Lecture 6: NTK Origin and Derivation

Adityanarayanan Radhakrishnan

Edited by: Max Ruiz Luyten, George Stefanakis, Cathy Cai

September 25, 2024

1 Introduction

Thus far, we established conditions under which training the last layer of an infinitely wide, 1-hidden layer neural network is equivalent to solving kernel regression with the Neural Network Gaussian Process (NNGP). We now present the result that training all layers of a neural network is equivalent to solving kernel regression with a kernel referred to as the Neural Tangent Kernel (NTK). We begin by introducing the idea of approximating a neural network via linearization around initial weights, and we show that training the corresponding linearization is equivalent to solving kernel regression with the NTK. We then utilize the theory of dual activations and the computation of the NNGP from the previous lecture to compute a closed form for the NTK in terms of dual activation functions. The remarkable aspect of the linearization is that as layer-wise widths approach infinity, training the linearization is equivalent to training all layers of a neural network. As we did for the NNGP, we then lastly present empirical examples highlighting the double descent phenomenon by comparing the performance of networks of increasing width to that of the NTK.

2 Linearization of Neural Networks

Following the notation from previous lectures, let $X = [x^{(1)}, x^{(2)}, \ldots, x^{(n)}] \in \mathbb{R}^{d\times n}$ denote a training samples and let $y = [y^{(1)}, y^{(2)}, \ldots, y^{(n)}] \in \mathbb{R}^{1\times n}$ denote training labels. Instead of using linear or kernel regression to fit the given training data, consider instead using a neural network implementing a function $f(x) : \mathbb{R}^d \to \mathbb{R}$. In the case where our neural network has one hidden layer, the network is traditionally parameterized by two weight matrices $A \in \mathbb{R}^{1\times k}$, $B \in \mathbb{R}^{k\times d}$ and an elementwise nonlinearity $\phi : \mathbb{R} \to \mathbb{R}$ such that:

$$f(x) = A\phi(Bx)$$

Importantly, the notation above emphasizes that the neural network is implementing a function f that acts on samples, x. While this perspective is useful for implementation in practice, it is not necessarily easily amenable for understanding parameter-dependent training dynamics. Indeed, it is important to remember that the neural network is actually a map on both parameters (A, B) and samples x. Hence, to make this relationship clear, we will instead think of a neural network as a function of both data and parameters, and write:

$$f(w\ ; x) = A\phi(Bx)\ ;$$

where the vector $w \in \mathbb{R}^{k+kd}$ is a concatentation of all parameters in the neural network. Namely,

$$w = \begin{bmatrix} A_{11} & A_{12} & \ldots & A_{1k} & B_{11} & B_{12} & \ldots & B_{kd} \end{bmatrix}^T \in \mathbb{R}^{k+kd}$$

While the difference in notation seems pedantic thus far, it importantly allows for an approximation of neural networks via Taylor series around initial weight values $w^{(0)}$. In particular, we will consider fixing the sample x and linearizing (i.e. performing a first order Taylor series approximation) a neural network around these initial weights.

Definition 1 (Linearization of Neural Network)**.** *Let $f_x(w) : \mathbb{R}^p \to \mathbb{R}$ denote a neural network operating on a fixed sample $x \in \mathbb{R}^d$. Then, the **linearization** of $f_x(w)$ around initial weights $w^{(0)}$ is given by:*

$$\tilde{f}_x(w) = f(w^{(0)}) + \nabla f_x(w^{(0)})^T (w - w^{(0)}).$$

Importantly, note that the linearization above is indeed equivalent to applying a linear model on top of transformed features, which is consistent with the prior frameworks studied throughout this course. Namely, training the linearization $\tilde{f}_x(w)$ using the MSE loss involves solving a linear regression problem after applying the feature map $\psi : \mathbb{R}^d \to \mathbb{R}^p$ with $\psi(x) = \nabla f_x(w^{(0)})$ to the samples. Note that even though the number of neural network parameters p can be extremely large, we already have the machinery to solve linear regression efficiently via the kernel trick. The corresponding kernel defined below is referred to as the Neural Tangent Kernel (NTK) [2].

