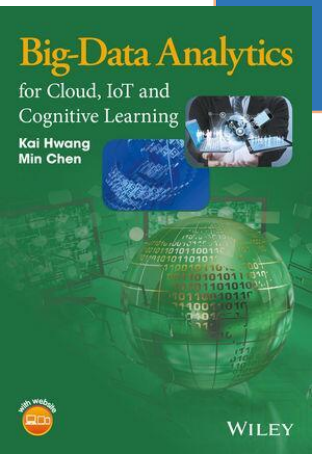
Big Data Analytics for Cloud, IoT and Cognitive Computing

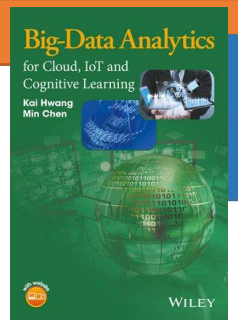
Part 2 Machine Learning and Deep Learning Algorithms

Chap. 6 Deep Learning with Artificial Neural Networks



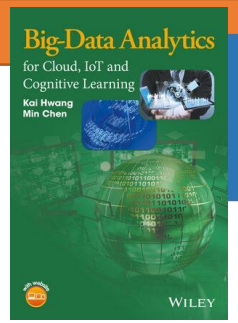
Big-Data Analytics for Cloud, IoT and Cognitive Computing, First Edition. Kai Hwang and Min Chen.

Companion Website: http://www.wiley.com/go/hwangIOT



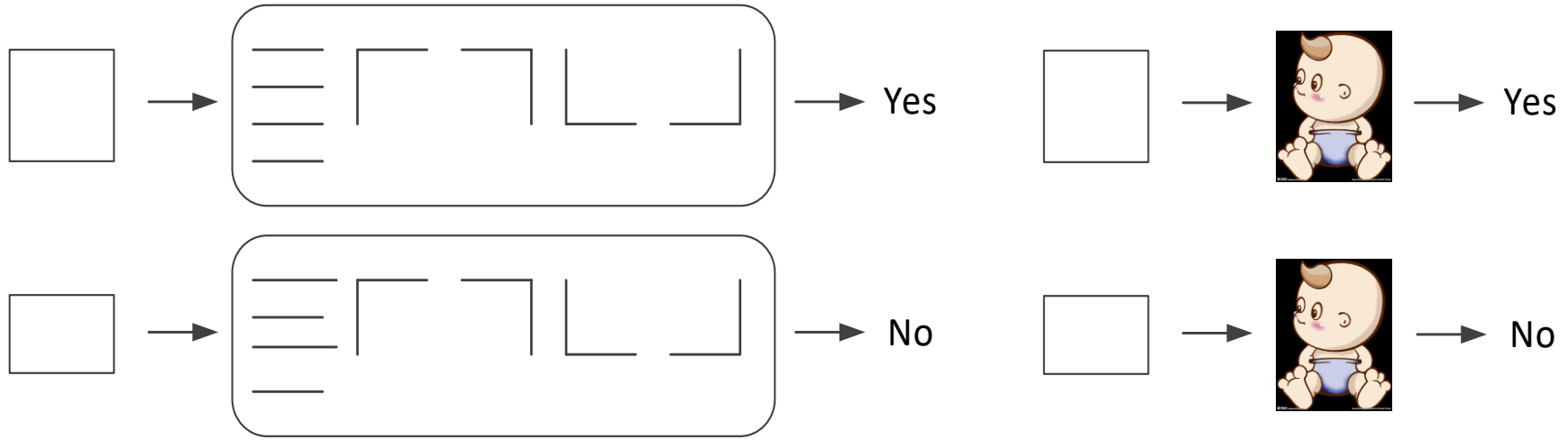
Deep Learning Introduction

- Deep learning simulates the operations in layers of artificial neural networks. This is heavily used to extract and learn features from data.
- Common deep learning architectures
  - The basic ANN
  - Convolutional neural networks
  - Deep belief network
  - Recurrent neural network



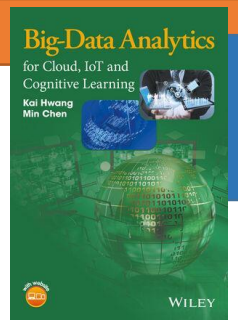
Deep Learning Introduction

How can deep learning complete with humans on self-learning through education? If we want to judge whether a quadrangle is square or not, the rationale approach is to seek features of square, such as same length for four sides and four 90 degree corner angles. This requires comprehension of the concept of right angle and the length of the side view as demonstrated below:



(a) Rationale mode *(b) Perceptron mode*

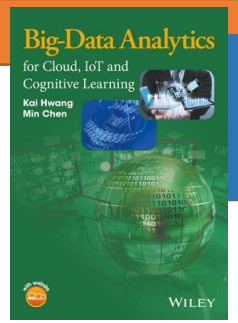
Figure 6.1 Rationale versus perceptron modes to recognize a square object.



Deep Learning Human Senses

DeepMind and Brain Projects at Google in the Past 5 Years

- In March 2016, Google AlphaGo program defeated a top Go player. This has opened up the debate between human vs. machine intelligence.
- Another reported advance is the Google Brain Project. In June 2012, tens of millions of random images from YouTube were recognized by a computer platform built over 16,000 CPU cores at Google.
- They use a training model over a deep neural network built with 1 billion artificial neurons. This model system identified basic features of images and succeeded in recognizing a cat out of thousands of classes of images.
- The project leader Andrew Ng once said: “We directly put massive data (images) into the artificial system and the system learned (remembered) from the key features of those images, automatically.”



Deep Learning Human Senses

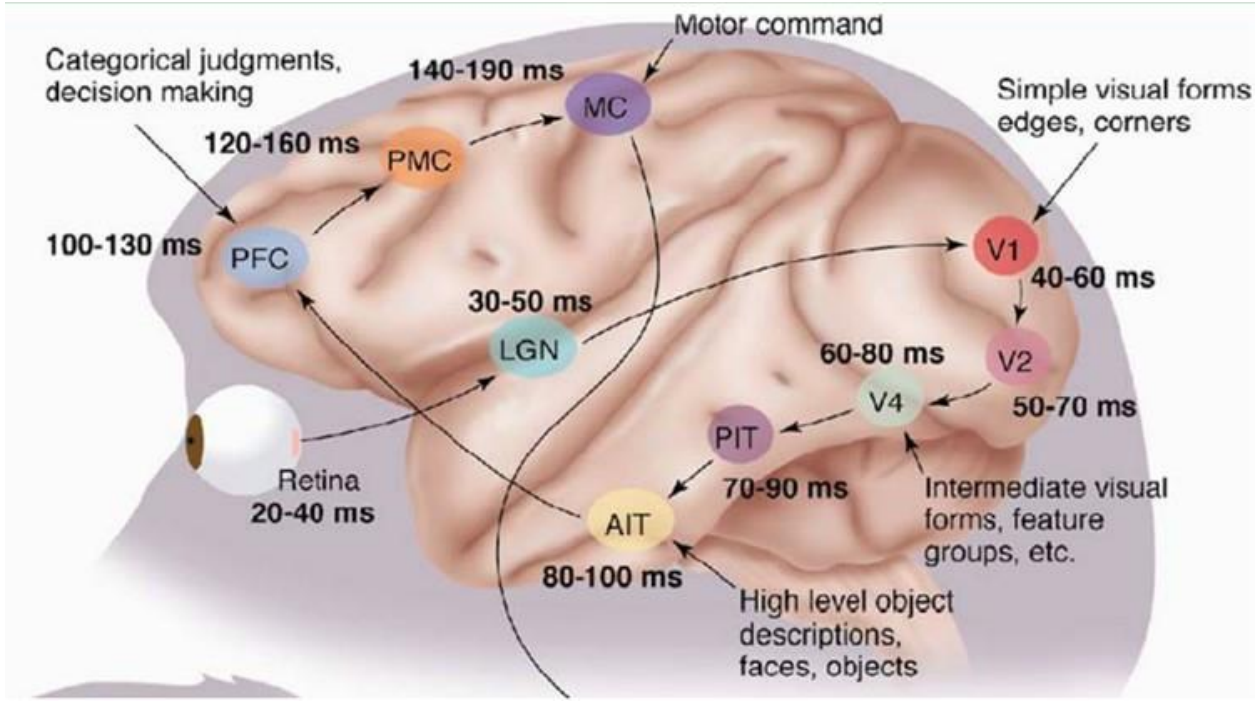
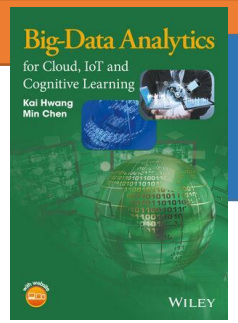


Figure 6.2: Schematic diagram showing the hierarchy of human visual cortex

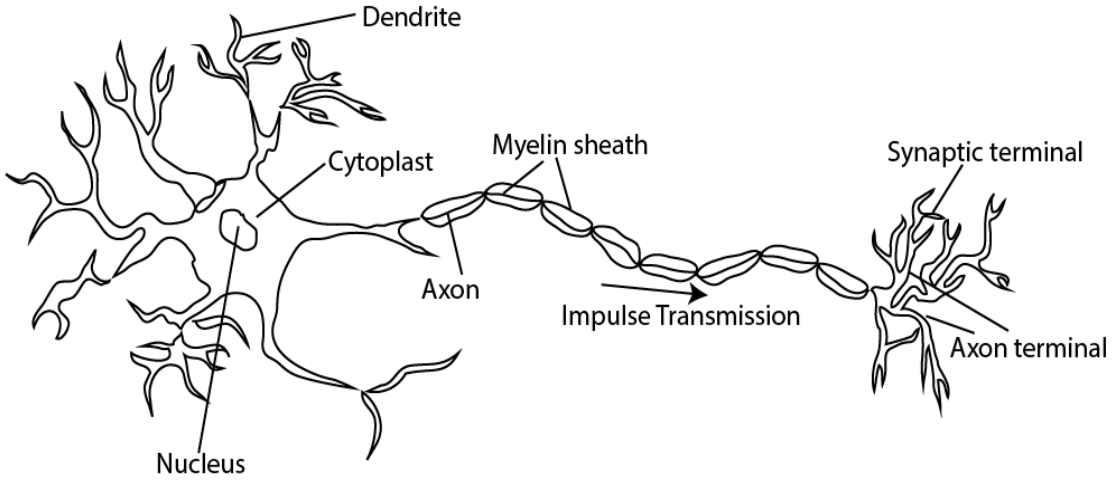
The main contribution of David Hubel and Torsten Wiesel (winners of 1981 Nobel Prize in medicine) is that they found out that information processing of the visual system (i.e., visual cortex) is hierarchical, as shown in Fig. 6.2. In 1958, they were studying the correspondence between the pupil area and neurons in the cerebral cortex at John Hopkins University. By many experiments, it has been proven that there is some kind of correspondence between stimulation received by the the pupils and different visual neurons located in the posterior cerebral cortex. They found a kind of neuron that is named "orientation selective cell". When the pupil captures the edge of an object, and if this edge points to a certain direction, the corresponding neurons would be active.



