
A Guide to Data Science



From mathematics to algorithms.

作者: Jinxiong & Zhang

时间: May 4, 2019

邮箱: jinxiongzhang@qq.com

Version: 3.00

目 录



1	Abstract	1
2	Statistical Foundation	2
2.1	Basic Probability	2
2.1.1	Random Variable	3
2.1.2	Discrete Distribution	3
2.1.3	Continuous Distribution	4
2.1.4	Bivariate Distribution	5
2.2	Representation of Random Variable	6
2.2.1	Mixture Representation	7
2.3	Bayes' Theorem	7
2.3.1	Conditional Probability	7
2.3.2	Bayes's Formula and Inverse Bayes' Formula	9
2.4	What determines a distribution?	11
2.4.1	Expectation, Variance and Entropy	11
2.4.2	Moment and Moment Generating Function	14
2.5	Sampling Methods	15
2.5.1	Sampling from Discrete Distribution	15
2.5.2	Sampling from Continuous Distribution	16
3	Numerical Optimization	18
3.1	Gradient Descent and More	18
3.2	Mirror Gradient Method	20
3.2.1	Projected Gradient Descent	20
3.2.2	Mirror Descent	20
3.3	Variable Metric Methods	21
3.3.1	Newton's Method	21
3.3.2	The Fisher Scoring Algorithm	21
3.3.3	Quasi-Newton Methods	22
3.3.4	Natural Gradient Descent	22
3.4	Expectation Maximization Algorithm	23
3.4.1	Generalized EM Algorithm	23

3.5	Alternating Direction Method of Multipliers	24
3.6	Stochastic Gradient Descent	25
4	Artificial Neural Network	29
4.1	Perceptron	31
4.1.1	Activation function	31
4.2	Feed-forward Neural Networks	32
4.2.1	Representation of Feedforward Neural Network	32
4.2.2	Evaluation and Optimization in Multilayer Perceptron	33
4.2.3	Evaluation for different tasks	34
4.3	Backpropagation, Training and Regularization	36
4.3.1	Backpropagation	36
4.3.2	Training Methods	40
4.3.3	Regularization	42
4.4	Bibliography	44
	参考文献	44





第 1 章 Abstract



This is a brief introduction to data science. **Data science** can be seen as an extension of statistics.

- All sciences are, in the abstract, mathematics.
- All judgments are, in their rationale, statistics. by C.P. Rao

On Chomsky and the Two Cultures of Statistical Learning or Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author) provides more discussion of statistical model or data science.



"The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work—that is, correctly to describe phenomena from a reasonably wide area."

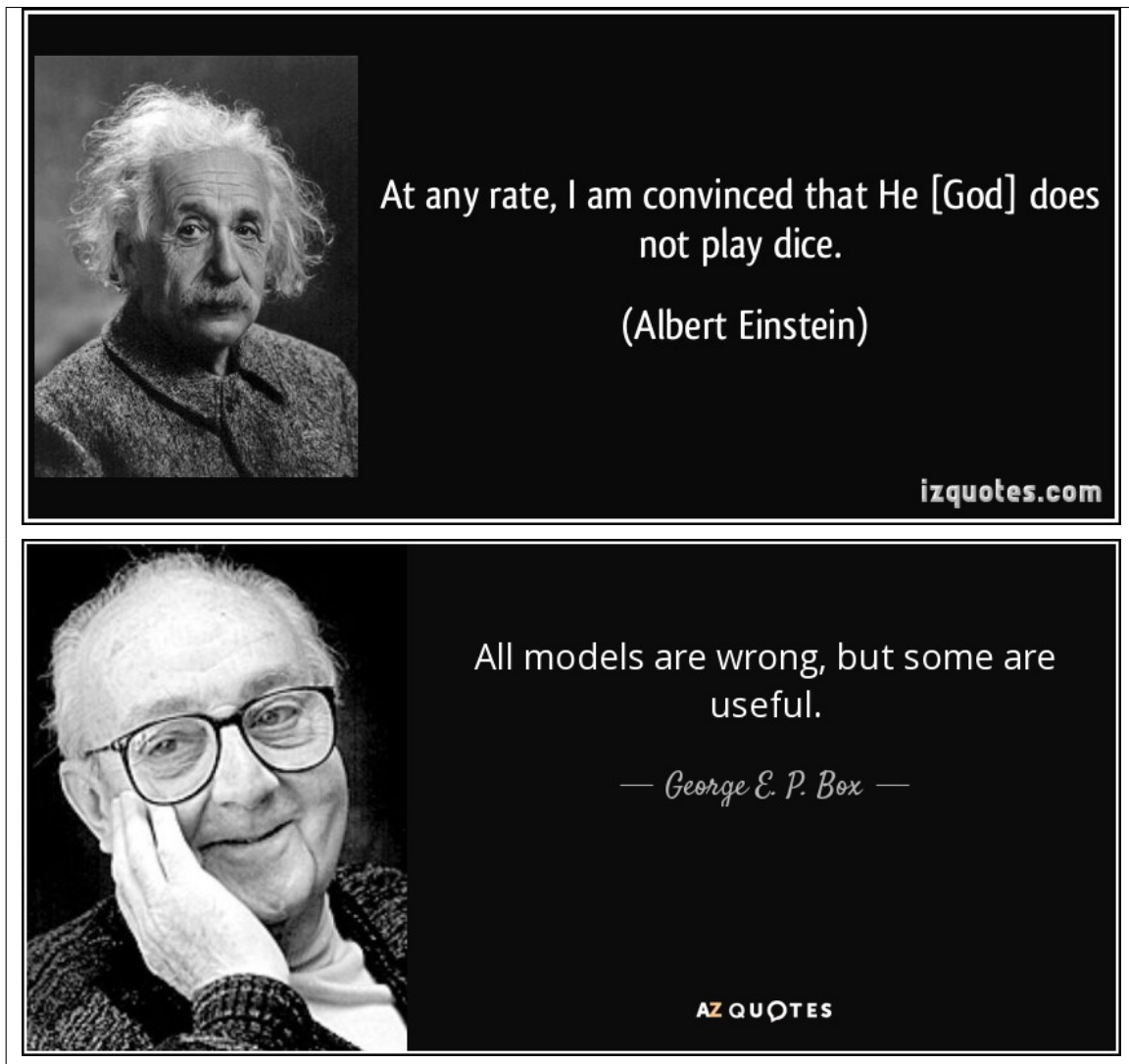
John von Neumann

第 2 章 Statistical Foundation



"Statistics, in a nutshell, is a discipline that studies the best ways of dealing with randomness, or more precisely and broadly, variation", Professor Xiao-Li Meng said.

表 2.1: On mathematical model



2.1 Basic Probability

Probability theory is regarded as the theoretical foundation of statistics, which provides the ideal models to measure the stochastic phenomenon, randomness, chance or luck. It is the distribution theory bridging probability and statistics.

The project [Seeing theory](#) is a brief introduction to statistics and probability theory and provides a [.pdf file](#).

2.1.1 Random Variable

Definition 2.1: Random Variable

A random variable is a mapping

$$X : \Omega \rightarrow \mathbb{R}$$

that assigns a real number $X(\omega)$ to each outcome ω .

- Ω is the sample space.
- ω in Ω are called sample outcomes or realization.
- Subsets of Ω are called Event.



Definition 2.2: Probability

A function P is called probability function with respect to the known sample space Ω and all the event $\forall \omega \in \Omega$ if

Properties	Conditions
Nonnegative	$P(\omega) \geq 0 \forall \omega \in \Omega$
Normalized	$P(\Omega) = 1$
Additive	If $\omega_1, \omega_2, \dots \in \Omega$ are disjoint, then $P(\bigcup_{i=1}^{\infty} \omega_i) = \sum_{i=1}^{\infty} P(\omega_i)$



These properties are called [Kolmogorov axioms](#).

2.1.2 Discrete Distribution

Random variable or event is countable or listed, then we can list the probability of every event. If so, we can compute the probability of a random variable less than the given value. For example, $P(X \leq 10) = \sum_{i=1}^{10} P(X = i)$ if the random variable X only takes positive integers.

Definition 2.3: Probability mass function

The probability mass function of a discrete random variable X is defined as

$$f_X(x) = P_X(X = x)$$

for all x .



The left subindex X is to point out that the probability is with respect to the random variable X and it can omit if no confusion.



2.1.3 Continuous Distribution

If random variables may take on a continuous range of values, it is hard or valueless to compute the probability of a given value. It is usually to find the cumulative distribution function.

Definition 2.4: Cumulative Distribution Function

A function $F(x)$ is called cumulative distribution function (CDF in short) if

- $\lim_{x \rightarrow -\infty} F(x) = 0$ and $\lim_{x \rightarrow \infty} F(x) = 1$;
- The function $F(x)$ is monotone increasing function with x ;
- $\lim_{x \rightarrow x_0^+} F(x) = F(x_0)$ for all x_0 .



In probability, $\forall x, F(x) = P(X \leq x)$ for some random variable X .

For example, if the random variable X has the cumulative distribution function (CDF)

$$F_X(x) = \begin{cases} x, & x \in [0, 1] \\ 1, & x \in (1, \infty), \\ 0, & \text{otherwise} \end{cases}$$

we say the random variable X is uniformly distributed in $[0, 1]$. We can see [this link](#).

The probability density function can be seen as the counterpart of pmf. For the continuous random variable X , its CDF $F_X(x)$ is not required to be differentiable. But if so, we can compute the derivative of the CDF $F_X(x)$: $\frac{dF_X(x)}{dx} = f_X(x)$.

Definition 2.5: Probability Density Function

We call the function $f_X(x)$ is the probability density function with respect to the continuous random variable X if

$$F_X(x) = \int_{-\infty}^x f_X(t) dt$$

for all x , where $F_X(x)$ is the probability cumulative function of the random variable X . ♣

For example, the random variable X uniformly distributed in $[0, 1]$ has the probability density function

$$f_X(x) = \begin{cases} 1, & x \in [0, 1] \\ 0, & \text{otherwise} \end{cases}.$$

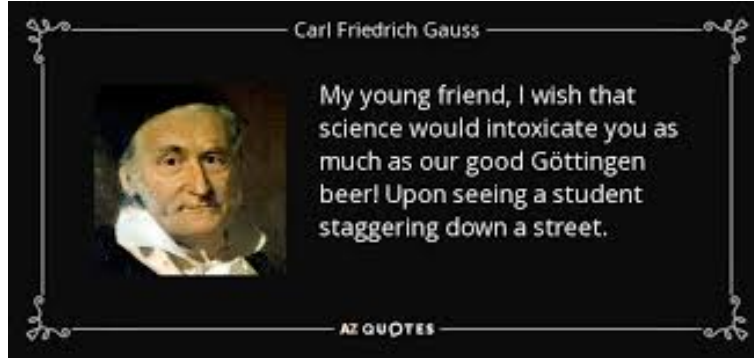
If the pdf f_X is positive in the set S , we call the set S is the support set or support of the distribution.

One common probability - **normal or Gaussian distribution** - is often given in the pdf:

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



where σ^2 is given positive and finite, μ is given and finite. And its support is $(-\infty, \infty)$.



The density function of a power law distribution is in the form of

$$f_X(x) = x^{-\alpha}, x \in [1, \infty)$$

where α is real. It is also called **Pareto-type distribution**.

The power law can be used to describe a phenomenon where a small number of items is clustered at the top of a distribution (or at the bottom), taking up 95% of the resources. In other words, it implies a small amount of occurrences is common, while larger occurrences are rare.

Every probability distribution is an ideal mathematical model that describes some real event. Here is [a field guide to continuous distribution](<http://threeplusone.com/FieldGuide.pdf>).

2.1.4 Bivariate Distribution

The cumulative probability function and probability density function are really functions with some required properties. We wonder the bivariate functions in probability theory.

Definition 2.6: Joint Distribution Function

The joint distribution function $F : \mathbb{R}^2 \rightarrow [0, 1]$, where X and Y are discrete variables, is given by

$$F_{(X,Y)}(x, y) = P(X \leq x, Y \leq y).$$



Their joint mass function $f : \mathbb{R}^2 \rightarrow [0, 1]$ is given by

$$f_{(X,Y)}(x, y) = P(X = x, Y = y).$$

Definition 2.7: Joint Density Function

The joint distribution function $F_{(X,Y)} : \mathbb{R}^2 \rightarrow [0, 1]$, where X and Y are continuous variables, is given by $F_{(X,Y)}(x, y) = P(X \leq x, Y \leq y)$. And their joint density function if $f_{X,Y}(x, y)$ satisfies that

$$F_{(X,Y)}(x, y) = \int_{v=-\infty}^y \int_{u=-\infty}^x f_{(X,Y)}(u, v) du dv$$



for each $x, y \in \mathbb{R}$.



Definition 2.8: Marginal Distribution

The marginal distributed of X and Y are

$$F_X(x) = P(X \leq x) = F_{(X,Y)}(x, \infty) \quad \text{and} \quad F_Y(y) = P(Y \leq y) = F_{(X,Y)}(\infty, y)$$

for discrete random variables;

$$F_X(x) = \int_{-\infty}^x \left(\int_{\mathbb{R}} f_{(X,Y)}(u, y) dy \right) du \quad \text{and} \quad F_Y(y) = \int_{-\infty}^y \left(\int_{\mathbb{R}} f_{(X,Y)}(x, v) dx \right) dv$$

for continuous random variables and the marginal probability density function is

$$f_X(x) = \int_{\mathbb{R}} f_{(X,Y)}(x, y) dy, \quad f_Y(y) = \int_{\mathbb{R}} f_{(X,Y)}(x, y) dx.$$



Two random variables X and Y are called as identically distributed if $P(x \in A) = P(y \in A)$ for any $A \subset \mathbb{R}$.

