Modelling and Optimization in Machine Learning Project

January 2024

Zsombor Malatinszki

The following project report, accompanied by the two documents „step3.html” and „step7.html” were made for the course „Modelling and Optimization in Machine Learning”, and is based on the MIT course „6.S088 Modern Machine Learning: Simple Methods that Work” by Adityanarayanan Radhakrishnan, Prof Caroline Uhler, Max Luyten, George Stefanakis and Cathy Cai (https://web.mit.edu/modernml/course/).

Specifically, Step 1, 2, 4 and 5 are summaries of lectures 2, 3, 4 and 5 of the „6.S088 Modern Machine Learning: Simple Methods that Work” course, while Step 3, 6, and 7 are implementations of the ideas and theory presented in the aformentioned lectures.

Step 1: Linear Regression

Suppose we are given a data distribution (x,y) ~ P(x,y) where $x \in \mathbb{R}^d$ is a feature vector and $y \in \mathbb{R}$ is the corresponding label. Assume we are given *n* independent, identically distributed (i.i.d.) training pairs $\{x^{(i)}, y^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d \times \mathbb{R}$. Our goal is to learn a function f^ in a class of functions $F = \{f : \mathbb{R}^d \to \mathbb{R}\}$ that is able to "predict well" on unseen samples, i.e. f^(x) ≈ y for x != x(i) for all i ∈ [n].

After framing the general problem above, the following need to be formalized:

1. How do we measure model performance, i.e. how do we assess f^(x) ≈ y ?
2. What class of functions F do we want to use?
3. How do we select a function f^∈ F?

- To address question 1 above, we will utilize the squared Euclidean distance from f^(x) and y, which is typically referred to as mean squared error:

Definition 1 (MSE). *Given $y, \tilde{y} \in \mathbb{R}^p$, the mean squared error (MSE) between $y, \tilde{y}$ is $\mathcal{L}(y, \tilde{y}) = \frac{1}{2}\|y - \tilde{y}\|_2^2$.*

- To address question 2, we will at the moment restrict ourselves to the simple class of linear functions, i.e. $F = \{f : \mathbb{R}^d \to \mathbb{R} \,;\, f(x) = wx \,,\, w \in \mathbb{R}^{1\times d}\}$.

- To address question 3: in order to learn a function $\hat{f}(x) = \hat{w}x$ from data, we find $\hat{w}$ by minimizing:

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - wx^{(i)})^2$$

After learning $\hat{w}$ by minimizing the loss above, given a new sample x, our prediction is just given by $\hat{w}x$.

Definition 2 (Gradient Descent). *Given a loss function $\mathcal{L}(w) : \mathbb{R}^d \to \mathbb{R}$, an initial value $w^{(0)} \in \mathbb{R}^d$, and a learning rate (a.k.a step size) $\eta \in \mathbb{R}$, **gradient descent** is used to minimize the loss $\mathcal{L}(w)$ by iteratively computing*

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}(w^{(t)})$$

for $t \in \mathbb{Z}_+$.

Theorem 1. *Let $\{x^{(i)}, y^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d \times \mathbb{R}$ denote training samples and labels. Let $X = [x^{(1)}|x^{(2)}|\ldots|x^{(n)}]$ and $y = [y^{(1)}, y^{(2)}, \ldots y^{(n)}]$. Let σ_1 be the top singular value of X. Given initialization $w^{(0)} = \mathbf{0}$, gradient descent used to minimize:*

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - wx^{(i)})^2$$

converges to the global minimum $w^{(\infty)} = yX^{\dagger}$, where $X^{\dagger}$ is the Moore-Penrose pseudoinverse of X, iff the learning rate η satisfies $0 < \eta < \frac{2}{\sigma_1^2}$.

Remark about Theorem 1: When $w^{(0)} = \mathbf{0}$, note that $w^{(t)}$ for any $t \in Z+$ is given as a linear combination of training examples. For $\{\alpha_t^{(i)}\}_{i=1}^{n} \subset \mathbb{R}$ and all $t \in Z+$, (based on the proof of Theorem 1) we have:

$$w^{(t)} = \sum_{i=1}^{n} \alpha_t^{(i)} {x^{(i)}}^T$$

This result implies that the output of the trained predictor is always a linear combination of training examples and thus lies in the span of the training data. This will be an extremely useful property for solving kernel regression, where it will appear as the Representer theorem.

Now that we have a closed form for the solution to linear regression, we can understand our solution in the context of sufficiently parameterized (n = d), under-parameterized (n > d), and over-parameterized (n < d) regimes (where is the number of samples and d is the dimensionality of data.

