Mathematics, Machine Learning and Deep Learning Notes

Jinming Su

Last update: July 15, 2019

Contents

Todo list

Chapter 1

Mathematical Foundation

1.1 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 introducted without distinction.

1.1.1 How to get expected value and variance?

X is a random variable whose values are $X_1, X_2, ..., X_n$. $P(X_1), P(X_2), ..., P(X_n)$ 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,

$$\begin{aligned} E(X) &= \sum_{i=1}^{n} X_i P(X_i) \text{ for discrete variable} \\ &= \int_X x f(x) dx \text{ for continous variable} \end{aligned}, \tag{1.1}$$

and

$$\begin{aligned} Var(X) &= \sum_{i=1}^{n} (X_i - E(X))^2 P(X_i) \text{ for discrete variable} \\ &= \int_X (x - E(X))^2 f(x) dx \text{ for continous variable} \end{aligned}. \tag{1.2}$$

For discrete variable,

$$\begin{aligned} Var(X) &= \sum_{i=1}^{n} (X_i - E(X))^2 P(X_i) \\ &= E[(X - E(X))^2] \\ &= E[X^2 + E(X)^2 - 2XE(X)] \\ &= E(X^2) + E(X)^2 - 2E(X)E(X) \\ &= E(X^2) - E(X)^2 \end{aligned}. \tag{1.3}$$

1.1.2 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 outcoems k, is

$$f(k; p) = \begin{cases} p & \text{if } k = 1, \\ q = 1 - p & \text{if } k = 0. \end{cases} \tag{1.4}$$

The expected value and invarance of a Bernoulli variable X are

$$\begin{cases} E(X) = P(X=1)\cdot 1 + P(X=0)\cdot 0 = p, \\ E(X^2) = P(X=1)\cdot 1^2 + P(X=0)\cdot 0^2 = P(X=1) = p, \\ Var(X) = E(X^2) - E(X)^2 = p - p^2 = p(1-p) = pq. \end{cases} \tag{1.5}$$

Note in Eq. 1.5, 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 ramdom 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 trails is given by the probability mass function:

$$f(k;n,p) = P(k;n,p) = P(X=k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{1.6}$$

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

$$\begin{cases} E(X) = E(X_1 + X_2 + \cdots X_n) \\ \qquad = E(X_1) + E(X_2) + \cdots + E(X_n) \\ \qquad = p + p + \cdots + p \\ \qquad = np, \\ Var(X) = Var(X_1) + Var(X_2) + \cdots + Var(X_n) \\ \qquad = nVar(X_1) \\ \qquad = np(1-p). \end{cases} \tag{1.7}$$

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 occure with a known constant rate λ and independently of the time since the last events. If X is a Poisson variable with the average number of events λ, we write $X \sim Possion(\lambda)$. The probability mass function is

$$f(k;n,\lambda) = P(X=k) = \frac{e^{-\lambda}\lambda^k}{k!}. \tag{1.8}$$

The expected value and invariance of a Poission variable X are

$$\begin{cases} E(X) = \sum_{i=0}^{\infty} iP(X=i) \\ \qquad = \sum_{i=1}^{\infty} i\frac{e^{-\lambda}\lambda^i}{i!} = \lambda e^{-\lambda}\sum_{i=1}^{\infty}\frac{\lambda^{i-1}}{(i-1)!} = \lambda e^{-\lambda}\sum_{i=0}^{\infty}\frac{\lambda^i}{i!} \\ \qquad = \lambda e^{-\lambda}e^{\lambda} \\ \qquad = \lambda \\ E(X^2) = \lambda + \lambda^2 \\ Var(X) = \lambda + \lambda^2 - \lambda^2 \\ \qquad = \lambda \end{cases} \tag{1.9}$$

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

Todo: Taylor's formula.

