Chapter 10

Optimization

This chapter depends heavily on the exposition by Dwaraknath (2019), which also has a lot of nice visualizations.

10.1 Linear Models

In this section, we show that if we train a linear regression model by gradient descent, it converges to the global optimum. Assume that the training examples are packed into $\mathbf{X} \in \mathbb{R}^{N \times d}$ and $\mathbf{y} \in \mathbb{R}^N$ such that $(\mathbf{X}[i], \mathbf{y}[i])$ is the i-th example. The model's prediction for the i-th example is

$$y = \mathbf{w} \cdot \mathbf{X}[i]. \tag{10.1}$$

The squared-error loss is

$$\begin{align}
\mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=0}^{N-1} (\mathbf{y}[i] - \mathbf{w} \cdot \mathbf{X}[i])^2 \tag{10.2} \\
&= \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2. \tag{10.3}
\end{align}$$

Training by gradient descent looks like this:

$$\begin{align}
\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= -(\mathbf{y} - \mathbf{X}\mathbf{w})^\top \mathbf{X} \tag{10.4} \\
\mathbf{w}(t + \eta) &= \mathbf{w}(t) - \eta \left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \right)^\top \tag{10.5} \\
&= \mathbf{w}(t) + \eta \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}(t)). \tag{10.6}
\end{align}$$

We did something a little funny with the numbering of the weight vectors. We imagine that training starts at time $t = 0$, and at each step, we increment the time by the learning rate η. Thought of in this way, this looks like the discretization of a continuous-time process. If we take the limit as $\eta \to 0$, we get the differential equation (called the *gradient flow*),

$$\begin{aligned} \frac{d\mathbf{w}}{dt} &= -\left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}\right)^\top && (10.7) \\ &= \mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}). && (10.8) \end{aligned}$$

We're interested in how the model's predictions improve or don't improve over time. So we write down a differential equation for the vector of errors $(\mathbf{y} - \mathbf{X}\mathbf{w})$:

$$\begin{aligned} \frac{d}{dt}(\mathbf{y} - \mathbf{X}\mathbf{w}) &= -\mathbf{X}\frac{d\mathbf{w}}{dt} && (10.9) \\ &= -\mathbf{X}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}). && (10.10) \end{aligned}$$

To make solving this differential equation easier, observe that since $\mathbf{X}\mathbf{X}^\top$ is symmetric, it is diagonalizable. That is, there is a matrix $\mathbf{Q}$ that is orthogonal ($\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$) and makes $\mathbf{\Lambda} = \mathbf{Q}\mathbf{X}\mathbf{X}^\top\mathbf{Q}^\top$ diagonal. Apply the transformation $\mathbf{Q}$ to transform the whole dataset:

$$\begin{aligned} \mathbf{X}' &= \mathbf{Q}\mathbf{X} && (10.11) \\ \mathbf{y}' &= \mathbf{Q}\mathbf{y}. && (10.12) \end{aligned}$$

Each new example $(\mathbf{X}'[i], \mathbf{y}'[i])$ is some linear combination of the old examples. Let $\mathbf{u}'$ be the vector of errors on the new data. We can write the loss in terms of this vector, so minimizing the new error is equivalent to minimizing the old error.

$$\begin{aligned} \mathbf{u}' &= \mathbf{y}' - \mathbf{X}'\mathbf{w} && (10.13) \\ \mathcal{L}(\mathbf{w}) &= \frac{1}{2}\|\mathbf{Q}^\top\mathbf{u}'\|^2. && (10.14) \end{aligned}$$

We're again interested in how the errors improve or don't improve over time:

$$\begin{aligned} \mathbf{u}' &= \mathbf{y}' - \mathbf{X}'\mathbf{w} && (10.15) \\ \frac{d\mathbf{u}'}{dt} &= -\mathbf{X}'\frac{d\mathbf{w}}{dt} && (10.16) \\ &= -\mathbf{X}'\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) && (10.17) \\ &= -\mathbf{X}'(\mathbf{Q}^\top\mathbf{X}')^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) && (10.18) \\ &= -\mathbf{X}'\mathbf{X}'^\top\mathbf{Q}(\mathbf{y} - \mathbf{X}\mathbf{w}) && (10.19) \\ &= -\mathbf{\Lambda}\mathbf{u}'. && (10.20) \end{aligned}$$

Now we can solve the differential equation for each $\mathbf{u}'[i]$ separately:

$$\mathbf{u}'[i] = \exp(-\mathbf{\Lambda}[i,i]t)\mathbf{u}'[i]. \tag{10.21}$$

For each i, we have $\mathbf{\Lambda}[i,i] = \|\mathbf{X}'[i]\|^2 \geq 0$.