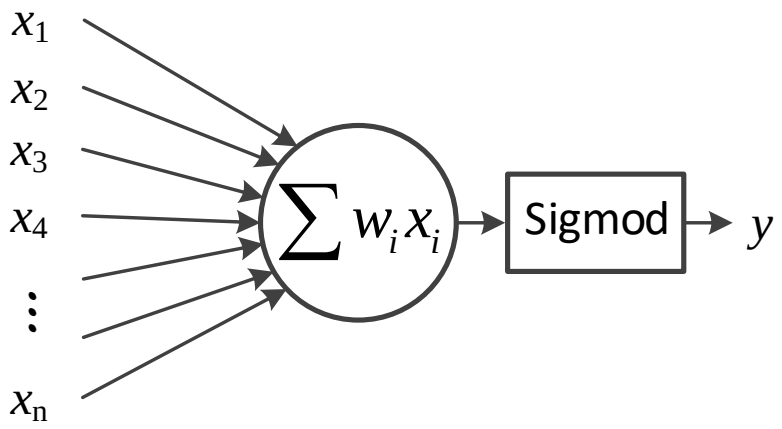
Biological Neurons versus Artificial Neurons

The human brain has a very complicated structure. But its constitutional unit is the neuron, that produces output (excitement) per input. The hierarchical information processing of the human brain is realized through numerous neurons that are interconnected. A *biological neuron* is modeled in Fig. 6.3(a).

The neuron obtains input from upper layer neurons, produces output and transmits it to the neuron in the next layer. If the human brain can be simulated, neurons should be simulated first. Fig. 6.3(b) shows the structure of an *artificial neuron*.

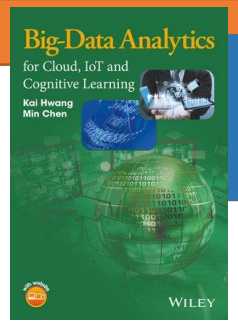


(a) Biological neuron in human brain



(b) Structure of an artificial neuron

Figure 6.3 Schematic diagrams of biological neuron versus artificial neuron



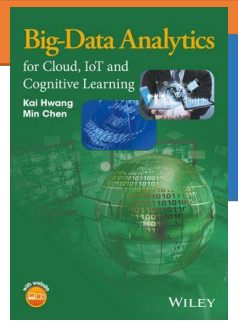
Biological Neurons versus Artificial Neurons

In 2006, an article by Hinton in *Science* inspired a new era of deep learning. *Artificial* neural networks with multiple self-learning hidden layers were suggested in that article. Each hidden layer includes several neurons. The output of the previous layer is made as the input of the next layer. The structure of such a deep neural network first adopts an artificial neuron to simulate a biological neuron in the human brain. Then, the layer-wise learning structure of the deep network simulates a hierarchical structure of information processing by the human brain.

The main conclusions in that article are:

1) The learning capability to obtain features by a deep artificial neural network with multiple hidden layers is strong. What is more, the features obtained by progressive learning in multiple layers can represent data accurately;

2) A layer-wise pre-training method solves difficulty in training ANN. Meanwhile, unsupervised learning is adopted during layer-wise pre-training.



Biological Neurons versus Artificial Neurons

As shown in Figure 6.4, the regions V1 and H1 compose the first layer with V1 as input and H1 as output, where the data in V1 came from original data. V2 and H2 compose the second layer. The input of V2 comes from the output of H1, while H2 is the output in second layer.

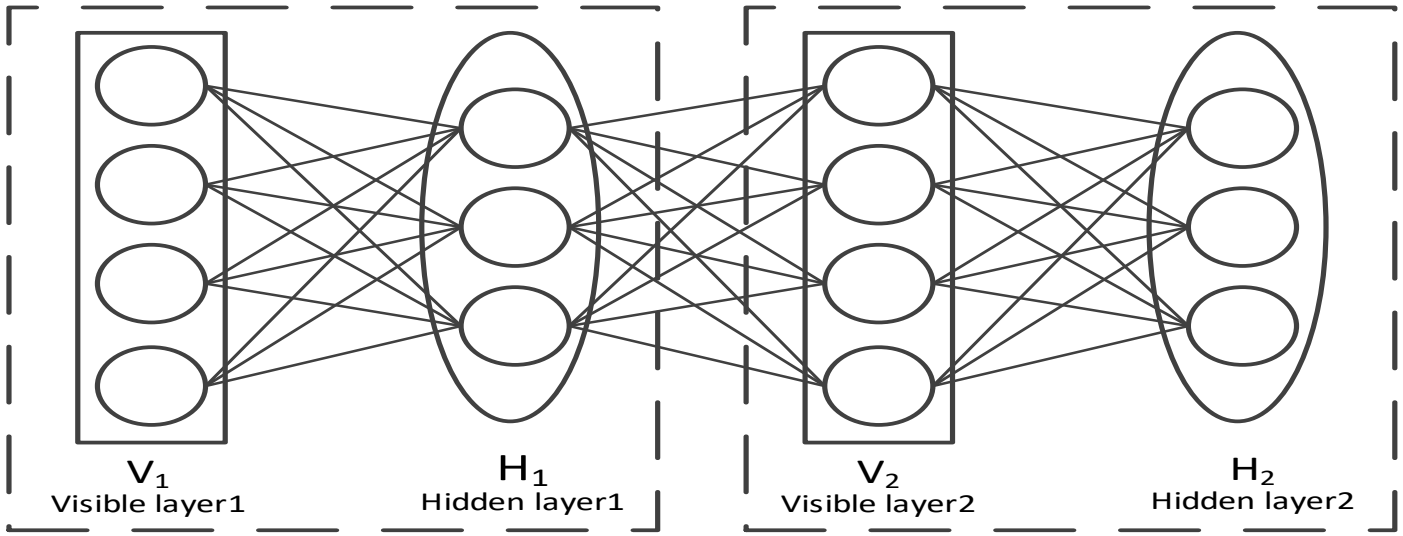
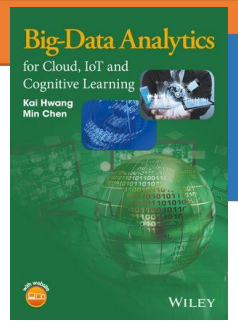
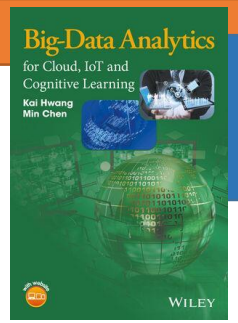


Figure 6.4 Concept of deep learning with an ANN having two hidden layers



Deep Learning Versus Shallow Learning

- **A deep neural network (DNN)** is an artificial neural network (ANN) with multiple hidden layers of units between the input and output layers.
- DNNs can model more complex non-linear relationships than shallow neural networks.
- In order to recognize simple patterns, basic classification tools based on shallow learning, for example decision trees, SVM, etc., are good enough. With the increase in number of input features, ANN starts to exhibit its superior performance.



Deep Learning Versus Shallow Learning

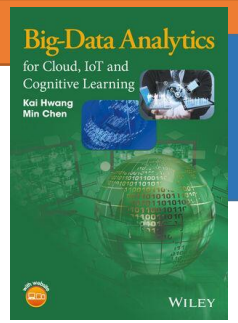
- Deep learning differs from shallow learning in four aspects:

a) It emphasizes the depth of the ANN structure. Compared to conventional shallow learning, more hidden layers are utilized in deep learning.

b) The importance of feature learning is highlighted. By the use of layer-wise feature transformation, data features from original space are represented by the features in a new feature space. With this method, classification or prediction becomes easier and more accurate.

c) Deep learning derives from ANN. However, their training models are different. The layer-wise training is adopted in deep learning, which solves the problem of the vanishing gradient.

d) Plenty of data are utilized to learn the features in deep learning, which is not a must in shallow learning.



Artificial Neural Networks (ANN)

- Single Layer Artificial Neural Networks

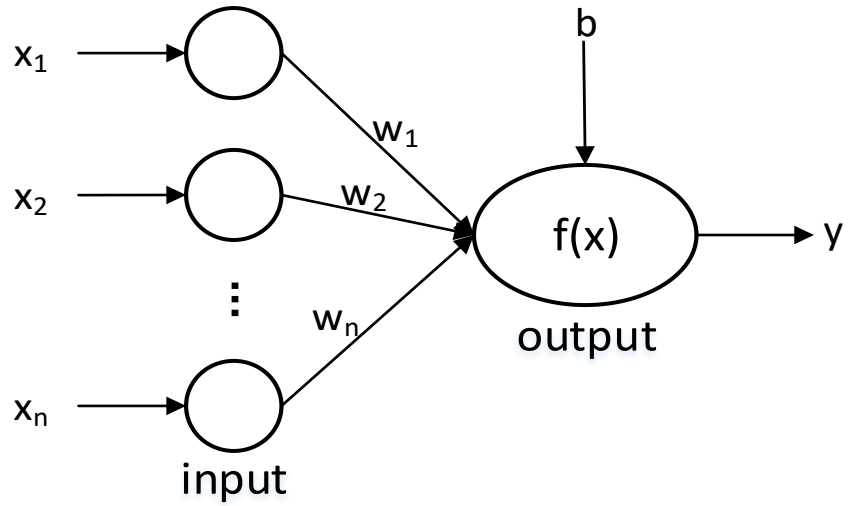
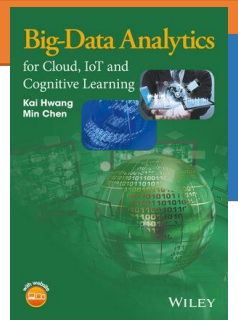


Figure 6.6 Conceptual diagram of a perceptron machine

The perceptron is the simplest ANN, which includes an input layer and an output layer without a hidden layer. The input layer node is used for receiving data, while the output layer node yields output data. Since we simulate perceptron as the human nervous system, the input node corresponds to input neuron and the output node corresponds to decision-making neuron while weight parameter corresponds to strength of connection between neurons.



Artificial Neural Networks (ANN)

- Single Layer Artificial Neural Networks

Input: $x = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \rightarrow y = f(x)$

Output: $\hat{y} = f(w \cdot x)$

Sigmoid function: $\text{sig mod}(x) = \frac{1}{1+e^{-x}}$

hyperbolic tangent function: $\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

If we expect good result, appropriate weights are needed. Therefore, the value of weight must be adjusted dynamically during training.

The weight renewal equation:

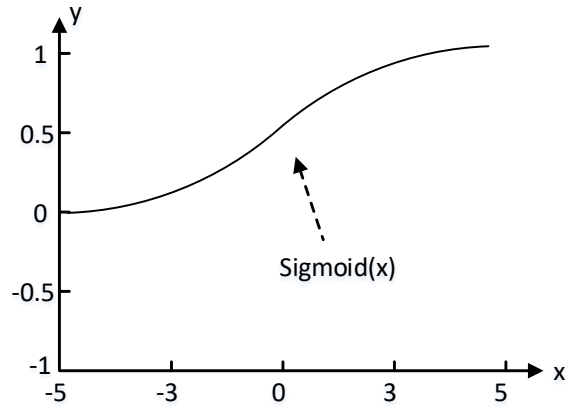
$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij} \quad j = 1, 2, \cdots n$$

$w_j^{(k)}$ *is the weight value of input node j after iterations of k times*

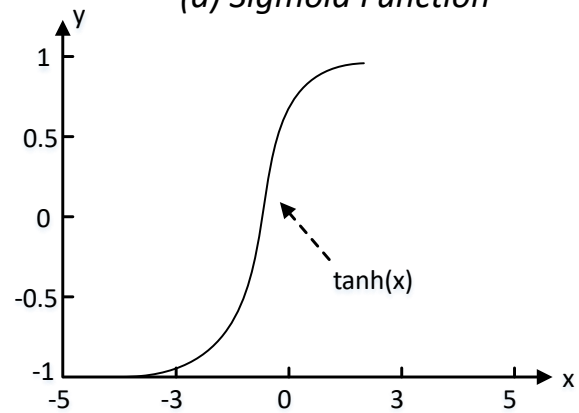
x_{ij} *is the input value of node j in the ith training data sample.*

λ *is known as* learning rate, and is within the interval of [0, 1].

If λ is close to 0, the new weight value is mainly influenced by old weight value. If λ is close to 1, the new weight value is mainly influenced by current adjustment amount.