Sufficiently Parameterized Regime ($n = d$)

In the case when n = d (and assuming the rank of X is d), we know that there is exactly one solution to linear regression from linear algebra. In particular, we can achieve zero training loss by solving the following system of equations:

$$wX = y \implies w = yX^{-1}$$

Under-parameterized Regime ($n > d$)

In the case when n > d (and assuming the rank of X is d), we know that there is no interpolating solution (no solution with zero training loss) to linear regression from linear algebra. Instead, a solution that minimizes the MSE is found by setting the gradient of the MSE equal to zero:

$$\nabla_w \mathcal{L}(w) = 0 \implies (y - wX)X^T = 0 \implies w = yX^T(XX^T)^{-1}$$

Here, XX^T is invertible since $n > d$ and the rank of X is d. However, by substituting $X = U\Sigma V^T$ given by SVD, we see that $yX^T(XX^T)^{-1} = yX^{\dagger}$.

Over-parameterized Regime ($n < d$)

There are infinitely many solutions, but gradient descent with $w^{(0)} = 0$ initialization will converge to

$$w^{(\infty)} = yX^{\dagger}$$

Step 2: Kernel Regression

In order to extend linear regression into nonlinear regression, we will simply apply a fixed nonlinear transform to samples *x* before performing linear regression. Formally, we will consider the set of class of functions

$$\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R} \; ; \; f(x) = \langle w, \psi(x) \rangle_{\mathcal{H}} \; , \; \psi : \mathbb{R}^d \to \mathcal{H} \; , \; w \in \mathcal{H}\} \; ;$$

where $\mathcal{H}$ is a Hilbert space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and ψ is a nonlinear feature map.

Proposition 1. *Let $\{x^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d$ and $\{y^{(i)}\}_{i=1}^{n} \subset \mathbb{R}$. Then there exist $\{\alpha_i\}_{i=1}^{n} \subset \mathbb{R}$, such that the minimum ℓ_2 norm minimizer, w^*, for the loss:*

$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - w x^{(i)})^2 \quad ;$$

has the form

$$w^* = \sum_{i=1}^{n} \alpha_i {x^{(i)}}^T \quad ;$$

Theorem 1 (Representer Theorem). *Let $\mathcal{H}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Let $\{\psi(x^{(i)})\}_{i=1}^{n} \subset \mathcal{H}$ and $\{y^{(i)}\}_{i=1}^{n} \subset \mathbb{R}$. Then there exist $\{\alpha_i\}_{i=1}^{n} \subset \mathbb{R}$, such that the minimum $\mathcal{H}$-norm minimizer, w^*, for the loss:*

$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \langle w, \psi(x^{(i)}) \rangle_{\mathcal{H}})^2 \quad ; \tag{1}$$

lies in the span of the samples $\{\psi(x^{(i)})\}_{i=1}^{n}$, i.e.

$$w^* = \sum_{i=1}^{n} \alpha_i \psi(x^{(i)}) \quad ;$$

We will now use the result of Theorem 1 to convert the seemingly intractable problem of minimizing the loss in Eq. (1) to solving a finite dimensional linear regression problem. In particular, we substitute $w = \sum_{i=1}^{n} \alpha_i \psi(x^{(i)})$ to simplify the loss in Eq. (1) as follows:

$$\begin{aligned}
\mathcal{L}(w) &= \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \langle w, \psi(x^{(i)}) \rangle_{\mathcal{H}})^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \langle \sum_{j=1}^{n} \alpha_j \psi(x^{(j)}), \psi(x^{(i)}) \rangle_{\mathcal{H}})^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \left(y^{(i)} - \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_n \end{bmatrix} \begin{bmatrix} \langle \psi(x^{(1)}), \psi(x^{(i)}) \rangle_{\mathcal{H}} \\ \langle \psi(x^{(2)}), \psi(x^{(i)}) \rangle_{\mathcal{H}} \\ \vdots \\ \langle \psi(x^{(n)}), \psi(x^{(i)}) \rangle_{\mathcal{H}} \end{bmatrix} \right)^2
\end{aligned} \tag{2}$$

Therefore, instead of minimizing $\mathcal{L}(w)$ over all w, we minimize the loss $\mathcal{L}$ with respect to the parameters $\{\alpha_i\}_{i=1}^{n}$. Theorem 1 implies that identifying these α_i's will yield the minimum $\mathcal{H}$-norm solution that minimizes the loss in Eq. (1).