Definition 2 (Neural Tangent Kernel)**.** *Let $f_x(w) : \mathbb{R}^p \to \mathbb{R}$ denote a neural network with initial parameters $w^{(0)}$. The **neural tangent kernel**, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, is given by:*

$$K(x, \tilde{x}) = \langle \nabla f_x(w^{(0)}), \nabla f_{\tilde{x}}(w^{(0)}) \rangle$$

To make this kernel concrete for a finite width network, we provide the following simple example below.

Example 1. *For $x \in \mathbb{R}$, let $f_x(w) : \mathbb{R}^4 \to \mathbb{R}$ denote a neural network parameterized as follows:*

$$f_x(w) = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \phi\left(\begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} x \right) = A_{11}\phi(B_{11}x) + A_{12}\phi(B_{21}x) \ ;$$

where $w = \begin{bmatrix} A_{11} & A_{12} & B_{11} & B_{21} \end{bmatrix}^T$. Then, for $x, \tilde{x} \in \mathbb{R}$:

$$K(x, \tilde{x}) = \left\langle \begin{bmatrix} \phi(B_{11}x) \\ \phi(B_{21}x) \\ A_{11}x\phi'(B_{11}x) \\ A_{12}x\phi'(B_{21}x) \end{bmatrix}, \begin{bmatrix} \phi(B_{11}\tilde{x}) \\ \phi(B_{21}\tilde{x}) \\ A_{11}\tilde{x}\phi'(B_{11}\tilde{x}) \\ A_{12}\tilde{x}\phi'(B_{21}\tilde{x}) \end{bmatrix} \right\rangle = \sum_{i=1}^{2} \phi(B_{i1}x)\phi(B_{i1}\tilde{x}) + \sum_{i=1}^{2} A_{1i}^2 \phi'(B_{i1}x)\phi'(B_{i1}\tilde{x})x\tilde{x}$$

The term in red in the example above is similar to that appearing in the computation of the NNGP. Indeed, as we will show in the next section, the NTK can be written as the sum of the NNGP plus a correction term, appearing in blue above, which intuitively accounts for training more than just the last layer.

Remarks on Training a Linearized Network. Note that there is a slight subtlety in using kernel regression with the NTK, as compared to training the linearization $\tilde{f}_x(w)$ directly. In particular, assuming $w = w^{(0)} + \tilde{w}$, we write the MSE for training $\tilde{f}_x(w)$ as follows:

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \tilde{f}_{x^{(i)}}(w))^2 \iff \mathcal{L}(\tilde{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - f_{x^{(i)}}(w^{(0)}) - \nabla f_{x^{(i)}}(w^{(0)})^T \tilde{w})^2$$

Hence, to match training the linearization exactly, we must use the NTK to map from $x^{(i)}$ to $y^{(i)} - f_{x^{(i)}}(w^{(0)})$ (e.g. mapping from the original samples to corrected labels). As will be shown in the exercises, if we want to numerically match the predictions of a trained neural network with the NTK, we will need to account for this difference. In practice, however, it is easier to ignore the correction term and just set $f_{x^{(i)}}(w^{(0)}) = 0$ for all $x^{(i)}$.

3 Infinite Width NTK Derivation

In the example above, we computed the NTK for a finite width network. Naturally, as we did for the NNGP, we can analyze the case when network depth k approaches infinity and under certain conditions on the initial weights $w^{(0)}$, we can write a closed form for the NTK.

Theorem 1. *Let $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$ and $\phi : \mathbb{R} \to \mathbb{R}$ an element-wise activation function.*[1] *For $x \in \mathcal{S}^{d-1}$, let $f_x(w) : \mathbb{R}^{kd+k} \to \mathbb{R}$ denote 1 hidden layer fully connected network with $f_x(w) = \frac{1}{\sqrt{k}} A\phi(Bx)$. If $A_{1i}, B_{ij} \overset{i.i.d}{\sim} \mathcal{N}(0,1)$, then as $k \to \infty$, the NTK is given by:*

$$K(x,\tilde{x}) = \check{\phi}(x^T\tilde{x}) + \check{\phi}'(x^T\tilde{x})x^T\tilde{x} \ ;$$

where $\check{\phi} : [-1,1] \to \mathbb{R}$ is the dual activation for ϕ for $x, \tilde{x} \in \mathcal{S}^{d-1}$.

