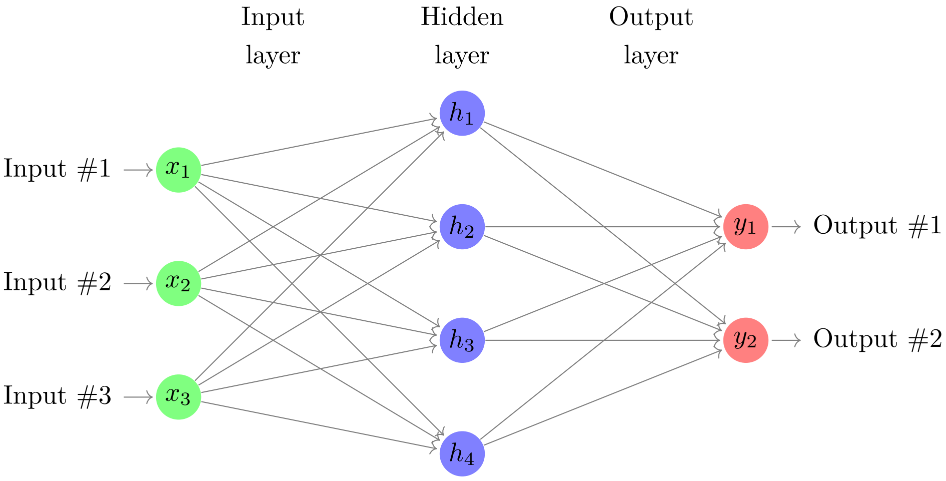
CSIE7435: Deep Learning Algorithms and Implementations

Vinsong

January 20, 2026

Abstract

The lecture note of 2025 Fall Deep Learning Algorithms and Implementations by professor 林智仁.



Vinsong

January 20, 2026

Contents

Chapter 1

Optimization Problems for Deep Learning

1.1 Linear Classification

1.1.1 Minimizing Training Errors

The most important goal in machine learning is to minimize the training error,

$$\min_{\text{model}} (\text{training error})$$

i.e. all the labels should be classified correctly by the model we train. Model can be a neural network, decision tree, SVM, etc. For the simplicity, we fisrt take the model to be a vector $\boldsymbol{w}$ (weight vector). We just predicted a hyperplane to separate different classes.

> **Definition 1.1.1** (decision function). A decision function is a function that maps the input features to predicted labels. It can be defined as
>
> $$f : \mathcal{X} \to \mathcal{Y}$$
>
> where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space (predicted labels).

That is, the decision function is inner product

$$\operatorname{sgn}(\boldsymbol{w}^\top \boldsymbol{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w}^\top \boldsymbol{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

For example, in 2D space,

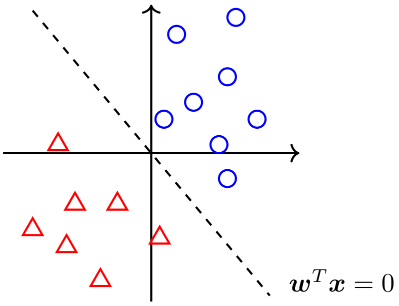


Figure 1.1: A linear classifier in 2-dimensional space

Note. In the reality, data is usually in the high-dimensional space. Which require hyperplanes to separate different classes but not just lines.

1.1.2 Loss Functions

To minimize the training error, we need to define a loss function to measure how well the model performs.

Definition 1.1.2 (Loss Function)**.** A loss function is a function that maps the predicted labels and true labels to a non-negative real number, representing the cost of the prediction. It can be defined as

$$\xi : \mathcal{M} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$$

where $\mathcal{M}$ is the model space, $\mathcal{X}$ is the feature vector, and $\mathcal{Y}$ is the output space (true labels).

We have to define the loss function $\xi(\boldsymbol{w}; \boldsymbol{x}, y)$ for each instance $(y, \boldsymbol{x})$, where $y \in \{+1, -1\}$ is the true label and $\boldsymbol{x} \in \mathbb{R}^n$ is the feature vector. Ideally, we want the loss function to be

$$\xi(\boldsymbol{w}; \boldsymbol{x}, y) = \begin{cases} 1 & \text{if } y \cdot (\boldsymbol{w}^\top \boldsymbol{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

However, this loss function is not continuous and not differentiable, which makes it hard to optimize. Therefore, we need to find some continuous approximation that are continuously differentiable.

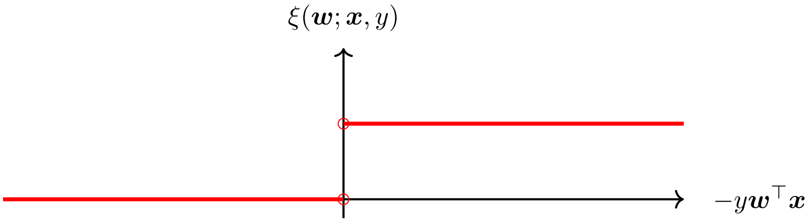


Figure 1.2: Discontinuous 0-1 Loss Function

Here are some commonly used loss functions that approximate the 0-1 loss function.

Definition (Common Loss Functions)**.** Here are two commonly used loss functions:

Definition 1.1.3 (Hinge Loss (L1 Loss))**.**

$$\xi_{\text{L1}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \max(0, 1 - y\boldsymbol{w}^\top \boldsymbol{x})$$

Definition 1.1.4 (Logistic Loss)**.**

$$\xi_{\text{LR}}(\boldsymbol{w}; \boldsymbol{x}, y) \equiv \log\bigl(1 + \exp\bigl(-y\boldsymbol{w}^\top \boldsymbol{x}\bigr)\bigr)$$

Note. Hinge Loss 在 $y\boldsymbol{w}^\top \boldsymbol{x} < 0$ 很多的時候就會發現 loss 很大，代表這個點的分類結果 $\boldsymbol{w}^\top \boldsymbol{x}$ 跟 y 相反，$|\boldsymbol{w}^\top \boldsymbol{x}|$ 越大，代表分類分的越錯。所以當我們發現異號並且 $|\boldsymbol{w}^\top \boldsymbol{x}|$ 很大時，我們會給予很大的懲罰 (loss 很大)。而 Logistic Loss 也是類似的概念，當正確時 $\exp\bigl(-y\boldsymbol{w}^\top \boldsymbol{x}\bigr)$ 很小，可以忽略 Loss 大約是 0，當錯誤時 $\exp\bigl(-y\boldsymbol{w}^\top \boldsymbol{x}\bigr)$ 很大，取 log 後大概就是 $-y\boldsymbol{w}^\top \boldsymbol{x}$ (很大的 positive error)。

Note. Support Vector Machine (SVM) uses Hinge Loss as its loss function, while Logistic Regression uses Logistic Loss. SVM and LR are both fundamental methods for classification.

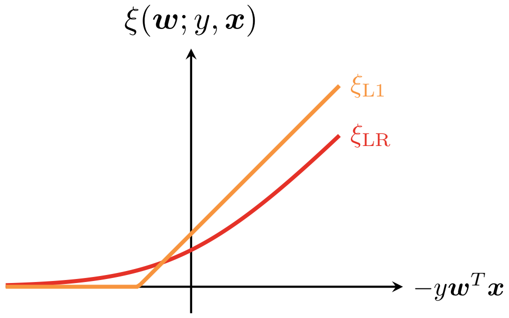


Figure 1.3: Comparison of Logistic Loss and Hinge Loss

And logistic regression is very related to SVM, they look very similar in on the graph.

1.1.3 Overfitting

Minimizing the training error alone may not give a good model, as it may lead to overfitting. For classification, you can easily achieve 100% training accuracy. But the model may not generalize well to unseen data.
In the following figure, model perfectly classifies all training data, but the perfermance on test data is not quite perfect.

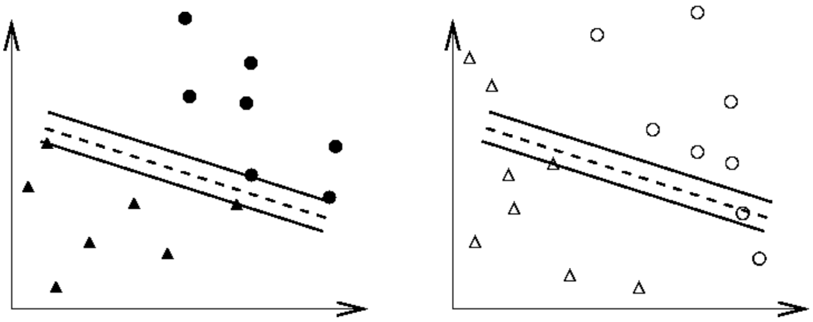


Figure 1.4: Overfitting Example

If we just give up some training accuracy (might misclassify some training data), consider them to be the noise, we may get a better model that generalizes well to unseen data（margin 比較大）。

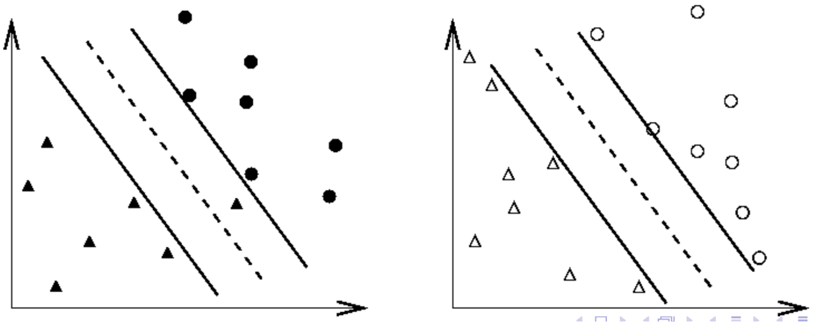


Figure 1.5: Regularization Example

1.1.4 Regularization

Hence, we introduce regularization to our optimization problem, which helps to make the value of $\|\boldsymbol{w}\|$ less extreme. One idea of regularization is to make $\|\boldsymbol{w}\|$ close to zero. For example, we can add

$$\frac{\boldsymbol{w}^\top \boldsymbol{w}}{2} \text{ (which is } \|\boldsymbol{w}\|_2^2 \cdot \frac{1}{2}) \quad \text{or} \quad \|\boldsymbol{w}\|_1 = \sum_{i=1}^{n} |w_i|$$

Note. 加上 penalty term $\frac{\boldsymbol{w}^\top \boldsymbol{w}}{2}$ 會讓這一個 $\|\boldsymbol{w}\|$ 變得不容易很大。因為當 $\|\boldsymbol{w}\|$ 很大時，$\frac{\boldsymbol{w}^\top \boldsymbol{w}}{2}$ 也會迅速增加。$\min_{\boldsymbol{w}} f(\boldsymbol{w})$ 就會把這個雖然符合，但 margin 太小。假設點落在 margin 上，we get

$$|\boldsymbol{w}^\top \boldsymbol{x}| = 1$$

這樣我們可以得到兩條 margin lines 之間的距離是

$$\text{distance} = \frac{|C_1 - C_2|}{\sqrt{w_1^2 + w_2^2}} = \frac{2}{\|\boldsymbol{w}\|}$$

當 $\|\boldsymbol{w}\|$ 越小，margin distance 就越大，代表這個 classifier 越 generalize

Therefore, the optimization problem becomes

Definition 1.1.5 (General Form of Linear Classification)**.** Let training data be

$$\{y, \boldsymbol{x}_i\}, \ \boldsymbol{x}_i \in \mathbb{R}^n, \ y_i \in \{+1, -1\}, \quad \text{for } i = 1, 2, \ldots, l$$

where l is # of training instances, n is # of features. The general form of linear classification optimization problem is

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad f(\boldsymbol{w}) = \frac{\boldsymbol{w}^\top \boldsymbol{w}}{2} + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

where $C > 0$ is a hyperparameter that controls the trade-off between minimizing the training error and minimizing the model complexity (regularization term).

1.2 Fully-connected Networks

1.2.1 Multi-class Classification

Definition 1.2.1. The training set of a fully-connected network is the same as linear classification.

$$(\boldsymbol{y}^i, \boldsymbol{x}^i), \quad i = 1, 2, \ldots, l$$

where $\boldsymbol{y}^i \in \mathbb{R}^K$ is the label vector (one-hot encoding) and $\boldsymbol{x}^i \in \mathbb{R}_1^n$ is the feature vector.

Due to the label is a vector now, we change

$$(\text{label, instance}) : (\boldsymbol{y}_i, \boldsymbol{x}_i) \quad \rightarrow \quad (\text{label vector, instance}) : (\boldsymbol{y}^i, \boldsymbol{x}^i)$$

where K is the number of classes and n_1 is the number of features. If x^i is in class k then

$$\boldsymbol{y}^i = [0, 0, \ldots, 1, \ldots, 0]^\top \in \mathbb{R}^K$$

A neural network maps each input feature vector to one of the class labels by the connection nodes. Betewwn layers, there are weights matrix maps input to the next layer output.