2.2 Representation of Random Variable

The random variable is nearly no sense without its probability distribution. We can choose the CDF or pdf to depict the probability, which depends on the case.

We know that a function of random variable is also a random variable. For example, supposing that X is a random variable, the function $g(X) = e^X$ is a random variable as well as $g(X) = \ln(X)$.

The function of random variable must have its distribution. We can take it to generate more distributions.

Theorem 2.1: Uniform Transformation

Suppose that X is with the cumulative distribution function CDF $F_X(x)$ and $Y = F_X(X)$, then Y is *uniformly* distributed in the interval $[0, 1]$.



Let X and Y be two different random variables, $Z = X + Y$ or $Z = X \cdot Y$ are typical functions of random variables, especially X is discrete while Y is continuous.



2.2.1 Mixture Representation

Let P_1, P_2, \dots, P_n be probability distribution and $\sum_{i=1}^n \lambda_i = 1, \lambda_i > 0 \quad \forall i \in \{1, 2, \dots, n\}$, we can get

$$P = \sum_{i=1}^n \lambda_i P_i$$

is also a probability distribution. If X is distributed in P , i.e. $X \sim P$, its probability is computed as

$$P(X = x) = \sum_{i=1}^n \lambda_i P_i(X = x)$$

for discrete random variable or

$$P(X \leq x) = \sum_{i=1}^n \lambda_i P_i(X \leq x)$$

for continuous random variable.

A random variable X is said to have a mixture distribution if the distribution of X depends on a quantity that also has a distribution.

Sometimes, it is quite difficult if we directly generate a random vector $X \sim f_X(x)$, but the augmented vector $(X, Z) \sim f_{(X,Z)}(x, z)$ is relatively easy to generate such as [Box-Muller algorithm]. And we can represent $f_X(x)$ as the marginal probability distribution of $f_{(X,Y)}(x, y)$ in integral form

$$\int_{\mathbb{R}} f_{(X,Z)}(x, z) dz.$$

Thanks to Professor Tian Guoliang(Gary) in SUSTech, who brought me to computational statistics. More information on probability can be founded in [The Probability web](#).

2.3 Bayes' Theorem

Bayes theorem is the foundation of Bayesian statistics in honor of the statistician [Thomas Bayes](#).

2.3.1 Conditional Probability

If the set A is the subset of B , i.e. $A \subset B$, we know that if $x \in A, x \in B$. We can say if A happened, the event B must have happened. However, if B happened, what is probability when A happened? It should be larger than the probability of A when B did not happened and it is also larger than the probability of A when the events including B happened. We know that $A \cap B = A$ when A is the subset of B .



Definition 2.9: Conditional Probability

Supposing $A, B \in \Omega$ and $P(B) > 0$, the conditional probability of A given B (denoted as $P(A|B)$) is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$



The vertical bar $|$ means that the right one is given or known and is parameter rather than variable.

Here is a [visual explanation](#).

Definition 2.10: Statistical Independence

If $P(A \cap B) = P(A)P(B)$, we call the event A and B are statistically independent.



From the definition of conditional probability, it is obvious that:

- $P(B) = \frac{P(A \cap B)}{P(A|B)}$;
- $P(A \cap B) = P(A|B)P(B)$;
- $P(A \cap B) = P(B|A)P(A)$.

Thus

- $P(A|B)P(B) = P(B|A)P(A)$;
- $P(A|B) = \frac{P(B|A)P(A)}{P(B)} = P(B|A) \frac{P(A)}{P(B)}$;
- $P(B) = \frac{P(B|A)P(A)}{P(A|B)} = \frac{P(B|A)}{P(A|B)} P(A)$.

Definition 2.11: Conditional Distribution Function

The conditional distribution function of Y given $X = x$ is the function $F_{Y|X}(y|x)$ given by

$$F_{Y|X}(y|x) = \int_{-\infty}^x \frac{f_{(X,Y)}(x, v)}{f_X(x)} dv = \frac{\int_{-\infty}^x f_{(X,Y)}(x, v) dv}{f_X(x)}$$

for the support of $f_X(x)$ and the conditional probability density function of $F_{Y|X}$, written $f_{Y|X}(y|x)$, is given by

$$f_{Y|X}(y|x) = \frac{f_{(X,Y)}(x, y)}{f_X(x)}$$

for any x such that $f_X(x) > 0$.

**Definition 2.12: Conditional Probability for Continuous Random Variable**

If X and Y are non-generate and jointly continuous random variables with density



$f_{X,Y}(x, y)$ then, if B has positive measure,

$$P(X \in A | Y \in B) = \frac{\int_{y \in B} \int_{x \in A} f_{X,Y}(x, y) xy}{\int_{y \in B} \int_{x \in \mathbb{R}} f_{X,Y}(x, y) xy}.$$



Definition 2.13: Chain Rule of Probability

$$P(x^{(1)}, \dots, x^{(n)}) = P(x^{(1)}) \prod_{i=2}^n P(x^{(i)} | x^{(1)}, \dots, x^{(i-1)})$$



Definition 2.14: Conditional independence

The event A and B are **conditionally independent** given C if and only if

$$P(A \cap B | C) = P(A | C)P(B | C).$$



Theorem 2.2: Total Probability Theorem

Supposing the events A_1, A_2, \dots are disjoint in the sample space Ω and $\bigcup_{i=1}^{\infty} A_i = \Omega$, B is any subset of Ω , we have

$$P(B) = \sum_{i=1}^{\infty} P(B \cap A_i) = \sum_{i=1}^{\infty} P(B | A_i)P(A_i).$$



Note:

- For the discrete random variable, the total probability theorem tells that any event can be decomposed to the basic event in the event space.
- For the continuous random variable, this theorems means that

$$\int_{\mathbb{B}} f_X(x) dx = \sum_{i=1}^{\infty} \int_{\mathbb{B} \cap A_i} f_X(x) x.$$

2.3.2 Bayes's Formula and Inverse Bayes' Formula

Theorem 2.3: Bayes's Formula

Supposing the events A_1, A_2, \dots are disjoint in the sample space Ω and $\bigcup_{i=1}^{\infty} A_i = \Omega$, B is any subset of Ω , we have

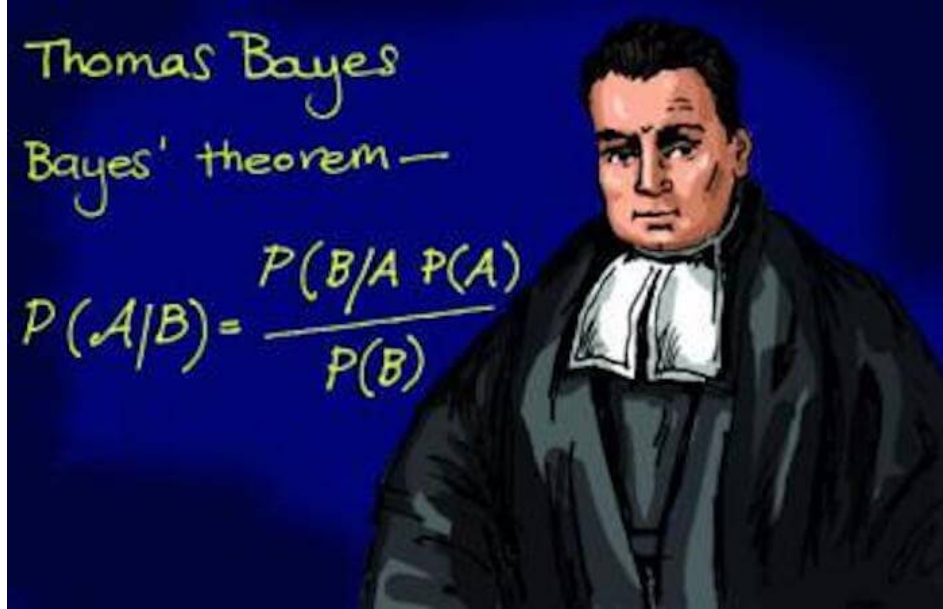
$$P(A_i | B) = \frac{P(B | A_i)P(A_i)}{\sum_{i=1}^{\infty} P(B | A_i)P(A_i)} = \frac{P(B | A_i)P(A_i)}{P(B)}.$$





Note:

- The probability $P(A_i)$ is called prior probability of event A_i and the conditional probability $P(A_i|B)$ is called posterior probability of the event A_i in Bayesian statistics.
- For any A_i , $P(A_i|B) \propto P(B|A_i)P(A_i)$ and $P(B)$ is the normalization constant as the sum of $P(B|A_i)P(A_i)$.



The joint pdf of the random variables X and Y can be expressed as

$$f_{(x,y)}(x, y) = f_{X|Y}(x|y)f_Y(y) = f_{Y|X}(y|x)f_X(x).$$

in some proper conditions. Thus we get by division

$$f_Y(y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_{X|Y}(x|y)}. \quad (1)$$

Integrating this identity with respect to y on support of $f_Y(y)$, we immediately have the **point-wise formula** as shown below

$$f_X(x) = \left\{ \int_{f_{X|Y}(x|y) \neq 0} \frac{f_{Y|X}(y|x)}{f_{X|Y}(x|y)} dy \right\}^{-1}. \quad (2)$$

Now substitute (2) into (1), we obtain the dual form of IBF for $f_Y(y)$ and hence by symmetry we obtain the **function-wise formula** of $f_Y(y)$ at y_0 as shown in (f1), or the sampling formula in (f2) when the normalizing constant is omitted.

$$f_X(x) = \left\{ \int_{f_{Y|X}(y_0|x) \neq 0} \frac{f_{X|Y}(x|y_0)}{f_{Y|X}(y_0|x)} dx \right\}^{-1} \frac{f_{X|Y}(x|y_0)}{f_{Y|X}(y_0|x)} \quad (f1)$$

$$\propto \frac{f_{X|Y}(x|y_0)}{f_{Y|X}(y_0|x)} \quad (f2)$$

There are more information on **inverse Bayes' formula**:



表 2.2: The Inventor of IBF



- <http://web.hku.hk/~kaing/Background.pdf>
- http://101.96.10.64/web.hku.hk/~kaing/Section1_3.pdf
- <http://web.hku.hk/~kaing/HKSSinterview.pdf>
- <http://web.hku.hk/~kaing/HKStalk.pdf>

2.4 What determines a distribution?

The cumulative density function (CDF) or probability density function(pdf) is roughly equal to random variable, i.e. one random variable is always attached with CDF or pdf. However, what we observed is not the variable itself but its realization or sample.

- In theory, what properties do some CDFs or pdfs hold? For example, are they all integrable?
- In practice, the basic question is how to determine the CDF or pdf if we only observed some samples?

2.4.1 Expectation, Variance and Entropy

Expectation and variance are two important factors that characterize the random variable. The expectation of a discrete random variable is defined as the weighted sum.

Definition 2.15: Mean

For a discrete random variable X , of which the range is x_1, \dots, x_∞ , its mean $\mathbb{E}(X)$ is defined as $\mathbb{E}(X) = \sum_{i=1}^{\infty} x_i P(X = x_i)$.




The mean of a discrete random variable is a weighted sum. Average is one special expectation with equiprob, i.e. $P(X = x_1) = P(X = x_2) = \dots = P(X = x_i) = \dots \forall i$.

The expectation of a continuous random variable is defined as one special integration.



Definition 2.16: Expectation

For a continuous random variable X with the pdf $f(x)$, if the integration $\int_{-\infty}^{\infty} |x|f(x)dx$ exists, the value $\int_{-\infty}^{\infty} xf(x)dx$ is called the (mathematical) **expectation** of X , which is often denoted as $\mathbb{E}(X)$. 

Note that **NOT** all random variables have expectation. For example, the expectation of standard **Cauchy distribution**

$$f_X(x) = \frac{1}{\pi(1+x^2)}$$

is undefined.

The expectation is the center of the probability density function in many cases. What is more,


$$\arg \max_b \mathbb{E}(X - b)^2 = \mathbb{E}(X)$$

which implies that $\mathbb{E}(X)$ is the most likeliest to appear in expectation.


Definition 2.17: Expectation of Random vector

Now suppose that X is a random vector $X = (X_1, \dots, X_d)$, where $X_j, j = 1, \dots, d$, is real-valued. Then $\mathbb{E}(X)$ is simply the vector

$$(\mathbb{E}(X_1), \dots, \mathbb{E}(X_d))$$

where $\mathbb{E}(X_i) = \int_{\mathbb{R}} x_i f_{X_i} dx_i \forall i \in \{1, 2, \dots, d\}$ and $f_{X_i}(x_i)$ is the marginal probability density function of X_i . 

Definition 2.18: Variance

If the random variable has finite expectation $\mathbb{E}(X)$, then the expectation of $[X - \mathbb{E}(X)]^2$ is called the variance of X (denoted as $Var(X)$), i.e. $\sum_{i=1}^{\infty} [x_i - \mathbb{E}(X)]^2 P(X = x_i)$ for discrete random variable and $\int_{-\infty}^{\infty} [x - \mathbb{E}(X)]^2 f_X(x) dx$ for continuous random variable. 

It is obvious that

$$Var(X) = \int_{\mathbb{R}} [x - \mathbb{E}(X)]^2 f_X(x) dx = \int_{\mathbb{X}} [x - \mathbb{E}(X)]^2 f_X(x) dx$$

where the set \mathbb{X} is the support of X .

In general, let X be a random variable with pdf $f_X(x)$, the expectation of the random variable $g(X)$ is equal to the integration $\int_{\mathbb{R}} g(x) f_X(x) dx$ if it exists.