Proof. As in Example 1, we denote:

$$w = \begin{bmatrix} A_{11} & A_{12} \dots & A_{1k} & B_{11} & B_{21} \dots & B_{kd} \end{bmatrix}^T \ ;$$

To compute the NTK, we need to compute $\nabla f_x(w)$, which requires computing $\frac{\partial f_x(w)}{\partial A_{1i}}$ and $\frac{\partial f_x(w)}{\partial B_{ij}}$ for $i \in [k], j \in [d]$. We perform these calculations directly. Namely, we first write $f_x(w)$ as:

$$f_x(w) = \frac{1}{\sqrt{k}} \sum_{i=1}^{k} A_{1,i}\phi(B_{i,:}x)$$

Then, we have:

$$\frac{\partial f_x(w)}{\partial A_{1i}} = \frac{1}{\sqrt{k}}\phi(B_{i,:}x) \ \ ; \ \ \frac{\partial f_x(w)}{\partial B_{ij}} = \frac{1}{\sqrt{k}} A_{1i}\phi'(B_{i,:}x)x_j$$

Lastly, the kernel is given by:

$$K(x,\tilde{x}) = \langle f_x(w^{(0)}), f_{\tilde{x}}(w^{(0)})\rangle = \frac{1}{k}\sum_{i=1}^{k} \phi(B_{i,:}x)\phi(B_{i,:}\tilde{x}) + \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{d} A_{1i}^2\phi'(B_{i,:}x)\phi'(B_{i,:}\tilde{x})x_j\tilde{x}_j$$

Now, it is evident that the terms in red and blue above can be evaluated using the law of large numbers (indeed, this is one reason for choosing the scaling factor of $\frac{1}{\sqrt{k}}$). Moreover, we observe that the term in red above is precisely the NNGP (see Definition 1 of Lecture 4). Additionally, we recall from Corollary 1 of Lecture 4, that if the data is restricted to the unit sphere, the NNGP of a network with activation ϕ is given by the dual activation $\check{\phi}$. Next, since A_{1i} are independent of B_{ij} and $\mathbb{E}_{A_{1i}}[A_{1i}^2] = 1$, we conclude that in probability:

$$\begin{aligned}\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k}\sum_{j=1}^{d} A_{1i}^2\phi'(B_{i,:}x)\phi'(B_{i,:}\tilde{x})x_j\tilde{x}_j &= \lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k} A_{1i}^2\phi'(B_{i,:}x)\phi'(B_{i,:}\tilde{x})x^T\tilde{x} \\ &= x^T\tilde{x}\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}\phi'(B_{i,:}x)\phi'(B_{i,:}\tilde{x})\end{aligned}$$

Lastly, we see that the term in blue above is the NNGP of a 1 hidden layer network with activation ϕ'. Thus, provided the data is on the sphere this term is given by the dual of ϕ', $(\check{\phi}')$. From Corollary 2 of Lecture 4, we additionally observe that differentiation and the dual operator commute and so, this term is given by $(\check{\phi})'$. Hence, we conclude:

$$K(x,\tilde{x}) = \check{\phi}(x^T\tilde{x}) + \check{\phi}'(x^T\tilde{x})x^T\tilde{x}$$

□

Remarks. Note that the derivation of a closed form for the NTK is simplified drastically by utilizing the derivation of the NNGP. Namely, we observe that the NTK is a sum of two terms involving the NNGP for

[1] Formally, we require $\phi \in L^2(\mu)$, where μ is the Gaussian measure.

networks with activations ϕ and ϕ'. Notably, as we will show in the next lecture, this technique of using the NNGP will be useful in writing a simple recurrence for the NTK of a deep neural network. When the data are not normalized to be on the sphere, we can still write a closed form for the recursion, but it will contain terms involving $\|x\|_2^2$ and $\|\tilde{x}\|_2^2$. A simple example for ReLU networks is presented in the homework.

