *local* approximation may still be able to handle *local* retraining, such as $f_{\mathcal{L}} \xrightarrow[t=\infty]{\mathcal{L}\cup\{(x,y)\}} f_{\mathcal{L}\cup\{(x,y)\}}$, even when the correspondence to the infinite-width limit $\mathcal{K}$ is loose. We thus model retraining of a finite

network by augmenting the empirical NTK as in (2.8), as justified by Theorem 2.2.1 and Remark 2.2.2. Specifically, we approximate the finite width network $f_{\mathcal{L}^+}(x)$ defined by $f_{\mathcal{L}} \xrightarrow[t=\infty]{\mathcal{L}^+} f_{\mathcal{L}^+}$ using

$$f_{\mathcal{L}^+}(x) \approx f^{lin}_{\mathcal{L}^+}(x) = f_{\mathcal{L}}(x) + \Theta_{\mathcal{L}}(x, \mathcal{X}^+)\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)^{-1}\left(\mathcal{Y}^+ - f_{\mathcal{L}}(\mathcal{X}^+)\right) \tag{2.11}$$

where $\Theta_{\mathcal{L}}$ is the empirical NTK obtained from the parameters of $f_{\mathcal{L}}$.

Efficient computation. To avoid computing the full empirical NTK with the shape of $LC \times LC$ where L is the size of the labeled set and C is the number of classes, we use the "single- logit" approximation as explained in Wei et al. [72] (Section 2.3) in which we only compute the corresponding NTK of the first logit of the network, i.e. $\Theta(x, y) = \nabla_\theta f^{(1)}(x)^\top \nabla_\theta f^{(1)}(y) \otimes I_C$ where $f^{(1)}(x)$ refers to the first neuron of the output of f on the datapoint x. For a kernel regression, we can take the kronecker product out, which decreases the memory and time complexity of computing the NTK by an order of $\mathcal{O}(C^2)$.

To compute (2.11), we must solve linear systems with $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$. This can also be done efficiently using the block structure of $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$: letting $\mathbf{v} = \Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})\Theta_{\mathcal{L}}(\mathcal{X}, x')$ and $u = \Theta_{\mathcal{L}}(x', x') - \Theta_{\mathcal{L}}(x', \mathcal{X})\Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})^{-1}\Theta_{\mathcal{L}}(\mathcal{X}, x')$,

$$f^{lin}_{\mathcal{L}^+}(x) = f^{lin}_{\mathcal{L}}(x) + \frac{1}{u}\left(\Theta_{\mathcal{L}}(x, \mathcal{X})\mathbf{v} - \Theta_{\mathcal{L}}(x, x')\right)\left(\mathbf{v}^T(\mathcal{Y} - f_{\mathcal{L}}(\mathcal{X})) - (y' - f_{\mathcal{L}}(x'))\right). \tag{2.12}$$

Thus, rather than inverting $\Theta_{\mathcal{L}}(\mathcal{X}^+, \mathcal{X}^+)$ for each $x \in \mathcal{U}$, we can invert $\Theta_{\mathcal{L}}(\mathcal{X}, \mathcal{X})$ only once, then use some matrix multiplications to find $f^{lin}_{\mathcal{L}^+}(x)$ for each query point.

Time complexity. We now provide the time complexity of the NTK approximation proposed in Equation (2.11) by using efficient block computation in Equation (2.12). Let L be the size of the labeled set, U of the unlabeled set, P the number of model parameters, and E the number of training epochs. The computation of $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{U}}, \mathcal{X}_{\mathcal{L}})$, $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})$, and $\Theta_{\mathcal{L}}(\mathcal{X}_{\mathcal{L}}, \mathcal{X}_{\mathcal{L}})^{-1}$ take $\mathcal{O}(LUP)$, $\mathcal{O}(L^2P)$ and $\mathcal{O}(L^3)$ time, respectively. If we compute (2.12) for all $x \in \mathcal{U}$, the matrix multiplication takes

Algorithm 1: Active learning using NTKs

Input: Initialize a model f_0, labeled pool $\mathcal{L}_0$, and unlabeled pool $\mathcal{U}_0$
Train f_0 on the initial labeled set $\mathcal{L}_0$, using SGD, to obtain $f_{\mathcal{L}_0}$
for $t = 0$ **to** $T - 1$ **do**
  Compute $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{U}_t}, \mathcal{X}_{\mathcal{L}_t})$, $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{L}_t}, \mathcal{X}_{\mathcal{L}_t})$ and $\Theta_{\mathcal{L}_t}(\mathcal{X}_{\mathcal{L}_t}, \mathcal{X}_{\mathcal{L}_t})^{-1}$
  for x' **in** $\mathcal{U}_t$ **do**
    Estimate the label for x' as $y' = \arg\max f_{\mathcal{L}_t}(x')$
    Approximate $f_{\mathcal{L}_t \cup (x', y')}$ using (2.11) and (2.12) for faster computation
    Track x^* as the point with maximal $A_{\textit{MLMOC}}(x', f_{\mathcal{L}_t}, \mathcal{L}_t, \mathcal{U}_t)$ we've seen so far
  end for
  Obtain the label y^* for x^* from the oracle
  Update the labeled set $\mathcal{L}_{t+1} = \mathcal{L}_t \cup \{(x^*, y^*)\}$ and unlabeled set $\mathcal{U}_{t+1} = \mathcal{U}_t \setminus \{(x^*, y^*)\}$
  Train $f_{\mathcal{L}_t}$ on $\mathcal{L}_{t+1}$, using SGD, to obtain $f_{\mathcal{L}_{t+1}}$
end for

$\mathcal{O}(UL^2)$. Putting altogether, the time complexity of the NTK approximation is $\mathcal{O}(LUP + L^2P + L^3 + UL^2)$. Since we assume $P > \max(L, U)$ for overparameterized models, the time complexity can be simplified to $\mathcal{O}(LUP + L^2P)$.

The dominant term for naïve SGD retraining is $\mathcal{O}(LUPE)$ whereas the proposed method using block structure is dominated by $\mathcal{O}(LUP)$ time if $U > L$, otherwise $\mathcal{O}(L^2P)$ time. If $U > L$, then the proposed method is faster by an order of $\mathcal{O}(E)$. In the other case, if $L > EU$, one can use the conjugate gradient kernel regression solvers as in Rudi et al. [61] and Gardner et al. [20] that further reduce the time complexity of our proposed method to $\mathcal{O}(\max(PL\sqrt{L}, PU\sqrt{U}))$ depending on whether $L > U$ or $L < U$, which is faster than the naïve SGD training by an order of $\mathcal{O}(UE/\sqrt{L})$ or $\mathcal{O}(LE/\sqrt{U})$.

MLMOC querying. Instead of the EMOC acquisition function (2.3), we define a new function more suitable for a linearized network, which we term *Most Likely Model Output Change* (MLMOC):

$$A_{\textit{MLMOC}}(x', f_{\mathcal{L}}, \mathcal{L}_t, \mathcal{U}_t) := \sum_{x \in \mathcal{U}} \| f_{\mathcal{L}}(x) - f^{\textit{lin}}_{\mathcal{L}^+}(x)) \|_2 \tag{2.13}$$

where $\mathcal{L}^+ = \mathcal{L} \cup \{(x', y')\}$ with $y' = \arg\max f_{\mathcal{L}}(x')$. We only consider the most likely pseudo-label, instead of the expectation; this saves a significant amount of computation, by a factor of the number of classes, but empirically does not hurt accuracy. We also suspect that, when $f_{\mathcal{L}}$ is already reasonably accurate, we can "trust" the most likely label more than we can trust accurate estimation of probabilities for low-probability losses, especially when training networks with L_2 loss. Algorithm 1 shows pseudo-code for the proposed algorithm.

Sequential query strategy. In the active learning literature, each query of new data points is essentially always followed by retraining the underlying model. This, however, may not be practical for the cases where annotation is quick but SGD training of a large neural network is slow. Although there has been significant effort towards batch selection strategies based on finding diverse examples to query in a batch [e.g. 6, 8, 32], which among other benefits can minimize the number of times we must retrain with SGD, they cannot fully take advantage of low-latency annotations if they are available. This may be the case if, for instance, labeling tasks can be quickly pushed out to a pool of human labelers available for many such annotation tasks at once, or other cases where a limited number of queries are desired to avoid expense, damage to a system being measured, or so on, but those individual measurements can still be taken promptly.

Our proposed linearized network can take advantage of sequentially added annotations without requiring SGD retraining steps. Given an empirical NTK Θ, adding another training point (x, y) requires nothing more than adding a row and column to the kernel. As a result, computing f^{lin} and following A_{MLMOC} with an additional data point (x, y) is almost instant. This approximation can utilize the information from the label y directly while batch selection strategies cannot. After adding a substantial number of points, we will want to retrain, but it is not needed after every new data point. We experimentally demonstrates the effectiveness of this sequential query strategy in Section 2.4.

2.3 Related Work

Two main approaches in pool-based active learning are uncertainty and representation-based methods. The general goal of the uncertainty-based methods is to query the most "informative" data points, thus, their acquisition functions vary depending on how to measure the informativeness of a data point. As simple but effective uncertainty-based querying methods, posterior probability [38, 39], entropy [70], margin sampling [62] and least confident [64], have been widely used in practice. They do not, however, take into account how a candidate data point would change a model, or how this change would interact with unseen examples. More advanced querying methods have thus been proposed as discussed in Section 2.1. There are also Bayesian uncertainty-based methods, particularly Bayesian Active Learning by Disagreement (BALD) [27], which measures the mutual information between a new data point and model parameters, which Gal et al. [19] propose to estimate efficiently with Monte Carlo dropout networks [18].

The goal of the representation-based methods is to query examples that are the most "representative" among the unlabeled set $\mathcal{U}$, hoping that doing well on those examples leads to doing well on the whole unseen dataset. Sener and Savarese [63] define a core set as a batch of images that minimize the distance between unlabeled and labeled images if added to the labeled pool. Kirsch et al. [32] further expand BALD to a batch setting using a greedy algorithm. Bıyık et al. [8] propose to use determinantal point processes to target diversity of the queried examples while maintaining informativeness. BADGE [6] measures the uncertainty using a gradient norm but also encourage the diversity of queried images using k-means++ seeding [5].

Other approaches include that of Yoo and Kweon [79], who employ an additional loss prediction module to an existing neural network. However, the additional loss computation module can make training unstable (as we observed in some experiments). Tran et al. [68] improves BALD by adding a variational autoencoder and auxiliary classifier generative adversarial network, from which a synthesized data point x' similar to the queried image x^* is generated.

Our proposed method is different from the previous works in that it "looks one step ahead" instead of relying on the current state of a model. Perhaps the most

similar proposal to ours is that of Borsos et al. [9], who propose using the infinite NTK to solve a bi-level optimization problem for semi-supervised batch active learning. Aside from the difference in our objective functions, however, using the infinite NTK to approximate a neural network in active learning does not seem to be a good choice; as the size of training set increases, the error of the approximation can be quite large. We explore the difference between infinite and empirical NTKs experimentally in Section 2.4.

2.4 Experiment Results

Datasets. To demonstrate the effectiveness of the proposed method, we provide experimental results on three benchmark datasets for classification tasks: MNIST [14], SVHN [47], CIFAR10 [34], and CIFAR100 [34]. MNIST consists of 10 handwritten digits with $60,000$ training and $10,000$ test images, at size 28×28. SVHN also consists of 10 digit numbers with $73,257$ training and $26,032$ test images, at size 32×32; it is more challenging than MNIST, containing images from pictures of house numbers, with much more variation and many distractions present. CIFAR10 contains $50,000$ training and $10,000$ test images, also 32×32 and equally split between 10 classes like `airplane`, `frog`, and `truck`. CIFAR100 is the same as the CIFAR10, except it has 100 classes; the resolution is still 32×32, and images are equally split between 100 classes. The 100 classes are grouped with 20 super-classes, but we use only the 100 sub-classes in this work.

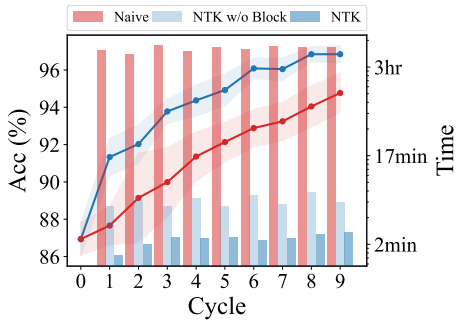
Implementation. We implement a pipeline for the proposed method using PyTorch [57] and Jax [10]; our neural networks f are implemented in PyTorch, whereas the linearized models f^{lin} are implemented in Jax with the `neural-tangents` library [52]. To make this possible, we wrote a model-migrator that automatically. transfers a PyTorch model to a Jax model. We employ a ResNet18 [24] and WideResNet [80] with one or two layers and maximum width of 640, which is wide enough for a linearized neural network to be a good approximation, while still being powerful enough for these datasets. As we will see in our experiments, even for narrower networks, the proposed method outperforms random acquisition

strategy by large margins and improves or achieves equal performance as other state-of-the-art strategies.

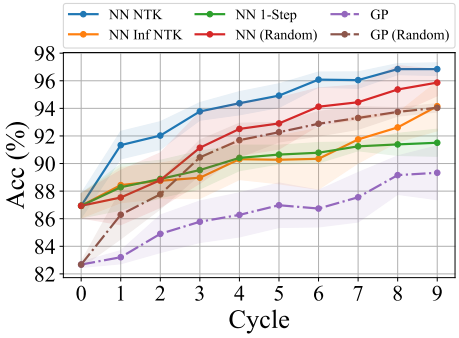
As mentioned earlier, we use L_2 loss, rather than cross-entropy; this allows for a faster NTK approximation, as in the formulas of Section 2.2, rather than requiring differential equation solvers. To ensure that the empirical NTK of the training data ($\mathcal{X}_{\mathcal{L}}$) is positive semidefinite, in batch normalization layers, we freeze the current running statistics during computation of eNTK. We implement LL4AL [79] and BADGE [6] based on their publicly available code. We provide the implementation for the full pipeline in https://github.com/mohamad-amin/ntk-lookahead-active-learning.

Query scheme. Following active learning literature, for each query step, we randomly draw a subset from an unlabeled set, and query a fixed number of examples from the subset. To build a batch, we take the points with the highest A_{MLMOC} scores, which we found to perform well without any batch diversity requirements. At each cycle, we query 20, $1\,000$, $1\,000$, and $1\,000$ new data points on MNIST, SVHN, CIFAR10, and CIFAR100 from the subset of $4\,000$, $6\,000$, $6\,000$, and $6\,000$ unlabeled data points, and initialize the labeled set $\mathcal{L}$ with 100, $1\,000$, $1\,000$, and $10\,000$ data points.

Making look-ahead acquisitions feasible. In Figure 2.1a, we demonstrate the NTK approximation indeed makes it feasible to use look-ahead acquisition functions. We run the MLMOC acquisition function with naïve SGD retraining (Naive) and proposed NTK approximation with (NTK) and without (NTK w/o Block) block computation in (2.12) on MNIST. For the naïve version, we retrain the neural network for 15 epochs using SGD. The line plots represent accuracy whereas bar plots represent wall-clock time to query data, using a NVIDIA V100 GPU. The NTK approximation is not only computationally efficient (more than 100 times faster than Naïve) but also yields significantly more accurate models. Although one might expect that Naïve is an upper bound for the NTK approximation, as it actually "looks ahead," in reality it is hard to retrain a neural network until convergence for every candidate data point; a "proper" look-ahead method should perhaps look at an



(a) Naïve look-ahead acquisition versus NTK approximation. Bars show runtime per cycle. Accuracy of NTK and NTK w/o Block are the same.



(b) MLMOC on various models, along with random-acquisition baselines.

Figure 2.1: Comparisons of several related approaches on MNIST.

Methods	NN NTK	NN 1-step	NN Inf NTK	GP
NN NTK	1.0000	0.0109	0.0093	0.0014
NN 1-step	0.0109	1.0000	0.6863	0.0422
NN Inf NTK	0.0093	0.6863	1.0000	0.0742
GP	0.0014	0.0422	0.0742	1.0000

Table 2.1: The p-values of the post-hoc paired t-test.

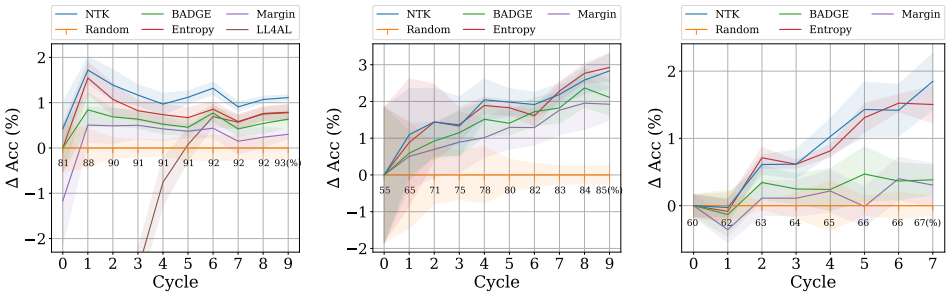
ensemble of several training runs, with different learning rate schedules and other hyperparameters.

Comparison of look-ahead acquisitions. Having seen that the empirical NTK effectively approximates and even outperforms the Naïve look-ahead acquisition function, we now show the superiority of the empirical NTK over existing look-ahead methods. In Figure 2.1b, we provide four look-ahead methods using MLMOC acquisition function along with their corresponding random baselines (Random). NN refers to a neural network-based model whereas GP refers to a model based on Gaussian process. Instead of using empirical NTK as we propose (denoted as NN NTK, blue), Borsos et al. [9] use infinite NTK (NN Inf NTK, orange). Käding

et al. [30] approximate SGD retraining using only one-step SGD update (NN 1-step, green) whereas Käding et al. [31] exactly compute look-ahead acquisition using GP for regression problems. We modify their model for classification tasks, use an infinite NTK, and denote it as GP (purple). As shown in Figure 2.1b, ours is the only of these methods to outperform random baselines here. Neither infinite NTK nor 1-step approximation are good approximation for retraining steps, as the former lacks information about learnt representation by the network and the latter hasn't been trained long enough on the new datapoints, especially as more datapoints are added. Also, although GP [31] works well on regression tasks, it struggles on classification tasks.

To validate the difference between different look-ahead strategies, we conduct Friedman's test followed by post-hoc paired t-test on the existing look-ahead strategies in Figure 2.1b: NN NTK, NN 1-step, NN Inf NTK, and GP. Friedman's test is a non-parametric statistical test. The null hypothesis in our case is that the population mean of the performance of each look-ahead strategy is the same and the alternative hypothesis is that at least one population mean is different from the rest of look-ahead strategies. The p-value of the Friedman's test is 0.0045 (< 0.05). Hence, we reject the null hypothesis. As it means at least one population mean is different from the rest of look-ahead strategies, we further conduct the post-hoc paired t-test to check where the significant difference is coming from. The result of the post-hoc paired t-test is shown in Table 2.1. Here, the null hypothesis for each pair is that the population mean of two methods is the same. As all the p-values between ours (NN NTK) and the other methods are less than the conventional significance level 0.05, we conclude that our proposed method is significantly better than the other look-ahead strategies.

Comparison with state-of-the-art. In Figure 2.2, we compare the proposed NTK active learning to state-of-the-art methods – Random, Entropy, Margin sampling (which chooses datapoints based on the margin between the probability of the two most-probable classification classes according to the model) [58], BADGE [6], and LL4AL [79] – on SVHN, CIFAR10, and CIFAR100 with different architectures; we provide additional results in Appendix A.2. For clarity, we show the difference between each method and Random. The numbers below Random line give the



(a) SVHN: 1-layer WideResNet

(b) CIFAR10: 2-layer WideResNet

(c) CIFAR100: Original ResNet18

Figure 2.2: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows difference from random acquisition, whose accuracy is shown in text.

accuracies of Random at each cycle. As mentioned earlier, training LL4AL is often unstable due to loss computation module; in particular, it performs too poorly to show on the same plot for Figure 2.2b and Figure 2.2c. We show the instability of training LL4AL with varying learning rate in Appendix A.3. Each experiment is run for six different seeds; lines show the mean performance, and shading shows a 95% confidence interval for the mean. Figure 2.2(a) shows that the NTK method outperforms all the state-of-the-art methods throughout training on SVHN. A similar pattern is observed on CIFAR10 and CIFAR100; NTK is consistently comparable to or better than the best competitor methods, far outperforming existing look-ahead approaches. Although entropy has quite similar performance to our proposed method in some settings, we believe that our method still has an advantage due to applicability of it in sequential active learning settings and better interpretability by locality of analysis.

Sequential query strategy. As mentioned in Section 2.2, one great advantage of the NTK approximation is that it can exploit additional annotations without needing to run SGD training for a neural network, which cannot be done in existing active learning methods for neural networks. On MNIST, Section 2.4 compares sequential NTK where the oracle provides true label for every new queried data point, to batch NTK where annotation is done batch-wise as with a normal active learning setting.