1.1.3 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

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b, \\ 0 & \text{for } x < a \text{ or } x > b. \end{cases} \tag{1.10}$$

The expected value and invariance of a Uniform variable X are

$$\begin{cases} E(X) = \int_a^b x\frac{1}{b-a} \\ \quad = \frac{x^2}{2(b-a)}\Big|_a^b = \frac{b^2-a^2}{2(b-a)} = \frac{(b+a)(b-a)}{2(b-a)} \\ \quad = \frac{b+a}{2} \\ E(X^2) = \int_a^b x^2\frac{1}{b-a} \\ \quad = \frac{x^3}{3(b-a)}\Big|_a^b = \frac{b^3-a^3}{3(b-a)} = \frac{(b-a)(b^2+a^2+ab)}{3(b-a)} \\ \quad = \frac{a^2+b^2+ab}{3} \\ Var(X) = E(X^2) - E(X)^2 = \frac{a^2+b^2+ab}{3} - (\frac{b+a}{2})^2 \\ \quad = \frac{4a^2+4b^2+4ab}{12} - \frac{3a^2+3b^2+6ab}{12} \\ \quad = \frac{(a-b)^2}{12} \end{cases} \tag{1.11}$$

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

$$f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1.12}$$

where μ is the expectation and σ is the standard deviation. If a random variable X is distributed normally with mean μ and variance σ^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

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \ge 0, \\ 0 & x < 0. \end{cases} \tag{1.13}$$

The expected value and invariance of a Exponential variable X are

$$\begin{cases} E(X) = \frac{1}{\lambda}, \\ Var(X) = \frac{1}{\lambda^2}. \end{cases} \tag{1.14}$$

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

Todo: conjugate distribution and Beta distribution.

1.1.4 Sample mean and sample variance

Sample mean is defined as

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i. \tag{1.15}$$

Sample Variance is defined as

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2. \tag{1.16}$$

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

proof:

$$\begin{aligned} E[\bar{X}] &= E[\frac{1}{n}\sum_{i=1}^{n} X_i] \\ &= \frac{1}{n}\sum_{i=1}^{n} E[X_i] \\ &= \frac{1}{n} nE(X) \\ &= E(X) \\ E[S^2] &= E[\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2] \\ &= \frac{1}{n-1}E[\sum_{i=1}^{n}(X_i^2 + \bar{X}^2 - 2X_i\bar{X})] \\ &= \frac{1}{n-1}(E[\sum_{i=1}^{n} X_i^2] + n\bar{X}^2 - 2n\bar{X}^2) \\ &= \frac{1}{n-1}(nE[X^2] - nE[\bar{X}^2]) \\ &= \frac{n}{n-1}(Var[X] + E[X]^2 - Var[\bar{X}] - E[\bar{X}]^2) \\ &= \frac{n}{n-1}(Var[X] - Var[\bar{X}]) \\ &= \frac{n}{n-1}(Var[X] - \frac{1}{n}Var[X]) \\ &= Var[X] \end{aligned} \tag{1.17}$$

Chapter 2

Machine Learning

2.1 Logistic regression (LR)

2.1.1 Form

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}. \tag{2.1}$$

The form of sigmoid function is

$$g(z) = \frac{1}{1+e^{-z}}. \tag{2.2}$$

The derivation of sigmoid function is

$$g'(z) = g(z)(1-g(z)). \tag{2.3}$$

Let us assume that

$$\begin{aligned} P(y=1|x;\theta) &= h_\theta(x), \\ P(y=0|x;\theta) &= 1-h_\theta(x). \end{aligned} \tag{2.4}$$

Above equation can be written more compactly as

$$p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}. \tag{2.5}$$

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

$$\begin{aligned} L(\theta) &= p(\vec{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x))^{y^{(i)}}(1-h_\theta(x))^{1-y^{(i)}}. \end{aligned} \tag{2.6}$$

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

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})). \end{aligned} \tag{2.7}$$

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

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

$$\frac{\partial l(\theta)}{\partial \theta_j} = (y-h_\theta(x))x_j. \tag{2.8}$$

Above, we get the stochastic gradient ascent rule:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(\overrightarrow{x^{(i)}}))x_j \tag{2.9}$$

2.2 Naive Bayes

2.2.1 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

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{2.10}$$

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.

