

the Deep Learning Book
–summary–

by
Ian Goodfellow
Yoshua Bengio
Aaron Courville

Summarized by HyeongGyu Choi

1 Introduction¹

1.1 Early Artificial Intelligence

In the early ages of AI, people tackled and solved intellectually challenging tasks, but relatively straight forward for computers. IBM Deep Blue's Chess challenge in 1997 is one example. Nevertheless, rather intuitive tasks for human beings, such as seeing or hearing, turned out 'un-simple' for computers.

1.2 Knowledge Base AI

Here, we hard-code knowledge about the world in formal languages. For example, Cyc(1989) was an inference engine and a database of statements in a language called CycL. But the engine had difficulties in figuring out logic in several cases.

1.3 Machine Learning

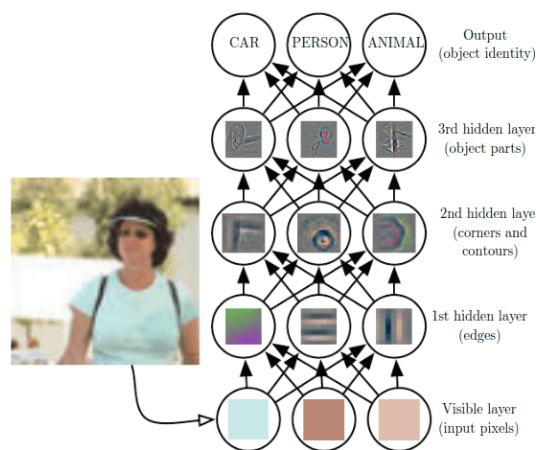
Through machine learning, AI systems have the ability to acquire their own knowledge, by extracting patterns from raw data. Logistic regression, naive Bayes are some of the examples. The performance of ML algorithms depend heavily on the representation of the data fed. The representation is called *features*. Deciding what feature to be fed into the system is an important part in ML, which requires domain expertise.

In machine learning, the goal is usually to separate the factors of variation. By factors of variation, we mean separate sources of influence that are not represented physically in data. That is, they are the sources of variation in data distribution. This makes it hard for the ML model to train on the data.

1.4 Representation Learning

The machine can not only learn the mapping from representation to output but also the representation itself. A representative example is the 'Autoencoder'. It consists of the encoder function and the decoder function. The encoder function converts the input data into a lower dimensional 'representation', and the decoder learns to reconstruct the original data with the 'representation'.

