Mathematics, Machine Learning and Deep Learning Notes

Jinming Su

Last update: April 22, 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,

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

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 are

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

> 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

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

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

> 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

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

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

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

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: Beta distribution.

1.2 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

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{1.15}$$

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 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.

Chapter 2

Machine Learning

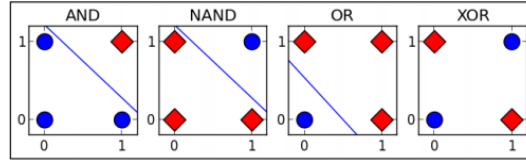## 2.1 Why can't perceptron solve XOR problem?

Figure 2.1: 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.

2.1.1 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:

$$f(\vec{x}) = \mathrm{sign}(\vec{w}^{\mathrm{T}}\vec{x} + b), \tag{2.1}$$

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

$$\mathrm{sign}(x) = \left\{ \begin{array}{ll} +1, & x \geq 0 \\ -1, & x < 0 \end{array} \right. \tag{2.2}$$

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

2.1.2 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:

$$\frac{1}{||\vec{w}||}|\vec{w}^{\mathrm{T}} \cdot \vec{x_0} + b| \tag{2.3}$$

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

$$-\frac{1}{||w||} \sum_{\vec{x_i} \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b), \tag{2.4}$$

where M is the set of misclassfied points.

In Eq. 2.6, $\frac{1}{||w||}$ 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. 2.6. 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

$$L(\vec{w}, b) = - \sum_{\vec{x}_i \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x}_i + b), \tag{2.5}$$

and the optimization object is

$$\min_{\vec{w}, b} L(\vec{w}, b) = \min_{\vec{w}, b} - \sum_{\vec{x}_i \in M} y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x}_i + b). \tag{2.6}$$

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

$$\begin{aligned} \nabla_{\vec{w}} L(\vec{w}, b) = - \sum_{\vec{x}_i \in M} y_i \vec{x}_i \\ \nabla_b L(\vec{w}, b) = - \sum_{\vec{x}_i \in M} y_i \end{aligned} \tag{2.7}$$

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

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x} \\ b &\leftarrow b + \eta y_i \end{aligned}, \tag{2.8}$$

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 $\eta(0 < \eta \le 1)$.

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 $y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x}_i + b) \le 0$:

$$\begin{aligned} \vec{w} &\leftarrow \vec{w} + \eta y_i \vec{x} \\ b &\leftarrow b + \eta y_i \end{aligned}; \tag{2.9}$$

(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.

2.1.3 Dual form of perceptron learning algorithm

From Eq. 2.8, $\vec{w}$ 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

$$\begin{aligned} \vec{w} &= \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i \\ b &= \sum_{i=1}^{N} \alpha_i y_i \end{aligned}. \tag{2.10}$$

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 $\eta(0 < \eta \le 1)$.

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

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

(2.11)

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

2.2 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

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

(2.12)

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,

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

(2.13)

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

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

(2.14)

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

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

(2.15)

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

2.3 Generative model and discriminative model

(1) Generative model: perceptron

2.4 Support vector machine

Todo: SVM

Todo: Random forest

Chapter 3

Deep Network
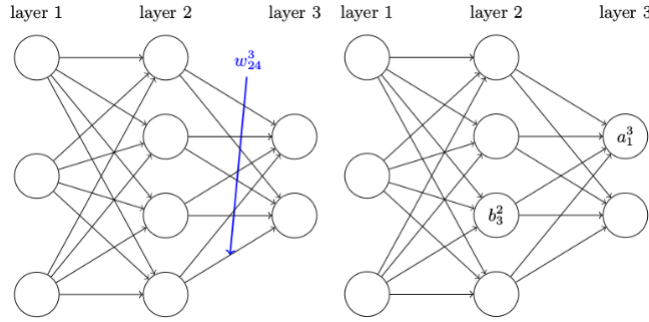
3.1 How does backpropagation work?

Figure 3.1: Neural network.

First, symbol definition is carried out. We use w_{jk}^l 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_j^l for the bias of jth neuron in lth layer and we use a_j^l for the activation of the jth neuron in lth layer. We also use z_j^l as the <u>weighted input</u> of jth neuron in lth layer. Then

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l,$$
$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{3.1}$$
$$= \sigma(z_j^l).$$

where σ 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 lth layer. $\vec{z^l}$ is the same. Moreover, we use $\boldsymbol{w^l}$ for the <u>weight matrix</u> and $\vec{b^l}$ for the bias vector in lth layer. Then,

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

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 $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_j^l}$. To compute those, we first introduce an intermedia quantity δ_j^l, which we call the <u>error</u> of the jth neuroon in the lth layer. In the actual calculation process, backpropagation will give us a predure to compute δ_j^l, and then will relate δ_j^l to $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_j^l}$.

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

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

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

$$\frac{\partial \mathcal{L}}{\partial \overrightarrow{b^l}} = \overrightarrow{\delta^l}. \tag{3.7}$$
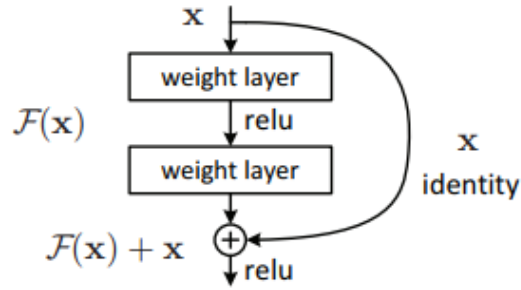
Figure 3.2: Residual Learning.

3.2   Why does residual learning work?

In the process of optimization in deep network, 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 becuase 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 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

- the chapter of (2)Perceptron in [1];

Bibliography

[1] Hang Li. Statistical learning method. <u>Tsinghua University Press</u>, 2012.