| CS 839: Theoretical Foundations of Deep Learning | | Spring 2022 |
|---|---|---|
| Lecture 11 Neural Tangent Kernel | | |
| *Instructor: Yingyu Liang* | *Date: Mar 1st, 2022* | *Scriber: Haotian Feng* |

# 1 Recap of Previous Lectures

In previous lectures, we talked about implicit regularization, implicit bias and training dynamics. There are several challenges: (1) for a big family of Neural Networks, we cannot directly use uniform convergence. So we need to show the training dynamics is restricted to certain subset, then we can have generalization. (2) non-convex optimization. We want to know why the optimization is achieving small loss and why it is restricted onto a certain regularized subset. We have discussed some implicit regularization about network learning in past lectures, where we only considers what happens if we reach a good solution, which is the ending stage of training dynamics. While the more interesting stage is we start from random initialization and train to achieve a good solution, which is not analyzed in previous lectures.

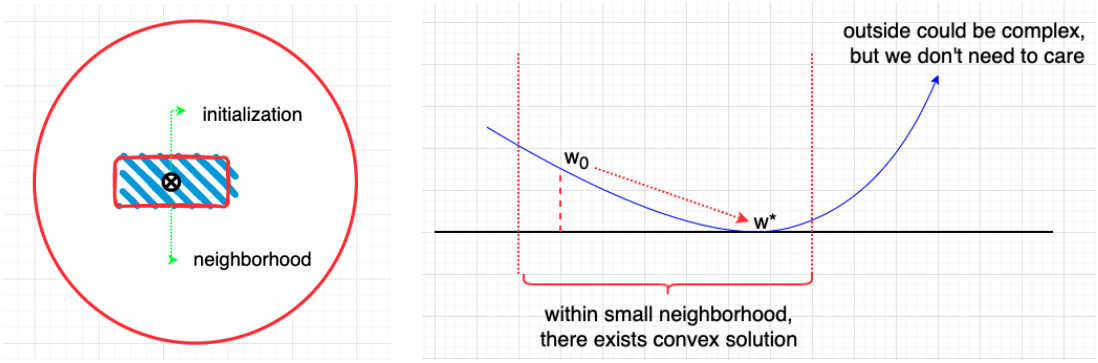
We talked about the soft margin problem, where we assume that at $t_0$ all the training data are correctly classified (i.e., $y_i f(x_i; w(t_0)) > 0, \forall i$), then the network can converge to a certain direction. Such analysis solely focuses on end stage. If we want to analyze what happens from random initialization to final prediction, we will have to consider more special conditions instead of the general training dynamics.

# 2 Overview

In this lecture, we will talk about a special case, where we consider training dynamics stay close to the random initialization, shown in Figure 1(a). To have strong generalization, we want training dynamics to stay in a small neighborhood of initialization, such that we can use the related good properties and existing mathematical tools. This is typically referred to the Neural Tangent Kernel view, or the kernel regime, or lazy training, or linearization. There are several approaches to formalize this view, and we will talk about one kind of such clean and typical tools – Neural Tangent Kernel, after which the view is named.

People discover that if we stay really close to the random initialization and use a large enough number of hidden layer neurons, then the properties within neighborhood will be similar to random initialization and the loss function in the small neighborhood will look like a convex optimization. This will automatically address the two key challenges: (1) non-convex problem will become a convex optimization; (2) over-parameterization will be explained within the small neighborhood as we will not use the huge hypothesis class.

We want the training dynamics to be "close" to initialization, but how to define "close" ? Intuitively, we want the optimization to be within initialization's neighborhood. Conceptually, the general neural network training includes NP-Hard cases, where we don't expect to get meaningful conclusion. Even for general cases, the initial steps are always close to the initialization. So there will be two cases: the hard case and the good case. For the hard




(a) (b)

Figure 1: (a) neighborhood around initialization (b) optimization within the neighborhood

case, we expect within limited steps, the solution is within the small neighborhood. After several steps, the solution goes out of the range, i.e., we have no control of the training (which can be NP-Hard). While for the good case, we can get the good solution within the small neighborhood, like convex optimization; See Figure 1(b) for an illustration.

Then why do we have a good solution in the neighborhood? As mentioned previously, researchers discover that considering one hidden layer Neural Network: for a small Neural Network with $m_0$ neurons, there's no good solution. As the number of hidden neurons goes larger to $m_i \gg m_0$, with certain random initialization and step size, the neighborhood allows almost convex optimization. If we have $m \to \infty$, under certain random initialization and step size, it becomes almost always convex. In fact, in the limit, the Neural Network becomes a kernel and kernel method is convex. That is, for any labelling over training data, there exists a combination that predicts the label perfectly. The kernel is called the Neural Tangent Kernel (NTK).