2.2.2 Naive Bayesian Classifier (NBC)

the naive bayesian classifier can be expressed as

$$\begin{aligned} y = f(x) &= \arg\max_{c_k} P(Y = c_k|X = x) \\ &= \arg\max_{c_k} \frac{P(Y = c_k)\prod_j P(X^{(j)} = x^{(j)}|Y = c_k)}{\sum_k P(Y = c_k)\prod_j P(X^{(j)} = x^{(j)}|Y = c_k)}, \end{aligned} \tag{2.11}$$

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

Since the denominator is the same any c_k, the above equation 2.11 can be simplified as

$$y = \arg\max_{c_k} P(Y = c_k)\prod_j P(X^{(j)} = x^{(j)}|Y = c_k). \tag{2.12}$$

2.2.3 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

$$\begin{aligned} P(Y = c_k) &= \frac{\sum_{i=1}^N I(y_i = c_k)}{N} \\ P(X^{(j)} = a^{(j)}|Y = c_k) &= \frac{\sum_{i=1}^N I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^N I(y_i = c_k)}, \end{aligned} \tag{2.13}$$

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

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

$$P(y_1, y_2, \cdots, y_n; \theta_k) = \prod_{i=1}^N P(y_i) = \prod_{i=1}^N \theta_k^{N_k}, \tag{2.14}$$

then the likelihood function is

$$L(\theta^k; y_1, y_2, \cdots, y_n) = P(y_1, y_2, \cdots, y_n; \theta_k). \tag{2.15}$$

Next, the logarithmic operation is applied.

$$\begin{gathered} \ln(L(\theta_k)) = \sum_{i=1}^N N_k \ln\theta_k, \\ s.t., \sum_{k=1}^K \theta_k = 1 \end{gathered} \tag{2.16}$$

Using Lagrange Multiplier Method, we get

$$\ln(L(\theta_k, \lambda)) = \sum_{i=1}^{N} N_k \ln \theta_k + \lambda(\sum_{k=1}^{K} \theta_k - 1). \tag{2.17}$$

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

$$P(Y = c_k) = \frac{N_k}{N} = \frac{\sum_{i=1}^{N} I(y_i = c_k)}{N}. \tag{2.18}$$

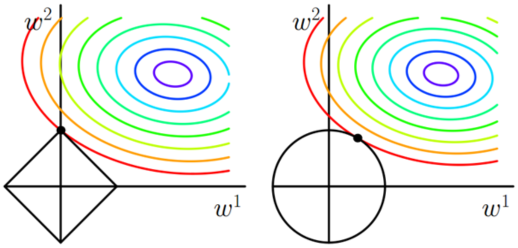
Moreover, the derivation of likelihood is similar.

2.3 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:

$$w* = \arg\min_{w} \sum_{i} L(y_i, f(x_i; w)) + \lambda\Omega(w) \tag{2.19}$$



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes."

Figure 2.1: L_1 norm and L_2 norm.

2.3.1 L0

$$L_0 = \sum_{i} \mathrm{I}(w_i \neq 0). \tag{2.20}$$

L_0 norm will make the weights sparse.

2.3.2 L1 (lasso regularization)

$$L_1 = \sum_{i} |w_i|. \tag{2.21}$$

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).

2.3.3 L2 (ridge regression or weight decay)

$$L_2 = \sum_i w_i^2 \tag{2.22}$$

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

2.4 Perceptron

2.4.1 Why can't perceptron solve XOR problem?

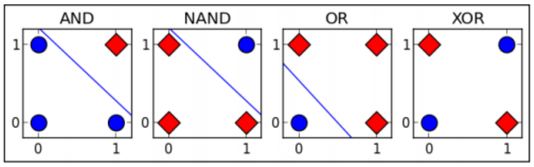


Figure 2.2: XOR problem.

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

2.4.2 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 $(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$, $y \in \mathcal{Y}$ represents the category of x. Then, we get the perceptron model from the input space to output space:

$$f(\vec{x}) = \mathrm{sign}(\vec{w}^{\mathrm{T}}\vec{x} + b), \tag{2.23}$$

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

$$\mathrm{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{2.24}$$

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

2.4.3 Learning Algorithm

Given a linear separable dataset $T = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$, where $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$, and $y_i \in \mathcal{Y} = +1, -1, i = 1, 2, \cdots, N$, 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:

$$\frac{1}{||\vec{w}||}|\vec{w}^{\mathrm{T}} \cdot \vec{x_0} + b| \tag{2.25}$$

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

As for missclassified points, $\vec{w}^{\mathrm{T}} \cdot \vec{x_0} + b > 0$ when $y_i = -1$, and $\vec{w}^{\mathrm{T}} \cdot \vec{x_0} + b < 0$ when $y_i = +1$. So the total distance from all the misscalssified points to hyperplane is

$$-\frac{1}{||w||}\sum_{\vec{x_i} \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b), \tag{2.26}$$

where M is the set of misclassfied points.