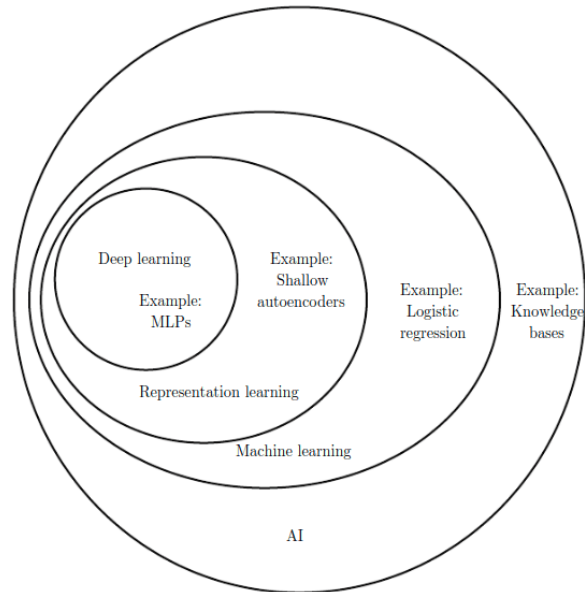
1.5 Deep Learning



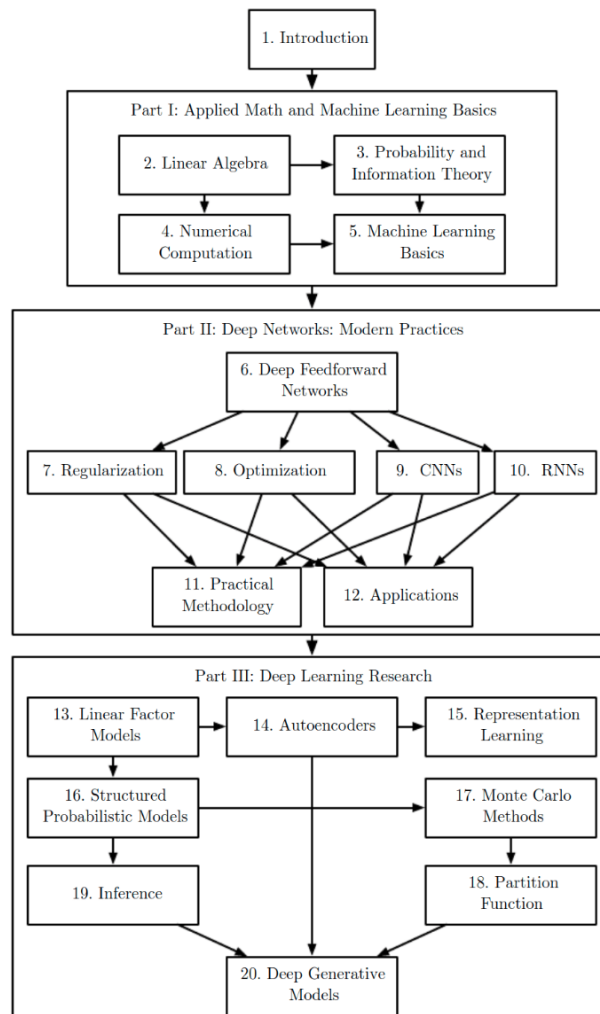
Generally, it is very difficult to learn representations from data and map them into higher level concepts. Deep Learning solves this central problem by introducing representations that are expressed in terms of other simpler representations. The image above illustrates how the deep learning works. The image of a person is represented by simpler concepts such as corners and contours. The layers that represent the lower level concepts are called the *hidden layers*. On the contrary, the input data are denoted *visible layers*.

¹Largely referenced leejunhyun.github.io

1.6 Hierarchy in Artificial Intelligence



1.7 Content Overview



PART I. Applied Math and Machine Learning Basics

2 Linear Algebra²

2.1 Scalars, Vectors, Matrices and Tensors

2.1.1 Scalars

A scalar is a single number. It's written in lowercase *italic*, and its numeric type have to be specified. For instance, 'Let $s \in \mathbb{R}$ be the slope of the line'.

2.1.2 Vectors

A vector is an array of numbers. It's written in lowercase **bold** typeface, and its elements are written in italic with a subscript. Also, if you define $S = \{1, 3, 6\}$ and call x_S , it refers to the elements x_1, x_3, x_6 . If you want to exclude certain vector elements, simply put a negative sign(-) in front of the subscript.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

2.1.3 Matrices

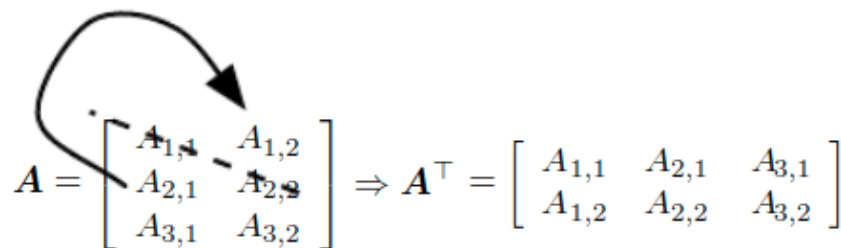
A matrix is a 2 dimensional array of numbers. It's written in uppercase **bold** typeface. If a real-valued matrix **A** has a height of m and a width of n , we may say that $\mathbf{A} \in \mathbb{R}^{m \times n}$. The elements in a matrix is denoted $A_{m,n}$, without bold typeface.

2.1.4 Tensors

A tensor is a higher dimensional array aligned in a grid. Tensor named 'A' is written **A**, and its elements are denoted $A_{i,j,k}$. The tensor encompasses a 2 dimensional matrix as well.

One of the important matrix operations is the **transpose** operation. It is written and represented as follows.

$$(\mathbf{A}^T)_{i,j} = A_{j,i}$$


$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \mathbf{A}^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

²Largely referenced leejunhyun.github.io

2.2 Multiplying Matrices and Vectors

If matrix **A**'s shape is $(m \times n)$ and **B**'s shape is $(n \times p)$, the multiplication of the two matrices will result in a matrix of shape $(m \times p)$.

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

An **element-wise product**, or the **Hadamard product** is denoted $A \odot B$. Furthermore, while the matrix multiplication is not always commutative, the **dot product** of vectors is commutative : $x^T y = y^T x$, and is denoted $x^T y$.

