Deep Learning Theory Talks November 15, 2024

Lecture 2: Neural Tangent Kernel

Yukun Dong

A flurry of papers in theoretical deep learning tackles the common theme of analyzing neural networks in the **infinite-width** limit. At first, this limit may seem impractical and even pointless to study. However, it turns out that neural networks in this regime simplify to linear models with a kernel called the **neural tangent kernel**[JGH20]. Gradient descent is therefore very simple to study.

1 Motivating Examples

1.1 A toy experiment

Let's start with an extremely simple example, and generalize our understanding once we get comfortable with this toy example. Let's work with a 1-D input and 1-D output network. A simple 2-hidden layer relu network with width m will do the trick for now.

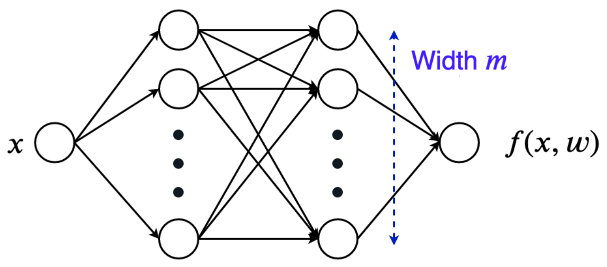

Figure 1: simple 2-hidden layer relu network

We can plot how the network function looks for a bunch of random initializations:

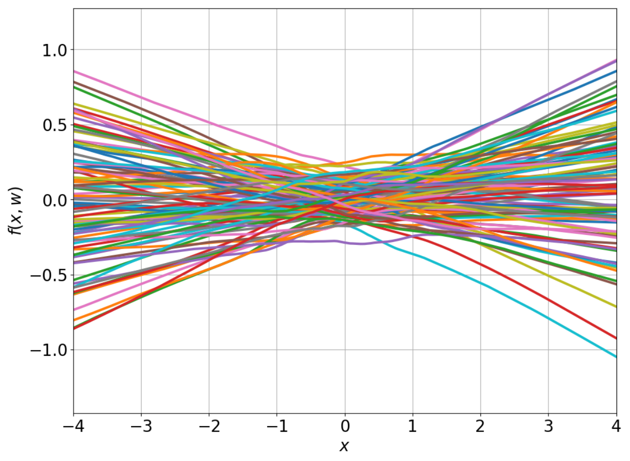

Figure 2: random initializations

Trained with two datapoints, we shall look at the relative change in the norm of the weight vector from initialization:

$$\frac{\|\boldsymbol{w}(n) - \boldsymbol{w}_0\|_2}{\|\boldsymbol{w}_0\|_2}$$

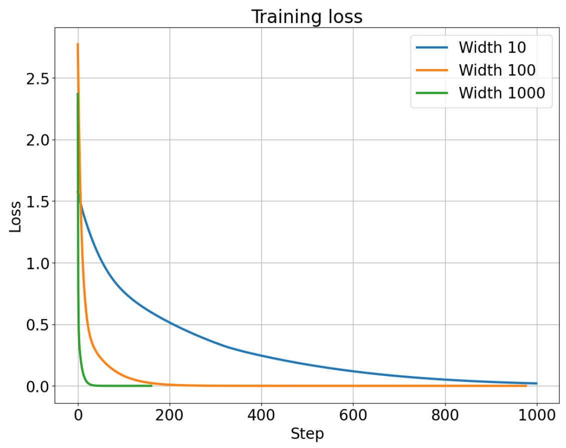

Figure 3: Loss curves

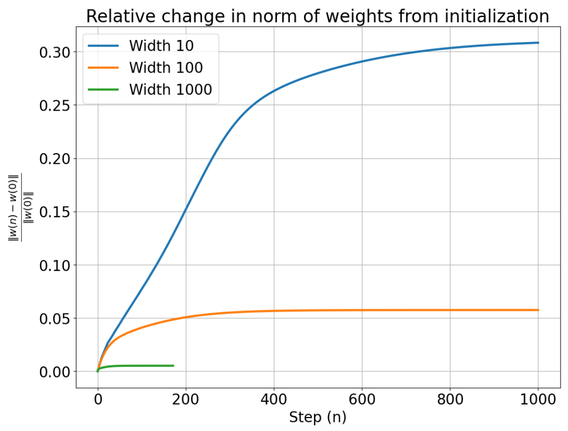

Figure 4: Weights budge through training

The above figures show the loss curves and how much the weights budge as training progress. The weight norms are calculated using all parameters in the model stacked into a single vector. Also, all other hyperparameters (like learning rate) are kept constant. We observe that **the weights don't change much at all for larger hidden widths**, it seems that these models are quite lazy!

