Mathematics of Deep Learning
Lecture 3: Neural networks close to initialisation

Bruno Loureiro
Département d'Informatique, École Normale Supérieure - PSL & CNRS, France

07/02/2025

Typos, comments or suggestions? Get in touch at: bruno.loureiro@di.ens.fr

1 Motivation

In the previous lectures, we have mostly focused our attention of the approximation question: given a hypothesis class, how well the the best predictor in the class can approximate the Bayes risk? The key take away from this discussion is that neural networks are pretty good in approximation. Even the simplest two-layer neural network is able to *uniformly* approximate an arbitrary *continuous function* on a *compact set* provided the *width is large enough*. In the worst case, "large enough" means exponentially in the input dimension. Nevertheless, we saw that this worst case result can be considerable improved if the target function is more regular, with the width scaling proportionally to a regularity measure (a.k.a. the Barron norm).

While this result is satisfying in many ways, it is only one piece of the puzzle. Sure, the best choice of first and second layer weights does the approximation job, but in the worst case finding this weights with an actual algorithm is a known NP-hard problem (Blum and Rivest, 1988). Nevertheless, training large neural networks with SGD on limited data seem to yield good generalisation performance. How come?

As we saw in Lecture 1, the approximation error is only one component of the generalisation gap:

$$R(\hat{\theta}) - R_\star = \underbrace{\left\{ R(\hat{\theta}) - \inf_{\theta\in\Theta} R(\theta) \right\}}_{\text{estimation}} + \underbrace{\left\{ \inf_{\theta\in\Theta} R(\theta) - R_\star \right\}}_{\text{approximation}}. \tag{1.1}$$

with the remaining part, the *estimation error*, accounting for how far our predictor is from the best predictor in the class. Understanding the estimation error involves both statistical and algorithmic aspects, and only recently we started developing a good understanding of this term for the case of two-layer neural trained with descent-based algorithms such as GD and SGD. In next few lectures, we will review some of this progress, starting from the influential result that in some regime, wide neural networks are equivalent to kernel methods.

2 The neural tangent kernel

Understanding the training dynamics of two-layer networks requires understanding how the weights move under a certain algorithm. Therefore, the starting point is initialisation.

	$\boldsymbol{a} \in \mathbb{R}^p$	$\boldsymbol{W} \in \mathbb{R}^{p \times d}$	$\boldsymbol{b} \in \mathbb{R}^p$
Kaiming (PyTorch)	$a_j \underset{i.i.d.}{\sim} \text{Unif}\left(\left[-\sqrt{\frac{6}{p}}, \sqrt{\frac{6}{p}}\right]\right)$	$w_{jk} \underset{i.i.d.}{\sim} \text{Unif}\left(\left[-\sqrt{\frac{6}{d}}, \sqrt{\frac{6}{d}}\right]\right)$	$\mathbf{0}$
Xavier (TensorFlow)	$a_j \underset{i.i.d.}{\sim} \text{Unif}\left(\left[-\sqrt{\frac{6}{p+1}}, \sqrt{\frac{6}{p+1}}\right]\right)$	$w_{jk} \underset{i.i.d.}{\sim} \text{Unif}\left(\left[-\sqrt{\frac{6}{p+d}}, \sqrt{\frac{6}{p+d}}\right]\right)$	$\mathbf{0}$

Table 1: Default initialisation for PyTorch and TensorFlow, also known the the Kaiming-Hu and the Xavier-Glorot initialisations, respectively.

2.1 Wide networks at initialisation

Consider a two-layer neural network with activation σ and width p:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle + b_j) \tag{2.1}$$

In a standard Python frameworks such as pytorch or tensorflow, the weights are typically initialised at random from a uniform distribution at a interval depending on the width and the dimension, see table 1. It is curious to note that irrespective of your choice, the interval of the second layer weights shrinks as the width grows, in particular, the standard deviation vanishes as:

$$\text{std}(a_j^0) = O(1/\sqrt{p}) \text{ as } p \to \infty \tag{2.2}$$

The discussion above motivates the following rewriting at initialisation:

$$f(\boldsymbol{x};\boldsymbol{\theta}^0) = \frac{1}{\sqrt{p}} \sum_{j=1}^{p} \tilde{a}_j^0 \sigma(\langle \boldsymbol{w}_j^0, \boldsymbol{x}\rangle) \tag{2.3}$$

where $\tilde{a}_j = O(1)$. For reasons that we will see next, the $1/\sqrt{p}$ scaling is also known as the *NTK scaling*, and can be understood as the right scaling in order to make the $f(\boldsymbol{x};\boldsymbol{\theta}^0) = O(1)$ at initialisation. Indeed, $f(\boldsymbol{x};\boldsymbol{\theta}^0)$ is given by a sum of p independent random variables with $O(1)$ variance.