NTK for Networks with Multidimensional Outputs. Above, we presented a derivation of the NTK for neural networks implementing functions $f : \mathbb{R}^d \to \mathbb{R}$. On the other hand, if we are interested in computing the NTK for neural networks implementing functions $f : \mathbb{R}^d \to \mathbb{R}^c$, remarkably, the kernel matrix remains the same. Thus, even in the multidimensional case, we simply compute the NTK assuming the network has 1 output, and then solve kernel regression with the labels $y \in \mathbb{R}^{c \times n}$. This case is analyzed in detail in the homework, and importantly, this property does not hold for neural networks with arbitrary layer structure, e.g. fully convolutional networks.

We now provide an example to demonstrate that the NTK is easy to compute given the NNGP.

Example 2 (ReLU NTK from NNGP)**.** *Let $\phi(z) = \sqrt{2}\max(0, z)$ for $z \in \mathbb{R}$ be an element-wise activation. If $x, \tilde{x} \in \mathcal{S}^{d-1}$ and $\xi = x^T\tilde{x}$, then the NNGP is given by*

$$\Sigma(x, \tilde{x}) = \check{\phi}(\xi) = \frac{1}{\pi}\left(\xi\left(\pi - \arccos\left(\xi\right)\right) + \sqrt{1-\xi^2}\right)$$

Now by differentiating $\check{\phi}$ with respect to ξ, we conclude:

$$\check{\phi}'(\xi) = \frac{1}{\pi}(\pi - \arccos(\xi))$$

Thus, the NTK is given by:

$$K(x, \tilde{x}) = \check{\phi}(\xi) + \xi\check{\phi}'(\xi) = \frac{1}{\pi}\left(\xi\left(\pi - \arccos\left(\xi\right)\right) + \sqrt{1-\xi^2}\right) + \xi\frac{1}{\pi}(\pi - \arccos(\xi))$$

4 Equivalence between NTK and Training Neural Networks

Thus far, we have demonstrated that the NTK arises as the kernel corresponding to training a linearization of a neural network around initial weights. Remarkably, as layer-wise widths approach infinity, solving kernel regression with the NTK is remarkably equivalent to training all layers of a neural network [4, 5].

> **A note to the reader:** While a full proof of the equivalence between kernel regression with the NTK and training infinitely wide neural networks is out of scope for this course, the goal of this section is to provide intuition around why such an equivalence holds. For a rigorous proof of this equivalence, we refer the reader to the works [4, 5].

The key idea behind the proof is that as layer-wise width approaches infinity, the Hessian of the neural network approaches 0, while the gradient (and thus the NTK) remains constant. Hence, the neural network is well approximated by a linear model. We outline at a high level the important steps in the proof of this equivalence below.

1. As layer-wise width approaches infinity, the Hessian of the neural network with respect to its weights approaches 0, while the NTK remains constant.
2. If the norm of the Hessian is uniformly small (i.e. $O(\epsilon)$) in a ball of fixed radius, R, then the change of the NTK in this ball is $O(\epsilon R)$.
3. As layer-wise width approaches infinity, gradient descent used to train a neural network converges linearly to a solution inside a ball of fixed radius R (see Theorem 4 of [5]).

To provide intuition for why the Hessian approaches 0 as width increases, we establish point (1) above for the case of a 1 hidden layer network. We refer the reader to Proposition 2.3 of [4] for a proof of (2), and to Theorem 4 of [5] for a proof of (3).

Proposition 1. *Let $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^k$, let $\phi : \mathbb{R} \to \mathbb{R}$ denote a twice differentiable elementwise activation function. Let $f_x(w) = \frac{1}{\sqrt{k}} A\phi(Bx)$ denote a 1 hidden layer fully connected neural network. Then assuming A_{1i}, B_{ij} are bounded, for any $x, \tilde{x} \in [-m, m]$ for $m \in \mathbb{R}$:*

$$|K(x, \tilde{x})| = O(1) \quad ; \quad \|H(f_x(w))\|_2 = O\left(\frac{1}{\sqrt{k}}\right)$$

