Optimization error for Deep Learning

김지수 (Jisu KIM)

딥러닝의 통계적 이해 (Deep Learning: Statistical Perspective), 2025 2nd Semester (fall)

This lecture note is a combination of Prof. Joong-Ho Won's "Deep Learning: Statistical Perspective" with other lecture notes. Main references are:

Tong Zhang, Mathematical Analysis of Machine Learning Algorithms, https://tongzhang-ml.org/lt-book.html

Matus Telgarsky, Deep learning theory lecture notes, https://mjt.cs.illinois.edu/dlt/

Weinan E, Chao Ma, Stephan Wojtowytsch, Lei Wu, Towards a Mathematical Understanding of Neural Network-Based Machine Learning: what we know and what we don't, https://arxiv.org/abs/2009.10713/

Namjoon Suh, Guang Cheng, A Survey on Statistical Theory of Deep Learning: Approximation, Training Dynamics, and Generative Models, 2024. https://arxiv.org/abs/2401.07187/

1 Review

1.1 Basic Model for Supervised Learning

- Input(입력) / Covariate(설명 변수) : $x \in \mathbb{R}^d$, so $x = (x_1, \ldots, x_d)$.
- Output(출력) / Response(반응 변수): $y \in \mathcal{Y}$. If y is categorical, then supervised learning is "classification", and if y is continuous, then supervised learning is "regression".
- Model(모형) :
$$y \approx f(x).$$
If we include the error ϵ to the model, then it can be also written as
$$y = \phi(f(x), \epsilon).$$
For many cases, we assume additive noise, so
$$y = f(x) + \epsilon.$$
- Assumption(가정): f belongs to a family of functions $\mathcal{M}$. This is the assumption of a model: a model can be still used when the corresponding assumption is not satisfied in your data.
- Loss function(손실 함수): $\ell(y, a)$. A loss function measures the difference between estimated and true values for an instance of data.
- Training data(학습 자료): $\mathcal{T} = \{(y_i, x_i), i = 1, \ldots, n\}$, where (y_i, x_i) is a sample from a probability distribution P_i. For many cases we assume i.i.d., or x_i's are fixed and y_i's are i.i.d..
- Goal(목적): we want to find f that minimizes the expected prediction error,
$$f^0 = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(Y,X)\sim P} \left[\ell(Y, f(X))\right].$$
Here, $\mathcal{F}$ can be different from $\mathcal{M}$; $\mathcal{F}$ can be smaller then $\mathcal{M}$.
- Prediction model(예측 모형): f^0 is unknown, so we estimate f^0 by $\hat{f}$ using data. For many cases we minimizes on the empirical prediction error, that is taking the expectation on the empirical distribution $P_n = \frac{1}{n}\sum_{i=1}^n \delta_{(Y_i,X_i)}$.
$$\hat{f} = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{P_n} \left[\ell(Y, f(X))\right] = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f(X_i)).$$
- Prediction(예측): if $\hat{f}$ is a predicted function, and x is a new input, then we predict unknown y by $\hat{f}(x)$.

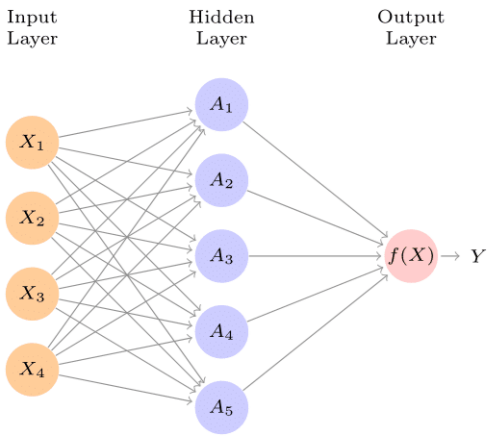

Figure 1: Neural network with a single hidden layer. The hidden layer computes activations $A_j = \sigma_j(x)$ that are nonlinear transformations of linear combinations of the inputs $x_1, \ldots, x_d$. Hence these A_j are not directly observed. The functions σ_j are not fixed in advance, but are learned during the training of the network. The output layer is a linear model that uses these activations A_j as inputs, resulting in a function $f(x)$. Figure 10.1 from [2].