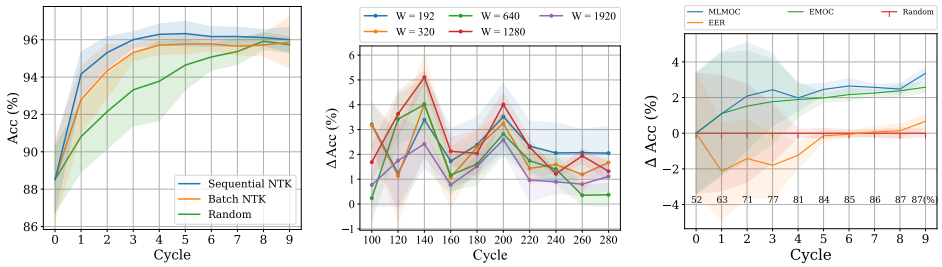


Figure 2.3: Further comparisons of algorithm variants on MNIST. **Left:** Sequential *vs* batch querying. **Middle:** Performance with varying width. **Right:** Different look-ahead strategies.

Sequential NTK outperforms batch NTK; especially, sequential NTK reaches more than 94% in one cycle, and in cycle 3, it has already reached 96% accuracy, where batch NTK plateaus. We expect the sequential NTK can be useful in practice when annotation is much faster than SGD training.

Varying model widths. As shown in Equation (2.9), the NTK approximation gets closer to the dynamics of a neural network as the width of the neural network increases. But, since very wide networks are computationally expensive in practice, it is important to empirically demonstrate that the NTK approximation works even with reasonably narrow networks. To this end, we vary the maximum width of the WideResNet with one block layer we use for the experiments as shown in Section 2.4. We can observe that regardless of the maximum width, the NTK method significantly outperforms Random strategy until convergence, indicating that the NTK approximation is good enough for the purpose of active learning.

Comparison of different acquisition strategies. We compare different look-ahead acquisition functions in Section 2.4. As mentioned with Equation (2.13) in Section 2.2, MLMOC and EMOC are not significantly different. As MLMOC is computationally cheaper than EMOC proportional to the number of classes, we use MLMOC over EMOC as an acquisition function. But note that, the proposed NTK approximation is applicable to any look-ahead acquisition functions including MLMOC, EMOC and EER. Thus, we hope that our method would provide a framework for other researchers to build upon and propose various look-ahead

active learning methods or enhance the look-ahead approximation to help further tackle the problem of pool-based active learning.

Chapter 3

Pseudo-NTK: A Novel Approximation to the Empirical NTK

Despite their effectiveness in gaining new theoretical insights about NNs or devleoping new practical applications, NTKs of practical networks are extremely challenging to compute, and usually not computationally feasible. The "empirical" NTK (eNTK; we discuss the difference from what others term "the NTK" shortly) is

$$\Theta_\theta(x_1, x_2) = \left[J_\theta\left(f_\theta(x_1)\right)\right]\left[J_\theta\left(f_\theta(x_2)\right)\right]^\top, \tag{3.1}$$

where $J_\theta\left(f_\theta(x)\right)$ denotes the Jacobian of the function f at a point x with respect to the flattened vector of all its parameters, $\theta \in \mathbb{R}^P$. If D is the input dimension of f and O the number of outputs, we have $J_\theta\left(f_\theta(x)\right) \in \mathbb{R}^{O\times P}$ and $\Theta_\theta(x_1, x_2) \in \mathbb{R}^{O\times O}$. Thus, computing the eNTK between N_1 and N_2 data points yields N_1N_2 matrices, each of shape $O \times O$; we usually arrange this as an $N_1O \times N_2O$ matrix.

When computing an eNTK on tasks involving large datasets and with multiple output neurons, *e.g.* in a classification model with O classes, the eNTK quickly becomes impractical regardless of how fast each entry is computed due to its $NO \times NO$ size. The full eNTK of a classification model even on the relatively small CIFAR-10 dataset [33], stored in double precision, takes over 1.8 terabytes in

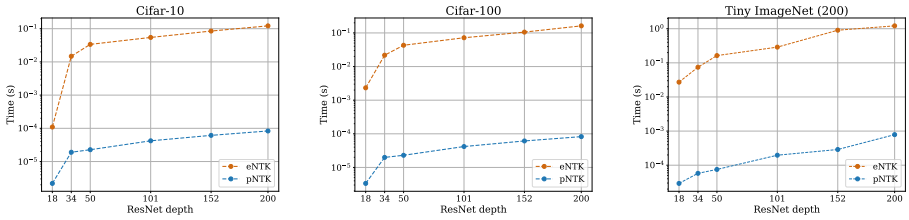


Figure 3.1: Wall-clock time to evaluate the eNTK and pNTK for one pair of inputs, across datasets and ResNet depths.

memory. For practical usage, we need something better.

This work presents a simple trick for a strong approximation of the eNTK that removes the O^2 from the size of the kernel matrix, resulting in a factor of O^2 improvement in the memory and up to O^3 in computation. Since for typical classification datasets O is at least 10 (*e.g.* CIFAR-10) and potentially $1{,}000$ or more [*e.g.* ImageNet, 13], this provides multiple orders of magnitude savings over the full eNTK. We prove this approximation converges to the original eNTK at a rate of $\tilde{\mathcal{O}}(n^{-1/2})$ for a standard-initialization NN of depth L and width n in each layer, and the predictions of kernel regression with the approximate kernel do the same. We also conduct diverse experimental investigations to support our theoretical results, across a range of architectures and settings. We hope this approximation further enables researches to employ NTKs towards theoretical and empirical advances in wide networks.

Infinite NTKs. In the infinite-width limit of appropriately initialized NNs, Θ_θ converges almost surely at initialization to a particular kernel, and remains constant through training. Algorithms are available to compute this expectation exactly, but they tend to be substantially more expensive than computing (3.1) directly for all but extremely wide networks. The convergence to this infinite-width regime is slow in practice, and moreover it eliminates some of the interest of the framework: neural architecture search, predicting generalization of a pre-trained representation, and meta-learning are all considerably less interesting when we only consider infinite-width networks that do essentially no feature learning. Thus we focus here only on the "empirical" eNTK as in (3.1).

3.1 Related Work

Among the numerous recent works that have used eNTKs either to gain insights about various phenomenons in deep learning or to propose new algorithms, not many have publicized the computational costs and implementation details of computing eNTKs. Nevertheless, all are in agreement about the expense of such computations [15, 26, 56].

Several recent works have, mostly "quietly," employed various techniques to avoid dealing with the full eNTK matrix; however, to the best of our knowledge, none provide any rigorous justifications. Wei et al. [72, Section 2.3] point out that if the final layer of a NN is randomly initialized, the *expected* eNTK can be written as $K_0 \otimes I_O$ for some kernel K_0, where I_O is the $O \times O$ identity matrix and $\otimes$ is the Kronecker product. Thus, they use the approximation in which they only compute the eNTK with respect to one of the logits of the NN. Although their approach to approximating the eNTK is similar to ours, they don't provide any rigorous bounds or empirical study of how closely the actual eNTK is approximated by its expectation in this regard. Wang et al. [71] employs the same "single-logit" strategy, though they only mention the infinite-width limit as a motivation supporting their trick. Despite these claims, we will see in our experiments that the eNTK is generally *not* diagonal. We will, however, prove upper bounds on distance of our approximation to the eNTK, and provide experimental support that this approximation captures the behaviour of the eNTK even when the NN's weights are not at initialization. Park et al. [56] and Chen et al. [11] also seem to use a form of "single-logit" approximation to eNTK, without explicitly mentioning it. Lee et al. [36], by contrast, do use the full eNTK, and hence never compute the kernel on more than 256 datapoints.

Novak et al. [54] recently performed an in-depth analysis of computational and memory complexity required for computing the eNTK, and proposed two new approaches (depending on the NN architecture) to reduce the time complexity, but not the memory burden, of computing the eNTK over explicitly implementing (3.1). Our approaches are complementary; in fact, we use their "structured derivatives" method to help compute our approximation.

3.2 Pseudo-NTK

We define the pseudo-NTK (pNTK), which we denote as $\hat{\Theta}_\theta(x_1, x_2)$, as

$$\underbrace{\left[\nabla_\theta \frac{1}{\sqrt{O}} \sum_{i=1}^{O} f_\theta^{(i)}(x_1)\right]}_{1\times P} \underbrace{\left[\nabla_\theta \frac{1}{\sqrt{O}} \sum_{i=1}^{O} f_\theta^{(i)}(x_2)\right]^\top}_{P\times 1}, \tag{3.2}$$

where $f_\theta^{(i)}(x)$ denotes the i-th output of f_θ on the input x. While the eNTK is a matrix-valued kernel for each pair of inputs, the pNTK is a traditional scalar-valued kernel.

Some recent work [2, 71, 72, 77] has pointed out that in the infinite width limit $\lim_{n\to\infty} \Theta(x_1, x_2)$, the NTK becomes a constant-diagonal matrix, where the class-class component becomes identity. Thus, one can avoid computing the off-diagonal entries of the infinite-width NTK of each pair through using $\Theta_\theta(x_1, x_2) = \hat{\Theta}_\theta(x_1, x_2) \otimes I_O$, giving a drastic $\mathcal{O}(O^2)$ time and memory complexity decrease.

Practitioners have accordingly used the same approach in computing the eNTK of a finite width network, but with little to no further justification. We see in our experiments that for finite width networks, the NTK is **not** diagonal. In fact, we show that for most practical networks, it is very far from being diagonal, casting doubts on the validity of arguments justifying the approximation with asymptotic diagonality. We justify this category of approximation with theoretical bounds on the difference of the true NTK from the approximation (3.2), which we also call "sum of logits."

Before our formal results and experimental evaluation, we give some intuition. First, suppose $f_\theta^{(i)}(x) = \phi(x) \cdot v_i$, so that $v_i \in \mathbb{R}^{n_{L-1}}$ is the ith row of a linear read-out layer; then $\frac{1}{\sqrt{O}} \sum_{i=1}^{O} f_\theta^{(i)}(x) = \phi(x) \cdot \left[\frac{1}{\sqrt{O}} \sum_{i=1}^{O} v_i\right]$. If the $v_i \sim \mathcal{N}(0, \sigma^2 I_{n_{L-1}})$ are independent, $\frac{1}{\sqrt{O}} \sum_{i=1}^{O} v_i$ has the same normal distribution as, say, v_1. Thus, at initialization, our sum of logits approximation agrees in distribution with the first-logit approximation. Our proof uses the sum-of-logits form, though, and we believe it may be more sensible for networks that are not at random initialization.

Calling this vector (whether the first logit or sum of logits) v, we can think of

(3.2) as the NTK of a model with a single scalar output as a function of ϕ, whose last layer has weights v. When we linearize a network with that kernel for an O-class classification problem, getting the formula (3.5) discussed in Section 3.3.3, we end up effectively using a one-vs-rest classifier scheme. Thus, we can think of the pseudo-NTK as approximating the process of training O one-vs-rest classifiers, rather than a single O-way classifier.

3.3 Approximation Quality of Pseudo-NTK

We will now study various aspects of the approximation of (3.2) to (3.1), both in theory and empirically. Our experiments compare different characteristics of pNTK and eNTK, both at initialization and throughout training. We evaluate four widely-used architectures: FCN [a fully-connected network of depth 3, as in 36, 37], ConvNet [a fully-convolutional network of depth 8, as in 2, 3, 37], ResNet18 [25], and WideResNet-16-k [80]. We evaluate each architecture at different widths, as mentioned in the plot legends: we show exact widths for FCN, while for others we show a widening factor. For consistency with most other recent papers studying NTKs and properties of NNs in general, we focus on data from CIFAR-10 [33]. Each experiment is repeated using three seeds; means and corresponding error bars are also shown, except when they interfered with clear interpretation of the plots. All models are trained for 200 epochs, using stochastic gradient descent (SGD), on 32GB NVIDIA V100 GPUs. More details on models and optimization are provided in Appendix A.6. The measured statistic for each experiment are reported after 0, 50, 100, 150, and 200 epochs.

A Note On Parameterization. In order to be maximally applicable to practical implementations, both our experiments and our theoretical results are based on standard parameterization ("fan-in" variance). Although most related work uses the so-called NTK parameterization ("fan-out" variance), this is rarely used in practice while training NNs, mostly due to the poor generalization results achieved in comparison to training with standard parameterization [55, Section I]. We encountered similarly poor behaviour when training NNs with NTK parameterization, but note that our theorems could also be adapted to the fan-out case.

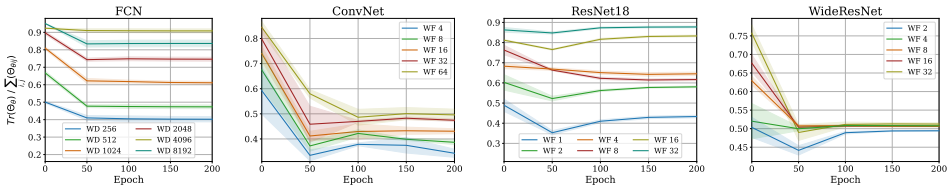


Figure 3.2: Comparing the **magnitude of sum of on-diagonal and off-diagonal elements of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. The reported numbers are the average of 1000×1000 matrices, each having a shape of 10×10. The same subset has then been used to train the NN using SGD. As the NN's width grows, the eNTK converges to being diagonal at initialization among all different architectures.

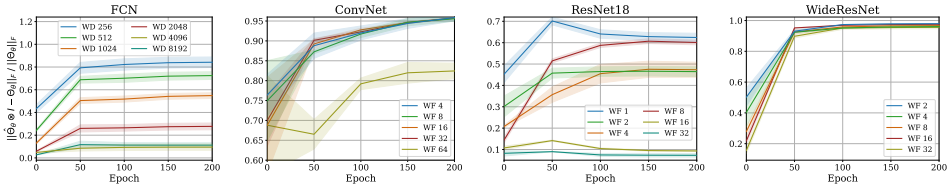


Figure 3.3: Evaluating the **relative difference of Frobenius norm of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D}) \otimes I_O$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have more similar $\|\Theta_\theta\|_F$ and $\|\hat{\Theta}_\theta \otimes I_O\|_F$ at initialization.

3.3.1 pNTK Converges to eNTK as Width Grows

The first crucial thing to verify is whether the pNTK kernel matrix approximates the true eNTK as a whole. We study this first in terms of Frobenius norm.

Theorem 3.3.1 (Informal)**.** *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^O$ be a fully-connected network with layers of width n whose parameters are initialized as in He et al. [23], with ReLU-type activations. Let $\hat{\Theta}_\theta(x_1, x_2)$ be the pNTK of f_θ as in* (3.2) *and $\Theta_\theta(x_1, x_2)$ the eNTK as in* (3.1) *for a fixed pair of inputs x_1, x_2. With high probability over the initialization,*

$$\frac{\|\hat{\Theta}_\theta(x_1, x_2) \otimes I_O - \Theta_\theta(x_1, x_2)\|_F}{\|\Theta_\theta(x_1, x_2)\|_F} \in \tilde{\mathcal{O}}(n^{-\frac{1}{2}}). \tag{3.3}$$

Remark 3.3.2. All of the results in the paper can be straightforwardly extended to networks with different widths, as long as the consecutive layers' widths satisfy $n_{l+1} = \Theta(n_l)$. Moreover, the results can be made architecturally universal with the techniques of Yang [77], Yang and Littwin [78].

Theorem 3.3.1 provides the first upper bound on the convergence rate of pNTK towards eNTK. A formal statement for Theorem 3.3.1 and its proof are in Appendix A.7.2.

Remark 3.3.3. Based on the provided proof, it is straightforward that the ratio of information between off-diagonal and on-diagonal elements of the eNTK matrix converges to zero with a rate of $\mathcal{O}(n^{-\frac{1}{2}})$ with high probability over random initialization, as depicted in Figure 3.2.

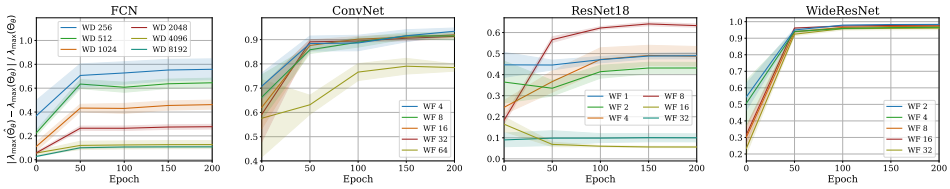


Figure 3.4: Evaluating the **relative difference of** $\lambda_{\max}$ **of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D})$ at initialization and throughout training, based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have more similar $\lambda_{\max}(\Theta_\theta(\mathcal{D}, \mathcal{D}))$ and $\lambda_{\max}(\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D}))$.

Figure 3.2 shows that as the width grows, the sum of off-diagonal elements of $\Theta_\theta(x_1, x_2)$ becomes small compared to the diagonal. Furthermore, Figure 3.3 provides experimental support that as width grows, $\hat{\Theta}_\theta \otimes I_O$ converges to Θ_θ in terms of relative Frobenius norm.

Theorem 3.3.1 only applies to epoch zero of these figures, as it assumes networks with weights at initialization. As can be seen in the figures, these results do not necessarily apply to the NNs not at initialization (i.e., after a few epochs of training). This naturally gives rise to the question: *Can the pNTK be used to analyze and represent NNs whose parameters are far from initialization?* We will now take various experimental approaches towards studying this question.

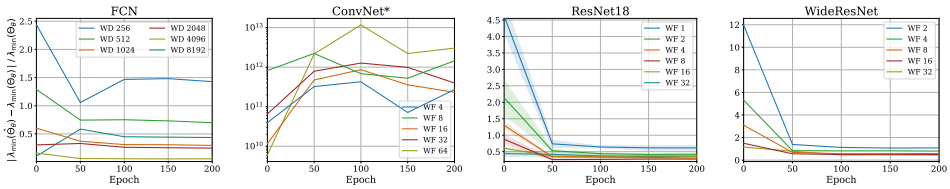


Figure 3.5: Evaluating the **relative difference of** $\lambda_{\min}$ **of** $\Theta_\theta(\mathcal{D},\mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D},\mathcal{D})$ based on $\mathcal{D}$ being 1000 random points from CIFAR-10. Wider nets have more similar $\lambda_{\min}(\Theta_\theta(\mathcal{D},\mathcal{D}))$ and $\lambda_{\min}(\hat{\Theta}_\theta(\mathcal{D},\mathcal{D}))$. Note, though, the extremely large values reported for ConvNet; as observed by Lee et al. [37] and Xiao et al. [75], it is ill-conditioned and $\lambda_{\min}(\Theta_\theta(\mathcal{D},\mathcal{D})) \to 0$, while $\lambda_{\min}(\hat{\Theta}_\theta(\mathcal{D},\mathcal{D})) > 0.001$, causing the huge discrepancy. More details in Appendix A.7.3.

3.3.2 Largest Eigenvalue Converges as Width Grows

As discussed before, the conditioning of a network's eNTK has been shown to be closely related to generalization properties of the network, such as trainability and generalization risk [72, 74, 75]. Thus, we would like to know how well the pNTK's eigenspectrum approximates that of the eNTK. The following theorem gives a bound on the rate of convergence between the maximum eigenvalues of the two kernels.

Theorem 3.3.4 (Informal)**.** *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^O$ be a fully-connected network with layers of width n whose parameters are initialized according to He et al. [23] initialization, with ReLU-type activations. Let $\hat{\Theta}_\theta(x_1, x_2)$ be the corresponding pNTK of f_θ as in* (3.2) *and $\Theta_\theta(x_1, x_2)$ the corresponding eNTK as in* (3.1) *for a fixed pair of inputs x_1, x_2. With high probability over the initialization,*

$$\frac{\lambda_{\max}(\hat{\Theta}_\theta(x_1,x_2)\otimes I_O) - \lambda_{\max}\left(\Theta_\theta\left(x_1,x_2\right)\right)}{\lambda_{\max}\left(\Theta_\theta\left(x_1,x_2\right)\right)} \in \tilde{\mathcal{O}}(n^{-\frac{1}{2}}).$$

Theorem 3.3.4 bounds the difference between the maximum eigenvalue of pNTK and the maximum eigenvalue of eNTK based on the NN's width, for networks at initialization. A formal statement for Theorem 3.3.4 and its proof are given in Appendix A.7.3. Figure 3.4 also supports this trend experimentally.

Figure 3.5 shows a similar trend for the minimum eigenvalues, although we

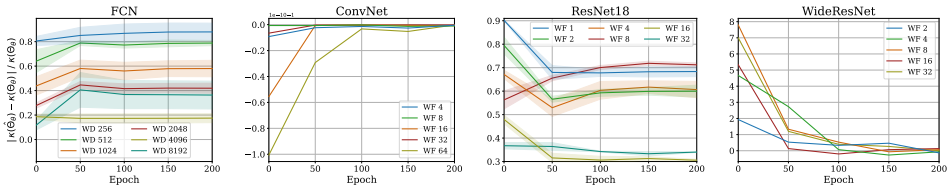


Figure 3.6: The **relative difference in condition number**, $\kappa(K) = \lambda_{\max}(K)/\lambda_{\min}(K)$, decreases for wider nets. The strange ConvNet plot is due to the issue in Figure 3.5; more details in Appendix A.7.3.

have not found a proof of this convergence. This suggests that the condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ should become similar as width grows; this is also supported by results in Figure 3.6.

