

# **Mathematics, Machine Learning and Deep Learning Notes**

Jinming Su

Last update: September 30, 2019

# Contents

<b>1</b>	<b>Mathematical Foundation</b>	<b>1</b>
1.1	Advanced math	1
1.1.1	Taylor formula	1
1.2	Probability theory and mathematical statistics	1
1.2.1	How to get expected value and variance?	1
1.2.2	Discrete probability distribution	2
1.2.3	Continuous probability distribution	3
1.2.4	Sample mean and sample variance	4
<b>2</b>	<b>Machine Learning</b>	<b>5</b>
2.1	Linear regression	5
2.1.1	Unitary linear regression	5
2.2	Linear Regression Least mean square (LMS)	5
2.3	Logistic regression (LR)	6
2.3.1	Form	6
2.4	Naive Bayes	7
2.4.1	Prior and posterior	7
2.4.2	Naive Bayesian Classifier (NBC)	7
2.4.3	Parameter estimation of NBC by maximum likelihood estimation (MLE)	7
2.5	Regularization	8
2.5.1	L0	8
2.5.2	L1 (lasso regularization)	9
2.5.3	L2 (ridge regression or weight decay)	9
2.6	Perceptron	9
2.6.1	Why can't perceptron solve XOR problem?	9
2.6.2	Definition of perceptron	9
2.6.3	Learning Algorithm	9
2.6.4	Dual form of perceptron learning algorithm	10
2.7	Support vector machine (SVM)	11
2.7.1	Form	11
2.7.2	Lagrange duality	12
2.7.3	Solution of SVM	12
2.7.4	Kernel	13
2.8	How to get the update rule of parameters of backpropagation in gradient descent algorithm?	14
2.9	Gaussian mixture model (GMM)	14
2.9.1	Maximum likelihood estimation	14
2.9.2	GMM and EM	15
2.10	Principal components analysis (PCA)	15
2.10.1	Maximum variance theory	15
2.10.2	PCA	15
2.11	Hidden Markov model (HMM)	17
2.11.1	Definition	17
2.11.2	Three basic problems	18
2.12	Conditional random field (CRF)	18
2.12.1	Probabilistic undirected graphical model, or Markov random field	18

2.12.2	Conditional random field . . . . .	18
2.13	Generative model and discriminative model . . . . .	18
<b>3</b>	<b>Deep Network</b>	<b>19</b>
3.1	How does backpropagation work? . . . . .	19
3.2	Backpropagation of Convolutional Neural Network . . . . .	21
3.2.1	Backpropagation of Fully Connectional Layer . . . . .	21
3.2.2	Backpropagation of Convolutional Layer . . . . .	21
3.3	Why does batch normalization work? . . . . .	23
3.3.1	Dataset shift and covariate shift . . . . .	23
3.3.2	Internal covariate shift . . . . .	23
3.3.3	Batch normalization . . . . .	24
3.4	Why does residual learning work? . . . . .	26
3.5	Understanding Deconvolution & bilinear interpolation . . . . .	26
3.5.1	Why is it called Transposed Convolution? . . . . .	27
3.5.2	Bilinear Interpolation . . . . .	28
3.6	Activation Function . . . . .	29
3.6.1	Sigmoid . . . . .	29
3.6.2	ReLU . . . . .	29
3.7	Parameter Initialization . . . . .	30
3.7.1	Why not initialized with all zeros? . . . . .	30
3.7.2	Why not initialized with Standard Gaussian? . . . . .	30
3.7.3	xavier . . . . .	30
3.7.4	msra . . . . .	31
<b>4</b>	<b>Contents specifically referenced</b>	<b>33</b>

# Todo list

□	Note in Eq. 1.7, maybe $P(X = 1)$ is equivalent $P(X^2 = 1^2)$ , which ensures the establishment of this equation. . .	2
□	There exists another solution directly through derivation of the probability mass function of Binomial distribution. .	2
□	Note there exists Taylor Expansion (at $x = 0$ ) $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ . . . . .	3
□	The derivation of the expectation and invariance of normal distribution requires multiple integration operations. . .	3
□	The derivation of the expectation and invariance of exponential distribution requires multiple integration operations.	4
■	Todo: conjugate distribution and Beta distribution. . . . .	4
□	The is the opposite number of binary cross entropy loss function. . . . .	6
□	It can be proved that the cross entropy function is a convex function. A detailed proof can be found in the blog <a href="https://blog.csdn.net/RHONYN/article/details/80342126">https://blog.csdn.net/RHONYN/article/details/80342126</a> . About the proof of convex function, the first-order function relies on the non-negativity of the second derivative and the higher-order function relies on the semi-positive characterization of Hessian Matrix. Therefore, we can get the global minimum value of cross entropy. . . . .	7
□	The convergence of this algorithm about perceptron learning can be proved, but it is not within the scope of this note at present. . . . .	10
■	Why? . . . . .	13
□	Sequential minimal optimization algorithm is used to efficiently solve the convex quadratic programming problem (as 2.56). . . . .	13
■	The derivation of EM algorithm . . . . .	15
□	forward-backward algorithm . . . . .	18
□	Baum-Welch algorithm . . . . .	18
□	Viterbi algorithm . . . . .	18
■	CRF algorithm . . . . .	18
■	Todo: Random forest . . . . .	18
■	A more precise definition of <b>covariate</b> can't be found. . . . .	23
■	PCA whitening and ZCA Whitening. . . . .	23
■	Why is multi-layer linear network equivalent to the representation ability of a single layer linear network. . . . .	24
□	There exists $\sum_i$ for $y_i$ because gradient from each $y_i$ will backpropagate for $\gamma$ and the effect of summation is achieved.	25
■	checkerboard artifacts, gridding issues . . . . .	27
□	We can know that Xavier and Msra can both belong to uniform distribution or gaussian distribution. . . . .	32

# Chapter 1

## Mathematical Foundation

### 1.1 Advanced math

#### 1.1.1 Taylor formula

General term formula:

$$f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x), \quad (1.1)$$

which is the Taylor expansion of  $f(x)$  at  $x_0$  and  $R_n(x)$  means the remainder of Taylor formula.

There are some common Taylor formulas:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + R_n = \sum_{i=0}^{\infty} \frac{x^i}{i!}. \quad (1.2)$$

### 1.2 Probability theory and mathematical statistics

**Probability theory** mainly focuses on the probability of occurrence of a single event, while **mathematical statistics** is more inclined to statistics. It focuses on the sampling probability of a group and the possible interval of occurrence of this probability.

In the following introduction, these two concepts are introduced without distinction.

#### 1.2.1 How to get expected value and variance?

$X$  is a random variable whose values are  $X_1, X_2, \dots, X_n$ .  $P(X_1), P(X_2), \dots, P(X_n)$  are the probability corresponding to these values. The expected value of  $X$  can be denoted as  $E(X)$ , and the variance is denoted as  $Var(X)$ . Then,

$$\begin{aligned} E(X) &= \sum_{i=1}^n X_i P(X_i) \text{ for discrete variable} \\ &= \int_X x f(x) dx \text{ for continuous variable} \end{aligned} \quad (1.3)$$

and

$$\begin{aligned} Var(X) &= \sum_{i=1}^n (X_i - E(X))^2 P(X_i) \text{ for discrete variable} \\ &= \int_X (x - E(X))^2 f(x) dx \text{ for continuous variable} \end{aligned} \quad (1.4)$$

For discrete variable,

$$\begin{aligned}
 Var(X) &= \sum_{i=1}^n (X_i - E(X))^2 P(X_i) \\
 &= E[(X - E(X))^2] \\
 &= E[X^2 + E(X)^2 - 2XE(X)] \\
 &= E(X^2) + E(X)^2 - 2E(X)E(X) \\
 &= E(X^2) - E(X)^2
 \end{aligned} \tag{1.5}$$

### 1.2.2 Discrete probability distribution

**Bernoulli distribution** is the discrete probability distribution of a random variable which takes the value 1 with probability  $p$  and the value 0 with probability  $q = 1 - p$ . We denote Bernoulli distribution as  $B(1, p)$ . Mathematically, if  $X$  is a random variable with  $B(1, p)$ , then  $P(X = 1) = p$ ,  $P(X = 0) = q = 1 - p$ . The probability mass function  $f$  of this distribution over possible outcomes  $k$ , is

$$f(k; p) = \begin{cases} p & \text{if } k = 1, \\ q = 1 - p & \text{if } k = 0. \end{cases} \tag{1.6}$$

The expected value and invariance of a Bernoulli variable  $X$  are

$$\begin{cases} E(X) = P(X = 1) \cdot 1 + P(X = 0) \cdot 0 = p, \\ E(X^2) = P(X = 1) \cdot 1^2 + P(X = 0) \cdot 0^2 = P(X = 1) = p, \\ Var(X) = E(X^2) - E(X)^2 = p - p^2 = p(1 - p) = pq. \end{cases} \tag{1.7}$$

Note in Eq. 1.7, maybe  $P(X = 1)$  is equivalent  $P(X^2 = 1^2)$ , which ensures the establishment of this equation.

**Binomial distribution** with parameters  $n$  and  $p$  is the discrete probability distribution of the number of successes in a sequence of  $n$  independent experiments. Each experiment is a Bernoulli trial. In general, if the random variable  $X$  follows the binomial distribution with parameters  $n \in \mathbb{N}$  and  $p \in [0, 1]$ , we write  $X \sim B(n, p)$ . The probability of getting exactly  $k$  successes in  $n$  trials is given by the probability mass function:

$$f(k; n, p) = P(k; n, p) = P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}. \tag{1.8}$$

The expected value and invariance of a Binomial variable  $X$  (converted into Bernoulli distribution for derivation, where  $X_i$  is independent of each other) are

$$\begin{cases} E(X) = E(X_1 + X_2 + \cdots + X_n) \\ \quad = E(X_1) + E(X_2) + \cdots + E(X_n) \\ \quad = p + p + \cdots + p \\ \quad = np, \\ Var(X) = Var(X_1) + Var(X_2) + \cdots + Var(X_n) \\ \quad = nVar(X_1) \\ \quad = np(1 - p). \end{cases} \tag{1.9}$$

There exists another solution directly through derivation of the probability mass function of Binomial distribution.

**Poisson distribution** is a discrete probability distribution that expresses the probability of a given number  $k$  of events occurring in a fixed interval of time or space if these events occur with a known constant rate  $\lambda$  and independently of the time since the last events. If  $X$  is a Poisson variable with the average number of events  $\lambda$ , we write  $X \sim Poisson(\lambda)$ . The probability mass function is

$$f(k; n, \lambda) = P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}. \tag{1.10}$$

The expected value and invariance of a Poisson variable  $X$  are

$$\left\{ \begin{array}{l} E(X) = \sum_{i=0}^{\infty} i P(X = i) \\ = \sum_{i=1}^{\infty} i \frac{e^{-\lambda} \lambda^i}{i!} = \lambda e^{-\lambda} \sum_{i=1}^{\infty} \frac{\lambda^{i-1}}{(i-1)!} = \lambda e^{-\lambda} \sum_{i=0}^{\infty} \frac{\lambda^i}{i!} \\ = \lambda e^{-\lambda} e^{\lambda} \\ = \lambda \\ E(X^2) = \lambda + \lambda^2 \\ Var(X) = \lambda + \lambda^2 - \lambda^2 \\ = \lambda \end{array} \right. \quad (1.11)$$

Note there exists Taylor Expansion (at  $x = 0$ )  $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \cdots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ .

### 1.2.3 Continuous probability distribution

For **continuous uniform distribution**, all intervals of the same length on the distribution's support are equally probable. The support is defined by the two parameters,  $a$  and  $b$ , which are its minimum and maximum values. The distribution is often abbreviated  $U(a, b)$ . The probability density function of the continuous uniform distribution is

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b. \end{cases} \quad (1.12)$$

The expected value and invariance of a Uniform variable  $X$  are

$$\left\{ \begin{array}{l} E(X) = \int_a^b x \frac{1}{b-a} \\ = \frac{x^2}{2(b-a)} \Big|_a^b = \frac{b^2 - a^2}{2(b-a)} = \frac{(b+a)(b-a)}{2(b-a)} \\ = \frac{b+a}{2} \\ E(X^2) = \int_a^b x^2 \frac{1}{b-a} \\ = \frac{x^3}{3(b-a)} \Big|_a^b = \frac{b^3 - a^3}{3(b-a)} = \frac{(b-a)(b^2 + a^2 + ab)}{3(b-a)} \\ = \frac{a^2 + b^2 + ab}{3} \\ Var(X) = E(X^2) - E(X)^2 = \frac{a^2 + b^2 + ab}{3} - \left(\frac{b+a}{2}\right)^2 \\ = \frac{4a^2 + 4b^2 + 4ab}{12} - \frac{3a^2 + 3b^2 + 6ab}{12} \\ = \frac{(a-b)^2}{12} \end{array} \right. \quad (1.13)$$

The probability density function of the **normal distribution** is

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (1.14)$$

where  $\mu$  is the expectation and  $\sigma$  is the standard deviation. If a random variable  $X$  is distributed normally with mean  $\mu$  and variance  $\sigma^2$ , one may write  $X \sim N(\mu, \sigma^2)$ .