In analysis, the expectation of a random variable is the integration of some functions.



Definition 2.19: Conditional Expectation

The conditional expectation of Y given X can be defined as

$$\Psi(X) = \mathbb{E}(Y|X) = \int_{\mathbb{R}} y f_{Y|X}(y|x) dy.$$



It is a **parameter-dependent integral**, where the parameter x can be considered as fixed constant. See **the content in Calculus II**.

Definition 2.20: Tower Rule

Tower Rule Let X be random variable on a sample space Ω , let the events A_1, A_2, \dots are disjoint in the sample space Ω and $\bigcup_{i=1}^{\infty} A_i = \Omega$, B is any subset of Ω , $\mathbb{E}(X) = \sum_{i=1}^{\infty} \mathbb{E}(X|A_i)P(A_i)$. In general, the conditional expectation $\Psi(X) = \mathbb{E}(Y|X)$ satisfies $\mathbb{E}(\Psi(X)) = \mathbb{E}(Y)$.



The probability of an event can be expressed as the expectation, i.e.

表 2.3: Probability as Integration

$$P(x \in A) = \int_{\mathbb{R}} \mathbb{I}_A f_X(x) dx$$

where

$$\mathbb{I}_A = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise} \end{cases}. \quad (2.1)$$

It is called as the indicator function of A .

Definition 2.21: Entropy

The Shannon entropy is defined as

$$H = - \sum_{i=1}^{\infty} P(X = x_i) \ln P(X = x_i)$$

for the discrete random variable X with the range $\{x_1, \dots, x_n\}$ with the convention $P(X = x) = 0$ that $P(X = x) \ln \frac{1}{P(X=x)} = 0$ and

$$H = \int_{\mathbb{X}} \ln\left(\frac{1}{f_X(x)}\right) f_X(x) dx$$

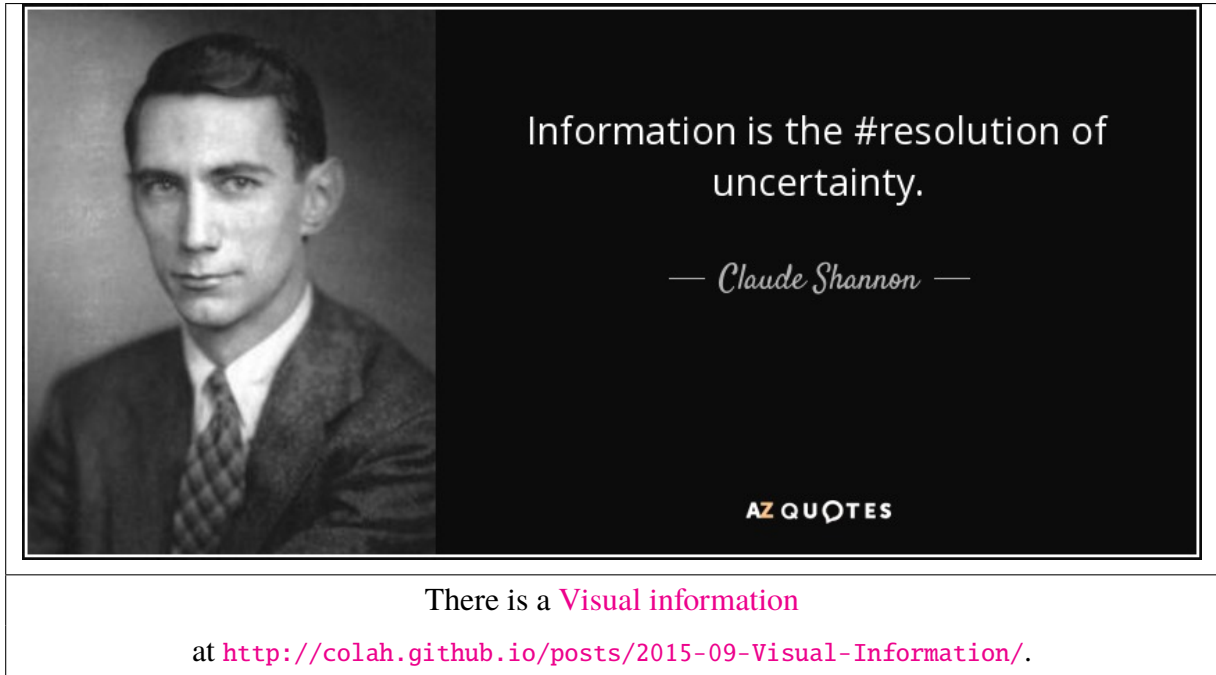
for the continuous random variable X with the support \mathbb{X} .



The Shannon entropy is often related to **finite** state discrete random variable, i.e. $n < \infty$ and the value of random variable is not involved in its entropy.



表 2.4: Claude Elwood Shannon, 1916-2001



2.4.2 Moment and Moment Generating Function

In calculus, we know that some proper functions can be extended as a Taylor series in a interval.

Moment is a specific quantitative measure of the shape of a function.

Definition 2.22: Moments

The n -th moment of a real-valued continuous function $f_X(x)$ of a real variable about a value c is

$$\mu_n = \int_{x \in \mathbb{R}} (x - c)^n f_X(x) dx.$$



And the constant c always take the expectation $\mathbb{E}(X)$ or 0.

Definition 2.23: Moment Generating Function

Moment generating function of a random variable X is the expectation of the random variable of e^{tX} , i.e.

$$M_X(t) = \mathbb{E}(e^{tX})$$



表 2.5: Moment generating function

$M_X(t) = \int_{\mathbb{R}} e^{tx} f_X(x) dx$ for continuous random variable
$M_X(t) = \sum_{i=1}^{\infty} e^{tx_i} P(x = x_i)$ for discrete random variable



It is **Laplace transformation** applied to probability density function. And it is also related with cumulants <http://scholarpedia.org/article/Cumulants>.

The moments are not unique, i.e. the different distribution may have the same moments such as

$$f_1(x) = \frac{1}{\sqrt{2\pi x}} \exp(-\log(x)^2/2)$$

$$f_2(x) = f_1(x)(1 + \sin(2\pi \log x)).$$

2.5 Sampling Methods

2.5.1 Sampling from Discrete Distribution

Definition 2.24: Inverse transform technique

Let F be a probability cumulative function of a random variable taking non-negative integer values, and let U be uniformly distributed on interval $[0, 1]$. The random variable given by $X = k$ if and only if $F(k - 1) < U < F(k)$ has the distribution function F . ♣

Sampling from Categorical Distribution

The *categorical distribution* (also called a *generalized Bernoulli distribution*, *multinoulli distribution*) is a discrete probability distribution that describes the possible results of a random variable that can take on one of K possible categories, with the probability of each category separately specified.

The category can be represented as **one-hot vector**, i.e. the vector $(1, 0, \dots, 0)$ is referred as the first category and the others are as similar as it. This representation is of some advantages:

1. Each category is identified by its position and equal to each other in norm;
2. The one-hot vectors can not be compared as the real number in value;
3. The probability mass function of the K categories distribution can be written in a compact form - $P(X) = [p_1, p_2, \dots, p_K] \cdot X^T$, where
 - The probability $p_i \geq 0, \forall i \in [1, \dots, K]$ and $\sum_{i=1}^K p_i = 1$;
 - The random variable X is one-hot vector and X^T is the transpose of X .

Sampling it by **inverse transform sampling**:

1. Pick a **uniformly distributed** number between 0 and 1.
2. Locate the greatest number in the CDF whose value is less than or equal to the number just chosen., i.e. $F^{-1}(u) = \inf\{x : F(x) \leq u\}$.
3. Return the category corresponding to this CDF value.

See more on **Categorical distribution**.

The categorical variable cannot be ordered, how to compute the CDF?



2.5.2 Sampling from Continuous Distribution

Direct Methods

Inverse transform technique

Theorem 2.4: Inverse transform technique

Let F be a probability cumulative function of a continuous random variable, and let U be uniformly distributed on interval $[0, 1]$. The random variable $X = F^{-1}(U)$ has the distribution function F . ♥

Khinchine's (1938) theorem: Suppose the random variable X with density given by

$$f_X(x) = \int_x^\infty z^{-1} f_Z(z) dz$$

or This mixture can be represented equivalently by

$$Z \sim f_Z(z), z > 0 \quad \text{and} \quad (1)$$

$$X|(Z = z) \sim \text{Unif}(0, z). \quad (2)$$

Hence $\frac{X}{Z}|(Z = z) = \frac{X}{z}|(Z = z) \sim \text{Unif}(0, 1)$ not depending on z , so that

$$\frac{X}{Z}|(Z = z) = \frac{X}{z}|(Z = z) \sim \text{Unif}(0, 1) \quad (2.2)$$

$$\frac{X}{Z} \stackrel{d}{=} U \sim \text{Unif}(0, 1) \quad (2.3)$$

$$X = ZU \quad (3)$$

and U and Z are mutually independent.

If $\nabla f_X(x)$ exists and $f_X(\infty) = 0$, we can obtain the following equation by Newton-Leibniz's theorem

$$f_X(x) = - \int_x^\infty \nabla f_X(z) dz$$

and comparing (4) and (0), it is obvious: $\nabla f_X(z) = -z^{-1} f_Z(z)$.

See more information on [On Khinchine's Theorem and Its Place in Random Variate Generation](#) and [Reciprocal symmetry, unimodality and Khinchine's theorem](#).

Rejection sampling

The **rejection sampling** is to draw samples from a distribution with the help of a proposal distribution.

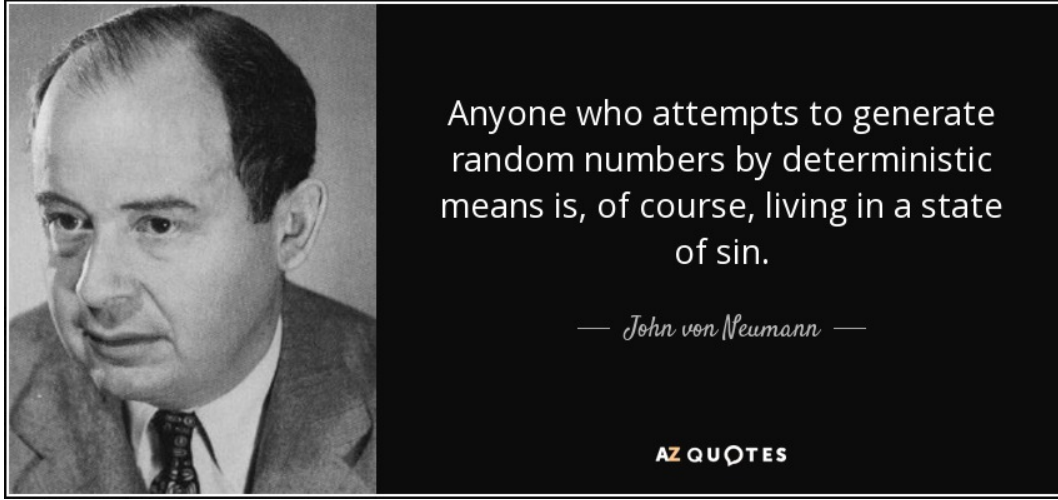
1. Obtain a sample y from distribution Y and a sample u from $\text{Unif}(0, 1)$ (the uniform distribution over the unit interval). 2. Check whether or not $u < f(y)/Mg(y)$.

- If this holds, accept y as a sample drawn from f ;



- if not, reject the value of y and return to the sampling step.

The algorithm will take an average of M iterations to obtain a sample. **Ziggural algorithm** is an application of rejection sampling to draw samples from Gaussian distribution.



Sampling-importance-resampling

It is also called SIR method in short. As name shown, it consists of two steps: sampling and importance sampling. The SIR method generates an approximate i.i.d. sample of size m from the target density $f(x)$, $x \in \mathbb{X} \subset \mathbb{R}^n$.

The SIR without replacement is as following:

1. Draw $X^{(1)}, X^{(2)}, \dots, X^{(J)}$ independently from the proposal density $g(\cdot)$.
2. Select a subset $\{X^{(k_i)}\}_{i=1}^m$ from $\{X^{(i)}\}_{i=1}^J$ via resampling without replacement from the discrete distribution $\{\omega_i\}_{i=1}^J$ where $w_i = \frac{f(X^{(i)})}{g(X^{(i)})}$ and $\omega_i = \frac{w_i}{\sum_{i=1}^J w_i}$.

The Conditional Sampling Method

The conditional sampling method due to the prominent Rosenblatt transformation is particularly available when the joint distribution of a d vector is very difficult to generate but one marginal distribution and $d - 1$ univariate conditional distributions are easy to simulate.

It is based on the chain rule of probability:

$$f(x) = f(x_d) \prod_{i=1}^{d-1} f_k(x_k | x_{k+1}, x_{k+2}, \dots, x_d)$$

where $x = (x_1, x_2, \dots, x_d)^T$ and $f(x)$ is the probability density function.

To generate X from $f(x)$, we only need to generate x_d from the marginal density $f_d(x_d)$, then to generate x_k sequentially from the conditional density $f_k(x_k | x_{k+1}, x_{k+2}, \dots, x_d)$.



第 3 章 Numerical Optimization



IN *A Few Useful Things to Know about Machine Learning*, Pedro Domingos put up a relation:

LEARNING = REPRESENTATION + EVALUATION + OPTIMIZATION.