Proof. The proof follows directly from the computation of the gradients $\nabla f_x(w)$ and the Hessian $H(f_x(w))$. In particular, we have:

$$\frac{\partial f}{\partial A_{1i}} = \frac{1}{\sqrt{k}}\phi(B_i x) \quad ; \quad \frac{\partial f}{\partial B_i} = \frac{1}{\sqrt{k}} A_{1i}\phi'(B_i x)x \quad ;$$

and similarly,

$$\frac{\partial^2 f}{\partial A_{1i}\partial A_{1j}} = 0 \quad ; \quad \frac{\partial^2 f}{\partial A_{1i}\partial B_i} = \frac{1}{\sqrt{k}}\phi'(B_i x)x \quad ; \quad \frac{\partial^2 f}{\partial B_i^2} = \frac{1}{\sqrt{k}} A_{1i}\phi''(B_i x)x^2.$$

Hence, the NTK is given by:

$$K(x, \tilde{x}) = \frac{1}{k}\sum_{i=1}^{k} \phi(B_i x)\phi(B_i \tilde{x}) + x\tilde{x}\frac{1}{k}\sum_{i=1}^{k} A_{1i}^2 \phi'(B_i x)\phi'(B_i \tilde{x}) = O(1)$$

On the other hand, the Hessian is sparse, and has the following form:

$$H(f_x(w)) = \begin{bmatrix} 0 & 0 & \dots & 0 & \phi'(B_1 x)x & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \phi'(B_2 x)x & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & \phi'(B_k x)x \\ \phi'(B_1 x)x & 0 & \dots & 0 & A_{11}\phi''(B_1 x)x^2 & 0 & \dots & 0 \\ 0 & \phi'(B_2 x)x & \dots & 0 & 0 & A_{12}\phi''(B_2 x)x^2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \phi'(B_k x)x & 0 & 0 & \dots & A_{1k}\phi''(B_k x)x^2 \end{bmatrix}$$

Via direct computation (See Appendix A), the spectral norm of $H(f_x(w))$ is given by:

$$\|H(f_w(x))\|_2 = \frac{1}{\sqrt{k}} \max_{i\in[k]} \left| \frac{A_{1i}\phi''(B_i x)x^2 + \sqrt{A_{1i}^2\phi''(B_i x)^2 x^4 + 4\phi'(B_i x)^2 x^2}}{2} \right| = \frac{C}{\sqrt{k}} = O\left(\frac{1}{\sqrt{k}}\right)$$

□

Remarks. Note that in the above computation, the Hessian was sparse, and the spectral norm of the Hessian was thus essentially controlled by the infinity norm over the gradient $\nabla f_x(w)$. On the other hand, the order of K is governed by the 2-norm of the gradient $\nabla f_x(w)$. This discrepancy in norms leads to the difference in scaling between the gradient and Hessian leading to the neural network being well approximated by a linear model. Lastly, we note that neural networks with non-linear activation on the output layer will not be well approximated by their linearization. This follows from the fact that the Hessian will contain an additive term from the chain rule, which does not vanish with increasing width (See Section 4 of [4]).