1.2 Two Layer Neural Networks

A two-layer neural network takes an input vector of d variables $x = (x_1, x_2, \ldots, x_d)$ and builds a nonlinear function $f(x)$ to predict the response $y \in \mathbb{R}^D$. What distinguishes neural networks from other nonlinear methods is the particular structure of the model:

$$f(x) = f_\theta(x) = g\left(\beta_0 + \sum_{j=1}^{m} \beta_j \sigma(b_j + w_j^\top x)\right),$$

where $x \in \mathbb{R}^d, b_j \in \mathbb{R}, w_j \in \mathbb{R}^d, \beta_0 \in \mathbb{R}^D, \beta_j \in \mathbb{R}^D$. See Figure 1.

- $\theta = \{[\beta, a_j, b_j, w_j] : j = 1, \ldots, m\}$ denotes the set of model parameters.
- $x_1, \ldots, x_d$ together is called an input layer.
- $A_j \coloneqq \sigma_j(x) = \sigma(b_j + w_j^\top x)$ is called an activation.
- $A_1, \ldots, A_m$ together is called a hidden layer or hidden unit; m is the number of hidden nodes.
- $f(x)$ is called an output layer.
- g is an output function. Examples are:
  - softmax $g_i(x) = \exp(x_i)/\sum_{l=1}^{D} \exp(x_l)$ for classification. The softmax function estimates the conditional probability $g_i(x) = P(y = i|x)$.
  - identity/linear $g(x) = x$ for regression.
  - threshold $g_i(x) = I(x_i > 0)$
- σ is called an activation function. Examples are:
  - sigmoid $\sigma(x) = 1/(1 + e^{-x})$ (see Figure 2)
  - rectified linear (ReLU) $\sigma(x) = \max\{0, x\}$ (see Figure 2)
  - identity/linear $\sigma(x) = x$
  - threshold $\sigma(x) = I(x > 0)$, threshold gives a direct multi-layer extension of the perceptron (as considered by Rosenblatt).

Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model. So the activations are like derived features - nonlinear transformations of linear combinations of the features.

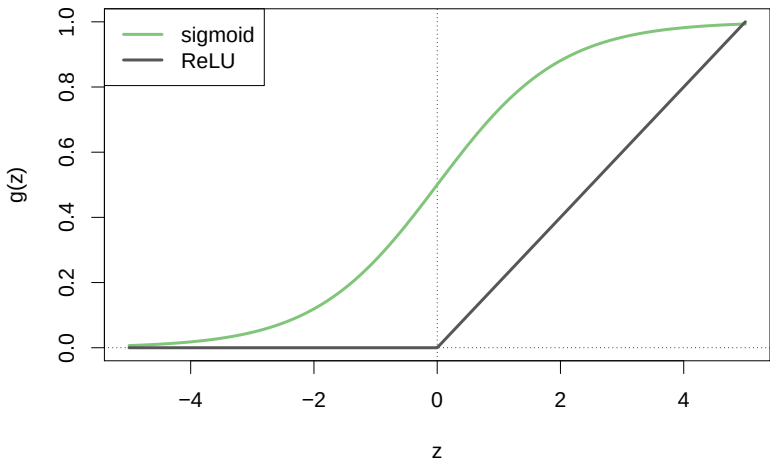

Figure 2: Activation functions. The piecewise-linear ReLU function is popular for its efficiency and computability. We have scaled it down by a factor of five for ease of comparison. Figure 10.2 from [2].