- If $\mathbf{\Lambda}[i,i] > 0$, then $\mathbf{u}'[i]$, the error on the i-th new example, decreases over time and decays to 0 (good).
- If $\mathbf{\Lambda}[i,i] = 0$, then $\mathbf{u}'[i]$ does not change over time. But this implies $\mathbf{X}'[i] = \mathbf{0}$, so $\mathbf{u}'[i] = \mathbf{y}'[i]$.
  - If $\mathbf{y}'[i] = 0$, then the error on the i-th new example was 0 all along (good).
  - If $\mathbf{y}'[i] > 0$, then the error on the i-th new example will never be 0 (bad) but it also does not depend on the initial weights, so it would be present even at the global minimum of $\mathcal{L}$ (good).

We can conclude that as $t \longrightarrow \infty$, the loss converges to its global minimum.

10.2 Nonlinear Models

10.2.1 Gradient flow

Now consider a nonlinear network $f(\mathbf{x}, \mathbf{w})$, where $\mathbf{x} \in \mathbb{R}^d$ is the input and $\mathbf{w} \in \mathbb{R}^p$ are the parameters. As before, we pack the training data into $\mathbf{X} \in \mathbb{R}^{N \times d}$ and $\mathbf{y} \in \mathbb{R}^N$. Let's also pack the network outputs into a vector,

$$\mathbf{f}(\mathbf{w}) = \begin{bmatrix} f(\mathbf{X}[0], \mathbf{w}) \\ \vdots \\ f(\mathbf{X}[N-1], \mathbf{w}) \end{bmatrix} \tag{10.22}$$

$$\frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial f(\mathbf{X}[0],\mathbf{w})}{\partial \mathbf{w}[0]} & \cdots & \frac{\partial f(\mathbf{X}[0],\mathbf{w})}{\partial \mathbf{w}[p-1]} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{X}[N-1],\mathbf{w})}{\partial \mathbf{w}[0]} & \cdots & \frac{\partial f(\mathbf{X}[N-1],\mathbf{w})}{\partial \mathbf{w}[p-1]} \end{bmatrix} \tag{10.23}$$

We again use the squared-error loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \mathbf{f}(\mathbf{w})\|^2. \tag{10.24}$$

If we repeat the above derivation of the gradient flow, we get

$$\frac{d\mathbf{w}}{dt} = -\left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}\right)^\top \tag{10.25}$$

$$= \frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}}^\top (\mathbf{y} - \mathbf{f}(\mathbf{w})) \tag{10.26}$$

$$\frac{d}{dt}(\mathbf{y} - \mathbf{f}(\mathbf{w})) = -\frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}} \frac{d\mathbf{w}}{dt} \tag{10.27}$$

$$= \underbrace{-\frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}} \left(\frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}}\right)^\top}_{(*)} (\mathbf{y} - \mathbf{f}(\mathbf{w})). \tag{10.28}$$

The term $(*)$ depends on $\mathbf{w}$, which depends on t. So it looks like we might be stuck.

10.2.2 Neural tangent kernel

For brevity, we write $\nabla\mathcal{L}$ for $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ and $D\mathbf{f}$ for $\frac{\partial \mathbf{f}(\mathbf{w})}{\partial \mathbf{w}}$. The trick we are going to use is to use a first-order Taylor approximation of f with respect to $\mathbf{w}$, around the initial weights $\mathbf{w}_0$. That is, in $(*)$, we replace $\nabla \mathbf{f}(\mathbf{w})$ with $\nabla \mathbf{f}(\mathbf{w}_0)$, giving the *neural tangent kernel* or *NTK* (Jacot et al., 2018). Then Eq. (10.28) can be solved exactly as before.