The derivation of the expectation and invariance of normal distribution requires multiple integration operations.

**Exponential distribution** is the probability distribution that describes the time between events in a Poisson point process, i.e. a process in which event occur continuously and independently at a constant average rate. The distribution is often abbreviated  $Exponential(\lambda)$ . The probability density function of an exponential distribution is

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \quad (1.15)$$

The expected value and invariance of a Exponential variable  $X$  are

$$\begin{cases} E(X) = \frac{1}{\lambda}, \\ Var(X) = \frac{1}{\lambda^2}. \end{cases} \quad (1.16)$$

The derivation of the expectation and invariance of exponential distribution requires multiple integration operations.

Todo: conjugate distribution and Beta distribution.

### 1.2.4 Sample mean and sample variance

**Sample mean** is defined as

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i. \quad (1.17)$$

**Sample Variance** is defined as

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2. \quad (1.18)$$

The above definitions follow the theorem: sample mean is unbiased estimation of population mean and sample variance is unbiased estimation of population variance.

**proof:**

$$\begin{aligned} E[\bar{X}] &= E\left[\frac{1}{n} \sum_{i=1}^n X_i\right] \\ &= \frac{1}{n} \sum_{i=1}^n E[X_i] \\ &= \frac{1}{n} n E(X) \\ &= E(X) \\ E[S^2] &= E\left[\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2\right] \\ &= \frac{1}{n-1} E\left[\sum_{i=1}^n (X_i^2 + \bar{X}^2 - 2X_i\bar{X})\right] \\ &= \frac{1}{n-1} (E[\sum_{i=1}^n X_i^2] + n\bar{X}^2 - 2n\bar{X}^2) \\ &= \frac{1}{n-1} (nE[X^2] - nE[\bar{X}^2]) \\ &= \frac{n}{n-1} (Var[X] + E[X]^2 - Var[\bar{X}] - E[\bar{X}]^2) \\ &= \frac{n}{n-1} (Var[X] - Var[\bar{X}]) \\ &= \frac{n}{n-1} (Var[X] - \frac{1}{n} Var[X]) \\ &= Var[X] \end{aligned} \quad (1.19)$$



## Chapter 2

# Machine Learning

### 2.1 Linear regression

#### 2.1.1 Unitary linear regression

The problem can be denoted as:

$$\begin{aligned} f_w(x) &= wx + b, \\ L(w) &= \sum_{i=1}^n (f_w(x^{(i)}) - y^{(i)})^2 = \sum_{i=1}^n (wx^{(i)} + b - y^{(i)})^2 \\ \frac{\partial L(w)}{\partial w} &= 2 \sum_{i=1}^n (wx^{(i)} + b - y^{(i)})x^{(i)}, \\ \frac{\partial L(w)}{\partial b} &= 2 \sum_{i=1}^n (wx^{(i)} + b - y^{(i)}). \end{aligned} \tag{2.1}$$

The Unitary linear regression has closed-form resolution. Let the partial derivative be 0, we have

$$\begin{aligned} w &= \frac{\sum_{i=1}^n (y^{(i)} - \bar{y})x^{(i)}}{\sum_{i=1}^n (x^{(i)} - \bar{x})x^{(i)}}, \\ b &= \bar{y} - w\bar{x}. \end{aligned} \tag{2.2}$$

### 2.2 Linear Gression Least mean square (LMS)

We define the prediction of formula about  $x$ :

$$f_{\vec{w}}(\vec{x}) = \vec{w}^\top \vec{x} = \sum_{i=1}^k w_i x_i. \tag{2.3}$$

We define the loss function:

$$L(\vec{w}) = \frac{1}{2} \sum_{i=1}^n (f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)})^2. \tag{2.4}$$

We can solve above problem by gradient descent algorithm (GD). For gradient descent, we usually use one example for one updation. We use only one training example  $(\vec{x}^{(i)}, y^{(i)})$  for simplicity.

$$\begin{aligned} \frac{\partial L(\vec{w})}{\partial w_j} &= \frac{1}{2} \cdot 2(f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}) \cdot \frac{\partial \sum_{i=1}^k w_i x_i}{\partial w_j} \\ &= (f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}) * x_j^{(i)}, \\ w_j &= w_j + \alpha (f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}) * x_j^{(i)}. \end{aligned} \tag{2.5}$$

Next, we can also solve the problem by the matrix form.

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_k^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_k^{(n)} \end{bmatrix}, Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \quad (2.6)$$

$$L_{\vec{w}}(\vec{w}) = \frac{1}{2} (X\vec{w} - \vec{Y})^\top (X\vec{w} - \vec{Y})$$

$$= X^\top X\vec{w} - X^\top \vec{Y},$$

For above derivation, we need to know

$$\nabla_A \text{tr} AB = B^\top, \quad (2.7)$$

$$\nabla_A \text{tr} ABA^\top C = CAB + C^\top AB^\top.$$

Next, we have

$$\vec{w} = (X^\top X)^{-1} X^\top \vec{Y} \quad (2.8)$$

## 2.3 Logistic regression (LR)

### 2.3.1 Form

$$h_\theta(x) = g(\theta^\top x) = \frac{1}{1 + e^{-\theta^\top x}}. \quad (2.9)$$

The form of sigmoid function is

$$g(z) = \frac{1}{1 + e^{-z}}. \quad (2.10)$$

The derivation of sigmoid function is

$$g'(z) = g(z)(1 - g(z)). \quad (2.11)$$

Let us assume that

$$P(y = 1|x; \theta) = h_\theta(x), \quad (2.12)$$

$$P(y = 0|x; \theta) = 1 - h_\theta(x).$$

Above equation can be written more compactly as

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}. \quad (2.13)$$

Assuming that the  $m$  training examples are generated independently, we can then obtain the likelihood of the parameters  $\theta$  as

$$L(\theta) = p(\vec{y}|X; \theta)$$

$$= \prod_{i=1}^m p(y^{(i)}|x^{(i)}; \theta) \quad (2.14)$$

$$= \prod_{i=1}^m (h_\theta(x))^{y^{(i)}} (1 - h_\theta(x))^{1-y^{(i)}}.$$

Usually, it will be easier to maximize the log likelihood:

$$l(\theta) = \log L(\theta)$$

$$= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})). \quad (2.15)$$

The is the opposite number of binary cross entropy loss function.

It can be proved that the cross entropy function is a convex function. A detailed proof can be found in the blog <https://blog.csdn.net/RHONYN/article/details/80342126>. About the proof of convex function, the first-order function relies on the non-negativity of the second derivative and the higher-order function relies on the semi-positive characterization of Hessian Matrix. Therefore, we can get the global minimum value of cross entropy.

Lets start by working with just one training example  $(x, y)$ , and take derivatives to derive the stochastic gradient ascent rule:

$$\frac{\partial l(\theta)}{\partial \theta_j} = (y^{(i)} - h_{\theta}(x^{(i)}))x_j^{(i)}. \quad (2.16)$$

Above, we get the stochastic gradient ascent rule:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_{\theta}(x^{(i)}))x_j^{(i)} \quad (2.17)$$

## 2.4 Naive Bayes

### 2.4.1 Prior and posterior

The prior probability is the probability of a cause inferred from experience, denoted as  $P(\theta)$ . The posterior probability is the probability of the cause estimated from the result, denoted as  $P(\theta|x)$ . The posterior probability is defined as

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \quad (2.18)$$

where  $P(x|\theta)$  represents likelihood of  $x$ . In fact, likelihood is the function of parameters. Likelihood is equal to the probability of the result occurring caused by a cause (paramters) based on the cause.

Likelihood is from the viewpoint of the frequency. In the frequency, the parameter is a true value, not a random variable, so the parameter has no distribution and no probability.

### 2.4.2 Naive Bayesian Classifier (NBC)

the naive bayesian classifier can be expressed as

$$\begin{aligned} y = f(x) &= \arg \max_{c_k} P(Y = c_k | X = x) \\ &= \arg \max_{c_k} \frac{P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k)}{\sum_k P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k)}, \end{aligned} \quad (2.19)$$

where  $x^{(j)}$  means the  $j$ th features in the feature vector  $x$ .

Since the denominator is the same any  $c_k$ , the above equation 2.19 can be simplified as

$$y = \arg \max_{c_k} P(Y = c_k) \prod_j P(X^{(j)} = x^{(j)} | Y = c_k). \quad (2.20)$$

### 2.4.3 Parameter estimation of NBC by maximum likelihood estimation (MLE)

The parameter in NBC that needed to estimate is the prior  $P(Y = c_k)$  and likelihood  $P(X^{(j)} = x^{(j)} | Y = c_k)$ . MLEs of prior and likelihood are

$$\begin{aligned} P(Y = c_k) &= \frac{\sum_{i=1}^N I(y_i = c_k)}{N} \\ P(X^{(j)} = a^{(j)} | Y = c_k) &= \frac{\sum_{i=1}^N I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^N I(y_i = c_k)}, \end{aligned} \quad (2.21)$$

where  $x_i^{(j)}$  is the  $j$ th feature in  $i$ th sample vector, and  $a_{jl}$  means the  $l$ th value in the set of values of  $j$ th feature.

The proof is as follows. Define  $\theta_k = P(Y = c_k)$ , then we can get the joint probability distribution.

$$P(y_1, y_2, \dots, y_n; \theta_k) = \prod_{i=1}^N P(y_i) = \prod_{i=1}^N \theta_k^{N_k}, \quad (2.22)$$

then the likelihood function is

$$L(\theta^k; y_1, y_2, \dots, y_n) = P(y_1, y_2, \dots, y_n; \theta_k). \quad (2.23)$$

Next, the logarithmic operation is applied.

$$\begin{aligned} \ln(L(\theta_k)) &= \sum_{i=1}^N N_k \ln \theta_k, \\ \text{s.t., } \sum_{k=1}^K \theta_k &= 1 \end{aligned} \quad (2.24)$$

Using Lagrange Multiplier Method, we get

$$\ln(L(\theta_k, \lambda)) = \sum_{i=1}^N N_k \ln \theta_k + \lambda \left( \sum_{k=1}^K \theta_k - 1 \right). \quad (2.25)$$

Let the derivative of the above equation be zero, and combine all the equations with  $\theta$ . We get

$$P(Y = c_k) = \frac{N_k}{N} = \frac{\sum_{i=1}^N I(y_i = c_k)}{N}. \quad (2.26)$$

Moreover, the derivation of likelihood is similar.

## 2.5 Regularization

The key purpose of machine learning is to minimize errors of problems while regularizing parameters. Minimizing errors is to let the model fit the training data, while regularizing parameters is to prevent the model from fitting the training data too much.

Regularization is equivalent to introducing a prior (Laplace or Gaussian). In general, supervised learning tasks can be regarded as:

$$w^* = \arg \min_w \sum_i L(y_i, f(x_i; w)) + \lambda \Omega(w) \quad (2.27)$$

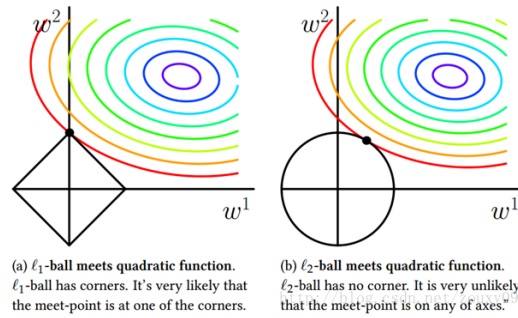


Figure 2.1:  $L_1$  norm and  $L_2$  norm.

### 2.5.1 L0

$$L_0 = \sum_i I(w_i \neq 0). \quad (2.28)$$

$L_0$  norm will make the weights sparse.

### 2.5.2 L1 (lasso regularization)

$$L_1 = \sum_i |w_i|. \quad (2.29)$$

$L_1$  norm will make the weights sparse. The main function of  $L_1$  is feature selection and interpretability.

#### Why does we usually use L1 to make the weights sparse instead of L0?

A theoretical explanation is that  $L_1$  is the optical convex approximation of  $L_0$  while  $L_0$  is difficult to solve (NP hard problem).

### 2.5.3 L2 (ridge regression or weight decay)

$$L_2 = \sum_i w_i^2 \quad (2.30)$$

$L_2$  norm can prevent the model from over-fitting. The main reason is the  $L_2$  constrain the model space.

