Deep Learning Framework

김지수 (Jisu KIM)

딥러닝의 통계적 이해 (Deep Learning: Statistical Perspective), 2024년 2학기

This lecture note is a a minor modification of the lecture notes from Prof. Joong-Ho Won's "Deep Learning: Statistical Perspective". Also, see Section 11 from [1] and Section 10 from [2].

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 $\{(y_i, x_{ij}) : 1 \leq i \leq n, 1 \leq j \leq p\}$. 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.

3 Multi Layer Nueral Networks

3.1 Two Layer Neural Networks

A two-layer neural network takes an input vector of d variables $x = (x_1, x_2, \ldots, x_p)$ 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 $x \in \mathbb{R}^p, b_j \in \mathbb{R}, w_j \in \mathbb{R}^p, \beta_0 \in \mathbb{R}^D, \beta_j \in \mathbb{R}^D$. See Figure 1.

- $\theta = \{[\beta, a_j, b_j, w_j] : j = 1, \ldots, M\}$ denotes the set of model parameters.
- $x_1, \ldots, x_p$ together is called an input layer.

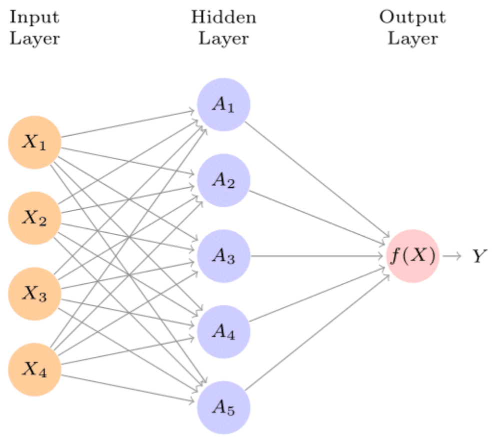

Figure 1: Neural network with a single hidden layer. The hidden layer computes activations $A_j = \sigma_j(x)$ that are nonlinear transformations of linear combinations of the inputs $x_1, \dots, x_p$. Hence these A_j are not directly observed. The functions σ_j are not fixed in advance, but are learned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

- $A_j := \sigma_j(x) = \sigma(b_j + w_j^\top x)$ is called an activation.
- $A_1, \dots, A_M$ 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.

3.2 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 $x \in \mathbb{R}^p$ 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 $m^{(0)} = p$ and $m^{(K)} = D$

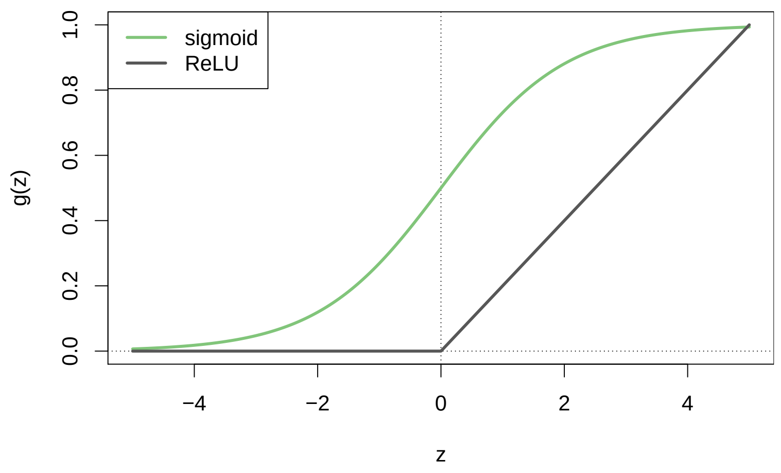

Figure 2: Activation functions. The piecewise-linear ReLU function is popular for its efficiency and computability. We have scaled it down by a factor of five for ease of comparison. Figure 10.2 from [2].

- Define recursively

$$
\begin{aligned}
x^{(0)} &= x, \quad (x \in \mathbb{R}^{m^{(0)}}), \\
x_j^{(k)} &= \sigma(b_j^{(k)} + (w_j^{(k)})^\top x^{(k-1)}), \quad w_j^{(k)}, x^{(k-1)} \in \mathbb{R}^{m^{(k-1)}}, b_j \in \mathbb{R}^{m^{(k)}} \\
f(x) &= g(x^{(K)}) = g\left(\beta_0 + \sum_{j=1}^{m^{(K-1)}} \beta_j x_j^{(K-1)}\right),
\end{aligned}
$$

- $\theta = \{[\beta_0, \beta_j, b_j^{(k)}, w_j^{(k)}] : k = 1, \dots, K-1,\ j = 1, \dots, m^{(k)}\}$ denotes the set of model parameters.
- $m^{(k)}$ is the number of hidden units at layer k.