2 Notation and Goal

From here, we only consider regression problem, so $g(x) = x$. We assume $\beta_0 = 0$. Hence, for the two-layer neural network with the width of the hidden layer m and activation function σ, the function space we consider is

$$\mathcal{F}_{m,\sigma} = \left\{ f_\theta : f_\theta(x) = \sum_{j=1}^{m} \beta_j \sigma(b_j + w_j^\top x) \right\},$$

$$\mathcal{F}_{m,\sigma} = \left\{ f_\theta : f_\theta(x) = \sum_{j=1}^{m} \beta_j \sigma(w_j^\top x) \right\},$$

and if we consider all two-layer neural network with arbitrary width, then

$$\mathcal{F}_\sigma = \bigcup_{m=1}^{\infty} \mathcal{F}_{m,\sigma} = \left\{ f_\theta : f_\theta(x) = \sum_{j=1}^{m} \beta_j \sigma(b_j + w_j^\top x),\ m \in \mathbb{N} \right\}.$$

$$\mathcal{F}_\sigma = \bigcup_{m=1}^{\infty} \mathcal{F}_{m,\sigma} = \left\{ f_\theta : f_\theta(x) = \sum_{j=1}^{m} \beta_j \sigma(w_j^\top x),\ m \in \mathbb{N} \right\}.$$

Suppose the true regression function f_* is in a function class $\mathcal{M}$, so

$$y \approx f_*(x), \qquad f_* \in \mathcal{M}.$$

Suppose are using the ℓ_2-loss, so we find f among deep neural network class $\mathcal{F}$ that minimizes the expected risk (평균위험),

$$f^0 = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(Y,X)\sim P} \left[(y - f(x))^2 \right].$$

f_0 is the expected risk mimizing function (평균위험최소함수). And we estimate f^0 by $\hat{f}$ using data by minimizes on the empirical risk (경험위험) on training dataset, so

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2. \tag{1}$$

$\hat{f}$ is the empirical risk mimizing function (경험위험최소함수). And we set $\tilde{f}$ be the approximation of $\hat{f}$ by optimization(최적화); $\tilde{f}$ is the learned function (학습된 함수).

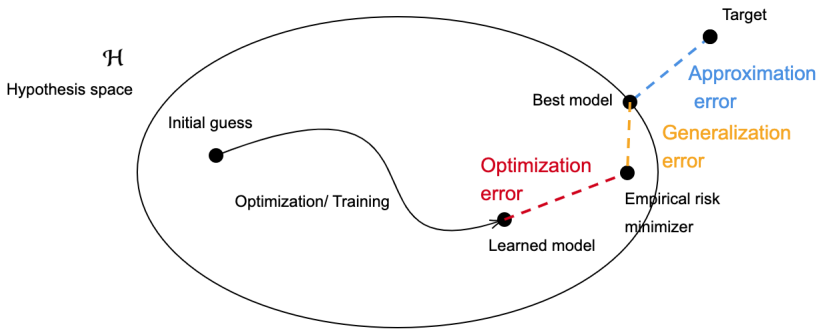

Figure 3: Diagram representing the learning procedure, the three main paradigms and their corresponding errors. Figure 2 from https://dcn.nat.fau.eu/breaking-the-curse-of-dimensionality-with-barron-spaces/.

So there are three sources of errors: approximation error, generalization error, and optimization error. See Figure 3.

$$f_* - \tilde{f} = \underbrace{f_* - f^0}_{\text{approximation error}} + \underbrace{f^0 - \hat{f}}_{\text{generalization error}} + \underbrace{\hat{f} - \tilde{f}}_{\text{optimization error}} .$$

Here, we focus on optimization error.

3 Training dynamics-based statistical guarantees

In the approximation and generalization lecture notes, we assumed that the global minimizer of the empirical risk, $\hat{f}$ in (1), is obtainable. However, due to the non-convex nature of the loss function, neural networks estimated using commonly employed gradient-based methods lack guarantees of finding $\hat{f}$, which leads to the following natural question:

Does the neural network estimated by gradient-based methods generalize well?

We will try to answer to the above question. Due to the complex nature of the problem, most work have considered the suggested shallow neural network: (i.e., networks with one hidden layer). Assume $b_j = 0$, and with a slightly different parameterrization, we consider the following functions:

$$f_\theta(x) = \frac{\alpha}{m} \sum_{j=1}^{m} a_j \sigma(w_j^\top x),$$

and the corresponding function class becomes

$$\mathcal{F}_{m,\sigma} = \left\{ f_\theta : f_\theta(x) = \frac{\alpha}{m} \sum_{j=1}^{m} a_j \sigma(w_j^\top x) \right\}.$$