- Representation as the core of the note is the general (mathematical) **model** that computer can handle.
- Evaluation is **criteria**. An evaluation function (also called objective function, cost function or scoring function) is needed to distinguish good classifiers from bad ones.
- Optimization is to aimed to find the parameters that optimizes the evaluation function, i.e.

$$\arg \min_{\theta} f(\theta) = \{\theta^* | f(\theta^*) = \min f(\theta)\} \text{ or } \arg \max_{\theta} f(\theta) = \{\theta^* | f(\theta^*) = \max f(\theta)\}.$$

The objective function to be minimized is also called cost function.

Evaluation is always attached with optimization; the evaluation which cannot be optimized is not a good evaluation in machine learning.

3.1 Gradient Descent and More

Each iteration of a line search method computes a search direction p^k and then decides how far to move along that direction. The iteration is given by

$$x^{k+1} = x^k + \alpha_k p^k$$

where the positive scalar α^k is called the step length. The success of a line search method depends on effective choices of both the direction p^k and the step length α_k .

Note: we use the notation x^k and α_k to represent the k th iteration of the vector variables x and k th step length, respectively. Most line search algorithms require p_k to be a descent direction — one for which $\langle p^k, \nabla f_k \rangle < 0$ — because this property guarantees that the function f can be reduced along this direction, where ∇f_k is the gradient of objective function f at the k th iteration point x_k i.e. $\nabla f_k = \nabla f(x^k)$.

Gradient descent and its variants are to find the local solution of the unconstrained optimization problem:

$$\min f(x)$$

where $x \in \mathbb{R}^n$.

Its iterative procedure is:

$$x^{k+1} = x^k - \alpha_k \nabla_x f(x^k)$$

where x^k is the k th iterative result, $\alpha_k \in \{\alpha | f(x^{k+1}) < f(x^k)\}$ and particularly $\alpha_k = \arg \min_{\alpha} \{f(x^k - \alpha \nabla_x f(x^k))\}$.

Some variants of gradient descent methods are not line search method. For example, the **heavy ball method**:

$$x^{k+1} = x^k - \alpha_k \nabla_x f(x^k) + \rho_k (x^k - x^{k-1})$$

where the momentum coefficient $\rho_k \in [0, 1]$ generally and the step length α_k cannot be determined by line search.

Nesterov accelerated gradient method is defined by

$$x^k = y^k - \alpha^{k+1} \nabla_x f(y^k) \quad \text{Descent} \quad (3.1)$$

$$y^{k+1} = x^k + \rho^k (x^k - x^{k-1}) \quad \text{Momentum} \quad (3.2)$$

where the momentum coefficient $\rho_k \in [0, 1]$ generally.

表 3.1: Inventor of Nesterov accelerated Gradient



3.2 Mirror Gradient Method

It is often called **mirror descent**. It can be regarded as non-Euclidean generalization of **projected gradient descent** to solve some constrained optimization problems.

3.2.1 Projected Gradient Descent

Projected gradient descent is aimed to solve convex optimization problem with explicit constraints, i.e.

$$\arg \min_{x \in \mathbb{S}} f(x)$$

where $\mathbb{S} \subset \mathbb{R}^n$. It has two steps:

$$z^{k+1} = x^k - \alpha_k \nabla_x f(x^k) \quad \text{Gradient descent} \quad (3.3)$$

$$x^{k+1} = \text{Proj}_{\mathbb{S}}(z^{k+1}) = \arg \min_{x \in \mathbb{S}} \|x - z^{k+1}\|^2 \quad \text{Projection} \quad (3.4)$$

3.2.2 Mirror Descent

Mirror descent can be regarded as the non-Euclidean generalization via replacing the ℓ_2 norm or Euclidean distance in projected gradient descent by **Bregman divergence**.

Bregman divergence is induced by convex smooth function f :

$$B(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle$$

where $\langle \cdot, \cdot \rangle$ is inner product. Especially, when f is quadratic function, the Bregman divergence induced by f is

$$B(x, y) = x^2 - y^2 - \langle 2y, x - y \rangle = x^2 + y^2 - 2xy = (x - y)^2$$

i.e. the Euclidean distance. A wonderful introduction to Bregman divergence is **Meet the Bregman Divergences** by **Mark Reid** at <http://mark.reid.name/blog/meet-the-bregman-divergences.html>. It is given by:

$$z^{k+1} = x^k - \alpha_k \nabla_x f(x^k) \quad \text{Gradient descent} \quad (3.5)$$

$$x^{k+1} = \arg \min_{x \in \mathbb{S}} B(x, z^{k+1}) \quad \text{Bregman projection} \quad (3.6)$$

One special method is called **entropic mirror descent** when $f = e^x$ and \mathbb{S} is simplex. See more on the following link list.



3.3 Variable Metric Methods

3.3.1 Newton's Method

NEWTON'S METHOD and QUASI-NEWTON METHODS are classified to variable metric methods.

It is also to find the solution of unconstrained optimization problems, i.e.

$$\min f(x)$$

where $x \in \mathbb{R}^n$.

Newton's method is given by

$$x^{k+1} = x^k - \alpha^{k+1} H^{-1}(x^k) \nabla_x f(x^k)$$

where $H^{-1}(x^k)$ is inverse of the Hessian matrix of the function $f(x)$ at the point x^k . It is called **Newton–Raphson algorithm** in statistics. Especially when the log-likelihood function $\ell(\theta)$ is well-behaved, a natural candidate for finding the MLE is the Newton–Raphson algorithm with quadratic convergence rate.

3.3.2 The Fisher Scoring Algorithm

In maximum likelihood estimation, the objective function is the log-likelihood function, i.e.

$$\ell(\theta) = \sum_{i=1}^n \log P(x_i|\theta)$$

where $P(x_i|\theta)$ is the probability of realization $X_i = x_i$ with the unknown parameter θ . However, when the sample random variable $\{X_i\}_{i=1}^n$ are not observed or realized, it is best to replace negative Hessian matrix (i.e. $-\frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T}$) of the likelihood function with the **observed information matrix**:

$$J(\theta) = \mathbb{E}\left(-\frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T}\right) = - \int \frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T} f(x_1, \dots, x_n|\theta) dx_1 \cdots dx_n$$

where $f(x_1, \dots, x_n|\theta)$ is the joint probability density function of X_1, \dots, X_n with unknown parameter θ .

And the **Fisher scoring algorithm** is given by

$$\theta^{k+1} = \theta^k + \alpha_k J^{-1}(\theta^k) \nabla_{\theta} \ell(\theta^k)$$

where $J^{-1}(\theta^k)$ is the inverse of observed information matrix at the point θ^k .

See <http://www.stats.ox.ac.uk/~steffen/teaching/bs2HT9/scoring.pdf> or <https://wiseodd.github.io/techblog/2018/03/11/fisher-information/> for more information.

Fisher scoring algorithm is regarded as an example of **Natural Gradient Descent** in information geometry such as <https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/>.



3.3.3 Quasi-Newton Methods

Quasi-Newton methods, like steepest descent, require only the gradient of the objective function to be supplied at each iterate. By measuring the changes in gradients, they construct a model of the objective function that is good enough to produce superlinear convergence. The improvement over steepest descent is dramatic, especially on difficult problems. Moreover, since second derivatives are not required, quasi-Newton methods are sometimes more efficient than Newton's method.

In optimization, quasi-Newton methods (a special case of variable-metric methods) are algorithms for finding local maxima and minima of functions. Quasi-Newton methods are based on Newton's method to find the stationary point of a function, where the gradient is 0. In quasi-Newton methods the Hessian matrix does not need to be computed. The Hessian is updated by analyzing successive gradient vectors instead. Quasi-Newton methods are a generalization of the secant method to find the root of the first derivative for multidimensional problems. In multiple dimensions the secant equation is under-determined, and quasi-Newton methods differ in how they constrain the solution, typically by adding a simple low-rank update to the current estimate of the Hessian. One of the chief advantages of quasi-Newton methods over Newton's method is that the Hessian matrix (or, in the case of quasi-Newton methods, its approximation) B does not need to be inverted. The Hessian approximation B is chosen to satisfy

$$\nabla f(x^{k+1}) = \nabla f(x^k) + B(x^{k+1} - x^k),$$

which is called the **secant equation** (the Taylor series of the gradient itself). In more than one dimension B is underdetermined. In one dimension, solving for B and applying the Newton's step with the updated value is equivalent to the **secant method**. The various quasi-Newton methods differ in their choice of the solution to the *secant equation* (in one dimension, all the variants are equivalent).

3.3.4 Natural Gradient Descent

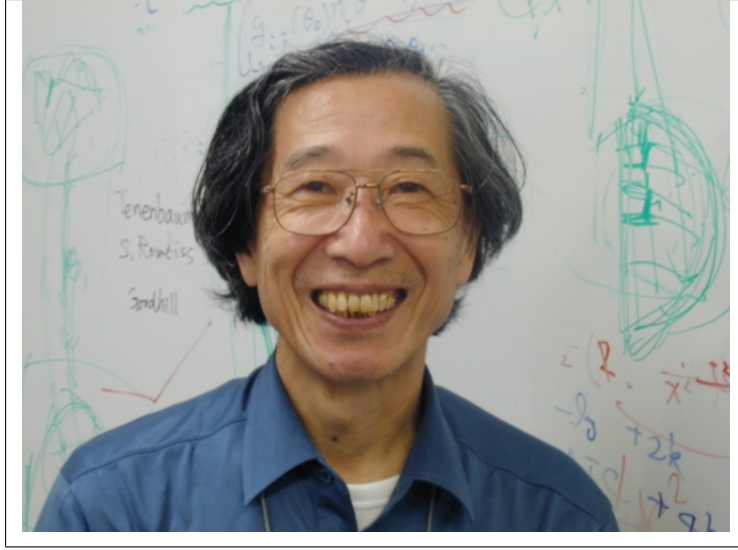
Natural gradient descent is to solve the optimization problem $\min_{\theta} L(\theta)$ by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_{(t)} F^{-1}(\theta^{(t)}) \nabla_{\theta} L(\theta^{(t)})$$

where $F^{-1}(\theta^{(t)})$ is the inverse of Riemann metric at the point $\theta^{(t)}$. And Fisher scoring algorithm is a typical application of Natural Gradient Descent to statistics. **Natural gradient descent** for manifolds corresponding to exponential families can be implemented as a first-order method through *mirror descent* at <https://www.stat.wisc.edu/~raskutti/publication/MirrorDescent.pdf>.



表 3.2: Originator of Information Geometry: Shun-ichi Amari(甘利俊一)



3.4 Expectation Maximization Algorithm

Expectation-Maximization algorithm, popularly known as the EM algorithm has become a standard piece in the statistician's repertoire. It is used in incomplete-data problems or latent-variable problems such as Gaussian mixture model in maximum likelihood estimation. The basic principle behind the EM is that instead of performing a complicated optimization, one augments the observed data with latent data to perform a series of simple optimizations.

Let $\ell(\theta|Y_{obs}) \triangleq \log L(\theta|Y_{obs})$ denote the log-likelihood function of observed datum Y_{obs} . We augment the observed data Y_{obs} with latent variables Z so that both the complete-data log-likelihood $\ell(\theta|Y_{obs}, Z)$ and the conditional predictive distribution $f(z|Y_{obs}, \theta)$ are available. Each iteration of the EM algorithm consists of an expectation step (E-step) and a maximization step (M-step). Specifically, let $\theta^{(t)}$ be the current best guess at the MLE $\hat{\theta}$. The E-step is to compute the **Q** function defined by

$$Q(\theta|\theta^{(t)}) = \mathbb{E}(\ell(\theta|Y_{obs}, Z)|Y_{obs}, \theta^{(t)}) \quad (3.7)$$

$$= \int_Z \ell(\theta|Y_{obs}, Z) \times f(z|Y_{obs}, \theta^{(t)}) dz, \quad (3.8)$$

and the M-step is to maximize **Q** with respect to θ to obtain

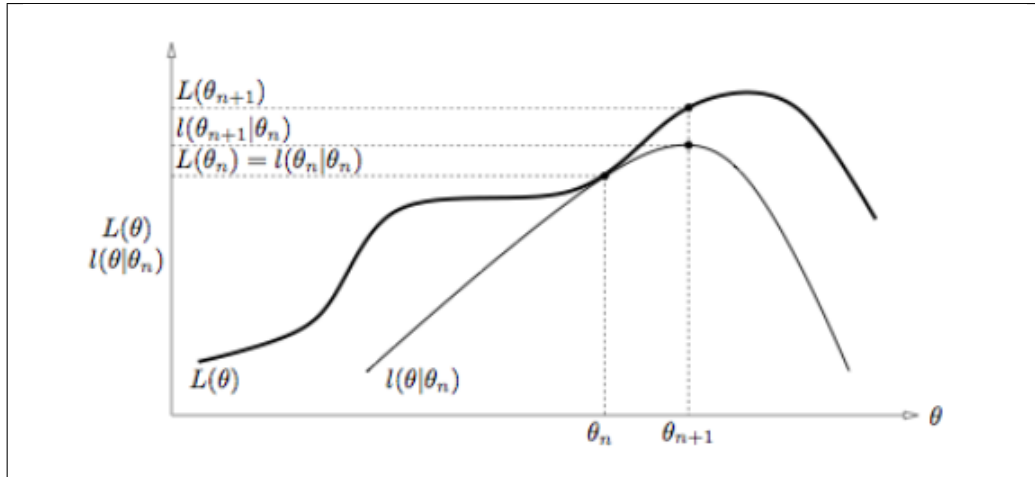
$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^{(t)}).$$

3.4.1 Generalized EM Algorithm

Each iteration of the **generalized EM** algorithm consists of an expectation step (E-step) and an ascent step instead of maximization step (M-step). Specifically, let $\theta^{(t)}$ be the current



表 3.3: Diagram of EM algorithm



best guess at the MLE $\hat{\theta}$. The E-step is to compute the **Q** function defined by

$$Q(\theta|\theta^{(t)}) = \mathbb{E}(\ell(\theta|Y_{obs}, Z)|Y_{obs}, \theta^{(t)}) \quad (3.9)$$

$$= \int_Z \ell(\theta|Y_{obs}, Z) \times f(z|Y_{obs}, \theta^{(t)}) dz, \quad (3.10)$$

and the another step is to find θ that satisfies $Q(\theta^{t+1}|\theta^t) > Q(\theta^t|\theta^t)$, i.e.

$$\theta^{(t+1)} \in \{\hat{\theta} | Q(\hat{\theta}|\theta^{(t)}) \geq Q(\theta|\theta^{(t)})\}.$$

It is not to maximize the conditional expectation.