The main question now becomes, under what conditions is the linear approximation a good approximation (known as the *NTK regime* or *lazy training*)? There are many ways of answering this question; Dwaraknath (2019) follows Chizat et al. (2019), so we will too.

Inverse relative scale If we take one step of gradient descent, $\mathbf{w} = \mathbf{w}_0 - \eta(\nabla\mathcal{L}(\mathbf{w}_0))^\top$, we'd like the change in the loss to be big and the change in $D\mathbf{f}$ to be small. The change in the loss is

$$\Delta\mathcal{L}(\mathbf{w}_0) = |\mathcal{L}(\mathbf{w}) - \mathcal{L}(\mathbf{w}_0)| \tag{10.29}$$

$$= \mathcal{L}(\mathbf{w}_0 - \eta(\nabla\mathcal{L}(\mathbf{w}_0))^\top) - \mathcal{L}(\mathbf{w}_0) \tag{10.30}$$

$$\approx \mathcal{L}(\mathbf{w}_0) - \eta\|\nabla\mathcal{L}(\mathbf{w}_0)\|^2 - \mathcal{L}(\mathbf{w}_0) \qquad \text{first-order Taylor approx.} \tag{10.31}$$

$$= -\eta\|\nabla\mathcal{L}(\mathbf{w}_0)\|^2. \tag{10.32}$$

(If that looks weird to you, it kind of is! It's the reason there are so many alternatives to gradient descent, like Adam.) The number of steps it would take to reach zero loss would be something like

$$K = \frac{\mathcal{L}(\mathbf{w}_0)}{\eta \|\nabla \mathcal{L}(\mathbf{w}_0)\|^2}. \tag{10.33}$$

Similarly, the change in $D\mathbf{f}$ is

$$\begin{aligned}
\Delta(D\mathbf{f}(\mathbf{w}_0)) &= \|D\mathbf{f}(\mathbf{w}) - D\mathbf{f}(\mathbf{w}_0)\| && (10.34)\\
&= \|D\mathbf{f}(\mathbf{w}_0 - \eta(\nabla \mathcal{L}(\mathbf{w}_0))^\top) - D\mathbf{f}(\mathbf{w}_0)\| && (10.35)\\
&\approx \|D\mathbf{f}(\mathbf{w}_0) - \eta\, D^2\mathbf{f}(\mathbf{w}_0)\, (\nabla \mathcal{L}(\mathbf{w}_0))^\top - D\mathbf{f}(\mathbf{w}_0)\| && (10.36)\\
&= \eta \|D^2\mathbf{f}(\mathbf{w}_0)\, (\nabla \mathcal{L}(\mathbf{w}_0))^\top\| && (10.37)\\
&\le \eta\, \|D^2\mathbf{f}(\mathbf{w}_0)\|\, \|\nabla \mathcal{L}(\mathbf{w}_0)\| && (10.38)
\end{aligned}$$

(Here, $\|\cdot\|$ around matrices is the operator 2-norm, and $D^2\mathbf{f}(\mathbf{w}_0)$ has type $\mathbb{R}^p \to \mathbb{R}^{N\times p}$, but it's really okay if you don't want to understand these details.) After K steps, the total *relative* change in $D\mathbf{f}$ would be at most something like