In Eq. 2.28, $\frac{1}{||w||}$ can be ignored. The reason is (a) $\frac{1}{||w||}$ doesn't affcet postive or negative judgment of $y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b)$ and (b) $\frac{1}{||w||}$ doesn't affcet the final optimization result of Eq. 2.28. 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 preceptron is

$$L(\vec{w}, b) = -\sum_{\vec{x_i} \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b), \tag{2.27}$$

and the optimization object is

$$\min_{\vec{w},b} L(\vec{w}, b) = \min_{\vec{w},b} -\sum_{\vec{x_i} \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b). \tag{2.28}$$

(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 $L(\vec{w}, b)$:

$$\begin{aligned} \nabla_{\vec{w}} L(\vec{w}, b) &= -\sum_{\vec{x_i} \in M} y_i \vec{x_i} \\ \nabla_b L(\vec{w}, b) &= -\sum_{\vec{x_i} \in M} y_i \end{aligned}. \tag{2.29}$$

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

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x} \\ b &\leftarrow b + \eta y_i \end{aligned}, \tag{2.30}$$

where η is the learning rate.

(3) The standard form of perceptron learing algorithm:

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

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

(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 traning set;

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

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x} \\ b &\leftarrow b + \eta y_i \end{aligned}; \tag{2.31}$$

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

(4)

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

2.4.4 Dual form of perceptron learning algorithm

From Eq. 2.30, $\vec{w}$ and b are updated for many times. Suppose the number of updates about each example is $n_i(i = 1, 2, \ldots N)$, and we define $\alpha_i = n_i \eta$. Then, the final learned $\vec{w}$ and b can be expressed as

$$\begin{aligned} \vec{w} &= \sum_{i=1}^{N} \alpha_i y_i \vec{x_i} \\ b &= \sum_{i=1}^{N} \alpha_i y_i \end{aligned}. \tag{2.32}$$

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

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

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

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

$$\begin{aligned} \vec{\alpha_i} &\leftarrow \alpha_i + \eta \\ b &\leftarrow b + \eta y_i \end{aligned}; \tag{2.33}$$

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

2.5 Support vector machine (SVM)

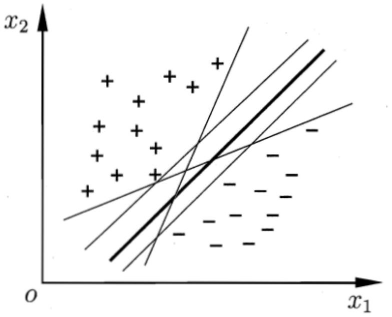


Figure 2.3: Classification hyperplane.

2.5.1 Form

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

$$\vec{w}^\top \vec{x} + b = 0. \tag{2.34}$$

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

$$\gamma = \frac{|\vec{w}^\top \vec{x} + b|}{|\vec{w}|}. \tag{2.35}$$

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

$$\gamma^{(i)} = \frac{y^{(i)}(\vec{w}^\top \overrightarrow{x^{(i)}} + b)}{|\vec{w}|}, \tag{2.36}$$

which is named geometric margin. Next, we have

$$\gamma = \min_{i=1,\cdots,m} \gamma^{(i)}. \tag{2.37}$$

Then, we give the form of SVM as

$$\begin{aligned} &\max_{\gamma,\vec{w},b} \gamma, \\ &\text{s.t. } \frac{y^{(i)}(\vec{w}^\top \overrightarrow{x^{(i)}} + b)}{|\vec{w}|} \geq \gamma \end{aligned} \tag{2.38}$$

Define functional margin as follows.

$$\begin{aligned} &\hat{\gamma}^{(i)} = y^{(i)}(\vec{w}^\top \overrightarrow{x^{(i)}} + b), \\ &\hat{\gamma} = \min_{i=1,\cdots,m} \hat{\gamma}. \end{aligned} \tag{2.39}$$

Then, the form of SVM 2.38 can be transformed.

$$\begin{aligned} &\max_{\gamma,\vec{w},b} \frac{\hat{\gamma}}{|\vec{w}|}, \\ &\text{s.t. } y^{(i)}(\vec{w}^{\top}\overrightarrow{x^{(i)}}+b) \geq \hat{\gamma}. \end{aligned} \tag{2.40}$$