1.2.2 Operation Between Layers

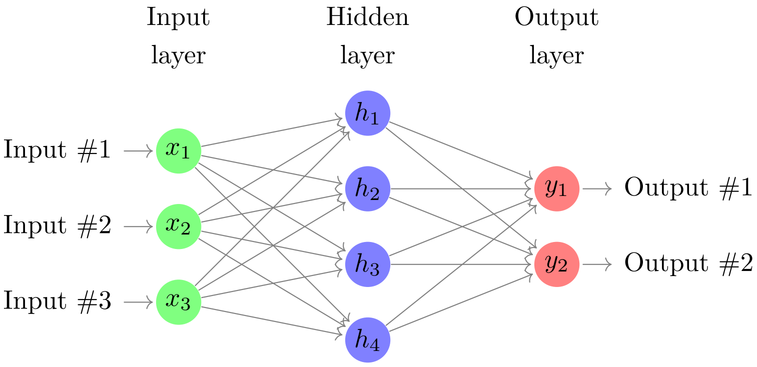


Figure 1.6: A Simple Fully-connected Neural Network

The weight matrix W^m at layer m is

$$W^m = \begin{pmatrix} w_{11}^m & w_{12}^m & \cdots & w_{1n_m}^m \\ w_{21}^m & w_{22}^m & \cdots & w_{2n_m}^m \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_{m+1}1}^m & w_{n_{m+1}2}^m & \cdots & w_{n_{m+1}n_m}^m \end{pmatrix}_{n_{m+1} \times n_m}$$

where

- n_m is the number of input features at layer m.
- n_{m+1} is the number of output features at layer m.
- w_{ij}^m is the weight connecting the jth node in layer m to the ith node in layer $m+1$.
- L is the total number of layers.
- n_1: number of features, n_{L+1}: number of classes.

Let $\boldsymbol{z}^m$ be the input of the m-th layer, $\boldsymbol{z}_1 = \boldsymbol{x}$ and $\boldsymbol{z}_{L+1} = \boldsymbol{y}$. From m-th layer to $(m+1)$-th layer, the operation is

$$\begin{aligned} \boldsymbol{s}^m &= W^m \boldsymbol{z}^m \\ \boldsymbol{z}_j^{m+1} &= \sigma(\boldsymbol{s}_j^m), \quad j = 1, 2, \ldots, n_{m+1} \end{aligned}$$

where

- $\sigma(\cdot)$ is the activation function.

Usually, we do a bias term $\boldsymbol{b}^m$ addition

$$\begin{pmatrix} b_1^m \\ b_2^m \\ \vdots \\ b_{n_{m+1}}^m \end{pmatrix}_{n_{m+1} \times 1}$$

so that

$$\boldsymbol{s}^m = W^m \boldsymbol{z}^m + \boldsymbol{b}^m$$

Note (Actication Functions). Activation function f is usually a scalar function

$$f : \mathbb{R} \to \mathbb{R}$$

for nonlinear transformation. 有些分類問題並非是 linear separable 的，所以需要這種 non-linear 來幫助分類，例如：

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ReLU: $\mathrm{ReLU}(x) = \max(0, x)$

如果沒有 activation function，整個 network 就只是 linear transformation 的組合，等同於 single layer 的 linear model

$$W^L W^{L-1} \cdots W^1 \boldsymbol{x} + \text{bias terms}$$

如果做了 activation function，network 就沒辦法簡化成 single layer

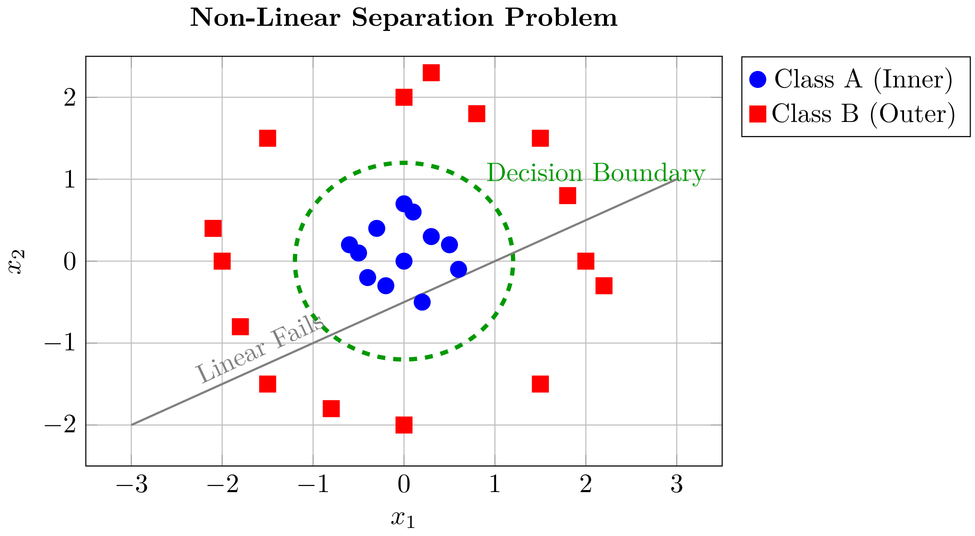


Figure 1.7: Non-Linear Separation Problem

To simplify the notation, we denote all the variables as a vector θ

$$\theta = \begin{pmatrix} \mathrm{vec}(W^1) \\ \boldsymbol{b}^1 \\ \vdots \\ \mathrm{vec}(W^L) \\ \boldsymbol{b}^L \end{pmatrix} \in \mathbb{R}^n$$

where n is total # parameters in the network $= (n_1 + 1)n_2 + \cdots (n_L + 1)n_{L+1}$. And the operator

$$\mathrm{vec}(\cdot)$$

stacks the columns of a matrix into a vector.

The optimization problem for training a fully-connected network should be write in

Definition 1.2.2 (Optimization Problem for Fully-connected Network)**.** For θ is the parameter vector, the optimization problem for training a fully-connected network is

$$\min_{\theta} f(\theta), \quad f(\theta) = \frac{1}{2}\theta^\top \theta + C\sum_{i=1}^{l} \xi(z^{L+1,i}(\theta); \boldsymbol{y}^i, \boldsymbol{x}^i)$$

where

- C: regularization hyperparameter.
- $z^{L+1}(\theta) \in \mathbb{R}^{n_{L+1}}$: last layer output (predicted label vector) of the feature $\boldsymbol{x}$
- $\xi(z^{L+1}(\theta); \boldsymbol{y}, \boldsymbol{x})$: loss function: for instance

$$\xi(z^{L+1}(\theta); \boldsymbol{y}, \boldsymbol{x}) = \|z^{L+1}(\theta) - \boldsymbol{y}\|^2$$

This is similar to the linear classification problem, but now the model is more complex. Futher, the optimization problem is non-convex due to the non-linear activation functions and multiple layers.

Remark. In the real world, deep learning models often have not single instance, so in the process of training, it is actually for $i = 1, \ldots, l$

$$\begin{aligned} \boldsymbol{s}^{m,i} &= W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m \\ \boldsymbol{z}_j^{m+1,i} &= \sigma(\boldsymbol{s}_j^{m,i}), \quad j = 1, 2, \ldots, n_{m+1} \end{aligned}$$

1.3 Convolutional Neural Networks

There are several types if neural networks, each designed to handle different types of data and tasks. Although neural networks might not always be the best choice for every problem, for instance, fully-connected-networks are not consistently better than random forests or SVM. However, for certain tasks, like image recognition, convolutional neural networks (CNNs) have shown remarkable performance.

1.3.1 Convolutional Neural Networks (CNNs)

Definition 1.3.1. Consider a K-class classification problem with training data

$$(\boldsymbol{y}^i, \boldsymbol{Z}^{1,i}), \quad i = 1, 2, \ldots, l$$

where $\boldsymbol{y}^i \in \mathbb{R}^K$ is the label vector and $\boldsymbol{Z}^{1,i}$ is the input image.

If $\boldsymbol{Z}^{1,i}$ is in class k, then

$$y^i = \left[\underbrace{0, \ldots, 0}_{k-1}, 1, 0, \ldots, 0\right]^T \in \mathbb{R}^K$$

CNN can map the input image $\boldsymbol{Z}^{1,i}$ to the label vector $\boldsymbol{y}^i$. Typically, CNN consists of several convolutional layers followed by fully-connected layers.

1.3.2 Convolutional Layer

The input and output of a convolutional layer are assume to be images.

Definition 1.3.2. For a single convolutional layer, the input image is denoted as

$$\boldsymbol{Z}^{\text{in}} : a^{\text{in}} \times b^{\text{in}} \times d^{\text{in}}$$

where a^{in} and b^{in} are the height and width of the image, and d^{in} is the number of channels (e.g., 3 for RGB images).

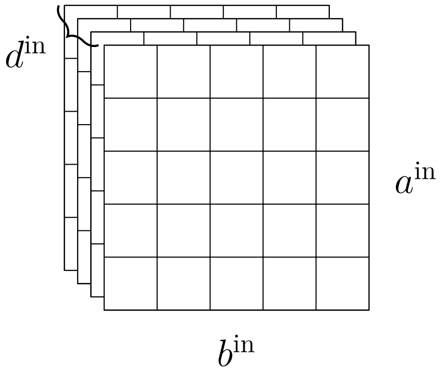


Figure 1.8: Input of a convolutional layer

The goal is to generate an output image

$$\boldsymbol{Z}^{\text{out},i} : a^{\text{out}} \times b^{\text{out}} \times d^{\text{out}}$$

Consider a set of d^{out} filters (or kernels), filter $j \in \{1, \ldots, d^{\text{out}}\}$ has dimensions

$$h \times h \times d^{\text{in}}$$

$$\begin{pmatrix} w^j_{1,1,1} & \cdots & w^j_{1,h,1} \\ \vdots & \ddots & \vdots \\ w^j_{h,1,1} & \cdots & w^j_{h,h,1} \end{pmatrix}, \cdots, \begin{pmatrix} w^j_{1,1,d^{\text{in}}} & \cdots & w^j_{1,h,d^{\text{in}}} \\ \vdots & \ddots & \vdots \\ w^j_{h,1,d^{\text{in}}} & \cdots & w^j_{h,h,d^{\text{in}}} \end{pmatrix}$$

where h is the filter size (height and width, layer index is omitted for simplicity)

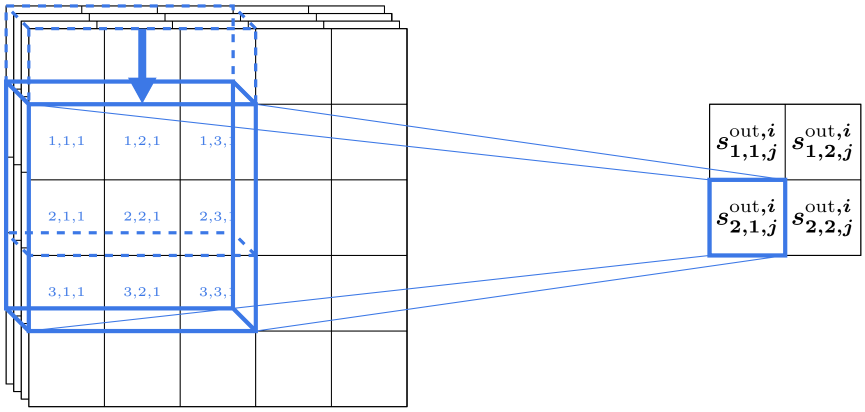

1,1,1
1,2,1
1,3,
2,1,1
2,2,1
2,3,
3,1,1
3,2,1
3,3,
$\boldsymbol{s}^{\text{out},\boldsymbol{i}}_{\boldsymbol{1,1,j}}$
$\boldsymbol{s}^{\text{out},\boldsymbol{i}}_{\boldsymbol{1,2,j}}$
$\boldsymbol{s}^{\text{out},\boldsymbol{i}}_{\boldsymbol{2,1,j}}$
$\boldsymbol{s}^{\text{out},\boldsymbol{i}}_{\boldsymbol{2,2,j}}$

Figure 1.9: Convolutional Layer Illustration

To compute the j-th channel of the output image, we scan the input from top-left to bottom-right. At each position obtain the sub-image

$$\boldsymbol{S}^{\text{in}} : h \times h \times d^{\text{in}}$$

We calculate the inner product between each sub-image and the j-th filter to get a single scalar value

$$s^{\text{out},i}_{u,v,j} = \langle \boldsymbol{S}^{\text{in}}, \boldsymbol{W}^{j} \rangle$$

The idea of doing this operation is to extract local features from the input image. For instance, edges, corners, and textures can be detected by appropriate filters. If we start from the top-left corner of the input image, the first sub-image of channel d is

$$\boldsymbol{S}^{i}_{1,1,d} = \begin{pmatrix} z^{i}_{1,1,d} & \cdots & z^{i}_{1,h,d} \\ \vdots & \ddots & \vdots \\ z^{i}_{h,1,d} & \cdots & z^{i}_{h,h,d} \end{pmatrix}$$