## 2.6 Perceptron

### 2.6.1 Why can't perceptron solve XOR problem?

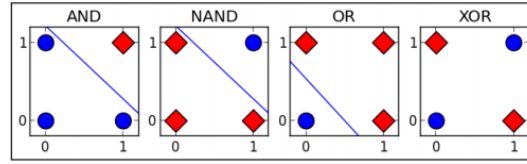


Figure 2.2: XOR problem.

Linear classification models can't classify linear non-separable problems. Perceptron is a linear classification model, and XOR problem is a linear non-separable problem, so perceptron can't solve the XOR problem.

### 2.6.2 Definition of perceptron

Suppose the input space is  $\mathcal{X} \subseteq \mathbb{R}^n$ , and the output space is  $\mathcal{Y} = \{+1, -1\}$ . For an example  $x \in \mathcal{X}$  where  $x$  is an  $n$ -dimensional vector  $(x_1, x_2, \dots, x_n)^T$ ,  $y \in \mathcal{Y}$  represents the category of  $x$ . Then, we get the perceptron model from the input space to output space:

$$f(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b), \quad (2.31)$$

where  $\vec{w}$  is weight and  $b$  is bias. And sign is a sign function, *i.e.*

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \quad (2.32)$$

There exists a geometric interpretation that a hyperplane whose normal vector is  $\vec{w}$  and intercept is  $b$ .

### 2.6.3 Learning Algorithm

Given a linear separable dataset  $T = (x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$ , where  $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$ , and  $y_i \in \mathcal{Y} = \{+1, -1\}$ ,  $i = 1, 2, \dots, N$ , we can construct a perspectron model to classify this dataset.

(1) construct the loss function. we define the loss function as the total distance from misclassified points to hyperplane. Toward this end, the distance from any point  $x_0$  to hyperplane:

$$\frac{1}{\|\vec{w}\|} |\vec{w}^T \cdot \vec{x}_0 + b| \quad (2.33)$$

where  $\|\vec{w}\|$  means the  $L_2$  norm of  $\vec{w}$ .

As for misclassified points,  $\vec{w}^T \cdot \vec{x}_0 + b > 0$  when  $y_i = -1$ , and  $\vec{w}^T \cdot \vec{x}_0 + b < 0$  when  $y_i = +1$ . So the total distance from all the misclassified points to hyperplane is

$$-\frac{1}{\|\vec{w}\|} \sum_{\vec{x}_i \in M} y_i(\vec{w}^T \cdot \vec{x}_i + b), \quad (2.34)$$

where  $M$  is the set of misclassified points.

In Eq. 2.36,  $\frac{1}{\|\vec{w}\|}$  can be ignored. The reason is (a)  $\frac{1}{\|\vec{w}\|}$  doesn't affect positive or negative judgment of  $y_i(\vec{w}^T \cdot \vec{x}_i + b)$  and (b)  $\frac{1}{\|\vec{w}\|}$  doesn't affect the final optimization result of Eq. 2.36. The final optimization result is there are no points with wrong classification, which leads to the loss of 0. Toward this end, the loss function of perceptron is

$$L(\vec{w}, b) = - \sum_{\vec{x}_i \in M} y_i(\vec{w}^T \cdot \vec{x}_i + b), \quad (2.35)$$

and the optimization object is

$$\min_{\vec{w}, b} L(\vec{w}, b) = \min_{\vec{w}, b} - \sum_{\vec{x}_i \in M} y_i(\vec{w}^T \cdot \vec{x}_i + b). \quad (2.36)$$

(2) We adopt stochastic gradient descent algorithm to train the perceptron model. Suppose the set of misclassified points is fixed, thus the gradient of loss function  $L(\vec{w}, b)$ :

$$\begin{aligned} \nabla_{\vec{w}} L(\vec{w}, b) &= - \sum_{\vec{x}_i \in M} y_i \vec{x}_i \\ \nabla_b L(\vec{w}, b) &= - \sum_{\vec{x}_i \in M} y_i \end{aligned} \quad (2.37)$$

So, the strategies of  $\vec{w}$  and  $b$  based on one misclassified point are:

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x}_i \\ b &\leftarrow b + \eta y_i \end{aligned} \quad (2.38)$$

where  $\eta$  is the learning rate.

(3) The standard form of perceptron learning algorithm:

input: linear separable training set:  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , where  $\vec{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$  and  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ . In addition, the learning rate is denoted as  $\eta$  ( $0 < \eta \leq 1$ ).

output:  $\vec{w}, b$ . And the perceptron model  $f(\vec{x}) = \text{sign}(\vec{w}^T \vec{x} + b)$ .

(a) choose initial values,  $\vec{w}_0, b_0$ , and usually  $\vec{w}_0 = 0, b_0 = 0$ ;

(b) choose one example  $(\vec{x}_i, y_i)$  from the training set;

(c) if  $y_i(\vec{w}^T \cdot \vec{x}_i + b) \leq 0$ :

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x}_i \\ b &\leftarrow b + \eta y_i \end{aligned} \quad (2.39)$$

(d) go to (b) until there are no misclassified points in training set.

(4)

The convergence of this algorithm about perceptron learning can be proved, but it is not within the scope of this note at present.

### 2.6.4 Dual form of perceptron learning algorithm

From Eq. 2.38,  $\vec{w}$  and  $b$  are updated for many times. Suppose the number of updates about each example is  $n_i$  ( $i = 1, 2, \dots, N$ ), and we define  $\alpha_i = n_i \eta$ . Then, the final learned  $\vec{w}$  and  $b$  can be expressed as

$$\begin{aligned} \vec{w} &= \sum_{i=1}^N \alpha_i y_i \vec{x}_i \\ b &= \sum_{i=1}^N \alpha_i y_i \end{aligned} \quad (2.40)$$

Then, we can obtain the dual form of perceptron learning algorithm.

input: linear seperable training set:  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , where  $\vec{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$  and  $y_i \in \mathcal{Y} = \{-1, +1\}$ ,  $i = 1, 2, \dots, N$ . In addition, the learning rate is denoted as  $\eta$  ( $0 < \eta \leq 1$ ).

output:  $\vec{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)^T, b$ . And the perceptron model  $f(\vec{x}) = \text{sign}(\sum_{j=1}^N \alpha_j y_j \vec{x}_j^T \vec{x} + b)$ .

(a) choose initial values,  $\vec{\alpha}, b_0$ , and usually  $\vec{\alpha} = 0, b_0 = 0$ ;

(b) choose one example  $(\vec{x}_i, y_i)$  from the training set;

(c) if  $y_i(\sum_{j=1}^N \alpha_j y_j \vec{x}_j^T \vec{x} + b) \leq 0$ :

$$\begin{aligned}\vec{\alpha}_i &\leftarrow \alpha_i + \eta; \\ b &\leftarrow b + \eta y_i;\end{aligned}\tag{2.41}$$

(d) go to (b) until there are no misclassified points in training set.

## 2.7 Support vector machine (SVM)

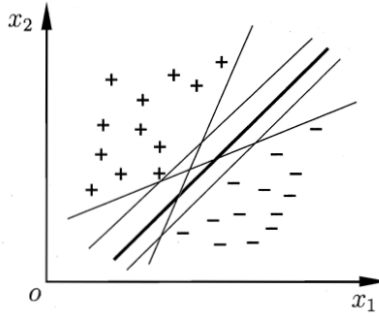


Figure 2.3: Classification hyperplane.

### 2.7.1 Form

For the classification problem, the mathematical form of hyperplane is defined as

$$\vec{w}^T \vec{x} + b = 0.\tag{2.42}$$

The distance from any point in the space to the hyperplane is

$$\gamma = \frac{|\vec{w}^T \vec{x} + b|}{|\vec{w}|}.\tag{2.43}$$

We know that the class of each points is  $y^{(i)}$ , where  $y^{(i)} = \pm 1$  means different classes. Then, the distance can be in another way by

$$\gamma^{(i)} = \frac{y^{(i)}(\vec{w}^T \vec{x}^{(i)} + b)}{|\vec{w}|},\tag{2.44}$$

which is named geometric margin. Next, we have

$$\gamma = \min_{i=1, \dots, m} \gamma^{(i)}.\tag{2.45}$$

Then, we give the form of SVM as

$$\begin{aligned}\max_{\gamma, \vec{w}, b} \quad & \gamma, \\ \text{s.t.} \quad & \frac{y^{(i)}(\vec{w}^T \vec{x}^{(i)} + b)}{|\vec{w}|} \geq \gamma\end{aligned}\tag{2.46}$$

Define functional margin as follows.

$$\begin{aligned}\hat{\gamma}^{(i)} &= y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b), \\ \hat{\gamma} &= \min_{i=1, \dots, m} \hat{\gamma}^{(i)}.\end{aligned}\tag{2.47}$$

Then, the form of SVM 2.46 can be transformed.

$$\begin{aligned}\max_{\gamma, \vec{w}, b} & \frac{\hat{\gamma}}{|\vec{w}|}, \\ \text{s.t.} & y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b) \geq \hat{\gamma}.\end{aligned}\tag{2.48}$$

We set the minimum functional margin is  $\hat{\gamma} = 1$ , then

$$\begin{aligned}\max_{\gamma, \vec{w}, b} & \frac{1}{|\vec{w}|}, \\ \text{s.t.} & y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b) \geq 1.\end{aligned}\tag{2.49}$$

Moreover, the form of SVM 2.49 can be transformed.

$$\begin{aligned}\min_{\gamma, \vec{w}, b} & \frac{1}{2}|\vec{w}|^2, \\ \text{s.t.} & y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b) \geq 1.\end{aligned}\tag{2.50}$$

### 2.7.2 Lagrange duality

Consider a constrained optimization problems:

$$\begin{aligned}\min_{\vec{w}} & f(\vec{w}), \\ \text{s.t.} & g_i(\vec{w}) \leq 0, i = 1, \dots, k, \\ & h_i(\vec{w}) = 0, i = 1, \dots, l.\end{aligned}\tag{2.51}$$

We can define generalized lagrangian

$$\mathcal{L}(\vec{w}, \vec{\alpha}, \vec{\beta}) = f(\vec{w}) + \sum_{i=1}^k \alpha_i g_i(\vec{w}) + \sum_{i=1}^l \beta_i h_i(\vec{w}).\tag{2.52}$$

If the primal constraints are indeed satisfied for  $\vec{w}$ , then  $\max_{\vec{\alpha}, \vec{\beta}: \alpha_i \geq 0} \mathcal{L}(\vec{w}, \vec{\alpha}, \vec{\beta}) = f(\vec{w})$ .

For primal ( $p$ ) and dual ( $d$ ) optimization problem, we have

$$d^* = \max_{\vec{\alpha}, \vec{\beta}: \alpha_i \geq 0} \min_{\vec{w}} \mathcal{L}(\vec{w}, \vec{\alpha}, \vec{\beta}) \leq \min_{\vec{w}} \max_{\vec{\alpha}, \vec{\beta}: \alpha_i \geq 0} \mathcal{L}(\vec{w}, \vec{\alpha}, \vec{\beta}) = p^*.\tag{2.53}$$

Therefore, the solution of primal problem can be transformed to solve the dual problem.

### 2.7.3 Solution of SVM

As shown in 2.54, the form of SVM is

$$\begin{aligned}\min_{\gamma, \vec{w}, b} & \frac{1}{2}|\vec{w}|^2, \\ \text{s.t.} & y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b) \geq 1.\end{aligned}\tag{2.54}$$

When we construct the Lagrangian for our optimization problem we have:

$$\mathcal{L}(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}|\vec{w}|^2 + \sum_{i=1}^m \alpha_i [1 - y^{(i)}(\vec{w}^\top \vec{x}^{(i)} + b)].\tag{2.55}$$



**Why?**

Lets find the dual form of the problem. To do so, we need to first minimize  $\mathcal{L}(\vec{w}, b, \vec{\alpha})$  with respect to  $\vec{w}$  and  $b$  (for fixed  $\alpha$ ) by setting the derivatives of  $\mathcal{L}$  with respect to  $\vec{w}$  and  $b$  to zero. We have:

$$\begin{aligned}\frac{\partial \mathcal{L}(\vec{w}, b, \vec{\alpha})}{\partial \vec{w}} &= \vec{w} - \sum_{i=1}^m \alpha_i y^{(i)} \vec{x}^{(i)} = 0, \vec{w} = \sum_{i=1}^m \alpha_i y^{(i)} \vec{x}^{(i)} \\ \frac{\partial \mathcal{L}(\vec{w}, b, \vec{\alpha})}{\partial b} &= -\sum_{i=1}^m \alpha_i y^{(i)} = 0.\end{aligned}\tag{2.56}$$

Plug the  $\vec{w}$  and  $b$  back into the Lagrangian Eq. 2.55, and simplify, we get

$$\begin{aligned}\mathcal{L}(\vec{w}, b, \vec{\alpha}) &= \frac{1}{2} \vec{w}^\top \vec{w} + \sum_{i=1}^m \alpha_i [1 - y^{(i)} (\vec{w}^\top \vec{x}^{(i)} + b)] \\ &= \frac{1}{2} \sum_{i=1}^m \alpha_i y^{(i)} (\vec{x}^{(i)})^\top \sum_{j=1}^m \alpha_j y^{(j)} \vec{x}^{(j)} + \sum_{i=1}^m \alpha_i - \sum_{i=1}^m \alpha_i y^{(i)} \vec{w}^\top \vec{x}^{(i)} - b \sum_{i=1}^m \alpha_i y^{(i)} \\ &= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (\vec{x}^{(i)})^\top \vec{x}^{(j)}.\end{aligned}\tag{2.57}$$

Then, we obtain the following dual optimization problem:

$$\begin{aligned}\max_{\alpha} \quad & \mathcal{L}(\vec{w}, b, \vec{\alpha}) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (\vec{x}^{(i)})^\top \vec{x}^{(j)} \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \dots, m, \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0\end{aligned}\tag{2.58}$$

Sequential minimal optimization algorithm is used to efficiently solve the convex quadratic programming problem (as 2.56).

