Deep Learning

김지수 (Jisu KIM)

통계적 기계학습(Statistical Machine Learning), 2024 1st semester

This lecture note is a combination of several references. 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/

Antonio Álvarez López, Breaking the curse of dimensionality with Barron spaces, https://dcn.nat.fau.eu/breaking-the-curse-of-dimensionality-with-barron-spaces/

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, An introduction to statistical learning - with applications in R, https://www.statlearning.com/ [3]

1 Review

1.1 Basic Model for Supervised Learning

- Input(입력) / Covariate(설명 변수) : $x \in \mathbb{R}^p$, so $x = (x_1, \ldots, x_p)$.
- 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 $\hat{f}(x)$.

1.2 Linear Regression

From the additive noise model

$$y = f(x) + \epsilon,\ f \in \mathcal{M},$$

Linear Regression Model (선형회귀모형) is that

$$\mathcal{M} = \mathcal{F} = \left\{ \beta_0 + \sum_{j=1}^{p} \beta_j x_j : \beta_j \in \mathbb{R} \right\}.$$

For estimating β, we use least squares: suppose the training data is $\{(y_i, x_{ij}) : 1 \le i \le n, 1 \le j \le p\}$. We use square loss

$$\ell(y, a) = (y - a)^2,$$

then the eimpirical loss becomes the residual sum of square (RSS) as

$$\begin{aligned} RSS(\beta) &= \sum_{i=1}^{n} (y_i - f(x_i))^2 \\ &= \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2. \end{aligned}$$

Let $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p)$ be the nimimizor of RSS, then the predicted function is

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j.$$

2 Introduction

- Neural networks became popular in the 1980s. Lots of successes, hype, and great conferences: NeurIPS, Snowbird.
- Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.
- Re-emerged around 2010 as Deep Learning. By 2020s very dominant and successful.
- Part of success due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch
- Much of the credit goes to three pioneers and their students: Yann LeCun, Geoffrey Hinton and Yoshua Bengio, who received the 2019 ACM Turing Award for their work in Neural Networks.

2.1 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. What distinguishes neural networks from other nonlinear methods is the particular structure of the model:

$$f(x) = f_\theta(x) = g\left(b' + \sum_{j=1}^{m} a_j h(b_j + w_j^\top x) \right),$$

or more succintly, where $x \in \mathbb{R}^d, b_j \in \mathbb{R}, w_j \in \mathbb{R}^d, b' \in \mathbb{R}^D, a_j \in \mathbb{R}^D$. See Figure 1.

- $\theta = \{[b', a_j, b_j, w_j] : j = 1, \ldots, m^{(k)},\ i = 1, \ldots, D\}$ denotes the set of model parameters.
- $x_1, \ldots, x_d$ together is called an input layer.
- $A_j \coloneqq h_j(x) = h(\theta_j^\top x + b_j)$ 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.

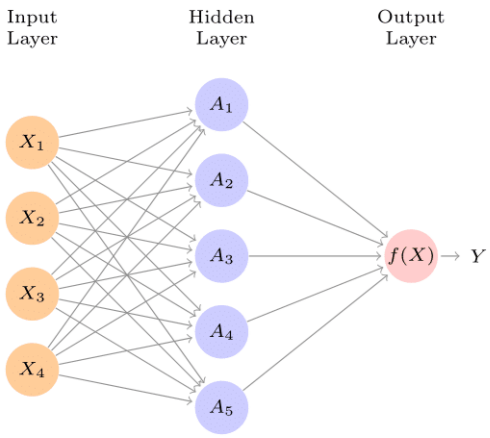

Figure 1: Neural network with a single hidden layer. The hidden layer computes activations $A_j = h_j(x)$ that are nonlinear transformations of linear combinations of the inputs $x_1, \ldots, x_d$. Hence these A_j are not directly observed. The functions h_j are not fixed in advance, but are learned 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 [3].

- $f(x)$ is called an output layer.
- g is an output function; identity $g(x) = x$ for regression, or softmax $g_i(x) = \exp(x_i)/\sum_{l=1}^{D} \exp(x_l)$ for classification. The softmax function estimates the conditional probability $g_i(x) = P(y = i|x)$.
- h is called an activation function. Popular are the sigmoids and rectified linear (ReLU), shown in Figure 2.
- 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.
- The model is fit by minimizing $\sum_{i=1}^{n}(y_i - f(x_i))^2$ (e.g. for regression).