We then calculate

$$s^{\text{out},i}_{1,1,j} = \sum_{d=1}^{d^{\text{in}}} \left\langle \begin{pmatrix} z^{i}_{1,1,d} & \cdots & z^{i}_{1,h,d} \\ \vdots & \ddots & \vdots \\ z^{i}_{h,1,d} & \cdots & z^{i}_{h,h,d} \end{pmatrix}, \begin{pmatrix} w^{j}_{1,1,d} & \cdots & w^{j}_{1,h,d} \\ \vdots & \ddots & \vdots \\ w^{j}_{h,1,d} & \cdots & w^{j}_{h,h,d} \end{pmatrix} \right\rangle + b_j \tag{1}$$

where $\langle \cdot, \cdot \rangle$ denotes the component-wise inner product between two matrices. This value becomes the $(1, 1)$ position of the j-th channel of the output image. Next, we use other sub-images to compute other positions of the output image. Let the stride s be the number of pixels vertically or horizontally to get the next sub-image. For the position $(2, 1)$, we move vertically down by s pixels to get the sub-image

$$\boldsymbol{S}^{i}_{2,1,d} = \begin{pmatrix} z^{i}_{1+s,1,d} & \cdots & z^{i}_{1+s,h,d} \\ \vdots & \ddots & \vdots \\ z^{i}_{h+s,1,d} & \cdots & z^{i}_{h+s,h,d} \end{pmatrix}$$

Then we can get $(2, 1)$ position of the j-th channel of the output image by

$$s^{\text{out},i}_{2,1,j} = \sum_{d=1}^{d^{\text{in}}} \left\langle \begin{pmatrix} z^{i}_{1+s,1,d} & \cdots & z^{i}_{1+s,h,d} \\ \vdots & \ddots & \vdots \\ z^{i}_{h+s,1,d} & \cdots & z^{i}_{h+s,h,d} \end{pmatrix}, \begin{pmatrix} w^{j}_{1,1,d} & \cdots & w^{j}_{1,h,d} \\ \vdots & \ddots & \vdots \\ w^{j}_{h,1,d} & \cdots & w^{j}_{h,h,d} \end{pmatrix} \right\rangle + b_j \tag{2}$$

Theorem 1.3.1. The output image size can be calculated respectively by

$$a^{\text{out}} = \left\lfloor \frac{a^{\text{in}} - h}{s} \right\rfloor + 1, \quad b^{\text{out}} = \left\lfloor \frac{b^{\text{in}} - h}{s} \right\rfloor + 1 \tag{3}$$

Proof. Vertically, the last valid starting position for the sub-image is at row

$$h, h + s, h + 2s, \ldots, h + \Delta s \leq a^{\text{in}}$$

Thus, we have

$$h + \Delta s \leq a^{\text{in}} \Rightarrow \Delta \leq \frac{a^{\text{in}} - h}{s}$$

Hence, $\Delta = \left\lfloor \dfrac{a^{\text{in}} - h}{s} \right\rfloor$ ■

1.3.3 Matrix Operations

To implement convolutional layers efficiently, we should use the matrix-matrix and matrix-vector operations to do the convalutional operation (simplify the optimization). We first collect images of all channels as the input matrix

$$\boldsymbol{Z}^{\text{in},i} = \begin{pmatrix} z^i_{1,1,1} & z^i_{2,1,1} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},1} \\ \vdots & \vdots & \ddots & \vdots \\ z^i_{1,1,d^{\text{in}}} & z^i_{2,1,d^{\text{in}}} & \cdots & z^i_{a^{\text{in}},b^{\text{in}},d^{\text{in}}} \end{pmatrix} \in \mathbb{R}^{d^{\text{in}} \times (a^{\text{in}} b^{\text{in}})}$$

and let all filters to be

$$W = \begin{pmatrix} w^1_{1,1,1} & w^2_{1,1,1} & \cdots & w^{d^{\text{out}}}_{1,1,1} \\ \vdots & \vdots & \ddots & \vdots \\ w^1_{h,h,d^{\text{in}}} & w^2_{h,h,d^{\text{in}}} & \cdots & w^{d^{\text{out}}}_{h,h,d^{\text{in}}} \end{pmatrix} \in \mathbb{R}^{(hhd^{\text{in}}) \times d^{\text{out}}}$$

by the parameters of current layer. The bias vector is

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{d^{\text{out}}} \end{pmatrix} \in \mathbb{R}^{d^{\text{out}}}$$

Then we get

Definition 1.3.3. The operation of a single convolutional layer as

$$\boldsymbol{S}^{\text{out},i} = W\phi(\boldsymbol{Z}^{\text{in},i}) + b\mathbb{1}^T_{a^{\text{out}}b^{\text{out}}} \tag{1.1}$$

where

$$\mathbb{1}_{a^{\text{out}}b^{\text{out}}} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{a^{\text{out}}b^{\text{out}} \times 1}$$

and $\phi(\boldsymbol{Z}^{\text{in},i})$ collects all sub-images in $\boldsymbol{Z}^{\text{in},i}$ as columns. Specifically, the

$$\phi(\boldsymbol{Z}^{\text{in},i}) = \begin{pmatrix} z^i_{1,1,1} & z^i_{1+s,1,1} & \cdots & z^i_{1+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\ z^i_{2,1,1} & z^i_{2+s,1,1} & \cdots & z^i_{2+(a^{\text{out}}-1)s,1+(b^{\text{out}}-1)s,1} \\ \vdots & \vdots & \ddots & \vdots \\ z^i_{h,h,1} & z^i_{h+s,h+s,1} & \cdots & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,1} \\ \vdots & \vdots & \ddots & \vdots \\ z^i_{h,h,d^{\text{in}}} & z^i_{h+s,h+s,d^{\text{in}}} & \cdots & z^i_{h+(a^{\text{out}}-1)s,h+(b^{\text{out}}-1)s,d^{\text{in}}} \end{pmatrix} \in \mathbb{R}^{(hhd^{\text{in}}) \times (a^{\text{out}}b^{\text{out}})}$$

Next we consider the activation function scales each element of $\boldsymbol{S}^{\text{out},i}$ to obtain the output matrix

$$\boldsymbol{Z}^{\text{out},i} = \sigma(\boldsymbol{S}^{\text{out},i}) \in \mathbb{R}^{d^{\text{out}} \times (a^{\text{out}}b^{\text{out}})} \tag{1.2}$$

For CNN, we usually use ReLU as the activation function.

$$\sigma(x) = \max(0, x) \tag{1.3}$$

We need it to be differentiable for backpropagation, but ReLU is not differentiable at $x = 0$. In practice, (Krizhevsky et al.) we can define

$$\sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the matrix-matrix product form

$$W\phi(\boldsymbol{Z}^{\text{in},i})$$

each element is the inner product between a filter and a sub-image. Clearly, ϕ is a linear mapping from $\mathbb{R}^{d^{\text{in}}\times(a^{\text{in}}b^{\text{in}})}$ to $\mathbb{R}^{(hhd^{\text{in}})\times(a^{\text{out}}b^{\text{out}})}$, so there exists a matrix P_ϕ such that

Definition. For the linear mapping ϕ, we have

Definition 1.3.4 (Convolutional Mapping)**.**

$$\phi(\boldsymbol{Z}^{\text{in},i}) = \text{mat}\left(P_\phi \operatorname{vec}(\boldsymbol{Z}^{\text{in},i})\right)_{(hhd^{\text{in}})\times(a^{\text{out}}b^{\text{out}})}, \quad \forall i \tag{1.4}$$

Definition 1.3.5 (Vectorization)**.** For a matrix M's columns concatenated to a vector $\mathbf{v}$

$$\operatorname{vec}(M) = \mathbf{v} = \begin{pmatrix} M_{:,1} \\ M_{:,2} \\ \vdots \\ M_{:,n} \end{pmatrix} \in \mathbb{R}^{ab\times 1}, \quad \text{where } M \in \mathbb{R}^{a\times b}$$

Definition 1.3.6 (Matrix Reshaping)**.** $\text{mat}(\mathbf{v})$ is the inverse operation of $\operatorname{vec}(M)$, which reshapes the vector $\mathbf{v} \in \mathbb{R}^{ab\times 1}$ back to the matrix $M \in \mathbb{R}^{a\times b}$

$$\text{mat}(\mathbf{v}_{ab\times 1})_{a\times b} = M = \begin{pmatrix} v_1 & v_{a+1} & \cdots & v_{(b-1)a+1} \\ v_2 & v_{a+2} & \cdots & v_{(b-1)a+2} \\ \vdots & \vdots & \ddots & \vdots \\ v_a & v_{2a} & \cdots & v_{ba} \end{pmatrix} \in \mathbb{R}^{a\times b}$$

Definition 1.3.7. P_ϕ is a huge sparse matrix

$$P_\phi \in \mathbb{R}^{(hhd^{\text{in}})(a^{\text{out}}b^{\text{out}})\times d^{\text{in}}(a^{\text{in}}b^{\text{in}})}$$

and

$$\phi : \mathbb{R}^{d^{\text{in}}\times(a^{\text{in}}b^{\text{in}})} \to \mathbb{R}^{(hhd^{\text{in}})\times(a^{\text{out}}b^{\text{out}})}$$

Remark. P_ϕ 是用來做區塊提取的 (Patch Extraction Matrix), 將 $\operatorname{vec}(Z^{\text{in},i})$ 裡面所有 sub-image 提取出來，最後再經過 mat() 變回矩陣形式，這樣就可以讓卷積運算變成矩陣乘法

$$W\phi(\boldsymbol{Z}^{\text{in},i}) + b\mathbb{1}^T_{a^{\text{out}}b^{\text{out}}}$$

並且，使用 P_ϕ 的好處是可以很方便地計算反向傳播的梯度，因為 P_ϕ 是 Linear Mapping，可以直接利用矩陣微積分的規則來計算梯度，直接 Chain Rule 就可以了

$$\frac{\partial\xi}{\partial \operatorname{vec}(\boldsymbol{Z}^{\text{in},i})} = P_\phi^T \frac{\partial\xi}{\partial \operatorname{vec}(\phi(\boldsymbol{Z}^{\text{in},i}))}$$

1.3.4 Optimization Problem for CNN

We collect all parameters of the convolutional layer as

$$\theta = \begin{pmatrix} \text{vec}(W^1) \\ \boldsymbol{b}^1 \\ \vdots \\ \text{vec}(W^L) \\ \boldsymbol{b}^L \end{pmatrix} \in \mathbb{R}^n, \quad \text{where } n = \# \text{ total parameters}$$

The output of last layer L is vector $\boldsymbol{z}^{L+1,i}(\theta)$. Consider the loss function such as squared loss

$$\xi_i(\theta) = \|\boldsymbol{y}^i - \boldsymbol{z}^{L+1,i}(\theta)\|_2^2$$

Definition 1.3.8. The overall optimization problem for CNN is

$$\min_{\theta} f(\theta)$$

where

$$f(\theta) = \frac{1}{2C}\theta^T\theta + \frac{1}{l}\sum_{i=1}^{l} \xi(z^{L+1,i}(\theta); \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

where $C > 0$ is the regularization parameter, which is same as the fully-connected networks.

Note. We devide the sum of loss by l to get the average loss per sample. 這樣做的好處是當我們改變訓練資料的大小 l 時，不需要重新調整 learning rate，因為 average loss 已經把資料數量的影響給 normalize 掉了。

1.3.5 Padding Layers

Besides convolutional layers, CNNs often include other operations such as pooling layers and padding layers.

First, we discuss padding layers. To better control the output size of convolutional layers, before the convolution operation, we can enlarge the input image by adding extra 0 around the border. This is called **zero padding**.

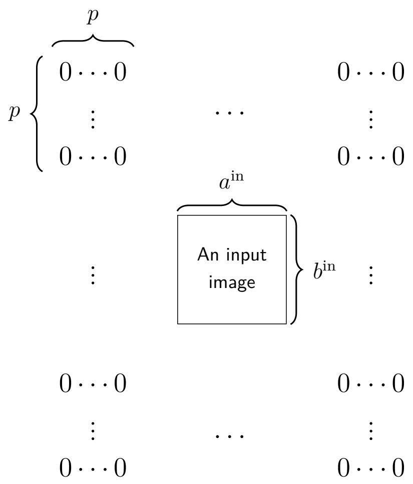


Figure 1.10: Zero Padding Layer Illustration

The size of the input image will increase from

$$a^{\text{in}} \times b^{\text{in}} \to (a^{\text{in}} + 2p) \times (b^{\text{in}} + 2p)$$

The operation can be treated as a layer of mapping an input $\boldsymbol{Z}^{\text{in},i}$ to an output $\boldsymbol{Z}^{\text{out},i}$.

Definition 1.3.9 (Padding Layer)**.** Assume that $d^{\text{in}} = d^{\text{out}}$ for all input, there must exist a $0/1$ matrix $P_{\text{pad}} \in \mathbb{R}^{d^{\text{out}}a^{\text{out}}b^{\text{out}} \times d^{\text{in}}a^{\text{in}}b^{\text{in}}}$ such that

$$\boldsymbol{Z}^{\text{out},i} \equiv \text{mat}\left(P_{\text{pad}}\,\text{vec}(\boldsymbol{Z}^{\text{in},i})\right)_{d^{\text{out}} \times (a^{\text{out}}b^{\text{out}})}$$

1.3.6 Pooling Layers

In practice, after convolutional layers, we often get very large output images which will increase the computational cost of the following layers. To reduce the cost, we can use pooling layers to down-sample the output images or do the approximation that can extract the main (invariant) rotation/translation features of the images. There are many types of pooling layers, such as max pooling, average pooling, and stochatic pooling. Here we only discuss max pooling.