Interestingly, the rate of increase/decrease in the difference between maximum and minimum eigenvalues and the condition numbers between pNTK and eNTK do not necessarily have a monotonic behaviour as the training goes on. Observing the exact values of $\lambda_{\min}$, $\lambda_{\max}$, and κ for different architectures, widths at initialization and throughout training reveals that in ConvNet, WideResNet and ResNet18 architectures, $\lambda_{\min}$ is close to zero at initialization, but grows during training; the inverse phenomenon is observed with FCNs. Further investigations of these statistics might reveal interesting insights about the behaviour of NNs trained with SGD and the connections between eNTK and trainibility of the architecture.

3.3.3 Kernel Regression Using pNTK vs. eNTK

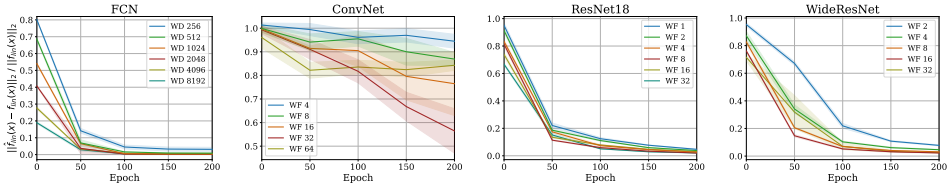


Figure 3.7: The **relative difference of kernel regression outputs**, (3.4) and (3.5), when training on $|\mathcal{D}| = 1000$ random CIFAR-10 points and testing on $|\mathcal{X}| = 500$. For wider NNs, the relative difference in $\hat{f}^{lin}(\mathcal{X})$ and $f^{lin}(\mathcal{X})$ decreases at initialization. Surprisingly, the difference between these two continues to quickly vanish while training the network.

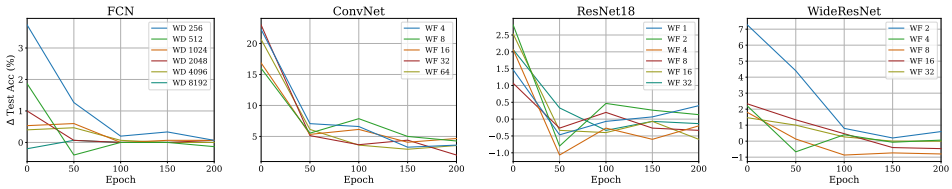


Figure 3.8: Using pNTK in kernel regression (as in Figure 3.7) **almost always achieves a higher test accuracy than using eNTK.** Wider NNs and trained nets have more similar prediction accuracies of $\hat{f}^{lin}$ and f^{lin} at initialization. Again, the difference between these two continues to vanish throughout the training process using SGD.

Lee et al. [36] proved that as a finite NN's width grows, its training dynamics can be well approximated using the first-order Taylor expansion of that NN around its initialization (a *linearized neural network*). Informally, they showed that when f is wide enough and trained on $\mathcal{D}$ with a suitably small learning rate, its predictions on x can be approximated by those of the linearized network f^{lin} given by

$$\underbrace{f_0(x)}_{O\times 1} + \underbrace{\Theta_0(x,\mathcal{D})}_{O\times NO}\ \underbrace{\Theta_0(\mathcal{D},\mathcal{D})^{-1}}_{NO\times NO}\ \underbrace{(\mathcal{Y}_\mathcal{D} - f_0(\mathcal{D}))}_{NO\times 1}, \tag{3.4}$$

where $\mathcal{Y}_\mathcal{D}$ is the matrix of one-hot labels for the training points $\mathcal{D}$, and Θ_0 is the eNTK of f at initialization f_0. This is simply kernel regression on the training data $\mathcal{D}$ using the kernel Θ_0 and prior mean f_0. Wei et al. [72] use the same kernel in a generalized cross-validation estimator [12] to predict the generalization risk of the NN. As discussed before, using the eNTK in these applications is practically infeasible, due to huge time and memory complexity of the kernel, but we show the pNTK approximates $f^{lin}(x)$ with much improved time and memory complexity.

Theorem 3.3.5 (Informal)**.** *Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^O$ be a fully-connected network with layers of width n whose parameters are initialized as in He et al. [23], with ReLU-type activations. Let $\hat{\Theta}_\theta(x_1, x_2)$ be the corresponding pNTK of f_θ as in* (3.2) *and $\Theta_\theta(x_1, x_2)$ the corresponding eNTK as in* (3.1) *for a fixed pair of inputs x_1, x_2. Define $\hat{f}^{lin}(x)$ as*

$$\underbrace{f_0(x)}_{1\times O} + \underbrace{\hat{\Theta}_\theta(x,\mathcal{D})}_{1\times N}\ \underbrace{\hat{\Theta}_\theta(\mathcal{D},\mathcal{D})^{-1}}_{N\times N}\ \underbrace{(\mathcal{Y}_\mathcal{D} - f_0(\mathcal{D}))}_{N\times O}. \tag{3.5}$$

After proper reshaping, with high probability over random initialization,

$$\|\hat{f}^{lin}(x) - f^{lin}(x)\|_F \in \tilde{\mathcal{O}}(n^{-\frac{1}{2}}). \tag{3.6}$$

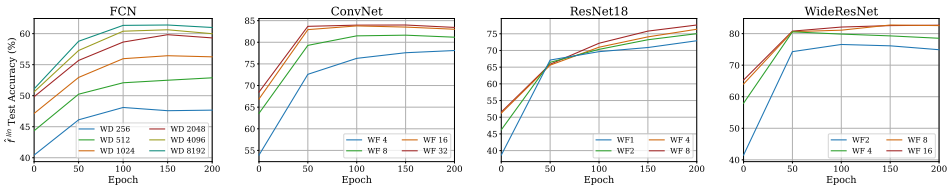


Figure 3.9: Evaluating the **test accuracy of kernel regression predictions using pNTK as in** (3.5) **on the full CIFAR-10 dataset**. As the NN's width grows, the test accuracy of $\hat{f}^{lin}$ also improves, but eventually saturates with the growing width. Using trained weights in computation of pNTK results in improved test accuracy of $\hat{f}^{lin}$.

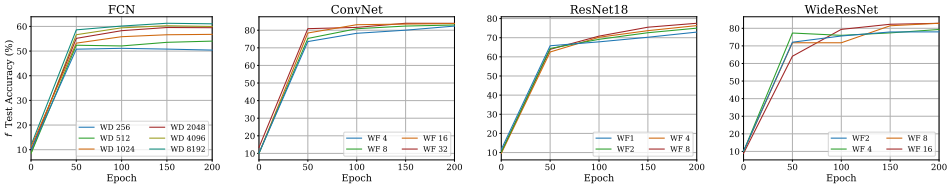


Figure 3.10: Evaluating the **test accuracy of model f throughout SGD training on the full CIFAR-10 dataset**. In contrast to $\hat{f}^{lin}$, the test accuracy of f does not significantly improve with growing width.

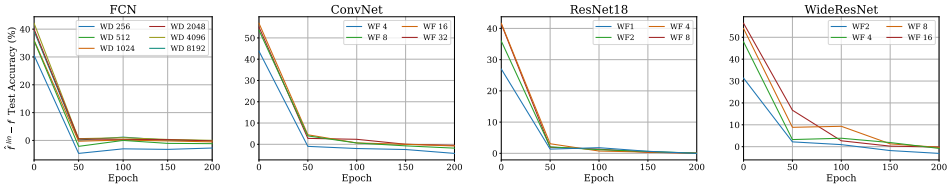


Figure 3.11: Evaluating the **difference in test accuracy of kernel regression using pNTK as in** (3.5) **vs the current model f** throughout SGD training on the full CIFAR-10 dataset: how much does a linearized predictor with the current representation improve prediction accuracy over the current model obtained by SGD?

A formal statement is given and proved in Appendix A.7.4.

Figure 3.7 also supports that as width grows, the predictions of kernel regression using $\hat{\Theta}_\theta$ converge to the prediction of those obtained from using Θ_θ, *while requiring orders of magnitude of less memory and time to compute*. Figure 3.8 shows similar results for the difference in prediction accuracies achieved using kernel regression through $\hat{\Theta}_\theta$ and Θ_θ kernels. Appendix A.7.4 also shows further analysis of how well the *linearized network* predicts the final accuracy of the trained model for each architecture and width pair. Although $\|\hat{f}^{lin}(x) - f^{lin}(x)\|_F$ decreases with width of the network in Figure 3.7 at initialization, this does not *necessarily* translate to a monotonic behaviour in prediction accuracies, a non-smooth function of the vector of predictions; we do see that the expected pattern more or less holds, however.

A surprising outcome depicted in Figures 3.7 and 3.8 is that while training the model's parameters, predictions of $\hat{f}^{lin}$ and f^{lin} converge very quickly. This is particularly intriguing, as it contrasts the results depicted in Figures 3.2 to 3.4 and 3.6. In other words, although the kernels Θ_θ and $\hat{\Theta}_\theta \otimes I_O$ seem to be diverging in Frobenius norm, eigenspectrum, and so on, kernel regression using those two kernels converges quickly, so that after 50 epochs the difference in predictions almost totally vanishes. We believe further investigation of why this phenomenon is observed could lead to new interesting insights about the training dynamics of NNs.

3.4 Application: Full Regression on CIFAR-10

Thanks to the reduction in time and memory complexity of pNTK over eNTK, motivated by Theorem 3.3.5 and experimental findings in Figure 3.8, we finally evaluate the corresponding pNTKs of the four architectures that we have used in our experimental evaluations in different widths using full CIFAR-10 data, at initialization and throughout training the models under SGD. As mentioned previously, running kernel regression with eNTK on all of CIFAR-10 would require evaluating 25×10^{10} Jacobian-vector products and more than ≈ 1.8 terabytes of memory; using pNTK, this can be done with a far more reasonable 25×10^8 Jacobian-vector products and ≈ 18 gigabytes of memory. This is still a heavy load compared to, say, direct SGD training, but is within the reach of standard compute nodes.

Figure 3.9 shows the test accuracy of $\hat{f}^{lin}$ on the full train and test sets of CIFAR-

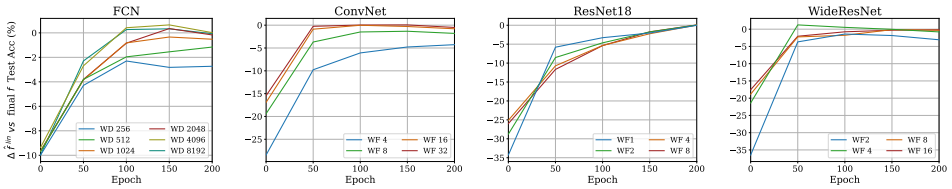


Figure 3.12: Evaluating the **difference in test accuracy of kernel regression using pNTK as in** (3.5) **vs the final model** f throughout SGD training on the full CIFAR-10 dataset. How much worse would it be to "give up" on SGD at this point and train $\hat{f}^{lin}$ with the current representation?

10. In the infinite-width limit, the test accuracy of $\hat{f}^{lin}$ at initialization (and later, because the kernel stays constant in this regime) should match the final test accuracy of f that is, the epoch 0 points in Figure 3.9 would agree with roughly the epoch 200 points in Figure 3.10. This comparison is plotted directly in Appendix A.8. Furthermore, the test accuracies of predictions of kernel regression using the pNTK are lower than those achieved by the NTKs of infinite-width counterparts for fully-connected and fully-convolution networks. This is consistent with results on eNTK by Arora et al. [2] and Lee et al. [37], although Arora et al. [2] studied only a "CIFAR-2" version.

It is worth noting from Figures 3.9 and 3.11 that, in contrast to the findings of Fort et al. [15], we observe that the corresponding pNTK of the NN continues to change even after epoch 50 of SGD training. Although for fully-connected networks and some versions of ResNet18, this change is not significant, in fully-convolutional networks and WideResNets, the pNTK continues to exhibit changes until epoch 150, where the training error has vanished. We remark that Fort et al. [15] analyzed eNTKs based on only 500 random samples from CIFAR-10, while the pNTK approximation has enabled us to run our analysis on the 100-times larger full dataset.

We hope that our contribution will enable further analyses and breakthroughs towards a stronger theoretical understanding of the training dynamics of deep NNs.

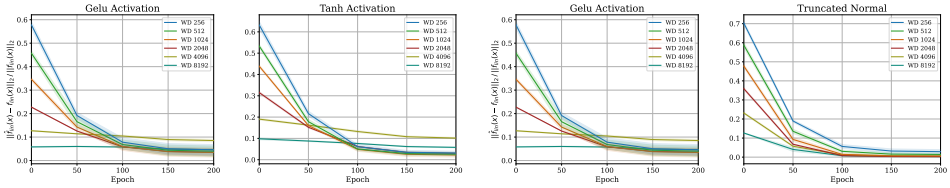


Figure 3.13: Evaluating the **relative norm difference of kernel regression outputs using eNTK and pNTK as in Equation (3.4) and Equation (3.5)** at initialization and throughout training. The kernel regression has been done on $|\mathcal{D}| = 1000$ training points and $|\mathcal{X}| = 500$ test points randomly selected from CIFAR-10's train and test sets.

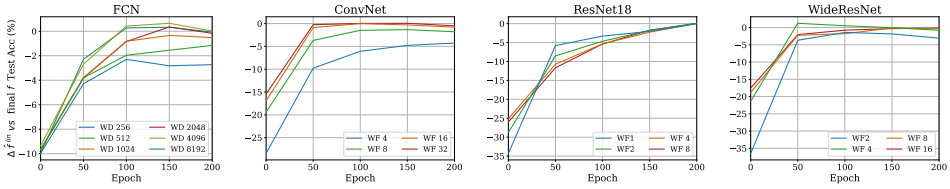


Figure 3.14: Evaluating the **difference in test accuracy of kernel regression using pNTK as in** (3.5) **vs the final model** f throughout SGD training on the full CIFAR-10 dataset. How much worse would it be to "give up" on SGD at this point and train $\hat{f}^{lin}$ with the current representation?

3.5 Application: Active Learning

One use case for eNTK is in active learning, where the model requests annotations for specific data points in an "active" manner. As discussed in Chapter 2 we can use kernel regression with the pNTK to approximate a re-trained model for the computation of "look-ahead" data acquisition functions in active learning. In Figure 3.15, we compare the performance of their scheme using pNTK to using the full eNTK, in terms of both active learning performance on the MNIST dataset (left axis), and wall-clock time needed to compute the acquisition function (right axis). Similarly to the previous chapter, we measure accuracy on the test set for a model which begins with 100 randomly trained points, then acquires 20 additional labelled points in each cycle. The acquisition performance with the pNTK matches that with the eNTK, but computation is much faster, taking about 15% as long in total on this problem with $O = 10$. Active learning performance is measured in accuracy on the

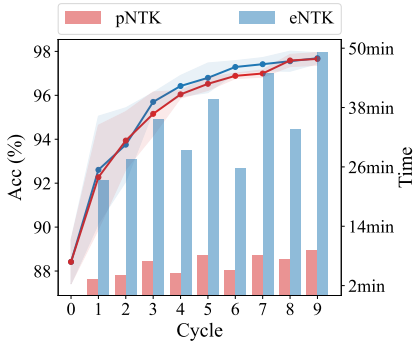


Figure 3.15: Comparison of pNTK with eNTK on a look-ahead active learning task. pNTK is much faster than eNTK without losing performance.

test set using a model trained on a labelled set.

Chapter 4

Discussion

Studying neural networks in the limit of infinite-width has provided us with numerous valuable theoretical and practical insights about the initialization of NNs, their training dynamics and properties of the learnt functions.

In this thesis, we use these insights to propose an algorithm to make look-ahead acquisition strategies feasible for active learning with fairly general neural network architectures. We prove that the outputs of wide enough neural networks can be efficiently approximated using a linearized network using empirical neural tangent kernels. We empirically show that this approximation is at least 100 times faster than naïve computation of look-ahead strategies, while even being more accurate. Armed with this approximation, we significantly outperform previous look-ahead strategies, and achieve equal or better performance compared to the state-of-the-art methods on four widely used benchmark datasets in active learning regime. We also propose a new sequential active learning algorithm that decouples querying and SGD training for the first time.

One limitation of the NTK approximation is that it is still slower than common "myopic" active learning methods such as Entropy and BADGE.

To alleviate this, we propose pNTK to approximate the eNTK. Our novel approximation has provable bounds, achieves empirically similar results as eNTK, and offers multiple orders of magnitude improvement in runtime speed and memory requirements over the direct eNTK. We evaluate our claims and the quality of the approximation under diverse settings, giving new insights into the behaviour of the

eNTK with trained representations. We help justify the correctness of recent approximation schemes, and hope that our rigorous results and thorough experimentation will help researchers develop a deeper understanding of the training dynamics of finite networks, and develop new practical applications of the NTK theory.

One major remaining question is to theoretically analyze what happens to the pNTK or eNTK during SGD training of the network. In particular, the fast convergence of $\hat{f}^{lin}$ and f^{lin} when training the network, as seen in Figures 3.7 and 3.8, runs counter to our expectations based on the approximation worsening in Frobenius norm (Figure 3.7), maximum eigenvalue (Figure 3.4), and condition number (Figure 3.6). This seems likely to be important to practical use of the pNTK.

Finally, even the pNTK is still rather expensive compared to running SGD on neural networks. It might make for a better starting point than the full eNTK for other speedup methods, like kernel approximation or advanced linear algebra solvers [*e.g.* 60].

Bibliography

[1] B. Adlam, J. Lee, L. Xiao, J. Pennington, and J. Snoek. Exploring the uncertainty properties of neural networks' implicit priors in the infinite-width limit. *arXiv preprint arXiv:2010.07355*, 2020. → page 2

[2] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. In *NeurIPS*, 2019. → pages 9, 12, 27, 28, 36

[3] S. Arora, S. S. Du, Z. Li, R. Salakhutdinov, R. Wang, and D. Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. *arXiv preprint arXiv:1910.01663*, 2019. → pages 1, 28

[4] D. Arpit and Y. Bengio. The benefits of over-parameterization at initialization in deep relu networks, 2019. → page 78

[5] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035, Philadelphia, PA, USA, 2007. → page 16

[6] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *ICLR*, 2020. → pages 6, 7, 15, 16, 18, 20

[7] G. Bachmann, T. Hofmann, and A. Lucchi. Generalization through the lens of leave-one-out error. In *ICLR*, 2022. → page 2

[8] E. Bıyık, K. Wang, N. Anari, and D. Sadigh. Batch active learning using determinantal point processes. *NeurIPS*, 2019. → pages 15, 16

[9] Z. Borsos, M. Tagliasacchi, and A. Krause. Semi-supervised batch active learning via bilevel optimization. In *ICASSP*. IEEE, 2021. → pages 17, 19

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang.

JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. → page 17

[11] X. Chen, C.-J. Hsieh, and B. Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. *arXiv preprint arXiv:2106.01548*, 2021. → pages 1, 2, 26

[12] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische mathematik*, 31(4):377–403, 1978. → page 33

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. → pages 3, 25

[14] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, pages 141–142, 2012. → page 17

[15] S. Fort, G. K. Dziugaite, M. Paul, S. Kharaghani, D. M. Roy, and S. Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020. → pages 2, 26, 36

[16] J.-Y. Franceschi, E. de Bézenac, I. Ayed, M. Chen, S. Lamprier, and P. Gallinari. A neural tangent kernel perspective of gans. *arXiv preprint arXiv:2106.05566*, 2021. → page 2

[17] A. Freytag, E. Rodner, and J. Denzler. Selecting influential examples: Active learning with expected model output changes. In *ECCV*, pages 562–577, 2014. → pages 6, 8

[18] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1050–1059, 2016. → page 16

[19] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *ICML*, pages 1183–1192, 2017. → page 16

[20] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *NeurIPS*, 2018. → page 14

[21] T. Hazan and T. Jaakkola. Steps toward deep kernel methods from infinite neural networks. *arXiv preprint arXiv:1508.05133*, 2015. → page 1

[22] B. He, B. Lakshminarayanan, and Y. W. Teh. Bayesian deep ensembles via the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:1010–1022, 2020. → page 2

[23] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://arxiv.org/abs/1502.01852. → pages 29, 31, 33, 63, 64

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. → page 17

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. → page 28

[26] D. Holzmüller, V. Zaverkin, J. Kästner, and I. Steinwart. A framework and benchmark for deep batch active learning for regression. *arXiv preprint arXiv:2203.09410*, 2022. → pages 2, 26

[27] N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. Bayesian active learning for classification and preference learning. 2011. → pages 5, 16

[28] L. Hui and M. Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. *ICLR*, 2021. → pages 10, 58

[29] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018. → pages 1, 3, 8, 9, 10, 49, 50, 51, 52, 53

[30] C. Käding, E. Rodner, A. Freytag, and J. Denzler. Active and continuous exploration with deep neural networks and expected model output changes. 2016. → pages 8, 20

[31] C. Käding, E. Rodner, A. Freytag, O. Mothes, B. Barz, J. Denzler, and C. Z. AG. Active learning for regression tasks with expected model output changes. In *BMVC*, page 103, 2018. → pages 8, 20

[32] A. Kirsch, J. Van Amersfoort, and Y. Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. pages 7026–7037, 2019. → pages 15, 16

[33] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. → pages 24, 28

[34] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. → page 17

[35] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017. → page 1

[36] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *NeurIPS*, 2019. → pages 1, 12, 26, 28, 33, 58, 61, 83

[37] J. Lee, S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33:15156–15172, 2020. → pages x, 28, 31, 36

[38] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine learning proceedings 1994*, pages 148–156. 1994. → page 16

[39] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR*, pages 3–12, 1994. → page 16

[40] Z. Li, R. Wang, D. Yu, S. S. Du, W. Hu, R. Salakhutdinov, and S. Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019. → page 12

[41] E. Malach, P. Kamath, E. Abbe, and N. Srebro. Quantifying the benefit of using differentiable learning over tangent kernels. In *ICML*, 2021. → page 12

[42] A. G. d. G. Matthews, M. Rowland, J. Hron, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. In *ICLR*, 2018. → page 8

[43] A. G. d. G. Matthews, M. Rowland, J. Hron, R. E. Turner, and Z. Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018. → page 1

[44] M. A. Mohamadi, W. Bae, and D. J. Sutherland. Making look-ahead active learning strategies feasible with neural tangent kernels. In *NeurIPS*, 2022. → page 2

[45] R. M. Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, 1996. → page 8

[46] R. M. Neal. *Priors for infinite networks*, pages 29–53. Springer, 1996. → page 1

[47] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPSW*, 2011. → page 17

[48] T. Nguyen, Z. Chen, and J. Lee. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020. → page 2

[49] T. Nguyen, R. Novak, L. Xiao, and J. Lee. Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, 34, 2021. → page 2

[50] V.-L. Nguyen, S. Destercke, and E. Hüllermeier. Epistemic uncertainty sampling. In *ICDS*, 2019. → page 8

[51] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148*, 2018. → page 1

[52] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *ICLR*, 2020. → page 17

[53] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *ICLR*, 2020. → page 9

[54] R. Novak, J. Sohl-Dickstein, and S. S. Schoenholz. Fast finite width neural tangent kernel. In *ICML*, 2022. → page 26

[55] D. Park, J. Sohl-Dickstein, Q. Le, and S. Smith. The effect of network width on stochastic gradient descent and generalization: an empirical study. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5042–5051. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/park19b.html. → page 28

[56] D. S. Park, J. Lee, D. Peng, Y. Cao, and J. Sohl-Dickstein. Towards NNGP-guided neural architecture search. *arXiv preprint arXiv:2011.06006*, 2020. → pages 2, 26

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. 2019. → page 17

[58] D. Roth and K. Small. Margin-based active learning for structured output spaces. In *European Conference on Machine Learning*, pages 413–424. Springer, 2006. → page 20

[59] N. Roy and A. McCallum. Toward optimal active learning through monte carlo estimation of error reduction. In *ICML*, 2001. → pages 6, 7, 8

[60] A. Rudi, L. Carratino, and L. Rosasco. FALKON: An optimal large scale kernel method. In *NeurIPS*, 2017. → page 40

[61] A. Rudi, L. Carratino, and L. Rosasco. Falkon: An optimal large scale kernel method. 2017. → page 14

[62] T. Scheffer, C. Decomain, and S. Wrobel. Active hidden markov models for information extraction. In *ISIDA*, 2001. → page 16

[63] O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. *ICLR*, 2018. → page 16

[64] B. Settles. Active learning literature survey. 2009. → pages 7, 16

[65] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. pages 1289–1296, 2007. → page 6

[66] L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, 2019. → page 57

[67] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. → page 2

[68] T. Tran, T.-T. Do, I. Reid, and G. Carneiro. Bayesian generative active deep learning. In *ICML*, pages 6295–6304, 2019. → page 16

[69] M. J. Wainwright. *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019. doi:10.1017/9781108627771. → pages 62, 63

[70] D. Wang and Y. Shang. A new active labeling method for deep learning. In *IJCNN*, pages 112–119, 2014. → pages 5, 16

[71] H. Wang, W. Huang, A. Margenot, H. Tong, and J. He. Deep active learning by leveraging training dynamics. *arXiv preprint arXiv:2110.08611*, 2021. → pages 2, 26, 27

[72] A. Wei, W. Hu, and J. Steinhardt. More than a toy: Random matrix models predict how real-world neural representations generalize. In *ICML*, 2022. → pages 2, 13, 26, 27, 31, 33

[73] C. Williams. Computing with infinite networks. *Advances in neural information processing systems*, 9, 1996. → page 1

[74] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pages 5393–5402. PMLR, 2018. → pages 1, 31

[75] L. Xiao, J. Pennington, and S. Schoenholz. Disentangling trainability and generalization in deep neural networks. In *International Conference on Machine Learning*, pages 10462–10472. PMLR, 2020. → pages x, 1, 31

[76] G. Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *Advances in Neural Information Processing Systems*, 32, 2019. → page 1

[77] G. Yang. Tensor programs ii: Neural tangent kernel for any architecture, 2020. → pages 1, 9, 12, 27, 30, 61, 82

[78] G. Yang and E. Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics, 2021. → pages 1, 9, 30, 49

[79] D. Yoo and I. S. Kweon. Learning loss for active learning. In *CVPR*, pages 93–102, 2019. → pages 16, 18, 20, 57

[80] S. Zagoruyko and N. Komodakis. Wide residual networks. *BMVC*, 2016. → pages 17, 28

[81] Y. Zhou, Z. Wang, J. Xian, C. Chen, and J. Xu. Meta-learning with neural tangent kernels. *NeurIPS*, 2021. → page 58

[82] Y. Zhou, Z. Wang, J. Xian, C. Chen, and J. Xu. Meta-learning with neural tangent kernels. *arXiv preprint arXiv:2102.03909*, 2021. → page 2

[83] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICMLW*, volume 3, 2003. → page 8

Appendix A

Supplementary Material

A.1 Proof of Theorem 2.2.1

In this section, we show that in the infinite width limit, warm starting a neural network will result in the same predictions as the cold started variant when trained using gradient descent for $t = \infty$. We give the proof for a feed-forward neural network, but as shown by Yang and Littwin [78], it is trivial to show that the same proof structure can be used for any other architecture.

Background We mostly use the same notation as Jacot et al. [29]. We focus on fully connected feed-forward neural networks with $L + 1$ hidden layers numbered from 0 (input) to L (output), where each layer has $n_0, \ldots, n_L$ neurons. We assume that its nonlinearity is a Lipschitz, twice differentiable function $\sigma : \mathbb{R} \to \mathbb{R}$. Such a network has $P = \sum_{l=0}^{L-1} (n_l + 1)n_{l+1}$ parameters: A weight matrix $W^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ and a bias vector $b^{(l)} \in \mathbb{R}^{n_{l+1}}$ per layer. This network is defined as f_θ where $\theta = \cup_{l=0}^{L} \theta^l$ and $\theta^l = vec(W^{(l)}, b^{(l)})$. The function f_θ is characterized according to

the definition:

$$\begin{aligned}
\alpha^{(0)}(x;\theta) &= x \\
\tilde{\alpha}^{(l+1)}(x;\theta) &= \frac{1}{\sqrt{n_l}} W^{(l}\alpha^{(l)}(x;\theta) + \beta b^{(l)} \\
\alpha^{(l)}(x;\theta) &= \sigma(\tilde{\alpha}^{(l+1)}(x;\theta)) \\
f_\theta(x) &\coloneqq \tilde{\alpha}^{(L)}(x;\theta)
\end{aligned} \tag{A.1}$$

We define the *realization function* of the network as $f_{(L)} : \mathbb{R}^P \to \mathcal{F}$, the function mapping parameters $\theta : \mathbb{R}^P$ to functions $f_\theta \in \mathcal{F}$ where $\mathcal{F}$ is the space of all neural networks of this architecture.

Assuming that the training set is defined as $\mathcal{D} \subseteq \mathbb{R}^{n_0} \times \mathbb{R}^{n_L}$, we define p^{in} to be the fixed empirical distribution of the finite input dataset $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\} : \frac{1}{N}\sum_{i=0}^{N} \delta_{x_i}$. Thus, $\mathcal{F}$ is defined as all the functions $\{f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}\}$. On this space, we consider the seminorm $||\cdot||_{p^{in}}$, defined as

$$\langle f, g \rangle_{p^{in}} = \mathbb{E}_{x \sim p^{in}}[f(x)^T g(x)]. \tag{A.2}$$

The dual space of $\mathcal{F}$ with respect to p^{in} can be defined as $\mathcal{F}^*$. $\mathcal{F}^*$ is the set of all linear forms $\mu : \mathcal{F} \to \mathbb{R}$. We can define each element of this set as $\mu = \langle d, \cdot \rangle_{p^{in}}$ for some $d \in \mathcal{F}$.

In the process of optimizing the parameters of a neural network f, we define the cost functional as: $C(f) = \frac{1}{N}\sum_{i=1}^{N} (f(x_i) - y_i)^2$. As we assumed that p^{in} is fixed, the value of $C(f)$ only depends on the values of $f \in \mathcal{F}$ at the datapoints $(x, y) \in p^{in}$. Thus, the functional derivative of the cost functional can be viewed as a member of the dual space of $\mathcal{F}$, namely $\mathcal{F}^*$. We note by $d|_{f_0} \in \mathcal{F}$ the corresponding dual element of partial derivative of the cost functional with respect to the function f at f_0, namely: $\partial_f^{in} C|_{f_0}$. Thus, $\partial_f^{in} C|_{f_0} = \langle d|_{f_0}, \cdot \rangle_{p^{in}}$.

According to Jacot et al. [29], *Kernel Gradient* $\nabla_{\mathcal{K}} C|_{f_0} \in \mathcal{F}$ is defined as $\Phi_{\mathcal{K}}\left(\partial_f^{in} C|_{f_0}\right)$ where Φ is a map from $\mathcal{F}^*$ to $\mathcal{F}$ defined as: $[\Phi_{\mathcal{K}}(\mu)]_{i,\cdot}(x) = \langle d, \mathcal{K}_{i,\cdot}(x, \cdot) \rangle_{p^{in}}$ where $\mu = \langle d, \cdot \rangle_{p^{in}}$. Here, $\mathcal{K}$ refers to a multi-dimensional kernel which is defined as a function $\mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$ such that $\mathcal{K}(x, x') = \mathcal{K}(x', x)^\top$. They showed that when trained using gradient descent on $C \circ F_{(L)}$, the neural network function's evolution in time can be captured using *kernel gradient de-*

scent with the corresponding Neural Tangent Kernel (NTK): $\partial_t f_{\theta(t)} = -\nabla_{\mathcal{K}} C|_{f_{\theta(t)}}$ where $\mathcal{K}$ is the corresponding Neural Tangent Kernel of $f_{\theta(t)}$. For simplicity, from here onwards, we drop the θ index from a function $f_{\theta_{(t)}}$ and show it by f_t. Since one can define a map between the time t and function $f_{\theta_{(t)}}$ when the training setting is fixed, this doesn't create any confusion. Thus, the cost functional evolves as

$$\partial_t C|_{f_t} = -\langle d|_{f_t}, \nabla_{\mathcal{K}} C|_{f_t} \rangle_{p^{in}} \tag{A.3}$$

Here, $-d|_{f_t} \in \mathcal{F}$ defines the training direction of function $f_{(t)}$ in the function space $\mathcal{F}$ while being trained using gradient descent (flow). As mentioned earlier, when the we train using $C \circ F_{(L)}$ loss, this training direction is the dual of $\partial_f^{in} C|_{f_t}$.

Setting A. We have C overlapping datasets, $S_1, S_2, \ldots, S_C$ such that $\forall i > j; S_j \subset S_i$ and $\forall i \in [1, C]; S_i = \{(x_1, y_1), \ldots, (x_{m_i}, y_{M_i})\}$ where M_i is the size of i'th dataset. We initialize the network f with parameters θ_0 such that θ_0 satisfies the initialization criteria mentioned in Jacot et al. [29] (namely, LeCun initialization). We define f_0 as f with parameters θ_0. f has $L+1$ layers such that in the limit $n_0, n_1, \ldots, n_L \to \infty$ sequentially. We train f sequentially on all the sets using gradient descent with $C \circ F_L$ loss for infinite time:

$$f_0 \xrightarrow[t=\infty]{S_1} f_{S_1} \xrightarrow[t=\infty]{S_2} f_{S_{1,2}} \to \ldots \xrightarrow[t=\infty]{S_C} f_{S_{1,2,\ldots,C}}. \tag{A.4}$$

In the following, we first note that starting from the initialization and training the network according to (A.4), the sum of integrals of the training directions remains stochastically bounded (Remark A.1.1). Thus, based on Theorem 2 in [29], the corresponding Neural Tangent Kernel of f remains asymptotically constant (and according to their Theorem 1, it converges in probability to the limiting Neural Tangent Kernel at initialization). Next, in Theorem A.1.2, we prove that under this setting, the resulting function of sequential warm-start training on S_1 to S_C, is the same as the function that we get when doing simple gradient descent on only the last dataset S_C when starting from initialization (Also known as cold-start). In other words, $\forall x \in \mathbb{R}^{n_0}; f_{S_{1,\ldots,C}}(x) = f_{S_C}(x)$ where f_{S_C} is derived using $f_0 \xrightarrow[t=\infty]{S_C} f_{S_C}$.

Remark A.1.1. Under Setting A, assuming that we have a function $\mathcal{T} : \mathbb{R} \to \mathbb{R}$ which gets past time since starting gradient descent as the input and outputs the

index i of the dataset S_i currently being trained in the sequential training process, $\lim_{T\to\infty} \int_{t=0}^{T} ||d|_{f_t}^{S_{\mathcal{T}(t)}}||_{p^{in}} dt$ is stochastically bounded.

Proof. We start by deriving the Gâteaux derivative of the cost functional $C(f) = \frac{1}{N}\sum_{i=1}^{N} ||f(x) - y||_2^2$:

$$\partial_f^{in} C|_{f_t}(f) = \frac{2}{N}\sum_{i=1}^{N} f(x_i)^\top \left(f_t(x_i) - y_i\right) \tag{A.5}$$

This shows the amount of change in $C(f_t)$ when f_t is moved towards f by an infinitesimal t. As mentioned before, this functional derivative only depends on the values of f on the datapoint in p^{in}. Thus, it's in the dual space of $\mathcal{F}$, noted by $\mathcal{F}^*$ and we can write it as $\langle d|_{f_t}, f\rangle_{p^{in}}$. We're interested in deriving the closed form of $d|f_t$ in this inner product.

$$\langle d|_{f_t}, f\rangle_{p^{in}} = \frac{1}{N}\sum_{i=1}^{N} d|_{f_t}(x_i)^\top f(x_i) = \frac{2}{N}\sum_{i=1}^{N} f(x_i)^\top \left(f_t(x_i) - y_i\right) \tag{A.6}$$

This implies

$$\forall (x_i, y_i) \in p^{in};\ d|_{f_t}(x_i) = \frac{1}{2}\left(f_t(x_i) - y_i\right). \tag{A.7}$$

If we define $f^*(x)$ as the function that maps each datapoint to its label on p^{in} and is arbitrary elsewhere, $d|_{f_t}(x) = \frac{1}{2}\left(f_t(x) - f^*(x)\right)$ on p^{in}. Accordingly, when we train using only a portion of the data $S_i \subseteq p^{in}$ such that $\mathcal{X}_{S_i} = \{x : (x, y) \in S_i\}$, but the cost functional still operates on whole of p^{in}, we have that:

$$d|_{f_t}^{S_i}(x) = \frac{I(x \in \mathcal{X}_{S_i})}{2}\left(f_t(x) - f_{S_i}^*(x)\right) \tag{A.8}$$

where $f_{S_i}^*$ is defined on S_i as f^* is on p^{in}. To analyze $\int_{t=0}^{T} ||d|_{f_t}^{S_i}||_{p^{in}} dt$, we first derive $||d|_{f_t}^{S_i}||_{p^{in}}$:

$$||d|_{f_t}^{S_i}||_{p^{in}} = \mathbb{E}_{x\sim p^{in}}\left[\left(d|_{f_t}^{S_i}(x)\right)^2\right] = \frac{1}{4N}\sum_{x_j \in \mathcal{X}_{S_i}} ||f_t(x_j) - f_{S_i}^*(x_j)||_2^2 \tag{A.9}$$

As Jacot et al. [29] (Section 5) mentioned, as t grows, this norm is strictly

decreasing and thus the integral is bounded. Moreover, as we're following the direction of gradient flow in the functional space, $\exists t \geq 0\ f_t(x_j) = f_t^*(x_j)$ for all datapoints x_j in S_i. Thus, $\lim_{T\to\infty} \int_{t=0}^{T} ||d|_{f_t}^{S_i}||_{p^{in}} dt$ is also stochastically bounded. We can apply the same structure for the case where we start from f_{S_i} and perform gradient flow towards $f_{S_{i+1}}$, showing that the integral in the infinite time limit is also stochastically bounded. Thus, it's straightforward to show that using induction, when training the neural network sequentially on $S_1, S_2, \ldots, S_C$ for sequentially infinite time, the integrals of training directions remain stochastically bounded. □

Theorem A.1.2. *Under Setting A, the following equality holds:*

$$\forall x \in \mathbb{R}^{n_0} \quad f_{S_{1,\ldots,C}}(x) = f_{S_C}(x) \tag{A.10}$$

Proof. As Remark A.1.1 showed, when training a neural network on multiple overlapping datasets under Setting A, the training direction remains stochastically bounded. Thus, as the width of the layers of the network tend to infinite sequentially, we can use Jacot et al. [29]'s Theorem 2 to show that the NTK $\mathcal{K}$ also remains constant during training in this setting. Accordingly, when trained only on a dataset S_i, the neural network function f evolves as

$$\begin{aligned}\partial_t f_t(x) = -\nabla_{\mathcal{K}} C|_{f_t}^{S_i}(x) = -\Phi_{\mathcal{K}}\left(\partial_f^{S_i} C|_{f_t}\right) = -\frac{1}{2N}\sum_{x_j \in S_i} \mathcal{K}(x, x_j)(f_t(x_j) - f_{S_i}^*(x_j)) \\ = \frac{1}{2N}\underbrace{\sum_{x_j \in S_i} \mathcal{K}(x, x_j) f_t(x_j)}_{\text{function of } f_t(x), x \text{ at } t} - \frac{1}{2N}\underbrace{\sum_{x_j \in S_i} \mathcal{K}(x, x_j) f^*(x_j)}_{\text{function of } x \text{ at } t}\end{aligned} \tag{A.11}$$

where $\mathcal{K}$ is the corresponding NTK of f_t, which remains constant during training. If we look closely, we witenss that this is a system of ODEs, whose solution for the finite dataset S_i from initialization f_0 can be written as

$$f_t(\mathcal{X}_{S_i}) = f_{S_i}^*(\mathcal{X}) + e^{-t\mathcal{K}_{S_i}}\left(f_0(\mathcal{X}_{S_i}) - f_{S_i}^*(\mathcal{X}_{S_i})\right). \tag{A.12}$$

where $\mathcal{K}_{S_i} = \mathcal{K}(\mathcal{X}_{S_i}, \mathcal{X}_{S_i})$. As $\Phi_{\mathcal{K}}(\cdot)$ helps us generalize the values of kernel

gradient (and consecutively f_t, as it evolves according to kernel gradient) to values x outside the dataset S_i (and also p^{in}). Now that we have derived the closed form solution of the outputs of f_t on X_{S_i}, we can also derive the outputs of f_t on any arbitrary x by taking the integral of (A.11) and replacing f_t according to (A.12):

$$\begin{aligned} f_t(x) &= \frac{1}{2N} \sum_{x_j \in S_i} \mathcal{K}(x, x_j) \left[\int_{t'=0}^{t} \left(f_{t',z}(x_j) - f^*_{S_i,z}(x_j) \right) dt' \right]_z \\ &= f_0(x) + \mathcal{K}(x, \mathcal{X}_{S_i}) \mathcal{K}_{S_i}^{-1} \left(I - e^{-t\mathcal{K}_{S_i}} \right) \left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i}) \right) \end{aligned} \tag{A.13}$$

where $f_{.,z}$ shows the zth index of output of f and $[\cdot]_z$ shows the stacked vector for different values of z, such that z is an integer in $[1 - n_L]$. Starting from initialization, we can use this to characterize the outputs of the trained neural network on S_i using gradient flow for time t. We're interested in starting from f_{S_i} and training on S_{i+1} for infinite time, according to the same loss function defined on p^{in}. For the sake of having more clear notations, we denote this function as $f_{S_i \to S_{i+1}}$ (defiend as $f_{S_i,S_{i+1}}$ in the statement that we are proving).

