Extra Mathematical Notes for Lectures and Classes*

Parley Ruogu Yang†

This version: Friday 4^{th} March, 2022

Abstract

This document consists of notes for Lectures (section 2) and Classes (section 3).

Contents

*Latest version: `https://parleyyang.github.io/ST456/index.html`

†Faculty of Mathematics, University of Cambridge

1 General notes

1.1 Notations

The default meaning of $\mathbb{N}$ is the set of integers greater or equal to 1. For $n \in \mathbb{N}$, denote $[n] := \{1, 2, ..., n-1, n\} = [1, n] \cap \mathbb{N}$. When $x \in \mathbb{R}^n$ is written, x_i stands for the i-th entry of x. If $\rho : \mathbb{R} \to \mathbb{R}$ is well-defined, then for $y \in \mathbb{R}^n$, $\rho(y) := (\rho(y_1), ..., \rho(y_n))$, also known as element-wise operation.

$N(\mu, \sigma^2)$ refers to a normal distribution with mean μ and variance σ^2, while a standard normal distribution refers to the case when $\mu = 0$ and $\sigma^2 = 1$.

Where ε or ε_i are written, the default meaning is that they are drawn from iid $N(0, \sigma^2)$ distribution with σ^2 unknown.

NN stands for Neural Networks.

$\odot$ stands for element-wise multiplication

1.2 Activation functions

Let $\rho^{\texttt{sigmoid}} : \mathbb{R} \to \mathbb{R}$ be the sigmoid function, it is defined by

$$x \mapsto (1 + \exp(-x))^{-1} \tag{1.2.1}$$

Let $\rho^{\texttt{thr}} : \mathbb{R} \to \mathbb{R}$ be the threshold function, it is defined by

$$x \mapsto \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases} \tag{1.2.2}$$

This function is also commonly written as $\mathbb{1}[x \geq 0]$

Let $\rho^{\texttt{relu}} : \mathbb{R} \to \mathbb{R}$ be the ReLU (Rectified Linear Unit) function, it is defined by

$$x \mapsto \max\{0, x\} \tag{1.2.3}$$

Let $K \in \mathbb{N}$, $\rho^{\texttt{sm}} : \mathbb{R}^K \to (0,1)^K$ be the softmax function, the i-th coordinate of the output $\rho(x)_i$ is defined by

$$x \mapsto \frac{e^{x_i}}{\sum_{j \in [K]} e^{x_j}} \tag{1.2.4}$$

2 Lectures

2.1 Lectures 3 and 4: Basic optimisation methods

2.1.1 Gradient Descent in general

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, a gradient descent sequence $\{x_n\}_{n=0}^{\infty}$ with learning rate scheduling $\{\eta_n\}_{n=0}^{\infty}$ and initialisation x_{ini} is defined as

$$x_0 = x_{ini} \tag{2.1.1}$$
$$x_n = x_{n-1} - \eta_{n-1}\nabla f(x_{n-1}) \quad \forall n \in \mathbb{N} \tag{2.1.2}$$

2.1.2 Gradient Descent in the ERM framework

In the framework of Empirical Risk Minimisation (ERM), we are in the business of solving

$$\min_f \mathbb{E}_{(x,y)\sim p_{data}}[L(f(x,\theta),y)] \tag{2.1.3}$$

where $f(x,\theta)$ is the predicted output when the input data is x. In parametric setting, we search over some parametric space $\theta \in \Theta$ (often $\Theta=\mathbb{R}^n$), but also note the minimisation over f applies in a more general setting, e.g. functional minimisation or hyper-parameter search. Write the (empirical) data as $D := \{(x_i, y_i) : i \in [M]\}$ where M is the sample size, and the distribution of empirical data as p_{data}. We note that $J(\theta)$ can be written as $M^{-1}\sum_{i\in[M]} L(f(x_i,\theta),y_i)$.

Let $\theta \in \mathbb{R}^n$ and $J(\theta) := \mathbb{E}_{(x,y)\sim p_{data}}[L(f(x,\theta),y)]$, then a batch / deterministic gradient descent method with learning rate scheduling $\{\eta_n\}_{n=0}^{\infty}$ and initialisation θ_{ini} is defined as

$$\theta_0 = \theta_{ini} \tag{2.1.4}$$
$$\theta_n = \theta_{n-1} - \eta_{n-1}\nabla_\theta J(\theta_{n-1}) \quad \forall n \in \mathbb{N} \tag{2.1.5}$$

2.1.3 Stochastic Gradient Descent

A Stochastic Gradient Descent (SGD) algorithm takes the average gradient on a minibatch of m examples drawn randomly from the data. Clearly, for m to make sense, we practically have $m << M$. On the other hand, when we have $m = M$, SGD is the same as GD.