The network dynamic is scaled with the factor $\frac{\alpha}{m}$. If the network width (i.e., m) is small, the scaling factor has negligible effects on the network dynamics. But for the wide enough networks (i.e., overparametrized setting), the scaling difference yields completely different behaviors in the dynamics. Given the m is large enough, we will focus on two specific regimes:

1. Neural Tangent Kernel regime: with $\alpha = \sqrt{m}$.
2. Mean Field regime: with $\alpha = 1$.

We consider the ℓ_2-loss function: $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$. The model parameter pairs θ are updated through the gradient-based methods. Let $\theta_{(0)}$ be the initialized weight pairs. Then, we have the following gradient descent (GD) update rule with step-size $\eta > 0$ and $k \geq 1$:

$$\text{GD}: \qquad \theta_{(k)} = \theta_{(k-1)} - \eta \nabla_\theta \mathcal{L}(\theta)|_{\theta=\theta_{(k)}}. \tag{2}$$

Another celebrated gradient-based method is Stochastic Gradient Descent (SGD). The algorithm takes a randomly sampled subset of the data, computes the gradient with the selected samples, and this significantly reduces the computational burdens in GD. Another frequently adopted algorithm in practice is Noisy Gradient Descent (NGD), which adds the centered Gaussian noise to the gradient of loss function in (2). It is known that adding noises to gradient helps training and generalization of neural networks.

4 Approximation near initialization and the Neural Tangent Kernel

In this section we consider networks close to their random initialization. We fix $\alpha = \sqrt{m}$, treat a_j as fixed and only allow w_j's to vary. We parametrize with W as

$$f(x; W) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(w_j^\top x), \qquad \text{where } W = \begin{pmatrix} w_1^\top \\ \vdots \\ w_m^\top \end{pmatrix} \in \mathbb{R}^{m \times d},$$

where σ will either be a smooth activation or the ReLU. Briefly, the core idea is to compare a network $f : \mathbb{R}^d \times \mathbb{R}^{m \times d} \to \mathbb{R}$, which takes input $x \in \mathbb{R}^d$ and has parameters $W \in \mathbb{R}^{m \times d}$, to its first-order Taylor approximation at random initialization W_0:

$$\tilde{f}(x; W) \coloneqq f(x, W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle .$$

The key property of this simplification is that while it is nonlinear in x, it is affine in W, which will greatly ease analysis.

4.1 Basic setup: Taylor expansion of shallow networks

Now let's consider the corresponding first-order Taylor approximation $\tilde{f}$ in detail. Consider any univariate activation σ which is differentiable except on a set of measure zero (e.g., countably many points), and Gaussian initialization $W_0 \in \mathbb{R}^{m \times d}$ as before. Consider the Taylor expansion at initialization:

$$\begin{aligned} \tilde{f}(x; W) &= f(x; W_0) + \langle \nabla f(x; W_0), W - W_0 \rangle \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(w_{0,j}^\top x) + \sigma'(w_{0,j}^\top x) x^\top (w_j - w_{0,j}) \right) \\ &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) w_{0,j}^\top x \right] + \sigma'(w_{0,j}^\top x) w_j^\top x \right) . \end{aligned}$$

If σ is nonlinear, then this mapping is nonlinear in x, , despite being affine in W. Indeed $\nabla f(\cdot; W_0)$ defines a feature mapping:

$$\nabla f(x; W_0) \coloneqq \begin{pmatrix} a_1 \sigma'(w_{0,1}^\top x) x^\top \\ \vdots \\ a_m \sigma'(w_{0,m}^\top x) x^\top \end{pmatrix};$$

$$\nabla f(x; W_0) \coloneqq \frac{1}{\sqrt{m}} \begin{pmatrix} a_1 \sigma'(w_{0,1}^\top x) x^\top \\ \vdots \\ a_m \sigma'(w_{0,m}^\top x) x^\top \end{pmatrix};$$

the predictor $\tilde{f}$ an affine function of the parameters, and is also affine in this feature-mapped data.