**2.7.4 Kernel**

Gaussian kernel:

$$\mathcal{K}(x_1, x_2) = e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}}.\tag{2.59}$$

**Why can gaussian kernel map to infinite dimension in SVM?**

$$\begin{aligned}\mathcal{K}(x_1, x_2) &= e^{-\frac{(x_1 - x_2)^2}{2\sigma^2}} = e^{-\frac{x_1^2 + x_2^2 - 2x_1 x_2}{2\sigma^2}} = e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}} \cdot e^{\frac{x_1 x_2}{\sigma^2}} \\ &= e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}} \cdot \left( 1 + \left(\frac{x_1 x_2}{\sigma^2}\right) \cdot \frac{1}{1!} + \left(\frac{x_1 x_2}{\sigma^2}\right)^2 \cdot \frac{1}{2!} + \left(\frac{x_1 x_2}{\sigma^2}\right)^3 \cdot \frac{1}{3!} + \dots + \left(\frac{x_1 x_2}{\sigma^2}\right)^n \cdot \frac{1}{n!} \right) \\ &= e^{-\frac{x_1^2 + x_2^2}{2\sigma^2}} \cdot \left( 1 + \left(\frac{x_1}{\sigma} \cdot \sqrt{\frac{1}{1!}}\right) \left(\frac{x_2}{\sigma} \cdot \sqrt{\frac{1}{1!}}\right) + \left(\frac{x_1^2}{\sigma^2} \cdot \sqrt{\frac{1}{2!}}\right) \left(\frac{x_2^2}{\sigma^2} \cdot \sqrt{\frac{1}{2!}}\right) \dots \left(\frac{x_1^n}{\sigma^n} \cdot \sqrt{\frac{1}{n!}}\right) \left(\frac{x_2^n}{\sigma^n} \cdot \sqrt{\frac{1}{n!}}\right) \right) \\ &= \phi(x_1)^\top \phi(x_2).\end{aligned}\tag{2.60}$$

$$\phi(x) = e^{-\frac{x^2}{2\sigma^2}} \left( 1, \frac{x}{\sigma} \cdot \sqrt{\frac{1}{1!}}, \frac{x^2}{\sigma^2} \cdot \sqrt{\frac{1}{2!}}, \dots, \frac{x^n}{\sigma^n} \cdot \sqrt{\frac{1}{n!}}, \dots \right).$$

## 2.8 How to get the update rule of parameters of backpropagation in gradient descent algorithm?

An intuitive and imprecise proof is as follows. Assuming that there is a differentiable loss function  $\mathcal{L}$  on training set with respect to parameter  $\vec{w} = (w_1, w_2, \dots, w_n)$  of the training model. Then, the total increment of  $\mathcal{L}$  is

$$\begin{aligned}\Delta\mathcal{L} &= \frac{\partial\mathcal{L}}{\partial w_1}\Delta w_1 + \frac{\partial\mathcal{L}}{\partial w_2}\Delta w_2 + \dots + \frac{\partial\mathcal{L}}{\partial w_n}\Delta w_n + o(\rho) \\ &\approx \frac{\partial\mathcal{L}}{\partial w_1}\Delta w_1 + \frac{\partial\mathcal{L}}{\partial w_2}\Delta w_2 + \dots + \frac{\partial\mathcal{L}}{\partial w_n}\Delta w_n\end{aligned}\quad (2.61)$$

Define  $\nabla\mathcal{L}_{\vec{w}} = (\frac{\partial\mathcal{L}}{\partial w_1}, \frac{\partial\mathcal{L}}{\partial w_2}, \dots, \frac{\partial\mathcal{L}}{\partial w_n})^T$ , which represents the vector of gradients. Therefore,

$$\Delta\mathcal{L} \approx \nabla\mathcal{L}_{\vec{w}}^T \cdot \Delta\vec{w}. \quad (2.62)$$

Our goal is to reduce the loss function  $\mathcal{L}$ , which is equivalent to making the total increment  $\Delta\mathcal{L}$  negative. Toward this end, construct

$$\Delta\vec{w} = -\eta\nabla\mathcal{L}_{\vec{w}}, \quad (2.63)$$

which ensure that  $\Delta\mathcal{L} \approx \nabla\mathcal{L}_{\vec{w}}^T \cdot \Delta\vec{w} = -\eta\|\nabla\mathcal{L}_{\vec{w}}\|_2^2 \leq 0$ . So, the update rule of parameter  $\vec{w}$  is

$$\vec{w} \leftarrow \vec{w} - \eta\nabla\mathcal{L}_{\vec{w}}. \quad (2.64)$$

In other word, in gradient descent algorithm, the direction of parameter update is opposite to its gradient direction.

## 2.9 Gaussian mixture model (GMM)

### 2.9.1 Maximum likelihood estimation

Let's start this problem with an example. Suppose that we want to make height statistics for 100 boys and 100 girls. (a) If boys and girls are investigated separately and their height follows normal distribution with parameter  $\mu, \sigma$ .

**Problem:** taking boyes for exmaple, the examples can be expressed as  $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ , where  $x^{(i)}$  represents the height of  $i$ th boys and  $n = 100$ . In this problem,  $p(x; \mu, \sigma)$  is the probability density funtion to be solved. We can suppose that  $x^{(i)}$  is independent and identically distributed.

**Solution:** we can use maximum likelihood estimation.

$$\begin{aligned}L(\mu, \sigma; x) &= p(x^{(1)}, x^{(2)}, \dots, x^{(n)}; \mu, \sigma) = \prod_{i=1}^n p(x^{(i)}; \mu, \sigma). \\ \ln L(\mu, \sigma; x) &= \ln \prod_{i=1}^n p(x^{(i)}; \mu, \sigma) \\ &= \ln\left(\frac{1}{\sqrt{2\pi}}\right)^n \left(\frac{1}{\sqrt{\sigma^2}}\right)^n \exp\left[\sum_{i=1}^n -\frac{(x^{(i)} - \mu)^2}{2\sigma^2}\right] \\ &= -\frac{n}{2} \ln 2\pi - \frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^n (x^{(i)} - \mu)^2.\end{aligned}\quad (2.65)$$

By  $x^{(i)} \sim N(\mu, \sigma)$  to solve the maximum likelihood estimation easily, we have

$$\hat{\mu}, \hat{\sigma} = \arg \max_{\mu, \sigma} L(\mu, \sigma; x). \quad (2.66)$$

Let the partial derivative of  $\mu$  and  $\sigma$  be zero, we have

$$\begin{aligned}\hat{\mu} &= \frac{1}{n} \sum_{i=1}^n x^{(i)}, \\ \hat{\sigma}^2 &= \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \hat{\mu})^2.\end{aligned}\quad (2.67)$$

**Extension:** If we use  $z^{(i)}$  express the sex that is choosed from  $\{1, \dots, k\}$ , and  $\phi$  as the parameter of the distribution of  $z$ , we can combine boys and girls to write

$$\begin{aligned}
 \ln L(\mu, \sigma; x) &= \ln \prod_{i=1}^n p(x^{(i)}; \mu, \sigma) \\
 &= \sum_{i=1}^n \ln p(x^{(i)}; \mu, \sigma) \\
 &= \sum_{i=1}^n \ln \sum_{z^{(i)}=1}^k p(x^{(i)}, z^{(i)}; \mu, \sigma) \\
 &= \sum_{i=1}^n \ln \sum_{z^{(i)}=1}^k p(x^{(i)}|z^{(i)}; \mu, \sigma) p(z^{(i)}; \phi),
 \end{aligned} \tag{2.68}$$

If we know that what the  $z^{(i)}$  is, we have

$$\ln L(\mu, \sigma; x) = \sum_{i=1}^n \ln p(x^{(i)}|z^{(i)}; \mu, \sigma) + \ln p(z^{(i)}; \phi). \tag{2.69}$$

Next, maximizing this weith respect to  $\phi$ ,  $\mu$  and  $\sigma$  gives the parameters:

$$\begin{aligned}
 \phi_j &= \frac{1}{n} \sum_{i=1}^n 1(z^{(i)} = j), \\
 \mu_j &= \frac{\sum_{i=1}^n 1(z^{(i)} = j) x^{(i)}}{\sum_{i=1}^n 1(z^{(i)} = j)}, \\
 \sigma_j &= \frac{\sum_{i=1}^n 1(z^{(i)} = j) (x^{(i)} - \mu_j)^\top (x^{(i)} - \mu_j)}{\sum_{i=1}^n 1(z^{(i)} = j)}.
 \end{aligned} \tag{2.70}$$

## 2.9.2 GMM and EM

In GMM,  $z^{(i)}$  is not known. The EM algorithm can be expressed as:

The derivation of EM algorithm

## 2.10 Principal components analysis (PCA)

### 2.10.1 Maximum variance theory

In signal processing, it is considered that the signal has a larger variance and the noise has a smaller variance. Signal-to-noise ratio is the variance ratio of the signal to noise, and the larger the better. Therefore, in PCA, we believe that the best  $k$ -dimensional feature is that after  $n$ -dimensional sample points are converted into  $k$ -dimensionas and the sample variance in each dimension is large.

### 2.10.2 PCA

1. Given a set of datasets  $X = \{\overrightarrow{x^{(1)}}, \overrightarrow{x^{(2)}}, \dots, \overrightarrow{x^{(n)}}\}$ , we centralize these data and we have

$$\begin{aligned}
 \overrightarrow{\mu} &= \frac{1}{n} \sum_{i=1}^n \overrightarrow{x^{(i)}}, \\
 \{\overrightarrow{x^{(1)}}, \overrightarrow{x^{(2)}}, \dots, \overrightarrow{x^{(n)}}\} &= \{\overrightarrow{x^{(1)}} - \overrightarrow{\mu}, \overrightarrow{x^{(2)}} - \overrightarrow{\mu}, \dots, \overrightarrow{x^{(n)}} - \overrightarrow{\mu}\}
 \end{aligned} \tag{2.72}$$

---

**Algorithm 1** EM algorithm.

**Input:** Dataset  $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$  and the number  $k$  of gaussian components.

**Output:** model parameter  $\{(\phi_j, \mu_j, \sigma_j) | 1 \leq j \leq k\}$ .

- 1: Given initial values  $\{(\phi_j, \mu_j, \sigma_j) | 1 \leq j \leq k\}$ ;
- 2: repeat until convergence: {
- 3: (E-step) For each  $i, j$ , set

$$\begin{aligned} w_j^{(i)} &= p(z^{(i)} = j | x^{(i)}; \phi, \mu, \sigma) \\ &= \frac{p(x^{(i)} | z^{(i)} = j; \mu, \sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; \mu, \sigma) p(z^{(i)} = l; \phi)} \end{aligned}$$

- 4: (M-step) Update the parameters (by MLE):

$$\begin{aligned} \phi_j &= \frac{1}{n} \sum_{i=1}^n w_j^{(i)}, \\ \mu_j &= \frac{\sum_{i=1}^n w_j^{(i)} x^{(i)}}{\sum_{i=1}^n w_j^{(i)}}, \\ \sigma_j &= \frac{\sum_{i=1}^n w_j^{(i)} (x^{(i)} - \mu_j)^\top (x^{(i)} - \mu_j)}{\sum_{i=1}^n w_j^{(i)}}. \end{aligned} \tag{2.71}$$

- 5: **return**  $\{(\phi_j, \mu_j, \sigma_j) | 1 \leq j \leq k\}$ ;
- 

2. Find a vector  $\vec{u}$  and the projection variance of the data in this direction is the largest. Then, the optimization objective is

$$\begin{aligned} \arg \max_{\vec{u}} \frac{1}{n} \sum_{i=1}^m (\vec{x}^{(i)\top} \vec{u})^2 &= \frac{1}{n} \sum_{i=1}^m \vec{u}^\top \vec{x}^{(i)} \vec{x}^{(i)\top} \vec{u} \\ &= \vec{u}^\top \left( \frac{1}{n} \sum_{i=1}^m \vec{x}^{(i)} \vec{x}^{(i)\top} \right) \vec{u} \\ &= \vec{u}^\top \left( \frac{1}{n} X X^\top \right) \vec{u} \\ &= \vec{u}^\top \left( \frac{1}{\sqrt{n}} X \frac{1}{\sqrt{n}} X^\top \right) \vec{u} \\ &= \vec{u}^\top (Y Y^\top) \vec{u}. \end{aligned} \tag{2.73}$$

We can know that  $Y Y^\top$  in above equation is a positive semi-definite symmetric matrix. Because (1)  $Y Y^\top$  is symmetric according to  $(Y Y^\top)^\top = Y Y^\top$ ; (2)  $Y Y^\top$  is positive semi-definite according to  $\xi^\top Y Y^\top \xi = (Y Y^\top \xi)^\top (Y Y^\top \xi) = \|Y Y^\top \xi\|_2^2 \geq 0$ . Therefore,  $\vec{u}^\top Y Y^\top \vec{u}$  is a positive semi-definite quadratic form, which has the maximum value.