1.2 Taylor Expand

Suppose we have overall n datapoints and p parameters, we can just Taylor expand the network function with respect to the weights around its initialization.

$$f(x, \boldsymbol{w}) \approx f(x, \boldsymbol{w_0}) + \nabla_{\boldsymbol{w}} f(x, \boldsymbol{w_0})^T (\boldsymbol{w} - \boldsymbol{w_0})$$

Using our more concise vector notation for the model outputs on a specific dataset we can rewrite as:

$$\boldsymbol{y}(\boldsymbol{w}) \approx y(\boldsymbol{w_0}) + \nabla_{\boldsymbol{w}} \boldsymbol{y}(\boldsymbol{w_0})^T (\boldsymbol{w} - \boldsymbol{w_0})$$

A lot of the things here like the initial output $y(\boldsymbol{w_0})$ and the model Jacobian $\nabla_{\boldsymbol{w}} \boldsymbol{y}(\boldsymbol{w_0})$ are just constants. Focus on the dependence on $\boldsymbol{w}$ – the approximation is a linear model in the weights, so minimizing the least squares loss reduces to just doing linear regression! But, notice that the model function is still non-linear in the input, because finding the gradient of the model is definitely not a linear operation. In fact, this is just a linear model using a feature map $\boldsymbol{\phi}(x)$ which is the gradient vector at initialization:

$$\boldsymbol{\phi}(x) = \nabla_{\boldsymbol{w}} f(x, \boldsymbol{w_0})$$

This feature map naturally induces a kernel on the input, which is called the **Neural Tangent Kernel(NTK)**.

1.3 When is Taylor linear approximation accurate?

We have a smooth loss $F(\boldsymbol{w_0}) \coloneqq R : \mathcal{F} \to \mathbb{R}^+$ and assume that $\boldsymbol{w_0}$ is not a minimizer so that $R(\boldsymbol{f}(\mathbf{x}, \boldsymbol{w_0})) > 0$, and not a critical point so that $\nabla F(\boldsymbol{w_0}) \neq 0$. Consider a gradient descent step $\boldsymbol{w_1} \coloneqq \boldsymbol{w_0} - \eta \nabla F(\boldsymbol{w_0})$, with a small stepsize $\eta > 0$. On the one hand, the relative change of the objective is $\Delta(F) \coloneqq \frac{|F(\boldsymbol{w_1}) - F(\boldsymbol{w_0})|}{F(\boldsymbol{w_0})} \approx \eta \frac{\|\nabla F(\boldsymbol{w_0})\|^2}{F(\boldsymbol{w_0})}$. On the other hand, the relative change of the differential of h measured in operator norm is $\Delta(\nabla \boldsymbol{f}) \coloneqq \frac{\|\nabla \boldsymbol{f}(\boldsymbol{w_1}) - \nabla \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla \boldsymbol{f}(\boldsymbol{w_0})\|} \leq \eta \frac{\|\nabla F(\boldsymbol{w_0})\| \cdot \|\nabla^2 \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla \boldsymbol{f}(\boldsymbol{w_0})\|}$. Lazy training refers to the case where the differential of h does not sensibly change while the loss enjoys a significant decrease, i.e., $\Delta(F) \gg \Delta(\nabla \boldsymbol{f})$. Using the above estimates, this is guaranteed when

$$\frac{\|\nabla F(\boldsymbol{w_0})\|}{F(\boldsymbol{w_0})} \gg \frac{\|\nabla^2 \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla \boldsymbol{f}(\boldsymbol{w_0})\|}.$$