This is precisely the scaling of the central limit theorem! Indeed, assuming $\boldsymbol{x} \in \mathbb{R}^d$ is fixed, the above is a sum of independent random variables with standard deviation $1/\sqrt{p}$ - meaning that in the limit $p \to \infty$ the network at initialisation will converge to a Gaussian function with mean zero and variance given by:

$$K_{\text{GP}}(\boldsymbol{x},\boldsymbol{x}') \coloneqq \mathbb{E}[f(\boldsymbol{x};\boldsymbol{\theta}^0) f(\boldsymbol{x}';\boldsymbol{\theta}^0)] = \mathbb{E}_{a,\boldsymbol{w}}\left[a^2 \sigma(\langle \boldsymbol{w},\boldsymbol{x}\rangle)\sigma(\langle \boldsymbol{w},\boldsymbol{x}'\rangle)\right] \tag{2.4}$$

This is known as a *Gaussian process*, and was first derived in (Neal, 1996). In other words:

$$f(\cdot;\boldsymbol{\theta}^0) \xrightarrow{d} \mathcal{GP}(0, K_{\text{GP}}(\cdot,\cdot)) \tag{2.5}$$

⚠ Note that the $1/\sqrt{p}$ CLT scaling in eq. (2.3) is different from the $1/p$ scaling assumed in the discussion of Barron spaces in last lecture. As we will see later, this has important consequences for the space of functions that can be expressed by networks in this scaling.

2.2 Wide networks close to initialisation

At initialisation, a wide neural network with standard scaling is a Gaussian random function. What happens as we move away from initialisation? Before looking at particular training algorithms, let's look at how our network looks *close* to initialisation. More concretely, let $\boldsymbol{\theta} \in B(\boldsymbol{\theta}^0, 1)$. By Taylor's theorem:

$$f(\boldsymbol{x};\boldsymbol{\theta}) \approx f(\boldsymbol{x};\boldsymbol{\theta}^0) + \langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}^0), \boldsymbol{\theta} - \boldsymbol{\theta}^0\rangle \tag{2.6}$$

Therefore, to first order the network $f(\boldsymbol{x};\boldsymbol{\theta})$ can be seen as linear method with features $\varphi(\boldsymbol{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta})$ — also known as the *neural tangent features*. But how close is the predictor to being linear? This is quantified by the next order or remainder term:

$$f(\boldsymbol{x};\boldsymbol{\theta}) \approx f(\boldsymbol{x};\boldsymbol{\theta}^0) + \langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}^0), \boldsymbol{\theta}-\boldsymbol{\theta}^0\rangle + \frac{1}{2}\langle \boldsymbol{\theta}-\boldsymbol{\theta}^0, \boldsymbol{H}(\boldsymbol{\theta}-\boldsymbol{\theta}^0)\rangle \tag{2.7}$$

where $\boldsymbol{H} = \nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}^0)$ is the Hessian matrix. Therefore, defining the linearised network:

$$f_{\text{lin.}}(\boldsymbol{x};\boldsymbol{\theta}) = f(\boldsymbol{x};\boldsymbol{\theta}^0) + \langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}^0), \boldsymbol{\theta}-\boldsymbol{\theta}^0\rangle \tag{2.8}$$

we can see that the deviation of f from $f_{\text{lin.}}$ in uniform norm is quantified by the operator norm of the Hessian:

$$\sup_{\boldsymbol{\theta}\in B(\boldsymbol{\theta}^0,1)} |f(\boldsymbol{x};\boldsymbol{\theta}) - f_{\text{lin.}}(\boldsymbol{x};\boldsymbol{\theta})| = \frac{1}{2}\sup_{\boldsymbol{\theta}\in B(\boldsymbol{\theta}^0,1)} \langle \boldsymbol{\theta}-\boldsymbol{\theta}^0, \boldsymbol{H}(\boldsymbol{\theta}-\boldsymbol{\theta}^0)\rangle \tag{2.9}$$

$$= \sup_{||\boldsymbol{v}||_2=1} \langle \boldsymbol{v}, \boldsymbol{H}\boldsymbol{v}\rangle \coloneqq \frac{1}{2}||\boldsymbol{H}||_{\text{op}} \tag{2.10}$$

Where the last equality follows from the fact that the quadratic form $f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}\rangle$ with $\boldsymbol{A} \succeq 0$ attains its maximum on $B(\boldsymbol{0},1)$ on the boundary. Up to now, our discussion is valid for an arbitrary parametric function $f(\boldsymbol{x};\boldsymbol{\theta})$. Let's now look at the case of the two-layer neural network under standard initialisation scaling:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{p}}\sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x} + b_j\rangle) \tag{2.11}$$

To simplify the exposition, we will consider both a_j and b_j to be fixed, and let $b_j = 0$ to lighten the notation. As you will show in exercise 3, the argument that follows carries over to the general case. Assuming σ is twice differentiable, the gradient and Hessian of the network with respect to $\boldsymbol{w}_j$ reads:

$$\nabla_{\boldsymbol{w}_j} f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{p}} a_j \sigma'(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle)\boldsymbol{x} \tag{2.12}$$

$$H_{jk}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{w}_j}\nabla_{\boldsymbol{w}_k} f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{p}} a_j \sigma''(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle)\delta_{ij}\boldsymbol{x}\boldsymbol{x}^\top \tag{2.13}$$