Importantly, Eq. (2) implies that we need only know the inner products $\langle \psi(x^{(i)}), \psi(x^{(j)}) \rangle_{\mathcal{H}}$ for all $i, j \in [n]$ to perform linear regression in a Hilbert space. Moreover, we do not even need to know the map ψ, but rather the functional that yields the required inner products. Namely, we only need some function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that $K(x, \tilde{x}) = \langle \psi(x), \psi(\tilde{x}) \rangle_{\mathcal{H}}$. This is formalized by the notion of kernels.

Definition 1 (Kernel)**.** *Given nonempty set $\mathcal{X}$, a **kernel** is a symmetric continuous function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.*

Note that requiring K to have this inner product form introduces constraints, e.g. $K(x, x) \geq 0$. Thus, we will consider kernels that satisfy the positive semi-definite constraint as defined below.

Definition 2 (Positive semi-definite kernel)**.** *Given nonempty set $\mathcal{X}$, a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is **positive semi-definite** iff for any $\{x^{(i)}\}_{i=1}^n \subset \mathcal{X}$ and for any $\{c_i\}_{i=1}^n \subset \mathbb{R}$,*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x^{(i)}, x^{(j)}) \geq 0.$$

Proposition 2. *Let $\mathcal{H}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Let $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that $K(x, \tilde{x}) = \langle \psi(x), \psi(\tilde{x}) \rangle_{\mathcal{H}}$ for $\psi : \mathbb{R}^d \to \mathcal{H}$. Then K is a positive semi-definite kernel.*

We here provide some classical examples of positive semi-definite kernels.

1. **Linear Kernel.** $K(x, \tilde{x}) = x^T \tilde{x}$.
2. **Gaussian (RBF) Kernel.** $K(x, \tilde{x}) = \exp\left(-L\|x - \tilde{x}\|_2^2\right)$ for $L \in \mathbb{R}_+$.
3. **Laplace Kernel.** $K(x, \tilde{x}) = \exp\left(-L\|x - \tilde{x}\|_2\right)$ for $L \in \mathbb{R}_+$.

Theorem 2 (Kernel Regression)**.** *Let $\mathcal{H}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. Let $\psi : \mathbb{R}^d \to \mathcal{H}$ and let $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel function such that $K(x, \tilde{x}) = \langle \psi(x), \psi(\tilde{x}) \rangle_{\mathcal{H}}$. The minimum $\mathcal{H}$-norm minimizer of the loss:*

$$\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \langle w, \psi(x^{(i)}) \rangle_{\mathcal{H}})^2$$

is given by $w^* = \sum_{i=1}^n [y\hat{K}^{\dagger}]_i \psi(x^{(i)})$ where $\hat{K} \in \mathbb{R}^{n \times n}$ is the positive semi-definite matrix with entries $\hat{K}_{i,j} = K(x^{(i)}, x^{(j)})$. Moreover, the corresponding predictor, $\hat{f} : \mathbb{R}^d \to \mathbb{R}$, is given by

$$\hat{f}(x) = y\hat{K}^{\dagger}\hat{K}(X, x) \ ;$$

where $\hat{K}(X, x) \in \mathbb{R}^n$ with entries $\hat{K}(X, x)_i = K(x^{(i)}, x)$.

Steps of computing kernel regression:

Step 1: *Compute the kernel (Gram) matrix $\hat{K} \in \mathbb{R}^{10 \times 10}$ where $\hat{K}_{i,j} = K(x^{(i)}, x^{(j)})$.*

Step 2: *Solve for the coefficient vector α by solving the linear system of equations $y = \alpha\hat{K}$.*

Step 3: *For any new sample x, compute the prediction given by $\hat{f}(x) = \alpha\hat{K}(X, x)$ where $\hat{K}(X, x)_i = K(x^{(i)}, x)$.*

Step 3: Recreating Kernel Figures

Please refer to „step3.html”

Step 4: NNGP

Example 1. *Let $X = [x^{(1)}, x^{(2)}, \ldots, x^{(n)}] \in \mathbb{R}^{d \times n}$ denote training samples and $y = [y^{(1)}, y^{(2)}, \ldots, y^{(n)}] \in \mathbb{R}^{1 \times n}$ denote training labels. Given $A \in \mathbb{R}^{1 \times k}, B \in \mathbb{R}^{k \times d}$ and $\phi : \mathbb{R} \to \mathbb{R}$ an elementwise activation function, let $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(x) = A\phi(Bx)$ denote a 1 hidden layer neural network. Suppose $B_{i,j} \overset{i.i.d.}{\sim} \mathcal{P}$ for some probability distribution $\mathcal{P}$ and is fixed. Then, using gradient descent with $A^{(0)} = \mathbf{0}$ to minimize the loss:*

$$\mathcal{L}(A) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}))^2 = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - A\phi(Bx^{(i)}))^2$$

is equivalent to using kernel regression with the kernel $\Sigma(x, \tilde{x}) = \langle \phi(Bx), \phi(B\tilde{x}) \rangle$ to map from X to y.