$$\begin{aligned}
K\frac{\Delta(D\mathbf{f}(\mathbf{w}_0))}{\|D\mathbf{f}(\mathbf{w}_0)\|} &= \frac{\mathcal{L}(\mathbf{w}_0)\, \eta\, \|D^2\mathbf{f}(\mathbf{w}_0)\|\, \|\nabla \mathcal{L}(\mathbf{w}_0)\|}{\eta\, \|\nabla \mathcal{L}(\mathbf{w}_0)\|^2\, \|D\mathbf{f}(\mathbf{w}_0)\|} && (10.39)\\
&= \frac{\mathcal{L}(\mathbf{w}_0)\, \|D^2\mathbf{f}(\mathbf{w}_0)\|}{\|\nabla \mathcal{L}(\mathbf{w}_0)\|\, \|D\mathbf{f}(\mathbf{w}_0)\|} && (10.40)\\
&= \frac{\|\mathbf{y} - \mathbf{f}(\mathbf{w}_0)\|^2\, \|D^2\mathbf{f}(\mathbf{w}_0)\|}{\|\mathbf{y} - \mathbf{f}(\mathbf{w}_0)\|\, \|D\mathbf{f}(\mathbf{w}_0)\|\, \|D\mathbf{f}(\mathbf{w}_0)\|} && (10.41)\\
&= \frac{\|\mathbf{y} - \mathbf{f}(\mathbf{w}_0)\|\, \|D^2\mathbf{f}(\mathbf{w}_0)\|}{\|D\mathbf{f}(\mathbf{w}_0)\|^2} && (10.42)\\
&\overset{\text{def}}{=} \kappa_f(\mathbf{w}_0). && (10.43)
\end{aligned}$$

And we want $\kappa_f(\mathbf{w}_0) \ll 1$.

Scaled models There's a very simple way to ensure this. Assume that $f(\mathbf{x}, \mathbf{w}_0) = 0$, which we can ensure by randomly initializing half of the network and then initializing the other half to be its negative. Consider the function αf:

$$\begin{aligned}
\kappa_{\alpha f}(\mathbf{w}_0) &= \|\mathbf{y} - \alpha\mathbf{f}(\mathbf{w}_0)\| \frac{\alpha\|D^2\mathbf{f}(\mathbf{w}_0)\|}{\alpha^2\|D\mathbf{f}(\mathbf{w}_0)\|^2} && (10.44)\\
&= \|\mathbf{y}\| \frac{\|D^2\mathbf{f}(\mathbf{w}_0)\|}{\alpha\|D\mathbf{f}(\mathbf{w}_0)\|^2}. && (10.45)
\end{aligned}$$

So we can make $\kappa_{\alpha f}$ small by making α big. Chizat et al. (2019) look at the relative difference between the gradient flow of αf and that of its first-order Taylor approximation $\alpha\bar{f}$, showing that it has an upper bound related to $\kappa_{\alpha f}(\mathbf{w}_0)$.

Two-layer FFNN In particular, let f be a two-layer neural network with hidden-layer width d_{hid},

$$f(\mathbf{x}, \mathbf{U}, \mathbf{v}) = \alpha(d_{\text{hid}})\ \mathbf{v} \cdot \sigma(\mathbf{U}\mathbf{x}) \tag{10.46}$$

where α depends on the width. Let $\mathbf{w}_0 = (\mathbf{U}_0, \mathbf{v}_0)$ be a random initialization. Then it can be shown that

$$E[\kappa_f(\mathbf{w}_0)] \lesssim \frac{1}{\sqrt{d_{\text{hid}}}} + \frac{1}{d_{\text{hid}}\, \alpha(d_{\text{hid}})}. \tag{10.47}$$

If we let $\alpha(d_{\text{hid}}) = 1/\sqrt{d_{\text{hid}}}$, as is standard in NTK papers, then we can make $\kappa_{\alpha f}$ small by making d_{hid} big.

There seems to be some dissonance about whether this scaling factor of $\sqrt{d_{\text{hid}}}$ matches up with how neural networks are initialized in practice: for example, so-called Xavier uniform initialization (Glorot and Bengio, 2010) would initialize the second-layer weights from Uniform $\left(\pm\sqrt{\frac{6}{d_{\text{hid}}+1}}\right)$. That does look similar, but as Chizat et al. (2019), scaling the second-layer weights versus scaling the model itself don't have the same gradient.

Limitations and alternatives The most serious objection to the above analysis is that neural networks in practice don't operate within the NTK regime. If they did, then we would expect their first-order approximations to work nearly as well, and they do not. Chizat et al. (2019) experimented with varying α and found that larger α indeed causes lazy training, but also does not perform well.