Remark 1. The factor $1/\sqrt{m}$ will make the most sense in the following sections, it gives a normalization that leads to a kernel. We only vary the inner layer W and keep the outer layer a fixed to have a nontrivial (nonlinear) model which still leads to non-convex training, but is arguably the simplest such. Random initialization is classical and used for many reasons, a classical one being a "symmetry break" which makes nodes distinct and helps with training.

Remark 2. "NTK regime" or "near initialization" are not well-defined, though generally the proofs in this setup require some combination of $\|W - W_0\|_F = O(1)$, and/or at most $1/\sqrt{m}$ fraction of the activations change. In practice, these all seem to be violately almost immediately (e.g., just one or two steps of gradient descent), but still the idea captures many interesting phenomena near initialization and do not degrade with overparameterization as do other approaches.

Remark 3. There are a few reasons why we do the Taylor expansion around initialization; the main one is that Taylor approximation improves the closer you get to the point you are approximating, another one is that bounds that scale with $\|W\|_F$ can be re-centered to now scale with the potentially much smaller quantity $\|W - W_0\|_F$, and lastly we get to invoke Gaussian concentration tools. Note however how things completely break down if we do what might initially seem a reasonable alternative: Taylor expansion around 0. Then we get

$$f(x;0) + \langle \nabla f(x;0), W - 0 \rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\sigma(0) + \sigma'(0) x^\top w_j \right).$$

This is once again affine in the parameters, but it is also affine in the inputs. So we don't have any of the usual power of neural networks.

Remark 4. If we use the ReLU $\sigma(z) = \max\{0, z\}$, then the property $\sigma(z) = z\sigma'(z)$ (which is fine even at 0) means

$$\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) w_{0,j}^\top x = 0,$$

and thus $\tilde{f}$ as above simplifies to give

$$\begin{aligned}
\tilde{f}(x;W) &= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \left(\left[\sigma(w_{0,j}^\top x) - \sigma'(w_{0,j}^\top x) w_{0,j}^\top x \right] + \sigma'(w_{0,j}^\top x) w_j^\top x \right). \\
&= \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma'(w_{0,j}^\top x) w_j^\top x = \langle \nabla f(x; W_0), W \rangle .
\end{aligned}$$

4.2 Networks near initialization are almost linear

Our first step is to show that $f - f_0$ shrinks as m increases, which has a few immediate consequences.

- It gives one benefit of "overparameterization."
- It gives us an effective way to do universal approximation with small $\|W - W_0\|$: we simply make m as large as needed and get more functions inside our RKHS.

First we handle the case that σ is L-smooth, by which we mean σ'' exists and satisfies $|\sigma''| \le L$ everywhere. This is not satisfied for the ReLU, but the proof is so simple that it is a good motivator for other cases.

Lemma 5. *If $\sigma : \mathbb{R} \to \mathbb{R}$ is L-smooth, and $|a_j| \le 1$, and $\|x\|_2 \le 1$, then for any parameters $W, V \in \mathbb{R}^{m \times d}$,*

$$\cancel{\left| f(x;W) - \tilde{f}(x;V) \right| \le \frac{L}{2\sqrt{m}} \|W - V\|_F^2 .}$$

$$\left| f(x;W) - \tilde{f}_V(x;W) \right| \le \frac{L}{2\sqrt{m}} \|W - V\|_F^2 .$$

Proof. By Taylor's theorem,

$$|\sigma(r) - \sigma(s) - \sigma'(s)(r - s)| = \left| \int_r^s \sigma''(z)(s - z) dz \right| \le \frac{L(r-s)^2}{2}.$$

Therefore,

$$\begin{aligned}
&|f(x;W) - f(x;V) - \langle \nabla f(x;V), W - V \rangle| \\
&\le \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |a_j| \cdot \left| \sigma(w_j^\top x) - \sigma(v_j^\top x) - \sigma'(v_j^\top x) x^\top (w_j - v_j) \right| \\
&\le \frac{1}{\sqrt{m}} \sum_{j=1}^{m} \frac{L(w_j^\top x - v_j^\top x)^2}{2} \\
&\le \frac{L}{2\sqrt{m}} \sum_{j=1}^{m} \|w_j - v_j\|_2^2 \\
&= \frac{L}{2\sqrt{m}} \|W - V\|_F^2 .
\end{aligned}$$

□

Remark 6. The preceding lemma holds for any W, and doesn't even need the Gaussian structure of W_0. This is unique to this shallow case, however; producing an analogous inequality with multiple layers of smooth activations will need to use random initialization.