A SGD algorithm with batch size m, learning rate scheduling $\{\eta_n\}_{n=0}^{\infty}$, and initialisation θ_{ini} is defined as

$$\theta_0 = \theta_{ini} \tag{2.1.6}$$
$$\theta_n = \theta_{n-1} - \eta_{n-1}m^{-1}\nabla_\theta \sum_{j\in[m]} L(f(\tilde{x_j},\theta_{n-1}),\tilde{y_j}) \quad \forall n \in \mathbb{N} \tag{2.1.7}$$

where, $\forall n \in \mathbb{N}$, a set of data $\{(\tilde{x_j},\tilde{y_j}) : j \in [m]\}$ is sampled from p_{data} uniformly.

Larger m provides a more accurate estimate of the gradient, but more computational costs[1] Training with small m may require a small learning rate may require a small learning rate to maintain stability due to high variance in the estimation of gradient.

[1] In case of parallel computing, then memory scales with m, in case of sequential computing, the computational time scales with m.

2.1.4 Momentum

Based on subsubsection 2.1.3, we rewrite Equation 2.1.7 into the following two lines:

$$v_n = \eta_{n-1} m^{-1} \nabla_\theta \sum_{j \in [m]} L(f(\tilde{x_j}, \theta_{n-1}), \tilde{y_j}) \tag{2.1.8}$$

$$\theta_n = \theta_{n-1} - v_n \tag{2.1.9}$$

Now, a momentum method with initial velocity v_0 and momentum parameter α varies the above into

$$v_n = \alpha v_{n-1} - \eta_{n-1} m^{-1} \nabla_\theta \sum_{j \in [m]} L(f(\tilde{x_j}, \theta_{n-1}), \tilde{y_j}) \tag{2.1.10}$$

$$\theta_n = \theta_{n-1} + v_n \tag{2.1.11}$$

2.2 Lecture 4: Adaptive Learning Rates

2.2.1 General notions

General idea: adapt a separate learning rate (or momentum for Adam) for the update towards θ_n.

We reconsider the system as per Equation 2.1.8 and Equation 2.1.9 and introduce the following notation:

- Gradient $g_n := m^{-1} \nabla_\theta \sum_{j \in [m]} L(f(\tilde{x_j}, \theta_{n-1}), \tilde{y_j})$
- Gradient accumulation variable $\{r_n\}_{n=0}^{\infty}$ where $r_0 = 0$
- $\delta \in [10^{-7}, 10^{-6}]$ for numerical stabilisation
- Decay rates $\rho, \rho_1, \rho_2 \in [0, 1)$

Also note that square roots and divisions are element-wise throughout this subsection.

2.2.2 Adagrad

In Adagrad (Adaptive Gradient Algorithm), we moderate Equation 2.1.8 and Equation 2.1.9 into:

$$r_n = r_{n-1} + g_n \odot g_n \tag{2.2.1}$$

$$\theta_n = \theta_{n-1} - \frac{\eta_{n-1}}{\delta + \sqrt{r_n}} \odot g_n \tag{2.2.2}$$

2.2.3 RMSProp

In RMSProp (Root Mean Square Propagation), we vary Equation 2.2.1 into

$$r_n = \rho r_{n-1} + (1 - \rho) g_n \odot g_n \tag{2.2.3}$$

2.2.4 Adam

In Adam (Adaptive Moment Estimation), we consider two moments: s_n and r_n respectively, with initialisation $s_0 = r_0 = 0$. We moderate Equation 2.1.8 and Equation 2.1.9 into:

$$s_n = (1 - \rho_1^n)^{-1}(\rho_1 s_{n-1} + (1 - \rho_1) g_n) \tag{2.2.4}$$

$$r_n = (1 - \rho_2^n)^{-1}(\rho_2 r_{n-1} + (1 - \rho_2) g_n \odot g_n) \tag{2.2.5}$$

$$\theta_n = \theta_{n-1} - \frac{\eta_{n-1}}{\delta + \sqrt{r_n}} \odot s_n \tag{2.2.6}$$