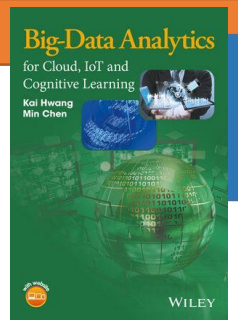


(a) Sigmoid Function



(b) Hyperbolic Tangent Function

Figure 6.7 Common activation functions for the perceptron.



Artificial Neural Networks (ANN)

Multilayer Artificial Neural Network

A multilayer artificial neural network is composed of one input layer, one or more hidden layer(s) and one output layer, as shown in Fig. 6.8.The main unit of ANN is a neuron.

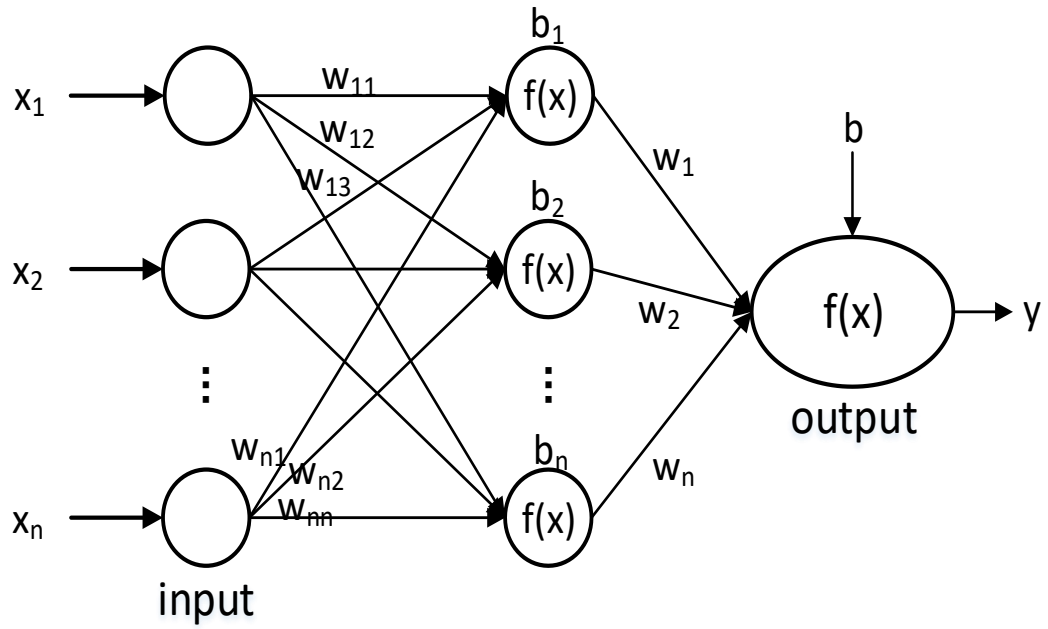
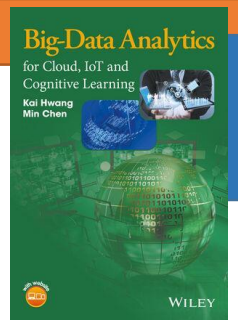


Figure 6.8: Structure of a multilayer artificial neural network



Artificial Neural Networks (ANN)

Multilayer Artificial Neural Network

There are three basic elements of a neuron:

A group of connections.The connection strength is expressed by weight value of each connection. The weight value represents activation if it is positive while it represents suppression when the value is negative. The mathematical equation is

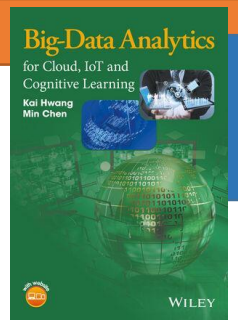
$$\begin{cases} w = (w_1,\ w_2,\ \cdots,\ w_n) \\ w_i = (w_{i1}, w_{i2}, \cdots, w_{in}) \quad i = 1, 2, \cdots n \end{cases}$$

One summation unit. It is used to work out the weighted sums (linear combination) for each input signal, and generally together with an offset or threshold. Its mathematical equation is

$$\begin{cases} \mu_k = \sum_{j=1}^{n} w_{kj} x_j \\ v_k = \mu_k + b_k \end{cases}$$

One nonlinear activation function. It plays a role of nonlinear mapping and restricts the output amplitude of neuron within a certain range [often (0, 1) or (–1, 1)]. Its mathematical equation is

$$y_k = f(v_k)$$



Artificial Neural Networks (ANN)

Four steps to generate an ANN model are given below

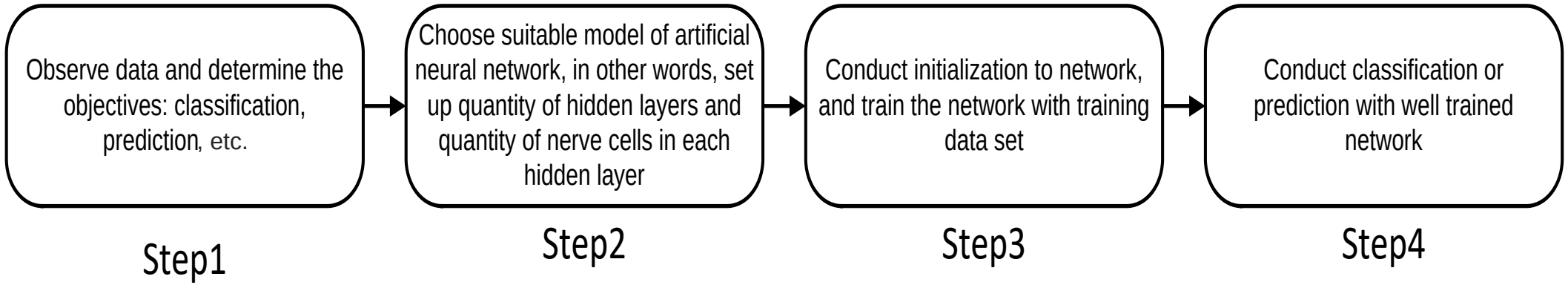
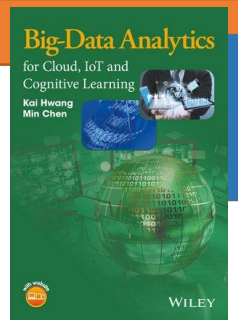


Figure 6.9: General steps for modeling an artificial neural network



Artificial Neural Networks (ANN)

Forward Propagation

The forward propagation starts from the input layer towards a hidden layer. As for hidden unit i, its input is h_i^k, h_i^k stands for the input of hidden unit i in layer k, while b_i^k stands for the offset of hidden unit I in layer k. The corresponding output state is

$$h_i^k = \sum_{j=1}^{n} w_{ij} x_j + b_i^k \rightarrow H_i^k = f(h_i^k) = f(\sum_{j=1}^{n} w_{ij} x_j + b_i)$$

For convenience, we often let $x_0 = b$, $\omega_{i0} = 1$. Then the equation of forward propagation from hidden units of layer k to layer k + 1 is

$$\begin{cases} h_i^{k+1} = \sum_{j=1}^{m_k} w_{ij}^k h_j^k \\ H_i^{k+1} = f(h_i^{k+1}) = f(\sum_{j=1}^{n} w_{ij}^k h_j^k) \end{cases} \quad i = 1, 2, \cdots, m_{k+1}$$

m_k stands for quantity of neurons in hidden units of layer k;

w_{ij}^k stands for weight vector matrix from layer k to layer k+1.

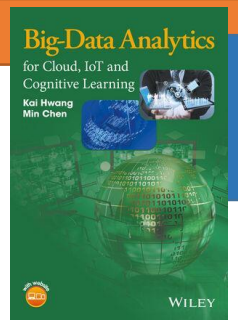
The final output is obtained as

$$O_i = f(\sum_{j=1}^{m_{M-1}} w_{ij}^{M-1} H_j^{M-1}) \quad i = 1, 2, \cdots, m_o$$

m_o stands for the number of output units

M stands for total number of layers of artificial neural network

O_i stands for the output of output unit *i.*



Artificial Neural Networks (ANN)

Example 6.1: Forward Propagation based Output Prediction in ANN

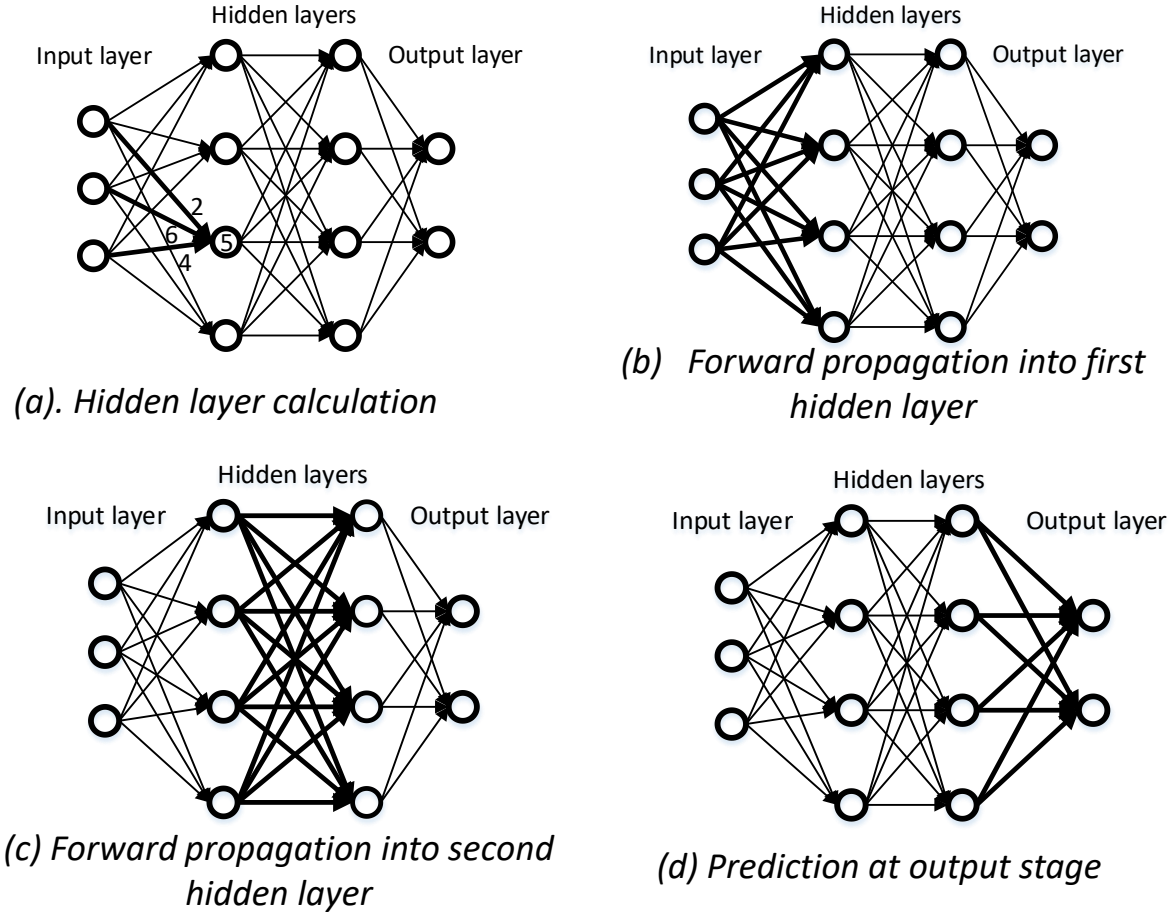
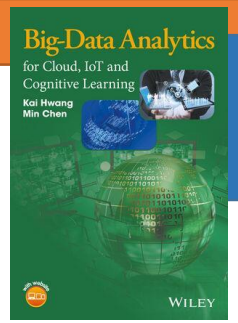


Figure 6.10: Forward Propagation based output prediction in a simple ANN

In an ANN, each set of inputs are modified by unique weights and biases. As shown in Fig. 6.10(a), when calculating the activation of the third neuron in the first hidden layer, the first input was modified by weight of 2, the second by 6, the third by 4, and then bias of 5 is added on the top. Each activation is unique because each edge has unique weight and each node has unique bias.