2.2 Multi Layer Neural Networks

Modern neural networks typically have more than one hidden layer, and often many units per layer. In theory a single hidden layer with a large number of units has the ability to approximate most functions. However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

A deep neural network refers to the model allowing to have more than 1 hidden layers: given input $x \in \mathbb{R}^d$ and response $y \in \mathbb{R}^D$, to predict the response y. K-layer fully connected deep neural network is to build a nonlinear function $f(x)$ as

- Let $m^{(0)} = d$ and $m^{(K)} = D$
- Define recursively

$$
\begin{aligned}
&x^{(0)} = x, \quad (x \in \mathbb{R}^{m^{(0)}}),\\
&x_j^{(k)} = h(b_j^{(k)} + (w_j^{(k)})^\top x^{(k-1)}), \quad w_j^{(k)}, x^{(k-1)} \in \mathbb{R}^{m^{(k-1)}}, b_j \in \mathbb{R}^{m^{(k)}}\\
&f(x) = g(x^{(K)}) = g\left(b' + \sum_{j=1}^{m^{(K-1)}} a_j x_j^{(K-1)}\right),
\end{aligned}
$$

- $\theta = \{[b'^{(i)}, a_j^{(i)}, b_j^{(k)}, w_j^{(k)}] : k = 1, \ldots, K-1,\ j = 1, \ldots, m^{(k)},\ i = 1, \ldots, D\}$ denotes the set of model parameters.
- $m^{(k)}$ is the number of hidden units at layer k.

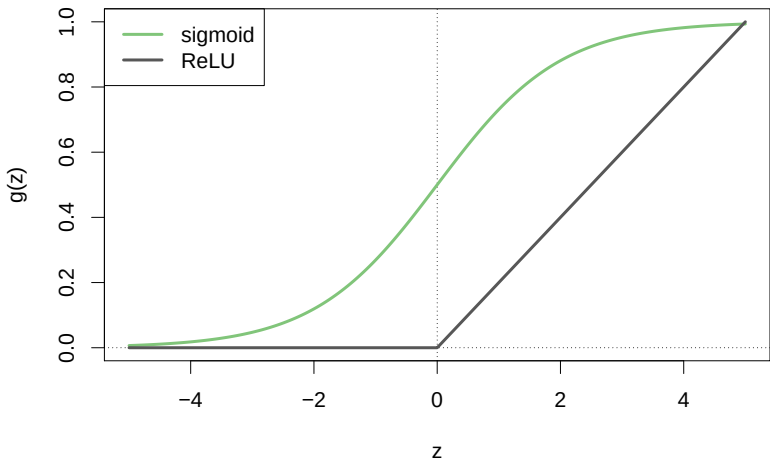

Figure 2: Activation functions. The piecewise-linear ReLU function is popular for its efficiency and computability. We have scaled it down by a factor of five for ease of comparison. Figure 10.2 from [3].

2.3 Example: MNIST

Example. MNIST dataset is a handwritten digit dataset. It is to classify images into digit class 0-9. Every image has $28 \times 28 = 784$ pixels of grayscale values $\in (0, 255)$. There are 60,000 train images and 10,000 test images.

- The goal is to build a classifier to predict the image class.
- We build a deep neural network with 256 units at first layer, 128 units at second layer, and 10 units at output layer.
- Along with intercepts (called biases) there are 235,146 parameters (referred to as weights).
- The output finction $g(x)$ is the softmax function: $g_i(x) = \exp(x_i)/\sum_{l=1}^{D} \exp(x_l)$.
- We fit the model by minimizing the negative multinomial log-likelihood (or cross-entropy):

$$-\sum_{j=1}^{m}\sum_{i=0}^{9} y_{ji} \log(f_i(x_j)),$$

  where y_{ji} is 1 if true class for observation j is i, else 0: one-hot encoded.

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

This is an early success for neural networks in the 1990s. With so many parameters, regularization is essential. Also, this is very overworked problem - best reported rates are $< 0.5\%$. Human error rate is reported to be around 0.2%, or 20 of the $10,000$ test images.