2.3 Lecture 4: Dropout

Suppose the input to a layer is $x \in \mathbb{R}^n$. Recall the definition of a layer with activation ρ is:

$$z = wx + b \tag{2.3.1}$$

$$y = \rho(z) \tag{2.3.2}$$

A Dropout layer with probability p for the same input and activation is described as, with $r := (r_1, ..., r_n)$:

$$r_j \overset{iid}{\sim} Bernoulli(p) \quad \forall j \in [n] \tag{2.3.3}$$

$$z = w(r \odot x) + b \tag{2.3.4}$$

$$y = \rho(z) \tag{2.3.5}$$

2.4 Lectures 5 and 6: Convolutional Neural Networks (CNN)

Let K be a 4-D kernel tensor with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with k and l being the offset weights. Let V be the input with $V_{i,j,k}$ giving the value of the input unit with channel i at row j and column k. The output Z can be written as

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n} \tag{2.4.1}$$

For instance, in class 5, we look at:

- CIFAR-10 problem where images are given as 32 × 32 pixel coloured photographs, hence $j = k = 32$ and $l = 3$
- The MNIST problem where images are given as 28 × 28 pixel bilevel, hence $j = k = 28$ and $l = 1$.

Let $s := (s_1, s_2)$ be the strides, then a downsampled convolution with s is defined as

$$Z_{i,j,k}(s) = \sum_{l,m,n} V_{l,j \times s_1 + m, k \times s_2 + n} K_{i,l,m,n} \tag{2.4.2}$$

Maximum Pooling is a commonly used layer in CNN to reduce spatial dimensions of our hidden representations. The mathematical representation varies by coding implementations, see `https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html` for example.

3 Classes

3.1 Class 1: Linear and logistic regressions

3.1.1 Linear regression and MSE loss

Let $x \in \mathbb{R}^m$ be the input variable. Let $y \in \mathbb{R}$ be the output variable.

We consider $y = f(x) + \varepsilon$ where $f(x) = x^T w + b$

If we have data $\{(x_i, y_i) : i \in [n]\}$, the MSE loss takes the following form:

$$l(w, b) = n^{-1} \sum_{i \in [n]} (y_i - f(x_i))^2 = n^{-1} \sum_{i \in [n]} (y_i - x_i^T w - b)^2 \tag{3.1.1}$$

3.1.2 Gradient of linear regression with MSE loss

It will be useful later in subsection 2.1 to have the gradient ∇l in hand. In particular:

$$\nabla_w l(w, b) = n^{-1} \sum_{i \in [n]} (2x_i)(f(x_i) - y_i) \tag{3.1.2}$$

$$\nabla_b l(w, b) = n^{-1} \sum_{i \in [n]} 2(f(x_i) - y_i) \tag{3.1.3}$$

3.1.3 Logistic regression model and binary cross entropy

Let ρ be the sigmoid function, then we consider $y = f(x) + \varepsilon$ where $f(x) = \rho(x^T w + b)$

In the event of binary classification problem, in which $y \in \{0, 1\}$, we clearly do not have ε as a Normally distributed error. In this occasion, with data $\{(x_i, y_i) : i \in [n]\}$, we consider the binary cross entropy as

$$l(f) = -n^{-1} \left(\sum_{i \in [n]} y_i \log(f(x_i)) + (1 - y_i) \log(1 - f(x_i)) \right) \tag{3.1.4}$$

3.2 Class 2: Perceptron and the XOR Problem

3.2.1 Perceptron

With an activation function $\rho : \mathbb{R} \to \mathbb{R}$ and a feature engineering ϕ, we have a single-layer NN as $x \mapsto \rho(\phi(x)^T w + b)$

For the rest of the class (as well as in the lecture), we ignore ϕ, or equivalently replace it by an identity map. A feed-forward NN with depth L can be written as

$$y = h_L \circ h_{L-1} \circ ... \circ h_1(x) \tag{3.2.1}$$

where $h_l(x) = a_l(W^{(l-1)}x + b^{(l-1)})$ for all $l \leq L-1$ and $h_L(x) = W^{L-1}x + b^{L-1}$.