See more on the book **The EM Algorithm and Extensions, 2nd Edition** by **Geoffrey McLachlan , Thriyambakam Krishna** at <https://www.wiley.com/en-cn/The+EM+Algorithm+and+Extensions,+2nd+Edition-p-9780471201700>.

See more on <https://www.stat.berkeley.edu/~aldous/Colloq/lange-talk.pdf>

3.5 Alternating Direction Method of Multipliers

Alternating direction method of multipliers is called **ADMM** shortly. It is aimed to solve the following convex optimization problem:

$$\min F(x, y) \{ = f(x) + g(y) \} \quad (\text{cost function})$$

$$Ax + By = b \quad (\text{constraint})$$

where $f(x)$ and $g(y)$ are convex; A and B are matrices.

Define the augmented Lagrangian:

$$L_\mu(x, y) = f(x) + g(y) + \lambda^T(Ax + By - b) + \frac{\mu}{2}\|Ax + By - b\|_2^2.$$

It is iterative procedure at k th step:



1. $x^{k+1} = \arg \min_x L_\mu(x, y^k, \lambda^k);$
2. $y^{k+1} = \arg \min_y L_\mu(x^{k+1}, y, \lambda^k);$
3. $\lambda^{k+1} = \lambda^k + \mu(Ax^{k+1} + By^{k+1} - b).$

Algorithm 1 ADMM**Require:** the initial points: x_0, y_0, λ_0 **Ensure:** $k = 0$ **while** $k \neq N$ **do**

$$x^{k+1} = \arg \min_x L_\mu(x, y^k, \lambda^k)$$

$$y^{k+1} = \arg \min_y L_\mu(x^{k+1}, y, \lambda^k)$$

$$\lambda^{k+1} = \lambda^k + \mu(Ax^{k+1} + By^{k+1} - b)$$

$$k \leftarrow k + 1$$

end while**Note:** Thanks to *Professor He Bingsheng* who taught me this.See *Boyden's website for ADMM* for more information.

Notice that the quadratic term $\frac{\mu}{2} \|Ax + By - b\|_2^2$. Many extension is to find proper alternatives of this term to some specific problems, such as Bregman ADMM(BADMM). We replace the quadratic penalty term in the augmented Lagrangian (2) by a Bregman divergence as follows:

$$L_\mu^\phi(x, y, \lambda) = f(x) + g(y) + \lambda^T(Ax + By - b) + \mu B_\phi(b - Ax, By).$$

BADMM is initially proposed in *Bregman Alternating Direction Method of Multipliers***Algorithm 2** BADMM**Require:** the initial points: x_0, y_0, λ_0 **Ensure:** $k = 0$ **while** $k \neq N$ **do**

$$x^{k+1} = \arg \min_x L_\mu^\phi(x, y^k, \lambda^k)$$

$$y^{k+1} = \arg \min_y L_\mu^\phi(x^{k+1}, y, \lambda^k)$$

$$\lambda^{k+1} = \lambda^k + \mu(Ax^{k+1} + By^{k+1} - b)$$

$$k \leftarrow k + 1$$

end while

3.6 Stochastic Gradient Descent

Stochastic gradient descent takes advantages of stochastic or estimated gradient to replace the true gradient in gradient descent. It is *stochastic gradient* but may not be *descent*.



The name **stochastic gradient methods** may be more appropriate to call the methods with stochastic gradient. It can date back to **stochastic approximation**.

It is aimed to solve the problem with finite sum optimization problem, i.e.

$$\arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n f(\theta|x_i)$$

where $n < \infty$ and $\{f(\theta|x_i)\}_{i=1}^n$ are in the same function family and $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$ are constants while $\theta \in \mathbb{R}^p$ is the variable vector.

The difficulty is that p , that the dimension of θ , is tremendous. In other words, the model is **overparameterized**. And the number n is far larger than p generally, i.e. $n \gg p \gg d$. What is worse, the functions $\{f(\theta|x_i)\}_{i=1}^n$ are not convex in most case.

The stochastic gradient method is defined as

$$\theta^{k+1} = \theta^k - \alpha_k \frac{1}{m} \sum_{j=1}^m \nabla f(\theta^k|x'_j)$$

where x'_j is stochastically draw from $\{x_i\}_{i=1}^n$ and $m \ll n$.

It is the fact $m \ll n$ that makes it possible to compute the gradient of finite sum objective function and its side effect is that the objective function is not always descent. There is fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

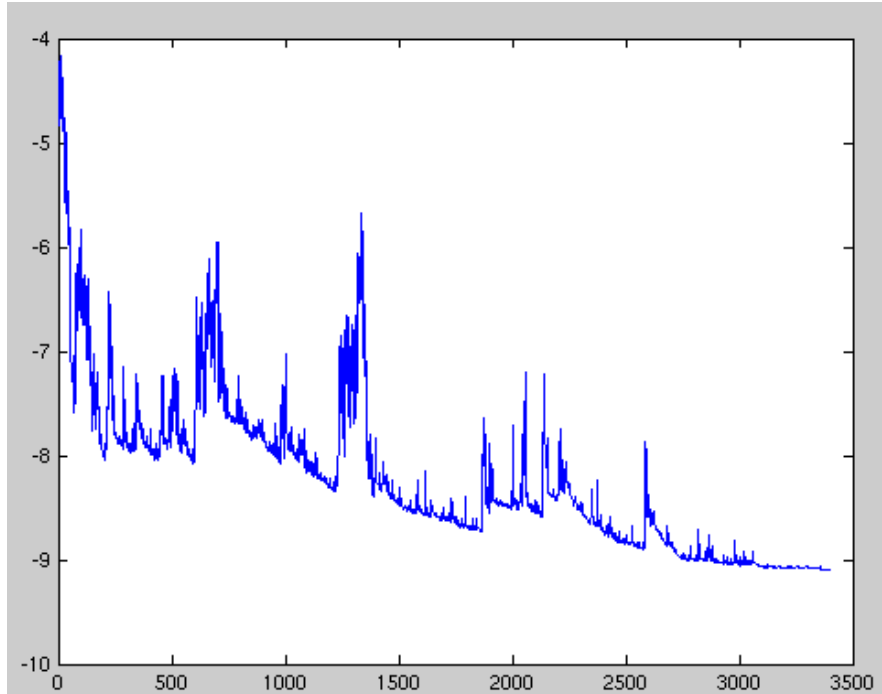


图 3.1: The fluctuations in the objective function as gradient with respect to mini-batches are taken

An heuristic proposal for avoiding the choice and for modifying the learning rate while the learning task runs is the bold driver (BD) method. The learning rate increases exponentially if



successive steps reduce the objective function f , and decreases rapidly if an “accident” is encountered (if objective function f increases), until a suitable value is found. After starting with a small learning rate, its modifications are described by the following equation:

$$\alpha_{k+1} = \begin{cases} \rho\alpha_k, & f(\theta^{k+1}) < f(\theta^k); \\ \eta^n\alpha_k, & f(\theta^{k+1}) > f(\theta^k) \text{ using } \alpha_k, \end{cases}$$

where ρ is close to 1 such as $\rho = 1.1$ in order to avoid frequent “accidents” because the objective function computation is wasted in these cases, η is chosen to provide a rapid reduction ($\eta = 0.5$), and n is the minimum integer such that the reduced rate η^n succeeds in diminishing the objective function.

The fact that the sample size is far larger than the dimension of parameter, $n \gg p$, that makes it expensive to compute total objective function $f(\theta) = \sum_{i=1}^n f(\theta|x_i)$. Thus it is not clever to determine the learning rate α_k by line search. And most stochastic gradient methods are to find proper step length α_k to make it converge at least in convex optimization. The variants of gradient descent such as momentum methods or mirror gradient methods have their stochastic counterparts.

- It is simplest to set the step length a constant, such as $\alpha_k = 3 \times 10^{-3} \forall k$.
- There are decay schemes, i.e. the step length α_k diminishes such as $\alpha_k = \frac{\alpha}{k}$, where α is constant.
- And another strategy is to tune the step length adaptively such as **AdaGrad**, **ADAM**.

PS: the step length α_k is called **learning rate** in machine learning and stochastic gradient descent is also named as *incremental gradient descent method* such as the survey at http://www.mit.edu/~dimitrib/Incremental_Survey_LIDS.pdf in some case.

See the following links for more information on *stochastic gradient descent*.



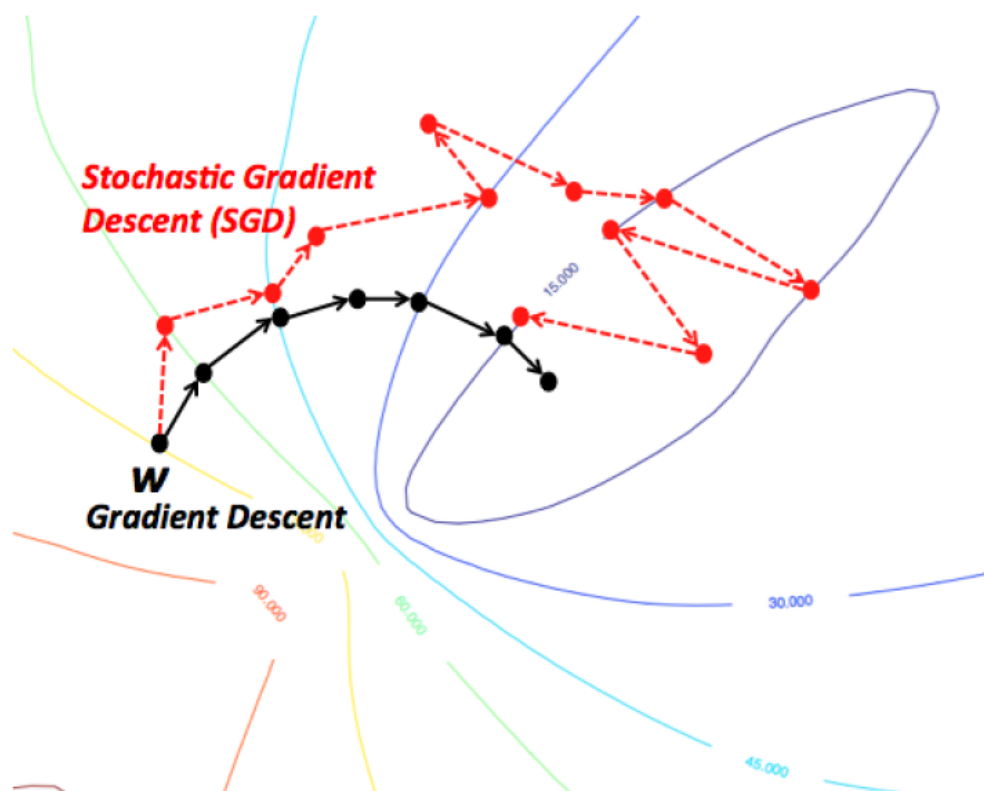


图 3.2: The Differences of Gradient Descent and Stochastic Gradient Descent



第 4 章 Artificial Neural Network



Artificial neural networks are most easily visualized in terms of a **directed graph**. In the case of sigmoidal units, node s represents sigmoidal unit and directed edge $e = (u, v)$ indicates that one of sigmoidal unit v 's inputs is the output of sigmoidal unit u . [A brief introduction to artificial neural network](#) tells the basics of artificial neural network in a visual intuitive way.