For the square loss $R(y) = \frac{1}{2}\|y - y^\star\|^2$, this leads to the simpler criterion

$$\kappa(\boldsymbol{w_0}) \coloneqq \|\boldsymbol{f}(\boldsymbol{w_0}) - y^\star\| \frac{\|\nabla^2 \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla \boldsymbol{f}(\boldsymbol{w_0})\|^2} \ll 1, \tag{1}$$

using the approximation $\|\nabla F(\boldsymbol{w_0})\| = \|\nabla \boldsymbol{f}(\boldsymbol{w_0})^\intercal (\boldsymbol{f}(\boldsymbol{w_0}) - y^\star)\| \approx \|\nabla \boldsymbol{f}(\boldsymbol{w_0})\| \cdot \|\boldsymbol{f}(\boldsymbol{w_0}) - y^\star\|$. This quantity $\kappa(\boldsymbol{w_0})$ could be called the inverse *relative scale* of the model h at $\boldsymbol{w_0}$. It has been proved in [COB20] that it indeed controls how much the training dynamics differs from the linearized training dynamics when R is the square loss.

Just multiply the output of our model by some factor α(where $\bar{y}$ is the true label):

$$\kappa_\alpha(\boldsymbol{w_0}) = \|(\alpha \boldsymbol{f}(\boldsymbol{w_0}) - \bar{\boldsymbol{y}})\| \frac{\|\nabla_w^2 \alpha \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla_w \alpha \boldsymbol{f}(\boldsymbol{w_0})\|^2}$$

We can get rid of it by assuming that our model always outputs 0 at initialization, i.e. $\boldsymbol{y}(\boldsymbol{w_0}) = 0$. Then we have:

$$\implies \kappa_\alpha(\boldsymbol{w_0}) \sim \frac{\|\nabla_w^2 \alpha \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla_w \alpha \boldsymbol{f}(\boldsymbol{w_0})\|^2} = \frac{\alpha \|\nabla_w^2 \boldsymbol{f}(\boldsymbol{w_0})\|}{\alpha^2 \|\nabla_w \boldsymbol{f}(\boldsymbol{w_0})\|^2} = \frac{1}{\alpha} \frac{\|\nabla_w^2 \boldsymbol{f}(\boldsymbol{w_0})\|}{\|\nabla_w \boldsymbol{f}(\boldsymbol{w_0})\|^2}$$

We can make the model linear (take $\kappa(\boldsymbol{w_0}) \to 0$) by simply jacking up $\alpha \to \infty$! It turns out that $\kappa \to 0$ as the width $m \to \infty$, indicating that scaling the network width to infinity contributes to a linear approximation.

2 Kernel Gradient

2.1 ANN structure

We consider fully-connected ANNs with layers numbered from 0 (input) to L (output), each containing $n_0, \ldots, n_L$ neurons, and with a Lipschitz, twice differentiable nonlinearity function $\sigma : \mathbb{R} \to \mathbb{R}$, with bounded second derivative.

This paper focuses on the ANN *realization function* $F^{(L)} : \mathbb{R}^P \to \mathcal{F}$, mapping parameters θ to functions f_θ in a space $\mathcal{F}$. The dimension of the parameter space is $P = \sum_{\ell=0}^{L-1}(n_\ell + 1)n_{\ell+1}$: the parameters consist of the connection matrices $W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ and bias vectors $b^{(\ell)} \in \mathbb{R}^{n_{\ell+1}}$ for $\ell = 0, ..., L-1$. In our setup, the parameters are initialized as iid Gaussians $\mathcal{N}(0,1)$.

For a fixed distribution p^{in} on the input space $\mathbb{R}^{n_0}$, the function space $\mathcal{F}$ is defined as $\{f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}\}$. On this space, we consider the seminorm $||\cdot||_{p^{in}}$, defined in terms of the bilinear form

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

We assume that the input distribution p^{in} is the empirical distribution on a finite dataset $x_1, ..., x_N$, i.e the sum of Dirac measures $\frac{1}{N}\sum_{i=0}^{N} \delta_{x_i}$. We define the network function by $f_\theta(x) := \tilde{\alpha}^{(L)}(x;\theta)$, where the functions $\tilde{\alpha}^{(\ell)}(\cdot;\theta) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_\ell}$ (called *preactivations*) and $\alpha^{(\ell)}(\cdot;\theta) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_\ell}$ (called *activations*) are defined from the 0-th to the L-th layer by:

$$\begin{aligned}
\alpha^{(0)}(x;\theta) &= x \\
\tilde{\alpha}^{(\ell+1)}(x;\theta) &= \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)} \\
\alpha^{(\ell)}(x;\theta) &= \sigma(\tilde{\alpha}^{(\ell)}(x;\theta)),
\end{aligned}$$

where the nonlinearity σ is applied entrywise. The scalar $\beta > 0$ is a parameter which allows us to tune the influence of the bias on the training.