4 Fitting Multi-Layer Neural Networks

Suppose we have training sample of size n: $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathcal{X} \subset \mathbb{R}^p$, $y_i \in \mathcal{Y} \subset \mathbb{R}^D$. Complete set of parameters (weights) are $\theta = \{W^{1\to 2}, W^{2\to 3}\}$, where

$$
\begin{aligned}
W^{1\to 2} &= \{b_j \in \mathbb{R}, w_j \in \mathbb{R}^p : j = 1, 2, \dots, M\}, \quad M \times (p+1) \text{ weights}, \\
W^{2\to 3} &= \{\beta_0 \in \mathbb{R}^D, \beta_k \in \mathbb{R}^D : k = 1, 2, \dots, M\}, \quad D \times (M+1) \text{ weights}.
\end{aligned}
$$

For the loss function, typically squared error is used for regression, and cross entropy is used for classification. Let $y_{ik} = I(y_i = k)$ if y_i's are categorical. then

$$
\mathcal{L}(\theta) = \begin{cases} \frac{1}{n}\sum_{i=1}^n \sum_{k=1}^K (y_{ik} - f_k(x_i))^2, & \text{(squared error)} \\ -\frac{1}{n}\sum_{i=1}^n \sum_{k=1}^K y_{ik} \log f_k(x_i), & \text{(cross entropy or deviance)} \end{cases}
$$

Typically the model is fitted by gradient descent:

$$
\theta \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}(\theta).
$$

Write $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^n E_i$, so that

$$
\nabla_\theta \mathcal{L} = \frac{1}{n}\sum_{i=1}^n \nabla_\theta E_i.
$$

In computing the gradient $\nabla_\theta E$, recall that each unit takes input $v_j = \sum_i w_{ij} u_i$ and outputs $u_j^+ = h_j(v_j)$ for some function h_j (equals to either σ or g_m).

We regard E as a function of all the weights, so changes in a weight w_{ij} affect the input and output of unit j and all units connected to j, including some output unit(s).

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_{ij}} = \frac{\partial E}{\partial v_j}u_i = h_j'(v_j)\frac{\partial E}{\partial u_j^+}u_i = u_i\delta_j.$$

- The first equality comes from the dependence of E on the weights only through the outputs.
- The second equality comes from $v_j = \sum_i w_{ij}u_i$.
- We evaluate $\partial E/\partial v_j$ by noting that v_j only affects the outputs through u_j^+, and this only acts through connections to output units:

$$\delta_j = \frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial u_j^+}\frac{\partial u_j^+}{\partial v_j} = h_j'(v_j)\frac{\partial E}{\partial u_j^+}.$$

- For output units $\partial E/\partial u_j^+$ can be calculated directly from the form of E:
  - For $K = 2$ and cross-entropy loss, $E = -y\log u - (1-y)\log(1-u)$, where $u = h(v)$ is the predicted probabilty of $Y = 1$. If h is sigmoid, then $h'(v) = u(1-u)$, yielding $\delta_j = y - u$.
  - Likewise,

$$\delta_j = \begin{cases} (\sum_{k=1}^K I(Y=k))u - I(Y=j), & \text{softmax, cross entropy,} \\ 2(u-y), & \text{linear, squared error,} \\ I(u \geq 0)\frac{u-y}{u(1-u)}, & \text{ReLU, cross entropy, } K=2. \end{cases}$$

- For units in earlier layers

$$\begin{aligned}\delta_j &= h_j'(v_j)\frac{\partial E}{\partial u_j^+} = h_j'(v_j)\sum_{k:j\to k}\frac{\partial E}{\partial v_k}\frac{\partial v_k}{\partial u_j^+} = h_j'(v_j)\sum_{k:j\to k}\frac{\partial E}{\partial v_k}w_{jk} \\ &= h_j'(v_j)\sum_{k:j\to k}w_{jk}\delta_k, \qquad v_k = \sum_j w_{jk}u_j^+,\end{aligned}$$

  the sum being over units k fed by unit j.
  - The second equality traces the effect of the output of an internal unit via the units to which it is connected.

Since this formula only contains terms in later layers, it is clear that it can be calculated *from output to input* on the network (*backpropagation*). It is often discussed as a *forward pass* to calculate the outputs from the inputs, followed by a *backward pass* to calculate (δ_j) and hence $\partial E/\partial w_{ij}$.