Therefore, the Hessian at initialisation $\boldsymbol{H} \in \mathbb{R}^{pd\times pd}$ is a block-diagonal matrix:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{H}_p \end{bmatrix} \tag{2.14}$$

with $\boldsymbol{H}_j \in \mathbb{R}^{d\times d}$ for $j \in [p]$ given by a rank-one matrix:

$$\boldsymbol{H}_j = \frac{1}{\sqrt{p}} a_j^0 \sigma''(\langle \boldsymbol{w}_j^0, \boldsymbol{x}\rangle)\boldsymbol{x}\boldsymbol{x}^\top \tag{2.15}$$

The operator norm of each block is easy to compute:

$$||\boldsymbol{H}_j||_{\text{op}} = \frac{1}{\sqrt{p}}|a_j^0| \cdot |\sigma''(\langle \boldsymbol{w}_j^0, \boldsymbol{x}\rangle)| \cdot ||\boldsymbol{x}||_2^2 \tag{2.16}$$

Hence, the operator norm of the full Hessian at initialisation is given by:

$$||\boldsymbol{H}||_{\text{op}} = \sup_{j\in[p]} ||\boldsymbol{H}||_{\text{op}} = \sup_{j\in[p]} \left[\frac{1}{\sqrt{p}}|a_j^0| \cdot |\sigma''(\langle \boldsymbol{w}_j^0, \boldsymbol{x}\rangle)| \cdot ||\boldsymbol{x}||_2^2\right] \tag{2.17}$$

As discussed in Section 2.1, $a^0 = O(1)$ is typically initialised uniformly in a bounded interval, say $[-1, 1]$. Moreover, σ has often bounded second derivative $\sigma''(z) < M$, e.g. for sigmoid-like functions or ReLU. Therefore:

$$||\boldsymbol{H}||_{\text{op}} \leq \frac{M||\boldsymbol{x}||_2^2}{\sqrt{p}} = O(1/\sqrt{p}) \text{ as } p \to \infty \tag{2.18}$$

Putting this together with eq. (2.9) gives the following remarkable result:

Proposition 1. Let $f(\boldsymbol{x};\boldsymbol{\theta})$ denote a two-layer neural network of width p and activation function σ:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{p}}\sum_{j=1}^{p} a_j\sigma(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle) \tag{2.19}$$

Assume σ has bounded second derivative. Then, under standard initialisation on the second-layer weights $a^0 \sim \text{Unif}([-m, m])$:

$$\sup_{\boldsymbol{\theta}\in B(0,1)} |f(\boldsymbol{x};\boldsymbol{\theta}) - f_{\text{lin.}}(\boldsymbol{x};\theta)| = O(1/\sqrt{p}) \text{ as } p \to \infty. \tag{2.20}$$

This is a quite remarkable result: it tell us that wide two-layer neural networks with standard scaling are close to linear functions!

⚠ Note that while the operator norm of the Hessian vanishes with $p \to \infty$, that's not the case for $f(\boldsymbol{x};\boldsymbol{\theta})$ (as we saw in section 2.1) and for the gradient $\nabla_{\boldsymbol{\theta}} f$. Indeed, for any $j \in [p]$ we have:

$$\sum_{j=1}^{p} ||\nabla_{\boldsymbol{w}_j} f(\boldsymbol{x};\boldsymbol{\theta})||_2^2 = \frac{1}{p}\sum_{j=1}^{p} a_j^2\sigma'(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle)^2||\boldsymbol{x}||_2^2 = O(1). \tag{2.21}$$

Hence the linear approximation itself $f_{\text{lin.}}$ is not a constant function — meaning the linear behaviour is actually non-trivial.

Remark 1. Some remarks are in order.

- This result first appeared in Liu et al. (2020), who also proved a more general result for deep networks.
- Note that in our derivation, we did not use any information about the initialisation of the first layer weights $\boldsymbol{w}_j^0$. Indeed, this is not necessary in the two-layer case. However, it does play an important role in the proof of the deep case.
- Note that this argument breaks down if we add a non-linearity at the last-layer $g(\boldsymbol{x};\boldsymbol{\theta}) = \phi(f(\boldsymbol{x};\boldsymbol{\theta}))$ — see exercise 2 for a discussion.

2.3 Wide networks away from initialisation

We now discuss what happens as we move away from initialisation. For that, we consider a supervised learning problem with training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \mathcal{Y} : i \in [n]\}$. For simplicity of exposition, we will consider the case where the network is trained under gradient flow on the square loss:

$$\begin{aligned}\dot{\boldsymbol{\theta}}(t) &= -\nabla\hat{R}(\boldsymbol{\theta};\mathcal{D})\\ &= \frac{1}{n}\sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta}(t)))\,\nabla f(\boldsymbol{x}_i;\boldsymbol{\theta}(t))\end{aligned} \tag{2.22}$$

where we denote $\dot{u} := {}^{\mathrm{d}u}/_{\mathrm{d}t}$. As the weights $\boldsymbol{\theta}(t)$ change, how does the predictor change? For that, we use the chain rule to write:

$$\frac{\mathrm{d}f(\boldsymbol{x};\boldsymbol{\theta})}{\mathrm{d}t} = \langle \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta}), \dot{\boldsymbol{\theta}} \rangle \tag{2.23}$$