2.2 The definition of kernel gradient

The training of an ANN consists in optimizing f_θ in the function space $\mathcal{F}$ with respect to a functional cost $C : \mathcal{F} \to \mathbb{R}$, such as a regression or cross-entropy cost. Even for a convex functional cost C, the composite cost $C \circ F^{(L)} : \mathbb{R}^P \to \mathbb{R}$ is in general highly non-convex.

A *multi-dimensional kernel* K is a function $\mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$, which maps any pair (x, x') to an $n_L \times n_L$-matrix such that $K(x, x') = K(x', x)^T$ (equivalently K is a symmetric tensor in $\mathcal{F} \otimes \mathcal{F}$). Such a kernel defines a bilinear map on $\mathcal{F}$, taking the expectation over independent $x, x' \sim p^{in}$:

$$\langle f, g \rangle_K := \mathbb{E}_{x,x' \sim p^{in}} \left[f(x)^T K(x, x') g(x') \right].$$

The kernel K is *positive definite with respect to* $||\cdot||_{p^{in}}$ if $||f||_{p^{in}} > 0 \implies ||f||_K > 0$.

We denote by $\mathcal{F}^*$ the dual of $\mathcal{F}$ with respect to p^{in}, i.e. the set of linear forms $\mu : \mathcal{F} \to \mathbb{R}$ of the form $\mu = \langle d, \cdot \rangle_{p^{in}}$ for some $d \in \mathcal{F}$. Two elements of $\mathcal{F}$ define the same linear form if and only if they are equal on the data. The constructions in the paper do not depend on the element $d \in \mathcal{F}$ chosen in order to represent μ as $\langle d, \cdot \rangle_{p^{in}}$. Using the fact that the partial application of the kernel $K_{i,\cdot}(x, \cdot)$ is a function in $\mathcal{F}$, we can define a map $\Phi_K : \mathcal{F}^* \to \mathcal{F}$ mapping a dual element $\mu = \langle d, \cdot \rangle_{p^{in}}$ to the function $f_\mu = \Phi_K(\mu)$ with values:

$$f_{\mu,i}(x) = \mu K_{i,\cdot}(x, \cdot) = \langle d, K_{i,\cdot}(x, \cdot) \rangle_{p^{in}}.$$

For our setup, which is that of a finite dataset $x_1, \ldots, x_n \in \mathbb{R}^{n_0}$, the cost functional C only depends on the values of $f \in \mathcal{F}$ at the data points. As a result, the (functional) derivative of the cost C at a point $f_0 \in \mathcal{F}$ can be viewed as an element of $\mathcal{F}^*$, which we write $\partial_f^{in} C|_{f_0}$. We denote by $d|_{f_0} \in \mathcal{F}$, a corresponding dual element, such that $\partial_f^{in} C|_{f_0} = \langle d|_{f_0}, \cdot \rangle_{p^{in}}$.

The *kernel gradient* $\nabla_K C|_{f_0} \in \mathcal{F}$ is defined as $\Phi_K\left(\partial_f^{in} C|_{f_0}\right)$. In contrast to $\partial_f^{in} C$ which is only defined on the dataset, the kernel gradient generalizes to values x outside the dataset thanks to the kernel K:

$$\begin{aligned}
\nabla_K C|_{f_0}(x) &= \Phi_K\left(\partial_f^{in} C|_{f_0}\right)(x) && (2)\\
&= \begin{pmatrix} \partial_f^{in} C|_{f_0}(K_{1,\cdot}(x,\cdot)) \\ \vdots \\ \partial_f^{in} C|_{f_0}(K_{n^L,\cdot}(x,\cdot)) \end{pmatrix} && (3)\\
&= \begin{pmatrix} \langle d|_{f_0}, K_{1,\cdot}(x,\cdot)\rangle \\ \vdots \\ \langle d|_{f_0}, K_{n^L,\cdot}(x,\cdot)\rangle \end{pmatrix} && (4)\\
&= \begin{pmatrix} \frac{1}{N}\sum_{j=1}^{N} K_{1,\cdot}(x,x_j) d|_{f_0}(x_j) \\ \vdots \\ \frac{1}{N}\sum_{j=1}^{N} K_{n^L,\cdot}(x,x_j) d|_{f_0}(x_j) \end{pmatrix} && (5)\\
&= \frac{1}{N}\sum_{j=1}^{N} K(x,x_j) d|_{f_0}(x_j). && (6)
\end{aligned}$$