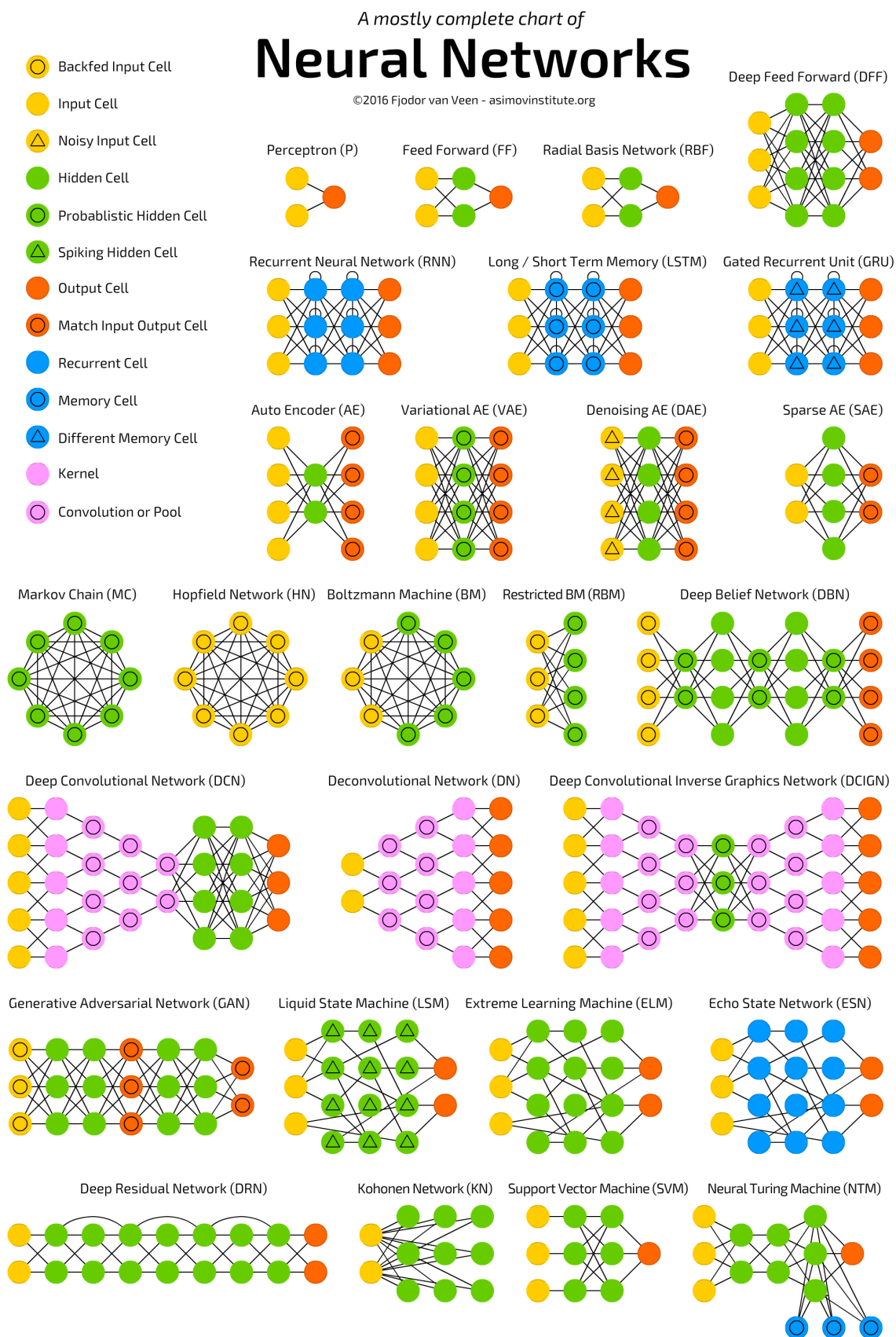


图 4.1: Neural Networks



4.1 Perceptron

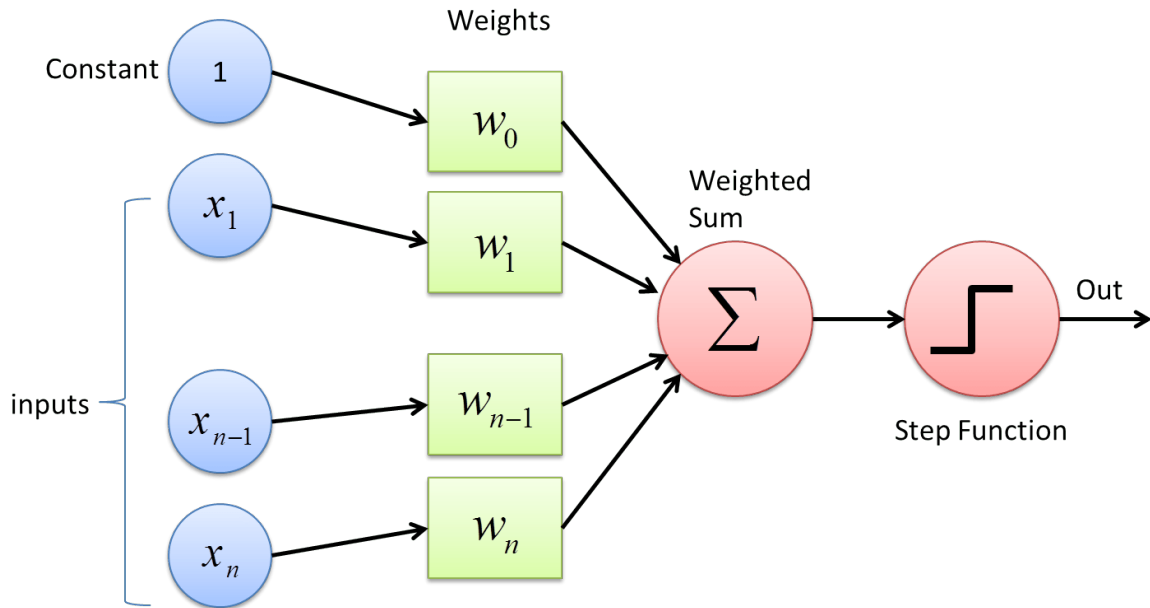


图 4.2: Perceptron

It is can be decomposed into 2 steps:

1. Aggregate all the information: $z = \sum_{i=1}^n w_i x_i + b_0 = (x_1, x_2, \dots, x_n, 1) \cdot (w_1, w_2, \dots, w_n, b_0)^T$.
2. Transform the information to activate something: $y = \sigma(z)$, where σ is nonlinear such as step function.

Convergence Proof for the Perceptron Algorithm shows the proof that perceptron is linear classifier.

4.1.1 Activation function

The nonlinear function σ is conventionally called activation function. There are some activation functions in history. The activation function σ can extend to more nonlinear function.

- Sign function

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ -1, & \text{if } x < 0 \end{cases}$$

- Step function

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$



- Radical base function

$$\rho(x) = e^{-\beta(x-x_0)^2}.$$

- TanH function

$$\tanh(x) = 2\sigma(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1.$$

- ReLU function

$$ReLU(x) = (x)_+ = \max\{0, x\} = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}.$$

4.2 Feed-forward Neural Networks

4.2.1 Representation of Feedforward Neural Network

Given that the function of a single neuron is rather simple, it subdivides the input space into two regions by a hyperplane, the complexity must come from having more layers of neurons involved in a complex action (like recognizing your grandmother in all possible situations). The "squashing" functions introduce critical nonlinearities in the system, without their presence multiple layers would still create linear functions. Organized layers are very visible in the human cerebral cortex, the part of our brain which plays a key role in memory, attention, perceptual awareness, thought, language, and consciousness.

The **feed-forward neural network** is also called multilayer perceptron. **The best way to create complex functions from simple functions is by *composition*.**

In mathematics, it can be considered as multi-layered non-linear composite function:

$$X \rightarrow \sigma \circ (W_1 X + b_1) = H_1 \rightarrow \sigma \circ (W_2 H_1 + b_2) = H_2 \rightarrow \cdots \sigma(WH + b) = y$$

where the notation \circ , $M_1, b_1, M_2, b_2, \dots$, W, b mean pointwise operation, the parameters in the affine mapping, respectively. Thus the data flow in the form of the chain:

$$f = H_1 \circ H_2 \circ \cdots \circ \sigma \quad \text{Composite form} \quad (4.1)$$

$$X \xrightarrow{\sigma} H_1 \xrightarrow{\sigma} H_2 \xrightarrow{\sigma} \cdots \xrightarrow{\sigma} y \quad \text{Hierarchy form} \quad (4.2)$$

$$\mathbb{R}^p \rightarrow \mathbb{R}^{l_1} \rightarrow \mathbb{R}^{l_2} \rightarrow \cdots \rightarrow \mathbb{R} \quad \text{Dimension} \quad (4.3)$$

where the circle notation \circ means forward composite or as the input of afterward operation. In hierarchy form, we omit the affine map.




Definition 4.1: Feed-forward Neural Networks

The feedforward neural networks are written in the *recursive form*:

$$\mathbf{z}_i = W_i H_{i-1} + b_i, \quad (4.4)$$

$$H_i = \sigma \circ (\mathbf{z}_i), \forall i \{1, 2, \dots, D\} \quad (4.5)$$

where $H_0 = X \in \mathbb{R}^p$, b_i is a vector and W_i is matrix. And the number of recursive times D is called the depth of network. 



Note: We can compute the input in each layer and sent the output to the next layer.

- In the first layer, we feed the input vector $X \in \mathbb{R}^p$ and connect it to each unit in the next layer $W_1 X + b_1 \in \mathbb{R}^{l_1}$ where $W_1 \in \mathbb{R}^{n \times l_1}$, $b_1 \in \mathbb{R}^{l_1}$. The output of the first layer is $H_1 = \sigma \circ (W_1 X + b_1)$, or in another word the output of j th unit in the first (hidden) layer is $h_j = \sigma(W_1 X + b_1)_j$ where $(W_1 X + b_1)_j$ is the j th element of l_1 -dimensional vector $W_1 X + b_1$.
- In the second layer, its input is the output of first layer, H_1 , and apply linear map to it: $W_2 H_1 + b_2 \in \mathbb{R}^{l_2}$, where $W_2 \in \mathbb{R}^{l_1 \times l_2}$, $b_2 \in \mathbb{R}^{l_2}$. The output of the second layer is $H_2 = \sigma \circ (W_2 H_1 + b_2)$, or in another word the output of j th unit in the second (hidden) layer is $h_j = \sigma(W_2 H_1 + b_2)_j$ where $(W_2 H_1 + b_2)_j$ is the j th element of l_2 -dimensional vector $W_2 H_1 + b_2$.
- The map between the second layer and the third layer is similar to (1) and (2): the linear maps datum to different dimensional space and the nonlinear maps extract better representations.
- In the last layer, suppose the input data is $H \in \mathbb{R}^l$. The output may be vector or scalar values and W may be a matrix or vector as well as y .

The ordinary feedforward neural networks take the *sigmoid* function $\sigma(x) = \frac{1}{1+e^{-x}}$ as the nonlinear activation function while the *RBF networks* take the **Radial basis function** as the activation function such as $\sigma(x) = e^{c\|x\|_2^2}$.

In theory, the universal approximation theorem show the power of feedforward neural network if we take some proper activation functions such as sigmoid function.

4.2.2 Evaluation and Optimization in Multilayer Perceptron

The problem is how to find the optimal parameters $W_1, b_1, W_2, b_2, \dots, W, b$? The multilayer perceptron is as one example of supervised learning, which means that we feed datum $D = \{(\mathbf{x}_i, d_i)\}_{i=1}^n$ to it and evaluate it.



The general form of the evaluation is given by:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \mathbb{L}[f(\mathbf{x}_i|\theta), \mathbf{d}_i]$$

where \mathbf{d}_i is the desired value of the input \mathbf{x}_i and θ is the parameters of multilayer perceptron. The notation $f(\mathbf{x}_i|\theta)$ is the output given parameters θ . The function \mathbb{L} is *loss function* to measure the discrepancy between the predicted value $f(\mathbf{x}_i|\theta)$ and the desired value \mathbf{d}_i .

In general, the number of parameters θ is less than the sample size n . And the objective function $J(\theta)$ is not convex.

We will solve it in the next section *Backpropagation, Optimization and Regularization*.

Visualizing level surfaces of a neural network with raymarching

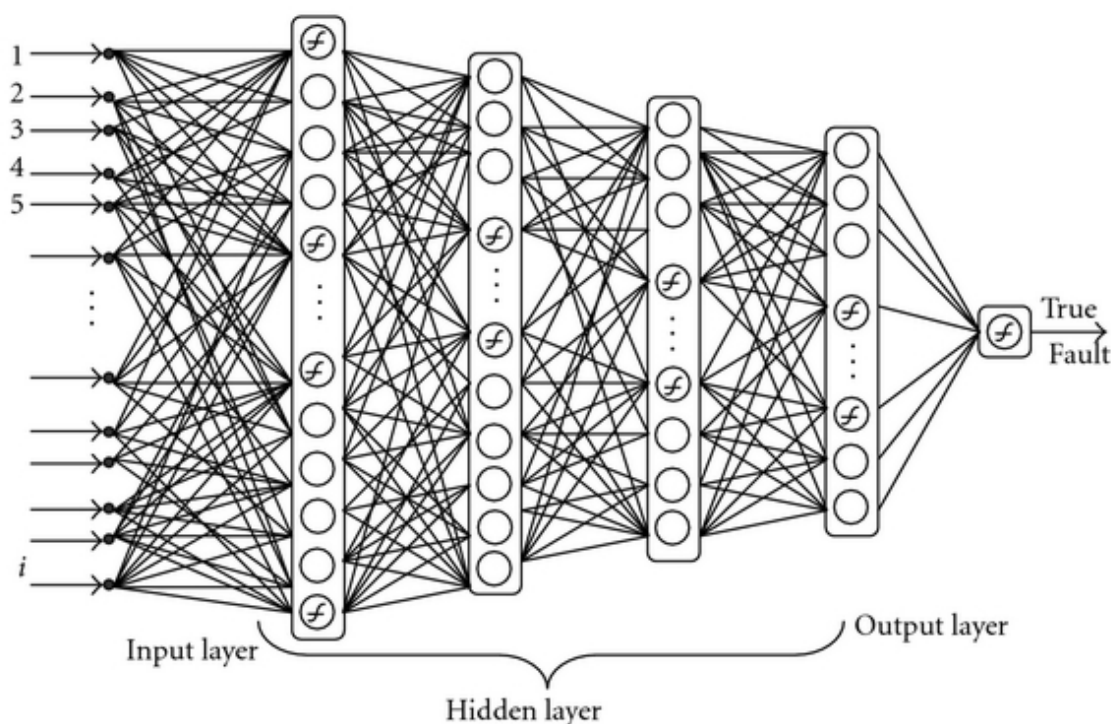


图 4.3: The diagram of MLP

4.2.3 Evaluation for different tasks

Evaluation is to judge the models with different parameters in some sense via objective function. In maximum likelihood estimation, it is likelihood or log-likelihood; in parameter estimation, it is bias or mean square error; in regression, it depends on the case.