A second limitation is that it's not clear how to make this analysis work for transformers. Why? A randomly initialized self-attention layer has close-to-uniform attention (Alberto Bietti, p.c.). So transformers near initialization, unlike FFNNs near initialization, are not universal approximators.

An alternative approach to the NTK is *mean-field* analysis (Mei et al., 2018), which studies the regime in which $\alpha = 1/m$ instead of $\alpha = 1/\sqrt{m}$.

10.3 Gradient Descent

So far, we've considered gradient flows, which approximate gradient descent using an infinitesimal learning rate (η). What about (batch, not stochastic) gradient descent – how does the choice of η affect training?

Definition 10.1. A differentiable function $\mathcal{L}\colon \mathbb{R}^d \to \mathbb{R}$ is β*-smooth* for $\beta \geq 0$ if its gradient $\nabla\mathcal{L}$ is β-Lipschitz. That is, for all $\mathbf{w}, \mathbf{w}'$,

$$\|\nabla\mathcal{L}(\mathbf{w}') - \nabla\mathcal{L}(\mathbf{w})\| \leq \beta\|\mathbf{w}' - \mathbf{w}\|.$$

If $\mathcal{L}$ is twice differentiable, then β-smoothness is equivalent to the eigenvalues of $\nabla^2\mathcal{L}(\mathbf{w})$ being all at most β.

Lemma 10.2. *Let $\mathcal{L}\colon \mathbb{R}^d \to \mathbb{R}$ be β-smooth, and let $\eta < 2/\beta$ be a learning rate. Then one step of gradient descent decreases $\mathcal{L}$:*

$$\begin{aligned}\mathbf{w}' &= \mathbf{w} - \eta(\nabla\mathcal{L}(\mathbf{w}^{(t)}))^\top \\ \mathcal{L}(\mathbf{w}') &< \mathcal{L}(\mathbf{w}).\end{aligned}$$

Proof. (Farina, 2024; Arora, 2022) For brevity write $\mathbf{h} = \mathbf{w}' - \mathbf{w}$.

$$\begin{aligned}
\mathcal{L}(\mathbf{w}') &= \mathcal{L}(\mathbf{w}) + \int_0^1 \nabla\mathcal{L}(\mathbf{w}+\tau\mathbf{h})\cdot\mathbf{h}\,d\tau && (10.48)\\
&= \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})\cdot\mathbf{h} + \int_0^1 (\nabla\mathcal{L}(\mathbf{w}+\tau\mathbf{h}) - \nabla\mathcal{L}(\mathbf{w}))\cdot\mathbf{h}\,d\tau && (10.49)\\
&\le \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})\cdot\mathbf{h} + \int_0^1 \|\nabla\mathcal{L}(\mathbf{w}+\tau\mathbf{h}) - \nabla\mathcal{L}(\mathbf{w})\|\,\|\mathbf{h}\|\,d\tau && (10.50)\\
&\le \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})\cdot\mathbf{h} + \int_0^1 \beta\tau\|\mathbf{h}\|^2\,d\tau && (10.51)\\
&= \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})\cdot\mathbf{h} + \frac{\beta}{2}\|\mathbf{h}\|^2 && (10.52)\\
&= \mathcal{L}(\mathbf{w}) - \nabla\mathcal{L}(\mathbf{w})\cdot\eta\nabla\mathcal{L}(\mathbf{w}) + \frac{\beta}{2}\|\eta\nabla\mathcal{L}(\mathbf{w})\|^2 && (10.53)\\
&= \mathcal{L}(\mathbf{w}) + \left(-1+\frac{\beta\eta}{2}\right)\eta\|\nabla\mathcal{L}(\mathbf{w})\|^2 && (10.54)\\
&< \mathcal{L}(\mathbf{w}). && (10.55)
\end{aligned}$$

□

From this, we would expect that smaller values of β are good, and we should set $\eta < 2/\beta$.