Artificial Neural Networks (ANN)

Backward Propagation

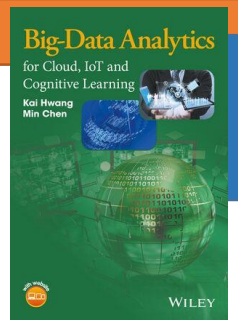
The symbols i and s represent the output result of output unit i if the training sample is denoted as s. O_i^s stands for the output result of output unit i if the training sample is s. Thus, the problem of finding a group of appropriate weights naturally comes down to the problem that E(W) reaches a minimum by figuring out appropriate values of W, as shown below:

$$E(W)=\frac{1}{2}\sum_{i,s}(T_i^s-O_i^s)^2=\frac{1}{2}\sum_{i,s}(T_i^s-f(\sum_{j=1}^{m_{M-1}}w_{ij}^{M-1}H_j^{M-1}))^2 \rightarrow \min \quad E(W) \qquad i=1,2,\cdots,m_o$$

As for each variable ω_{ij}^k , this is a continuously differentiable nonlinear function. In order to work out the minimum, we generally adopt the steepest descent method. As per this method, we constantly renew weight in the direction of the negative gradient until the conditions set up by the customer are satisfied. The so-called direction of gradient is to work out the partial derivative of function: Assume the weight is ω_{ij}^k after renewal of k times. If ∇E(W) ≠ 0, then the renewed weight at time k + 1 times is expressed by

$$\nabla E(W)=\frac{\partial E}{\partial w_{ij}^k}\rightarrow w_{ij}^{(k+1)}=w_{ij}^{(k)}-\eta\nabla E(w_{ij}^{(k)})$$

η is the learning rate of that network, When ∇E(W) = 0 or ∇E(W) < ε (ε is permissible error), it stops renewing.



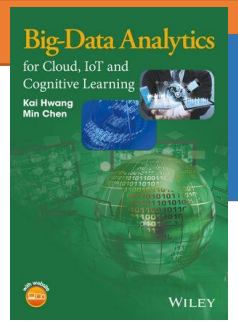
Artificial Neural Networks (ANN)

Example 6.2: hyperlipemic Diagnosis Using an Artificial Neural Network

For instance, Table 6.1 is a data set of triglyceride, high-density lipoprotein, low-density lipoprotein and whether hyperlipemic or not (1 for yes and 0 for no) in health examination data of some people in a grade-A hospital of second class in Wuhan City. Let's attempt to conduct preliminary judgment on whether the person who received a health examination has hyperlipemic if his or her health examination data are {3.16, 5.20, 0.97, 3.49} in sequence.

Table 6.1: Patient Examination Data for Those Suspected to Have Hyperlipemic

Patient ID	Triglyceride (mmol/L)	Total Cholesterol (mmol/L)	High-Density Lipoprotein (mmol/L)	Low-Density Lipoprotein (mmol/L)	Hyperlipemic or not
1	3.62	7	2.75	3.13	1
2	1.65	6.06	1.1	5.15	1
3	1.81	6.62	1.62	4.8	1
4	2.26	5.58	1.67	3.49	1
5	2.65	5.89	1.29	3.83	1
6	1.88	5.4	1.27	3.83	1
7	5.57	6.12	0.98	3.4	1
8	6.13	1	4.14	1.65	0
9	5.97	1.06	4.67	2.82	0
10	6.27	1.17	4.43	1.22	0
11	4.87	1.47	3.04	2.22	0
12	6.2	1.53	4.16	2.84	0
13	5.54	1.36	3.63	1.01	0
14	3.24	1.35	1.82	0.97	0



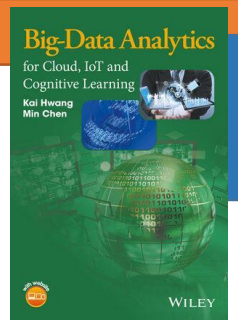
Artificial Neural Networks (ANN)

Example 6.2: Hyperlipemia Diagnosis Using an Artificial Neural Network

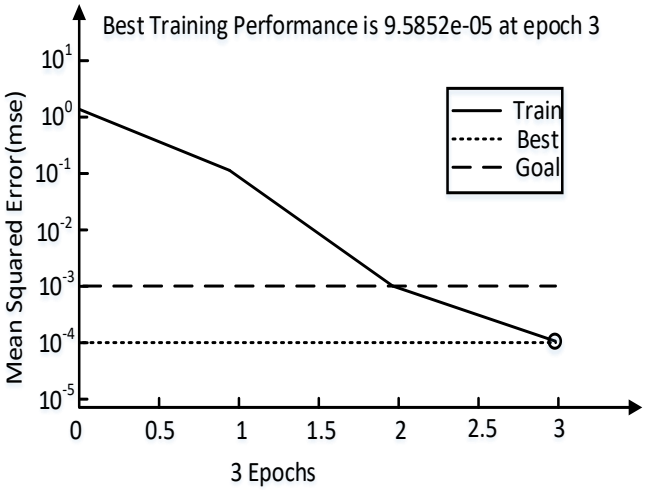
From the data in the table, it is known that this problem is a dichotomy problem ("1" for hyperlipemia or "0" for healthy) with four attributes. Therefore, we can conduct prediction and classification using the artificial neural network. As there are not enough sample data for training in this case, it is not necessary to set up too many hidden layers and neurons. Here 1 hidden layer is set up, and the quantity of neurons in each layer is 5. Tansig function is chosen as an activation function between input layer and hidden layer. While the purelin function is chosen as the function between hidden layer and output layer (choosing other functions has little effect on the results of this case.). Its network parameters are listed in Table 6.2.

Table 6.2 Table of parameters for artificial neural network.

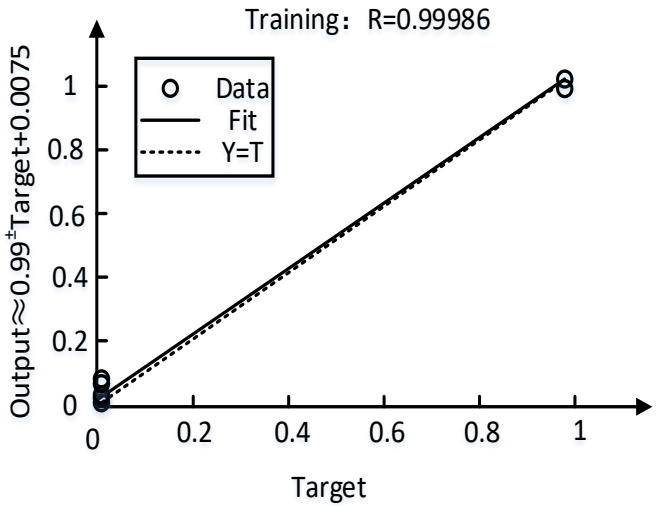
Neurons at input layer	Hidden layers	Neurons in hidden layers	Neurons in output layer
4	1	5	1
Permissible error	**Times for training**	**Learning rate**	**Activation function**
10-3	10000	0.9	Tansigand purelin



Artificial Neural Networks (ANN)



(a) Training error

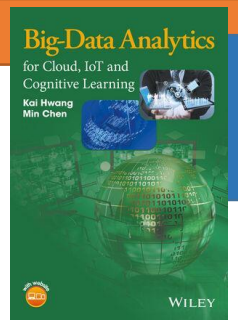


(b) Training results

Figure 6.11 Training error and results on the ANN in Example 6.3.

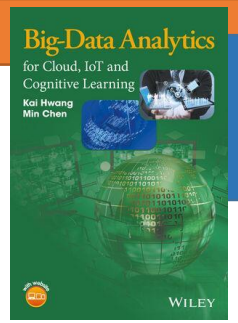
Then we train the network with the data in Table 6.1 above. MATLAB is used for programming. The training process of the network is shown in Figure 6.11(a). The error between actual output of the network in the training process and ideal output is reduced gradually. A satisfactory state is reached after the second backward propagation. The final training results are shown in Figure 6.11(b). The neural network classifier divides the training data into two categories.

The training data are divided at both ends and form two classes. The class 0 stands for healthy while the class 1 stands for hyperlipemia. The results of classification are {1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0}. The accuracy for classification reaches 100%, so this network can be used for prediction. Finally, let us predict whether a person whose data are *{3.16, 5.20, 0.97, 3.49} respectively has hyperlipemia or not, with the artificial neural* network mentioned above. The result is *class = 1. Therefore, we can predict the person* who has hyperlipemia.



AutoEncoder

- What is AutoEncoder
  - The AutoEncoder can be treated as a special ANN. There are sample data but no corresponding label data in this stage of network training.
  - AutoEncoder extracts output data to reconstruct input data and compare it with original input data.
  - After many iterations, the value of the objective function reaches its optimality, which means that the reconstructed input data is able to approximate the original input data to a maximum extent.
  - AutoEncoder is a self-supervised learning tool and belongs to unsupervised training.

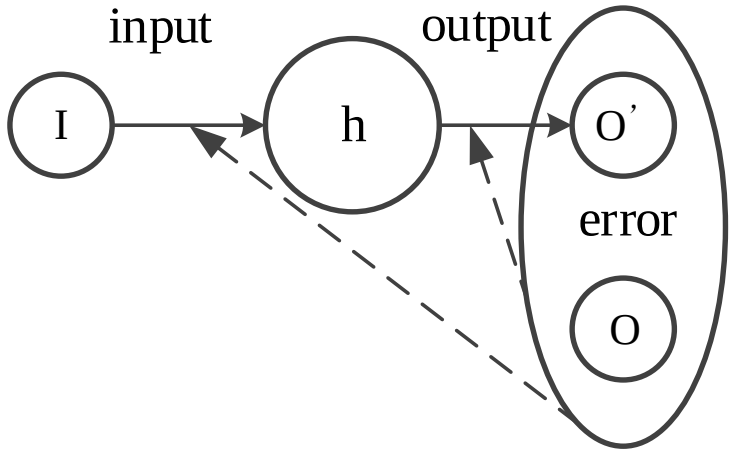


AutoEncoder

Supervised learning in an ANN versus self-supervised learning in AutoEncoder

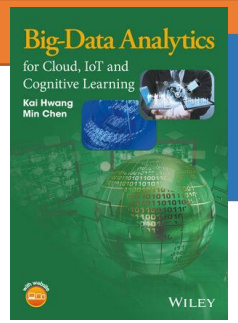
With the supervised learning method, all input data (samples) of neutral network training correspond to their expected values (labels). The difference between the current output value and expected value is used to adjust parameters at all input and output layers. The error will reach the minimum after many iterations.

Figure 6.12(a) shows the workflow structure of a supervised neutral network, including the input layer, one hidden layer and the output layer. The output value O' is derived from input data I through the hidden layer h. O means the label of I. The error between O' and O is calculated. Then, backward propagation is used to adjust the network parameters. With the increasing number of iterations, the error is reduced until a minimized value is obtained.