Example. Here is an example of max pooling layer with pool size 2×2 and stride 2.

$$\text{image A :} \quad \left(\begin{array}{cc|cc} 2 & 3 & 6 & 8 \\ \boxed{5} & 4 & \boxed{9} & 7 \\ \hline 1 & 2 & \boxed{6} & 0 \\ \boxed{4} & 3 & 2 & 1 \end{array}\right) \xrightarrow[\text{pool size } 2\times 2, s=2]{\text{max pooling}} \begin{pmatrix} 5 & 9 \\ 4 & 6 \end{pmatrix}$$

$$\text{image B :} \quad \left(\begin{array}{cc|cc} 3 & 2 & 3 & 6 \\ 4 & \boxed{5} & 4 & \boxed{9} \\ \hline 2 & 1 & 2 & \boxed{6} \\ 3 & \boxed{4} & 3 & 2 \end{array}\right) \xrightarrow[\text{pool size } 2\times 2, s=2]{\text{max pooling}} \begin{pmatrix} 5 & 9 \\ 4 & 6 \end{pmatrix}$$

In this example, B is a translation horizontally for one pixel of A, we split the image into four 2×2 sub-images and take the maximum value of each sub-image as the output. In each sub-image, only some elements are changed, so the output maximum remains the same or similar, which is called **translation invariance**.

For the mathematical representation of max pooling layer, it can be treated as a mapping from input matrix $\boldsymbol{Z}^{\text{in},i}$ to output matrix $\boldsymbol{Z}^{\text{out},i}$. We can partition every channel of $\boldsymbol{Z}^{\text{in},i}$ into non-overlapping sub-matrices by $h \times h$ filters with stride $s = h$.

By the same idea of convolutional layers, we can generate a huge sparse matrix

$$\phi(\boldsymbol{Z}^{\text{in},i}) = \text{mat}\left(P_{\phi}\,\text{vec}(\boldsymbol{Z}^{\text{in},i})\right)_{hh \times d^{\text{in}}a^{\text{out}}b^{\text{out}}} \tag{1.5}$$

where

$$a^{\text{out}} = \left\lfloor \frac{a^{\text{in}} - h}{s} \right\rfloor + 1 = \left\lfloor \frac{a^{\text{in}}}{h} \right\rfloor, \quad b^{\text{out}} = \left\lfloor \frac{b^{\text{in}}}{h} \right\rfloor, \quad d^{\text{out}} = d^{\text{in}}$$

Note that we now consider $hh \times d^{\text{out}}a^{\text{out}}b^{\text{out}}$ but not $hhd^{\text{out}} \times a^{\text{out}}b^{\text{out}}$ because can then do the max operation channel by channel. To select the maximum value from each sub-image, there exists a $0/1$ matrix $M^i \in \mathbb{R}^{d^{\text{out}}a^{\text{out}}b^{\text{out}} \times hhd^{\text{out}}a^{\text{out}}b^{\text{out}}}$ so that each row of M^i has only one 1 corresponding to the maximum value's position from $\text{vec}(\phi(\boldsymbol{Z}^{\text{in},i}))$. Therefore, we have

$$\boldsymbol{Z}^{\text{out},i} = \text{mat}\left(M^i\,\text{vec}(\phi(\boldsymbol{Z}^{\text{in},i}))\right)_{d^{\text{out}} \times (a^{\text{out}}b^{\text{out}})} \tag{1.6}$$

Note. A comparison between convolutional layer and pooling layer is shown below

$$\boldsymbol{S}^{\text{out},i} = W\phi(\boldsymbol{Z}^{\text{in},i}) + \boldsymbol{b}\mathbb{1}^T_{a^{\text{out}}b^{\text{out}}}$$

M^i is a similar role as W in convolutional layers, while M^i is 0/1, which is not a constant, it is positions of 1's depend on the value of $\phi(\boldsymbol{Z}^{\text{in},i})$.

By combining (1.5) and (1.6), we have

Definition 1.3.10 (Max Pooling Layer).

$$\boldsymbol{Z}^{\text{out},i} = \text{mat}\left(P^i_{\text{pool}}\,\text{vec}(\boldsymbol{Z}^{\text{in},i})\right)_{d^{\text{out}}\times(a^{\text{out}}b^{\text{out}})}$$

where

$$P^i_{\text{pool}} = M^i P_\phi \in \mathbb{R}^{d^{\text{out}}a^{\text{out}}b^{\text{out}}\times d^{\text{in}}a^{\text{in}}b^{\text{in}}}$$

1.3.7 Summary of CNN Layers

In summary, padding layers, pooling layers are optional in a convolutional layer, but they are frequently used in practice. A general convolutional layer can be represented as follows:

$$\boldsymbol{Z}^{m,i} \rightarrow \text{ padding } \rightarrow \text{ convolution } \rightarrow \text{ pooling } \rightarrow \boldsymbol{Z}^{m+1,i}$$

where the m-th layer's input is $\boldsymbol{Z}^{m,i}$ and output is $\boldsymbol{Z}^{m+1,i}$. Let the following notations denote the image sizes at each step:

$$\begin{aligned} a^m, b^m &: \text{input image size} \\ a^m_{\text{pad}}, b^m_{\text{pad}} &: \text{padded image size} \\ a^m_{\text{conv}}, b^m_{\text{conv}} &: \text{convolved image size} \end{aligned}$$

Below are the operations and corresponding input/output for both image matrices and image sizes:

Operation	Input	Output
Padding	$\boldsymbol{Z}^{m,i}$	$\text{pad}(\boldsymbol{Z}^{m,i})$
Convolution	$\text{pad}(\boldsymbol{Z}^{m,i})$	$\boldsymbol{S}^{m,i}$
Activation	$\boldsymbol{S}^{m,i}$	$\sigma(\boldsymbol{S}^{m,i})$
Pooling	$\sigma(\boldsymbol{S}^{m,i})$	$\boldsymbol{Z}^{m+1,i}$

Operation	$a^{\text{in}}, b^{\text{in}}, d^{\text{in}}$	$a^{\text{out}}, b^{\text{out}}, d^{\text{out}}$
Padding	a^m, b^m, d^m	$a^m_{\text{pad}}, b^m_{\text{pad}}, d^m$
Convolution	$a^m_{\text{pad}}, b^m_{\text{pad}}, d^m$	$a^m_{\text{conv}}, b^m_{\text{conv}}, d^{m+1}$
Activation	$a^m_{\text{conv}}, b^m_{\text{conv}}, d^{m+1}$	$a^m_{\text{conv}}, b^m_{\text{conv}}, d^{m+1}$
Pooling	$a^m_{\text{conv}}, b^m_{\text{conv}}, d^{m+1}$	$a^{m+1}, b^{m+1}, d^{m+1}$

Definition 1.3.11 (General Convolutional Layer). Let the filter size, mapping matrices, weight matrices at m-th layer be

$$h^m, P^m_{\text{pad}}, P^m_\phi, W^m, \boldsymbol{b}^m$$

Then all operations of the m-th convolutional layer can be summarized as

$$\boldsymbol{S}^{m,i} = W^m \,\text{mat}\left(P^m_\phi\, P^m_{\text{pad}}\,\text{vec}(\boldsymbol{Z}^{m,i})\right)_{(h^m h^m d^m)\times a^m_{\text{conv}}b^m_{\text{conv}}} + \boldsymbol{b}^m\mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}}$$

$$\boldsymbol{Z}^{m+1,i} = \text{mat}\left(P^{m,i}_{\text{pool}}\,\text{vec}\left(\sigma(\boldsymbol{S}^{m,i})\right)\right)_{d^{m+1}\times a^{m+1}b^{m+1}}$$

1.3.8 Fully-connected Layers

Assume that there are L^C convolutional layersm the input of the first fully-connected layer is

$$\boldsymbol{z}^{m,i} = \text{vec}(\boldsymbol{Z}^{m,i}),\ i = 1, \ldots, l, \quad m = L^C + 1$$

In each fully-connected layer ($L^C < m \leq L$), we consider weight matrix and bias vector as

$$W^m \in \mathbb{R}^{n_{m+1} \times n_m}, \quad \boldsymbol{b}^m \in \mathbb{R}^{n_{m+1} \times 1}$$

Then the operations of fully-connected layers are the same as before, if $\boldsymbol{z}^{m,i} \in \mathbb{R}^{n_m}$ is the input vector, and $\boldsymbol{z}^{m+1,i} \in \mathbb{R}^{n_{m+1}}$ is the output vector:

$$\begin{aligned} \boldsymbol{s}^{m,i} &= W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m \\ \boldsymbol{z}_j^{m+1,i} &= \sigma(\boldsymbol{s}_j^{m,i}), \quad j = 1, 2, \ldots, n_{m+1} \end{aligned}$$

Due to objective function is non-convex, it may have multiple local minima. It's known that global optimization is much more difficult than local minimization. In practice, we usually just find a local minimum by gradient-based methods such as gradient descent or stochastic gradient method.

Chapter 2

Stochastic gradient methods for deep learning

2.1 Gradient Descent

> **As previously seen.** NN optimization problem:
>
> $$\min_{\boldsymbol{\theta}} \; f(\boldsymbol{\theta})$$
>
> where
>
> $$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l}\xi(z^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$
>
> which can be simplify as
>
> $$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

Now the issue is how to solve this optimization problem, here is a simple method called gradient descent.

Definition 2.1.1 (Gradient Descent)**.** We let the first order approximation

$$f(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta}$$

where

$$\nabla f(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} \end{pmatrix}$$

The problem is to solve

$$\begin{aligned} &\min_{\Delta\boldsymbol{\theta}} \nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta} \\ &\text{subject to } \|\Delta\boldsymbol{\theta}\| = 1 \end{aligned} \tag{2.1}$$

to find the direction of update $\Delta\boldsymbol{\theta}$.

> **Note.** The constraint $\|\Delta\boldsymbol{\theta}\| = 1$ is needed to prevent the solution from goes to $-\infty$

Note. 單變數函數 $f(x)$ 的一階近似為

$$f(x+\Delta x) \approx f(x) + f'(x)\Delta x$$

要讓 $f(x+\Delta x)$ 盡可能小，我們需要讓 $f'(x)\Delta x$ 越小越好，因此 Δx 的方向應該與 $f'(x)$ 相反，即 $\Delta x = -kf'(x), k > 0$，推到多變數函數上也是一樣的道理，多變數的一階近似是

$$\begin{aligned} f(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) &\approx f(\boldsymbol{\theta}) + \sum_i \left(\frac{\partial f}{\partial \theta_i} \cdot \Delta\theta_i\right) \\ &= f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} \end{aligned}$$

Theorem 2.1.1 (Gradient Descent Direction)**.** The solution of (2.1) is

$$\Delta\boldsymbol{\theta} = -\frac{\nabla f(\boldsymbol{\theta})}{\|\nabla f(\boldsymbol{\theta})\|} \tag{2.2}$$

, which is called the **steepest descent direction**（最陡下降方向）.

However, because we only consider an approximation, which may not strictly decrease the function value, i.e.

$$f(\boldsymbol{\theta}) < f(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})$$

might happen. But in general we need the desent property to get convergence, we do the Taylor expansion

$$f(\boldsymbol{\theta}+\alpha\Delta\boldsymbol{\theta}) = f(\boldsymbol{\theta}) + \underbrace{\alpha\nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta}}_{\text{critical one while } \alpha\to 0} + \frac{\alpha^2}{2}\Delta\boldsymbol{\theta}^T\nabla^2 f(\boldsymbol{\theta})\Delta\boldsymbol{\theta} + o(\alpha^2)$$

where

$$\nabla^2 f(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1^2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1 \partial \boldsymbol{\theta}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n \partial \boldsymbol{\theta}_1} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n^2} \end{pmatrix}$$

is the Hessian of f at $\boldsymbol{\theta}$.
If

$$\nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta} < 0$$

then a small enough α can ensure the descent property (the higher order terms can be ignored).

$$f(\boldsymbol{\theta}+\alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta})$$

Thus in the optimization algorithm, it is called a desent direction if it satisfies

$$\nabla f(\boldsymbol{\theta})^T\Delta\boldsymbol{\theta} < 0$$

The direction chose in (2.2) is indeed a descent direction.

$$-\nabla f(\boldsymbol{\theta})^T \frac{\nabla f(\boldsymbol{\theta})}{\|\nabla f(\boldsymbol{\theta})\|} = -\|\nabla f(\boldsymbol{\theta})\| < 0$$

2.1.1 Line Search

We have known that we have to find a step size α such that

$$f(\boldsymbol{\theta}+\alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta})$$

In optimization, this process is called **line search**.