In machine learning, evaluation is aimed to measure the discrepancy between the predicted values and trues value by 、 emphasisloss function. It is expected to be continuous smooth and



differential but it is not necessary. The principle is the loss function make the optimization or learning tractable. For classification, the loss function always is cross entropy; for regression, the loss function can be any norm function such as the ℓ_2 norm; in probability models, the loss function always is joint probability or logarithm of joint probability.

In classification, the last layer is to predict the degree of belief of the labels via **softmax function**, i.e.

$$\text{softmax}(z) = \left(\frac{\exp(z_1)}{\sum_{i=1}^n \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^n \exp(z_i)}, \dots, \frac{\exp(z_n)}{\sum_{i=1}^n \exp(z_i)} \right)$$

where n is the number of total classes. The labels are encoded as the one-hot vector such as $d = (1, 0, 0, \dots, 0)$. The **cross entropy** is defined as:

$$\mathbf{H}(d, p) = - \sum_{i=1}^n d_i \log(p_i) = \sum_{i=1}^n d_i \log\left(\frac{1}{p_i}\right),$$

where d_i is the i th element of the one-hot vector d and $p_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}$ for all $i = 1, 2, \dots, n$.

Suppose $d = (1, 0, 0, \dots, 0)$, the cross entropy is $\mathbf{H}(d, p) = -\log(p_1) = \log \sum_{i=1}^n \exp(z_i) - z_1$. The cost function is $\frac{1}{n} \sum_{i=1}^n \mathbf{H}(d^i, p^i)$ in the training data set $\{(\mathbf{x}_i, d^i)\}_{i=1}^n$ where \mathbf{x}_i is the features of i th sample and d^i is the desired true target label encoded in **one-hot** vector meanwhile p^i is the predicted label of \mathbf{x}_i .

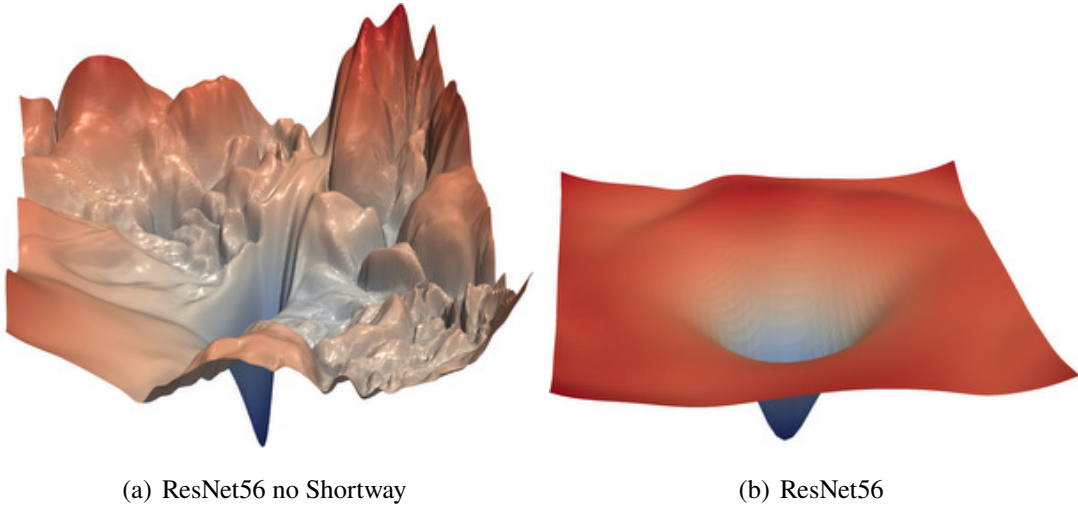


图 4.4: VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS|

In regression, the loss function may simply be the squared ℓ_2 norm, i.e. $\mathbb{L}(d, p) = (d - p)^2$ where d is the desired target and p is the predicted result. And the cost function is *mean squared error*:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n [f(\mathbf{x}_i | \theta) - d_i]^2.$$

In **robust statistics**, there are more loss functions such as *Huber loss*, *hinge loss*, *Tukey loss*.



- Huber loss function

$$Huber_{\delta}(x) = \begin{cases} \frac{|x|^2}{2}, & \text{if } |x| \leq \delta \\ \delta(|x| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

- Hinge loss function

$$Hinge(x) = \max\{0, 1 - tx\}$$

where $t = +1$ or $t = -1$.

- Tukey loss function

$$Tukey_{\delta}(x) = \begin{cases} (1 - [1 - x^2/\delta^2]^3)\frac{\delta^2}{6}, & \text{if } |x| \leq \delta \\ \frac{\delta^2}{6}, & \text{otherwise} \end{cases}$$

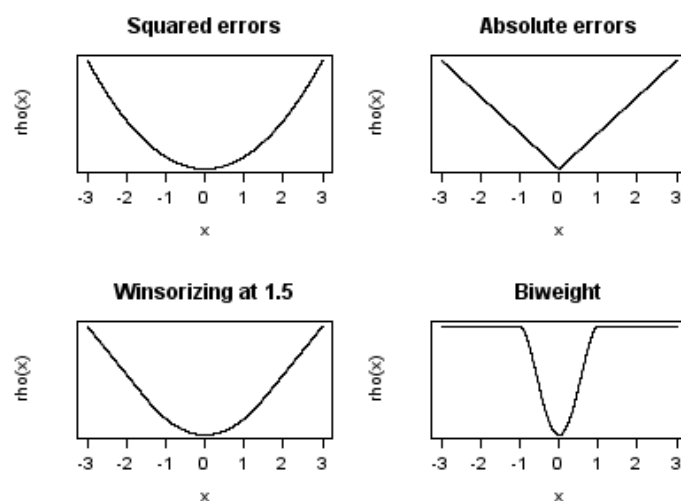


图 4.5: LOSS FUNCTIONS

It is important to choose or design loss function or more generally objective function, which can select variable as LASSO or confirm prior information as Bayesian estimation. Except the representation or model of neural network, it is the objective function that affects the usefulness of learning algorithms.

4.3 Backpropagation, Training and Regularization

4.3.1 Backpropagation

Automatic differentiation is the generic name for techniques that use the computational representation of a function to produce **analytic** values for the derivatives. Automatic differentiation techniques are founded on the observation that any function, no matter how complicated, is evaluated by performing a sequence of simple elementary operations involving



just one or two arguments at a time. Backpropagation is one special case of automatic differentiation, i.e. *reverse-mode automatic differentiation*.

The backpropagation procedure to compute the gradient of an objective function with respect to the weights of a multilayer stack of modules is nothing more than a practical application of the **chain rule for derivatives**. The key insight is that the derivative (or gradient) of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (or the input of the subsequent module). The backpropagation equation can be applied repeatedly to propagate gradients through all modules, starting from the output at the top (where the network produces its prediction) all the way to the bottom (where the external input is fed). Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights of each module.

Suppose that $f(x) = \sigma \circ (WH + b)$, where $H = \sigma \circ (W_4H_3 + b_4)$, $H_3 = \sigma \circ (W_3H_2 + b_3)$, $H_2 = \sigma \circ (W_2H_1 + b_2)$, $H_1 = \sigma \circ (W_1x + b_1)$, we want to compute the gradient $L(x_0, d_0) = \|f(x_0) - d_0\|_2^2$ with respect to all weights W_1, W_2, W_3, W :

$$\frac{\partial L(x_0, d_0)}{\partial W_n^i} = \frac{\partial L(x_0, d_0)}{\partial f(x_0)} \frac{\partial f(x_0)}{\partial W_n^i}$$

where $\forall i \in \{1, 2, \dots, l_n\}, \forall n \in \{1, 2, 3, 4\}$ and it is fundamental to compute the gradient with respect to the last layer as below.

- the gradient of loss function with respect to the prediction function:

$$\frac{\partial L(x_0, d_0)}{\partial f(x_0)} = 2[f(x_0) - d_0],$$

- the gradient of each unit in prediction function with respect to the weight in the last layer:

$$\frac{\partial f^j(x_0)}{\partial W^j} = \frac{\partial \sigma(W^j H + b^j)}{\partial W^j} = \sigma'(W^j H + b^j) H \quad \forall j \in \{1, 2, \dots, l\},$$

- the gradient of prediction function with respect to the last hidden state:

$$\frac{\partial f^j(x_0)}{\partial H} = \frac{\partial \sigma(W^j H + b^j)}{\partial H} = \sigma'(W^j H + b^j) W^j \quad \forall j \in \{1, 2, \dots, l\},$$

where $f^j(x_0)$, W^j , b^j and $\sigma'(z)$ is the j th element of $f(x_0)$, the j -th row of matrix W , the j th element of vector b and $\frac{d\sigma(z)}{dz}$, respectively.



Note: Each connection, the black line, is attached with a weight parameter.

Recall the chain rule with more variables:

$$\frac{\partial f(m(x_0), n(x_0))}{\partial x_0} = \frac{\partial f(m(x_0), n(x_0))}{\partial m(x_0)} \frac{\partial m(x_0)}{\partial x_0} + \frac{\partial f(m(x_0), n(x_0))}{\partial n(x_0)} \frac{\partial n(x_0)}{\partial x_0}.$$

Similarly, we can compute following gradients:

$$\frac{\partial H^j}{\partial W_4^j} = \frac{\partial \sigma(W_4^j H_3 + b_4^j)}{\partial W_4^j} = \sigma'(W_4^j H_3 + b_4^j) H_3 \quad \forall j \in \{1, 2, \dots, l\};$$



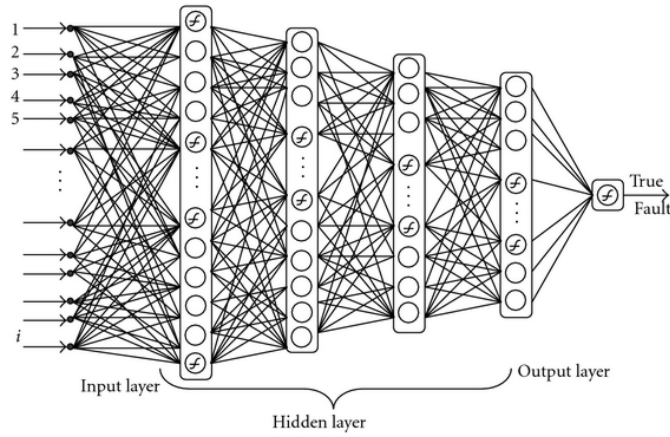


图 4.6: The Architecture of Feedforward Neural Networks

$$\begin{aligned}\frac{\partial H_3^j}{\partial H_3} &= \frac{\partial \sigma(W_4^j H_3 + b_4^j)}{\partial H_3} = \sigma'(W_4^j H_3 + b_4^j) W_4^j & \forall j \in \{1, 2, \dots, l_4\}; \\ \frac{\partial H_3^j}{\partial W_3^j} &= \frac{\partial \sigma(W_3^j H_2 + b_3^j)}{\partial W_3^j} = \sigma'(W_3^j H_2 + b_3^j) H_2 & \forall j \in \{1, 2, \dots, l_3\}; \\ \frac{\partial H_3^j}{\partial H_2} &= \frac{\partial \sigma(W_3^j H_2 + b_3^j)}{\partial H_2} = \sigma'(W_3^j H_2 + b_3^j) W_3^j & \forall j \in \{1, 2, \dots, l_3\}; \\ \frac{\partial H_2^j}{\partial W_2^j} &= \frac{\partial \sigma(W_2^j H_1 + b_2^j)}{\partial W_2^j} = \sigma'(W_2^j H_1 + b_2^j) H_1 & \forall j \in \{1, 2, \dots, l_2\}; \\ \frac{\partial H_2^j}{\partial H_1} &= \frac{\partial \sigma(W_2^j H_1 + b_2^j)}{\partial H_1} = \sigma'(W_2^j H_1 + b_2^j) W_2^j & \forall j \in \{1, 2, \dots, l_2\}; \\ \frac{\partial H_1^j}{\partial W_1^j} &= \frac{\partial \sigma(W_1^j x_0 + b_1^j)}{\partial W_1^j} = \sigma'(W_1^j x_0 + b_1^j) x_0 & \forall j \in \{1, 2, \dots, l_1\}.\end{aligned}$$

The multilayer perceptron $f(x)$ can be written in a chain form:

$$\begin{aligned}X &\xrightarrow{\sigma} H_1 \xrightarrow{\sigma} H_2 \xrightarrow{\sigma} H_3 \xrightarrow{\sigma} H_4 \xrightarrow{\sigma} H \xrightarrow{\sigma} f(x) \\ \mathbb{R}^p &\rightarrow \mathbb{R}^{l_1} \rightarrow \mathbb{R}^{l_2} \rightarrow \mathbb{R}^{l_3} \rightarrow \mathbb{R}^{l_4} \rightarrow \mathbb{R}^l \rightarrow \mathbb{R}^o \\ X &\rightarrow W_1 X \rightarrow W_2 H_1 \rightarrow W_3 H_2 \rightarrow W_4 H_3 \rightarrow W H \rightarrow y\end{aligned}$$

while the backpropagation to compute the gradient is in the reverse order:

$$\frac{\partial y}{\partial W} \rightarrow \frac{\partial y}{\partial H} \rightarrow \frac{\partial H}{\partial W_4} \rightarrow \frac{\partial H}{\partial H_3} \rightarrow \frac{\partial H_3}{\partial W_3} \rightarrow \frac{\partial H_3}{\partial H_2} \rightarrow \frac{\partial H_2}{\partial W_2} \rightarrow \frac{\partial H_2}{\partial W_1} \rightarrow \frac{\partial H_1}{\partial W_1}. \quad (4.6)$$

In general, the gradient of any weight can be computed by *backpropagation* algorithm. The first step is to compute the gradient of loss function with respect to the output $f(x_0) =$



$y \in \mathbb{R}^o$, i.e. $\frac{\partial L(x_0, d_0)}{\partial f(x_0)} = 2(f(x_0) - d_0) = 2(\sigma \circ (WH + b) - d_0)$, of which the i th element is $2(y^i - d_0^i) = 2(\sigma(W^i H + b^i) - d_0^i) \forall i \in \{1, 2, \dots, o\}$. Thus

$$\frac{\partial L(x_0, d_0)}{\partial W^i} = \frac{\partial L(x_0, d_0)}{\partial y^i} \frac{\partial y^i}{\partial W^i} = 2(y^i - d_0^i) \sigma'(W^i H + b^i) H.$$

Thus we can compute all the gradients of W columns. Note that H has been computed through forwards propagation in that layer.