**Limitations.** The NTK view has several limitations. (1) Practical Neural Networks do not satisfy the theoretical assumptions, like the infinite Neural Network is not practical. (2) More importantly, NTK cannot explain some important behavior of practical Neural Networks. For example, practical training dynamics might not be convex and solution might be far from initialization. Another example is feature learning. For example on images, Neural Network's lower layer learns simple features like lines and higher layer neurons learns complex and semantic features like human faces. That is, practical Neural Network depends on data, which researchers believe these learned features give Neural Network stable and strong performance. But NTK or other kernel methods will use existing fixed features (data-independent feature mapping), so there is no feature learning for NTK.

But NTK is still useful in that it provides us the knowledge to perform different analysis under different conditions. Also it provides us a way to view special cases related to practical Neural Network, like very wide Neural Networks. Also NTK provides us a view at the very beginning stage of Neural Network training dynamics.

# 3 Neural Tangent Kernel Formalization

## 3.1 Neural Network Training

Now we begin with the general setting: consider $(x_i, y_i)_{i=1}^n, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$. Let $f(x; w)$ be a network with parameters $w$. We will do regression with squared loss:

$$L(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i; w))^2 \tag{1}$$

For Gradient Flow, we assume chain rule holds here:

$$\frac{dw(t)}{dt} = -\nabla L(w) \tag{2}$$

**Lemma 1.** Let $u(t) = (f(x_i; w(t)))_{i=1}^n$, $y = (y_1, \ldots, y_n)$. Then

$$\frac{du(t)}{dt} = -H(t)[u(t) - y] \tag{3}$$

where $H \in \mathbb{R}^{n\times n}$ and

$$H_{ij}(t) = \left\langle \frac{\partial f(x_i; w(t))}{\partial w}, \frac{\partial f(x_j; w(t))}{\partial w} \right\rangle.$$

*Proof.* The proof for the lemma is straightforward, where we directly apply the chain rule.

$$\left(\frac{du(t)}{dt}\right)_i = \frac{df(x_i; w(t))}{dt} = \left\langle \frac{\partial f(x_i; w(t))}{\partial w}, \frac{dw(t)}{dt} \right\rangle \tag{4}$$

Then considering for the term $\frac{dw(t)}{dt}$:

$$\begin{aligned} \frac{dw(t)}{dt} = -\nabla L(w) &= -\sum_{j=1}^{n}(f(x_j; w) - y_j)\frac{\partial f(x_j; w(t))}{\partial w} \\ &= -\sum_{j=1}^{n}(u_j(t) - y_j)\frac{\partial f(x_j; w(t))}{\partial w} \end{aligned} \tag{5}$$

Thus, substitute Equation 5 into Equation 4, we have:

$$\left(\frac{du(t)}{dt}\right)_i = -\sum_{j=1}^{n}(u_j(t) - y_j)\left\langle \frac{\partial f(x_i; w(t))}{\partial w}, \frac{\partial f(x_j; w(t))}{\partial w} \right\rangle = -\sum_{j=1}^{n}(u_j(t) - y_j)H_{ij}(t). \tag{6}$$

The lemma is proved. □

An intuition from Lemma 1: $u(t) - y$ is the residual vector and $\frac{du}{dt}$ is the gradient. For stationary points (i.e., the gradient is 0 and thus $\frac{du}{dt} = 0$), if $H(t)$ is positive and sufficiently large, then the lemma means the residual will be 0. However, $H(t)$ is hard to control as a function of $t$. So we want to use NTK to show $H(t) \approx H(0)$ at any location within the neighborhood, and show that $H(0) \gg 0$, so that $H(t) > 0$ and the training will become

an almost convex problem. In summary, the lemma shows that $H(t)$ controls the training dynamics described by $u(t)$, and we want $H(t) > 0$, which will be achieved by approximating $H(t)$ by $H(0)$ in the small neighborhood and bounding $H(0)$.