2.3 Identity and Inverse Matrices

An **Identity matrix** is a matrix that does not change any vector when multiplied to another vector. We denote an n -dimensional identity matrix as $I_n \in \mathbb{R}^{n \times n}$. Thus, $\forall x \in \mathbb{R}^n, I_n x = x$

The **matrix inverse** of A is denoted as A^{-1} such that $A^{-1} A = I_n$. However, an inverse matrix for A may not always exist.

2.4 Linear Dependence and Span

2.4.1 Linear Combination

We can write a matrix equation as a vector equation as follows

$$Ax = \begin{bmatrix} | & | & | \\ a_1 & a_2 & a_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} | \\ a_1 \\ | \end{bmatrix} x_1 + \begin{bmatrix} | \\ a_2 \\ | \end{bmatrix} x_2 + \begin{bmatrix} | \\ a_3 \\ | \end{bmatrix} x_3 = b$$

2.4.2 Span

A **span** of a set of vectors is the set of all points obtainable by linear combination of the original vectors. That is, given vectors $\{v_1, v_2, \dots, v_i\}$, a space obtainable from a linear combination of $\sum c_i v_i$ is the span of the vectors. For the equation above, in order for x to have its solution, vector b should be in the span of vector a_1, a_2, a_3 ($b \in \text{Span}\{a_1, a_2, a_3\}$).

If the solution exists, whether it is a single solution or a multiple solution depends on the linear dependency.

- Multiple solutions : vectors a_1, a_2, a_3 are linearly dependent
- Single solution : vectors a_1, a_2, a_3 are linearly independent

When using the inverse matrix to solve the equation $Ax = b$, matrix A must be an $m \times m$ square matrix and the column vectors should be linearly independent. A matrix whose inverse matrix cannot be derived because of linear dependency is called a **singular matrix**.

2.5 Norms

A **Norm** is used to represent a vectors magnitude. The L^p norm is calculated as follows.

$$||x||_p = (\sum_i |x_i|^p)^{\frac{1}{p}}, \text{ for } p \in \mathbb{R}, p \geq 1$$

More specifically, a norm is any function f that satisfies the following properties :

- $f(x) = 0 \Rightarrow x = 0$
- $f(x + y) \leq f(x) + f(y)$ (the triangle inequality)
- $\forall \alpha \in \mathbb{R}, f(\alpha x) = |\alpha| f(x)$

2.5.1 L^2 Norm

Commonly called the **Euclidean norm**, this is the Euclidean distance from the origin to the vector point. In machine learning, the subscript is omitted and written as $\|x\|$. Generally, the norm is squared to a **squared L^2 norm** ($= x^T x$) for computational convenience.

2.5.2 L^1 Norm

The L^1 norm is $\|x\|_1 = \sum_i |x_i|$. This is used when the difference between zero and nonzero elements with very small magnitude is crucial.

2.5.3 L^∞ Norm

Often called the **max norm**, this norm selects the element with the largest absolute value in the vector. $\|x\|_\infty = \max_i |x_i|$

2.5.4 Frobenius Norm

This norm measures the size of a matrix. Similar to the L^2 norm, $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$

2.5.5 Dot Products

The dot product of two vectors can be rewritten in terms of norms.

$$x^T y = \|x\|_2 \|y\|_2 \cos \theta$$

2.6 Special Kinds of Matrices and Vectors

2.6.1 Diagonal Matrix

It consists of zeros, and have nonzero entries only along the main diagonal. It is denoted $diag(v)$. $diag(v)x$ is the element-wise multiplication $v \odot x$. The inverse matrix of $diag(v)$ is also often used. $diag(v)^{-1} = diag([\frac{1}{v_1}, \dots, \frac{1}{v_n}]^T)$

2.6.2 Symmetric Matrix

A matrix that satisfies $A = A^T$

2.6.3 Orthogonal Matrix

A matrix that satisfies $A^T A = A A^T = I$. This implies that, $A^{-1} = A^T$. If vectors not only are orthogonal but also have unit norm, we call them **orthonormal**.

2.7 Eigendecomposition

For an integer 12, we can decompose it into prime factors : $2 \times 2 \times 3$. Similarly, we can decompose mathematical objects such as scalars, vectors, matrices and tensors. By doing so, we can have a deeper look into their functional properties. One of the most widely used method is the **eigendecomposition**. In eigendecomposition, we decompose a matrix into a set of eigenvectors and eigenvalues.

$$Av = \lambda v$$

where, v is a nonzero vector, and λ is a scalar.