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

$$\begin{aligned} &\max_{\gamma,\vec{w},b} \frac{1}{|\vec{w}|}, \\ &\text{s.t. } y^{(i)}(\vec{w}^{\top}\overrightarrow{x^{(i)}}+b) \geq 1. \end{aligned} \tag{2.41}$$

Moreovere, the form of SVM 2.41 can be transformed.

$$\begin{aligned} &\min_{\gamma,\vec{w},b} \frac{1}{2}|\vec{w}|^2, \\ &\text{s.t. } y^{(i)}(\vec{w}^{\top}\overrightarrow{x^{(i)}}+b) \geq 1. \end{aligned} \tag{2.42}$$

2.5.2 Lagrange duality

Consider a constrained optimization problems:

$$\begin{aligned} &\min_{w} f(\vec{w}), \\ &\text{s.t. } g_i(\vec{w}) \leq 0, i=1,\cdots,k, \\ &\quad\;\; h_i(\vec{w}) = 0, i=1,\cdots,l. \end{aligned} \tag{2.43}$$

We can define generalized lagrangian

$$\mathcal{L}(\vec{w},\vec{\alpha},\vec{\beta}) = f(\vec{w}) + \sum_{i=1}^{k}\alpha_i g_i(\vec{w}) + \sum_{i=1}^{l}\beta_i h_i(\vec{w}). \tag{2.44}$$

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

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

$$d^* = \max_{\vec{\alpha},\vec{\beta}:\alpha_i\geq 0}\min_{\vec{w}}\mathcal{L}(\vec{w},\vec{\alpha},\vec{\beta}) \leq \min_{\vec{w}}\max_{\vec{\alpha},\vec{\beta}:\alpha_i\geq 0}\mathcal{L}(\vec{w},\vec{\alpha},\vec{\beta}) = p^*. \tag{2.45}$$

Therefore, the soluation of primal problem can be transfored to solve the dual problem.

2.5.3 Solution of SVM

As shown in 2.46, the form of SVM is

$$\begin{aligned} &\min_{\gamma,\vec{w},b} \quad \frac{1}{2}|\vec{w}|^2, \\ &\text{s.t.} \quad y^{(i)}(\vec{w}^{\top}\overrightarrow{x^{(i)}}+b) \geq 1. \end{aligned} \tag{2.46}$$

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

$$\mathcal{L}(\vec{w},b,\vec{\alpha}) = \frac{1}{2}|\vec{w}|^2 + \sum_{i=1}^{m}\alpha_i[1-y^{(i)}(\vec{w}^{\top}\overrightarrow{x^{(i)}}+b)]. \tag{2.47}$$

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 α) by setting the derivatives of $\mathcal{L}$ with respect to $\vec{w}$ and b to zero. We have:

$$\begin{aligned}\frac{\partial \mathcal{L}(\vec{w}, b, \vec{\alpha})}{\vec{w}} &= \vec{w} - \sum_{i=1}^{m} \alpha_i y^{(i)} \overrightarrow{x^{(i)}} = 0, \vec{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \overrightarrow{x^{(i)}} \\ \frac{\partial \mathcal{L}(\vec{w}, b, \vec{\alpha})}{b} &= -\sum_{i=1}^{m} \alpha_i y^{(i)} = 0.\end{aligned} \tag{2.48}$$

Plug the $\vec{w}$ and b back into the Lagrangian Eq. 2.47, and simplify, we get

$$\mathcal{L}(\vec{w}, b, \vec{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\overrightarrow{x^{(i)}})^\top x^{(j)}. \tag{2.49}$$

Then, we obtain the following dual optimization problem:

$$\begin{aligned}\max_{\alpha} \quad & \mathcal{L}(\vec{w}, b, \vec{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (\overrightarrow{x^{(i)}})^\top x^{(j)} \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \cdots, m, \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0\end{aligned} \tag{2.50}$$

Sequential minimal optimization algorithm is used to efficiently solve the convex quadratic programming problem (as 2.48).