Inserting eq. (2.22) in the above:

$$\frac{\mathrm{d}f(\boldsymbol{x};\boldsymbol{\theta})}{\mathrm{d}t} = -\frac{1}{n}\sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta})) \langle \nabla f(\boldsymbol{x};\boldsymbol{\theta}), f(\boldsymbol{x}_i;\boldsymbol{\theta}) \rangle \tag{2.24}$$

Defining the *empirical neural tangent kernel*:

$$\hat{K}_t(\boldsymbol{x},\boldsymbol{x}') = \langle \nabla f(\boldsymbol{x};\boldsymbol{\theta}(t)), \nabla f(\boldsymbol{x}';\boldsymbol{\theta}(t)) \rangle \tag{2.25}$$

We can rewrite:

$$\frac{\mathrm{d}f(\boldsymbol{x};\boldsymbol{\theta})}{\mathrm{d}t} = -\frac{1}{n}\sum_{i=1}^{n} \hat{K}_t(\boldsymbol{x}_i,\boldsymbol{x}) (y_i - f(\boldsymbol{x}_i;\boldsymbol{\theta})) \tag{2.26}$$

In particular, if we are interested to track the prediction function over the training data, we can define a vector: $\boldsymbol{f} \in \mathbb{R}^n$ with elements $f_i := f(\boldsymbol{x}_i;\boldsymbol{\theta})$ and write the above in a matrix form:

$$\dot{\boldsymbol{f}}(t) = -\frac{1}{n}\hat{\boldsymbol{K}}_t(\boldsymbol{y} - \boldsymbol{f}(t)) \tag{2.27}$$

where we defined the kernel matrix $\hat{\boldsymbol{K}}_t \in \mathbb{R}^{n\times n}$ with elements $\hat{K}_{t,ij} = \hat{K}_t(\boldsymbol{x}_i,\boldsymbol{x}_j)$.

Remark 2. A few comments are in order.

- The above derivation is valid for any parametric function $f(\boldsymbol{x};\boldsymbol{\theta})$. In particular, it is valid for networks with depth $L > 2$.
- Let $p = |\Theta|$ denote the total number of parameters. Then, if $\hat{\boldsymbol{K}}_t$ has rank n (which is possible if $p > n$, for instance), stationary points of the flow consist of interpolators $\boldsymbol{f} = \boldsymbol{y}$.
- Moreover, if at some time $t > T$ the empirical NTK becomes constant $\hat{K}_t \approx \hat{K}$, eq. (2.27) simply describes an exponential relaxation to the interpolator $\boldsymbol{f} = \boldsymbol{y}$, with the relaxation rate given by the largest eigenvalue of $\boldsymbol{K}$.

Now let's go back to our favourite model: the two-layer neural network. Motivated by the discussion in section 2.1, we will consider the NTK initialisation:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{\sqrt{p}}\sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle) \tag{2.28}$$

with Kaiming-like initialisation $a_j(0) \simeq \mathrm{Unif}([-1,1])$, $\boldsymbol{w}_j(0) \sim \mathcal{N}(\boldsymbol{0}, {}^1/_d\boldsymbol{I}_d)$ and for simplicity consider $b_j = 0$. Then, since $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{a})$, the neural tangent kernel in eq. (2.25) explicitly reads:

$$\hat{K}_t(\boldsymbol{x},\boldsymbol{x}') = \frac{1}{p}\sum_{j=1}^{p} \sigma(\langle \boldsymbol{w}_j(t), \boldsymbol{x} \rangle)\sigma(\langle \boldsymbol{w}_j(t), \boldsymbol{x}' \rangle) + \frac{\langle \boldsymbol{x}, \boldsymbol{x}' \rangle}{p}\sum_{j=1}^{p} a_j^2 \sigma'(\langle \boldsymbol{w}(t), \boldsymbol{x} \rangle)\sigma'(\langle \boldsymbol{w}(t), \boldsymbol{x}' \rangle) \tag{2.29}$$

Just as in the argument we made in section 2.1, at initialisation $t = 0$ the empirical NTK kernel $\hat{K}_0$ can be seen as a discretisation of a limiting infinite width kernel. Indeed, by the law of large numbers we have

$$\hat{K}_0(\boldsymbol{x},\boldsymbol{x}') \xrightarrow{a.s.} K_0(\boldsymbol{x},\boldsymbol{x}') := K_{\mathrm{RF}}(\boldsymbol{x},\boldsymbol{x}') + K_{\mathrm{NTK}}(\boldsymbol{x},\boldsymbol{x}'), \text{ as } p \to \infty \tag{2.30}$$

where:

$$K_{\mathrm{RF}}(\boldsymbol{x}, \boldsymbol{x}') \coloneqq \mathbb{E}_{\boldsymbol{w}}\left[\sigma(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)\sigma(\langle \boldsymbol{w}, \boldsymbol{x}'\rangle)\right]$$

$$K_{\mathrm{NTK}}(\boldsymbol{x}, \boldsymbol{x}') \coloneqq \langle \boldsymbol{x}, \boldsymbol{x}'\rangle \mathbb{E}_{a,\boldsymbol{w}}\left[a^2\sigma'(\langle \boldsymbol{w}, \boldsymbol{x}\rangle)\sigma'(\langle \boldsymbol{w}, \boldsymbol{x}'\rangle)\right] \tag{2.31}$$

The kernel K_{RF} is known as the *random features* kernel. Some authors refer to the full K_0 as the *neural tangent kernel* (NTK), while others only the part proportional to the derivative of the activation K_{NTK}.