3 Notation

From here, we only consider regression problem, so $D = 1$ and $g(x) = x$. We assume no intercept, so $b' = b_j = 0$. Hence, for the two-layer neural network with the width of the hidden layer m and activation function h, the function space we consider is

$$\mathcal{F}_{m,h} = \left\{ f_w : f_w(x) = \sum_{j=1}^{m} u_j h(\theta_j^\top x) \right\},$$

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

$$\mathcal{F}_h = \bigcup_{m=1}^{\infty} \mathcal{F}_{m,h} = \left\{ f_w : f_w(x) = \sum_{j=1}^{m} u_j h(\theta_j^\top x), m \in \mathbb{N} \right\}.$$

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 prediction error,

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

And we estimate f^0 by $\hat{f}$ using data by minimizes on the empirical prediction error on training dataset, so

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

So there are two sources of errors: approximation error and generalization error.

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

What we would like to achieve is that:

For the approximation error: we would like to control $\left\| f_* - f^0 \right\|_{L^2(P)}$ appropriately in terms of the width of the neural network m. Ideally, we would like to restrict the function class $\mathcal{M}$ where f_*comes from, and define an appropriate norm $\|f_*\|_*$, so that

$$\inf_{f \in \mathcal{F}_{m,h}} \|f_* - f\|_{L^2(P)}^2 \lesssim \frac{\|f_*\|_*^2}{m}.$$

For the generalization error: we have seen from the concentration lecture note that with probability at least $1 - \delta$,

$$\sup_{f \in \mathcal{F}_{m,h}} \left| \frac{1}{n} \sum_{i=1}^{n} f(x_i) - \mathbb{E}[f] \right| \leq 2\mathrm{Rad}(\mathcal{F}_{m,h}) + \sqrt{\frac{1}{2n} \log\left(\frac{2}{\delta}\right)}.$$

Hence we would like to see that, with appropriate norm $\|f\|_{**}$ for $f \in \mathcal{F}_{m,h}$, define $\mathcal{F}_{m,h,Q} \coloneqq \{f \in \mathcal{F}_{m,h} : \|f\|_{**} \leq Q\}$, and then

$$\mathrm{Rad}(\mathcal{F}_{m,h,Q}) \lesssim \frac{Q}{\sqrt{n}}.$$

If both holds, then

$$\left\| f_* - \hat{f} \right\|_{L^2(P)}^2 = O_P\left(\frac{\|f_*\|_*^2}{m} + \frac{Q}{\sqrt{n}} \right).$$

4 Approximation error: Universal Approximation

4.1 Classical Universal Approximation

Definition. A class of functions $\mathcal{F}$ is a universal approximator over a compact set S if for every continuous function g and target accuracy $\epsilon > 0$, there exists $f \in \mathcal{F}$ with

$$\sup_{x \in S} |f(x) - g(x)| < \epsilon.$$

The classical Weierstrass theorem establishes that polynomials are universal approximators (Weierstrass 1885), and its generalization, the Stone-Weierstrass theorem, says that any family of functions satisfying some of the same properties as polynomials will also be a universal approximator. Stone-Weistrass theorem is a fairly standard way to prove universal approximation; this approach was first suggested in (Hornik, Stinchcombe, and White 1989).

Theorem. *Let functions $\mathcal{F}$ be given as follows.*

- *Each $f \in \mathcal{F}$ is continuous.*
- *For every $x \in S$, there exists $f \in \mathcal{F}$ with $f(x) \neq 0$.*
- *For every $x \neq y \in S$, there exists $f \in \mathcal{F}$ with $f(x) \neq f(y)$ ($\mathcal{F}$ separates points).*
- *$\mathcal{F}$ is closed under multiplication and vector space operations ($\mathcal{F}$ is an algebra).*

Then $\mathcal{F}$ is a universal approximator.

Then two-layer neural network is a universal approximator.

Theorem ([2])**.** *Suppose $h : \mathbb{R} \to \mathbb{R}$ is sigmoidal: it is continuous, and*

$$\lim_{z \to -\infty} h(z) = 0, \qquad \lim_{z \to \infty} h(z) = 1.$$

Then $\mathcal{F}_h$ is a universal approximator.

Theorem ([4])**.** *If $h : \mathbb{R} \to \mathbb{R}$ is continuous and non-polynomial, then $\mathcal{F}_h$ is a universal approximator.*

4.2 Universal Approximation on Baron Class

Classical universal approximation guarantees an approximation of arbitrary small error. But it has a drawback: we consider the two-layer neural network with arbitrary width, and to achieve a given accuracy we don't know how wide the hidden layer should be. If we fix the width of the neural network as m, then for f_* in Sobolev space,

$$\inf_{f^0 \in \mathcal{F}_{m,h}} \left\| f^0 - f_* \right\|_{L^p(S)} = O\left(\frac{1}{m^{\alpha/d}} \right),$$

so the approximation error suffers from the curse of dimensionality.