2.6 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 $\vec{w} = (w_1, w_2, \cdots, w_n)$ of the training model. Then, the total increment of $\mathcal{L}$ is

$$\begin{aligned}\Delta \mathcal{L} &= \frac{\partial \mathcal{L}}{\partial w_1} \Delta w_1 + \frac{\partial \mathcal{L}}{\partial w_2} \Delta w_2 + \cdots + \frac{\partial \mathcal{L}}{\partial w_n} \Delta w_n + o(\rho) \\ &\approx \frac{\partial \mathcal{L}}{\partial w_1} \Delta w_1 + \frac{\partial \mathcal{L}}{\partial w_2} \Delta w_2 + \cdots + \frac{\partial \mathcal{L}}{\partial w_n} \Delta w_n\end{aligned}. \tag{2.51}$$

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

$$\Delta \mathcal{L} \approx \nabla \mathcal{L}_{\vec{w}}^{\mathrm{T}} \cdot \Delta \vec{w}. \tag{2.52}$$

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

$$\Delta \vec{w} = -\eta \nabla \mathcal{L}_{\vec{w}}, \tag{2.53}$$

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

$$\vec{w} \leftarrow \vec{w} - \eta \nabla \mathcal{L}_{\vec{w}}. \tag{2.54}$$

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

2.7 Generative model and discriminative model

(1) Generative model:

(2) Discriminative model: perceptron

Todo: Random forest

Chapter 3

Deep Network

3.1 How does backpropagation work?

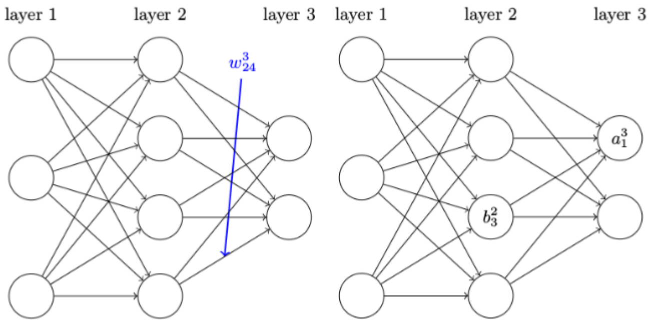


Figure 3.1: Neural network.

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

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

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

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

The understanding of backpropagation is about how to chang the weights and biases in a network to change the cost function. Utimately, 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 intermedia quantity δ^l_j, which we call the error of the jth neuroon in the lth layer. In the actual calculation process, backpropagation will give us a predure to compute δ^l_j, and then will relate δ^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 jth neuroon in the lth layer by

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}. \tag{3.3}$$

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w^l}} = \overrightarrow{\delta^l} (\overrightarrow{a^{l-1}})^{\mathrm{T}}. \tag{3.6}$$

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

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

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

$$\frac{\partial \mathcal{L}}{\partial \overrightarrow{b^l}} = \overrightarrow{\delta^l}. \tag{3.7}$$

3.2 Backpropagation of Convolutional Neural Network

3.2.1 Backpropagation of Fully Connectional Layer

The backprogapation equations of element form are as follows:

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

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

$$\begin{cases} \overrightarrow{\delta^L} = \nabla_{\overrightarrow{a^L}} \mathcal{L} \otimes \sigma'(\overrightarrow{z^L}), \\ \overrightarrow{\delta^l} = ((\overrightarrow{w^{l+1}})^{\mathrm{T}} \overrightarrow{\delta^{l+1}}) \otimes \sigma'(\overrightarrow{z^l}), \\ \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w^l}} = \overrightarrow{\delta^l} (\overrightarrow{a^{l-1}})^{\mathrm{T}}, \\ \dfrac{\partial \mathcal{L}}{\partial \overrightarrow{b^l}} = \overrightarrow{\delta^l}. \end{cases} \tag{3.8}$$

3.2.2 Backpropagation of Convolutional Layer

We use w_{pq}^l to denote the weight of the element in the pth row and qth column of the convolutional kernel in lth layer. Moreover, a_{ij}^l and z_{ij}^l as the activation and weighted input in ith row and jth column in lth layer, respectively. Then,

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

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

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

Similar to fully connectional layer, we define δ_{ij}^l as the error in element at ith row and jth column in l layer. In detail,