Because there are another constraint  $\|\vec{u}\| = \vec{u}^\top \vec{u} = 1$ , we usually can solve above problem by Lagrange multiplier method.

$$\begin{aligned} f(\vec{u}) &= \vec{u}^\top Y Y^\top \vec{u} + \lambda(1 - \vec{u}^\top \vec{u}), \\ \frac{\partial f(\vec{u})}{\partial \vec{u}} &= 2(Y Y^\top \vec{u}) - 2\lambda \vec{u} = 0 \Rightarrow Y Y^\top \vec{u} = \lambda \vec{u}. \end{aligned} \tag{2.74}$$

Obviously,  $\vec{u}$  is the eigenvector corresponding to the eigenvalue  $\lambda$  of  $Y Y^\top$ . And all the eigenvalues (eigenvectors) of  $Y Y^\top$  satisfy the above formula. Then, we have

$$\vec{u}^\top (Y Y^\top) \vec{u} = \lambda \vec{u}^\top \vec{u} = \lambda. \tag{2.75}$$

So, we can see that the eigenvalue  $\lambda$  is the value of objective function. If the maximum eigenvalue is taken, then the value of objective function is taken to the maximum. Therefore, we can decompose the eigenvalues of  $Y Y^\top$  to obtain the largest  $k$  eigenvalues, and then obtain new samples of  $k$ -dimensional features by  $\vec{x}^{(i)\top} \vec{u}$ .

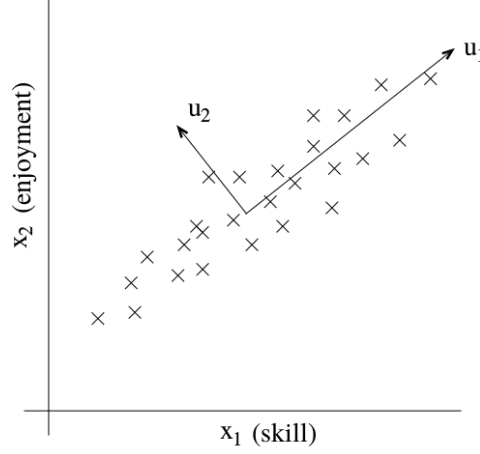


Figure 2.4: Examples.

## 2.11 Hidden Markov model (HMM)

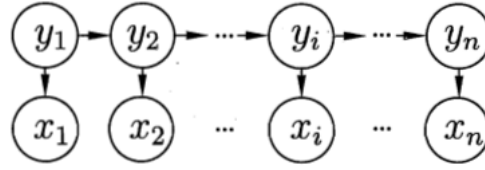


Figure 2.5: HMM

### 2.11.1 Definition

As shown in Fig. 2.11,  $y_i$  is the state sequence that is produced by the hidden Markov chain, while  $x_i$  is the observation sequence, which is produced by each state  $y_i$ .

Suppose that the set of state is  $Q$  and the set of observation is  $V$ . Moreover, we give the state sequence  $I$  with length  $T$  and the observation sequence  $O$  with length  $T$ .

$$\begin{aligned} Q &= \{q_1, q_2, \dots, q_N\}, V = \{v_1, v_2, \dots, v_M\}, \\ I &= \{i_1, i_2, \dots, i_T\}, O = \{o_1, o_2, \dots, o_T\}. \end{aligned} \quad (2.76)$$

Next, we define the state transition probability matrix as  $A$ :

$$A = [a_{ij}]_{N \times N}, a_{ij} = P(i_{t+1} = q_j | i_t = q_i). \quad (2.77)$$

Also, we define the observation probability matrix as  $B$ :

$$B = [b_{ij}]_{N \times M}, b_{ij} = P(o_t = q_j | i_t = q_i). \quad (2.78)$$

In addition, we define the initial state probability vector  $\vec{\pi}$ :

$$\vec{\pi} = (\pi_i), \pi_i = P(i_1 = q_i). \quad (2.79)$$

The two main hypotheses of Hidden Markov model includes: (1) homogeneous Markov hypothesis, (2) observation independence hypothesis.

### 2.11.2 Three basic problems

1. Probability calculation problem: given the model  $\lambda = \{A, B, \bar{\pi}\}$  and observation sequence  $O = \{o_1, o_2, \dots, o_T\}$ , calculate the probability of  $P(O|\lambda)$ ;

forward-backward algorithm

2. Learning problem: given the observation sequence  $O = \{o_1, o_2, \dots, o_T\}$ , estimate the model parameters  $\lambda = \{A, B, \bar{\pi}\}$ ;

Baum-Welch algorithm

3. Predicting problem (decoding problem): given the model  $\lambda = \{A, B, \bar{\pi}\}$  and observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  estimate the state sequence.

Viterbi algorithm

## 2.12 Conditional random field (CRF)

### 2.12.1 Probabilistic undirected graphical model, or Markov random field

### 2.12.2 Conditional random field

Let  $X$  and  $Y$  be random variables, and  $P(Y|X)$  be the conditional probability distribution of  $Y$  given  $X$ . If the random variable  $Y$  forms a Markov random field represented by an undirected graph  $G = (V, E)$ , i.e.

$$P(Y_v|X, Y_w, w \neq v) = P(Y_v|X, Y_w, w \sim v), \quad (2.80)$$

where  $w \neq v$  and  $w \sim v$  represents no connection and connection respectively, the conditional probability distribution  $P(Y|X)$  is called a conditional random field.

Let  $P(Y|X)$  be a linear chain conditional random field, the conditional probability of random variable  $Y = y$  given random variable  $X = x$ :

$$\begin{aligned} P(y|x) &= \frac{1}{Z(x)} \exp \left( \sum_k \lambda_k \sum_i t_k(y_{i-1}, y_i, x, i) + \sum_l \mu_l \sum_i s_l(y_i, x, i) \right), \\ Z(x) &= \sum_y \exp \left( \sum_k \lambda_k \sum_i t_k(y_{i-1}, y_i, x, i) + \sum_l \mu_l \sum_i s_l(y_i, x, i) \right). \end{aligned} \quad (2.81)$$

where  $t_k$  and  $s_l$  are feature function, and the former is transition feature and latter is state feature.  $\lambda_k$  and  $\mu_l$  are weights.

Above formula has a simpler form:

$$\begin{aligned} P(y|x) &= \frac{1}{Z(x)} \exp \sum_k w_k f_k(y, x), \\ Z(x) &= \sum_y \exp \sum_k w_k f_k(y, x). \end{aligned} \quad (2.82)$$

CRF algorithm

## 2.13 Generative model and discriminative model

1. Generative model: naive bayes; hidden Markov model (HMM);
2. Discriminative model: perceptron; Conditional random field (CRF)

Todo: Random forest

## Chapter 3

# Deep Network

### 3.1 How does backpropagation work?

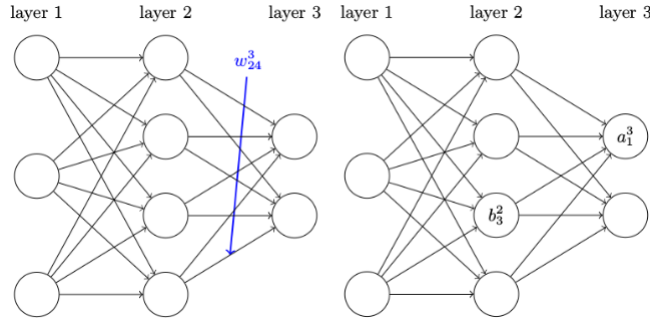


Figure 3.1: Neural network.

First, symbol definition is carried out. We use  $w^l_{jk}$  to denote the weight for the connection from the  $k$ th neuron in the  $(l-1)$ th layer to the  $j$ th neuron in the  $l$ th layer. Moreover, we use  $b^l_j$  for the bias of  $j$ th neuron in  $l$ th layer and we use  $a^l_j$  for the activation of the  $j$ th neuron in  $l$ th layer. We also use  $z^l_j$  as the weighted input of  $j$ th neuron in  $l$ th layer. Then

$$\begin{aligned} z^l_j &= \sum_k w^l_{jk} a^{l-1}_k + b^l_j, \\ a^l_j &= \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) \\ &= \sigma(z^l_j). \end{aligned} \tag{3.1}$$

where  $\sigma$  is the activation function, such as Sigmoid function. Then, Eq. 3.1 can be expressed in a vector form. In detail, we use  $\vec{a}^l$  for the activation vector in  $l$ th layer.  $\vec{z}^l$  is the same. Moreover, we use  $\vec{w}^l$  for the weight matrix and  $\vec{b}^l$  for the bias vector in  $l$ th layer. Then,

$$\begin{aligned} \vec{z}^l &= \vec{w}^l \vec{a}^{l-1} + \vec{b}^l, \\ \vec{a}^l &= \sigma(\vec{w}^l \vec{a}^{l-1} + \vec{b}^l) \\ &= \sigma(\vec{z}^l). \end{aligned} \tag{3.2}$$

The understanding of backpropagation is about how to change the weights and biases in a network to change the cost function. Ultimately, this means computing the partial derivatives  $\frac{\partial \mathcal{L}}{\partial w^l_{jk}}$  and  $\frac{\partial \mathcal{L}}{\partial b^l_j}$ . To compute those, we first introduce an intermediate quantity  $\delta^l_j$ , which we call the error of the  $j$ th neuron in the  $l$ th layer. In the actual calculation process, backpropagation will give us a prelude to compute  $\delta^l_j$ , and then will relate  $\delta^l_j$  to  $\frac{\partial \mathcal{L}}{\partial w^l_{jk}}$  and  $\frac{\partial \mathcal{L}}{\partial b^l_j}$ .