(a) Supervised learning neutral network

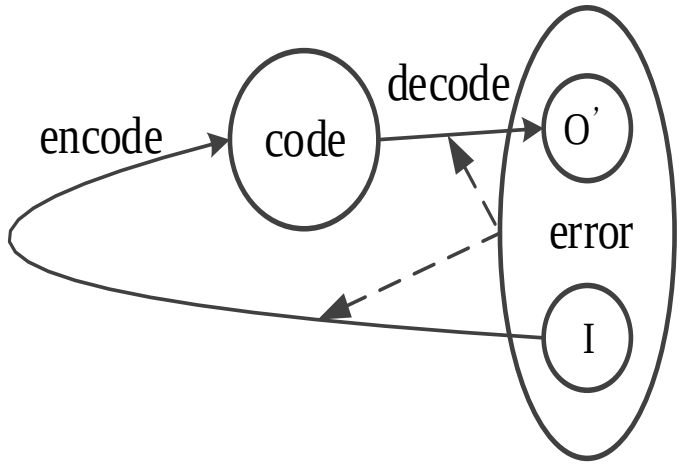
Figure 6.12 Supervised learning in an ANN versus self-supervised learning in AutoEncoder



AutoEncoder

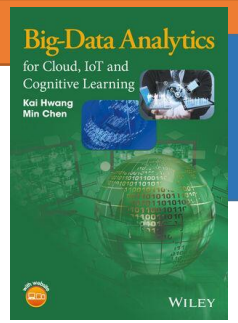
Supervised learning in an ANN versus self-supervised learning in AutoEncoder

Figure 6.12(b) shows the workflow of self-supervised learning in the AutoEncoder. The input data I represents the m-dimension vector *I ∈ Rm. Self-supervised learning* of AutoEncoder consists of two parts: encoder and decoder. By the encoding process, *code, which is an n-dimension vector code ∈ Rn, is calculated. By the decoding process,* O′ is reconstructed, where O′∈ *Rm represents an m-dimension vector. Then, the error* between O′ and I is obtained. By the use of a stochastic gradient descent, the parameters of encoder and decoder are adjusted for error reduction. When the error between I and O′ is minimized, code can be considered as the incarnation of I. In other words, code is a kind of feature representation extracted from I.



(b) AutoEncoder self-supervised learning

Figure 6.12 Supervised learning in an ANN versus self-supervised learning in AutoEncoder

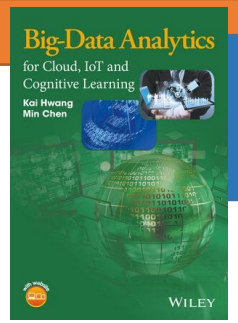


AutoEncoder

- Algorithm 6.1 : Construction of An AutoEncoder
  - Input:

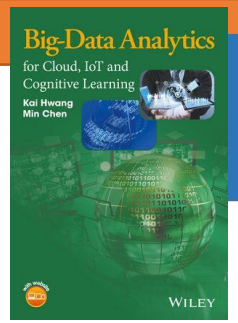
    T: Sample set;
  - Output:

    Feature representation of input data
  - Procedure:
    1. Initialize parameter set $\theta = \{w_1, w_2, b_1, b_2\}$
    2. Encode: Calculate the representation of the hidden layer
    3. Decode: Use the representation of the hidden layer to reconstruct input
    4. Calculate $L_{recon}(I, O')$ and objective function value $J(\theta)$
    5. Judge whether or not $J(\theta)$ meets end conditions?
    6. If yes, return 7), else return 6).
    7. Modify the parameter set with backward propagation and return 2);
    8. End



AutoEncoder

- The brief algorithm of AutoEncoder includes three main steps
  - **Step 1 – Encode:** Convert input data I into *code of the hidden layer by code = f (I) =s1(w1 · I + b1), where w1 ∈ Rm×n and b1 ∈ Rn. S1 is an activate function. The sigmoid* or hyperbolic tangent function can be used.
  - **Step 2 – Decode:** Based on the above code, reconstruct input value I by equation *g(code) = s2(w2 · code + b2), where w2 ∈ Rn×m and b2 ∈ Rm. The activate function* S2 is the same as S1.
  - **Step 3 – Calculate square error:** *Lrecon(I,O′) =‖I − O′‖2, which is the error cost* function. Error minimization is achieved by optimizing Σ the objective function: $J(\theta) = \sum_{I \in D} L(I, g(f(I))\theta = \{w_1, w_2, b_1, b_2\}$



AutoEncoder

Through AutoEncoder, feature representation is obtained to represent original input by code. For the purpose of classification, we still need a classifier. The classifier is connected with the hidden layer of AutoEncoder. Both SVM or softmax can be used as the classifier. The supervised training method such as stochastic gradient descent can be used to train the classifier. The code obtained from AutoEncoder works as input to the classifier, while its output (Y' in Figure 6.16) is compared to labeled data (Y) for supervised fine tuning.

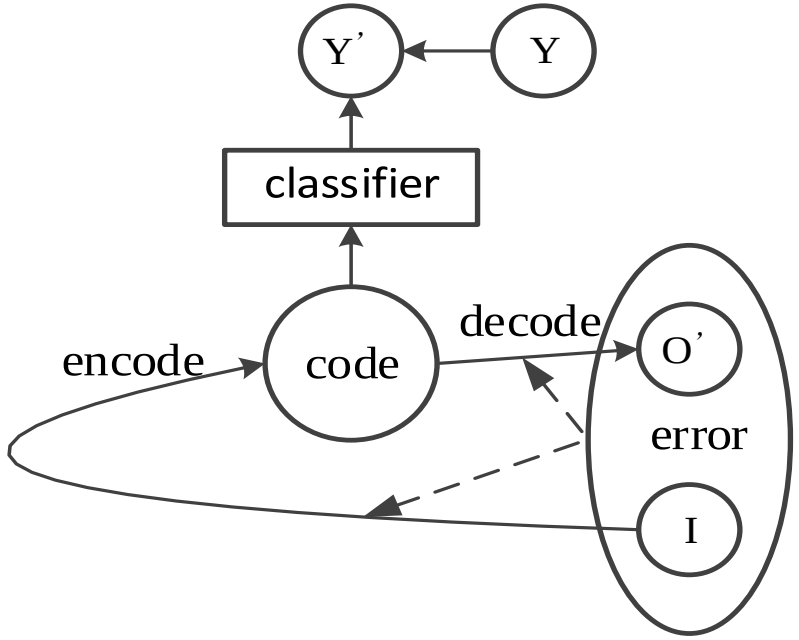
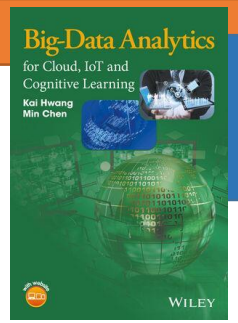


Figure 6.16: Classification process by an autoEncoder



Stacked AutoEncoder

What is Stacked AutoEncoder

- Stacked AutoEncoder (SAE) is a neutral network with several hidden layers between the input layer and the output layer, while each hidden layer corresponds to an AutoEncoder.

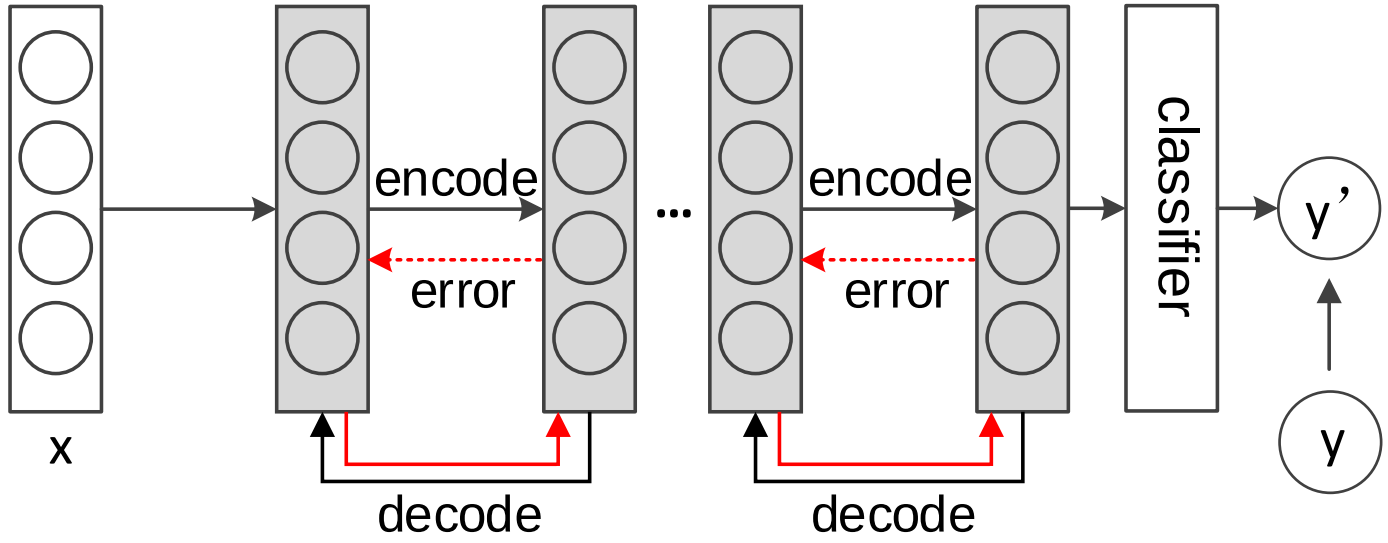
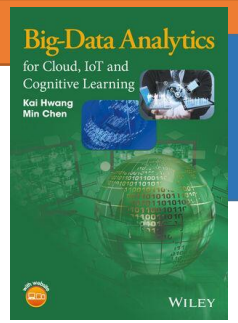
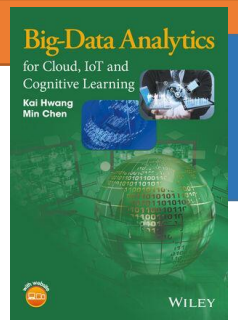


Figure 6.17: Structure of a stacked AutoEncoder



Multi-layer AutoEncoder

- The original input data are used in the first AutoEncoder layer.
- Go through several rounds of encoding and decoding.
- Finally obtain code of the first layer, which represents the feature of the original input data.
- After constructing network structure of the first AutoEncoder layer, the second AutoEncoder layer is trained in the same way. But the difference is the output code of the first layer is treated as the input data of the second layer.



Supervised Fine Tuning

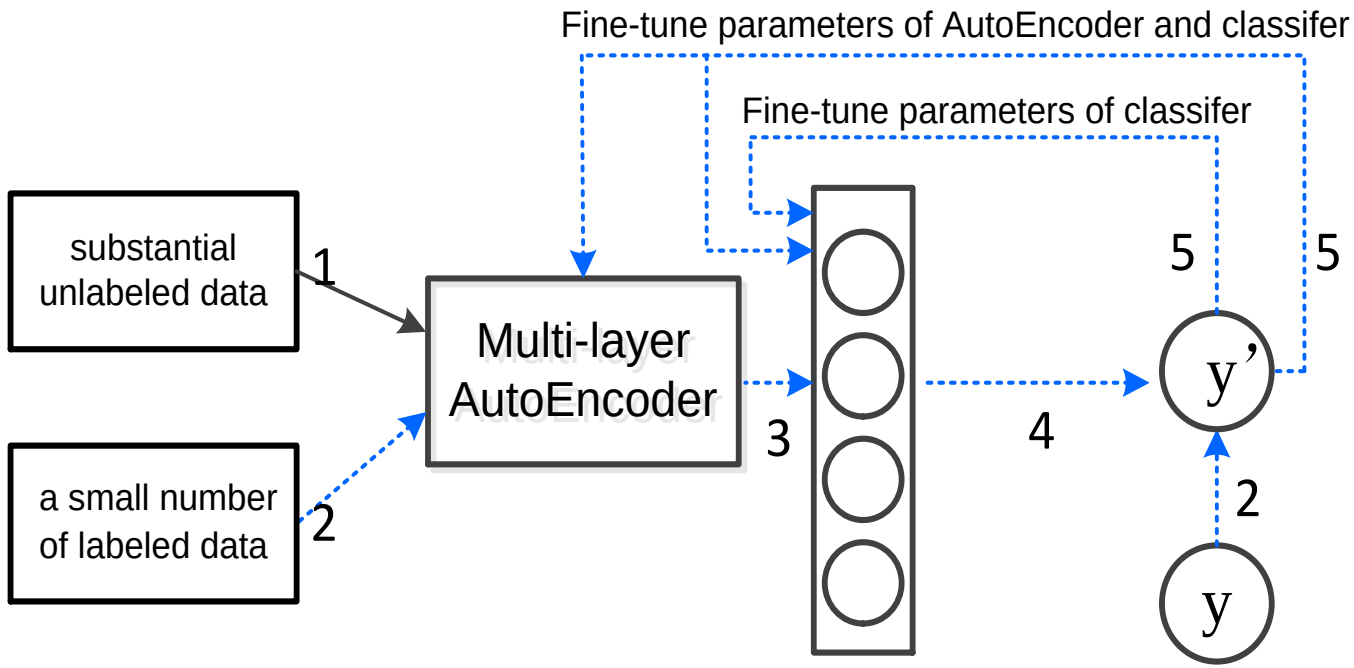
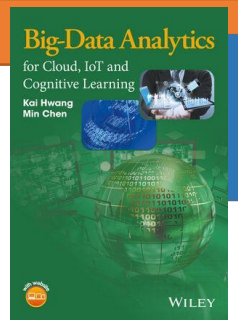


Figure 6.18: Sketch Map of Training Stacked AutoEncoder

There are two methods for supervised fine-tuning.