⚠ This argument only applies to initialisation $t = 0$ at this point. Moreover, note that the choice of initial scaling $1/\sqrt{p}$ is crucial. Also, note that K_0 is quite different from the Gaussian process kernel defined by the wide limit of initialisation in eq. (2.4).

We are now interested in understanding how much does $f(\boldsymbol{x}, \boldsymbol{\theta}(t))$ deviates from $f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))$ along the flow eq. (2.26). For that, define:

$$E(t) = \sup_{\boldsymbol{x}} |f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))| \tag{2.32}$$

and note that at $t = 0$, we have $E(0) = 0$ if $\boldsymbol{\theta}(0) = \boldsymbol{\theta}^0$. Then, we have:

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} \leq \sup_{\boldsymbol{x}} \left| \frac{\mathrm{d}}{\mathrm{d}t}\left[f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))\right]\right| \tag{2.33}$$

Therefore, we need to control:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left[f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))\right] &= \left\langle \nabla f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - \nabla f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t)), \dot{\boldsymbol{\theta}}(t)\right\rangle \\ &= \left\langle \nabla f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - \nabla f(\boldsymbol{x}, \boldsymbol{\theta}^0), \dot{\boldsymbol{\theta}}(t)\right\rangle\end{aligned} \tag{2.34}$$

since $\nabla f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t)) = \nabla f(\boldsymbol{x}, \boldsymbol{\theta}^0)$. Hence, by Cauchy-Schwarz:

$$\left|\frac{\mathrm{d}}{\mathrm{d}t}\left[f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))\right]\right| \leq ||\nabla f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - \nabla f(\boldsymbol{x}, \boldsymbol{\theta}^0)||_2 \cdot ||\dot{\boldsymbol{\theta}}(t)||_2 \tag{2.35}$$

By the mean value theorem, for any $u \in [0, 1]$:

$$||\nabla f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - \nabla f(\boldsymbol{x}, \boldsymbol{\theta}^0)|| \leq ||\nabla^2 f(\boldsymbol{x}, u\boldsymbol{\theta}^0 + (1-u)\boldsymbol{\theta}(t))||_{\mathrm{op}} \cdot ||\boldsymbol{\theta}(t) - \boldsymbol{\theta}^0||_2 \tag{2.36}$$

Therefore, the main ingredient to control $E(t)$ is to ensure that the velocity is bounded $||\boldsymbol{\theta}(t)||_2 \leq V$. Indeed, assuming this is the case implies, by continuity, that $\boldsymbol{\theta}(t)$ cannot move away from $\boldsymbol{\theta}^0$ by a fixed distance for a given time horizon:

$$||\boldsymbol{\theta}(t) - \boldsymbol{\theta}^0||_2 = \left|\left|\int_0^t \dot{\boldsymbol{\theta}}(s)\mathrm{d}s\right|\right|_2 \leq \int_0^t ||\dot{\boldsymbol{\theta}}(s)||\mathrm{d}s = Vt \tag{2.37}$$

In other words, for any time horizon $t \in [0, R/V)$, we have $||\boldsymbol{\theta}(t) - \boldsymbol{\theta}^0||_2 \leq R$. Together with our bound on the Hessian eq. (2.18) this implies that in this time horizon:

$$\left|\frac{\mathrm{d}}{\mathrm{d}t}\left[f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - f_{\mathrm{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t))\right]\right| \leq \frac{VM||\boldsymbol{x}||_2}{\sqrt{p}}||\boldsymbol{\theta}(t) - \boldsymbol{\theta}^0||_2 \tag{2.38}$$

Or in other words:

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} \leq \frac{Vc}{\sqrt{p}}||\boldsymbol{\theta}(t) - \boldsymbol{\theta}^0||_2 \leq \frac{RVc}{\sqrt{p}} \tag{2.39}$$

for some other constant $c > 0$. By Gronwall's inequality:

$$E(t) \leq \underbrace{E(0)}_{=0} + \frac{RVc}{\sqrt{p}} t \tag{2.40}$$

Therefore, as long as $t \in [0, {}^{R}\!/\!{}_{V}]$, we have $E(t) = O({}^{1}\!/\!{}_{\sqrt{p}})$ and $f(\boldsymbol{x}, \boldsymbol{\theta}(t)) \approx f_{\text{lin.}}(\boldsymbol{x}, \boldsymbol{\theta}(t)))$. It remains just to justify why the velocity is bounded along the flow. This is rather intuitive. Indeed, recall that under the flow:

$$\dot{\boldsymbol{f}}(t) = -\frac{1}{n} \hat{\boldsymbol{K}}_t (\boldsymbol{y} - \boldsymbol{f}(t)) \tag{2.41}$$