Definition 2.1.2 (Exact Line Search). For a one-dimensional optimization problem

$$\min_{\alpha} \; f(\boldsymbol{\theta} + \alpha \Delta \boldsymbol{\theta})$$

In practice, people usually use **backtracking line search** or other inexact line search methods to find a suitable step size α with a fixed parameter set $\Delta\boldsymbol{\theta}$. We check

$$\alpha = 1, \beta, \beta^2, \dots$$

with $\beta \in (0, 1)$ until

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu \nabla f(\boldsymbol{\theta})^T (\alpha \Delta \boldsymbol{\theta})$$

where $\nu \in (0, 1/2)$.

Definition 2.1.3 (Armijo Condition). A step size α is said to satisfy the **Armijo condition** if

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu \nabla f(\boldsymbol{\theta})^T (\alpha \Delta \boldsymbol{\theta})$$

Note (Armijo Condition). The condition

$$f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) < f(\boldsymbol{\theta}) + \nu \nabla f(\boldsymbol{\theta})^T (\alpha \Delta \boldsymbol{\theta})$$

is called the **Armijo condition**, which is come from the Taylor expansion

$$\begin{aligned} f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) &\approx f(\boldsymbol{\theta}) + \alpha \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} \\ \Rightarrow f(\boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta}) - f(\boldsymbol{\theta}) &\approx \alpha \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} < \nu(\alpha \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta}) \end{aligned}$$

Since $\nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} < 0$, we have

$$\underbrace{\alpha \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta}}_{\text{Directional Derivative of Taylor Expansion's 1st order term}} < \nu(\alpha \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta})$$

Thus the Armijo condition ensures the descent property. 實際下降有到達 ν 倍的方向導數，就代表這個步長 α 是足夠的.

The convergence of gradient descent with backtracking line search can be guaranteed.

Corollary 2.1.1. Every limit point $\bar{\boldsymbol{\theta}}$ of a convergent subsequence of $\{\boldsymbol{\theta}^k\}$ generated by the gradient descent method with backtracking line search, we have

$$\nabla f(\bar{\boldsymbol{\theta}}) = 0$$

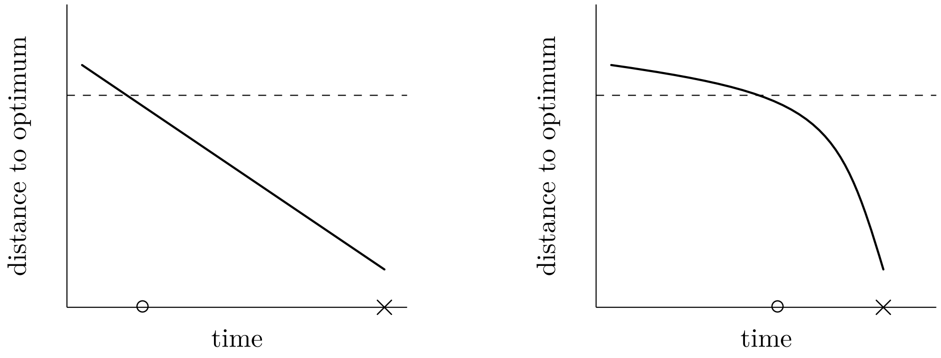
which means $\bar{\boldsymbol{\theta}}$ is a stationary point of non-convex problem.

2.1.2 Practical Gradient Descent Algorithm

It is known that the convergence of gradient descent is slow for difficult problems, thus in practice, method of using **second-order information** such as Newton's method or Quasi-Newton method are preferred.

$$f(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})^T \Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^T \nabla^2 f(\boldsymbol{\theta}) \Delta\boldsymbol{\theta}$$

These methods have faster **final** convergence rate, below is an illustration of convergence rates.



(a) Linear convergence (b) Quadratic convergence

Figure 2.1: Convergence Rates of Different Methods

But in deep learning, fast final convergence may not be that important, since the optimal solution $\boldsymbol{\theta}^*$ is not necessarily the best generalization solution.

2.2 Stochastic Gradient Method

2.2.1 Estimation of Gradient

> **As previously seen.** The objective function is
>
> $$f(\boldsymbol{\theta}) = \frac{1}{2C}\boldsymbol{\theta}^T\boldsymbol{\theta} + \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$
>
> The gradient is
>
> $$\frac{\boldsymbol{\theta}}{C} + \frac{1}{l}\nabla_{\boldsymbol{\theta}}\left(\sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})\right) = \frac{\boldsymbol{\theta}}{C} + \frac{1}{l}\sum_{i=1}^{l} \nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$
>
> > **Note.** $\nabla_{\boldsymbol{\theta}}$ denotes the gradient with respect to $\boldsymbol{\theta}$, which is used to compute the gradient of function ξ only.

Going over all data is time-consuming, thus if data are from same distribution

$$\mathbb{E}(\nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^1)) = \frac{1}{l}\sum_{i=1}^{l} \nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

then we can just use a **subset** of data S, which is often called a **batch**, to estimate the gradient

$$\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\sum_{i:i\in S} \nabla_{\boldsymbol{\theta}}\xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

With the estimated gradient, and the for η is the **learning rate** or **step size**, which is hard to choose in practice. We can get the **stochastic gradient algorithm** as below.

Algorithm 2.1: Stochastic Gradient Algorithm

```
Given an initial learning rate $\eta$.
while do
    Choose $S \subset \{1, \ldots, l\}$.
    Calculate
        $$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \left( \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i: i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i}) \right)$$
    May adjust the learning rate $\eta$
end
```

By the name of stochastic gradient, we know that the gradient used here is a random estimation of the true gradient. Compared to the true gradient, this method does not do “searching” for a good step size α. Without the full gradient information, the decrease condition such as Armijo condition cannot be guaranteed. Thus, we don't have an algorithm with “decent” property, i.e. it is possible for

$$f(\boldsymbol{\theta}^{\text{next}}) > f(\boldsymbol{\theta})$$

2.2.2 Momentum Method

To improve the convergence speed of stochastic gradient method, we can use the **momentum method**.

Definition 2.2.1 (Momentum Method)**.** Let $\boldsymbol{v}$ be the velocity vector, initialized as $\boldsymbol{v} = 0$. The momentum method updates $\boldsymbol{\theta}$ as follows:

$$\begin{aligned} \boldsymbol{v} &\leftarrow \alpha \boldsymbol{v} - \eta \left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i: i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \right) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{aligned}$$

with $\alpha \in [0, 1)$ being the momentum parameter.

In this method, we use the previous gradient information to smooth the update direction. We use exponential moving average to compute the $\boldsymbol{\theta}$, i.e expend the recurrence:

$$\begin{aligned} \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} &- \eta \text{ (current sub-gradient)} \\ &- \alpha\eta \text{ (previous sub-gradient)} \\ &- \alpha^2\eta \text{ (sub-gradient before previous)} \end{aligned}$$

The effect of the sub-gradient from earlier iterations decays exponentially with factor $\alpha \in [0, 1)$. The data we select in each iteration is to approximate the true gradient, thus the resulting update direction is noisy. By using momentum (averaging), we can reduce the variance of the update direction, thus the convergence speed is improved. However, the rule of Definition 2.2.1 might be biased since the initial velocity, which will disscussed in later lectures.

2.2.3 AdaGrad (Adaptive Gradient Algorithm)

> Scaling learning rates inversely proportional to the **square root of sum of past gradient squares**. (Duchi et al., 2011)

The idea of AdaGrad is to adapt the learning rate for each parameter according to the historical gradient information.

> **Notation** ($\odot$). For two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$, the **Hadamard product** (element-wise product) is defined as
>
> $$\boldsymbol{a} \odot \boldsymbol{b} = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{pmatrix}$$

> **Definition 2.2.2** (AdaGrad). Let $\boldsymbol{g}$ be the stochastic gradient, $\boldsymbol{r}$ be the adjusted gradient. AdaGrad updates $\boldsymbol{\theta}$ as follows:
>
> $$\begin{aligned} \boldsymbol{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i: i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \\ \boldsymbol{r} &\leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \boldsymbol{g} \end{aligned}$$
>
> where ϵ, δ are hyperparameters and $\boldsymbol{r}$ is the accumulated squared gradient.

The learning rate for each parameter is scaled by $\frac{1}{\boldsymbol{r}}$, which means that if a parameter has a large historical gradient, its learning rate will be reduced, and vice versa.

> **Note.** Square is used to ensure all movements of parameters not be canceled out. δ is used to prevent division by zero.

For this computation, we can let the frequently updated parameters have smaller learning rates, while the infrequently updated parameters have larger learning rates, which is to let learner take notice on those infrequently features.

> **Recall.** In linear classification problem, the optimization problem is
>
> $$\frac{\boldsymbol{w}^T \boldsymbol{w}}{2} + C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$$

For method like SVM or logistic regression, the loss function can be written as a function of $\boldsymbol{w}^T \boldsymbol{x}$.

$$\xi(\boldsymbol{w}; y, \boldsymbol{x}) = \hat{\epsilon}(\boldsymbol{w}^T \boldsymbol{x})$$

Then the gradient is

$$\nabla_{\boldsymbol{w}} \xi(\boldsymbol{w}; y, \boldsymbol{x}) = \boldsymbol{w} + C \sum_{i=1}^{l} \hat{\epsilon}'(\boldsymbol{w}^T \boldsymbol{x}_i) \boldsymbol{x}_i$$

The expression shows that

$$\nabla \boldsymbol{w} = \boldsymbol{w} + C \sum (\text{error}) \cdot \mathbf{x}_i$$

which indicates that the gradient is affected by the **density** of features.

However, the above analysis is not suitable for "non-convex" case such as neural networks. Empirically, AdaGrad will meet the problem of "too fast decay of learning rate", thus other adaptive methods such as RMSProp or Adam are proposed to solve this problem.

2.2.4 RMSProp (Relative Mean Square Propagation)

For convex problems, such as SVM or logistic regression, AdaGrad works well since the objective function is convex. However, for non-convex problems such as neural networks, AdaGrad's learning rate may be too small before reaching the convex region, i.e. $\boldsymbol{r}$ becomes too large while not reaching the optimal solution in the non-convex case. Thus, RMSProp is proposed to solve this problem, which uses **exponential weighted moving average** to compute the accumulated squared gradient.

> **Definition 2.2.3** (RMSProp)**.** Let $\boldsymbol{g}$ be the stochastic gradient, $\boldsymbol{r}$ be the adjusted gradient. RMSProp updates $\boldsymbol{\theta}$ as follows:
>
> $$\begin{aligned} \boldsymbol{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \\ \boldsymbol{r} &\leftarrow \rho \boldsymbol{r} + (1-\rho)(\boldsymbol{g}\odot \boldsymbol{g}) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}+\delta} \odot \boldsymbol{g} \end{aligned}$$
>
> where ϵ, δ are hyperparameters, $\rho \in [0,1)$ is the decay rate, and $\boldsymbol{r}$ is the exponentially weighted moving average of squared gradients.

However, RMSProp is somehow heuristic compared to AdaGrad, since there is no theoretical guarantee for its convergence.

2.2.5 Adam (Adaptive Moment Estimation)

> **Definition 2.2.4** (Adam)**.** Let $\boldsymbol{g}$ be the stochastic gradient, $\boldsymbol{s}$ be the first moment vector, $\boldsymbol{r}$ be the second moment vector. Adam updates $\boldsymbol{\theta}$ as follows:
>
> $$\begin{aligned} \boldsymbol{g} &\leftarrow \frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \\ \boldsymbol{s} &\leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g} \\ \boldsymbol{r} &\leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)(\boldsymbol{g}\odot \boldsymbol{g}) \\ \hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t} \\ \hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}}+\delta} \odot \hat{\boldsymbol{s}} \end{aligned}$$
>
> where t is the iteration number.

Roughly speaking, Adam combines the ideas of momentum method and RMSProp. But the use of

$$\frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}}+\delta} \odot \hat{\boldsymbol{s}}$$

is somewhat heuristic, and there is no theoretical guarantee for better convergence. However, Adam works fairly robust to the choice of hyperparameters in practice. However, recent research shows that Adam actually find worse solutions than SGD in some cases.

Now we consider

$$\begin{aligned}\hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}\\ \hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}\end{aligned}$$

which are called "**bias correction**".

Note that we use $\boldsymbol{s}$ as the direction of update $\boldsymbol{\theta}$. We hope that its expectation is similar to the expected gradient, i.e.

$$\begin{cases}\mathbb{E}[\boldsymbol{s}_t] = \mathbb{E}[\boldsymbol{g}_t]\\ \mathbb{E}[\boldsymbol{r}_t] = \mathbb{E}[\boldsymbol{g}_t \odot \boldsymbol{g}_t]\end{cases}$$

where t is the iteration number. However, since $\boldsymbol{s}$ and $\boldsymbol{r}$ are affected a lot by the initial value by the moving average, i.e. the vector is biased towards the initial value (which is $\mathbf{0}$).