One fundamental reason is that maybe setting the regression function class $\mathcal{M}$ as the Sobolev space is too large; to overcome the curse of dimensionality, one approch would be to restrict $\mathcal{M}$.

We first recall the Fourier transform:

$$\tilde{f}(\omega) \coloneqq \int \exp(-2\pi i \omega^\top x) f(x) dx.$$

Then we have the Fourier inversion: if $f, \tilde{f} \in L^1$,

$$f(x) = \int \exp(2\pi i \omega^\top x) \tilde{f}(\omega) d\omega.$$

Barron class is a function that the Fourier transform of its gradient is integrable: note that $\widetilde{\nabla f}(\omega) = 2\pi i \omega \tilde{f}(\omega)$.

Definition. The quantity

$$\|f\|_{\mathcal{B}} \coloneqq \int \left\| \widetilde{\nabla f}(\omega) \right\|_2 d\omega = 2\pi \int \|\omega\|_2 \left| \tilde{f}(\omega) \right| d\omega$$

is called the Barron norm of a function f. The corresponding Barron class is

$$\mathcal{M}_{\mathcal{B}} = \left\{ f : \mathbb{R}^d \to \mathbb{R} : \|f\|_{\mathcal{B}} < \infty \right\}.$$

Theorem ([1, Theorem 11])**.** *For any $f_* \in \mathcal{M}_{\mathcal{B}}$ and $m \in \mathbb{N}$, there exists a two-layer neural network $f^0 \in \mathcal{F}_{m,h}$ with m neurons such that*

$$\left\| f_* - f^0 \right\|_{L_2(P)} \lesssim \frac{\|f_*\|_{\mathcal{B}}}{\sqrt{m}}.$$

Theorem ([1, Theorem 12])**.** *For any $f_* \in \mathcal{M}_{\mathcal{B}}$ and $m \in \mathbb{N}$, there exists a two-layer neural network $f^0 \in \mathcal{F}_{m,h}$ with m neurons such that*

$$\left\| f_* - f^0 \right\|_{L^\infty([0,1]^d)} \lesssim \|f_*\|_{\mathcal{B}} \sqrt{\frac{d+1}{m}}.$$

5 Generalization error

For $f_w \in \mathcal{F}_{m,h}$, we can write $f_w(x) = \sum_{j=1}^m a_j h(\theta_j^\top x)$. Define the 1-norm of w as

$$\|w\|_1 \coloneqq \frac{1}{m} \sum_{j=1}^m |a_j| \, \|\theta_j\|_1 .$$

Theorem ([1, Theorem 15]). *Let $\mathcal{F}_{m,h,Q} \coloneqq \{f_w \in \mathcal{F}_{m,h} : \|w\|_1 \le Q\}$. Then we have*

$$\mathrm{Rad}(\mathcal{F}_{m,h,Q}; Z^n) \le 2Q\sqrt{\frac{2\log(2d)}{n}}.$$

Instead of minimizing the training error, we can also consider the regularized term as

$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^n (y_i - f_w(x_i))^2 + \lambda \sqrt{\frac{\log(2d)}{n}} \, \|w\|_1 ,$$

and let $\hat{w}^{(1)}$ be its minimizer.

Theorem ([1, Theorem 16]). *Suppose $\mathcal{X} \subset \mathbb{R}^d$ is compact, and assume $f_* : \mathcal{X} \to [0,1]$. There exists some $\lambda_0 > 0$ such that for $\lambda \ge \lambda_0$, with probability $1 - \delta$,*

$$\frac{1}{n} \sum_{i=1}^n (y_i - f_{\hat{w}^{(1)}}(x_i))^2 \lesssim \frac{\|f_*\|_{\mathcal{B}}^2}{m} + \lambda \|f_*\|_{\mathcal{B}} \sqrt{\frac{\log(2d)}{n}} + \sqrt{\frac{\log(n/\delta)}{n}}.$$

References

[1] Weinan E, Chao Ma, Stephan Wojtowytsch, and Lei Wu. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. *CoRR*, abs/2009.10713, 2020.

[2] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[3] 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].

[4] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.