Hence for any fixed finite width k, we can construct the kernel by computing the product $\langle \phi(Bx), \phi(B\tilde{x}) \rangle$. Interestingly, under certain conditions on the initialization, we can tractably compute the inner product even in the limit as $k \to \infty$, and the resulting kernel is called the Neural Network Gaussian Process (NNGP).

Definition 1. *Given $A \in \mathbb{R}^{1 \times k}, B \in \mathbb{R}^{k \times d}$ and $\phi : \mathbb{R} \to \mathbb{R}$ an elementwise activation function, let $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(x) = \frac{1}{\sqrt{k}} A\phi(Bx)$ denote a 1 hidden layer neural network. If $B_{i,j} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$, then the* ***Neural Network Gaussian Process (NNGP)*** *is given by the kernel $\Sigma(x, \tilde{x}) = \lim_{k \to \infty} \left[\frac{1}{k}\langle \phi(Bx), \phi(B\tilde{x}) \rangle\right]$.*

The reason for including the $\frac{1}{\sqrt{k}}$ scaling factor in Definition 1 is such that the kernel $\Sigma(x, \tilde{x})$ can be evaluated using the law of large numbers as follows.

Proposition 1. *Under the setting of Definition 1, $\Sigma(x, \tilde{x}) = \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda)}[\phi(u)\phi(v)]$ where*

$$\Lambda = \begin{bmatrix} \|x\|_2^2 & x^T\tilde{x} \\ x^T\tilde{x} & \|\tilde{x}\|_2^2 \end{bmatrix}.$$

A list of activation functions for which the NNGP kernel has a closed form: ReLU, Leaky ReLU, GeLU, Sine, Cosine, Error function (erf), Hermite polynomials.

Given that we know $\Sigma(x, \tilde{x})$ if we can evaluate the expectation in Proposition 1, we now present tools that are useful for calculating this expectation. In particular, when data lie on the unit sphere, $\mathcal{S}^{d-1}$, the expectation we wish to compute is referred to as the dual activation.

Definition 2. *Let $\phi(x) : \mathbb{R} \to \mathbb{R}$ be an activation function. Then the* ***dual activation*** *of ϕ is given by $\check{\phi} : [-1, 1] \to \mathbb{R}$ where:*

$$\check{\phi}(\xi) = \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda)}[\phi(u)\phi(v)] \quad ; \quad \Lambda = \begin{bmatrix} 1 & \xi \\ \xi & 1 \end{bmatrix}.$$

In particular, we have the following explicit connection between the NNGP and the dual activation, when data lies on the unit sphere.

Corollary 1. *Under the setting of Definition 1, if $x, \tilde{x} \in \mathcal{S}^{d-1}$ and $\xi = x^T\tilde{x}$, then $\Sigma(x, \tilde{x}) = \check{\phi}(\xi)$.*

In order to simplify the expectation term in the dual activation, we turn to identifying a basis for the functions ϕ for which the expectations are easy to compute. In particular, we will show that the expectation term in the dual activation is easy to compute for the Hermite polynomials, which are defined below.

Definition 3 (Probabilist's Hermite Polynomial)**.** *The Hermite polynomials $\{h_i(x)\}_{i=0}^{\infty}$ are an orthonormal basis for the Hilbert space, $L^2(\mu)$, with inner product $\langle f, g\rangle = \int_{\mathbb{R}} f(x)g(x)e^{-\frac{x^2}{2}}dx$ constructed by performing Gram-Schmidt orthogonalization in $L^2(\mu)$ using the polynomials $\{x^i\}_{i=0}^{\infty}$.*

We list the first few probabilisit's Hermite polynomials below.

Example 3. $h_0(x) = 1, h_1(x) = x, h_2(x) = \frac{x^2-1}{\sqrt{2!}}, h_3(x) = \frac{x^3-3x}{\sqrt{3!}}$.

The dual of of a Hermite polynomial is convenient to compute as follows.

Proposition 2. *If $\phi(x) = h_n(x)$ for $x \in \mathcal{S}^{d-1}$, then $\check{\phi}(\xi) = \xi^n$ for $\xi \in [-1, 1]$.*

Theorem 1. *Let $\phi \in L^2(\mu)$ such that $\phi(x) = \sum_{n=0}^{\infty} a_n h_n(x)$ for $x \in \mathcal{S}^{d-1}$. Then, $\check{\phi}(\xi) = \sum_{n=0}^{\infty} a_n^2 \xi^n$ for $\xi \in [-1, 1]$.*

Theorem 1 implies the following key properties of the NNGP that would be nontrivial to observe otherwise.