A time-dependent function $f(t)$ follows the *kernel gradient descent with respect to* K if it satisfies the differential equation

$$\partial_t f(t) = -\nabla_K C|_{f(t)}.$$

During kernel gradient descent, the cost $C(f(t))$ evolves as

$$\partial_t C|_{f(t)} = -\left\langle d|_{f(t)}, \nabla_K C|_{f(t)}\right\rangle_{p^{in}} = -\left\|d|_{f(t)}\right\|_K^2.$$

Convergence to a critical point of C is hence guaranteed if the kernel K is positive definite with respect to $||\cdot||_{p^{in}}$: the cost is then strictly decreasing except at points such that $||d|_{f(t)}||_{p^{in}} = 0$. If the cost is convex and bounded from below, the function $f(t)$ therefore converges to a global minimum as $t \to \infty$.

3 Neural Tangent Kernel Main Results

NTK is central to describe the generalization features of ANNs. While the NTK is random at initialization and varies during training, in the infinite-width limit it converges to an explicit limiting kernel and it stays constant during training. During training, the network function f_θ follows a descent along the kernel gradient with respect to the NTK. This makes it possible to study the training of ANNs in the function space $\mathcal{F}$, on which the cost C is convex.

The first key result of NTK is the following: in the same limit, the Neural Tangent Kernel (NTK) converges in probability to an explicit deterministic limit.

Theorem 1. *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as the layers width $n_1, ..., n_{L-1} \to \infty$, the NTK $\Theta^{(L)}$ converges in probability to a deterministic limiting kernel:*

$$\Theta^{(L)} \to \Theta^{(L)}_\infty \otimes Id_{n_L}.$$

The scalar kernel $\Theta^{(L)}_\infty : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined recursively by

$$\begin{aligned} \Theta^{(1)}_\infty(x, x') &= \Sigma^{(1)}(x, x') \\ \Theta^{(L+1)}_\infty(x, x') &= \Theta^{(L)}_\infty(x, x')\dot{\Sigma}^{(L+1)}(x, x') + \Sigma^{(L+1)}(x, x'), \end{aligned}$$

where

$$\dot{\Sigma}^{(L+1)}(x, x') = \mathbb{E}_{f \sim \mathcal{N}\left(0, \Sigma^{(L)}\right)}\left[\dot{\sigma}\left(f\left(x\right)\right)\dot{\sigma}\left(f\left(x'\right)\right)\right],$$

taking the expectation with respect to a centered Gaussian process f of covariance $\Sigma^{(L)}$, and where $\dot{\sigma}$ denotes the derivative of σ.

The second key result is that the NTK stays asymptotically constant during training.

Theorem 2. *Assume that σ is a Lipschitz, twice differentiable nonlinearity function, with bounded second derivative. For any T such that the integral $\int_0^T \|d_t\|_{p^{in}} dt$ stays stochastically bounded, as $n_1, ..., n_{L-1} \to \infty$, we have, uniformly for $t \in [0, T]$,*

$$\Theta^{(L)}(t) \to \Theta^{(L)}_\infty \otimes Id_{n_L}.$$

As a consequence, in this limit, the dynamics of f_θ is described by the differential equation

$$\partial_t f_{\theta(t)} = \Phi_{\Theta^{(L)}_\infty \otimes Id_{n_L}}\left(\langle d_t, \cdot\rangle_{p^{in}}\right).$$

References

[COB20] Lenaic Chizat, Edouard Oyallon, and Francis Bach, *On lazy training in differentiable programming*, 2020.

[JGH20] Arthur Jacot, Franck Gabriel, and Clément Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, 2020.