$H(0)$ is bounded due to the randomness of initialization. Note that

$$H_{ij}(0) = \langle \frac{\partial f(x_i; w(t))}{\partial w}, \frac{\partial f(x_j; w(t))}{\partial w} \rangle|_{w=w(0)},$$

where $w(0)$ is random. We can get rid of the randomness by taking expectation on $H(0)$, which is defined as **Gram matrix** of NTK:

$$H^* = \mathbb{E}_{w(0)}[H(0)].$$

Then $H^*_{ij} = k(x_i, x_j)$ for some function $k$ which can be viewed as a kernel function. Indeed, this is called the Neural Tangent Kernel. When we have large number of neurons, we are taking expectations w.r.t neuron draws, and $H(0)$ is expected to be close to $H^*$. In the extreme of infinitely large wide Neural Network, it corresponds to the expectation.

## 3.2 NTK on Two-Layer Neural Networks with ReLU

For 2-layer Neural Network with ReLU, we define

$$f(x; W, a) = \frac{1}{\sqrt{m}} \sum_{k=1}^{m} a_k \sigma(\langle w_k, x \rangle),$$

where $\sigma(z) = \max(0, z)$. We will use special initialization scheme: $w_k(0) \sim N(0, I_d)$ and $a_k \sim \text{uniform}\{-1, +1\}$. During training, $a_k$ is fixed and we only update $w_k$, which is the hidden neurons' weights.

Then we have:

$$\begin{aligned} H_{ij}(0) &= \left\langle \frac{\partial f(x_i)}{\partial w}, \frac{\partial f(x_j)}{\partial w} \right\rangle \\ &= \sum_{k=1}^{m} \left\langle \frac{\partial f(x_i)}{\partial w_k}, \frac{\partial f(x_j)}{\partial w_k} \right\rangle \\ &= \sum_{k=1}^{m} \left\langle \frac{1}{\sqrt{m}} a_k \sigma'(\langle w_k, x_i \rangle) x_i, \frac{1}{\sqrt{m}} a_k \sigma'(\langle w_k, x_j \rangle) x_j \right\rangle \\ &= x_i^T x_j \frac{1}{m} \sum_{k=1}^{m} \sigma'(\langle w_k, x_i \rangle) \sigma'(\langle w_k, x_j \rangle) \end{aligned} \tag{7}$$

$$\begin{aligned} H^*_{ij} &= \mathbb{E}_{w \sim N(0,I)}[H_{ij}(0)] = x_i^T x_j \mathbb{E}_{w \sim N(0,I)}[\sigma'(x_i^T w_k) \sigma'(x_j^T w_k)] \\ &= \frac{\pi - \arccos(\frac{x_i^T x_j}{\|x_i\| \|x_j\|})}{2\pi} \end{aligned} \tag{8}$$

Under mild conditions, one can have the smallest eigenvalue $\lambda_{\min}(H^*) > 0$, i.e., $H^*$ is positive. So, if we want $H(t) > 0$, we can first prove that $H(0) \approx H^*$ and then prove $H(t) \approx H(0)$.

We now show $H(0) \approx H^*$ when the number of neurons $m$ is large. Intuitively, this is because $H(0)$ is the empirical average with $m$ samples and $H^*$ is the expectation. To rigorously prove it, we can use concentration bounds like Hoeffding's inequality.

**Lemma 2.** Assume $\|x_i\| \leq 1$ and $\sigma(z) = \max\{0, z\}$. If the number of hidden neuron $m \geq \Omega(\epsilon^{-2} n^2 \log(\frac{n}{\delta}))$, then with probability at least $1 - \delta$ over the random initialization,

$$\|H(0) - H^*\|_2 \leq \epsilon.$$

*Proof.* From the assumption, we know $H_{ij}(0) \in [-1, +1]$, and we know $\mathbb{E}[H_{ij}(0)] = H^*_{ij}(0)$. For any fixed pair of $(i, j)$, using Hoeffding's inequality:

$$\Pr[|H_{ij}(0) - H^*_{ij}| \geq t] \leq 2e^{-\frac{mt^2}{2}} \tag{9}$$

Then for all $(i, j)$:

$$\Pr[\exists (i,j), |H_{ij}(0) - H^*_{ij}| \geq t] \leq 2n^2 e^{\frac{-mt^2}{2}}. \tag{10}$$

Now set $t = \epsilon/n$. If for all $(i, j)$, $|H_{ij}(0) - H^*_{ij}| < t$, then

$$\|H(0) - H^*\|_2^2 \leq \|H(0) - H^*\|_F^2 \leq n^2 \frac{\epsilon^2}{n^2} = \epsilon^2.$$

So it is sufficient to ensure $2n^2 e^{\frac{-mt^2}{2}} \leq \delta$. This is satisfied by $m \geq \Omega(\epsilon^{-2} n^2 \log(\frac{n}{\delta}))$. □