And $H = \sigma \circ (W_4 H_3 + b_3)$, of which the i th element is $H^i = \sigma(W_4 H_3 + b_4)^i = \sigma(W_4^i H_3 + b_4^i)$.

And we can compute the gradient of columns of W_4 :

$$\frac{\partial L(x_0, y_0)}{\partial W_4^i} = \sum_{j=1}^o \left[\frac{\partial L(x_0, y_0)}{\partial f^j(x_0)} \right] \left[\frac{\partial f^j(x_0)}{\partial z} \right] \frac{\partial z}{\partial W_4^i} \quad (4.7)$$

$$= \sum_{j=1}^o \left[\frac{\partial L(x_0, y_0)}{\partial y^j} \right] \left[\frac{\partial y^j}{\partial H^i} \right] \frac{\partial H^i}{\partial W_4^i} \quad (4.8)$$

$$= \sum_{j=1}^o \left[\frac{\partial L}{\partial y^j} \right] \left[\frac{\partial y^j}{\partial (W^j H + b^j)} \right] \left[\frac{\partial (W^j H + b^j)}{\partial H^i} \right] \frac{\partial (H^i)}{\partial W_4^i} \quad (4.9)$$

$$= \sum_{j=1}^l \left[\frac{\partial L}{\partial y^j} \right] [\sigma'(W^j H + b^j) W^{j,i}] [\sigma'(W_4^i H_3 + b_4^i) H_3], \quad (4.10)$$

where $W^{j,i}$ is the i th element of j th column in matrix W .

$$\frac{\partial L(x_0, y_0)}{\partial W_3^i} = \sum_{j=1}^o \frac{\partial L(x_0, y_0)}{\partial f^j(x_0)} \left[\frac{\partial f^j(x_0)}{\partial z} \right] \frac{\partial z}{\partial W_3^i} \quad (4.11)$$

$$= \sum_{j=1}^o \frac{\partial L}{\partial y^j} \left[\frac{\partial y^j}{\partial H_3^i} \right] \frac{\partial H_3^i}{\partial W_3^i} \quad (4.12)$$

$$= \sum_{j=1}^o \frac{\partial L}{\partial y^j} \left[\sum_{k=1}^o \frac{\partial y^j}{\partial H^k} \frac{\partial H^k}{\partial H_3^i} \right] \frac{\partial H_3^i}{\partial W_3^i} \quad (4.13)$$

where all the partial derivatives or gradients have been computed or accessible. It is nothing except to add or multiply these values in the order when we compute the weights of hidden layer.

$$\frac{\partial L(x_0, y_0)}{\partial W_2^i} = \sum_{j=1}^o \frac{\partial L(x_0, y_0)}{\partial f^j(x_0)} \left[\frac{\partial f^j(x_0)}{\partial z} \right] \frac{\partial z}{\partial W_2^i} \quad (4.14)$$

$$= \sum_{j=1}^l \frac{\partial L}{\partial y^j} \left[\frac{\partial y^j}{\partial H_2^i} \right] \frac{\partial H_2^i}{\partial W_2^i} \quad (4.15)$$

$$= \sum_{j=1}^o \frac{\partial L}{\partial y^j} \left\{ \sum_{k=1}^o \frac{\partial y^j}{\partial H^k} \left[\sum_m \frac{\partial H^k}{\partial H_3^m} \frac{\partial H_3^m}{\partial H_2^i} \right] \right\} \frac{\partial H_2^i}{\partial W_2^i} \quad (4.16)$$



And the gradient of the first layer is computed by

$$\frac{\partial L(x_0, y_0)}{\partial W_1^i} = \sum_j \frac{\partial L(x_0, y_0)}{\partial y^j} \frac{\partial y^j}{\partial z} \frac{\partial z}{\partial W_1^i} \quad (4.17)$$

$$= \sum_j \frac{\partial L}{\partial y^j} \left[\sum_k \frac{\partial y^j}{\partial H^k} \sum_m \frac{\partial H^k}{\partial H_3^m} \sum_n \frac{\partial H_3^k}{\partial H_2^n} \sum_r \frac{\partial H_2^n}{\partial H_1^r} \right] \frac{\partial H_1^i}{\partial W_1^i}. \quad (4.18)$$

See more information on backpropagation in the following list

- [Back-propagation, an introduction at offconvex.org](https://offconvex.org/);
- [Automatic differentiation on Wikipedia](#);
- [backpropagation on brilliant](#);
- [Who invented backpropagation ?](#);
- For more information on automatic differentiation see the book *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition* by Andreas Griewank and Andrea Walther at <https://epubs.siam.org/doi/book/10.1137/1.9780898717761>.

4.3.2 Training Methods

The training is to find the optimal parameters of the model based on the **training data set**. The training methods are usually based on the gradient of cost function as well as back-propagation algorithm in deep learning. See *Stochastic Gradient Descent* in *Numerical Optimization* for details. In this section, we will talk other optimization tricks such as *Normalization*.

Concepts
<i>Overfitting and Underfitting</i>
<i>Memorization and Generalization</i>
<i>Normalization and Standardization</i>

- <https://srdas.github.io/DLBook/GradientDescentTechniques.html#InitializingWeights>.
- <https://srdas.github.io/DLBook/ImprovingModelGeneralization.html>.
- <https://srdas.github.io/DLBook/HyperParameterSelection.html>.
- https://github.com/scutan90/DeepLearning-500-questions/tree/master/ch13_%E4%BC%98%E5%8C%96%E7%AE%97%E6%B3%95

Initialization and More

- <https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html>.
- <http://www.princeton.edu/~congml/Publication/RandomInit/main.pdf>.
- <http://www.cnblogs.com/yymn/articles/4995755.html>

Batch Size



Normalization

Batch Normalization

Algorithm 3 Batch Normalization

Require: a mini batch: $\mathbb{B} = \{x_1, x_2, \dots, x_m\}$

Ensure: $k = 0$

while $k \neq m$ **do**

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

$$k \leftarrow k + 1$$

end while

It is an effective way to accelerate deep learning training. And *Batch Normalization* transformation is differentiable so that we can compute the gradient in backpropagation. For example, let ℓ be the cost function to be minimized, then we could compute the gradients.

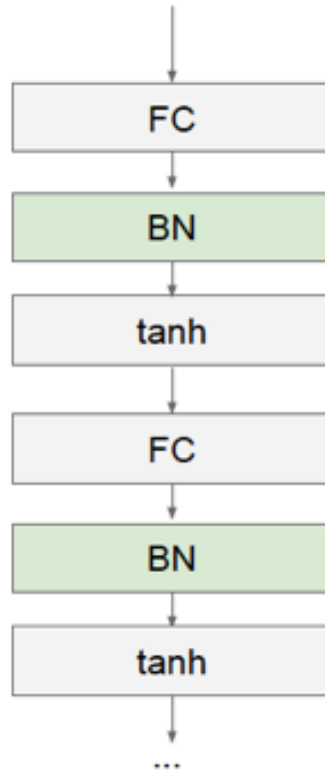


图 4.7: Batch Normlization



$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \frac{\partial y_i}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \gamma \quad (4.19)$$

$$\frac{\partial \ell}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \frac{-\frac{1}{2}(x_i - \mu_B)}{(\sigma_B^2 + \epsilon)^{\frac{3}{2}}} \quad (4.20)$$

$$\frac{\partial \ell}{\partial \mu_B} = \frac{\partial \ell}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial \mu_B} + \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \mu_B} = \frac{\partial \ell}{\partial \sigma_B^2} \left[\frac{2}{m} \sum_{i=1}^m (x_i - \mu_B) \right] + \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \frac{-1}{\sqrt{(\sigma_B^2 + \epsilon)}} \quad (4.21)$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial \ell}{\partial \mu_B} \frac{\partial \mu_B}{\partial x_i} + \frac{\partial \ell}{\partial \sigma_B^2} \frac{\partial \sigma_B^2}{\partial x_i} \quad (4.22)$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \hat{x}_i \quad (4.23)$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \hat{x}_i \quad (4.24)$$

Layer Normalization

Different from the Batch Normalization, *Layer Normalization* compute the average μ or variance σ for all the inputs of the given hidden layer, as

$$\mu = \frac{1}{n} \sum_i^n x_i \quad \sigma = \sqrt{\frac{1}{n} \sum_i^n (x_i - \mu)^2 + \epsilon}$$

where n is the number of units in the given hidden layer.

See *Improve the way neural networks learn* at <http://neuralnetworksanddeeplearning.com/chap3.html>. See more on nonconvex optimization at <http://sunju.org/research/nonconvex/>.

4.3.3 Regularization

In mathematics, statistics, and computer science, particularly in the fields of machine learning and inverse problems, regularization is a process of introducing additional information in order to solve an ill-posed problem or to prevent over-fitting. In general, regularization is a technique that applies to objective functions in ill-posed optimization problems. It changes the objective function or more generally the optimization procedure. However, it is not crystal clear that what is the relationship between the optimization techniques and generalization ability. See the following links for more information on optimization and generalization.

- <https://www.inference.vc/sharp-vs-flat-minima-are-still-a-mystery-to-me/>
- <https://arxiv.org/abs/1506.02142>
- <https://arxiv.org/abs/1703.04933>



- <https://arxiv.org/abs/1810.05369>
- <http://www.offconvex.org/2017/12/08/generalization1/>
- <http://www.offconvex.org/2018/02/17/generalization2/>
- <http://lcs1.mit.edu/courses/regml/regml2016/>
- <https://chunml.github.io/ChunML.github.io/tutorial/Regularization/>
- <http://www.mit.edu/~9.520/fall16/>
- https://blog.csdn.net/xzy_thu/article/details/80732220

Parameter norm penalty

The ℓ_2 norm penalty is to add the squares of ℓ_2 norm of parameters to the objective function $J(\theta)$ to reduce the parameters(or weights) as shown in ridge regression with regular term coefficient λ , i.e. $J(\theta) + \lambda\|\theta\|_2^2$. Suppose $E(\theta) = J(\theta) + \lambda\|\theta\|_2^2$, the gradient descent take approximate (maybe inappropriate) form

$$\theta = \theta - \eta \frac{\partial E(\theta)}{\partial \theta} = \theta - \eta \frac{\partial J(\theta)}{\partial \theta} - 2\eta\lambda\theta$$

thus

$$\frac{\partial J(\theta)}{\partial \theta} = -2\lambda\theta \implies J(\theta) = e^{-2\lambda\theta}.$$

If we want to find the minima of $E(\theta)$, θ will decay to 0. It extends to the following iterative formula:

$$\theta^{t+1} = (1 - \lambda)\theta^t - \alpha_t \frac{\partial J(\theta^t)}{\partial \theta},$$

where λ determines how you trade off the original cost $J(\theta)$ with the large weights penalization. The new term λ coming from the regularization causes the weight to decay in proportion to its size.

- The moment method and weight: <https://stats.stackexchange.com/questions/70101/neural-networks-weight-change-momentum-and-weight-decay>;
- Weight decay at neural networks https://metacademy.org/graphs/concepts/weight_decay_neural_networks;
- Optimization and regularization at machine mastery <https://machinelearningmastery.com/>.

The ℓ_1 norm penalty is also used in deep learning as in **LASSO**. It is to solve the following optimization problem:

$$\min_{\theta} J(\theta) + \lambda\|\theta\|_1,$$

where λ is a hyperparameter. Sparsity brings to the model as shown as in ****LASSO****.



Early stop

Its essential is to make a balance in memorization and generalization. Early stopping is to stop the procedure before finding the minima of cost in training data. It is one direct application of *cross validation*.

- https://www.wikiwand.com/en/Early_stopping;
- [https://www.wikiwand.com/en/Cross-validation_\(statistics\)](https://www.wikiwand.com/en/Cross-validation_(statistics));
- <https://machinelearningmastery.com/>

Dropout

It is to cripple the connections stochastically, which is often used in visual tasks. See the original paper [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#).

- <https://www.zhihu.com/question/24529483>;
- <https://www.jiqizhixin.com/articles/2018-11-10-7>;
- <https://www.jiqizhixin.com/articles/112501>;
- <http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html>;
- <https://www.jeremyjordan.me/deep-neural-networks-preventing-overfitting/>;

Data Augmentation

Data augmentation is to augment the training datum specially in visual recognition. **Overfitting** in supervised learning is data-dependent. In other words, the model may generalize better if the data set is more diverse. It is to collect more datum in the statistical perspective.

4.4 Bibliography

This template uses BibTeX to generate the bibliography, the default bibliography style is aer. ? use data from a major peer-to-peer lending marketplace in China to study whether female and male investors evaluate loan performance differently. You can add bib items (from Google Scholar, Mendeley, EndNote, and etc.) to reference.bib file, and cite the bibkey in the tex file.