Now we switch to the ReLU. The proof is much more complicated and omitted here.

Lemma 7. *For any radius $B \geq 0$, for any fixed $x \in \mathbb{R}^d$ with $\|x\|_2 \leq 1$, with probability at least $1-\delta$ over the draw of W_0, for any $W \in \mathbb{R}^{m \times d}$ with $\|W - W_0\|_F \leq B$,*

$$\left| f(x;W) - \tilde{f}(x;W) \right| \leq \frac{2B^{4/3} + B\log(1/\delta)^{1/4}}{m^{1/6}},$$

and given any additional $V \in \mathbb{R}^{m \times d}$ with $\|V - W_0\|_F \leq B$,

$$|f(x;V) - (f(x;W) - \langle \nabla_W f(x;W), V - W\rangle)| \leq \frac{6B^{4/3} + 2B\log(1/\delta)^{1/4}}{m^{1/6}}.$$

4.3 Properties of the kernel at initialization

So far, we've said that $f - \tilde{f}$ is small when the width is large. Now we will focus on $\tilde{f}$, showing that it is a large class of functions; thus, when the width is large, f obtained with small $\|W - W_0\|_F$ can also capture many functions.

To start, let us see how to define a kernel. To further simplify, we further assume that $a_j \in \{-1, +1\}$. In the standard kernel setup, the kernel can be written as the inner product between feature mappings for two data points:

$$\begin{aligned}
k_m(x;x') &\coloneqq \langle \nabla f(x;W_0), \nabla f(x';W_0)\rangle \\
&= \left\langle \begin{pmatrix} a_1 x^\top \sigma'(w_{0,1}^\top x) \\ \vdots \\ a_m x^\top \sigma'(w_{0,m}^\top x) \end{pmatrix}, \begin{pmatrix} a_1 x'^\top \sigma'(w_{0,1}^\top x') \\ \vdots \\ a_m x'^\top \sigma'(w_{0,m}^\top x') \end{pmatrix} \right\rangle \\
&= \frac{1}{m}\sum_{j=1}^m a_j^2 \left\langle x\sigma'(w_{j,0}^\top x), x'\sigma'(w_{j,0}^\top x')\right\rangle \\
&= x^\top x' \left(\frac{1}{m}\sum_{j=1}^m \sigma'(w_{j,0}^\top x)\sigma'(w_{j,0}^\top x') \right),
\end{aligned}$$

This gives one justification of the $1/\sqrt{m}$ factor: now this kernel is an average and not a sum, and we should expect it to have a limit as $m \to \infty$. To this end, and noting that the rows $(w_{0,j}^\top)_{j=1}^m$ are iid, then each term of the summation is iid, so by the SLLN, almost surely

$$k_m(x;x') \overset{m\to\infty}{\longrightarrow} k(x,x') \coloneqq x^\top x' \mathbb{E}_{a,w}\left[\sigma'(w^\top x)\sigma'(w^\top x')\right].$$

For now, let us calculate the closed form for the ReLU; let's do this geometrically.

- Consider the plane spanned by x and x'. Since projections of standard Gaussians are again standard Gaussians, we can consider a Gaussian random vector $v \in \mathbb{R}^2$ in this plane.
- The integrand in the expectation is 1 iff $v^\top x \geq 0$ and $v^\top x' \geq 0$. Since $\|v\|$ does not affect these expressions, we can simplify $v \in \mathbb{R}^2$ further to be sampled uniformly from the surface of the sphere.
- Suppose $\|x\|_2 = 1 = \|x'\|_2$, and define $\theta \coloneqq \arccos(x^\top x')$; then the integrand is 1 if v has positive inner product with both x and x', which has probability
$$\frac{\pi - \theta}{2\pi}.$$