where $\boldsymbol{K}_t \in \mathbb{R}^{n \times n}$ is the matrix with entries $\hat{K}_{t,ij} = \langle \nabla f(\boldsymbol{x}_i; \boldsymbol{\theta}(t)), \nabla f(\boldsymbol{x}_j; \boldsymbol{\theta}(t)) \rangle$. Since in the interval $t \in [0, {}^{R}\!/\!{}_{V}]$ we have $f \approx f_{\text{lin.}}$, then $\hat{K}_t \approx \hat{K}_0$ since the NTK for the linear model is constant. This means that eq. (2.27) is simply an exponential relaxation. In particular, the speed is controlled by the minimum eigenvalue of the NTK gram matrix:

$$||\boldsymbol{y} - \boldsymbol{f}(t)||_2 \leq e^{-\lambda_{\min}(\hat{\boldsymbol{K}}_0)t} ||\boldsymbol{y} - \boldsymbol{f}(0)||_2 \tag{2.42}$$

Therefore, as long as the minimum eigenvalue of $\hat{\boldsymbol{K}}_0$ is non-zero (i.e. $\hat{\boldsymbol{K}}_0 \succ 0$) — which can be proven for specific activations and distributions of initial conditions $\boldsymbol{\theta}^0$, see Exercise 1 — then gradient flow relaxes exponentially fast, meaning that the velocity $\dot{\boldsymbol{\theta}}$ remains bounded and that t never exceeds ${}^{R}\!/\!{}_{V}$, which closes the argument. This can be formalised in the following theorem, which appeared in Du et al. (2019):

Theorem 1 (Du et al. (2019))**.** Assume $\lambda_0 = \lambda_{\min}(\boldsymbol{K}_0) > 0$. Then, for $p \geq O({}^{n^6}\!/\!{}_{\lambda_0 \delta^3})$, with probability at least $1 - \delta$ we have:

$$||\boldsymbol{y} - \boldsymbol{f}(t)||_2 \leq e^{-\lambda_0 t} ||\boldsymbol{y} - \boldsymbol{f}(0)||_2 \tag{2.43}$$

along the gradient flow in eq. (2.41).

Remark 3 (Generalisations)**.** In this section, we discussed the NTK in the specific case of wide two-layer neural networks trained under gradient flow. However, the results discussed hold on a much more general scope. First, it can be generalise to deeper networks (Jacot et al., 2018; Lee et al., 2019) and other architectures, such as convolutional networks and transformers (Arora et al., 2019; Yang, 2020). Similarly, it can be generalised to other algorithms, such as gradient descent and stochastic gradient descent.

Epilogue

Together, the results in this section tell us that in the standard initialisation from Table 1 (a.k.a. *NTK scaling*), infinite width networks are equivalent to a particular class of kernel methods, where the so-called neural tangent kernels are functions of the architecture at initialisation. This result is striking, and suggests that to understand deep networks, it suffices to understand their NTK at initialisation. But don't be fooled: there is actually a reason why neural networks are used in practice, instead of kernels, and you should regard this result with a pinch of suspicion.

As we discussed in Lecture 1, one of the reasons behind the practical success of neural networks is their capacity to adapt to the data during training, learning relevant features which can help solving particular tasks. On the other hand, despite being powerful, kernels are not adaptive, and generally require a large amount of data to learn functions that lack regularity. This strongly suggests that the NTK cannot be the end of the story.

3 Lazy regime in optimisation

The key step in the argument of Section 2.3 is to show that the loss is minimised in timescales which are much shorter than the time it takes for the parameters to move away from a neighbourhood of initialisation. This behaviour, known as the *lazy regime* of wide neural networks, is more general than the particular context of two-layer neural networks. Indeed, a close inspection reveals that the crucial ingredient in the argument is the scaling of parameters at initialisation. This was noted by Chizat et al. (2019), and we closely follow their argument here.

To formalise this, let's consider a generic parametric model $h : \mathbb{R}^m \to \mathcal{H}$ and cost function $L : \mathcal{H} \to \mathbb{R}_+$, where $\mathcal{H}$ is a Hilbert space. In the following, we will assume both h and F are smooth. We are interested in the following optimation objective $F : \mathbb{R}^m \to \mathbb{R}_+$:

$$F(\boldsymbol{\theta}) \coloneqq L(h(\boldsymbol{\theta})). \tag{3.1}$$

Let $\boldsymbol{\theta}^0 \in \mathbb{R}^m$ denote an initial parameter, which we assume is not a critical point of the objective, i.e. $\nabla F(\boldsymbol{\theta}) \neq 0$. Then, under a single step of gradient descent with learning rate $\eta > 0$, we have:

$$\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 - \eta \nabla F(\boldsymbol{\theta}). \tag{3.2}$$

Define the relative change of the objective function with respect to the gradient step:

$$\Delta(F) \coloneqq \frac{|F(\boldsymbol{\theta}^1) - F(\boldsymbol{\theta}^0)|}{F(\boldsymbol{\theta}^0)}. \tag{3.3}$$

For small enough step-size η, this is effectively controlled by the gradient relative to the initial value of the loss:

$$\Delta(F) = \eta \frac{||\nabla F(\boldsymbol{\theta}^0)||_2}{F(\boldsymbol{\theta}^0)} + o(\eta) \tag{3.4}$$