3.2.2 The XOR Problem statement

Consider $x \in \mathbb{R}^2$ and $y \in \mathbb{R}$, in particular, our data is as follows:

$$D = \{((-1,-1),-1),((-1,1),1),((1,-1),1),((1,1),-1)\} \tag{3.2.2}$$

The objective is to separate the points, mathematically one uses

$$L(f) = \sum_{i\in[4]} \max(-y_i f(x_i), 0) \tag{3.2.3}$$

3.2.3 Theoretical result

Theorem 1 (Failure of linear functions compared against two-layer NN)**.** *Let $\mathcal{L}$ be the class of all non-zero linear functions $\mathbb{R}^2 \to \mathbb{R}$ and let*

$$\mathcal{N}(\rho) = \{f : \mathbb{R}^2 \to \mathbb{R} : f(x) = \rho(w_1 x + b_1)^T w_2 + b_2, w_1 \in \mathbb{R}^{2\times 2}, w_2, b_1 \in \mathbb{R}^2, b_2 \in \mathbb{R}\} \tag{3.2.4}$$

where ρ is the threshold function. Then

$$\min_{f\in\mathcal{L}} L(f) > 0 = \min_{f\in\mathcal{N}(\rho)} L(f) \tag{3.2.5}$$

Proof. The left hand side can be proved by a 2-D diagram, or analytically via the diagram-induced geometry. The right hand side can be proved by showing an element $f \in \mathcal{N}(\rho)$ satisfies $L(f) = 0$, which is equivalent to show $y_i = f(x_i)\forall i$. Consider

$$b_1 = (0,0), b_2 = -1, w_2 = (-2,2)$$
$$w_1 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

which offers one specification that works. □

Remarks:

1. $\mathcal{N}(\rho)$ can also be thought as the class of all two-layer NNs with architecture as $(2,2,1)$ and activation function as the threshold function.
2. Note that the loss function can be 0 if $f(x_i) = 0 \forall i$. This is a bug of the loss function, hence when considering linear function, we restrict to the non-linear ones.

3.3 Class 3: Options Pricing

3.3.1 Background

A (European) call option at maturity T gives the owner the right to buy an underlying asset at strike price K. This price of such an option is denoted as $V(S_t, t; K)$ at time $t \in [0, T]$, where S_t is the price of the underlying asset at time t. It is natural to relate this to various parameters in the market: in the Black-Scholes model, we relate this to the interest rate r and volatility σ. A PDE expression is provided as

$$\partial_t V + rS\partial_S V + \frac{1}{2}\sigma^2 S^2 \partial_S^2 V = rV \tag{3.3.1}$$

The solution of this is complicated and non-linear:

$$V(S_t, t; K) = S_t N(d_1) - Ke^{r(T-t)}N(d_2) \tag{3.3.2}$$

where $d_1 = (\sigma\sqrt{T-t})^{-1}(\log(S_t K^{-1}) + (r + \frac{\sigma^2}{2}(T-t))$ and $d_2 = d_1 - \sigma\sqrt{T-t}$

3.3.2 Class 3 Notebook 1

In this notebook, we keep other parameters the same and study the relationship between strike price K and the associated price of call option V. In particular, we select a number of strike prices, denoted $x_1, ..., x_n \in \mathbb{R}$ and generate the call option prices $y_1, ..., y_n \in \mathbb{R}$ in accordance with Equation 3.3.2. The dataset is hence $\{(x_i, y_i) : i \in [n]\}$ and that we would like to approximate a function $f : \mathbb{R} \to \mathbb{R}$ as we generate our data $y_i = f(x_i) \ \ \forall i$

3.3.3 Class 3 Notebook 2

In practice, one would be asked for the implied volatility σ given the data they receive — in this notebook, we fix 16 different strike prices and collect their corresponding call prices: for now, assume no noise. Then, for each $y_i = \sigma_i \in \mathbb{R}$, we have a 16-dimensional data $x_i \in \mathbb{R}^{16}$, so the dataset is $\{(x_i, y_i) : i \in [n]\}$ and that we would like to approximate a function $f : \mathbb{R}^{16} \to \mathbb{R}$ as we generate our data $y_i = f(x_i) \ \ \forall i$

3.3.4 Class 3 Homework

Realistically, the data contains noise. In the Homework, we will work with noisy data, in particular, we consider the same function $f : \mathbb{R}^{16} \to \mathbb{R}$ as was in Notebook 2, but that we generate $\varepsilon_i \sim N(0_{16}, \sigma^2 I_{16\times16}) \forall i \in [n]$, and observe $\tilde{x_i} = \max\{x_i + \varepsilon_i, 0\}$ instead of x_i. The maximum is in place because the practical world would not accept a negative prices on an option — so whilst there are noises, there is an obvious truncation.

So, we are still in the business of approximating f, but this time we have data $\{(\tilde{x_i}, y_i) : i \in [n]\}$.