$$\begin{aligned} f_{S_i \to S_{i+1}}(x) &= f_{S_i}(x) + \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \mathcal{K}_{S_{i+1}}^{-1} \left(f^*_{S_{i+1}}(\mathcal{X}_{S_i}) - f_{S_i}(\mathcal{X}_{S_{i+1}}) \right) \\ &= \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \mathcal{K}_{S_{i+1}}^{-1} f^*_{S_{i+1}}(\mathcal{X}_{S_{i+1}}) \\ &\quad + \underbrace{\left(\mathcal{K}(x, \mathcal{X}_{S_1}) - \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \mathcal{K}_{S_{i+1}}^{-1} \mathcal{K}(\mathcal{X}_{S_{i+1}}, \mathcal{X}_{S_i}) \right)}_{A} \mathcal{K}_{S_i}^{-1} \left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i}) \right) \\ &\quad + f_0(x) - \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \mathcal{K}_{S_{i+1}}^{-1} f_0(\mathcal{X}_{S_{i+1}}) \\ &= f_{S_{i+1}}(x) + A \mathcal{K}_{S_i}^{-1} \left(f^*_{S_i}(\mathcal{X}_{S_i}) - f_0(\mathcal{X}_{S_i}) \right) \end{aligned} \tag{A.14}$$

If we look closely, A can be written as:

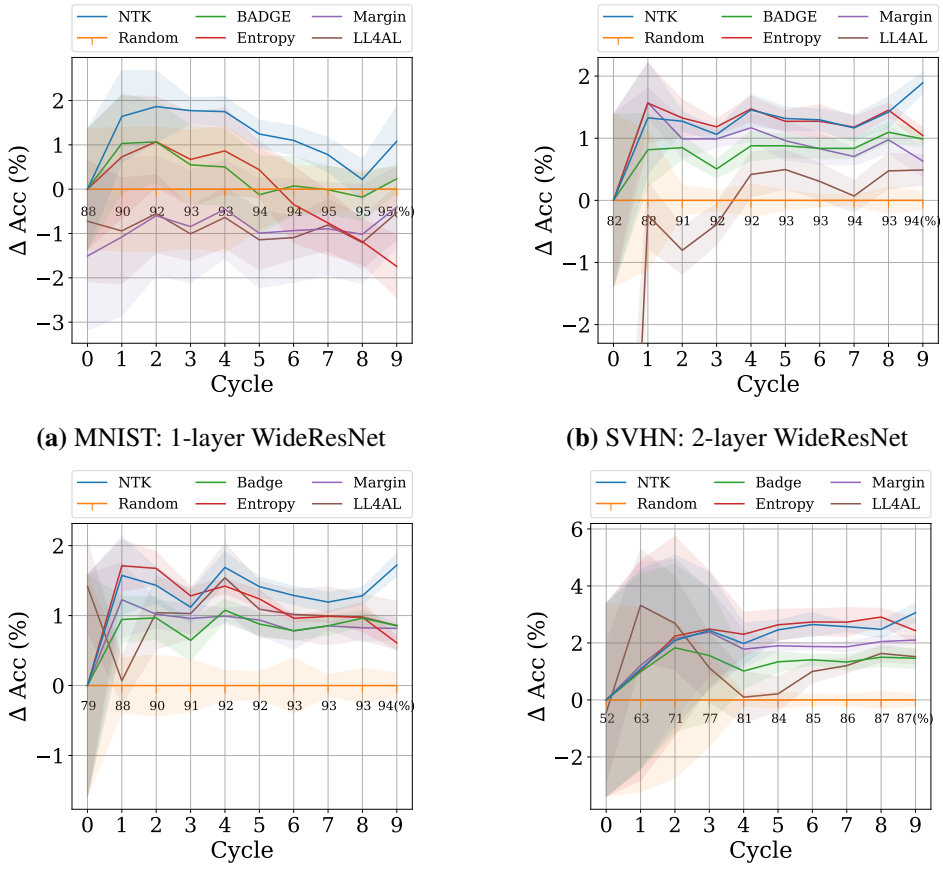
$$
\begin{aligned}
A &= \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i} \end{bmatrix} - \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \mathcal{K}_{S_{i+1}}^{-1} \mathcal{K}(\mathcal{X}_{S_{i+1}}, \mathcal{X}_{S_i}) \\
&= \mathcal{K}(x, \mathcal{X}_{S_{i+}}) \left(\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i} \end{bmatrix} - \mathcal{K}_{S_{i+1}}^{-1} \mathcal{K}(\mathcal{X}_{S_{i+1}}, \mathcal{X}_{S_i}) \right) \\
&= \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \left(\begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i} \end{bmatrix} - \mathcal{K}_{S_{i+1}}^{-1} \mathcal{K}(\mathcal{X}_{S_{i+1}}, \mathcal{X}_{S_{i+1}}) \begin{bmatrix} I_{M_i} \\ O_{(M_{i+1}-M_i)\times M_i} \end{bmatrix} \right) \\
&= \mathcal{K}(x, \mathcal{X}_{S_{i+1}}) \times 0 = 0
\end{aligned}
\tag{A.15}
$$

where M_i denotes the number of datapoints in S_i. Note that (A.15) doesn't depend on the indices of S_i and S_{i+1}. Rather, it just relies on $S_i \subseteq S_{i+1}$, which is the case in Setting A for any consecutive batches. Thus, for any arbitrary x, $f_{S_i \to S_{i+1}}(x) = f_{S_{i+1}}(x)$. Using basic forward induction, starting from f_{S_i}, we can prove that $f_{S_1 \to S_2 \to \cdots \to S_C}(x) = f_{S_C}(x)$ for any arbitrary x. The proof is complete. □

A.2 Additional Comparison with State-of-the-art Methods

In Figure 2.2, we demonstrate that our proposed method is equal or better compared to other state-of-the-art methods on various benchmark datasets in active learning, with different architectures. In this section, we provide additional experiment results as shown in Figure 2.2. Among all the combinations of the datasets we use (MNIST, SVHN, CIFAR10 and CIFAR100), and architectures (1-layer WideResNet, 2-layer WideResNet, and ResNet18), we do not provide results on MNIST with 2-layer WideResNet and ResNet18 since 1-layer WideResNet is large enough for MNIST. Also, we do not provide the results with 1-layer WideResNet on CIFAR10 since 1-layer WideResNet is too shallow for complicate data like CIFAR10 and 100. Similarly, according to our experiments, WideResNet with 1 or 2-layers are too shallow for CIFAR 100, unlike ResNet18.

As a result, in Figure A.1, we provide the experiment results of all the other combinations. We visualize in the same way as we do for Figure 2.2 where we plot



(a) MNIST: 1-layer WideResNet

(b) SVHN: 2-layer WideResNet

(c) SVHN: ResNet18

(d) CIFAR10: ResNet18

Figure A.1: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows difference from random acquisition, whose accuracy is shown in text.

the difference between each method and random acquisition function at each cycle. Figure A.1 shows that the proposed NTK approximation is equal or better than the other state-of-the-art methods; especially, NTK is better than any other methods for Figure A.1a and Figure A.1c, and comparable to entropy acquisition function for Figure A.1b and Figure A.1d.

A.3 Instability of LL4AL

In Figure 2.2, we do not include LL4AL [79] performance on CIFAR10 or CIFAR100; its performance is too poor to show on the same plots. To validate this was not due to a simple poor hyperparameter choice, we show the performance of LL4AL on CIFAR100 with varying learning rates in Figure A.2a. We run each experiment 3 times with the maximum learning rates of $\{0.3, 0.1, 0.03, 0.01, 0.003, 0.001\}$ for the 1cycle learning rate policy [66], which we use for all the other methods. When the maximum learning rate is 0.3, the gradient blows up and the model does not learn anything.

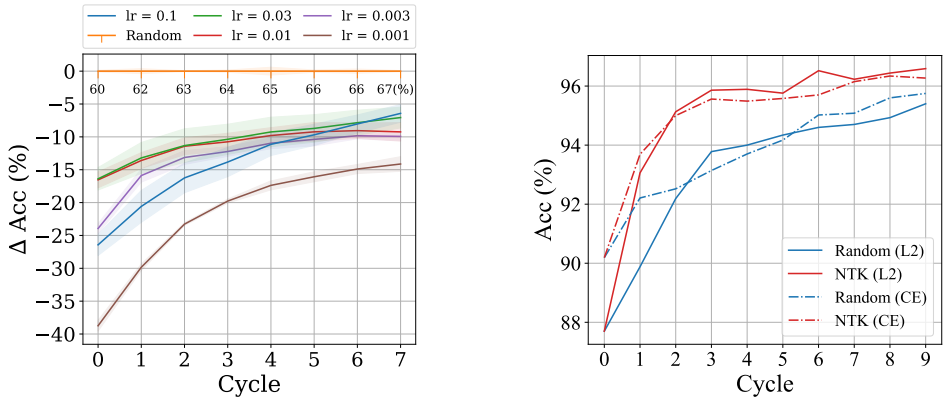
Figure A.2a shows none of the learning rates are comparable with random acquisitions (trained with the maximum learning rate of 1.0), although it has been reported that LL4AL performs better than the random acquisition function. We conjecture this is because, if we train with random acquisitions using a learning rate that "works" for LL4AL, LL4AL performs better. On the other hand, when the learning rate is tuned to maximize the performance of each method, random acquisition can perform better.

We added the figures Figure A.3 based on the results shared in this GitHub repository which is a direct implementation of the LL4AL method. We further elaborate on each plot below:

CIFAR10, ResNet18 (Figure A.3a): We observe that although when using the learning rate of 0.1 LL4AL's performance is 4% better than random (91.3% vs 87.3%), when the learning rate is tuned in favor of random (0.3 LR instead of 0.1) and use that for LL4AL too, the difference shrinks down to 1.2% (90.0% vs 87.8%). Moreover, using larger learning rates like 1.0 would result in LL4AL to diverge while random still gets descent results.

SVHN, ResNet18 (Figure A.3b): Again, the difference is 3.3% (93.8% vs 90.5%) when LR is tuned in favor of LL4AL, but drops to 1.1% (91.9% vs 90.8%) when using a LR of 1.0. This time however, when LR is fully tuned in favor of random (2.0), LL4AL diverges.

CIFAR10, 1-block WideResNet (Figure A.3c): The 2.6% difference when using 0.1 LR (82.2% vs 79.6%) drops to -1.8% (78.8% vs 80.6%) when LR is tuned



(a) Varying LL4AL's learning rate (CIFAR100).

(b) L_2 and cross-entropy losses on MNIST.

Figure A.2: Additional comparisons.

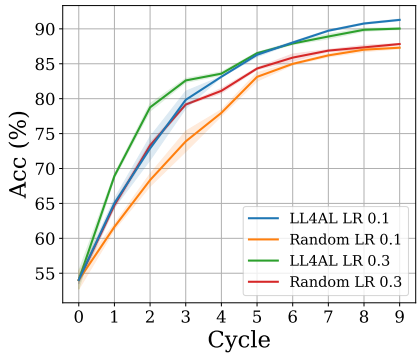
in favor of random (0.3). We remind that as we're using one block layer here, the LL4AL's performance drop is much more noticeable to the point that it performs worse than random when LR is tuned in favor of random!

SVHN, 1-block WideResNet (Figure A.3d): The 2% difference when using 0.1 LR (95% vs 93%) drops to -1.4% (92.4% vs 93.8%) when LR is tuned in favour of random (0.5).

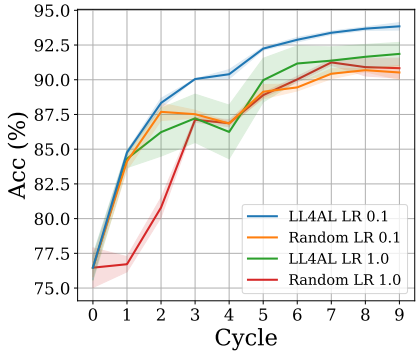
A.4 Comparison of L_2 and Cross-entropy Loss

We use L_2 loss instead of cross-entropy loss to train a classifier. As mentioned in the main paper, Hui and Belkin [28] empirically show L_2 loss is just as effective as cross-entropy loss for various classification tasks in computer vision and natural language processing. Because of this, previous works that exploit a linearized network using empirical NTKs also use L_2 loss as a replacement for cross-entropy [36, 81].

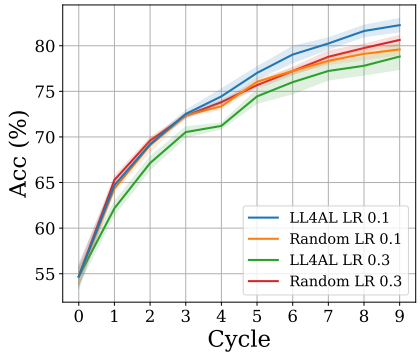
To further demonstrate that L_2 loss is indeed as effective as cross-entropy in active learning, we provide experimental results with L_2 and cross-entropy loss (CE) for both random and the proposed NTK acquisition functions in Figure A.2b. Although cross-entropy starts with a higher accuracy, L_2 quickly catches up. Overall, the difference between L_2 and cross-entropy is not significant for either random or



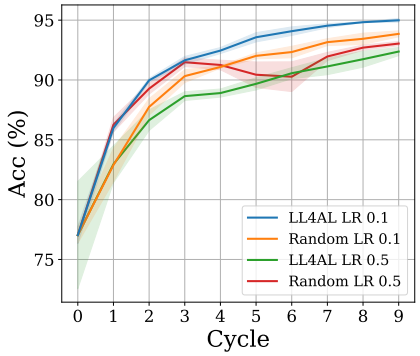
(a) Varying LL4AL's learning rate (CIFAR10 - ResNet18).



(b) Varying LL4AL's learning rate (SVHN - ResNet18).



(c) Varying LL4AL's learning rate (Cifar10 - 1-layer WideResNet).



(d) Varying LL4AL's learning rate (SVHN - 1-layer WideResNet).

Figure A.3: Additional experiments on instability/poor performance of LL4AL on large learning rates.

NTK acquisition functions.

A.5 Conventional Format for Comparison of the Proposed Method vs. SOTA

We have provided the conventional accuracy-based plots for the experimental results of comparing our proposed method with other state-of-the-art methods. We think the ΔAcc based plots are better at letting readers observe the differences between performances of each method on each task. However, for the sake of completeness

and in case one finds the plots in the main paper confusing, we have provided the conventional plots in Figure A.4.

A.6 Details of Experimental Setup

In this section, we present the details on the experimental setup used for the plots depicted in the main body of the paper. As mentioned, the exact width for FCNs have been reported. For WideResNet-16-k we use two block layers, and the initial convolution in the network has a width of $16\mathrm{WF}$ where WF is the reported WF. For instance, $\mathrm{WF} = 16$ means that the first block layer has a width of 256 and the second block layer has a width of 512. For ResNet18, we also used the same approach, multiplying WF by 16. Thus, when $\mathrm{WF} = 4$, the constructed network will have the exact architecture as the classical ResNet18 architecture reported. A WF of 16 means a ResNet18 with each layer being 4 times wider than the original width.

When training the neural networks using SGD, a constant batch size of 128 was used across all different networks and different dataset sizes used for training. The learning rate for all networks was also fixed to 0.1. However, not all networks were trainable with this fixed learning rate, as the gradients would sometimes blow up and give `NaN` training loss, typically for the largest width of each mentioned architecture. In those cases, we decreased the learning rate to 0.01 to train the networks.

Note that to be consistent with the literature on NTKs, techniques like data augmentation have been turned off, but a weight decay of 0.0001 along with a momentum of 0.9 for SGD is used. Data augmentation here plays an important role in the attained test accuracies of the fully trained networks.

A.7 Further Results on the Approximation Quality

In this section we'll lay out the proofs of the theorems provided in the main text, mainly Theorems 3.3.1, 3.3.4 and 3.3.5. Towards this, we first define some notation and show a simple recursive formula for computing the tangent kernel that we take advantage of to prove the theorems. Consider a NN $f : \mathbb{R}^D \to \mathbb{R}^O$. We assume the final read-out layer of the NN f is a dense layer with width w. Assuming

the NN f has L layers, we define θ_l to be the corresponding parameters of layer $l \in \{1, 2, \ldots, L\}$. Furthermore, let's define $g : \mathbb{R}^D \to \mathbb{R}^w$ as the output of the immediate last layer of the NN f, such that $f(x) = \theta_L g(x)$ for some $\theta_L \in \mathbb{R}^{O \times w}$.

As shown by Lee et al. [36], Yang [77], the NTK can be reformulated as the layer-wise sum of gradients (when the parameters of each layer θ_l are assumed to be vectorized) of the output with respect to θ_l. Accordingly, we denote eNTK of a NN f as

$$\Theta_f(x_1, x_2) = \sum_{l=1}^{L} \nabla_{\theta_l} f(x_1) \nabla_{\theta_l} f(x_2)^\top. \tag{A.16}$$

Now, noting that as the final layer of f is a dense layer, we can use the chain rule to write $\nabla_{\theta_l} f(x)$ as $\frac{\partial f}{\partial g(x)} \frac{\partial g(x)}{\partial \theta_l}$ where $\frac{\partial f(x)}{\partial g(x)} = \theta_L$. Thus, we can rewrite (A.16) as

$$\begin{aligned}
\Theta_f(x_1, x_2) &= \sum_{l=1}^{L-1} \theta_L \nabla_{\theta_l} g(x_1) \, \nabla_{\theta_l} g(x_2)^\top {\theta_L}^\top + \nabla_{\theta_L} f(x_1) \nabla_{\theta_L} f(x_2)^\top \\
&= \theta_L \left(\sum_{l=1}^{L} \nabla_{\theta_l} g(x_1) \, \nabla_{\theta_l} g(x_2)^\top \right) {\theta_L}^\top + g(x_1)^\top g(x_2) \, I_O \\
&= \theta_L \, \Theta_g(x_1, x_2) \, {\theta_L}^\top + g(x_1)^\top g(x_2) \, I_O.
\end{aligned} \tag{A.17}$$

Applying Equation (A.17), we can already see that the pNTK of a network f simply calculates a weighted summation of all elements of eNTK into a scalar, since it can be seen as adding a new final dense layer to the network f with the fixed weight vector $\frac{1}{\sqrt{O}} \mathbf{1}_O$ where $\mathbf{1}_O$ is the O-dimensional vector consisting of all 1s. Note that, however, this $\mathbf{1}_O$ vector is not trainable in this context and is a fixed vector.

Before moving on with the approximation proofs, we would like to mention that the proofs in this section rely heavily on concentration inequalities of *sub-exponential* random variables. Thus, we start by providing some background about sub-exponential random variables and the related concentration inequalities that we will use later on.

A.7.1 Background on Sub-Exponential Random Variables

A real-valued random variable X with mean μ is called *sub-exponential* [69] if there are non-negative parameters (ν, α) such that

$$\mathbb{E}[e^{\lambda(X-\mu)}] \leq e^{\frac{\nu^2\lambda^2}{2}} \quad \text{for all } |\lambda| < \frac{1}{\alpha}.$$

We use $X \sim SE(\nu, \alpha)$ to denote that X is a sub-exponential random variable with parameters (ν, α), but note that this is not a particular distribution.

SA famous sub-exponential random variable is the product of two standard normal distributions, $z_i \sim \mathcal{N}(0, 1)$, such that the two factors are independent ($X_1 = |z_1||z_2| \sim SE(\nu_p, \alpha_p)$ with mean $2/\pi$) or the same ($X_2 = z^2 \sim SE(2, 4)$ with mean 1). We now present a few lemmas regarding sub-exponential random variables that will come in handy in the later subsections of the appendix.

Lemma A.7.1. *If a random variable X is sub-exponential with parameters (ν, α), then the random variable sX where $s \in \mathbb{R}^+$ is also sub-exponential with parameters $(s\nu, s\alpha)$.*

Proof. Consider $X \sim SE(\nu, \alpha)$ and $X' = sX$ with $\mathbb{E}[X'] = s\,\mathbb{E}[X]$, then according to the definition of a sub-exponential random variable

$$\begin{aligned}
&\mathbb{E}\left[\exp\left(\lambda(X-\mu)\right)\right] \leq \exp(\frac{\nu^2\lambda^2}{2}) \quad \text{for all } |\lambda| < \frac{1}{\alpha} \\
&\implies \mathbb{E}\left[\exp\left(\frac{\lambda}{s}(sX - s\mu)\right)\right] \leq \exp(\frac{\nu^2 s^2 \frac{\lambda^2}{s^2} 2}{2}) \quad \text{for all } |\frac{\lambda}{s}| < \frac{1}{s\alpha} \\
&\xrightarrow{\lambda' = \frac{\lambda}{s}} \mathbb{E}\left[\exp\left(\lambda'(X' - \mu')\right)\right] \leq \exp(\frac{\nu^2 s^2 \lambda'^2}{2}) \quad \text{for all } |\lambda'| < \frac{1}{s\alpha}
\end{aligned} \tag{A.18}$$

Defining $\alpha' = s\alpha$ and $\nu' = s\nu$ we recover that $X' \sim SE(s\nu, s\alpha)$. □

Proposition A.7.2. *If the random variables X_i for $i \in [1 - N]$ for $N \in \mathbb{N}^+$ are all sub-exponential with parameters (ν_i, α_i) and independent, then $\sum_{i=1}^N X_i \in SE(\sqrt{\sum_{i=1}^N \nu_i^2}, \max_i \alpha_i)$, and $\frac{1}{N}\sum_{i=1}^N X_i \sim SE\left(\frac{1}{\sqrt{N}}\sqrt{\frac{1}{N}\sum_{i=1}^N \nu_i^2}, \frac{1}{N}\max_i \alpha_i\right)$.*

Proof. This is a simplification of the discussion prior to equation 2.18 in Wainwright [69]. □

Proposition A.7.3. *For a random variable $X \sim SE(\nu, \alpha)$, the following concentration inequality holds:*

$$\Pr\left(|X - \mu| \geq t\right) \leq 2\exp\left(-\min\left(\frac{t^2}{2\nu^2}, \frac{t}{2\alpha}\right)\right).$$

Proof. The proof directly follows from applying a scalar multiplication to the result derived in Equation 2.18 in [69]. □

Corollary A.7.4. *For a random variable $X \sim SE(\nu, \alpha)$, the following inequality holds with probability at least $1 - \delta$:*

$$|X - \mu| < \max\left(\nu\sqrt{2\log\frac{2}{\delta}}, 2\alpha\log\frac{2}{\delta}\right).$$

A.7.2 pNTK Relatively Converges To eNTK As Width Grows

Let's denote a neural network with L dense hidden layers whose width is n as:

$$\begin{aligned} f^0(x) &= x \\ f^{l+1}(x) &= \phi(W^{(l+1)} f^l(x)) \\ f(x) &= f^L(x) = W^{(L)} f^{L-1}(x) \end{aligned} \tag{A.19}$$

such that ϕ is a differentiable coordinate-wise activation function.