We define the error of the  $j$ th neuroon in the  $l$ th layer by

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}. \quad (3.3)$$

Then, we can get the error of the output layer  $\delta^L$ , and the components of  $\delta^L$  are given by

$$\begin{aligned} \delta_j^L &= \frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L). \end{aligned} \quad (BP1)$$

Moreover, we can give the matrix-based form of Eq. BP1 by

$$\vec{\delta}^L = \nabla_{\vec{a}^L} \mathcal{L} \otimes \sigma'(\vec{z}^L). \quad (3.4)$$

Then, we can obtain the error  $\delta^l$  in terms of the error in the next layer  $\delta^{l+1}$ . In particular,

$$\begin{aligned} \delta_j^l &= \frac{\partial \mathcal{L}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \frac{\partial \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^l}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) \end{aligned} \quad (BP2)$$

Moreover, we can give the matrix-based form of Eq. BP2 by

$$\vec{\delta}^l = ((\vec{w}^{l+1})^T \delta^{l+1}) \otimes \sigma'(\vec{z}^l). \quad (3.5)$$

Next, we can obtain the rate of change of the cost with respect to any weight in the network. In particular,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{jk}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial \sum_k w_{jk}^l a_k^{l-1} + b_j^l}{\partial w_{jk}^l} \\ &= \frac{\partial \mathcal{L}}{\partial z_j^l} a_k^{l-1} \\ &= \delta_j^l a_k^{l-1}. \end{aligned} \quad (BP3)$$

Moreover, we can give the matrix-based form of Eq. BP3 by

$$\frac{\partial \mathcal{L}}{\partial \vec{w}^l} = \vec{\delta}^l (\vec{a}^{l-1})^T. \quad (3.6)$$

We can obtain the rate of change of the cost with respect to any bias in the network. In particular,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b_j^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^l} \frac{\partial \sum_k w_{jk}^l a_k^{l-1} + b_j^l}{\partial b_j^l} \\ &= \frac{\partial \mathcal{L}}{\partial z_j^l} \\ &= \delta_j^l. \end{aligned} \quad (BP4)$$

Moreover, we can give the matrix-based form of Eq. BP4 by

$$\frac{\partial \mathcal{L}}{\partial \vec{b}^l} = \vec{\delta}^l. \quad (3.7)$$



## 3.2 Backpropagation of Convolutional Neural Network

### 3.2.1 Backpropagation of Fully Connectional Layer

The backpropagation equations of element form are as follows:

$$\begin{cases} \delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L), \\ \delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l), \\ \frac{\partial \mathcal{L}}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}, \\ \frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l. \end{cases} \quad (\text{FC-BP})$$

The backpropagation equations of matrix-based form are as follows:

$$\begin{cases} \vec{\delta}^L = \nabla_{\vec{a}^L} \mathcal{L} \otimes \sigma'(\vec{z}^L), \\ \vec{\delta}^l = ((\vec{w}^{l+1})^T \vec{\delta}^{l+1}) \otimes \sigma'(\vec{z}^l), \\ \frac{\partial \mathcal{L}}{\partial \vec{w}^l} = \vec{\delta}^l (\vec{a}^{l-1})^T, \\ \frac{\partial \mathcal{L}}{\partial \vec{b}^l} = \vec{\delta}^l. \end{cases} \quad (3.8)$$

### 3.2.2 Backpropagation of Convolutional Layer

We use  $w_{pq}^l$  to denote the weight of the element in the  $p$ th row and  $q$ th column of the convolutional kernel in  $l$ th layer. Moreover,  $a_{ij}^l$  and  $z_{ij}^l$  as the activation and weighted input in  $i$ th row and  $j$ th column in  $l$ th layer, respectively. Then,

$$\begin{aligned} z_{ij}^l &= \sum_{p=1}^r \sum_{q=1}^c a_{i+p-1, j+q-1}^{l-1} w_{pq}^l + b^l, \\ a_{ij}^l &= \sigma(z_{ij}^l), \end{aligned} \quad (3.9)$$

where  $\sigma$  is the activation function, such as Sigmoid function. The matrix-based form of Eq. 3.9 can be given by

$$\begin{aligned} \mathbf{z}^l &= \mathbf{a}^{l-1} * \mathbf{w}^l + \mathbf{b}^l, \\ \mathbf{a}^l &= \sigma(\mathbf{z}^l). \end{aligned} \quad (3.10)$$

Similar to fully connectional layer, we define  $\delta_{ij}^l$  as the error in element at  $i$ th row and  $j$ th column in  $l$  layer. In detail,

$$\delta_{ij}^l = \frac{\partial \mathcal{L}}{\partial z_{ij}^l}. \quad (3.11)$$

In this equation,  $\delta^L$  can be obtain by the same manner as Eq. BP1. Then, we can obtain the error  $\delta^l$  in the terms of the error in the next layer  $\delta^{l+1}$ . In particular,

$$\begin{aligned} \delta_{ij}^l &= \frac{\partial \mathcal{L}}{\partial z_{ij}^l} = \sum_m \sum_n \frac{\partial \mathcal{L}}{\partial z_{mn}^{l+1}} \frac{\partial z_{mn}^{l+1}}{\partial z_{ij}^l} \\ &= \sum_m \sum_n \delta_{mn}^{l+1} \frac{\partial z_{mn}^{l+1}}{\partial z_{ij}^l} \\ &= \sum_m \sum_n \delta_{mn}^{l+1} \frac{\partial \sum_{p=1}^r \sum_{q=1}^c \sigma(z_{m+p-1, n+q-1}^l) w_{pq}^{l+1} + b^{l+1}}{\partial z_{ij}^l} \\ &= \sum_m \sum_n \delta_{mn}^{l+1} w_{i+1-m, j+1-n}^{l+1} \sigma'(z_{ij}^l). \end{aligned} \quad (\text{CONV-BP2})$$

In this case, an example can be found:

$$\begin{aligned}\delta_{1,1}^l = & \delta_{-1,-1}^{l+1} w_{3,3}^{l+1} + \delta_{-1,0}^{l+1} w_{3,2}^{l+1} + \delta_{-1,1}^{l+1} w_{3,1}^{l+1} + \\ & \delta_{0,-1}^{l+1} w_{2,3}^{l+1} + \delta_{0,0}^{l+1} w_{2,2}^{l+1} + \delta_{0,1}^{l+1} w_{2,1}^{l+1} + \\ & \delta_{1,-1}^{l+1} w_{1,3}^{l+1} + \delta_{1,0}^{l+1} w_{1,2}^{l+1} + \delta_{1,1}^{l+1} w_{1,1}^{l+1},\end{aligned}\quad (3.12)$$

which can be represented as convolutional operations like:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \delta_{1,1}^{l+1} & \delta_{1,2}^{l+1} & 0 & 0 \\ 0 & 0 & \delta_{2,1}^{l+1} & \delta_{2,2}^{l+1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} w_{3,3}^{l+1} & w_{3,2}^{l+1} & w_{3,1}^{l+1} \\ w_{2,3}^{l+1} & w_{2,2}^{l+1} & w_{2,1}^{l+1} \\ w_{1,3}^{l+1} & w_{1,2}^{l+1} & w_{1,1}^{l+1} \end{bmatrix}.\quad (3.13)$$

Therefore, Eq. CONV-BP2 can be deduced into matrix-based form, *i.e.*,

$$\delta^l = \delta^{l+1} * \text{rot180}(\mathbf{w}) \otimes \sigma'(\mathbf{z}^l). \quad (3.14)$$

Next, we can calculate the change rate of the cost with respect to any weight in the network. In particular,

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{pq}^l} &= \sum_i \sum_j \frac{\partial \mathcal{L}}{\partial z_{ij}^l} \frac{\partial z_{ij}^l}{\partial w_{pq}^l} \\ &= \sum_i \sum_j \frac{\partial \mathcal{L}}{\partial z_{ij}^l} \frac{\partial \sum_{p=1}^r \sum_{q=1}^c a_{i+p-1,j+q-1}^{l-1} w_{pq}^l + b^l}{\partial w_{pq}^l} \\ &= \sum_i \sum_j \delta_{ij}^l a_{i+p-1,j+q-1}^{l-1}.\end{aligned}\quad (\text{CONV-BP3})$$

As for bias, we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b^l} &= \sum_i \sum_j \frac{\partial \mathcal{L}}{\partial z_{ij}^l} \frac{\partial z_{ij}^l}{\partial b^l} \\ &= \sum_i \sum_j \frac{\partial \mathcal{L}}{\partial z_{ij}^l} \frac{\partial \sum_{p=1}^r \sum_{q=1}^c a_{i+p-1,j+q-1}^{l-1} w_{pq}^l + b^l}{\partial b^l} \\ &= \sum_i \sum_j \delta_{ij}^l.\end{aligned}\quad (\text{CONV-BP4})$$

In summary, the backpropagation equations of element form for convolutional layer as follows:

$$\begin{cases} \delta_{ij}^l = \sum_m \sum_n \delta_{mn}^{l+1} w_{i+1-m,j+1-n}^{l+1} \sigma'(z_{ij}^l), \\ \frac{\partial \mathcal{L}}{\partial w_{pq}^l} = \sum_i \sum_j \delta_{ij}^l a_{i+p-1,j+q-1}^{l-1}, \\ \frac{\partial \mathcal{L}}{\partial b^l} = \sum_i \sum_j \delta_{ij}^l. \end{cases} \quad (\text{CONV-BP})$$

Moreover, the corresponding matrix form is as follows:

$$\begin{cases} \delta^l = \delta^{l+1} * \text{rot180}(\mathbf{w}) \otimes \sigma'(\mathbf{z}^l), \\ \frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \mathbf{a}^{l-1} * \delta^l, \\ \frac{\partial \mathcal{L}}{\partial \mathbf{b}^l} = \sum_{i,j} b^l. \end{cases} \quad (3.15)$$

### 3.3 Why does batch normalization work?

#### 3.3.1 Dataset shift and covariate shift

There exists an inherent hypothesis in supervised learning: The training data in source domain and testing data in target domain are independent and identically drawn from the same distribution.

Usually, the distributions of training data and testing data are not exactly the same, especially when the training data is insufficient. This phenomenon is named as “dataset shift”.

In analysis of covariance (ANOVA), covariate is any continuous variable that is usually not controlled during data collection. (referring to <https://support.minitab.com/en-us/minitab/18/help-and-how-to/modeling-statistics/anova/supporting-topics/anova-models/understanding-covariates/>).

For the input variable  $x$  (also named as explanatory variable or covariate in scene of batch normalization),  $y$  is the corresponding label, which is regarded as response variable. If the conditional probabilities of training data and testing data are the same but the marginal probabilities are different, *i.e.*

$$\begin{aligned} P_{tr}(y|x) &= P_{te}(y|x) \\ P_{tr}(x) &\neq P_{te}(x) \end{aligned} \quad (3.16)$$

we call this phenomenon as “covariate shift”. Covariate shift is a special kind of dataset shift.

Moreover, we can deduce the Eq. 3.16 as follows:

$$\begin{aligned} P_{tr}(x, y) &= P_{tr}(y|x)P_{tr}(x) \\ P_{te}(x, y) &= P_{te}(y|x)P_{te}(x) \end{aligned} \quad (3.17)$$

and we know the joint probabilities of training data and testing data are different, *i.e.*,

$$P_{tr}(x, y) \neq P_{te}(x, y). \quad (3.18)$$

. From Eq. 3.18, this scene violates the independence hypothesis of supervised learning, so the prediction is inaccurate.

A more precise definition of **covariate** can't be found.

#### 3.3.2 Internal covariate shift

Internal covariate shift is defined as the change in the distribution of network activations due to the change in network parameters during training.

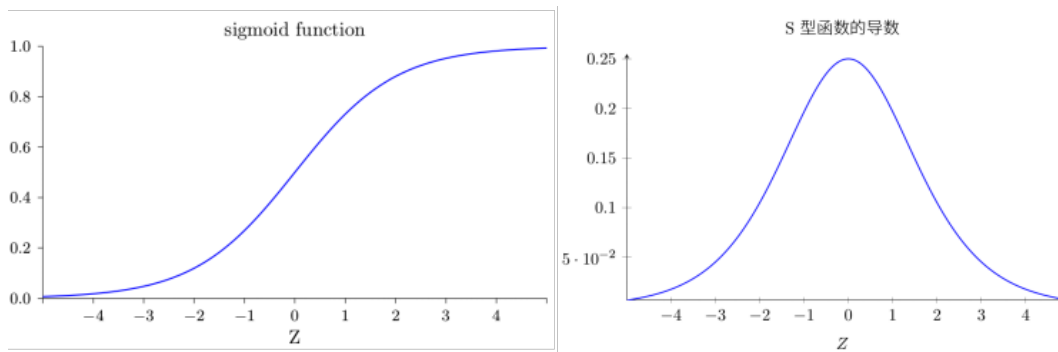


Figure 3.2: Derivative of sigmoid function.

**Consequences of internal covariate shift:** (1) Intuitively, the layers need to continuously adapt to the new distribution, which lead to a decrease in training speed. (2) Objectively, the changes of  $W$  and  $b$  during training will likely move many dimensions of  $x$  into the saturated regime of the nonlinearity (*e.g.* sigmoid and tanh) and slow down the convergence. The saturation problem is usually addressed by using ReLU, careful initialization and small learning rates. Moreover, if the distribution of nonlinearity inputs remains more stable at the sensitive regions in nonlinearity as the network trains, the optimizer would be less likely to get stuck in the saturated regime, and the training would accelerate.

PCA whitening and ZCA Whitening.

### 3.3.3 Batch normalization

**Problem:** because whitening is expensive and maybe change the representation ability of a network by discarding the absolute scale of activations, batch normalization emerges.

There are two treatments for batchnorm to address above two problems. (1) The first is instead of whitening the features in layer inputs and outputs jointly, batchnorm normalizes each scalar feature independently, by making it have the mean of 0 and the variance of 1. For a layer with  $d$ -dimensional input  $x = [x^{(1)}, x^{(2)}, \dots, x^{(k)}]$ , each dimension is normalized as follows

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}(x^{(k)})}}. \quad (3.19)$$

(2) Simply normalizing each input of a layer may reduce what the layer can represent. For example, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. In this way, batchnorm turns the multi-layer

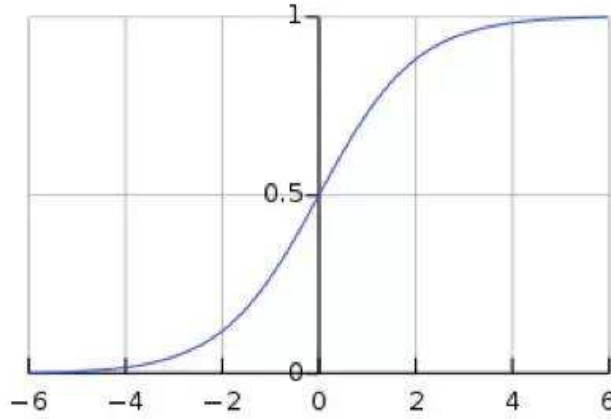


Figure 3.3: Sigmoid function.

nonlinear neural network into a multi-layer linear neural network, which is equivalent to the representation ability of a single layer linear network. This reduces the representation ability of the network. So, introduce  $\gamma^{(k)}, \beta^{(k)}$  to scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}, \quad (3.20)$$

which guarantee the representation ability of the network.