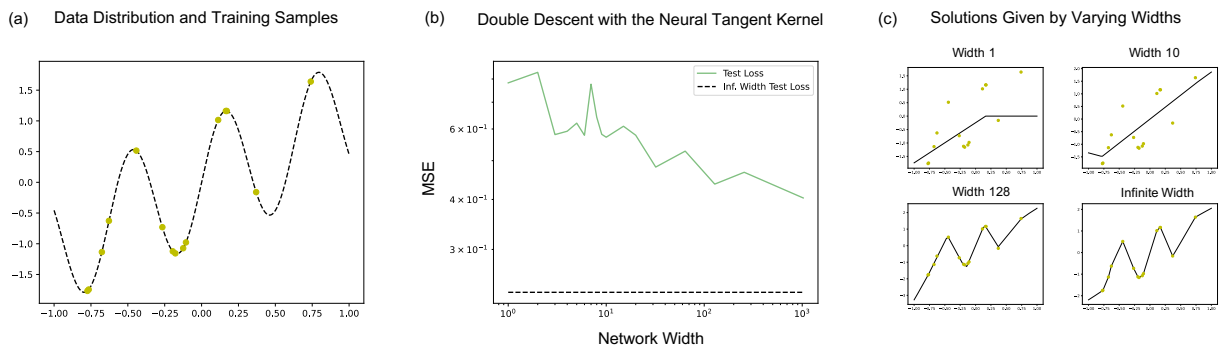

Figure 1: A demonstration of the benefit of using over-parameterized neural networks and the NTK in regression. (a) A visualization of the data distribution given by the function $g(x) = x + \sin(10x)$; the training samples are shown in yellow. (b) Increasing network width leads to lower test loss, when using 1 hidden layer ReLU networks, and the model with lowest test loss corresponds to the NTK. (c) A visualization of the functions learned by networks of varying width. We see that networks of width 1 and 10 are under-parameterized and are unable to fit the data exactly, while the network of width 128 and the NTK interpolate the data, yet generalize well to test data.

5 Empirical Demonstration

Just as we did for the NNGP, we compare the performance of finite width neural networks with that of the NTK. Unlike the case for the NNGP, we now compare training all layers of a neural network with the performance of the NTK. In particular, in the example below, we again demonstrate the double descent curve, which indicates that larger (e.g. wider) neural networks have lower test error and that the NTK (corresponding to an infinite width network) has the lowest test error.

Example 3. *Let $X \in \mathbb{R}^{1\times 15}$ denote 15 samples of real numbers drawn from the uniform distribution on the interval from $[-1, 1]$ (denoted $U[-1, 1]$). Let $g(x) : \mathbb{R} \to \mathbb{R}$ such that $g(x) = x + \sin(10x)$ and let $y = f(X) \in \mathbb{R}^{1\times 15}$ such that $y_i = g(X_i)$. Given these 15 samples, the goal is to recover $g(x)$.*

In particular, we consider the class of 1 hidden layer neural networks of width k given by:

$$f_k(x) = \frac{\sqrt{2}}{\sqrt{k}} A\phi(Bx) \quad ;$$

where $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$, and ϕ is the ReLU function (i.e. $\phi(z) = \max(0, z)$). We then use gradient descent with a learning rate of 10^{-2} to minimize the MSE

$$\mathcal{L}(A, B) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_k(X_i))^2 \quad ,$$

for varying widths k. For $k \geq 15$, we simply train until the MSE is less than $2 \cdot 10^{-3}$ (e.g. close to 0) and for $k < 15$, we use a patience strategy [1] with a max patience of 50.[2]

In Fig. 1, we visualize the test error of the models f_k on 1000 points sampled evenly from $[-1, 1]$ for $k \in [10] \cup \{15, 20, 32, 64, 128, 256, 512, 1024\}$. We observe that the test error again follows the double descent model. In particular, the test loss decreases for models of width $k > 15$ and the test error initially decreases and then increases for $k < 15$.

Lastly, we compare the performance of the finite width networks to that of the NTK corresponding to the case where k goes to infinity. In particular, we observe that the NTK provides the lowest test error. Note that we are not using the empirical NTK (given by computing inner products of the gradients of f_k at each data point),

[2]When using a patience strategy, one keeps track of the number of times t that the loss did not decrease. If t is larger than the max patience, training concludes.

and importantly, we are not correcting the labels via the predictions of the neural network at initialization. This clarifies why the performance of the finite width network is not limiting to the performance of the NTK used here.

Remarks. While the NTK achieved the lowest test error in the example above, this need not always be the case (as is shown in [3] for the convolutional NTK). Nevertheless, a recurring theme of this course is that the NTK serves as a simple, fast, and strong baseline for many machine learning tasks.

6 Discussion

Using the tools developed in prior lectures, we established the equivalence between training infinitely wide neural networks and solving kernel regression with the NTK. We presented a closed form for the NTK of a 1 hidden layer fully connected network and showed that the NTK can be written as the sum of the NNGP and a correction term. Hence, knowing the NNGP for a given activation easily allows for computation of the NTK. As we did for the NNGP, we lastly showcased the double descent phenomenon by comparing the generalization of the NTK and finite width fully connected networks.