Theorem 2.2.1. The bias correction of $\boldsymbol{s}_t$ is

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$$

Proof. For $\boldsymbol{s}_t$ we have

$$\begin{aligned}\boldsymbol{s}_t &= \rho_1 \boldsymbol{s}_{t-1} + (1-\rho_1)\boldsymbol{g}_t\\ &= \rho_1(\rho_1 \boldsymbol{s}_{t-2} + (1-\rho_1)\boldsymbol{g}_{t-1}) + (1-\rho_1)\boldsymbol{g}_t\\ &= (1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}\boldsymbol{g}_i\end{aligned}$$

Then,

$$\mathbb{E}[\boldsymbol{s}_t] = \mathbb{E}\left[(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}\boldsymbol{g}_i\right] = \mathbb{E}[\boldsymbol{g}_i]\cdot(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}$$

Note that we assume

$$\mathbb{E}[\boldsymbol{g}_i] = \mathbb{E}[\boldsymbol{g}_t], \forall i \le t$$

Thus,

$$(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i} = (1-\rho_1)(1+\rho_1+\rho_1^2+\cdots+\rho_1^{t-1}) = 1-\rho_1^t$$

Hence,

$$\mathbb{E}[\boldsymbol{s}_t] = (1-\rho_1^t)\mathbb{E}[\boldsymbol{g}_t]$$

Therefore, to correct the bias, we have

$$\hat{\boldsymbol{s}}_t = \frac{\boldsymbol{s}_t}{1-\rho_1^t} \Rightarrow \mathbb{E}[\hat{\boldsymbol{s}}_t] = \mathbb{E}[\boldsymbol{g}_t]$$

■

Use the same idea, we can prove the bias correction of $\boldsymbol{r}_t$.

Theorem 2.2.2. The bias correction of $\boldsymbol{r}_t$ is

$$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$$

Proof. For $\boldsymbol{r}_t$ we have

$$\begin{aligned}\boldsymbol{r}_t &= \rho_2 \boldsymbol{r}_{t-1} + (1-\rho_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \\ &= \rho_2(\rho_2 \boldsymbol{r}_{t-2} + (1-\rho_2)(\boldsymbol{g}_{t-1} \odot \boldsymbol{g}_{t-1})) + (1-\rho_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \\ &= (1-\rho_2)\sum_{i=1}^{t} \rho_2^{t-i}(\boldsymbol{g}_i \odot \boldsymbol{g}_i)\end{aligned}$$

Then,

$$\mathbb{E}[\boldsymbol{r}_t] = \mathbb{E}\left[(1-\rho_2)\sum_{i=1}^{t} \rho_2^{t-i}(\boldsymbol{g}_i \odot \boldsymbol{g}_i)\right] = \mathbb{E}[\boldsymbol{g}_i \odot \boldsymbol{g}_i] \cdot (1-\rho_2)\sum_{i=1}^{t} \rho_2^{t-i}$$

Note that we assume

$$\mathbb{E}[\boldsymbol{g}_i \odot \boldsymbol{g}_i] = \mathbb{E}[\boldsymbol{g}_t \odot \boldsymbol{g}_t], \forall i \leq t$$

Thus,

$$(1-\rho_2)\sum_{i=1}^{t} \rho_2^{t-i} = (1-\rho_2)(1+\rho_2+\rho_2^2+\cdots+\rho_2^{t-1}) = 1-\rho_2^t$$

Hence,

$$\mathbb{E}[\boldsymbol{r}_t] = (1-\rho_2^t)\mathbb{E}[\boldsymbol{g}_t \odot \boldsymbol{g}_t]$$

■

Here is an interesting story about BERT, which is an important NLP technique using Adam as the optimizer. The original BERT forget to use bias correction, which seems to cause lengthy training iterations.

2.2.6 AdamW (Adam with Weight Decay)

Now we consider the weight decay term,

Recall. The update of stochastic gradient method is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, Z^{1,i})\right)$$

In this case,

$$\frac{\boldsymbol{\theta}}{C}$$

come from the regularization term $\boldsymbol{\theta}^T\boldsymbol{\theta}/2$ in $f(\boldsymbol{\theta})$. However, in neural networks, people usually use **weight decay** to replace the regularization term, which is implemented as

$$\boldsymbol{\theta} \leftarrow (1-\lambda)\boldsymbol{\theta} - \eta\left(\frac{1}{|S|}\nabla_{\boldsymbol{\theta}} \sum_{i:i\in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})\right)$$

where λ is the rate of weight decay factor. There is no a theoretical reason for this modification. But clearly, if

$$\lambda = \frac{\eta}{C}$$

then weight decay is equivalent to the regularization. However, if using adaptive methods such as Adam, the equivalence does not hold. For instance, in AdaGrad, the update is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \left(\frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \right) - \frac{\epsilon}{\sqrt{\boldsymbol{r}} + \delta} \odot \frac{\boldsymbol{\theta}}{C}$$

so the regularization term is scaled in a component-wise way. Hence, we try to decouple the weight decay step from the gradient step. For example, for momentum method

$$\begin{aligned} \boldsymbol{v} &\leftarrow \alpha \boldsymbol{v} - \eta \left(\frac{\boldsymbol{\theta}}{C} + \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \right) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v} \end{aligned}$$

we use weight decay as the equivalence

$$\begin{aligned} \boldsymbol{v} &\leftarrow \alpha \boldsymbol{v} - \eta \left(\frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \right) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{v} - \eta \frac{\boldsymbol{\theta}}{C} \end{aligned}$$

This modification is called AdamW when using Adam as the optimizer.

Definition 2.2.5 (AdamW)**.** Let $\boldsymbol{g}$ be the stochastic gradient, $\boldsymbol{s}$ be the first moment vector, $\boldsymbol{r}$ be the second moment vector. AdamW updates $\boldsymbol{\theta}$ as follows:

$$\begin{aligned} \boldsymbol{g} &\leftarrow \frac{1}{|S|} \nabla_{\boldsymbol{\theta}} \sum_{i:i \in S} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i}) \\ \boldsymbol{s} &\leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g} \\ \boldsymbol{r} &\leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)(\boldsymbol{g} \odot \boldsymbol{g}) \\ \hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t} \\ \hat{\boldsymbol{r}} &\leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t} \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\sqrt{\hat{\boldsymbol{r}}} + \delta} \odot \hat{\boldsymbol{s}} - \epsilon \frac{\boldsymbol{\theta}}{C} \end{aligned}$$

AdamW is not equivalent to Adam with $\boldsymbol{\theta}/C$ is added after the gradient calculation. There is the disscussion about why AdamW works better than Adam in practice in the paper Decoupled Weight Decay Regularization.

2.2.7 Why Stochastic Gradient?

We can simply conclude four reasons to use stochastic gradient methods in practice.

1° **Final convergence is not important.** In deep learning, the optimal solution $\boldsymbol{\theta}^*$ may not lead to the best model. Futher, we don't need to reach the optimal solution, instead, we find

$$\arg\max_k z_k^{L+1}(\boldsymbol{\theta})$$

, so not-so-accurate solution is acceptable.

2° **Cheap iterations.** There is a property for data classification in statistics

$$\mathbb{E}[\nabla_{\boldsymbol{\theta}} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})] = \frac{1}{l} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{l} \xi(\boldsymbol{\theta}; \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

thus, we can cheaply estimate the gradient by using a subset of data. Indeed, stochastic gradient is less used outside deep learning.

> **Remark.** Nowadays, in complicated models we stochastic gradient is simpler than calculating the full gradient with Hessian. Gradient calculation of stochastic gradient can be easily implemented by **automatic differentiation** which will discussed later.

3° **Non-convex problems.** For convex problems, second-order methods such as Newton's method or Quasi-Newton method are preferred since they find global optimal solutions more efficiently. However, for non-convex problems such as neural networks, efficiency to find stationary points is less important since local optimal solutions may generalize better.

All these are the reasons why stochastic gradient methods are widely used in deep learning.

2.3 Convergence of Stochastic Gradient Method

For the simplicity, we don't consider the regularization term. Therefore,

$$f(\boldsymbol{\theta}) = \frac{1}{l}\sum_{i=1}^{l} \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

and we define

$$f_i(\boldsymbol{\theta}) := \xi(\boldsymbol{z}^{L+1,i}(\boldsymbol{\theta}); \boldsymbol{y}^i, \boldsymbol{Z}^{1,i})$$

In here we consider the simplest case of stochastic gradient method, at each iteration, an index $\tilde{i}$ such that

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)$$

with t is the iteration index. Then with the following assumptions:

- limit of gradient, there exists a constant $G > 0$ such that

$$\|\nabla f_i(\boldsymbol{\theta})\| \leq G, \quad \forall i, \forall \boldsymbol{\theta} \tag{1}$$

- limit of Hessian, there exists is a constant $L > 0$ such that

$$|\boldsymbol{u}^T \nabla^2 f(\boldsymbol{\theta}) \boldsymbol{u}| \leq L\|\boldsymbol{u}\|^2, \quad \forall \boldsymbol{u}, \forall \boldsymbol{\theta} \tag{2}$$

Theorem 2.3.1 (Taylor Expansion Theorem). Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is two times differentiable, then

$$f(\boldsymbol{x}+\boldsymbol{h}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{h} + \frac{1}{2}\boldsymbol{h}^T \underbrace{\nabla^2 f(\boldsymbol{\xi})}_{\text{Hessian}} \boldsymbol{h}$$

From Taylor's theorem, we have

$$\begin{aligned} f(\boldsymbol{\theta}_{t+1}) &= f(\boldsymbol{\theta}_t - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)) \\ &= f(\boldsymbol{\theta}_t) - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}(\alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t))^T \nabla^2 f(\boldsymbol{\xi}_t)(\alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)) \end{aligned} \tag{3}$$

where $\boldsymbol{\xi}_t$ is between $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t+1}$
Using assumption we get

$$\begin{aligned} f(\boldsymbol{\theta}_{t+1}) &= f(\boldsymbol{\theta}_t) - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) + \frac{1}{2}(\alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t))^T \nabla^2 f(\boldsymbol{\xi}_t)(\alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)) \\ &\leq f(\boldsymbol{\theta}_t) - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) + \frac{\alpha_t^2 L}{2}\|\nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)\|^2 && \text{(by assumption (2))} \\ &\leq f(\boldsymbol{\theta}_t) - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) + \frac{\alpha_t^2 L G^2}{2} && \text{(by assumption (1))} \end{aligned}$$

Earlier in gradient descent method, we have

$$-\alpha_t \nabla f(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) < 0$$

with small enough step size α_t, we have

$$f(\boldsymbol{\theta}_{t+1}) < f(\boldsymbol{\theta}_t)$$

However the term

$$-\alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t)$$

is not necessarily negative, since $\nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)$ is a random gradient. So we calculate the expectation. The expectation is on the randomness of selecting $\tilde{i}_t$

$$\begin{aligned}\mathbb{E}\left[f(\boldsymbol{\theta}_{t+1})\right] &\leq \mathbb{E}\left[f(\boldsymbol{\theta}_t) - \alpha_t \nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) + \frac{\alpha_t^2 G^2 L}{2}\right] \\ &= \mathbb{E}\left[f(\boldsymbol{\theta}_t)\right] - \alpha_t \mathbb{E}\left[\nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t)\right] + \frac{\alpha_t^2 G^2 L}{2} \qquad (4)\end{aligned}$$

> **Note.** α_t depends only on t, not on the randomness of selecting $\tilde{i}_t$, so it can be taken out of the expectation.