On the other hand, as we have discussed in eq. (2.9), the rate of change of the features is controlled by the Hessian at initialisation:

$$\Delta(\nabla h) \coloneqq \frac{||\nabla h(\boldsymbol{\theta}^1) - \nabla h(\boldsymbol{\theta}^0)||_2}{||\nabla h(\boldsymbol{\theta}^0)||_2} \leq \eta \frac{||\nabla F(\boldsymbol{\theta})||_2 \cdot ||\nabla^2 h(\boldsymbol{\theta})||_{\text{op}}}{||\nabla h(\boldsymbol{\theta}^0)||_2} \tag{3.5}$$

We say that our model is lazy whenever $\Delta(F) \gg \Delta(\nabla h)$, in other words:

$$\frac{||\nabla F(\boldsymbol{\theta}^0)||_2}{F(\boldsymbol{\theta}^0)} \gg \frac{||\nabla^2 h(\boldsymbol{\theta}^0)||_{\text{op}}}{||\nabla h(\boldsymbol{\theta}^0)||_2} \tag{3.6}$$

For the square loss function where $R(h(\boldsymbol{\theta})) = 1/2||y - h(\boldsymbol{\theta})||^2_{\mathcal{H}}$, this is equivalent to:

$$\kappa_h(\boldsymbol{\theta}_0) \coloneqq ||y - h(\boldsymbol{\theta}^0)||_{\mathcal{H}} \frac{||\nabla^2 h(\boldsymbol{\theta}^0)||_{\text{op}}}{||\nabla h(\boldsymbol{\theta}^0)||_2^2} \ll 1 \tag{3.7}$$

where κ_h is the inverse relative scale of the model h at initialisation. Whenever this is small, the relative change in the loss is faster than the change in the features, meaning that the timescale for the loss to be minimised will be faster than the timescale for the model to exit a linear behaviour, leading to a lazy regime. For an extensive discussion of the lazy regime in different optimisation problems, see Chizat et al. (2019).

Before concluding, it is instructive to go look back at out discussion in Section 2 through the lens of lazy training. Consider our usual two-layer neural network, now introducing a generic scaling factor $\alpha(p)$:

$$h(\boldsymbol{\theta}) = f(\boldsymbol{x}; \boldsymbol{\theta}) = \alpha(p) \sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle) \tag{3.8}$$

For any initialisation where the weights $a_j^0, \boldsymbol{w}_j^0$ are drawn independently and $O(1)$ we can repeat the discussion in Section 2 to show that asymptotically in $p \to \infty$:

$$
\begin{aligned}
|y - h(\boldsymbol{\theta}^0)| &= O(1) \\
||\nabla^2 h(\boldsymbol{\theta}^0)||_{\text{op}} &\sim \alpha(p) \\
||\nabla h(\boldsymbol{\theta}^0)||_2^2 &\sim p\alpha(p)^2
\end{aligned}
\tag{3.9}
$$

and hence:

$$
\kappa_h(\boldsymbol{\theta}^0) \sim \frac{1}{\sqrt{p}} + \frac{1}{p\alpha(p)} \tag{3.10}
$$

For any $\alpha(p)$ decreasing slower than $1/\sqrt{p}$, we indeed have $\kappa_h(p) \to 0$ as $p \to \infty$, leading to a lazy behaviour. The critical scaling to exit the lazy regime is given by $\alpha(p) = O(1/p)$, known in the machine learning literature as *mean-field scaling*.[1] Note this is precisely the scaling of the two-layer neural network in Barron's theorem from Lecture 3, and for which a large width two-layer neural network can be represented as an integral over a probability measure μ over $\mathbb{R}^{p+1}$:

$$
f(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{p}\sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x}\rangle) \approx \int \mu(\mathrm{d}a, \mathrm{d}\boldsymbol{w}) a\sigma(\langle \boldsymbol{w}, \boldsymbol{x}\rangle), \qquad \text{as } p \to \infty. \tag{3.11}
$$

In fact, this is the scaling that maximises the relative change of the features with respect to the loss — or in other words, maximises *feature learning*. In this *feature rich* regime, the dynamics of the network is far from being a simple exponential relaxation to the bottom of a minimum. Indeed, in this regime the gradient flow for infinite width neural networks can be written in terms of a flow for the measure μ_t:

$$
\partial_t \mu_t = \nabla_{\boldsymbol{\theta}} \cdot \left(\mu_t \nabla_{\boldsymbol{\theta}} \hat{R}_n(\mu_t)\right) \tag{3.12}
$$

which is simply a continuity or transport equation for the density μ_t, see (Mei et al., 2018; Chizat and Bach, 2018; Rotskoff and Vanden-Eijnden, 2022; Sirignano and Spiliopoulos, 2020) for a detailed discussion. Analysing this flow is a mathematically much more challenging problem than the NTK gradient flow eq. (2.41) with $\hat{\boldsymbol{K}}_t \approx \hat{\boldsymbol{K}}_0$. In particular, precisely characterising the stationary measure is a hard problem.

Remark 4. While the limiting eq. (3.12) is only well-understood for shallow networks, the equivalent of the mean-field scaling for deep neural networks is known as the μ-parametrisation (μP), see Yang et al. (2022) for a discussion.