Setting B (ReLU-MLP)**.** We assume the following assumptions hold in our setting:

- We assume $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ for $l \in 1, \ldots, L$ is initialized according to the He et al. [23] initialization, meaning that each scalar parameter is distributed according to $\mathcal{N}(0, 1/n_{l-1})$.
- We assume the width of all hidden layers are identical (and equal to n). The proof extends naturally to the case of non-equal widths as long as $n_{l+1}/n_l \to c_l \in (0, \infty)$ for each consecutive pair of layers.

- We assume ϕ is the ReLU activation. This can be generalized to 1-Lipschitz, ReLU-like functions such as GeLU, PReLU, and so on, as discussed in Appendix A.7.5.
- We assume the training data $\mathcal{X}$ is finite and contained in a compact set and there are no overlapping datapoints.

A Note On Parameterization Although we assume a Gaussian distribution for each scalar variable, the proofs in this section apply to any other distribution used for scalar parameters as long as:

- The variance of the parameters is set according to He et al. [23], and the mean is zero.
- Each scalar parameter is initialized independently of all other ones.
- The distribution used is sub-Gaussian.

This applies to all bounded initialization methods, like truncated normal or uniform on an interval. In what follows, we generally assume each $w_{ij}^{l+1} \in SG(1/n_l)$ with mean zero. In general, the product of two sub-Gaussian distributions has a sub-exponential distribution. For the product of two independent weights, $w_{ij}^{l+1} w_{ab}^{l+1}$ with $i \neq a$ and/or $j \neq b$, we denote the parameters as $\frac{1}{n_l} SE(\nu_p, \alpha_p)$. For $(w_{ij}^{l+1})^2$, we use the parameters $\frac{1}{n_l} SE(\nu_s, \alpha_s)$, whose mean is $\mu_s \neq 0$.

Note that we can recursively define the eNTK of f^{l+1} using the eNTK of f^l as

$$\begin{aligned}
\Theta^{(l+1)}(x_1, x_2) &= \sum_{i=1}^{l} \frac{\partial f^{l+1}(x_1)}{\partial W^{(i)}} \frac{\partial f^{l+1}(x_2)}{\partial W^{(i)}}^\top + \overbrace{\frac{\partial f^{l+1}(x_1)}{\partial W^{l+1}} \frac{\partial f^{l+1}(x_2)}{\partial W^{l+1}}^\top}^{K_D^{l+1}(x_1,x_2)} \\
&= \sum_{i=1}^{l} \frac{\partial \phi(W^{(l+1)} f^l(x_1))}{\partial W^{(i)}} \frac{\partial \phi(W^{(l+1)} f^l(x_2))}{\partial W^{(i)}}^\top + K_D^{l+1}(x_1, x_2) \\
&= \sum_{i=1}^{l} \frac{\partial \phi(W^{(l+1)} f^l(x_1))}{\partial f^l(x_1)} \frac{\partial f^l(x_1)}{\partial W^{(i)}} \frac{\partial f^l(x_2)}{\partial W^{(i)}}^\top \frac{\partial \phi(W^{(l+1)} f^l(x_2))}{\partial f^l(x_2)}^\top + K_D^{l+1}(x_1, x_2) \\
&= \frac{\partial \phi(W^{(l+1)} f^l(x_1))}{\partial f^l(x_1)} \left[\sum_{i=1}^{l} \frac{\partial f^l(x_1)}{\partial W^{(i)}} \frac{\partial f^l(x_2)}{\partial W^{(i)}}^\top \right] \frac{\partial \phi(W^{(l+1)} f^l(x_2))}{\partial f^l(x_2)}^\top + K_D^{l+1}(x_1, x_2) \\
&= \frac{\partial \phi(W^{(l+1)} f^l(x_1))}{\partial f^l(x_1)} \Theta^{(l)}(x_1, x_2) \frac{\partial \phi(W^{(l+1)} f^l(x_2))}{\partial f^l(x_2)}^\top + K_D^{l+1}(x_1, x_2)
\end{aligned} \tag{A.20}$$

where

$$\frac{\partial \phi(W^{(l+1)} f^l(x))}{\partial f^l(x)} = W^{(l+1)} \odot \left[\dot{\phi}(W^{(l+1)} f^l(x)) \right]_{1 \times n} \tag{A.21}$$

and $K_D^{l+1}(x_1, x_2) = f^l(x_1)^\top f^l(x_2) I_n$ is a diagonal matrix. We can think of the last layer as following the same equations with ϕ the identity function, so that $\phi'(x) = 1$. Furthermore, note that using the same approach we can show that pNTK of the layer l can be derived as

$$\hat{\Theta}^{(l+1)}(x_1, x_2) = \frac{\mathbf{1}_n}{\sqrt{n}} \Theta^{(l+1)}(x_1, x_2) \frac{\mathbf{1}_n^\top}{\sqrt{n}} \tag{A.22}$$

where $\mathbf{1}_n$ is the vector of 1s with size n.

We now sketch the proof idea first and then move onto rigorously proving each part of the sketch. First, note that using Equation (A.20) we can recursively calculate the eNTK of a general MLP. We take advantage of this recursive definition and derive bounds for the magnitude of the elements of the eNTK on a layer-to-layer basis recursively. To do so, we first show that the eNTK of the first layer of the NN,

$\Theta^{(1)}(x_1, x_2)$, is in general a diagonal matrix. Then, we present a series of Lemmas that bound the elements of the eNTK of layer $l+1$ based on the magnitude (bounds) of the eNTK of layer l. Finally, based on the derived bounds on the magnitude of elements of the eNTK of a NN with l layers and Equation (A.22), we prove that the Frobenius norm of the pNTK relatively converges to the Frobenius norm of the corresponding eNTK with high probability over random initialization.

Before moving on, it's useful to first show a simple inequality on the elements of a tangent kernel based on the Lipschitz-ness of the activation function; this will help us further in deriving the aforementioned bounds. Define $V^{(l)}(x) = W^{(l)} \odot \left[\dot{\phi}(W^{(l)} f^{l-1}(x))\right]_{1\times n}$. We can write each entry of $\Theta^{(l+1)}(x_1, x_2)$ as

$$\begin{aligned}
\Theta^{(l+1)}(x_1, x_2)_{ij} &= \sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{jb} \Theta^{(l)}(x_1, x_2)_{ab} + f^l(x_1)^\top f^l(x_2)\, \mathcal{I}(i=j) \\
|\Theta^{(l+1)}(x_1, x_2)_{ij}| &\leq |\sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l)}(x_1)_{ia} V^{(l)}(x_2)_{jb} \Theta^{(l)}(x_1, x_2)_{ab}| + |f^l(x_1)^\top f^l(x_2)|\, \mathcal{I}(i=j) \\
&\leq |\sum_{a=1}^{n}\sum_{b=1}^{n} SE_{iajb}(\nu_p, \alpha_p) \Theta^{(l)}(x_1, x_2)_{ab}| + |f^l(x_1)^\top f^l(x_2)|\, \mathcal{I}(i=j)
\end{aligned} \tag{A.23}$$

where $\mathcal{I}$ denotes the 0-1 indicator function, the first inequality follows from the activation function ϕ being 1-Lipschitz and the second inequality follows from the product of two sub-gaussian distributions being distributed as a sub-exponential variable. Note that we use $SE_{iajb}(\nu_p, \alpha_p)$ for the sake of generality but in the case $i = j \wedge a = b$ it actually refers to $SE_{ia}(\nu_s, \alpha_s)$.

Lemma A.7.5 (Diagonality of the first layer's tangent kernel)**.** *For a NN under Setting B, the corresponding eNTK of the first layer $\Theta^{(1)}(x_1, x_2)$ is diagonal. Moreover, there is a corresponding constant $C^{(1)} > 0$ such that for all diagonal elements $\Theta^{(1)}(x_1, x_2)_{ii}$, we have that*

$$|\Theta^{(1)}(x_1, x_2)_{ii}| \leq C^{(1)}.$$

Proof. Consider the one layer NN $f^1(x) = \phi(W^{(1)}x)$. For this case, we have:

$$\Theta^{(1)}(x_1, x_2)_{ij} = \begin{cases} \sum_{a=1}^{D} x_{1a}\dot{\phi}(W_i x_1) x_{2a}\dot{\phi}(W_i x_2) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{A.24}$$

and thus, since the activation function ϕ is 1-Lipschitz we can conclude that for all i, j

$$|\Theta^{(1)}(x_1, x_2)_{ij}| \leq \begin{cases} |x_1^\top x_2| & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}. \tag{A.25}$$

Thus, the tangent kernel of the first layer is a diagonal matrix whose entries are independent of the width of the first layer (n), and can be bounded by a positive constant, given by $C^{(1)} = \max_{(x_1,x_2)\in\mathcal{X}\times\mathcal{X}} |x_1^\top x_2|$. □

Next, we present a series of lemmas that will help us derive the bounds on the elements of the tangent kernel of layer $l+1$ based on the bounds of the tangent kernel of layer l. The next lemma specifically bounds the values of the tangent kernel of the *second* layer, based on the diagonality of the first layer's tangent kernel and (A.23).

Lemma A.7.6. *Consider a NN under Setting B with depth $\geq l+1$. Assume there is a constant $C^{(l)} > 0$ such that $|\Theta^{(l)}(x_1, x_2)_{ii}| < C^{(l)}$ for all $i \in [n_l]$ and every non-diagonal element of $\Theta^{(l)}(x_1, x_2)$ is zero. Then for any small $\delta > 0$ there are corresponding constants $C_1^{(l+1)} = \mathcal{O}(\log(n/\delta)), C_2^{(l+1)} = \mathcal{O}(\log(n/\delta)), n^{l+1} > 0$ such that for any $n > n^{l+1}$, it holds with probability at least $1-\delta$ that, simultaneously for all i, j,*

$$|\Theta^{(l+1)}(x_1, x_2)_{ij}| \leq \begin{cases} C_1^{(l+1)} n & \text{if } i = j \\ C_2^{(l+1)}/\sqrt{n} & \text{if } i \neq j \end{cases}. \tag{A.26}$$

Proof. Recall that

$$\Theta^{(l+1)}(x_1,x_2)_{ij} = \sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{jb} \Theta^{(l)}(x_1,x_2)_{ab} + f^l(x_1)^\top f^l(x_2)\, \mathcal{I}(i=j),$$

where

$$V^{(l+1)}(x_1)_{ia} = W^{(l+1)}{}_{ia} \dot{\phi}\left(\sum_{c=1}^{n} W_{ic}^{(l+1)} f^l(x)_c\right).$$

The weight $W_{ia}^{(l+1)}$ is sub-Gaussian, and the $\dot{\phi}$ term has absolute value at most one, so their product is also sub-Gaussian (as can be easily seen *e.g.* from the moment-based characterization of sub-Gaussianity). As noted before, when the indices match we replace $V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{ia}$ by $\frac{1}{n_l} SE_{ia}(\nu_s, \alpha_s)$ (where $SE_{ia}(\nu_s,\alpha_s)$ is a sub-exponential random variable with mean $\mu_s > 0$ where its mean, variance, ν_s and α_s are independent of n_l). In the case when the indices are not the same, the two are independent but **do not necessarily have mean zero**. Hence, we replace it by $\frac{1}{n_l^2} SE_{iajb}(\nu_p,\alpha_p)$ with mean μ_p where again the mean and variance of $SE_{iajb}(\nu_p,\alpha_p)$ along with ν_p and α_p are independent of n_l.

Based on Equation (A.23) we can expand the elements of $\Theta^{(l+1)}(x_1,x_2)$ as

$$\begin{aligned}
|\Theta^{(l+1)}(x_1,x_2)_{ij}| &\leq \begin{cases} |\sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{ib}\, \Theta^{(l)}(x_1,x_2)_{ab}| + |f^l(x_1)^\top f^l(x_2)| & \text{if } i=j \\ |\sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{jb}\, \Theta^{(l)}(x_1,x_2)_{ab}| & \text{if } i\neq j \end{cases} \\
&= \begin{cases} \sum_{a=1}^{n} \frac{1}{n} SE_{ia}(\nu_s,\alpha_s) |\Theta^{(l)}(x_1,x_2)_{aa}| + |f^l(x_1)^\top f^l(x_2)| & \text{if } i=j \\ |\sum_{a=1}^{n} \frac{1}{n} SE_{iaja}(\nu_p,\alpha_p) \Theta^{(l)}(x_1,x_2)_{aa}| & \text{if } i\neq j\,. \end{cases}
\end{aligned} \tag{A.27}$$

We've assumed an upper bound on the diagonal elements of $\Theta^{(l)}(x_1,x_2)$ of

$C^{(l)}$.We have then that

$$\sum_{a=1}^{n} \frac{1}{n} SE(\nu_s, \alpha_s) \left|\Theta^{(l)}(x_1, x_2)_{aa}\right| \leq \frac{C^{(l)}}{n} \sum_{a=1}^{n} SE_{ia}(\nu_s, \alpha_s) \sim SE\left(\frac{\nu_s C^{(l)}}{\sqrt{n}}, \frac{\alpha_s C^{(l)}}{n}\right),$$

since each item in the last sum is independent of all others. Noting this term has mean $C^{(l)}\mu_s$, we get a high-probability upper bound via Corollary A.7.4. Since there are n independent such terms, one for each $i \in [n]$, we'll want an upper bound that holds over all of them: with probability at least $1 - \delta_1$ we have

$$\begin{aligned}\sum_{a=1}^{n} SE_{ia}(\nu_s, \alpha_s) \left|\Theta^{(l)}(x_1, x_2)_{aa}\right| &\leq C^{(l)}\mu_s + \max\left(2C^{(l)}\sqrt{\frac{2}{n}\log\frac{2n}{\delta}}, \frac{8C^{(l)}}{n}\log\frac{2n}{\delta_1}\right) \\ &\leq C^{(l)}\left[\mu_s + 2\sqrt{\frac{2}{n}\log\frac{2n}{\delta_1}}\right]\end{aligned}$$

for n large enough that $2\sqrt{2} > 8\sqrt{\frac{1}{n}\log\frac{2n}{\delta_1}}$, which will be the case for $n = \Omega\left(\text{polylog}\,\frac{1}{\delta_1}\right)$.

Likewise, Lemma A.7.12 shows a high-probability upper bound on $|f^l(x_1)^\top f^l(x_2)|$ for all $n = \Omega\left(\log\frac{1}{\delta_g}\right)$ with probability at least $1 - \delta_g$. Combining the two with a union bound, we have with probability at least $1 - \delta_1 - \delta_g$ that

$$\max_i |\Theta^{(l+1)}(x_1, x_2)_{ii}| \leq C^{(l)}\left(\mu_s + \sqrt{\frac{8}{n}\log\frac{2n}{\delta_1}}\right) + G^{(l)}n.$$

This inequality is dominated by the $G^{(l)}n$ term. Thus we can find an upper bound $(G^{(l)} + \varepsilon)n$, where for $n \geq \max\left(\Omega\left(\log\frac{1}{\delta_g}\right), \Omega\left(\text{polylog}(\frac{1}{\delta_1})\right)\right)$, we have that $\varepsilon = \mathcal{O}\left(\frac{1}{n} + \frac{1}{n^{3/2}}\sqrt{\log\frac{2n}{\delta_1}}\right) = \mathcal{O}\left(\sqrt{\log\frac{1}{\delta_1}}\right)$ can be chosen to be independent of n.

For the off-diagonal terms, we have

$$|\Theta^{(l+1)}(x_1, x_2)_{ij}| \leq \frac{C^{(l)}}{n}|\sum_{a=1}^{n} SE_{iaja}(\nu_p, \alpha_p)|.$$

Again applying Corollary A.7.4 for each i, j and taking a union bound gives that

with probability at least $1-\delta_2$,

$$\max_{i,j}|\Theta^{(l+1)}(x_1,x_2)_{ij}| \leq C^{(l)}\left(\frac{\mu_p}{n}+\sqrt{\frac{2}{n}\nu_p\log\frac{n(n-1)}{\delta_2}}\right)$$

as long as n is large enough that $\nu_p > \alpha_p\sqrt{\frac{2}{n}\log\frac{n(n-1)}{\delta_2}}$, again true as long as $n=\Omega\left(\text{polylog}\,\frac{1}{\delta_2}\right)$.

Hence, based on this bound and also the bound provided for the diagonal entries, we can achieve the desired result by setting $n \geq \max\left(\Omega\left(\log\frac{1}{\delta_g}\right),\Omega\left(\text{polylog}\,\frac{1}{\delta_2}\right),\Omega\left(\text{polylog}\,\frac{1}{\delta_1}\right)\right)$ with probability at least $1-(\delta_1+\delta_2+\delta_g)$. □

Lemma A.7.7. *Consider a NN under Setting B with depth $\geq l+1$. Assume there are constants $C_1^{(l)}=\mathcal{O}(\log(n/\delta_p)), C_2^{(l)}=\mathcal{O}(\log(n/\delta_p))$ such that we have $|\Theta^{(l)}(x_1,x_2)_{ii}| < C_1^{(l)}n$ and $|\Theta^{(l)}(x_1,x_2)_{ij}| < C_2^{(l)}\sqrt{n}$ with probability at least $1-\delta_p$ and any width $n > n_p$ **simultaneously** for all i,j. Then for any arbitrary small $\delta>0$ there are constants $C_1^{(l+1)}=\mathcal{O}(\log(n/\delta)), C_2^{(l+1)}=\mathcal{O}(\log(n/\delta)), n^{l+1}>0$ such that for any $n > n^{l+1}$ and **simultaneously** for all i,j*

$$|\Theta^{(l+1)}(x_1,x_2)_{ij}| \leq \begin{cases} C_1^{(l+1)}n & \text{if } i=j \\ C_2^{(l+1)}\sqrt{n} & \text{if } i\neq j \end{cases} \tag{A.28}$$

with probability at least $1-\delta$. In other words, the magnitude of elements of the tangent kernel in the recursive definition will not grow.

Proof. Using the same expansion that we utilized in the proof for the previous Lemma, for the diagonal elements of $\Theta^{(l+1)}(x_1,x_2)$ we have:

$$
\begin{aligned}
|\Theta^{(l+1)}(x_1,x_2)_{ii}| &\leq |\sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia}V^{(l+1)}(x_2)_{ib}\Theta^{(l)}(x_1,x_2)_{ab}| + G^{(l)}n \\
&= \underbrace{\frac{1}{n}\sum_{a=1}^{n} SE_{ia}(\nu_s,\alpha_s)|\Theta^{(l)}(x_1,x_2)_{aa}|}_{\Theta_1^{(l+1)}(x_1,x_2)_{ii}} \\
&\quad + \underbrace{\frac{1}{n}|\sum_{a=1}^{n}\sum_{b=1,b\neq a}^{n} SE_{iaib}(\nu_p,\alpha_p)\Theta^{(l)}(x_1,x_2)_{ab}|}_{\Theta_2^{(l+1)}(x_1,x_2)_{ii}} + G^{(l)}n
\end{aligned}
\tag{A.29}
$$

Starting with $\Theta_1^{(l+1)}(x_1,x_2)_{ii}$ and using the assumed bound on the elements of $\Theta_1^{(l)}(x_1,x_2)$ we have

$$
|\Theta_1^{(l+1)}(x_1,x_2)_{ii}| \leq \frac{C_1^{(l)}n}{n}\sum_{a=1}^{n} SE_{ia}(\nu_s,\alpha_s) \sim SE\left(C_1^{(l)}\sqrt{n}\nu_s, C_1^{(l)}\alpha_s\right) \tag{A.30}
$$

with probability at least $1-\delta_p$. Noting that this term has a mean of $C_1^{(l)}\mu_s$, we can get a high-probability upper bound via Corollary A.7.4 that applies over all $i \in [n]$ terms:

$$
|\Theta_1^{(l+1)}(x_1,x_2)_{ii}| \leq C_1^{(l)}\left[\mu_s + \sqrt{8n\log\frac{2n}{\delta_1}}\right] \tag{A.31}
$$

with probability at least $1-\delta_p-\delta_1$ for n large enough that $n > \max\left(n_p, \Omega\left(\text{polylog}\,\frac{1}{\delta_1}\right)\right)$. Similarly, for all $i \in [n]$ we have:

$$
\begin{aligned}
|\Theta_2^{(l+1)}(x_1,x_2)_{ii}| &\leq C_2^{(l)}\sqrt{n}|\sum_{a=1}^{n}\sum_{b=1,b\neq a}^{n} SE_{iaib}(\nu_p,\alpha_p)| \\
&\leq C_2^{(l)}\left[\mu_p\sqrt{n} + \sqrt{8n\log\frac{2n}{\delta_2}}\right]
\end{aligned}
\tag{A.32}
$$

with probability at least $1-\delta_p-\delta_2$ for n large enough that $n > \max\left(n_p, \Omega\left(\text{polylog}\,\frac{1}{\delta_2}\right)\right)$.