Corollary 2. *Let $\phi \in L^2(\mu)$ be an activation function and let $\check{\phi}$ denote the dual activation. Then, $\check{\phi}$ satisfies the following:*

(a) $\check{\phi}$ is non-decreasing and convex in $[0, 1]$.

(b) $(\check{\phi})' = (\check{\phi'})$ (the dual commutes with differentiation).

(c) $\check{\phi}$ is continuous in $[-1, 1]$ and smooth (infinitely differentiable) in $(-1, 1)$.

(d) The range of $\check{\phi}$ is given by $[-\|\phi\|_{L^2(\mu)}^2, \|\phi\|_{L^2(\mu)}^2]$.

The curious case of double descent, the benefit of over-parameterization:

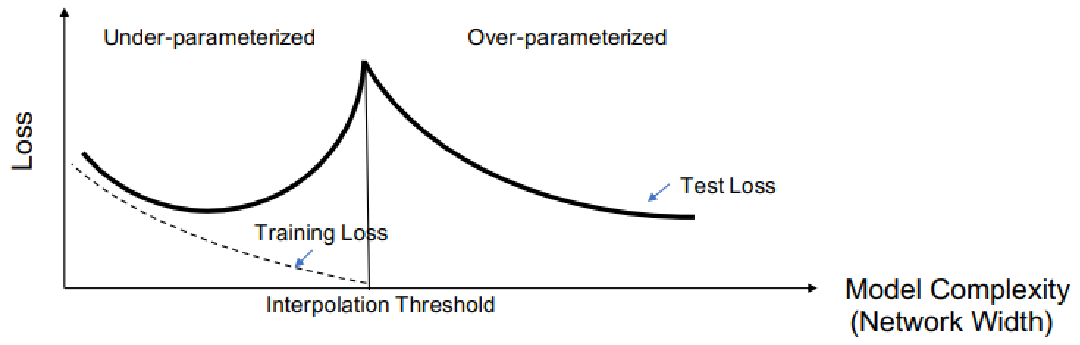

Figure 3: A schematic depicting the double descent phenomenon shown empirically in Fig. 2c. In particular, in the the under-parameterized (or classical) regime, we observe traditional overfitting: we are unable to interpolate the data and as the training loss decreases, the test loss increases. In the over-parameterized (or modern) setting, all models perfectly fit the training data, but the test loss remarkably decreases as the model complexity (for our purposes, neural network width increases. Additionally, as observed in Fig. 2, the loss is maximized at the *interpolation threshold*, which occurs when the model complexity is near the number of training examples.

Step 5: NTK

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,

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

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:*

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

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:*

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

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 $x^{(i)}$.

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. 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.

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 property does not hold for neural networks with arbitrary layer structure, e.g. fully convolutional networks.

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:

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

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 .

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 $m \in \mathbb{R}$:*

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

Step 6: NTK vs. NN Proposal

Neural Network:

I propose a simulation that is in some ways similar to what is discussed in the empirical demonstartion of Lecture 5, namely: a one hidden layer neural network, with an activation function of ReLU. The mathematical notation for this neural network is the following:

Let $X \in R^d$ denote *d* samples of real numbers, where each sample x is drawn from a uniform distribution on the unit sphere in R^1 , which corresponds to the interval [-1, 1]. Let $g(x) : R \rightarrow R$ such that $g(x) = x + \sin(10x)$ and let $y = f(X) \in R^d$ such that $y_i = g(X_i)$. Given these *d* samples, the goal is to recover g(x). In particular, the class of 1 hidden layer neural networks of width *k* is given by:

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

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

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

for varying widths k.

NTK:

For the ReLU activation function, if $x, \tilde{x} \in \mathcal{S}^{d-1}$ and $\xi = x^T\tilde{x}$, then the NTK is given by

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

,

as seen in Example 2 of Lecture 5.

Implementation:

Important properties that must be satisfied:

- A and B should be initialized i.i.d. N(0,1)
- The scaling factor of sqrt(2)/sqrt(k) must be included
- The datapoints should be on the unit sphere, $x, \tilde{x} \in \mathcal{S}^{d-1}$

Based on the lectures covered in the previous steps, I will show through a numerical simulation that as the layer-wise widths approach infinity, solving kernel regression with the NTK is equivalent to training all layers of a neural network.

Step 7: Implementation

Please refer to “step7.html”