While the NTK of a 1 hidden layer fully connected network is useful for establishing a simple nonlinear baseline on vectorized data, we need to compute the NTK corresponding to deep networks or convolutional networks to match the effectiveness of neural networks on general datasets (e.g. image datasets). In the following lecture, we first show that the computations of the NTK for 1 layer are easily extended to give a formula for a fully connected network with depth L. We then lastly will demonstrate how to compute the NTK for a convolutional network (the CNTK).

References

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, volume 1. MIT Press, 2016.

[2] A. Jacot, F. Gabriel, and C. Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

[3] J. Lee, S. S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Shol-Dickstein. Finite Versus Infinite Neural Networks: an Empirical Study. In *Advances in Neural Information Processing Systems*, 2020.

[4] C. Liu, L. Zhu, and M. Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. In *Neural Information Processing Systems*, 2020.

[5] C. Liu, L. Zhu, and M. Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv:2003.00307*, 2020.

A Hessian Eigenvalue Computation

Below, we review the computation used to compute the eigenvalues of the Hessian of a 1 hidden layer neural network. Recall that the Hessian is given as follows:

$$H(f_x(w)) = \begin{bmatrix} 0 & 0 & \dots & 0 & \phi'(B_1x)x & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \phi'(B_2x)x & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & \phi'(B_kx)x \\ \phi'(B_1x)x & 0 & \dots & 0 & A_{11}\phi''(B_1x)x^2 & 0 & \dots & 0 \\ 0 & \phi'(B_2x)x & \dots & 0 & 0 & A_{12}\phi''(B_2x)x^2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \phi'(B_kx)x & 0 & 0 & \dots & A_{1k}\phi''(B_kx)x^2 \end{bmatrix}$$

To compute the eigenvalues of this Hessian, we first note that the Hessian has sparse structure and in particular, each row has at most 2 entries. Hence, we first check whether vectors that share a similar sparsity pattern are eigenvectors. In particular, consider the vector:

$$v = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} ;$$

where v has two nonzero entries $(1, \alpha)$ in the first coordinate and the $k+1$ coordinate. Suppose that v is an

eigenvector with eigenvalue λ, then we must have:

$$H(f_x(w))v = \begin{bmatrix} 0 & 0 & \dots & 0 & \phi'(B_1x)x & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \phi'(B_2x)x & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & \phi'(B_kx)x \\ \phi'(B_1x)x & 0 & \dots & 0 & A_{11}\phi''(B_1x)x^2 & 0 & \dots & 0 \\ 0 & \phi'(B_2x)x & \dots & 0 & 0 & A_{12}\phi''(B_2x)x^2 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \phi'(B_kx)x & 0 & 0 & \dots & A_{1k}\phi''(B_kx)x^2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} \alpha\phi'(B_1x)x \\ 0 \\ \vdots \\ 0 \\ \phi'(B_1x)x + \alpha A_1 1\phi''(B_1x)x^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Hence, as $H(f_x(w))v = \lambda v$, we conclude:

$$\alpha\phi'(B_1x)x = \lambda$$
$$\phi'(B_1x)x + \alpha A_{11}\phi''(B_1x)x^2 = \lambda\alpha$$

The above is a system of two equations with two unknowns (α, λ). We thus solve the above system of equations for λ to conclude:

$$\lambda = \frac{A_{11}\phi''(B_1x)x^2 \pm \sqrt{A_{11}^2\phi''(B_1x)^2x^4 + 4\phi'(B_1x)^2x^2}}{2}$$

By shifting the nonzero entries of v to positions i and $k+i$ for $i \in [k]$, we recover $2k$ eigenvalues. As the Hessian is a matrix with $2k$ rows, we thus recover all eigenvalues and conclude that pairs of eigenvalues are given by:

$$\lambda_i, \lambda_{k+i} = \frac{A_{1i}\phi''(B_ix)x^2 \pm \sqrt{A_{1i}^2\phi''(B_ix)^2x^4 + 4\phi'(B_ix)^2x^2}}{2}$$