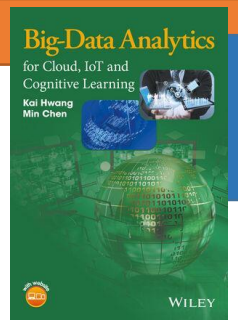
- Adjust classifier parameters only.
- Adjust the parameters of classifier and all AutoEncoder layers.

The marks of No. 1 to No. 5 in Fig. 6.18 represent the sequence of operations in the training. There are two No. 5 operation, which represent two kinds of fine-tuning operations.



Stacked AutoEncoder

- **Five steps for using the algorithm of parameter fine-tuning in SAE**
  - **Step 1:** Substantial unlabeled data are used to train multi-layer AutoEncoder. And multi-layer data feature extraction is constructed.
  - **Step 2:** Obtain code at the highest layer AutoEncoder.
  - **Step 3:** Calculate classification result Y' (i.e., output of the classifier). Calculate error between Y and Y' based on cost function. Note that a small number of labeled data will be utilized.
  - **Step 4:** Judge whether or not end condition is met based on cost function? If yes, the structure of SAE is well trained, and the supervised fine-tuning of parameters is terminated. Otherwise, continue Step 5.
  - **Step 5:** Adjust the parameters of classifier (adjustment for the parameters of multi-layer Autocoder is optional) via stochastic gradient descent. Then return to Step 3.



Restricted Boltzmann Machine (RBM)

Restricted Boltzmann machine **(RBM)** is a neural network model that can realize unsupervised learning.

- It includes two layers, visible layer V and hidden layer H, which are connected by the way of undirected graph. There is no connection among neurons in the same layer.
- The input of RBM is an m-dimensional vector data V, where V= (v_1, v_2, …, v_m) and $v_i \in \{0.1\}$. vi stands for binary state of neuron i in visible layer.
- The output of RBM is an n-dimensional vector H, where H= (h_1, h_2, …, h_n) and $h_j \in \{0.1\}$. h_j stands for binary state of neuron j in hidden layer (Fig. 6.19).

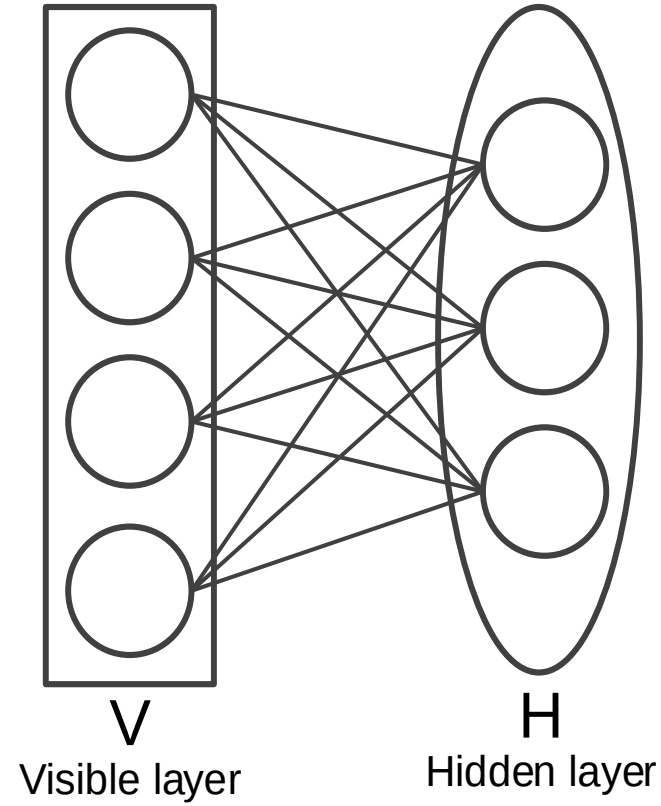
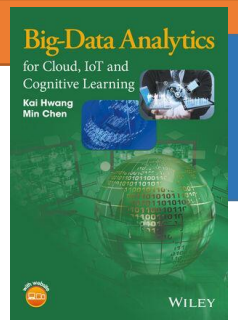


Figure 6.19: The structure of a single stage of restricted Boltzmann machine (RBM)



Restricted Boltzmann Machine (RBM)

Simple Example

The goal is to obtain the answer to the following question: which shapes compose the input graph?

There are only two kinds of components in the graph, i.e., square and triangle.

Let's use code 1 and code 0 to represent square shape and triangle shape, respectively. Let's assume the coding sequences are upper-left corner, upper-right corner, lower-left corner and lower-right corner. Then, the input graph corresponds to a four digit code, i.e., 1011.

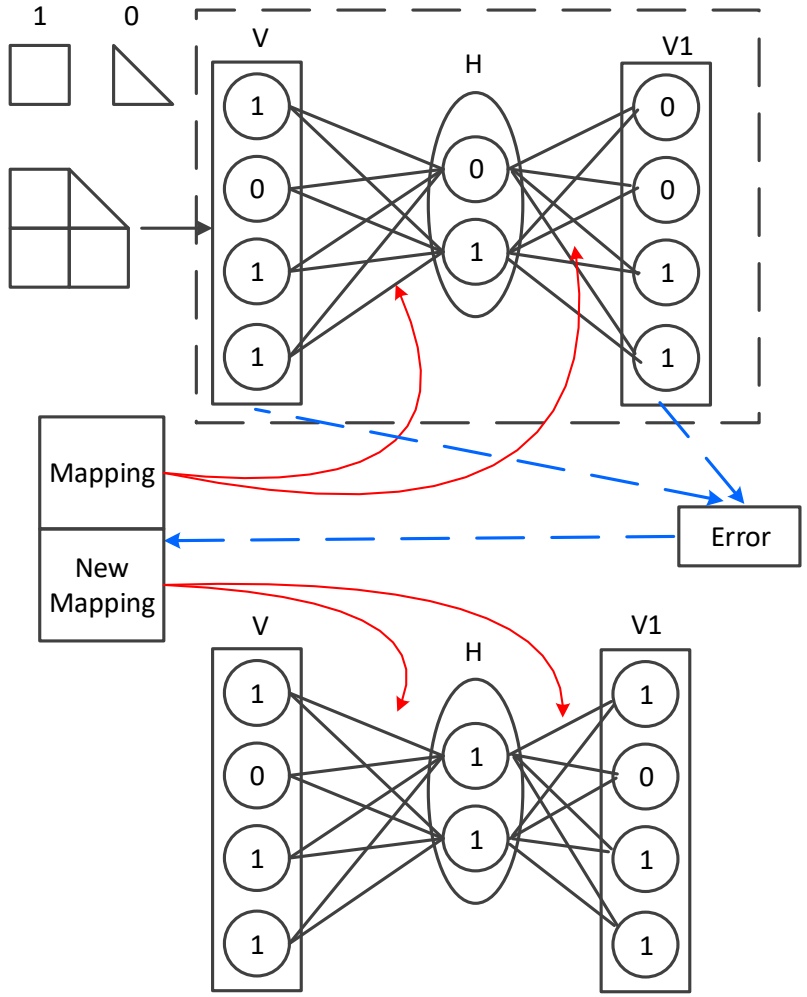
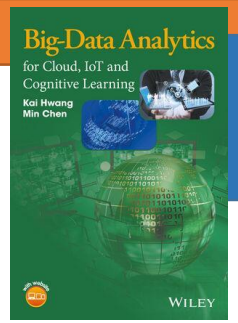


Figure 6.20: Schematic diagram for learning image composition by RBM



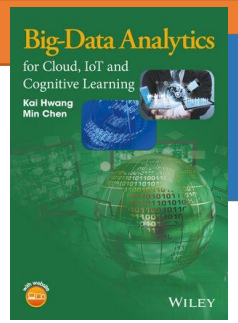
Restricted Boltzmann Machine (RBM)

In Fig. 6.20, we obtain layer H by the mapping from layer V to layer H. Then we use symmetric mapping to reconstruct layer V based on layer H. As shown in Fig. 6.20, we perform the following operations:

(1) A code of 1011 is obtained as V. Through mapping, the expression of 01 in layer H is calculated, which means the input graph consists of triangles. Then, symmetrical mapping is adopted to obtain 0011 as the value of V1.

(2) Calculate error between the distribution of V and V1, revise the mapping parameters according to the error.

(3) Repeat step 1) and step 2) with new mapping. Conduct training and establish RBM after completion of training. It includes layer V, layer H and the mapping between the two layers.

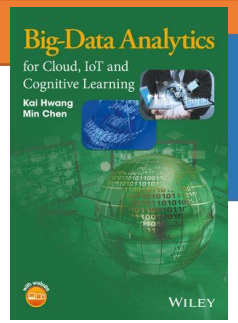


Restricted Boltzmann Machine (RBM)

The RBM learning algorithm utilizes the Boltzmann distribution and solve the parameter θ through maximum likelihood estimation. In other words, P (v | θ) is subject to the Boltzmann distribution:

$$P(\mathrm{v} \mid \theta) = \frac{\sum_{\mathrm{h}} e^{-E(\mathrm{v},\mathrm{h}|\theta)}}{Z(\theta)}$$

Where $E(\mathrm{v},\mathrm{h} \mid \theta) = -\sum_{i=1}^{\mathrm{m}} bv_i v_i - \sum_{j=1}^{\mathrm{n}} bh_j h_j - \sum_{i=1}^{\mathrm{m}} \sum_{j=1}^{\mathrm{n}} v_i w_{ij} h_j$, $Z(\theta) = \sum_{\mathrm{v},\mathrm{h}} e^{-E(\mathrm{v},\mathrm{h}|\theta)}$



Restricted Boltzmann Machine (RBM)

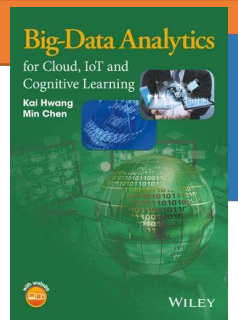
Maximum likelihood function is described in Equation 6.17, which can be solved by gradient descent method.

$$L(\theta) = \sum_{t=1}^{T} \log P(\mathrm{v}^{(t)} \mid \theta)$$

Adopt Equation (8.12) to calculate activation probability of hidden neuron *j* in hidden layer, i.e. calculate the probability when the state of hidden neuron *j* is equal to 1. Substitute the activation probability to obtain h_j of neuron *j.* Thus we can obtain the state *h* of hidden layer.

$$P(h_{1j} = 1 \mid \mathrm{v}_1, \theta) = \sigma(bh_j + \sum_{i=1}^{\mathrm{m}} v_{1i} w_{ij})$$

Where function σ is sigmod function.