4 Exercises

Exercise 1. Consider a two-layer neural network with ReLU activation $\sigma(x) = x_+$:

$$
f(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{\sqrt{p}}\sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}, \boldsymbol{x}\rangle). \tag{4.1}
$$

Assume that the weights are initialised as $a_j^0 \sim \text{Unif}(\{-1, 1\})$, $\boldsymbol{w}^0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$.

(a) Show that the NTK kernel is given by:

$$
\begin{aligned}
K_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') &:= \langle \boldsymbol{x}, \boldsymbol{x}'\rangle \mathbb{E}_{a,\boldsymbol{w}}\left[a^2 \sigma'(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle)\sigma'(\langle \boldsymbol{w}, \boldsymbol{x}_j\rangle)\right] \\
&= \frac{1}{2} - \frac{1}{2\pi}\arccos\left(\frac{\langle \boldsymbol{x}, \boldsymbol{x}'\rangle}{||\boldsymbol{x}||_2 \cdot ||\boldsymbol{x}'||_2}\right)
\end{aligned}
\tag{4.2}
$$

[1]Note that the word *mean-field* is used with different meanings in other contexts in Physics.

(b) Let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote a batch of n independently sampled covariates, and assume $\boldsymbol{x}_i \in B(\mathbf{0}, 1)$. Using Hoeffding's inequality, show that if $p \geq \Omega(\epsilon^{-2} n^2 \log {}^{n}/_{\delta})$, then with probability at least $1 - \delta$ over the random initialisation we have:

$$||\hat{\boldsymbol{K}}_{\text{NTK}} - \boldsymbol{K}_{\text{NTK}}||_{\text{F}} \leq \epsilon \tag{4.3}$$

where $\hat{\boldsymbol{K}}_{\text{NTK}}, \boldsymbol{K}_{\text{NTK}} \in \mathbb{R}^{n \times n}$ with:

$$\begin{aligned} \hat{\boldsymbol{K}}_{\text{NTK},ij} &= \frac{\langle \boldsymbol{x}, \boldsymbol{x}' \rangle}{p} \sum_{j=1}^{p} a_j^2 \sigma'(\langle \boldsymbol{w}_j, \boldsymbol{x}_i \rangle \sigma'(\langle \boldsymbol{w}_j, \boldsymbol{x}_j \rangle), \\ \boldsymbol{K}_{\text{NTK},ij} &= K_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') \end{aligned} \tag{4.4}$$

(c) Conclude that for large enough width p, we have that $\lambda_{\min}(\hat{\boldsymbol{K}}_{\text{NTK}}) > 0$ with high-probability.

Exercise 2. Let $g(f(\boldsymbol{x}; \boldsymbol{\theta})) = \phi(f(\boldsymbol{x}; \boldsymbol{\theta}))$ where ϕ is a twice differentiable function as $f(\boldsymbol{x}; \boldsymbol{\theta})$ a two-layer neural network:

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{\sqrt{p}} \sum_{j=1}^{p} a_j \sigma(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle) \tag{4.5}$$

where σ is a twice-differentiable function.

(a) Considering a_j to be fixed, show that for any $\boldsymbol{\theta}$, the Hessian matrix $\boldsymbol{H}_g$ of g can be related to the Hessian matrix $\boldsymbol{H}(\boldsymbol{\theta})$ of f by:

$$\boldsymbol{H}_g(\boldsymbol{\theta}) = \phi'(f(\boldsymbol{x}; \boldsymbol{\theta}))\boldsymbol{H}(\boldsymbol{\theta}) + \phi''(f(\boldsymbol{x}; \boldsymbol{\theta}))\nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{\theta}) \nabla_{\boldsymbol{w}} f(\boldsymbol{x}; \boldsymbol{\theta})^\top \tag{4.6}$$

(b) Under standard initialisation $a^0 \sim \text{Unif}([-1, 1])$, what is the scaling in p of the operator norm of each of the terms above?

(c) Conclude that $g(\boldsymbol{x}; \boldsymbol{\theta})$ does not linearise as $p \to \infty$.

Exercise 3. Generalise the argument leading to Proposition 1 to the case where the second layer weights a_j are not fixed.

References

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *Advances in neural information processing systems*, 32, 2019.

Avrim Blum and Ronald Rivest. Training a 3-node neural network is np-complete. *Advances in neural information processing systems*, 1, 1988.

Lenaic Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in neural information processing systems*, 31, 2018.

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 32, 2019.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eK3i09YQ.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Chaoyue Liu, Libin Zhu, and Misha Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33:15954–15964, 2020.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

Radford M Neal. Priors for infinite networks. *Bayesian learning for neural networks*, pages 29–53, 1996.

Grant Rotskoff and Eric Vanden-Eijnden. Trainability and accuracy of artificial neural networks: An interacting particle system approach. *Communications on Pure and Applied Mathematics*, 75(9): 1889–1935, 2022.

Justin Sirignano and Konstantinos Spiliopoulos. Mean field analysis of neural networks: A law of large numbers. *SIAM Journal on Applied Mathematics*, 80(2):725–752, 2020.

Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*, 2020.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.