Why is multi-layer linear network equivalent to the representation ability of a single layer linear network.

In addition, use mini-batch in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation. And obtain the forward algorithm of batch normalization as follows:

---

**Algorithm 2** Forward algorithm of batch normalization.

---

**Input:** values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1, x_2, \dots, x_m\}$ , parameters to be learned:  $\gamma, \beta$

**Output:**  $y_i = BN_{\gamma, \beta}(x_i)$

$$\begin{aligned} \mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta = BN_{\gamma, \beta}(x_i) \end{aligned}$$


---

In this manner, we know

$$\begin{aligned}
E[\hat{x}] &= E\left[\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right] \\
&= \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}(E[x_i] - E[\mu_{\mathcal{B}}]) \\
&= 0 \\
Var[\hat{x}] &= E[\hat{x}^2] - (E[\hat{x}])^2 \\
&= E\left[\left(\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}\right)^2\right] + 0 \\
&= \frac{1}{\sigma_{\mathcal{B}}^2 + \epsilon} E[(x_i - \mu_{\mathcal{B}})^2] \\
&= \frac{1}{\sigma_{\mathcal{B}}^2 + \epsilon} \sigma_{\mathcal{B}}^2 \\
&\approx 1
\end{aligned} \tag{3.21}$$

The distribution of values of any  $\hat{x}$  has the expected value of 0 and the invariance of 1, as long as the elements of each mini-batch are sampled from the same distribution, and if we neglect  $\epsilon$ . This distribution can be seen as an approximate standard normal distribution  $N(0, 1)$ .

We compute the gradients with respect to the parameters of the BN transform of loss  $\mathcal{L}$  for backpropagation. The chain rule is used as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \gamma} &= \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial \gamma} = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \hat{x}_i, \\
\frac{\partial \mathcal{L}}{\partial \beta} &= \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial \beta} = \sum_i \frac{\partial \mathcal{L}}{\partial y_i}.
\end{aligned} \tag{3.22}$$

There exists  $\sum_i$  for  $y_i$  because gradient from each  $y_i$  will backpropagate for  $\gamma$  and the effect of summation is achieved.

For the process of **backpropagation**, we also need to compute  $\frac{\partial \mathcal{L}}{\partial x_i}$ , which is got by:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \hat{x}_i} &= \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial \hat{x}_i} \\
&= \frac{\partial \mathcal{L}}{\partial y_i} \gamma \\
\frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} &= \sum_i \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \sigma_{\mathcal{B}}^2} \\
&= \sum_i \frac{\partial \mathcal{L}}{\partial \hat{x}_i} (x_i - \mu_{\mathcal{B}}) \left(-\frac{1}{2}\right) (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}} \\
&= -\frac{1}{2} \sum_i \frac{\partial \mathcal{L}}{\partial \hat{x}_i} (x_i - \mu_{\mathcal{B}}) (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{3}{2}} \\
\frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} &= \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \mu_{\mathcal{B}}} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \frac{\partial \sigma_{\mathcal{B}}^2}{\partial \mu_{\mathcal{B}}} \\
&= \sum_i \frac{\partial \mathcal{L}}{\partial \hat{x}_i} (-1) (\sigma_{\mathcal{B}}^2 + \epsilon)^{-\frac{1}{2}} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \frac{1}{m} \sum_i (-2) (x_i - \mu_{\mathcal{B}}) \\
&= -\sum_i \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} - \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \frac{2}{m} \sum_i (x_i - \mu_{\mathcal{B}}) \\
\frac{\partial \mathcal{L}}{\partial x_i} &= \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} \frac{\partial \sigma_{\mathcal{B}}^2}{\partial x_i} + \frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}} \frac{\partial \mu_{\mathcal{B}}}{\partial x_i} \\
&= \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{2}{m} \frac{\partial \mathcal{L}}{\partial \sigma_{\mathcal{B}}^2} (x_i - \mu_{\mathcal{B}}) + \frac{1}{m} \frac{\partial \mathcal{L}}{\partial \mu_{\mathcal{B}}}
\end{aligned} \tag{3.23}$$

For the **inference** of the network, sometimes we want the output to depend only on the few or one input. At this time,  $\mu$  and  $\sigma^2$  are computed by each mini-batch  $\mu_B$  and  $\sigma_B^2$  by their unbiased estimations as proved in 1.19:

$$\begin{aligned}\mu &= E(\mu_B), \\ \sigma^2 &= \frac{n}{n-1} E(\sigma_B^2), \\ BN(X) &= \gamma \frac{X - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta.\end{aligned}\tag{3.24}$$

The advantages of batch normalization is as follows: (1) batch normalization ensure the stable distributions of input in each layer, which reduces the learning pressure at the later layer and improves the learning speed; (2) batch normalization makes the model less sensitive to the parameters in the network, which simplifies the parameter adjustment precess, and makes the learning of this network more stable; (3) batch normalization can make the input of activation fall into the unsaturated region through the normalization operation, thus alleviating the problem of gradient vanishing.

### 3.4 Why does residual learning work?

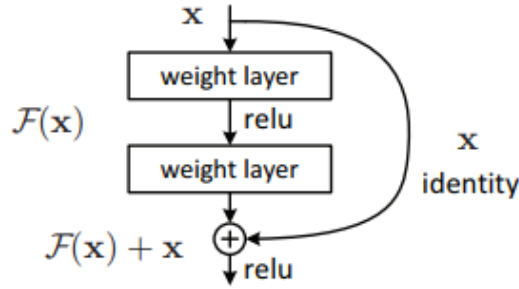


Figure 3.4: Residual Learning.

In the process of optimization in deep network k, the input and output are usually close in the latter layers. In some latter layers, we define the input is  $x$  and the output is  $\mathcal{H}(x)$ . From above experience, we assume  $x$  and  $\mathcal{H}(x)$  are close. Thus, the function of these layers is mapping from  $x$  to  $\mathcal{H}(x)$ . But usually this process is difficult because the relative gap between  $x$  and  $\mathcal{H}(x)$  is small. Toward this end, we construct the residual as  $\mathcal{F}(x) = \mathcal{H}(x) - x$  to learning the gap. We can hypothesize that it is easier to optimize the residual mapping than to optimize the original mapping. To the extreme, if the mapping from input to output is identity, it would be easier to push the residual to zero than to directly fit an identity mapping.

The real purpose of residual learning is that even if the network deepens, the performance of this network will not degenerate, thus ensuring the leaning of deeper network (even 1000 layers).

### 3.5 Understanding Deconvolution & bilinear interpolation

For the convolutional operation, if we have the input map  $I$  and the kernel  $K$ , we can get the output  $O$ ,

$$I = \begin{bmatrix} 4 & 5 & 8 & 7 \\ 1 & 8 & 8 & 8 \\ 3 & 6 & 6 & 4 \\ 6 & 5 & 7 & 8 \end{bmatrix}, K = \begin{bmatrix} 1 & 4 & 1 \\ 1 & 4 & 3 \\ 3 & 3 & 1 \end{bmatrix}.\tag{3.25}$$

When computing convolutional operation by computer, convolutional operation is usually written as matrix multiplication. The form is

$$M(O) = M(W)M(I),\tag{3.26}$$

where  $M(O)$ ,  $M(W)$  and  $M(I)$  are matrices of output, weight and input. Then, we have the next form,

$$M(W) = \begin{bmatrix} 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 \end{bmatrix}, M(I) = \begin{bmatrix} 4 \\ 5 \\ 8 \\ 8 \\ 7 \\ 1 \\ 8 \\ 8 \\ 8 \\ 8 \\ 3 \\ 6 \\ 6 \\ 6 \\ 4 \\ 6 \\ 5 \\ 7 \\ 8 \end{bmatrix}, M(O) = M(W)M(I) = \begin{bmatrix} 122 \\ 148 \\ 126 \\ 134 \end{bmatrix} \quad (3.27)$$

### 3.5.1 Why is it called Transposed Convolution?

We can observe that the size of input matrix is  $4 \times 4$  and the one of weight is  $3 \times 3$ , so that the size of output matrix is  $2 \times 2$ . If we want to get the input by the output  $M(O)$  with the convolutional kernel  $M(W)$ , we can get the input,

$$M(I) = M(W)^\top M(O). \quad (3.28)$$

which is not the reverse operation of Eq. 3.27 and only represents the shape restoration.

In the implement of Transposed Convolution, the matrix of weights is transposed, so that the operation is called tranposed convolution.

There are three implementations of transposed convolution. Suppose that  $I = [5 \ 8 \ 11]$ ,  $W = [1 \ 2]$ .

(1) above matrix form

$$M(W)^\top = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}, M(I) = \begin{bmatrix} 5 \\ 8 \\ 11 \end{bmatrix}, M(O) = M(W)^\top M(I) = \begin{bmatrix} 5 \\ 18 \\ 27 \\ 22 \end{bmatrix}. \quad (3.29)$$

(2) pad input and convolve it with  $\text{rot180}(W)$ . For example, input  $I$ , weight  $O$

$$\begin{aligned} O &= I * \text{rot180}(W) \\ &= [0 \ 5 \ 0 \ 8 \ 0 \ 11] * [2 \ 1]. \\ &= [5 \ 18 \ 27 \ 22] \end{aligned} \quad (3.30)$$

(3) For example,

$$[5 \ 8 \ 11] \oslash [1 \ 2] = [5 * 1 \ 5 * 2 + 8 * 1 \ 8 * 2 + 11 * 1 \ 11 * 2] = [5 \ 18 \ 27 \ 22]. \quad (3.31)$$

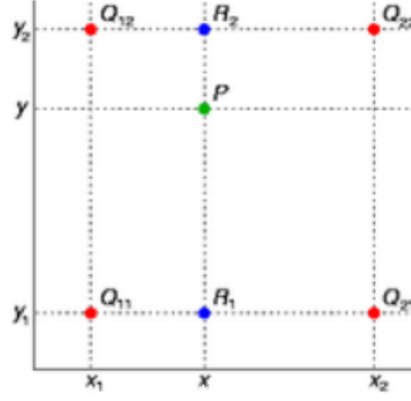


Figure 3.5: Bilinear interpolation

### 3.5.2 Bilinear Interpolation

As shown in Figure, suppose that  $Q_{11} = (x_1, y_1)$ ,  $Q_{12} = (x_1, y_2)$ ,  $Q_{21} = (x_2, y_1)$ ,  $Q_{22} = (x_2, y_2)$ . For the point  $P = (x, y)$ , we can compute the value of  $R_1$  and  $R_2$  firstly.

$$\begin{aligned}
 f(R_1) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \\
 f(R_2) &= \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}), \\
 f(P) &= \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2) \\
 &= \frac{(x_2 - x)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{11}) + \frac{(x - x_1)(y_2 - y)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{21}) \\
 &\quad + \frac{(x_2 - x)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{12}) + \frac{(x - x_1)(y - y_1)}{(x_2 - x_1)(y_2 - y_1)} f(Q_{22})
 \end{aligned} \tag{3.32}$$

In a deep neural network, we usually use bilinear interpolation to upsample a feature map, which is usually implemented by a transposed convolution initialing by bilinear kernel. And the transposed convolution is usually needed to learn. For example, there is a bilinear kernel of size  $4 \times 4$ .

$$\begin{bmatrix} 0.0625 & 0.1875 & 0.1875 & 0.0625 \\ 0.1875 & 0.5625 & 0.5625 & 0.1875 \\ 0.1875 & 0.5625 & 0.5625 & 0.1875 \\ 0.0625 & 0.1875 & 0.1875 & 0.0625 \end{bmatrix}. \tag{3.33}$$

There is the code to get the bilinear kernel of size *size*.



```

1 import numpy as np
2 def upsample_filt(size): # size = 4
3     """
4     Make a 2D bilinear kernel suitable for upsampling of the given (h, w) size.
5     """
6     factor = (size + 1) // 2 # factor = 2
7     if size % 2 == 1:
8         center = factor - 1
9     else:
10        center = factor - 0.5 # size = 1.5
11    og = np.ogrid[:size, :size]
12    return (1 - abs(og[0] - center) / factor) * \
13           (1 - abs(og[1] - center) / factor)
14 upsample_filt(4)
15 # output
16 # array([[0.0625, 0.1875, 0.1875, 0.0625],
17 #        [0.1875, 0.5625, 0.5625, 0.1875],
18 #        [0.1875, 0.5625, 0.5625, 0.1875],
19 #        [0.0625, 0.1875, 0.1875, 0.0625]])
20 upsample_filt(6)
21 # array([[0.02777778, 0.08333333, 0.13888889, 0.13888889, 0.08333333,
22 #        0.02777778],
23 #        [0.08333333, 0.25, 0.41666667, 0.41666667, 0.25,
24 #        0.08333333],
25 #        [0.13888889, 0.41666667, 0.69444444, 0.69444444, 0.41666667,
26 #        0.13888889],
27 #        [0.13888889, 0.41666667, 0.69444444, 0.69444444, 0.41666667,
28 #        0.13888889],
29 #        [0.08333333, 0.25, 0.41666667, 0.41666667, 0.25,
30 #        0.08333333],
31 #        [0.02777778, 0.08333333, 0.13888889, 0.13888889, 0.08333333,
32 #        0.02777778]])

```

## 3.6 Activation Function

Since the preception is equivalent to a NAND gate (which can be regarded as universal gate), the multilayer perception can fit any linear function. However, in general, many problems are not linearly seperable, so it is necessary to introduce nonlinear transformation.