If matrix A has n dimensional linearly independent eigenvector $V = \{v^{(1)}, \dots, v^{(n)}\}$ and a corresponding eigenvalue $\{\lambda_1, \dots, \lambda_n\}$, we can express it as an eigendecomposition as $A = V diag(\lambda) V^{-1}$

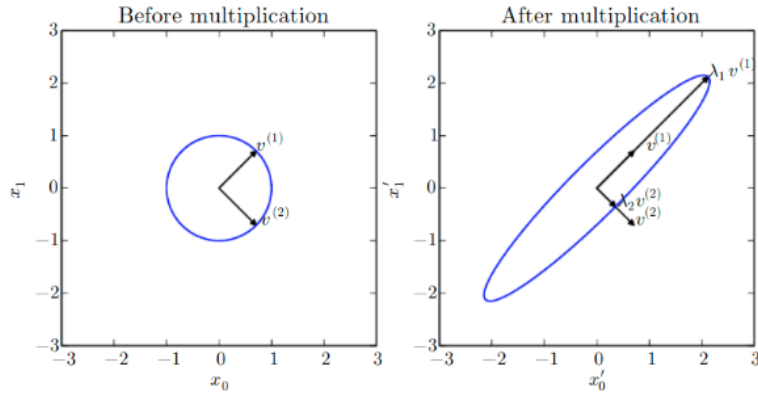


Figure 2.3: An example of the effect of eigenvectors and eigenvalues. Here, we have a matrix A with two orthonormal eigenvectors, $\mathbf{v}^{(1)}$ with eigenvalue λ_1 and $\mathbf{v}^{(2)}$ with eigenvalue λ_2 . (Left) We plot the set of all unit vectors $\mathbf{u} \in \mathbb{R}^2$ as a unit circle. (Right) We plot the set of all points $A\mathbf{u}$. By observing the way that A distorts the unit circle, we can see that it scales space in direction $\mathbf{v}^{(i)}$ by λ_i .

Every real symmetric matrix can be decomposed into an expression using only real-valued eigenvectors and eigenvalues : $A = Q\Lambda Q^T$, where Q is an orthogonal matrix(eigenvector) and Λ is a diagonal matrix(eigenvalue). For matrix A , we cannot guarantee that this is the unique decomposition. If any two or more eigenvectors share the same eigenvalue, all sets of orthogonal vectors in their span are eigenvectors of that eigenvalue. On the other hand, if the eigenvalues are unique, then the eigendecomposition is unique.

The eigendecomposition holds a lot of information of a matrix.

- a matrix is singular \Leftrightarrow at least one of the eigenvalues is 0
- function f 's minimum is MIN(eigenvalues) and maximum is *vice versa*

For matrix A , if all eigenvalues are positive, it is a **positive definite**, and if all eigenvalues are greater than or equal to 0, it is a **positive semidefinite**. For the negative portion, *vice versa* (**negative definite** / **negative semidefinite**).

2.8 Singular Value Decomposition

The singular value decomposition decomposes a matrix into singular vectors and singular values. It's similar to eigendecomposition, but more generally applicable. We express SVD as $A = UDV^T$, where A is an $m \times m$ matrix, matrix D an $m \times n$, and matrix V an $n \times n$. Matrix U and V are orthogonal matrices, and matrix D is a diagonal matrix, which does not have to be a square matrix, and is called the **singular value** of matrix A . The columns of eigenvectors of $U = AA^T$ are the left-singular vector, and those of $V = A^T A$ is a right=singular vector. The most useful feature of the SVD is that it is used to partially generalize matrix inversion to nonsquare matrices.

2.9 The Moore-Penrose Pseudoinverse

When matrix A in $Ax = y$ cannot be mapped to a unique inverse matrix, we may use the **Moore-Penrose pseudoinverse**.

$$A^+ = \lim_{\alpha \searrow 0} (A^T A + \alpha I)^{-1} A^T$$

However, practically, the formula $A^+ = VD^+U^T$ is more commonly used for actual computation. The matrices in the formula are from the SVD; D^+ is the reciprocal of the diagonal matrix D 's nonzero elements then taking the transpose of the resulting matrix.