Together, still using $\|x\|_2 = 1 = \|x'\|_2$,

$$k(x,x') = x^\top x' \mathbb{E}_w\left[1[w^\top x \geq 0]1[w^\top x' \geq 0]\right] = x^\top x' \left(\frac{\pi - \arccos(x^\top x')}{2\pi}\right).$$

Now let's return to the task of assessing how many functions we can represent near initialization. For this part, we will fix one degree of freedom in the data to effectively include a bias term; this is not necessary, but gives a shorter proof by reducing to standard kernel approximation theorems. We will show that this class is a universal approximator. Moreover $\|W - V\|$ will correspond to the RKHS norm, thus by making the width large, we can approximate elements of this large RKHS arbitrarily finely.

Proceeding in detail, first let's define our domain

$$\mathcal{X} \coloneqq \left\{ x \in \mathbb{R}^d : \|x\|_2 = 1, x_d = 1/\sqrt{d} \right\},$$

and our predictors

$$\mathcal{H} \coloneqq \left\{ x \mapsto \sum_{j=1}^{m} \alpha_j k(x, x_j) : m \geq 0, \alpha_j \in \mathbb{R}, x_j \in \mathcal{X} \right\}.$$

This might look fancy, but is the same as the functions we get by starting with $x \mapsto \langle \nabla f(x; W_0), W - W_0 \rangle$ and allowing the width to go to infinity, and $\|W - W_0\|$ be arbitrarily large; by the results in previous section, we can always choose an arbitrarily large width so that $f \approx \tilde{f}$ even when $\|W - W_0\|$ is large, and large width approximates infinite width. As such, it suffices to show that $\mathcal{H}$ is a universal approximator over $\mathcal{H}$.

Theorem 8. *$\mathcal{H}$ is a universal approximator over $\mathcal{X}$, that is, for every continuous function and every $\epsilon > 0$, there exists $h \in \mathcal{H}$ with $\sup_{x \in \mathcal{X}} |g(x) - h(x)| < \epsilon$.*

4.4 Optimization of neural nets in NTK regime

Many papers have come out in the sequel to tackle the optimization properties of neural networks in the NTK regime.

We first see that we can stay close to initialization long enough to get a small risk with an analysis that is essentially convex, essentially following the NTK (Taylor approximation). This proof is a simplification of one by [1].

We consider $f(x; w)$ be a parametrized prediction function, with $w \in \mathbb{R}^p$. For example, you can think w as a vectorization of W. Suppose we have training data $x = \{x_1, \ldots, x_n\}$ and $y = \{y_1, \ldots, y_n\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. we consider the predicted values on the training set as

$$f(w) \coloneqq \begin{bmatrix} f(x_1; w) \\ \vdots \\ f(x_n; w) \end{bmatrix} \in \mathbb{R}^n.$$

We will consider the squared loss regression:

$$\hat{\mathcal{R}}(\alpha f(w)) \coloneqq \frac{1}{2} \|\alpha f(w) - y\|_2^2, \qquad \hat{\mathcal{R}}_0 \coloneqq \hat{\mathcal{R}}(\alpha f(w(0))).$$

We'll consider the gradient flow:

$$\dot{w}(t) \coloneqq -\nabla_w \hat{\mathcal{R}}(\alpha f(w(t))) = -\alpha J_t^\top \nabla \hat{\mathcal{R}}(\alpha f(w(t))),$$

where

$$J_t \coloneqq J_{w(t)} \coloneqq \begin{bmatrix} \nabla_w f(x_1; w(t))^\top \\ \vdots \\ \nabla_w f(x_n; w(t))^\top \end{bmatrix} \in \mathbb{R}^{n \times p}.$$

We will also explicitly define and track a flow $u(t)$ over the tangent model; what we care about is $w(t)$, but we will show that indeed $u(t)$ and $w(t)$ stay close in this setting. (Note that $u(t)$ is not needed for the analysis of $w(t)$.)

$$\tilde{f}(u) \coloneqq f(w(0)) + J_0(u - w(0)).$$
$$\dot{u}(t) \coloneqq -\nabla_w \hat{\mathcal{R}}(\alpha \tilde{f}(w(t))) = -\alpha J_0^\top \nabla \hat{\mathcal{R}}(\alpha \tilde{f}(u(t))),$$

Both gradinet flows have the same initial condition:

$$u(0) = w(0), \qquad \tilde{f}(u(0)) = \tilde{f}(w(0)) = f(w(0)).$$

Remark 9. Notice that the setup so far doesn't make any mention of width, neural networks, random initialization, etc. It's all abstracted away! This is good and bad: the good is that it highlights the "scale" phenomenon, as α is the only concretely interpretable parameter here. On the downside, we need to do some work to get statements about width etc.