Example: linear regression

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{2}\sum_{i=0}^{N-1}(\mathrm{y}[i] - \mathbf{w}\cdot\mathbf{X}[i])^2 && (10.56)\\
&= \frac{1}{2}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 && (10.57)\\
\nabla\mathcal{L}(\mathbf{w}) &= (\mathbf{X}\mathbf{w} - \mathbf{y})^\top\mathbf{X} && (10.58)\\
\nabla^2\mathcal{L}(\mathbf{w}) &= \mathbf{X}^\top\mathbf{X}. && (10.59)
\end{aligned}$$

Remember that $\mathbf{X}^\top\mathbf{X}$ appeared before in the calculation of the gradient flow, and we diagonalized it into Λ; the leading entry of this matrix is β.

The edge of stability However, for neural networks (Cohen et al., 2021), the condition $\eta < 2/\beta$ is routinely violated. If we think of β as something that can vary from region to region (we'd have to modify the above definitions accordingly), then β increases until it reaches $2/\eta$, and then it enters a regime called "the edge of stability" in which β remains the same, but the loss continues to improve. Why this happens, and what the analogous results for other training methods like stochastic gradient descent or adaptive methods like Adam, is still an active area of research.

10.4 Optimization is NP-hard

Finally, we consider the most general version of the optimization problem: with no constraints on $\mathcal{L}$ and no constraints on the optimization method, how hard is it to find the global minimum of $\mathcal{L}$?

Theorem 10.3. *Given a function $\mathcal{L}\colon \mathbb{R}^d \to \mathbb{R}$ and $\ell \in \mathbb{R}$, it is NP-hard to decide whether $\ell = \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$.*

Proof. By reduction from SUBSET-SUM, which is the problem of deciding, given $\{a_0, \ldots, a_{n-1}\} \subseteq \mathbb{N}$ and $t \in \mathbb{N}$, whether there is a subset $C \subseteq [n]$ such that $\sum_{i \in C} a_i = t$. Given a SUBSET-SUM instance $(\{a_0, \ldots, a_{n-1}\}, t)$, construct the following global minimization problem:

$$\begin{aligned}
\mathcal{L}&\colon \mathbb{R}^n \to \mathbb{R} \\
\mathcal{L}(\mathbf{w}) &= \left(\sum_{i \in [d]} \mathbf{w}[i]\, a_i - t\right)^2 + \sum_{i \in [d]} (\mathbf{w}[i](1 - \mathbf{w}[i]))^2 \\
\ell &= 0.
\end{aligned}$$

If there is a $C \subseteq [n]$ such that $\sum_{i \in C} a_i = t$, then

$$\mathcal{L}\left(\begin{bmatrix} \mathbb{I}[0 \in C] \\ \vdots \\ \mathbb{I}[n-1 \in C] \end{bmatrix}\right) = 0.$$

Otherwise, $\mathcal{L}(\mathbf{w}) > 0$ for all $\mathbf{w}$. □

(This is the usual way that global minimization is formulated as a decision problem. The other way would be: given $\mathcal{L}$ and $\mathbf{w}$, decide whether $\mathbf{w}$ is a minimizer of $\mathcal{L}$. Presumably this is also NP-hard, but seems trickier to prove.)

Bibliography

Arora, Sanjeev (Mar. 2022). *Theory of Deep Learning*.

Chizat, Lénaïc, Edouard Oyallon, and Francis Bach (2019). On lazy training in differentiable programming. In: *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

Cohen, Jeremy M., Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar (2021). Gradient descent on neural networks typically occurs at the edge of stability. In: *Proceedings of the Ninth International Conference on Learning Representations*.

Dwaraknath, Rajat Vadiraj (2019). Understanding the neural tangent kernel. Eigen-Tales (blog).

Farina, Gabriele (Mar. 2024). Lecture 7: gradient descent.

Glorot, Xavier and Yoshua Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics(AISTATS)*. Vol. 9. Proceedings of Machine Learning Research, pp. 249–256.

Jacot, Arthur, Franck Gabriel, and Clement Hongler (2018). Neural tangent kernel: convergence and generalization in neural networks. In: *Advances in Neural Information Processing Systems*. Vol. 31.

Mei, Song, Andrea Montanari, and Phan-Minh Nguyen (July 2018). A mean field view of the landscape of two-layer neural networks. In: 115.33, E7665–E7671.