2.10 The Trace Operator

$$\text{Tr}(A) = \sum_i A_{i,i}$$

The trace operator is useful in that several complex computations may be expressed with the matrix products and the trace operator. One example is the Frobenius norm.

$$\|A\|_F = \sqrt{\text{Tr}(AA^T)}$$

Some other useful identities of the trace operator are as follows :

- $\text{Tr}(A) = \text{Tr}(A^T)$
- $\text{Tr}(ABC) = \text{Tr}(CAB) = \text{Tr}(BCA)$
- That is, $\text{Tr}(\prod_{i=1}^n F^{(i)}) = \text{Tr}(F^{(n)} \prod_{i=1}^{n-1} F^{(i)})$
- Therefore, $\text{Tr}(AB) = \text{Tr}(BA)$
- for scalar a , $\text{Tr}(a) = a$

2.11 The Determinant

For the determinant of square matrix A , it is denoted $\det(A)$. This maps matrices to real scalars, and is equivalent to the product of all the eigenvalues of the matrix. The absolute value of the determinant can be thought of as a measure of how much multiplication by the matrix expands or contracts space.

3 Probability and Information Theory

Here, we describe probability theory and information theory. Probability theory is a mathematical framework for representing uncertain statements. In AI, this theory is used (1) to tell us how AI systems should reason, and (2) to analyze the behavior of proposed AI systems.

3.1 Why Probability?

Nearly all activities require some ability to reason in the presence of uncertainty. Machine Learning is no exception. And in many cases, it is more practical to use a simple but uncertain rule rather than a complex but certain one. There are three possible sources of uncertainty.

1. Inherent stochasticity in the system being modeled. That is, such randomness as in shuffling cards in a game.
2. Incomplete observability.
3. Incomplete modeling.

There are basically two types of probability. One is the **frequentist probability**, which is related directly to the rates at which events occur in repetitive actions, and another is the **bayesian probability**, which is related to qualitative levels of certainty.

3.2 Random Variables

A **random variable** is a variable that can take on different values randomly. We generally denote it with a lowercase letter in plain typeface with lowercase script letters. Random variables can be discrete or continuous.

3.3 Probability Distributions

A **probability distribution** is a description of how likely a random variable or set of random variables is to take on each of its possible states. The distribution largely depends on whether the variables are discrete or continuous.

3.3.1 Discrete Variables and Probability Mass Functions

A probability distribution over discrete variables is described with a **probability mass function (PMF)**. This function maps from a state of a random variable to the probability of it; the probability that $X = x$ is denoted $P(x)$, and is written $x \sim P(x)$ to specify which distribution the variable follows. To obtain a **normalized** distribution, the PMF should suffice $\sum_{x \in X} P(x) = 1$. One example is the uniform distribution where $P(X = x_i) = \frac{1}{k}$.

3.3.2 Continuous Variables and Probability Density Functions

A probability distribution over continuous variables is described with a **probability density function(PDF)**. To be a PDF, a function p must satisfy the following properties:

1. The domain of p must be the set of all possible states of x .
2. $\forall x \in X, p(x) \geq 0$. Note that we do not require $p(x) \leq 1$
3. $\int p(x)dx = 1$

3.4 Marginal Probability

4 Numerical Computation

5 Machine Learning Basics

PART II. Deep Networks: Modern Practices

6 Deep Feedforward Networks³

6.1 Feedforward Net Overview

(Deep) feedforward networks, or multilayer perceptrons (MLPs), aims to approximate some function f , to map an input x to a category y . It is commonly denoted $y = f(x; \theta)$, where θ is the parameters that are learned to best fit the data. The network consists of multiple hidden layers, and each of its dimension is called **width**.

Feedforward nets can be best understood by their nonlinear capacity. To overcome limitations of linear functions, the feedforward nets utilize a nonlinear transformation function $\phi(x)$. Here, the question is how to choose the mapping ϕ :

1. Use a very generic ϕ , such as the infinite-dimensional ϕ used in the RBF kernel. However, the generalization ability remains poor.
2. Manually Engineer ϕ . Requires domain expertise.
3. Learn ϕ . This is largely used in Deep Feedforward nets, but gives up on the convexity of the training problem. However, we may take a generic approach through the broad $\phi(x; \theta)$, and may enable designing ϕ .

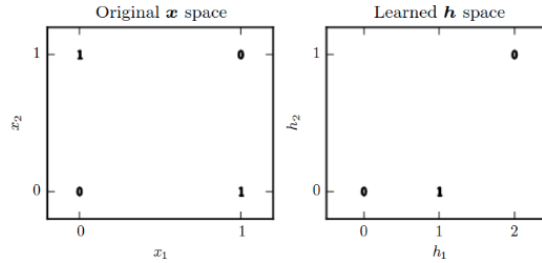
6.2 Example : Learning XOR

· **XOR (exclusive or)**: operation on two binary values, which returns 1 only when exactly one of the binary values is 1.