Assumption 10.

$$\begin{aligned}
&\operatorname{rank}(J_0) = n, \\
&\qquad \sigma_{\min} \coloneqq \sigma_{\min}(J_0) = \sqrt{\lambda_{\min}(J_0 J_0^\top)} = \sqrt{\lambda_n(J_0 J_)^\top)} > 0, \\
&\qquad \sigma_{\max} \coloneqq \sigma_{\max}(J_0) > 0, \\
&\|J_w - J_v\| \leq L \|w - v\|_2 .
\end{aligned}$$

Remark 11. $J_0 J_0^\top$ having a full rank is a "representation assumption" in an explicit sense: it implies the tangent model has exact solutions to the least squares problem, regardless of the choice of y, meaning the training error can always be made 0. In detail, consider the least squares problem solved by the tangent space:

$$\min_{u \in \mathbb{R}^p} \frac{1}{2} \|f_0(u) - y\|_2^2 = \min_{u \in \mathbb{R}^p} \frac{1}{2} \|J_0 u - y_0\|_2^2 ,$$

where we have chosen $y_0 \coloneqq y + J_0 w(0) - f(w(0))$ for convenience. The normal equations for this least squares problems are

$$J_0^\top J_0 u = J_0^\top y_0.$$

Let $J_0 = \sum_{i=1}^n s_i u_i v_i^\top$ denote the SVD of J_0, which has n terms by the rank assumption: the corresponding pseudoinverse if $J_0^\dagger = \sum_{i=1}^n s_i^{-1} v_i u_i^\top$. Multiplying both sides by $(J_0^\dagger)^\top$,

$$J_0 u = (J_0^\dagger)^\top J_0^\top J_0 u = (J_0^\dagger)^\top J_0^\top y_0 = \left(\sum_{i=1}^n u_i u_i^\top \right) y_0 = y_0.$$

where the last step follows since $\sum_{i=1}^n u_i u_i^\top$ is idempotent and full rank, and therefore the identity matrix. In particular, we can choose $\hat{u} = J_0^\dagger y_0$, then $J_0 \hat{u} = \left(\sum_{i=1}^n u_i u_i^\top \right) y_0 = y_0$, and in partyicular,

$$\frac{1}{2} \|f_0(\hat{u}) - y\|_2^2 = \frac{1}{2} \|J_0 \hat{u} - y_0\|_2^2 = 0.$$

Theorem 12 (See also Theorem 3.2 in [1])**.** *Suppose Assumption 10 holds, and* $\alpha \geq \frac{L\sqrt{1152\sigma_{\max}^2 \hat{\mathcal{R}}_0}}{\sigma_{\min}^2}$*. Then*

$$\begin{aligned}
\max\left\{ \hat{\mathcal{R}}(\alpha f(w(t))), \hat{\mathcal{R}}(\alpha \tilde{f}(w(t))) \right\} &\leq \hat{\mathcal{R}}_0 \exp\left(-t\alpha^2 \sigma_{\min}^2 / 2\right), \\
\max\left\{ \|w(t) - w(0)\|_2 , \|u(t) - u(0)\|_2 \right\} &\leq \frac{3\sqrt{8\sigma_{\max}^2 \hat{\mathcal{R}}_0}}{\alpha \sigma_{\min}^2}.
\end{aligned}$$

References

[1] Lénaïc Chizat and Francis R. Bach. A note on lazy training in supervised differentiable programming. *CoRR*, abs/1812.07956, 2020.

[2] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning—with applications in R.* Springer Texts in Statistics. Springer, New York, [2021] ©2021. Second edition [of 3100153].