Just like the previous Lemma, both of these terms are dominated by the $G^{(l)}n$ term in the inequality for diagonal elements and

thus, similar to the previous Lemma, we can show that there is an $n^{l+1} = \max\left(n_p, \Omega\left(\log\frac{1}{\delta_g}\right), \Omega\left(\text{polylog}\frac{1}{\delta_1}\right), \Omega\left(\text{polylog}\frac{1}{\delta_2}\right)\right)$ (where accordingly $\Omega\left(\log\frac{1}{\delta_g}\right)$ is the minimum width coming from Lemma A.7.12) such that for all $n > n^{l+1} \wedge i \in [n]$; $|\Theta^{(l+1)}(x_1, x_2)_{ii}| < (G^{(l)}+\varepsilon)n$ with probability at least $1-(\delta_1+\delta_2+\delta_p+\delta_g)$ where $\varepsilon = \mathcal{O}\left(\sqrt{\log\frac{1}{\delta_1}} + \sqrt{\log\frac{1}{\delta_2}}\right)$ can be selected to be independent of n.

For the non-diagonal elements of $\Theta^{(l+1)}(x_1, x_2)$ we have:

$$\begin{aligned}
|\Theta^{(l+1)}(x_1,x_2)_{ij}| &= |\sum_{a=1}^{n}\sum_{b=1}^{n} V^{(l+1)}(x_1)_{ia} V^{(l+1)}(x_2)_{jb} \Theta^{(l)}(x_1,x_2)_{ab}| \\
&\leq \underbrace{\frac{1}{n}|\sum_{a=1}^{n} SE_{iaja}(\nu_p,\alpha_p)\Theta^{(l)}(x_1,x_2)_{aa}|}_{\Theta_1^{(l+1)}(x_1,x_2)_{ij}} \\
&\quad + \underbrace{\frac{1}{n}|\sum_{a=1}^{n}\sum_{b=1,b\neq a}^{n} SE_{iajb}(\nu_p,\alpha_p)\Theta^{(l)}(x_1,x_2)_{ab}|}_{\Theta_2^{(l+1)}(x_1,x_2)_{ij}}.
\end{aligned} \tag{A.33}$$

Using the same strategy, we start by $\Theta_1^{(l+1)}(x_1,x_2)_{ij}$ and use the assumed bound on the elements of $\Theta_1^{(l)}(x_1,x_2)$ to show that

$$\begin{aligned}
|\Theta_1^{(l+1)}(x_1,x_2)_{ij}| &\leq \frac{C_1^{(l)} n}{n}\sum_{a=1}^{n}|SE_{iaja}(\nu_p,\alpha_p)| \sim |SE\left(C_1^{(l)}\sqrt{n}\nu_p, C_1^{(l)}\alpha_p\right)| \\
&\leq C_1^{(l)}\left[\frac{\mu_p}{n} + \sqrt{8n\log\frac{2n(n-1)}{\delta_1'}}\right]
\end{aligned} \tag{A.34}$$

for all $i \in [n] \wedge j \in [n] \wedge i \neq j \wedge n > \max\left(n_p, \Omega\left(\text{polylog}\frac{1}{\delta_1'}\right)\right)$ with probability at least $1-\delta_p-\delta_1'$.

Similarly, for $\Theta_2^{(l+1)}(x_1,x_2)_{ij}$ we have:

$$
\begin{aligned}
|\Theta_2^{(l+1)}(x_1,x_2)_{ij}| &\leq \frac{C_2^{(l)}\sqrt{n}}{n}|\sum_{a=1}^{n}\sum_{b=1,b\neq a}^{n} SE_{iajb}(\nu_p,\alpha_p)| \\
&\sim \frac{C_2^{(l)}\sqrt{n}}{n}|\sum_{a=1}^{n^2} SE\left(\nu_p,\alpha_p\right)| = |SE\left(C_2^{(l)}\sqrt{n}\nu_p, \frac{C_2^{(l)}\alpha_p}{\sqrt{n}}\right)| \\
&\leq C_2^{(l)}\left[\frac{\mu_p}{\sqrt{n}} + \sqrt{8n\log\frac{2n(n-1)}{\delta_2'}}\right]
\end{aligned}
\tag{A.35}
$$

for all $i \in [n] \wedge j \in [n] \wedge i \neq j \wedge n > \max\left(n_p, \Omega\left(\text{polylog}\,\frac{1}{\delta_2'}\right)\right)$ with probability at least $1-\delta_p-\delta_2'$.

Setting $n^{l+1} = \max_{\delta\in\{\delta_1,\delta_2,\delta_1',\delta_2'\}}\left(n_p, n_m, \Omega\left(\text{polylog}\,\frac{1}{\delta}\right)\right)$, we can see that the lemma's claim holds with probability at least $1-(\delta_1+\delta_2+\delta_1'+\delta_2'+\delta_p+\delta_g)$ and $n > n^{l+1}$ as desired.

□

An alert reader already may notice that connecting the previous three Lemmas would result in an upper bound for the diagonal and non-diagonal elements of the eNTK of the NN f with arbitrary depth at initialization.

Lemma A.7.8. *Consider a NN f under Setting B. For every arbitrary small $\delta > 0$, there are constants $C_1 = \mathcal{O}(\log(n/\delta)), C_2 = \mathcal{O}(\log(n/\delta)), n_0 > 0$ such that with probability at least $1-\delta$ over the random initialization, it holds simultaneously for all i, j that if $n > n_0$, the corresponding eNTK of f on the arbitrary datapoints x_1 and x_2 satisfies*

$$
|\Theta(x_1,x_2)_{ij}| \leq \begin{cases} C_1 n & \text{if } i = j \\ C_2\sqrt{n} & \text{if } i \neq j\,. \end{cases}
\tag{A.36}
$$

Proof. Starting with Lemma A.7.5, we have a bound for the entries of the first layer's tangent kernel. Plugging this bound into Lemma A.7.6 we get a bound on the elements of the second layer. Next, we can recursively apply Lemma A.7.7. Note that, however, this recursion will induce a new factor in our bound that depends on the depth of our NN, and thus, would enforce the minimum width to also depend on depth. Assume we have applied the first three lemmas and we want to consider the next $L-3$ recursions of application of Lemma A.7.7 for an NN with $L > 3$

layers. As our assumption, for any arbitrary small $\delta > 0$ there are constants $C_1^{(3)} = \mathcal{O}(\text{polylog}(n/\delta)), C_2^{(3)} = \mathcal{O}(\text{polylog}(n/\delta)), n^{(3)} > 0$ such that for any $n > n^{(3)}$ and i, j

$$|\Theta^{(3)}(x_1, x_2)_{ij}| \leq \begin{cases} C_1^{(3)} n & \text{if } i = j \\ C_2^{(3)} \sqrt{n} & \text{if } i \neq j \end{cases} \tag{A.37}$$

Now, through recursively applying Lemma A.7.7 we can see that for the layer $L > 3$ for any $\delta > 0$ and $n > n^L$ there are constants $C_1^{(L)} = \mathcal{O}\left(\log(n/\delta)\right), C_2^{(L)} = \mathcal{O}\left(\log(n/\delta)\right), n^{(L)} > 0$ such that

$$|\Theta^{(L)}(x_1, x_2)_{ij}| \leq \begin{cases} C_1^{(L)} n & \text{if } i = j \\ C_2^{(L)} \sqrt{n} & \text{if } i \neq j \end{cases} \tag{A.38}$$

with probability at least $1 - \delta$. The change in n^L is also trackable as presented in Lemma A.7.7. □

Lemma A.7.9. *Consider a NN f under Setting B. For every arbitrary small $\delta > 0$ and the arbitrary datapoints x_1 and x_2, it holds that*

$$\|\Theta(x_1, x_2) - \hat{\Theta}(x_1, x_2)\|_F \leq \mathcal{O}(\sqrt{n} \log n) \tag{A.39}$$

with probability at least $1 - \delta$ over random initialization for any $n > n_0$ where $n_0 = \Omega\left(\text{polylog}(L/\delta)\right)$. In other words, the Frobenius norm of the difference between eNTK and pNTK evaluated on two datapoints are bounded by $\mathcal{O}(\sqrt{n} \log n)$.

Proof. We note by $D(x_1, x_2) = \Theta^{(L)}(x_1, x_2) - \hat{\Theta}^{(L)}(x_1, x_2) \otimes I_O$. Using the

expansion provided in Equation (A.22) we can write the

$$|D(x_1,x_2)_{ij}| \leq \begin{cases} |\sum_{a=1}^{n}\sum_{b=1}^{n} SE_{iaib}(\nu_p,\alpha_p)\Theta^{(L-1)}(x_1,x_2)_{ab} \\ \qquad -\frac{1}{O}\sum_{c=1}^{O}\sum_{d=1}^{O}\sum_{a=1}^{n}\sum_{b=1}^{n} SE_{cadb}(\nu_p,\alpha_p)\Theta^{(L-1)}(x_1,x_2)_{ab}| \\ |\sum_{a=1}^{n}\sum_{b=1}^{n} SE_{iajb}(\nu_p,\alpha_p)\Theta^{(L-1)}(x_1,x_2)_{ab}| \end{cases} \tag{A.40}$$

where first option is for diagonal elements $i = j$ and second is for non-diagonal ones.

Applying Lemma A.7.8 we can assume there are constants $C_1^{(L-1)} = \log(n/\delta_p), C_2^{(L-1)} = \log(n/\delta_p)$ such that $|\Theta^{(L-1)}(x_1,x_2)_{aa}| < C_1^{(L-1)}n$ and $|\Theta^{(L-1)}(x_1,x_2)_{ab}| < C_2^{(L-1)}\sqrt{n}$ with probability at least $1-\delta_p$ **simultaneously** for all a, b. Thus we can write the diagonal elements of $D(x_1,x_2)$ as

$$|D(x_1,x_2)_{ii}| \leq \underbrace{C_1^{(L-1)}n|\sum_{a=1}^{n}\left(\overbrace{SE_{ia}(\nu_s,\alpha_s) - \frac{1}{O}\sum_{c=1}^{O}SE_{ca}(\nu_s,\alpha_s) - \frac{1}{O}\sum_{c=1}^{O}\sum_{d=1,d\neq c}^{O}SE_{cada}(\nu_p,\alpha_p)}^{SE(\nu_s',\alpha_s')}\right)|}_{D_1(x_1,x_2)_{ii}}$$
$$+ \underbrace{C_2^{(L-1)}\sqrt{n}|\sum_{a=1}^{n}\sum_{b=1,b\neq a}^{n}\left(\overbrace{SE_{iaib}(\nu_p,\alpha_p) - \frac{1}{O}\sum_{c=1}^{O}\sum_{d=1}^{O}SE_{cadb}(\nu_p,\alpha_p)}^{SE(\nu_p',\alpha_p')}\right)|}_{D_2(x_1,x_2)_{ii}}$$

It can be easily seen that the random variables being summed over in each of the two lines above are linear combinations of independent sub-exponential random variables and thus, their linear combinations are also sub-exponential random variables. Moreover, we can simplify the analysis from here on through noting that all summation terms follow the same pattern. If

$$X \sim a|\sum_{i=1}^{b} SE\left(\nu_p, \alpha_p\right)| = a|SE\left(\nu_p\sqrt{b}, \alpha_p\right)|. \tag{A.41}$$

applying Corollary A.7.4 reveals

$$|X| < 2a\left(\max\left(\nu_p\sqrt{b\log\frac{4}{\delta}}, \alpha_p\log\frac{4}{\delta}\right) + \frac{b}{2}\left|\mathbb{E}[X]\right|\right) \tag{A.42}$$

with probability at least $1-\delta/2$.

Now, starting with $D_1(x_1,x_2)_{ii}$ we have that:

$$\begin{aligned}|D_1(x_1,x_2)_{ii}| &\leq C_1^{(L-1)}\sum_{a=1}^{n} SE(\nu_s', \alpha_s') \\ &\leq 2C_1^{(L-1)}\max\left(\nu_s'\sqrt{n\log\frac{4}{\delta}}, \alpha_s'\log\frac{4}{\delta}\right) + C_1^{(L-1)}\mu_s'\end{aligned} \tag{A.43}$$

with probability at least $1-\delta/2-\delta_p$.

Accordingly we can claim

$$|D_2(x_1,x_2)_{ii}| < 2C_2^{(L-1)}\max\left(\nu_p'\sqrt{n\log\frac{4}{\delta}}, \alpha_p'\log\frac{4}{\delta}\right) + \frac{C_2^{(L-1)}\mu_p'}{n}, \tag{A.44}$$

again with probability at least $1-\delta/2-\delta_p$. Moreover, one can easily apply the same technique and see that $D(x_1,x_2)_{ij}$ for $i \neq j$ follows a similar bound to the one of Equation (A.44).

Thus, loosening the off-diagonal terms for simplicity, applying a union bound on the previous three inequalities yields

$$|D(x_1,x_2)_{ij}| < 4\left(C_1^{(L-1)} + C_2^{(L-1)}\right)\sqrt{n}\left(\max\left(\nu_p'\sqrt{\log\frac{4n^2}{\delta}}, \alpha_p'\log\frac{4n^2}{\delta}\right)\right) \tag{A.45}$$

with probability at least $1-(\delta+\delta_p)$.

Finally, as $\|D(x_1,x_2)\|_F = \sqrt{\sum_{i,j} D(x_1,x_2)_{ij}^2}$, if each entry's absolute value is less than $t > 0$ then the Frobenius norm is less than tO. Thus we can combine a

bound on each of the O^2 entries to see that

$$\Pr\left(\|D(x_1,x_2)\|_F \leq 4O\left(C_1^{(L-1)}+C_2^{(L-1)}+\mu_s'\right)\sqrt{n}\max\left(2\sqrt{\log\frac{4O^2n^2}{\delta}},4\log\frac{4O^2n^2}{\delta}\right)\right)\geq 1-(\delta+\delta_p) \tag{A.46}$$

as desired.

□

Lemma A.7.10. *Consider a NN f under Setting B. For every arbitrary small $\delta > 0$ and the arbitrary datapoints x_1 and x_2, it holds that*

$$\|\Theta(x_1,x_2)\|_F \geq \Omega(n) \tag{A.47}$$

with probability at least $1-\delta$ over random initialization.

Proof. Considering that the dot product of post-activations appear in the diagonal elements of the eNTK in conjunction with Lemma A.7.12, this is straightforward. Note that this bound also applies to the maximum eigenvalue of the eNTK matrix since the maximum eigenvalue is bigger than (or equal to) the sum of elements of the matrix divided by the number of columns. □

We are finally ready to present the proof of Theorem 3.3.1.

Theorem A.7.11. *Consider a NN f under Setting B. For every arbitrary small $\delta > 0$ and the arbitrary datapoints x_1 and x_2, there exists n_0 such that*

$$\frac{\|\Theta^{(L)}(x_1,x_2)-\hat{\Theta}^{(L)}(x_1,x_2)\otimes I_O\|_F}{\|\Theta^{(L)}(x_1,x_2)\|_F}=\mathcal{O}\left(\frac{\log n}{\sqrt{n}}\right)=\tilde{\mathcal{O}}\left(\frac{1}{\sqrt{n}}\right) \tag{A.48}$$

with probability at least $1-\delta$ for $n > n_0$.

Proof. The proof is straightforward from applying Lemmas A.7.9 and A.7.10. □

Lemma A.7.12. *Consider a NN under Setting B with $L \geq 2$ and ReLU activation function. The dot product of two post-activations $|f^{(l)}(x_1)^\top f^{(l)}(x_2)|$ grows linearly with the width of the network with high probability over random initialization.*

Proof. We begin by showing that the dot product of the post-activations of the first layer of the NN under setting Setting B grow linearly using a simple Hoeffding bound. Next, we apply Thorem 1 from [4] to show that the magnitude of this dot product is preserved in the next layers. First, note that as we assume the data lies in a compact set and as the post-activations are all positive, one can easily see that for each $x_1, x_2 \in \mathcal{X}$ and for all $l \in [L]$ we have that:

$$\min_{x\in\mathcal{X}} \|f^{(l)}(x)\|^2 \leq f^{(l)}(x_1)^\top f^{(l)}(x_2) \leq \max_{x\in\mathcal{X}} \|f^{(l)}(x)\|^2. \tag{A.49}$$

To simplify the proofs in this Lemma, we use this fact and instead work with the norm of the post-activations and we note that the final result on the norms can be accordingly applied to dot products of post-activations of different inputs. For the first layer, we have that $f^{(1)}(x) = \phi(W^{(1)}x)$ where $W_{ij}^{(l+1)} \sim \mathcal{N}(0, \frac{1}{n_l})$ and $x \in \mathbb{R}^{n_0}$. Hence, each $f^{(1)}(x)_i$ is i.i.d and distributed as $\mathcal{N}^R(0, \frac{\|x\|^2}{n_0})$ where $\mathcal{N}^R$ is the Rectified Normal Distribution. Using the properties of the Rectified Normal distribution we get that:

$$\mathbb{E}[\|f^{(1)}(x)\|^2] = \frac{n\|x\|^2}{n_0} \tag{A.50}$$

Next, as the Rectified Normal is a sub-gaussian distribution, we can apply the Hoeffding bound to see that

$$\Pr\left[|\|f^{(1)}(x)\|^2 - n\mu_1| \leq \varepsilon_1\right] \geq 1 - \delta_1 \tag{A.51}$$

where $\delta_1 = 2\exp\left(-\frac{\varepsilon_1^2}{2\sigma^2}\right)$, $\mu_1 = \frac{\|x\|^2}{n_0}$ and σ is the standard deviation of $\|f^{(1)}(x)\|^2$ over random initialization of the weights of the first layer. Next, we can adapt Theorem 1 from [4] to see that for post activations of layer $l \in [2 - L]$

$$\Pr\left[(1-\varepsilon)^{l-1}\|f^{(1)}(x)\|^2 \leq \|f^{(l)}(x)\|^2 \leq (1+\varepsilon)^{l-1}\|f^{(1)}(x)\|^2\right] \geq 1 - \delta_2 \tag{A.52}$$

where $\delta_2 = 2N(l-1)\exp\left(-n\left(\frac{\varepsilon}{4} + \log\frac{2}{1+\sqrt{1+\varepsilon}}\right)\right)$, N is the size of our dataset and ε is any positive small constant. Combining this with the result from the first layer's post-activations we can see that

$$(1-\varepsilon_2)^{l-1}(n\mu_1 - \varepsilon_1) \leq \|f^{(l)}(x)\|^2 \leq (1+\varepsilon_2)^{l-1}(n\mu_1 + \varepsilon_1) \tag{A.53}$$

with probability at least $1-\delta_1-\delta_2$. Hence, for any $\delta>0$, $n=\Omega\left(\log\frac{1}{\delta}\right)$ and $(x_1,x_2)\in\mathcal{X}\times\mathcal{X}$ one can come up with constants $G_1^{(l)}=\Omega\left(\log(\frac{n}{\delta})\right), G_2^{(l)}=\mathcal{O}\left(\log(\frac{n}{\delta})\right)$ for post activations of layer l such that

$$G_1^{(l)}n\leq f^{(l)}(x_1)^\top f^{(l)}(x_2)<G_2^{(l)}n \tag{A.54}$$

with probability at least $1-\delta$ (note that exact values of $G_1^{(l)}$ and $G_1^{(l)}$ depend on l and N too). □

A.7.3 pNTK's Maximum Eigenvalue Converges to eNTK's Maximum Eigenvalue as Width Grows

In this subsection, we present a formal proof for Theorem 3.3.4.