### 3.6.1 Sigmoid

Simoid is widely used for improving the nondifferentiable jump function. However, in biology, neurons are considered to be sparsely activated, with only 1%-4% activated at the same time. Meanwhile, Sigmoid has a amount of computation for derivative calculation and has the problem of gradient vanishing, which leads to the difficult of deep network training. Thus ReLU (2011, Bengio and Xavier) appears.

### 3.6.2 ReLU

ReLU maybe has the problem of Dead Neurons, i.e., some parameters never are activated. The reasons are: (1) bad initialization; (2) the learning rete is too high and the parameters are updated greatly, which makes the network neter this state.

The solutions to above problem include: (1) initializing with Xavier; (2) use adagrad or other methods to adaptively adjust learning rate; (3) use some improvement of ReLU (e.g., leaky ReLU, PReLU and ELU (Exponential ReLU)).

However, the non-linearity of ReLU is weak, which leads to better fitting in deeper network.

## 3.7 Parameter Initialization

### 3.7.1 Why not initialized with all zeros?

Take the task of image classification as an example, the sensitivity of neurons in the output layer is  $\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L)$  and the gradient of weights in the output layer is  $\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \delta_j^L a_k^{L-1}$ .

For single-layer perceptron (logistic regression), the model can be trained normally. The reason is the different weights  $w_{jk}^L$  have different updates.

But for multi-layer neural network, the training fails. (1) If the activation function of hidden layers is ReLU, all the outputs, gradients and weights in layers are zeros, which means that all the neurons except inputs are dead neurons. Thus the learning fails. (2) If Sigmoid is the used activation function, the neurons can be learned but all the outputs, gradients and weights are same, which means that the learning doesn't work.

### 3.7.2 Why not initialized with Standard Gaussian?

If there are  $n_{in}$  (such as 50) ones in the input layer, the output after first layer is belongs to  $\mathcal{N}(0, \sqrt{50})$ . After activated, the output saturates. The gradient is about 0 (for Sigmoid) and the learning is slow.

Solution: we usually use the Gaussian Distribution  $\mathcal{N}(0, \sqrt{n_{in}})$  to initialize the weights, so that the activation does't fall into the saturation region.

### 3.7.3 xavier

**Glorot Condition:** In order to keep information flowing, the invariance of the activation value remains same during forward-propagation and the invariance of the gradient with respect to the state value remains same during back-propagation.

There are some assumptions: 1) the linear activation function is used and the derivative at zero is 1; 2) activation value  $x$  and weight  $w$  are independent.

Then, we have

$$\begin{aligned} a_i^l &= \sum_{k=1}^{n_{in}} w_{ik}^l x_k^l, \\ \text{Var}(w_{ik} x_k) &= E(w_{ik}^2 x_k^2) - E^2(w_{ik} x_k) \\ &= E(w_{ik}^2) E(x_k^2) - E^2(w_{ik}) E^2(x_k) \\ &= E^2(w_{ik}) \text{Var}(x_k) + E^2(x_k) \text{Var}(w_{ik}) + \text{Var}(x_k) \text{Var}(w_{ik}). \end{aligned} \quad (3.34)$$

Where  $n_{in}$  is the number of input neurons.

If  $E(w_{ik}^l) = E(x_k^l) = 0$ , we have

$$\begin{aligned} \text{Var}(a_i^l) &= \sum_{k=1}^{n_{in}} \text{Var}(x_k^l) \text{Var}(w_{ik}^l) \\ &= n_{in} \text{Var}(x_k^l) \text{Var}(w_{ik}^l) \\ &= n_{in} \text{Var}(x^l) \text{Var}(w^l) \\ &= n_{l-1} \text{Var}(a^{l-1}) \text{Var}(w^l). \end{aligned} \quad (3.35)$$

If the invariance of the activation value remains same during forward-propagation, we need  $\text{Var}(a) = \text{Var}(x)$  and we have  $\text{Var}(w) = \frac{1}{n_{in}}$ .

If the invariance of the gradient with respect to the state value remains same during back-propagation, we need  $\text{Var}(\frac{\partial \mathcal{L}}{\partial z^l}) =$

$Var(\frac{\partial \mathcal{L}}{\partial z_k^{l+1}})$ , and we have

$$\begin{aligned}
 Var(\frac{\partial \mathcal{L}}{\partial z_i^l}) &= Var(\sum_{k=1}^{n_{out}} w_{ik}^{l+1} \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \sigma'(z_k^l)) \\
 &= Var(n_{out} w_{ik}^{l+1} \frac{\partial \mathcal{L}}{\partial z_k^{l+1}}) \\
 &= n_{out} Var(w_{ik}^{l+1}) Var(\frac{\partial \mathcal{L}}{\partial z_k^{l+1}}) \\
 &= n_{l+1} Var(w_{ik}^{l+1}) Var(\frac{\partial \mathcal{L}}{\partial z_k^{l+1}}).
 \end{aligned} \tag{3.36}$$

Then, we have  $Var(w) = \frac{1}{n_{out}}$ . Since  $n_{in}$  and  $n_{out}$  are usually not equal, the average is taken. Thus we have  $Var(w) = \frac{2}{n_{in} + n_{out}}$ .

Under this condition, if  $w$  obeys uniform distribution with a mean of 0, then we have  $w \sim \mathcal{U}[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}]$ .

This is Xavier initialization method. A detailed derivation can be found in the blog <https://blog.csdn.net/victoriaw/article/details/73000632>.

### 3.7.4 msra

In the derivation of xavier in Eq. 3.35 and ??, a linear activation function is required. However, ReLU is not a linear function. In general, if ReLU is used, we can't guarantee that **Glorot Condition** is met (i.e., the invariance of the activation value remains same during forward-propagation and the invariance of the gradient with respect to the state value remains same during back-propagation).

In order to maintain a stable variance, Kaiming et al. propose a new initialization method and the purpose is to achieve that the invariance of the state value remains same during forward-propagation and the invariance of the gradient with respect to the activation value remains same during back-propagation.

From this assumption, we have Then, we have

$$\begin{aligned}
 z_i^l &= \sum_{k=1}^{n_{in}} w_{ik}^l x_k^l, \\
 a_i^l &= f(z_i^l), \\
 Var(w_{ik} x_k) &= E(w_{ik}^2 x_k^2) - E^2(w_{ik} x_k) \\
 &= E(w_{ik}^2) E(x_k^2) - E^2(w_{ik}) E^2(x_k)
 \end{aligned} \tag{3.37}$$

Where  $n_{in}$  is the number of input neurons.

If  $E(w_{ik}) = 0$  and  $E(x_k) \neq 0$  because the activation function is ReLU, we have

$$\begin{aligned}
 Var(z_i^l) &= \sum_{k=1}^{n_{in}} Var(w_{ik}^l x_k^l) \\
 &= n_{in} E([w_{ik}^l]^2) E([x_k^l]^2) \\
 &= n_{in} Var(w_{ik}^l) E([x_k^l]^2). \\
 E(z_i^{l-1}) &= E(w_{ik}^{l-1} x_k^{l-1}) \\
 &= E(w_{ik}^{l-1}) E(x_k^{l-1}) \\
 &= 0
 \end{aligned} \tag{3.38}$$

$$\begin{aligned}
E([x_i^l]^2) &= E([f(z_i^{l-1})]^2) \\
&= \int_{-\infty}^{\infty} p(z_i^{l-1}) [f(z_i^{l-1})]^2 dz_i^{l-1} \\
&= \int_{-\infty}^0 p(z_i^{l-1}) [f(z_i^{l-1})]^2 dz_i^{l-1} + \int_0^{\infty} p(z_i^{l-1}) [f(z_i^{l-1})]^2 dz_i^{l-1} \\
&= 0 + \int_0^{\infty} p(z_i^{l-1}) [z_i^{l-1}]^2 dz_i^{l-1} \\
&= \frac{1}{2} \int_0^{\infty} p(z_i^{l-1}) [z_i^{l-1}]^2 dz_i^{l-1} \\
&= \frac{1}{2} E([z_i^{l-1}]^2) \\
&= \frac{1}{2} Var(z_i^{l-1})
\end{aligned} \tag{3.39}$$

$$\begin{aligned}
Var(z_i^l) &= n_{in} Var(w_{ik}^l) E([x_i^l]^2) \\
&= \frac{1}{2} n_{in} Var(w_{ik}^l) Var(z_i^{l-1})
\end{aligned} \tag{3.40}$$

If the invariance of the state value remains same during forward-propagation, we need  $Var(z^l) = Var(z^{l-1})$  and we have  $Var(w) = \frac{2}{n_{in}}$ .

If the invariance of the graident with respect to the activation value remains same during back-propagation, we need  $Var(\frac{\partial \mathcal{L}}{\partial a^l}) = Var(\frac{\partial \mathcal{L}}{\partial a^{l+1}})$ , and we have

$$\frac{\partial \mathcal{L}}{\partial a_i^l} = \sum_{k=1}^{n_{l+1}} w_{ik}^{l+1} \frac{\partial \mathcal{L}}{\partial a_k^{l+1}} \tag{3.41}$$

$$\begin{aligned}
Var(\frac{\partial \mathcal{L}}{\partial a_i^l}) &= n_{l+1} Var(w_{ik}^{l+1} \frac{\partial \mathcal{L}}{\partial a_k^{l+1}}) \\
&= n_{l+1} E([w_{ik}^{l+1}]^2) E([\frac{\partial \mathcal{L}}{\partial a_k^{l+1}}]^2) \\
&= n_{l+1} Var(w_{ik}^{l+1}) E([\frac{\partial \mathcal{L}}{\partial a_k^{l+1}}]^2 [\frac{\partial a_k^{l+1}}{\partial z_k^{l+1}}]^2) \\
&= n_{l+1} Var(w_{ik}^{l+1}) E([\frac{\partial \mathcal{L}}{\partial a_k^{l+1}}]^2) E([f'(z_k^{l+1})]^2)
\end{aligned} \tag{3.42}$$

We can know that  $E([f'(z_k^{l+1})]^2) = \frac{1}{2}$ . Moreover, from Eq. 3.41, we can know that  $E(\frac{\partial \mathcal{L}}{\partial a_i^{l+1}}) = 0$ . Thus we have

$$Var(\frac{\partial \mathcal{L}}{\partial a_i^l}) = \frac{1}{2} n_{l+1} Var(w_{ik}^{l+1}) Var(\frac{\partial \mathcal{L}}{\partial a_k^{l+1}}). \tag{3.43}$$

And we have that  $Var(w_{ik}^{l+1}) = \frac{2}{n_{l+1}}$ , i.e.  $Var(w_{ik}^l) = \frac{2}{n_l}$ .

Under this condition, if  $w$  obeys gaussian distribution with a mean of 0, then we have  $w \sim \mathcal{N}[0, \frac{2}{n_{in}}]$ .

This is Msra initialization method. A detailed derivation can be found in the blog <https://blog.csdn.net/VictoriaW/article/details/73166752> and <https://www.cnblogs.com/wangguchangqing/p/11013698.html>.

We can know that Xavier and Msra can both belong to uniform distribution or gaussian distribution.

## Chapter 4

### Contents specifically referenced

- Sect. 2.6.1: the chapter of (2)Perceptron in [2];
- Sect. 3.3.3: BatchNorm from [1];



# Bibliography

- [1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.
- [2] Hang Li. Statistical learning method. Tsinghua University Press, 2012.