Restricted Boltzmann Machine (RBM)

Algorithm 6.2: Rapid learning algorithm based on Contrastive divergence

Input: *X*: a training sample -- original inputted datum x

m: number of neurons in hidden layer

λ: learning rate

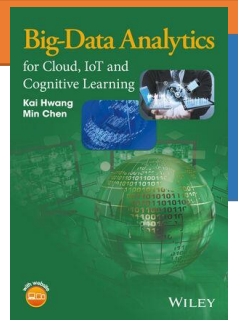
P: iterative numbers of training

Output:

Establish RBM; network parameter

Procedure:

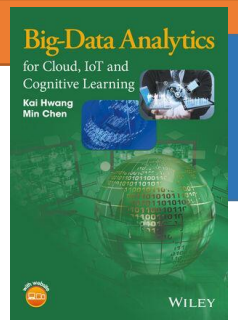
1. Initialization: initial state V_1 for neurons in visible layer; V_1=x; for initialization,
2. for t=1,2,3,…P
3. Calculate activation probability for all neurons in hidden layer, $\mathrm{P}(h_1 = 1|v_1, \theta)$
4. Calculate activation probability for all neurons in hidden layer, V_2
5. Calculate activation probability for all neurons in hidden layer, $\mathrm{P}(h_2 = 1|v_2, \theta)$.
6. Renew weight:
7. Renew deviation in visible layer:
8. Renew deviation in hidden layer:
9. end



Restricted Boltzmann Machine (RBM)

Fives steps to calculate the $\mathbf{P}(h_1 = 1|v_1, \theta)$

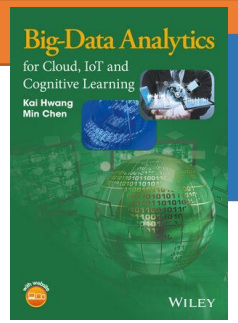
- **Step 1**: Input state of visible layer v_1, network parameter θ, number of neurons in visible layer m, number of neurons in hidden layer n.
- **Step 2**: Input j=1
- **Step 3**: Calculate activation probability for neuron in hidden layer as Equation (8.18)
- **Step 4**: As per uniform distribution, produce a random floating point number that is between 0 and 1. If it is less than $\mathrm{P}(h_{1j} = 1|v_1, \theta)$, the value of h_{1j} is 1; otherwise value is 0.
- **Step 5**: If j<m, then return to Step 3.



Restricted Boltzmann Machine (RBM)

After obtaining all states in hidden layer, the activation probability of neuron *i* in visible layer can be calculated by symmetrically utilizing Equation (6.13). Then, the activation state for neuron *i* can be obtained by utilizing the result of activation probability $v_i \in \{0,1\}$

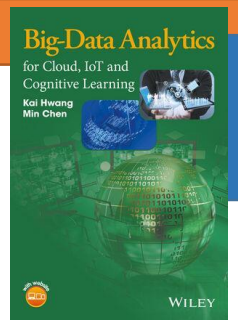
$$P(v_{2i} = 1 | \mathrm{h}_1, \theta) = \sigma(bv_i + \sum_{j=1}^{\mathrm{n}} w_{ij} h_{1j})$$



Restricted Boltzmann Machine (RBM)

How to calculate activation probability of neuron in visible layer

- **Step 1:** Input state of hidden layer h_1, network parameter θ, number of neurons in visible layer m, number of neurons in hidden layer n.
- **Step 2:** Input *i=1.*
- **Step 3:** Calculate activation probability for neuron in visible layer as Equation (8.19) .
- **Step 4:** As per uniform distribution, produce a random floating point number that is between 0 and 1. If it is less than $P(v_{2i} = 1|h_1, \theta)$, the value of v_{2j} is 1; otherwise value is 0.
- **Step 5:** i++.
- **Step 6:** If *i<n*, then return to Step 3.



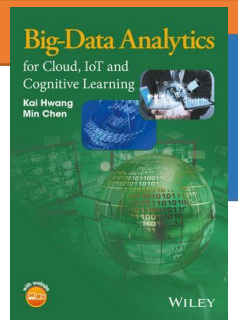
Restricted Boltzmann Machine (RBM)

Example 6.4: Recommend Movies with Restricted Boltzman Machine

Assuming there are four movies. The movies' ratings are integers and vary from 1 to 5, 0 denotes no rating. Table 6.3 shows the rating of user 1 and user 15.

Table 6.3 Movie Rating by Viewers

id	Movie 1	Movie 2	Movie 3	Movie 4
User1	3	4	4	1
User15	3	5	0	0



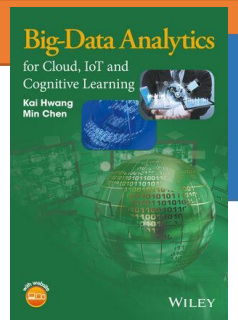
Restricted Boltzmann Machine (RBM)

Step 1: Data set and RBM structure

The ratings of all movies by one user can be represented by a 5*4 matrix *v*, where $v(i, j) = 1$ means the user rate i for movie j. Thus, the rating matrices of user 1 and user 15 are as follows.

$$v_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad v_{15} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Let the quantity of neurons in visible layers and hidden layers be 20 and 5 respectively in RBM model. We can transform the rating matrix into a vector with 20 elements. Thus v_1 and v_{15} can be transformed into {0,0,1,0,0,0,0,0,1,0, 0,0,0,1,0, 1,0, 0,0,0} and {0,0,1,0,0,0,0,0,0,1, 0,0,0,0,0, 0,0, 0,0,0}, respectively.



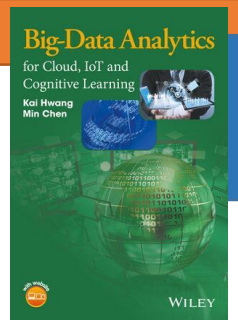
Restricted Boltzmann Machine (RBM)

Step 2: Training

The training stage: After inputting users rating data, we can utilize Algorithm 6.2 to train RBM. The corresponding deviations of visible layers can be obtained as following. After training, the corresponding deviations of visible layers can be obtained as follows.

$$a = \begin{pmatrix} -0.6 & -0.6 & -0.3 & 0 \\ -0.3 & -0.3 & 0.6 & 0.3 \\ 0.6 & 0.3 & 0 & -0.6 \\ -0.3 & 0.3 & 0.6 & 0.0 \\ -0.3 & 0.3 & -0.9 & 0 \end{pmatrix}$$

The deviations of hidden layers are $b = (-1.2 \quad -0.6 \quad -0.6 \quad -0.3 \quad -0.3)$.

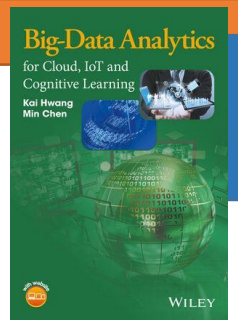


Restricted Boltzmann Machine (RBM)

The weight matrix is a 20*5 matrix, where it can be represented by five matrices according to neurons of hidden layers. Each matrix denotes the connection weights among twenty neurons of a visible layer and one neuron of hidden layer.

$$w(:,:,1)=\begin{pmatrix} -0.8388 & -0.1559 & -1.1827 & -0.2224 \\ -0.7873 & 0.0986 & -0.0791 & -0.8191 \\ -0.6809 & -1.0458 & -0.2480 & -0.3867 \\ -1.2027 & 0.0872 & -0.1243 & -0.5868 \\ -1.1445 & -0.6589 & -0.8836 & -0.1172 \end{pmatrix} \quad w(:,:,2)=\begin{pmatrix} -0.1580 & -0.6287 & 0.3352 & -0.2182 \\ 0.1031 & 0.0961 & 0.3511 & -0.2471 \\ 0.1854 & 0.6668 & 0.4663 & -0.1070 \\ -0.1052 & -0.2801 & -0.2151 & -0.7534 \\ -0.2325 & -0.0457 & -0.0993 & 0.4049 \end{pmatrix} \quad w(:,:,3)=\begin{pmatrix} 0.0467 & -0.2369 & -0.0054 & 0.5772 \\ -0.0050 & -1.1134 & 0.1793 & 0.0143 \\ 0.5374 & -0.2505 & -0.1696 & 0.2379 \\ 0.5298 & -0.1375 & 0.4546 & -0.8972 \\ 0.8843 & -0.2688 & -0.6814 & -0.3714 \end{pmatrix}$$

$$w(:,:,4)=\begin{pmatrix} -0.8423 & 0.0999 & -0.7604 & 0.0074 \\ -0.3207 & -0.6556 & 0.4505 & 0.2922 \\ 1.1088 & -0.1351 & -0.1854 & -0.2619 \\ 0.4764 & 0.2176 & 0.2992 & 0.0226 \\ 0.4378 & 0.1666 & -0.6949 & -0.2281 \end{pmatrix} \quad w(:,:,5)=\begin{pmatrix} -0.7458 & -0.5607 & -0.0101 & 0.1658 \\ -0.4698 & 0.0969 & 0.4129 & 0.7707 \\ 0.7059 & 0.7355 & 0.0868 & -0.3635 \\ 0.5583 & 0.2992 & 0.4809 & -0.1271 \\ 0.7589 & 1.2223 & -0.2460 & 0.3042 \end{pmatrix}$$



Restricted Boltzmann Machine (RBM)

Step 3: The movie recommendation

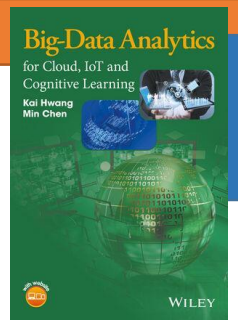
1) Input the rating data of user 15, and utilize Equation (6.16) to calculate the activate probability of all hidden units, i.e. *ph*= (0.0214, 0.0151, 0.0453, 0.1404, 0.6583). The weights and deviations of hidden layers can be obtained through training:

$$p(h_j = 1 | V) = \frac{1}{1 + \exp(-b_j - \sum_{i=1}^{M} \sum_{k=1}^{K} v_i^k W_{ij}^k)}$$

The uniform distribution generates a random floating point number between 0 and 1. If it is less than ph_j, then $h_j = 1$, otherwise $h_j = 0$. Then we can get h = (0, 0, 0, 0, 1).