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

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

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

In this case, a example can be found:

$$\begin{aligned}\delta^{l}_{1,1} =& \delta^{l+1}_{-1,-1} w^{l+1}_{3,3} + \delta^{l+1}_{-1,0} w^{l+1}_{3,2} + \delta^{l+1}_{-1,1} w^{l+1}_{3,1} + \\ & \delta^{l+1}_{0,-1} w^{l+1}_{2,3} + \delta^{l+1}_{0,0} w^{l+1}_{2,2} + \delta^{l+1}_{0,1} w^{l+1}_{2,1} + \\ & \delta^{l+1}_{1,-1} w^{l+1}_{1,3} + \delta^{l+1}_{1,0} w^{l+1}_{1,2} + \delta^{l+1}_{1,1} w^{l+1}_{1,1},\end{aligned} \tag{3.12}$$

which can be represented as convolutional operations like:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \delta^{l+1}_{1,1} & \delta^{l+1}_{1,2} & 0 & 0 \\ 0 & 0 & \delta^{l+1}_{2,1} & \delta^{l+1}_{2,2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} w^{l+1}_{3,3} & w^{l+1}_{3,2} & w^{l+1}_{3,1} \\ w^{l+1}_{2,3} & w^{l+1}_{2,2} & w^{l+1}_{2,1} \\ w^{l+1}_{1,3} & w^{l+1}_{1,2} & w^{l+1}_{1,1} \end{bmatrix}. \tag{3.13}$$

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

$$\boldsymbol{\delta^{l}} = \boldsymbol{\delta^{l+1}} * \mathrm{rot180}(\boldsymbol{w}) \otimes \sigma'(\boldsymbol{z^{l}}). \tag{3.14}$$

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

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w^{l}_{pq}} &= \sum_{i}\sum_{j} \frac{\partial \mathcal{L}}{\partial z^{l}_{ij}} \frac{\partial z^{l}_{ij}}{\partial w^{l}_{pq}} \\ &= \sum_{i}\sum_{j} \frac{\partial \mathcal{L}}{\partial z^{l}_{ij}} \frac{\partial \sum_{p=1}^{r}\sum_{q=1}^{c} a^{l-1}_{i+p-1,j+q-1} w^{l}_{pq} + b^{l}}{\partial w^{l}_{pq}} \\ &= \sum_{i}\sum_{j} \delta^{l}_{ij} a^{l-1}_{i+p-1,j+q-1}.\end{aligned} \tag{CONV-BP3}$$

As for bias, we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b^{l}} &= \sum_{i}\sum_{j} \frac{\partial \mathcal{L}}{\partial z^{l}_{ij}} \frac{\partial z^{l}_{ij}}{\partial b^{l}} \\ &= \sum_{i}\sum_{j} \frac{\partial \mathcal{L}}{\partial z^{l}_{ij}} \frac{\partial \sum_{p=1}^{r}\sum_{q=1}^{c} a^{l-1}_{i+p-1,j+q-1} w^{l}_{pq} + b^{l}}{\partial b^{l}} \\ &= \sum_{i}\sum_{j} \delta^{l}_{ij}.\end{aligned} \tag{CONV-BP4}$$

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

$$\begin{cases} \delta^{l}_{ij} = \sum_{m}\sum_{n} \delta^{l+1}_{mn} w^{l+1}_{i+1-m,j+1-n} \sigma'(z^{l}_{ij}), \\ \dfrac{\partial \mathcal{L}}{\partial w^{l}_{pq}} = \sum_{i}\sum_{j} \delta^{l}_{ij} a^{l-1}_{i+p-1,j+q-1}, \\ \dfrac{\partial \mathcal{L}}{\partial b^{l}} = \sum_{i}\sum_{j} \delta^{l}_{ij}. \end{cases} \tag{CONV-BP}$$

Moreover, the corresponding matrix form is as follows:

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

3.3 Why does batch normalization work?

3.3.1 Dataset shift and covariate shift

There exists a 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 distrbutions of training data and testing data are not exactly the same, especiallt when the training data is insufficient. This phenomenon is named as "dataset shift".

In analysis of covariance (ANOVA), convate 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 normalizatio), 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 marinal probabilities are different, $i.e.$

$$\begin{array}{c} P_{tr}(y|x) = P_{te}(y|x) \\ P_{tr}(x) \neq p_{te}(y) \end{array}, \tag{3.16}$$

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:

$$\begin{array}{c} P_{tr}(x,y) = P_{tr}(y|x)P_{tr}(x) \\ P_{te}(x,y) = P_{te}(y|x)P_{te}(x) \end{array}, \tag{3.17}$$

and we know the joint probabilities of traning data and testing data are different, $i.e.$,

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

. 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.