In practice $\frac{1}{n}\sum_{i=1}^n \nabla_\theta E_i$ is replaced by $\frac{1}{|B|}\sum_{i\in B}E_i$ where B is a *minibatch* (stochatic gradient descent).

4.1 Automatic differentiation and Computation graphs

The backpropagation rule is an instance of automatic differentiation (AD). AD refers to a collection of techniques that evaluate the derivatives of a function specified by a computer program accurately. Modern deep learning software features AD to train complex models via SGD.

Most AD techniques rely on decomposition of the target function into elementary functions (primitives) whose derivatives are known, and the computational graph, either explicitly or implicitly, that describes the dependency among the primitives.

Example. See Figure 3. The internal nodes represent intermediate variables corresponding to the primitives: $z_{-1} = x_1$, $z_0 = x_2$, $z_1 = z_{-1} + z_0$, $z_2 = \log z_1$, $z_3 = z_0^2$, and $z_4 = z_2 - z_3$; $y = z_4$. Suppose we want to evaluate ∇f at $(3, 2)$.

1. Forward-mode AD: evaluate from input to output
   - Evaluating $\partial f/\partial x_1$ and $\partial f/\partial x_2$ requires separate passes. Inefficient if the whole gradient of a function with many input variables is needed, e.g., the loss function of a high-dimensional model.
2. Reverse-mode AD: generalization of backpropagation

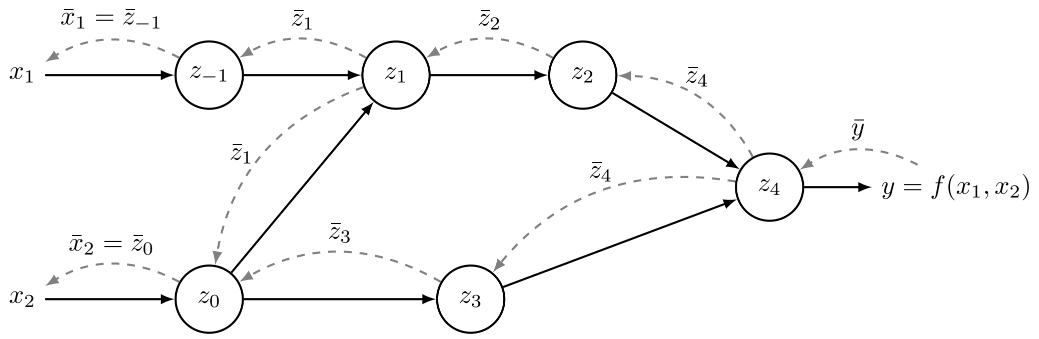

Figure 3: Computational graph for evaluating function $f(x_1, x_2) = \log(x_1 + x_2) - x_2^2$. Dashed arrows indicate the direction of backpropagation evaluating $\nabla f(x_1, x_2)$. Figure from [3].

- Forward pass: the original function and the associated intermediate variables z_i are evaluated from input to output.
- Backward pass: the "adjoint" variables $\bar{z}_i \equiv \frac{\partial y}{\partial z_i}$ are initialized to zero and updated from output to input.
- In the previous computational graph, $\bar{z}_4 \mathrel{+}= \frac{\partial y}{\partial z_4} = 1$, $\bar{z}_3 \mathrel{+}= \bar{z}_4 \frac{\partial z_4}{\partial z_3} = -1$, $\bar{z}_2 \mathrel{+}= \bar{z}_4 \frac{\partial z_4}{\partial z_2} = 1$, $\bar{z}_0 \mathrel{+}= \bar{z}_3 \frac{\partial z_3}{\partial z_0} = \bar{z}_3(2z_0) = -4$, $\bar{z}_1 \mathrel{+}= \bar{z}_2 \frac{\partial z_2}{\partial z_1} = \frac{\bar{z}_2}{z_2} = 1/5$, $\bar{z}_0 \mathrel{+}= \bar{z}_1 \frac{\partial z_1}{\partial z_0} = 1/5$, and $\bar{z}_{-1} \mathrel{+}= \bar{z}_1 \frac{\partial z_1}{\partial z_{-1}} = 1/5$.
- Finally, $\frac{\partial f}{\partial x_1} = \bar{x}_1 = \bar{z}_{-1} = 0.2$ and $\frac{\partial f}{\partial x_2} = \bar{x}_2 = \bar{z}_0 = -3.8$.

5 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.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. Data mining, inference, and prediction.

[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].

[3] Seyoon Ko, Hua Zhou, Jin J. Zhou, and Joong-Ho Won. High-performance statistical computing in the computing environments of the 2020s. *Statistical Science*, 37(4):494–518, 11 2022.