Proof. Note that, as both pNTK and eNTK are symmetric PSD matrices, their maximum eigenvalues are equal to their spectral norm. Furthermore, the spectral norm of a matrix is upper-bounded by its Frobenius norm. Now, note that according to the triangle inequality, we have

$$\begin{aligned}\|\Theta(x_1,x_2)\|&=\|\hat{\Theta}(x_1,x_2)\otimes I_O+\left(\Theta(x_1,x_2)-\hat{\Theta}(x_1,x_2)\otimes I_O\right)\|\\&\leq\|\hat{\Theta}(x_1,x_2)\otimes I_O\|+\|\Theta(x_1,x_2)-\hat{\Theta}(x_1,x_2)\otimes I_O\|\end{aligned} \tag{A.55}$$

Thus

$$\|\Theta(x_1,x_2)\|-\|\hat{\Theta}(x_1,x_2)\otimes I_O\|\leq\|\Theta(x_1,x_2)-\hat{\Theta}(x_1,x_2)\otimes I_O\|. \tag{A.56}$$

which according to (A.46) together with the fact that for any matrix A, $\lambda_{\max}(A\otimes I)=\lambda_{\max}(A)$ implies that with probability at least $1-\delta$,

$$\begin{aligned}&\left|\lambda_{\max}\left(\Theta(x_1,x_2)\right)-\lambda_{\max}\left(\hat{\Theta}(x_1,x_2)\right)\right|\leq\\&\qquad 4O\left(C_1^{(L-1)}+C_2^{(L-1)}\right)\sqrt{n}\max\left(\sqrt{\log\frac{4O^2}{\delta}},\sqrt{2}\log\frac{4O^2}{\delta}\right).\end{aligned} \tag{A.57}$$

Moreover, as mentioned in the proof of Lemma A.7.10, combining the previous

inequality with the fact that $\lambda_{\max}\left(\Theta(x_1, x_2)\right) \geq \Omega(n)$ with high probability shows that there exists δ' and n_0 such that

$$\left| \frac{\lambda_{\max}\left(\Theta(x_1, x_2)\right) - \lambda_{\max}\left(\hat{\Theta}(x_1, x_2)\right)}{\lambda_{\max}\left(\Theta(x_1, x_2)\right)} \right| \leq \tilde{\mathcal{O}}(1/\sqrt{n}) \tag{A.58}$$

with probability $1 - \delta'$ over random initialization for $n > n_0$ as desired.

□

A.7.4 Kernel Regression Using pNTK vs Kernel Regression Using eNTK

In this subsection we provide a formal proof for Theorem 3.3.5.

Proof. We start by proving a simpler version of a theorem, and then show a correspondence that expands the result of the simpler proof to the original Theorem. Assuming $|\mathcal{X}| = |\mathcal{Y}| = N$ (training data), we define

$$h(x) = \Theta(x_1, \mathcal{X})\Theta(\mathcal{X}, \mathcal{X})^{-1}\mathcal{Y} \text{ and } \hat{h}(x) = \left(\hat{\Theta}(x_1, \mathcal{X}) \otimes I_O\right)\left(\hat{\Theta}(\mathcal{X}, \mathcal{X}) \otimes I_O\right)^{-1}\mathcal{Y}. \tag{A.59}$$

Note that as the result of kernel regression (without any regularization) does not change with scaling the kernel with a fixed scalar, we can use a weighted version of the kernels mentioned in the previous equation without loss of generality. Accordingly, we define

$$\alpha = \left(\frac{1}{n}\Theta(\mathcal{X}, \mathcal{X})\right)^{-1}\mathcal{Y} \text{ and } \hat{\alpha} = \left(\frac{1}{n}\hat{\Theta}(\mathcal{X}, \mathcal{X}) \otimes I_O\right)^{-1}\mathcal{Y}. \tag{A.60}$$

Using the fact that $\hat{M}^{-1} - M^{-1} = -\hat{M}^{-1}(\hat{M} - M)M^{-1}$ and $(A \otimes I)^{-1} = A^{-1} \otimes I$ we can show that

$$\hat{\alpha} - \alpha = -\hat{\Theta}(\mathcal{X}, \mathcal{X})^{-1} \otimes I_O \left(\frac{1}{n}\hat{\Theta}(\mathcal{X}, \mathcal{X}) \otimes I_O - \frac{1}{n}\Theta(\mathcal{X}, \mathcal{X})\right)^{-1} \Theta(x_1, x_2)\mathcal{Y} \tag{A.61}$$

Assume $\lambda = \min\left(\lambda_{\min}(\Theta(\mathcal{X}, \mathcal{X})), \lambda_{\min}(\hat{\Theta}(\mathcal{X}, \mathcal{X}))\right)$. Then

$$\|\hat{\alpha} - \alpha\| \leq \frac{1}{\lambda^2}\|\frac{1}{n}\hat{\Theta}(\mathcal{X},\mathcal{X}) \otimes I_O - \frac{1}{n}\Theta(\mathcal{X},\mathcal{X})\|\|\mathcal{Y}\| \tag{A.62}$$

Plugging into the formula for kernel regression, we get that

$$\begin{aligned}\hat{h}(x) - h(x) &= \left(\frac{1}{n}\hat{\Theta}(x,\mathcal{X}) \otimes I_O\right)\hat{\alpha} - \frac{1}{n}\Theta(x,\mathcal{X})\alpha \\ &= \left(\frac{1}{n}\hat{\Theta}(x,\mathcal{X}) \otimes I_O - \frac{1}{n}\Theta(x,\mathcal{X})\right)\hat{\alpha} + \frac{1}{n}\Theta(x,\mathcal{X})(\hat{\alpha} - \alpha)\end{aligned} \tag{A.63}$$

Thus

$$\begin{aligned}\|\hat{h}(x) - h(x)\| &\leq \|\frac{1}{n}\hat{\Theta}(x,\mathcal{X}) \otimes I_O - \frac{1}{n}\Theta_f(x,\mathcal{X})\|\|\hat{\alpha}\| + \|\frac{1}{n}\Theta(x,\mathcal{X})\|\|\hat{\alpha} - \alpha\| \\ &\leq \frac{1}{\lambda}\|\frac{1}{n}\hat{\Theta}(x,\mathcal{X}) \otimes I_O - \frac{1}{n}\Theta(x,\mathcal{X})\|\|\mathcal{Y}\| \\ &\quad + \frac{1}{\lambda^2}\|\frac{1}{n}\Theta(x,\mathcal{X})\|\|\frac{1}{n}\hat{\Theta}(\mathcal{X},\mathcal{X}) \otimes I_O - \frac{1}{n}\Theta(\mathcal{X},\mathcal{X})\|\|\mathcal{Y}\|.\end{aligned} \tag{A.64}$$

Now, note that as for a block matrix A of A_{ij} blocks we have that $\|A\| \leq \sum_{i,j}\|A_{ij}\|$ it follows that for any matrix valued kernel K

$$\|K(\mathcal{X},\mathcal{X})\| \leq \sum_{x_1,x_2 \in \mathcal{X}} \|K(x_1,x_2)\|. \tag{A.65}$$

Using this fact, we can rewrite the bound as

$$\begin{aligned}\|\hat{h}(x) - h(x)\| &\leq \frac{N}{\lambda}\|\frac{1}{n}\hat{\Theta}(x,x_1^*) \otimes I_O - \frac{1}{n}\Theta(x,x_1^*)\|\|\mathcal{Y}\| \\ &\quad + \frac{N^2}{\lambda^2}\|\frac{1}{n}\Theta(x,\mathcal{X})\|\|\frac{1}{n}\hat{\Theta}(x_2^*,x_3^*) \otimes I_O - \frac{1}{n}\Theta(x_2^*,x_3^*)\|\|\mathcal{Y}\|\end{aligned} \tag{A.66}$$

for some particular $x_1^*, x_2^*, x_3^* \in \mathcal{X}$. Using (A.46), we can see with probability at least $1 - \delta$ that

$$\left\|\hat{h}(x) - h(x)\right\| \leq \frac{4NO\alpha}{\lambda\sqrt{n}} \max\left(\sqrt{\log\frac{4O^2}{\delta}}, \sqrt{2}\log\frac{4O^2}{\delta}\right)\|\mathcal{Y}\|\left(1 + \frac{N}{\lambda}\|\frac{1}{n}\Theta(x,\mathcal{X})\|\right). \tag{A.67}$$

To show the correspondence between $\hat{h}(x)$ and $\hat{f}^{lin}(x)$, as in (3.5), note that

$$\begin{aligned}\hat{h}(x) &= \left(\hat{\Theta}(x,\mathcal{X}) \otimes I_O\right)\left(\hat{\Theta}(\mathcal{X},\mathcal{X})^{-1} \otimes I_O\right)\mathcal{Y} \\ &= \left(\hat{\Theta}(x,\mathcal{X})\hat{\Theta}(\mathcal{X},\mathcal{X})^{-1} \otimes I_O\right)\mathcal{Y} \\ &= \text{vec}\left(I_O \mathcal{Y}_v \hat{\Theta}(x,\mathcal{X})\hat{\Theta}(\mathcal{X},\mathcal{X})^{-1}\right)\end{aligned} \tag{A.68}$$

where $\mathcal{Y}_v = \text{vec}^{-1}(\mathcal{Y})$ is the result of inverse of the vectorization operation, converting the $NO \times 1$ vector to a $O \times N$ matrix. Thus, $\hat{h}(x) = \hat{\Theta}(x,\mathcal{X})\hat{\Theta}(\mathcal{X},\mathcal{X})^{-1}\mathcal{Y}'$ where $\mathcal{Y}'$ is the $N \times O$ matrix derived from reshaping the $NO \times 1$ vector $\mathcal{Y}$. The proof is complete. □

A.7.5 Extending the Proofs to Other Architectures

In this subsection we elaborate on how one can extend the current proofs to different architectures. We start by providing a sketch on how the dense weight vectors can be replaced by other layers of choice like convolutions. First, note how the linear weights are used in Equation (A.20). As mentioned in Section 6 of Yang [77], we can accordingly write the same expansion for other forward computational graphs and derive the corresponding canonincal decomposition for them. In subsection 6.2.1, Yang [77] provides a concrete example on how one can derive this expansion for a general RNN-like architecture. As the proofs provided in this section depend on the MLP structure only by means of the canonical decomposition, one can extend them to a general architecture by deriving the corresponding canonical decomposition of that architecture.

Non-Gaussian Weights: According to the strategy used in the proofs, we need the individual weights to be distributed such that the product of two independent scalar weights (as in Equation (A.20)) remain sub-exponential. Hence, any sub-gaussian initialization method, such as any bounded initialization (*e.g.* truncated normal or uniform on an interval) can be used, and the same proof structure would support the same convergence rate, albeit with different constants in convergence (independent of n).

Non-ReLU activations: In general, the proofs rely on the ReLU activation

through Lemma A.7.12, which gives a concentration bound on the absolute value of the dot product of post-activations of each layer of the NN. To use other nonlinearities, we would only need an analagous result for that nonlinearity; the other proofs follow without requiring any other significant change.

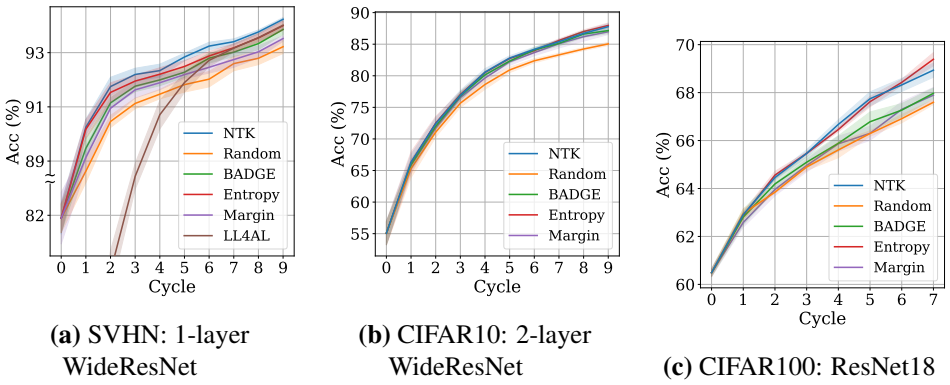
Experimental Evaluation: To provide further experimental support for this argument, we have conducted an ablation study on the FCN architecture with different nonlinearities and with truncated Gaussian initialization (Figures 3.3, A.5, A.7 and A.8). As seen in the provided figures, the impact of nonlinearity and initialization method as long as they follow the provided setting in Setting B, is marginal.

A.8 More Details on Kernel Regression Using pNTK on Full CIFAR-10 Dataset

In this section we provide another figure comparing the accuracy of $\hat{f}^{lin}(x)$ with parameters derived at epoch $E \in \{0, 50, 100, 150, 200\}$ of training the NN with SGD. On the y-axis, the reported number is $f^{lin}(x) - f^*(x)$ where f^* denotes the final model obtained after training f for 200 epochs. As seen in Figure A.9 the architecture of the model has a significant impact on how good the linearization predicts the final accuracy of the fully-trained model. However, as proven in Theorem 3.3.1 in conjunction with the linearization approximations provided in Lee et al. [36], as width grows, this approximation becomes more accurate. One unexplored fact regarding this experiment is that fact that lineraization with trained parameters significantly outperforms linearization at initialization, which is intuitive but not rigorously investigated yet.

A.9 Experimental Evaluation: Tightness of Bounds

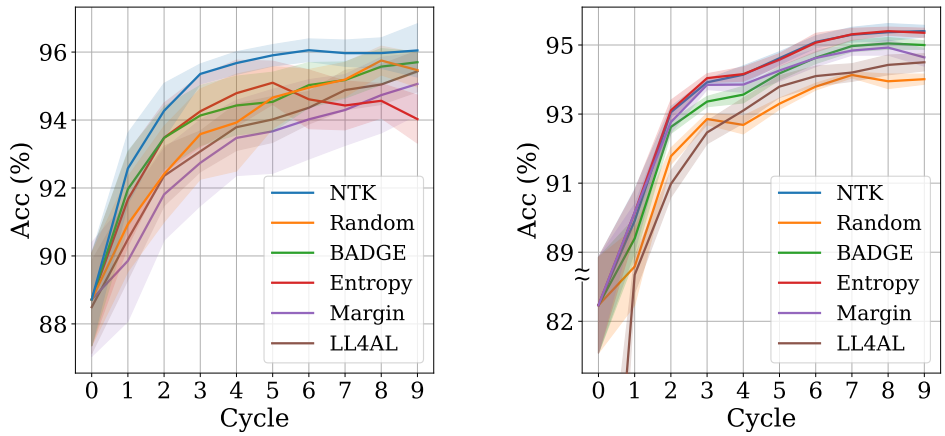
In this section, we present experimental evaluations that analyze the tightness of the approximation bound. The results are presented for the fully connected network used in the experiment. Results are available in Figure A.10.



(a) SVHN: 1-layer WideResNet

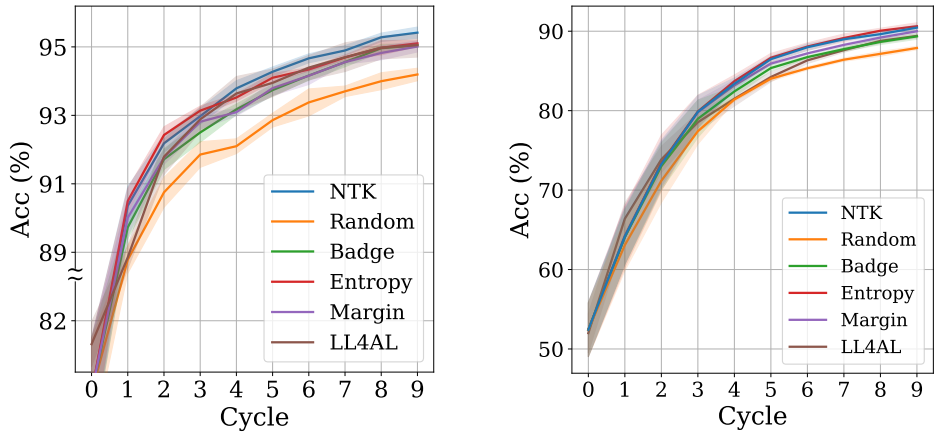
(b) CIFAR10: 2-layer WideResNet

(c) CIFAR100: ResNet18



(d) MNIST: 1-layer WideResNet

(e) SVHN: 2-layer WideResNet



(f) SVHN: ResNet18

(g) CIFAR10: ResNet18

Figure A.4: Comparison of the-state-of-the-art active learning methods on various benchmark datasets. Vertical axis shows attained accuracy of each acquisition method.

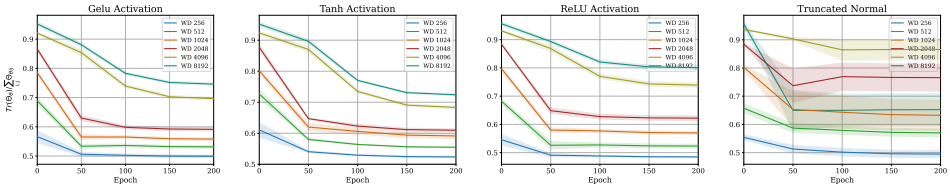


Figure A.5: Comparing the **magnitude of sum of on-diagonal and off-diagonal elements of** Θ_θ at initialization and throughout training, based on 1000 points from CIFAR-10. The reported numbers are the average of 1000×1000 kernels each having a shape of 10×10. The same subset has then been used to train the NN using SGD.

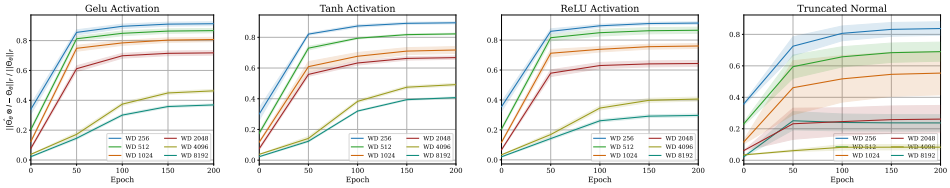


Figure A.6: Evaluating the **relative difference of Frobenius norm of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D}) \otimes I_O$ at initialization and throughout training, based on 1000 points from CIFAR-10.

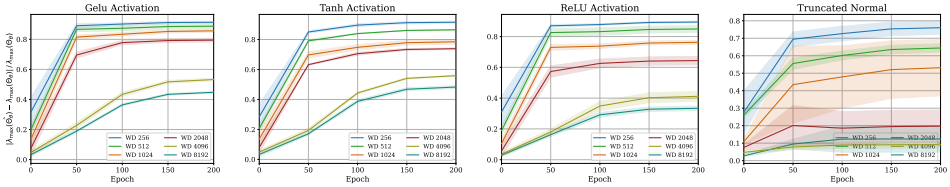


Figure A.7: Evaluating the **relative difference of** $\lambda_{\max}$ **of** $\Theta_\theta(\mathcal{D}, \mathcal{D})$ **and** $\hat{\Theta}_\theta(\mathcal{D}, \mathcal{D})$ at initialization and throughout training, based on kernels on a subset ($|\mathcal{D}| = 1000$) of points from CIFAR-10.

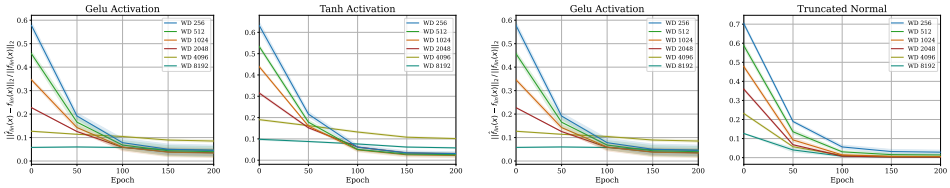


Figure A.8: Evaluating the **relative norm difference of kernel regression outputs using eNTK and pNTK as in Equation (3.4) and Equation (3.5)** at initialization and throughout training. The kernel regression has been done on $|\mathcal{D}| = 1000$ training points and $|\mathcal{X}| = 500$ test points randomly selected from CIFAR-10's train and test sets.

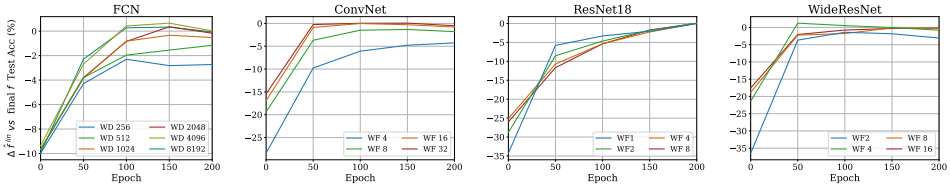


Figure A.9: Evaluating the **difference in test accuracy of kernel regression using pNTK as in** (3.5) **vs the final model** f throughout SGD training on the full CIFAR-10 dataset. How much worse would it be to "give up" on SGD at this point and train $\hat{f}^{lin}$ with the current representation?

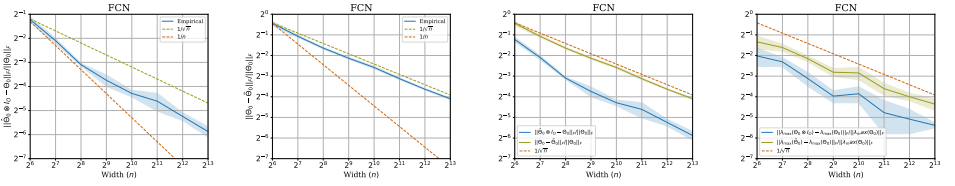


Figure A.10: Experimental evaluation of tightness of approximation bounds