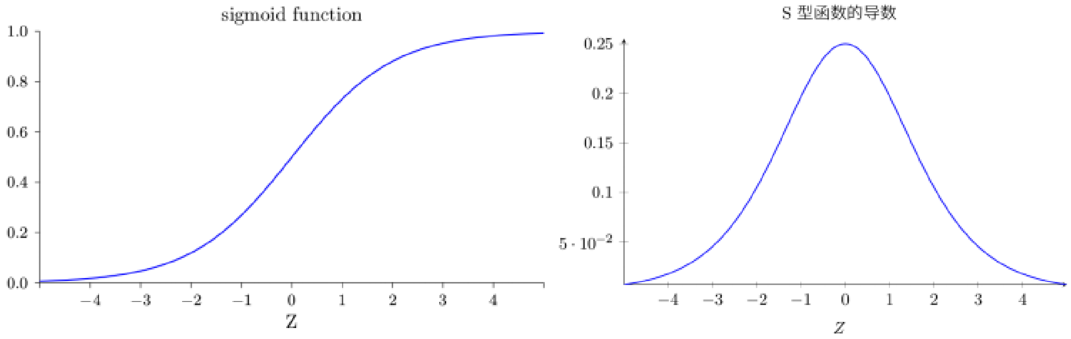


Figure 3.2: Derivative of sigmoid function.

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 staturated 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 nomalization 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 jountly, 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 $x = [x^{(1)}, x^{(2)}, \cdots, x^{(k)}]$, each dimension is normalized as follows

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

(2)Simply normalizeing each input of a layer may reduce what the layer can represent. For example, nomalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. In this way, batchnorm turns the multi-layer nonlinear neural network into a multi-layer linear neural network, which is equivalent to the representation ability of a single layer linear network. The reduces the representation ability of the network. So, introduce $\gamma^{(k)}, \beta^{(k)}$ to scale and shift the normalized value:

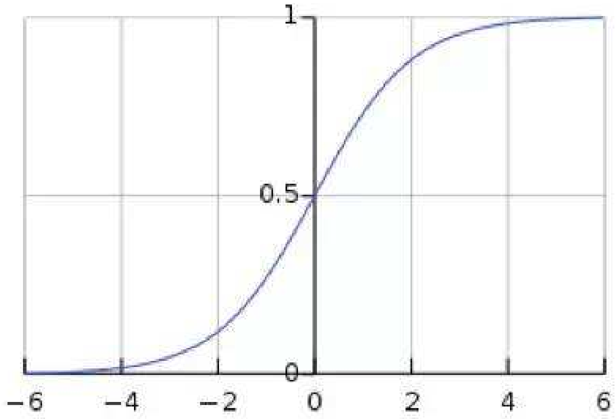


Figure 3.3: Sigmoid function.

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

which guarantee the representation ability of the network.

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

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

Algorithm 1 Forward algorithm of batch normalization.

```
Input: values of $x$ over a mini-batch: $\mathcal{B} = \{x_1, x_2, \cdots, x_m\}$, parameters to be learned: $\gamma, \beta$
Output: $y_i = BN_{\gamma,\beta}(x_i)$

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

In this manner, we know

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

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 ϵ. 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 thansform 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 γ 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}})(-\frac{1}{2})(\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, μ and σ^2 are computed by each mini-batch $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ by their unbiased estimations as proved in 1.17:

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

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?

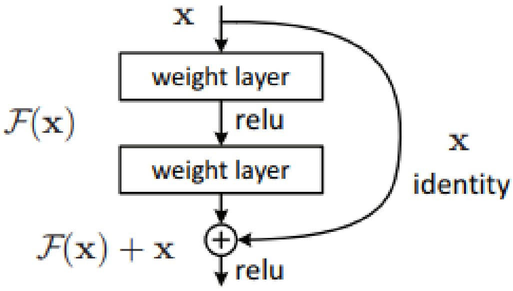

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.4: Residual Learning.

In the process of optimization in deep networ 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 $\mathcal{H}(\mathrm{x})$. From above experience, we assume x and $\mathcal{H}(\mathrm{x})$ are close. Thus, the function of these layers is mapping from x to $\mathcal{H}(\mathrm{x})$. But usually this process is difficult becuase the relative gap between x and $\mathcal{H}(\mathrm{x})$ is small. Toward this end, we construct the residual as $\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$ 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 indentity, 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).

Chapter 4

Contents specifically referenced

- Sect. 2.4.1: 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.