Our expectation is on $\tilde{i}_t$, $\forall t$, so we have

$$\mathbb{E}\left[f(\boldsymbol{\theta}_{t+1})\right] = \mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_t}\left[f(\boldsymbol{\theta}_{t+1})\right]$$

Thus in RHS, we still have randomness on $\tilde{i}_0, \ldots, \tilde{i}_{t-1}$, so we need $\mathbb{E}\left[f(\boldsymbol{\theta})\right]$ instead of $f(\boldsymbol{\theta})$ because

$$\mathbb{E}\left[f(\boldsymbol{\theta}_t)\right] = \mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_{t-1}}\left[f(\boldsymbol{\theta}_t)\right]$$

Next, we calculate the term $\mathbb{E}\left[\nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t)\right]$ by checking the expected value of $\nabla f_{\tilde{i}}(\boldsymbol{\theta}_t)$ given $\boldsymbol{\theta}_t$:

$$\begin{aligned}\mathbb{E}_{\tilde{i}_t}\left[\nabla f_{\tilde{i}_t}(\boldsymbol{\theta}_t) \mid \boldsymbol{\theta}_t\right] &= \sum_{i=1}^{l} \nabla f_i(\boldsymbol{\theta}_t) \cdot \Pr\left[\tilde{i}_t = i \mid \boldsymbol{\theta}_t\right] \\ &= \sum_{i=1}^{l} \nabla f_i(\boldsymbol{\theta}_t) \cdot \frac{1}{l} \\ &= \nabla f(\boldsymbol{\theta}_t)\end{aligned}$$

Therefore, we have

$$\mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_t}\left[\nabla f_{\tilde{i}_t}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t)\right] = \mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_{t-1}}\left[\mathbb{E}_{\tilde{i}_t}\left[\nabla f_{\tilde{i}_t}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) \mid \tilde{i}_0, \ldots, \tilde{i}_{t-1}\right]\right]$$

we can reduce $\mathbb{E}\left[\cdots \mid \tilde{i}_0, \ldots, \tilde{i}_{t-1}\right] \to \mathbb{E}\left[\cdots \mid \boldsymbol{\theta}_t\right]$ since $\boldsymbol{\theta}_t$ is determined by $\tilde{i}_0, \ldots, \tilde{i}_{t-1}$, thus we have

$$\mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_{t-1}}\left[\mathbb{E}_{\tilde{i}_t}\left[\nabla f_{\tilde{i}_t}(\boldsymbol{\theta}_t)^T \nabla f(\boldsymbol{\theta}_t) \mid \boldsymbol{\theta}_t\right]\right] = \mathbb{E}_{\tilde{i}_0,\ldots,\tilde{i}_{t-1}}\left[\|\nabla f(\boldsymbol{\theta}_t)\|^2\right]$$

Substituting back to (4), we have

$$\mathbb{E}\left[f(\boldsymbol{\theta}_{t+1})\right] \leq \mathbb{E}\left[f(\boldsymbol{\theta}_t)\right] - \alpha_t \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|^2\right] + \frac{\alpha_t^2 G^2 L}{2}$$

Then we rearrange the terms and sum up over T iterations:

$$\begin{aligned}
\sum_{t=0}^{T-1} \alpha_t \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|^2\right] &\leq \sum_{t=0}^{T-1} \left(\mathbb{E}\left[f(\boldsymbol{\theta}_t)\right] - \mathbb{E}\left[f(\boldsymbol{\theta}_{t+1})\right] + \frac{\alpha_t^2 G^2 L}{2}\right) \\
&= \mathbb{E}\left[f(\boldsymbol{\theta}_0)\right] - \mathbb{E}\left[f(\boldsymbol{\theta}_T)\right] + \frac{G^2 L}{2}\sum_{t=0}^{T-1}\alpha_t^2 \\
&= f(\boldsymbol{\theta}_0) - \mathbb{E}\left[f(\boldsymbol{\theta}_T)\right] + \frac{G^2 L}{2}\sum_{t=0}^{T-1}\alpha_t^2 \\
&\leq f(\boldsymbol{\theta}_0) - f^* + \frac{G^2 L}{2}\sum_{t=0}^{T-1}\alpha_t^2 \qquad (5)
\end{aligned}$$

where f^* is the global optimal value of $f(\boldsymbol{\theta})$. The LHS is of all T iterations, we need to rewrite it in single iteration form.
Assume we randomly select an iteration index τ, and τ is uniformly distributed in $\{0, 1, \ldots, T-1\}$

$$\Pr[\tau = t] = \frac{\alpha_t}{\sum_{k=0}^{T-1}\alpha_k}$$

The expected squared-norm of gradient at iteration τ is

$$\begin{aligned}
\mathbb{E}_{\tau, \tilde{i}_0, \ldots, \tilde{i}_{\tau-1}}\left[\|\nabla f(\boldsymbol{\theta}_\tau)\|^2\right] &= \sum_{t=0}^{T-1} \Pr[\tau = t] \cdot \mathbb{E}_{\tilde{i}_0, \ldots, \tilde{i}_{t-1}}\left[\|\nabla f(\boldsymbol{\theta}_t)\|^2\right] \\
&= \left(\sum_{k=0}^{T-1}\alpha_k\right)^{-1} \sum_{t=0}^{T-1}\alpha_t \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|^2\right] \qquad (6)
\end{aligned}$$

From (6) and (5), we have

$$\mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_\tau)\|^2\right] \leq \left(\sum_{k=0}^{T-1}\alpha_k\right)^{-1}\left(f(\boldsymbol{\theta}_0) - f^* + \frac{G^2 L}{2}\sum_{t=0}^{T-1}\alpha_t^2\right) \qquad (7)$$

If the learning rate is a constant, i.e., $\alpha_t = \alpha$, then

$$\begin{aligned}
\mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_\tau)\|^2\right] &\leq (T\alpha)^{-1}\left(f(\boldsymbol{\theta}_0) - f^* + \frac{G^2 L}{2} T\alpha^2\right) \\
&= \frac{f(\boldsymbol{\theta}_0) - f^*}{\alpha T} + \frac{\alpha G^2 L}{2}
\end{aligned}$$

The RHS does not go to zero as $T \to \infty$ due to the second term. However, we can choose to **diminish step size (learning rate) over time**. To make RHS of (7) go to zero, we need

$$\sum_{t=0}^{T-1}\alpha_t \text{ grows faster than } \sum_{t=0}^{T-1}\alpha_t^2$$

Lemma 2.3.1. Let

$$\alpha_t = \frac{1}{\sqrt{t+1}}$$

we have

$$\mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_\tau)\|^2\right] = O\left(\frac{\log T}{\sqrt{T}}\right) \to 0 \text{ as } T \to \infty$$

Proof. We have

$$\begin{aligned}\sum_{t=0}^{T-1} \alpha_t &= \sum_{t=0}^{T-1} \frac{1}{\sqrt{t+1}} \\ &\approx \int_0^T \frac{1}{\sqrt{x}}\,\mathrm{d}x = 2x^{1/2}\Big|_0^T = 2\sqrt{T}\end{aligned}$$

and

$$\begin{aligned}\sum_{t=0}^{T-1} \alpha_t^2 &= \sum_{t=0}^{T-1} \frac{1}{t+1} \\ &\approx \int_1^{T+1} \frac{1}{x}\,\mathrm{d}x = \log x\Big|_1^{T+1} = \log(T+1)\end{aligned}$$

Therefore, we have

$$\begin{aligned}\mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_\tau)\|^2\right] &\leq \left(2\sqrt{T}\right)^{-1}\left(f(\boldsymbol{\theta}_0) - f^* + \frac{G^2L}{2}\log(T+1)\right) \\ &= \frac{2(f(\boldsymbol{\theta}_0) - f^*) + G^2L\log(T+1)}{4\sqrt{T}} \\ &= O\left(\frac{\log T}{\sqrt{T}}\right) \to 0 \text{ as } T \to \infty\end{aligned}$$

The RHS goes to zero as $T \to \infty$. ■

Hence, by diminishing learning rate, the stochastic gradient method can also converge to a stationary point.

Chapter 3

Gradient Calculation

For a regular deep learning course, usually the contents will be

- Fully-connected Neural Network
- Optimization Problem of FNN
- Gradient Calculation of FNN (Backpropagation)
- ...
- Other types of Neural Networks (CNN, RNN, Transformer, ...)
- ...

However, there is a significant gap between FNN and other types of Neural Networks (e.g. CNN). Therefore, we discuss the gradient calculation (backpropagation) in detail.

3.1 Vector Form

Consider two layer m and layer $m+1$. The variables between them are W^m and $\boldsymbol{b}^m$, so we aim to calculate

$$\frac{\partial f}{\partial W^m} = \frac{1}{C} W^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial W^m} \tag{1}$$

$$\frac{\partial f}{\partial \boldsymbol{b}^m} = \frac{1}{C} \boldsymbol{b}^m + \frac{1}{l} \sum_{i=1}^{l} \frac{\partial \xi_i}{\partial \boldsymbol{b}^m} \tag{2}$$

to get the average gradient of the loss function ξ over the training set.

Note that (1) now is in matrix form, and (2) is in vector form. But it will be easier to transform them to a vector form for the derivation (gradient).

> **Recall.** For the convolution layer,
>
> $$S^{m,i} = W^m \underbrace{\text{mat}\left(P_\phi^m P_{\text{pad}}^m \text{vec}\left(Z^{m,i}\right)_{h^m h^m d^m \times a^m_{\text{conv}} b^m_{\text{conv}}}\right)}_{\phi(\text{pad}(Z^{m,i}))} + \boldsymbol{b}^m \mathbb{1}^T_{a^m_{\text{conv}} b^m_{\text{conv}}}$$
>
> $$Z^{m+1,i} = \text{mat}\left(P_{\text{pool}}^{m,i} \text{vec}\left(\sigma(S^{m,i})\right)\right)_{d^{m+1} \times a^{m+1} b^{m+1}} \tag{3}$$

Notation (Kronecker Product). $\otimes$ is the Kronecker product. If

$$A \in \mathbb{R}^{m\times n}, \quad B \in \mathbb{R}^{p\times q}$$

then

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{R}^{mp\times nq}$$

Notation (Identity Matrix). $\mathcal{I}$ is an identity matrix. For example,

$$\mathcal{I}_{a^m_{\text{conv}}b^m_{\text{conv}}} \in \mathbb{R}^{a^m_{\text{conv}}b^m_{\text{conv}}\times a^m_{\text{conv}}b^m_{\text{conv}}}$$

is an identity matrix for eqs. (4) and (5).

Lemma 3.1.1.

$$\begin{aligned} \text{vec}\,(AB) &= (\mathcal{I}\otimes A)\text{vec}\,(B) && (6) \\ &= (B^T \otimes \mathcal{I})\text{vec}\,(A) && (7) \end{aligned}$$

Proof. First, denote the columns of B as $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$, i.e.

$$B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$$

$$AB = [A\mathbf{b}_1, A\mathbf{b}_2, \ldots, A\mathbf{b}_n]$$

and

$$\text{vec}\,(AB) = \begin{bmatrix} A\mathbf{b}_1 \\ A\mathbf{b}_2 \\ \vdots \\ A\mathbf{b}_n \end{bmatrix}$$

(1): Note that

$$(\mathcal{I}\otimes A)\text{vec}\,(B) = \begin{bmatrix} A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{bmatrix} = \begin{bmatrix} A\mathbf{b}_1 \\ A\mathbf{b}_2 \\ \vdots \\ A\mathbf{b}_n \end{bmatrix} = \text{vec}\,(AB)$$

(2): Note that

$$(B^T \otimes \mathcal{I})\text{vec}\,(A) = \begin{bmatrix} \mathbf{b}_1^T \otimes \mathcal{I} \\ \mathbf{b}_2^T \otimes \mathcal{I} \\ \vdots \\ \mathbf{b}_n^T \otimes \mathcal{I} \end{bmatrix} \text{vec}\,(A)$$

The i-th row is

$$(\mathbf{b}_i^T \otimes \mathcal{I})\text{vec}\,(A) = \text{vec}\,(A\mathbf{b}_i)$$

Thus we have

$$(B^T \otimes \mathcal{I})\text{vec}\,(A) = \begin{bmatrix} \text{vec}\,(A\mathbf{b}_1) \\ \text{vec}\,(A\mathbf{b}_2) \\ \vdots \\ \text{vec}\,(A\mathbf{b}_n) \end{bmatrix} = \text{vec}\,(AB)$$

■

By using the lemma eqs. (6), (7), we have

$$\begin{aligned} \text{vec}\left(S^{m,i}\right) &= \text{vec}\left(W^m \phi(\text{pad}\left(Z^{m,i}\right))\right) + \text{vec}\left(\boldsymbol{b}^m \mathbb{1}^T\right) \\ &= (\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes W^m)\text{vec}\left(\phi(\text{pad}\left(Z^{m,i}\right))\right) + (\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^m})\boldsymbol{b}^m & (4) \\ &= \left(\phi(\text{pad}\left(Z^{m,i}\right))^T \otimes \mathcal{I}_{d^{m+1}}\right) \text{vec}\,(W^m) + (\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^m})\boldsymbol{b}^m & (5) \end{aligned}$$

getting eqs. (5) we can simply differtiate $\text{vec}\left(S^{m,i}\right)$ w.r.t. $\text{vec}\,(W^m)$.

For the fully-connected layer, we have

$$\begin{aligned} \boldsymbol{s}^{m,i} &= W^m \boldsymbol{z}^{m,i} + \boldsymbol{b}^m \\ &= (\mathcal{I}_1 \otimes W^m)\boldsymbol{z}^{m,i} + (\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}})\boldsymbol{b}^m & (8) \\ &= \left((\boldsymbol{z}^{m,i})^T \otimes \mathcal{I}_{n_{m+1}}\right) \text{vec}\,(W^m) + (\mathbb{1}_1 \otimes \mathcal{I}_{n_{m+1}})\boldsymbol{b}^m & (9) \end{aligned}$$

where eqs. (8) and (9) are from eqs. (6) and (7) respectively.

> **Note.** $\text{vec}\left(\boldsymbol{z}^{m,i}\right) = \boldsymbol{z}^{m,i}, \quad \text{since } \boldsymbol{z}^{m,i} \in \mathbb{R}^{n_m \times 1}$

The eqs. (4) and (8) are in the same form. Further, if for fully-connected layer, we define

$$\phi(\text{pad}\left(Z^{m,i}\right)) := \mathcal{I}_{n_m} \boldsymbol{z}^{m,i}, \quad L^c < m \leq L + 1$$

then eqs. (5) and (9) are also in the same form. Thus we can derive the gradient of both convolution layer and fully-connected layer together.