Let's regard the XOR as a regression problem and define an MSE loss function for a linear function f^* .

$$f(x; \theta) = f(x; w, b) = x^T w + b, J(\theta) = \frac{1}{4} \sum_{x \in \mathbb{X}} (f^*(x) - f(x; \theta))^2$$

If you fit the network, you'll get a model that has $w = 0, b = \frac{1}{2}$. The reason of this is because the XOR x space cannot be linearly separated.



Now, let's fit a model with a hidden layer $f^{(1)}(x) = h = g(W^T x + c)$ and an output layer of $f(h) = h^T w + b$, where function g is the rectified linear unit (ReLU)⁴. Here, fitting a feedforward network of multiple layers without a nonlinear function is equivalent to fitting a single layered network. That is, a linear feedforward network can be represented with a single linear function. So the final model can be defined as follows.

$$f(x; W, c, w, b) = w^T \cdot \max(0, W^T x + c) + b$$

³Largely referenced [leejunhyun.github.io](https://github.com/leejunhyun)

⁴ReLU : $g(x) = \max(0, x)$

6.3 Gradient-Based Learning

The main difference between Neural Networks and other machine learning algorithms that utilize gradient descent is that the nonlinear functions make the loss function non-convex. Thus, neural networks require a stochastic process that enhances the parameter step by step with gradients. This makes the neural network sensitive to parameter initialization.

6.3.1 Cost Functions

In most cases, we define the model distribution $p(y|x;\theta)$, and use the MLE(Maximum Likelihood Estimation). That is, the cross entropy loss between the data label and the model output is used as the cost function.

6.3.1.1 Learning Conditional Distributions with Maximum Likelihood

Most modern neural networks are trained using maximum likelihood. The cost function is then given as follows. The specific form of the cost function changes from model to model, depending on the specific form of $\log p_{model}$.

$$J(\theta) = -\mathbb{E}_{x,y \sim \hat{p}_{data}} \log p_{model}(y|x)$$

One important point to keep in mind is that the gradient of the cost function must be large enough not to be saturated(become very flat). Also, in order to prevent the cross entropy from approaching negative infinity, regularization should be considered as well.

6.3.1.2 Learning Conditional Statistics

We may want to learn just one conditional statistic of y given x , instead of a full probability distribution $p(y|x;\theta)$. For example, we may have a predictor $f(x;\theta)$ that we wish to employ to predict the mean of y . In this case, we may view the cost function as a **functional**⁵, rather than a function. That is, we may fit a cost functional to have its minimum value where x maps to y . In order to solve this optimization problem, a mathematical tool called **calculus of variations** is required. This will be further discussed in section 19.4.2.

6.3.2 Output Units

The output units is closely related with the choice of cost functions. The Output layer's role is to alter the feature to an appropriate form for the task.

6.3.2.1 Linear Units for Gaussian Output Distributions

This unit adds no nonlinearity to the output layer. These layers are often used to produce the mean of a conditional Gaussian distribution $p(y|x) = N(y; \hat{y}, I)$. Thus, maximizing the log-likelihood is equivalent to minimizing the MSE. One advantage of the linear units is that they do not saturate, thereby alleviating the difficulties in gradient based optimization.

6.3.2.2 Sigmoid Units for Bernoulli Output Distributions

Many tasks require predicting the value of a binary variable y . The MLE approach is to define a Bernoulli distribution over y conditioned on x . The Bernoulli distribution is defined as a single probabilistic number $P(y = 1|x)$ between 0 and 1. Instead of a linear approach, we add a sigmoid function to the linear unit : $\hat{y} = \sigma(z) = \sigma(w^T h + b)$. The computed z is called the **logit**. Using z , we define a probability distribution over y . First, we construct an unnormalized probability distribution $\tilde{P}(y)$. Then, it is normalized to obtain the sigmoid function $P(y)$ as follows.

$$\log \tilde{P}(y) = yz, \quad \tilde{P}(y) = \exp(yz)$$
$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^1 \exp(y'z)} = \frac{1}{1 + \exp(-z)}, \quad P(y) = \sigma((2y - 1)z)$$

This probability distribution in log space is apt for maximum likelihood learning as well. If we set $J(\theta) = -\log P(y|x)$ to undo the sigmoidal effect.

⁵a functional is the mapping of a function to a real value.

6.3.2.3 Softmax Units for Multinoulli Output Distributions

7 Regularization for Deep Learning