3.2 Gradient Calculation

3.2.1 Gradient Calculation for Convolution Layer

> **As previously seen.** For convolution layer,
>
> $$\text{vec}\left(S^{m,i}\right) = \left(\phi(\text{pad}\left(Z^{m,i}\right))^T \otimes \mathcal{I}_{d^{m+1}}\right) \text{vec}\,(W^m) + (\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^m})\boldsymbol{b}^m$$

Let's first give the definition of the partial derivative w.r.t. matrix (or vector) variable.

> **Definition 3.2.1** (Partial Derivative w.r.t. Matrix/Vector). Given $\boldsymbol{y} \in \mathbb{R}^{|y| \times 1}$ and $\boldsymbol{x} \in \mathbb{R}^{|x| \times 1}$
>
> $$\frac{\partial \boldsymbol{y}}{\partial (\boldsymbol{x})^\top} := \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_{|x|}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{|y|}}{\partial x_1} & \cdots & \frac{\partial y_{|y|}}{\partial x_{|x|}} \end{pmatrix}$$

Theorem 3.2.1. Given $\boldsymbol{x}, \boldsymbol{y}$ as column vectors, and A as a matrix, if

$$\boldsymbol{y} = A\boldsymbol{x}$$

then

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^\top} = A$$

Proof. If $\boldsymbol{y} = A\boldsymbol{x}$, then

$$y_1 = A_{11}x_1 + A_{12}x_2 + \cdots + A_{1|x|}x_{|x|}$$

i.e.

$$y_i = \sum_{j=1}^{|x|} A_{ij}x_j$$

and

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^\top} = \begin{pmatrix} A_{11} & \cdots & A_{1|x|} \\ \vdots & \ddots & \vdots \\ A_{|y|1} & \cdots & A_{|y||x|} \end{pmatrix} = A$$

■

Then we can compute the derivatives

$$\underbrace{\frac{\partial \xi_i}{\partial \operatorname{vec}\left(W^m\right)^\top}}_{\text{a part of gradient}} \overset{\text{chain}}{=} \frac{\partial \xi_i}{\partial \operatorname{vec}\left(S^{m,i}\right)^\top} \frac{\partial \operatorname{vec}\left(S^{m,i}\right)}{\partial \operatorname{vec}\left(W^m\right)^\top} = \frac{\partial \xi_i}{\partial \operatorname{vec}\left(S^{m,i}\right)^\top} \left(\phi(\operatorname{pad}\left(Z^{m,i}\right))^\top \otimes \mathcal{I}_{d^{m+1}}\right)$$

$$\overset{(3)}{=} \operatorname{vec}\left(\frac{\partial \xi_i}{\partial S^{m,i}} \phi(\operatorname{pad}\left(Z^{m,i}\right))^\top\right)^\top \tag{1}$$

ξ_i is a scalar, so $\partial \xi_i / \partial \operatorname{vec}\left(S^{m,i}\right)^\top \in \mathbb{R}^{1\times(d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}})}$, which is a row vector. Then times a matrix build from definition 3.2.1, it is still a row vector. Then by the theorem above, $A = \left(\phi(\operatorname{pad}\left(Z^{m,i}\right))^T \otimes \mathcal{I}_{d^{m+1}}\right)$, $\boldsymbol{x} = \operatorname{vec}\left(W^m\right)$, and $\boldsymbol{y} = \operatorname{vec}\left(S^{m,i}\right)$, we get eq. (1).

Lemma 3.2.1. Given A, B as matrices,

$$\operatorname{vec}(AB) = \operatorname{vec}(B)^\top \left(\mathcal{I} \otimes A^\top\right) \tag{2}$$

$$= \operatorname{vec}(A)^\top \left(B^T \otimes \mathcal{I}\right) \tag{3}$$

This eqs. shows that we can first calculate $\partial \xi_i / \partial S^{m,i}$ to get the gradient part w.r.t. W^m. Similarly, for the bias term, we have

$$\frac{\partial \xi_i}{\partial (\boldsymbol{b}^m)^\top} \overset{\text{chain}}{=} \frac{\partial \xi_i}{\partial \operatorname{vec}\left(S^{m,i}\right)^\top} \frac{\partial \operatorname{vec}\left(S^{m,i}\right)}{\partial (\boldsymbol{b}^m)^\top} = \frac{\partial \xi_i}{\partial \operatorname{vec}\left(S^{m,i}\right)^\top} \left(\mathbb{1}_{a^m_{\text{conv}}b^m_{\text{conv}}} \otimes \mathcal{I}_{d^{m+1}}\right)$$

$$\overset{(3)}{=} \operatorname{vec}\left(\frac{\partial \xi_i}{\partial S^{m,i}} \mathbb{1}_{a^m_{\text{conv}}b^m_{\text{conv}}}\right)^\top \tag{4}$$

To calculate the eqs. (1), we need to compute $\phi(\operatorname{pad}\left(Z^{m,i}\right))$ which is already given in the forward pass. In (1) and (4), $\partial \xi_i / \partial S^{m,i}$ is also needed. Now we are going to show how to compute it backwardly. To do this, we assume that

$$\frac{\partial \xi_i}{\partial Z^{m+1,i}}$$

is available from the upper layer. Then we are going to compute

$$\frac{\partial \xi_i}{\partial S^{m,i}}, \quad \frac{\partial \xi_i}{\partial Z^{m,i}}$$

This process is called **backpropagation** process in neural networks training. We have the following workflow:

$$Z^{m,i} \longleftarrow \text{padding} \longleftarrow \text{convolution} \longleftarrow \sigma(S^{m,i}) \longleftarrow \text{pooling} \longleftarrow Z^{m+1,i} \tag{5}$$

From chain rule, we have

$$\frac{\partial \xi_i}{\partial (S^{m,i})^\top} = \frac{\partial \xi_i}{\partial (\sigma(S^{m,i}))^\top} \frac{\partial (\sigma(S^{m,i}))}{\partial (S^{m,i})^\top}$$

we get a column vector $\partial \xi_i / \partial (\sigma(S^{m,i}))^\top$ and a matrix $\partial (\sigma(S^{m,i})) / \partial (S^{m,i})^\top$. Now assume σ is a scalar function, then

$$\frac{\partial (\sigma(S^{m,i}))}{\partial (S^{m,i})^\top}$$

is a squared. diagonal matrix of $|\text{vec}\left(S^{m,i}\right)| \times |\text{vec}\left(S^{m,i}\right)|$ dimension, because $\sigma(S^{m,i})$ have same dimension as $S^{m,i}$ and from theorem 3.2.1 we can get a squared matrix. Further, we assume that σ is ReLU function, i.e.

$$\sigma'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

then we can define

$$I[S^{m,i}]_{(p,q)} := \begin{cases} 1, & S^{m,i}_{(p,q)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

and have

$$\frac{\partial \xi_i}{\partial (S^{m,i})^\top} = \frac{\partial \xi_i}{\partial (\sigma(S^{m,i}))^\top} \odot \text{vec}\left(I[S^{m,i}]\right)^T$$

> **Remark.** This process can be extended to other scalar (component-wise) activation functions easily. The general form is
>
> $$\frac{\partial \xi_i}{\partial (S^{m,i})^\top} = \frac{\partial \xi_i}{\partial (\sigma(S^{m,i}))^\top} \odot \text{vec}\left(\sigma'(S^{m,i})\right)^\top$$

Next, we do

$$\begin{aligned} \frac{\partial \xi_i}{\partial (S^{m,i})^\top} &\overset{\text{chain}}{=} \frac{\partial \xi_i}{\partial (Z^{m+1,i})^\top} \frac{\partial (Z^{m+1,i})}{\partial (\sigma(S^{m,i}))^\top} \frac{\partial (\sigma(S^{m,i}))}{\partial (S^{m,i})^\top} = \left(\frac{\partial \xi_i}{\partial (Z^{m+1,i})^\top} \frac{\partial (Z^{m+1,i})}{\partial (\sigma(S^{m,i}))^\top} \right) \odot \text{vec}\left(I[S^{m,i}]\right)^\top \\ &= \left(\frac{\partial \xi_i}{\partial (Z^{m+1,i})^\top} P^{m,i}_{\text{pool}} \right) \odot \text{vec}\left(I[S^{m,i}]\right)^\top \end{aligned} \tag{6}$$

Now we can get $\partial \xi_i / \partial S^{m,i}$ from $\partial \xi_i / \partial Z^{m+1,i}$ which is available from the upper layer and $P^{m,i}_{\text{pool}}$, $I[S^{m,i}]$ from the forward pass.

> **Note.** In the forward pass, we have
>
> $$Z^{m+1,i} = \text{mat}\left(P^{m,i}_{\text{pool}} \text{vec}\left(\sigma(S^{m,i})\right)\right)_{d^{m+1} \times a^{m+1} b^{m+1}}$$
>
> hence
>
> $$\frac{\partial (Z^{m+1,i})}{\partial (\sigma(S^{m,i}))^\top} = P^{m,i}_{\text{pool}}$$

> **Remark.** If general scalar activation function σ is used, then
>
> $$\frac{\partial \xi_i}{\partial (S^{m,i})^\top} = \left(\frac{\partial \xi_i}{\partial (Z^{m+1,i})^\top} P^{m,i}_{\text{pool}} \right) \odot \text{vec}\left(\sigma'(S^{m,i})\right)^\top$$

In the end, we have to compute $\partial \xi_i / \partial Z^{m,i}$ to pass to the lower layer.

> **As previously seen.**
>
> $$\mathrm{vec}\left(S^{m,i}\right) = (\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes W^m)\mathrm{vec}\left(\phi(\mathrm{pad}\left(Z^{m,i}\right))\right) + (\mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes \mathcal{I}_{d^m})\boldsymbol{b}^m$$

Similarly, we have

$$\begin{aligned}
\frac{\partial \xi_i}{\partial (Z^{m,i})^\top} &= \frac{\partial \xi_i}{\partial (S^{m,i})^\top} \frac{\partial (S^{m,i})}{\partial (\phi(\mathrm{pad}\left(Z^{m,i}\right)))^\top} \frac{\partial (\phi(\mathrm{pad}\left(Z^{m,i}\right)))}{\partial (\mathrm{pad}\left(Z^{m,i}\right))^\top} \frac{\partial (\mathrm{pad}\left(Z^{m,i}\right))}{\partial (Z^{m,i})^\top} \\
&= \frac{\partial \xi_i}{\partial (S^{m,i})^\top} \left(\mathcal{I}_{a^m_{\text{conv}} b^m_{\text{conv}}} \otimes W^{m^\top}\right) P^m_\phi P^m_{\text{pad}} \qquad (7) \\
&= \mathrm{vec}\left((W^m)^\top \frac{\partial \xi_i}{\partial S^{m,i}}\right)^\top P^m_\phi P^m_{\text{pad}} \qquad (8)
\end{aligned}$$

We have everything to compute the gradient of convolution layer.

> **Note.** In the forward pass, we have
>
> $$\phi(\mathrm{pad}\left(Z^{m,i}\right)) = P^m_\phi P^m_{\text{pad}} \mathrm{vec}\left(Z^{m,i}\right)$$
>
> hence
>
> $$\frac{\partial (\phi(\mathrm{pad}\left(Z^{m,i}\right)))}{\partial (\mathrm{pad}\left(Z^{m,i}\right))^\top} \frac{\partial (\mathrm{pad}\left(Z^{m,i}\right))}{\partial (Z^{m,i})^\top} = P^m_\phi P^m_{\text{pad}}$$

3.2.2 Gradient Calculation for Fully-connected Layer

> **As previously seen.**
>
> $$\frac{\partial \xi_i}{\partial \mathrm{vec}\left(W^m\right)^\top} = \mathrm{vec}\left(\frac{\partial \xi_i}{\partial S^{m,i}}\ \phi(\mathrm{pad}\left(Z^{m,i}\right))^\top\right)^\top$$
>
> and
>
> $$\frac{\partial \xi_i}{\partial (\boldsymbol{b}^m)^\top} = \mathrm{vec}\left(\frac{\partial \xi_i}{\partial S^{m,i}}\ \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}}\right)^\top$$

Considering the fully-connected layer as a special case of the convolutional layer, we get

$$\frac{\partial \xi_i}{\partial \mathrm{vec}\left(W^m\right)^\top} = \mathrm{vec}\left(\frac{\partial \xi_i}{\partial \boldsymbol{s}^{m,i}}\ (\boldsymbol{z}^{m,i})^\top\right)^\top \qquad (1)$$

$$\frac{\partial \xi_i}{\partial (\boldsymbol{b}^m)^\top} = \frac{\partial \xi_i}{\partial (\boldsymbol{s}^{m,i})^\top} \qquad (2)$$

Chapter 4

Implementation

Chapter 5

GPU Programming

Chapter 6

Automatic Differentiation

6.1 Basic Concepts

6.2 Implementation

Chapter 7

Large Language Models (LLM)

7.1 High-level Overview

7.2 Auto-regressive Models

7.3 Detailed Operations