2) According to the activate probability *ph* of hidden layers calculated in previous steps, use the following equation to calculate the activate probability *pv* of visible layers. The weight matrix and deviation of visible layer can be obtained by training. We can calculate the corresponding activate probability of every rating k for each movie i, where k=1,…,5. $pv(i.j)$means the probability of movie j rating i.

$$p(v_i^k = 1 | h) = \frac{1}{1 + \exp(-a_i^k - \sum_{j=1}^{F} W_{ij}^k h_j)}$$



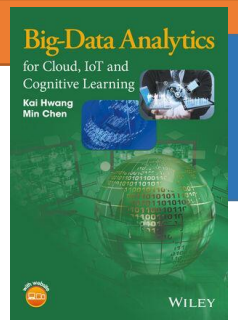
Restricted Boltzmann Machine (RBM)

Step 3: The movie recommendation

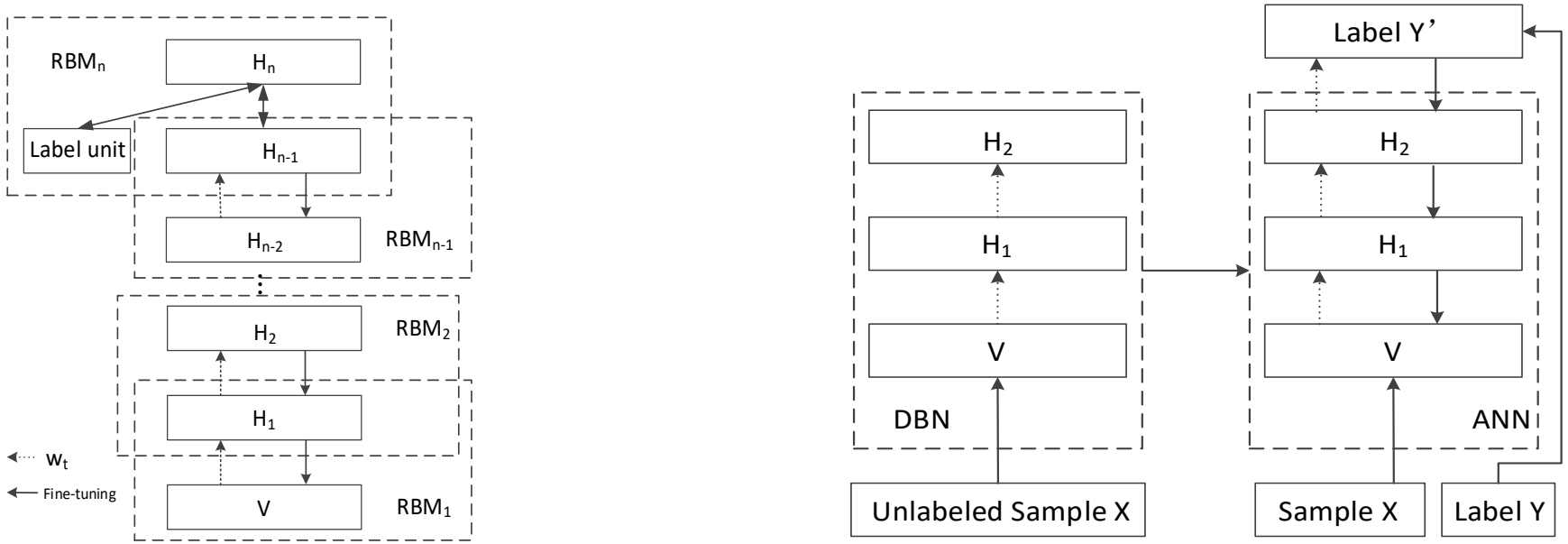
3) We choose the highest probability of each column as the rating by user 15 to movie i. For example, for mov 1, the probability of the rating 1,2,3,4,5 is respectively 0.2066, 0.3165, 0.7868, 0.5642, 0.6128. The highest probability is 0.7868, corresponding rating is 3, so user15 rates movie 1 as 3. After deducing the rest from this, the user's rating for the four movie is (3,5,4,2).

$$pv = \begin{pmatrix} 0.2066 & 0.2385 & 0.4231 & 0.5414 \\ 0.3165 & 0.4494 & 0.7336 & 0.7447 \\ 0.7868 & 0.7380 & 0.5217 & 0.2762 \\ 0.5642 & 0.6455 & 0.7467 & 0.4683 \\ 0.6128 & 0.8209 & 0.2412 & 0.5755 \end{pmatrix}$$

4) Because user15 haven’t rated movie 3 and movie 4, it is likely user15 haven’t seen these two movies. According to ratings, we recommend movie 3 first, and movie 4 second.



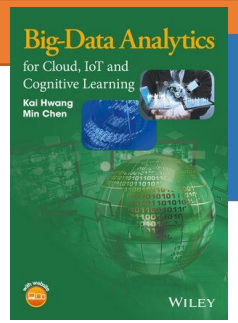
Deep Belief Networks(DBN)



(a) DBN for classification *(b) Initialization for ANN*

Figure 6.21: The structure of a deep belief network (DBN)

Fig. 6.21(a) shows that a DBN includes one visible layer V and n hidden layers. DBN is composed of n stacked RBM which means the hidden layer of the previous RBM is as the visible layer of the next RBM. The original input is visible layer V. This layer V together with hidden layer H_1 consist of another RBM. H_1 is the visible layer V_2 of the second RBM. This hidden layer H_1 together with hidden layer H_2, consists of one RBM and so on. All adjacent two layers form an RBM, where the visible layer and hidden layer of the top layer are part of an undirected connection which is called associative memory.

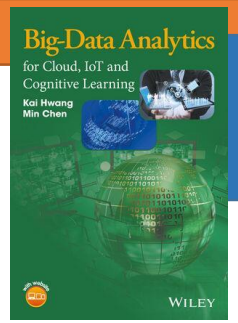


Deep Belief Networks(DBN)

- Unsupervised training adopt original input V as the visible layer of the first RBM. Using the method introduced in Section 6.3.1 to train RBM_1 and fix the connection parameters until the end of the training.
- Then we get the hidden layer H_1 and let this H_1 as the visible layer V_2 of the second RBM. A similar method is applied to train RBM_2 and obtain the hidden layer H_2.
- Repeat this process until all the RBM training is done.

The unsupervised training of multi-layer RBM is given in Algorithm 6.3.

Fine-tuning uses sample data as the input data of the visible layer of DBN and the label of sample as a part of the visible layer of the top RBM layer. The category labels data y of the sample are expressed with neurons. In other words, if there are m kinds of category labels data of the sample, there will be m neuron to express them. Setting the neuron which corresponds to the category is equal to 1. BP algorithm can be used to adjust network parameters when utilizing DBN for fine-tuning.



Deep Belief Networks(DBN)

Algorithm 6.3 : Unsupervised training of multi-layer RBM

Input：

V：Training data

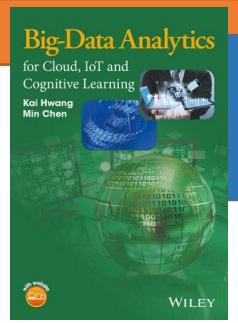
r_n：The number of RBM layers

Output：

The activation status of each neuron in visible layer V

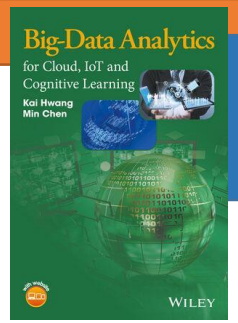
procedure：

1. V_1=V
2. for t=1 to rn
3. Initial parameter of the t_{th} RBM network
4. Use contrast divergence algorithm to train t_{th} RBM and get h_t
5. t++；
6. V_t=H_{t-1}
7. endfor



Convolutional Neural Networks (CNN)

The convolutional neural network (CNN) is a kind of feed-forward neural network, which uses convolution and reduces quantity of weights in the network and reduces complexity of calculation compared with traditional feed-forward neural networks. This kind of network structure is similar to biological neural networks. The CNN belongs to the supervised learning method, and it is widely applied in voice recognition and image fields.



Convolutional Neural Networks (CNN)

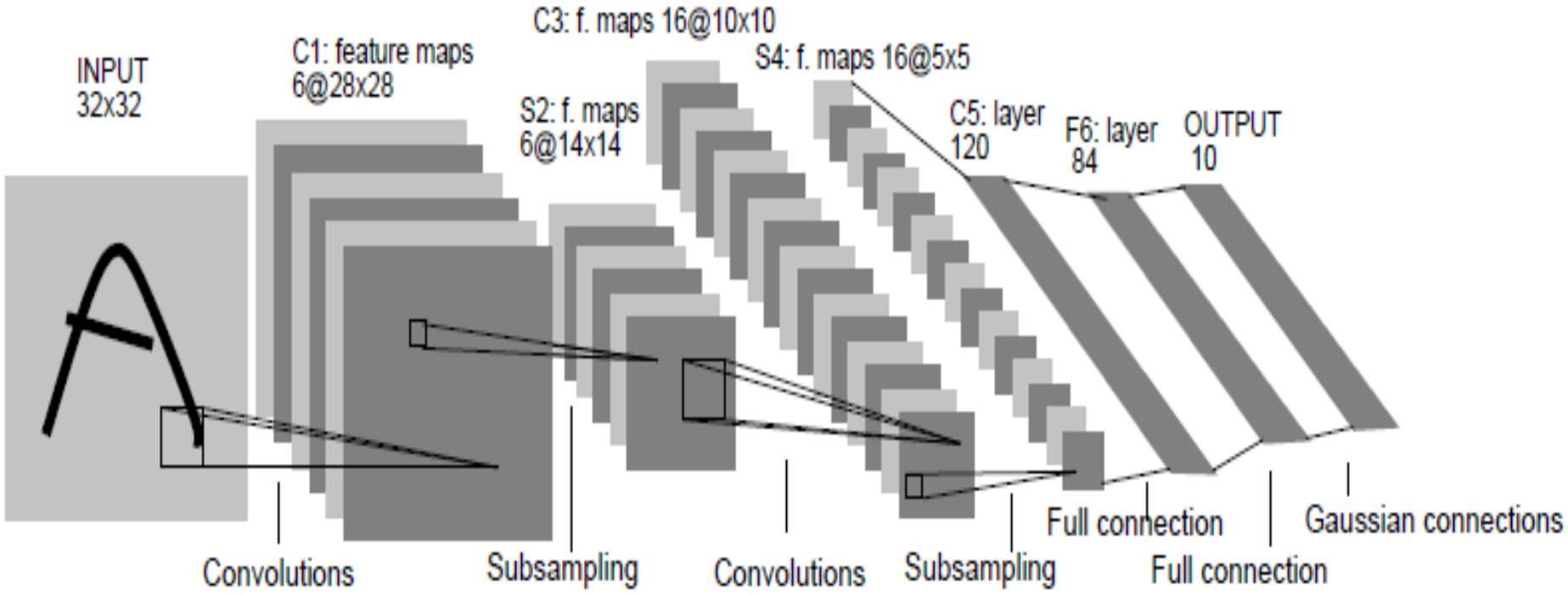
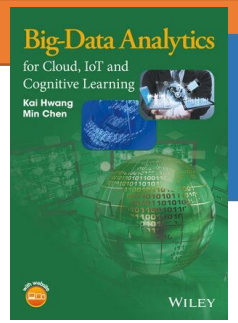


Figure 6.22: Convolutional Neural Network utilized in LeNet-5

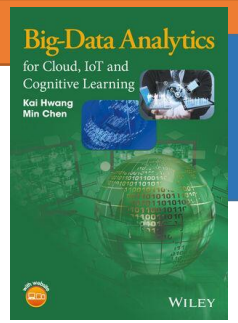
- The CNN is composed of many layer. Generally, it includes such basic components as: input layer, convolution layer, pooling layer, fully connected layer, output layer.



Convolution in CNN

- When considering the need to process an image of 500 × 500 pixels, the quantity of neurons in the input layer should be set to 500 × 500. We assume a set 10^8 neurons in the hidden layer. Each connection between one neuron in the input layer and one neuron in the hidden layer should be set as a weight parameter. Here let us compare the number of weight parameters needed in the two cases when using fully connected neural networks and using locally connected neural networks.

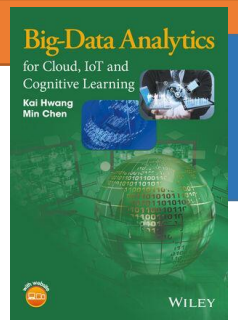
- When establishing a fully connected neural network to deal with the image, the quantity of weight parameters between input layer and hidden layer shall be $500*500*10^8 = 25*10^{12}$. Understanding a large image with the method of fully connection faces the problem that the parameters are too many and the computational burden is unbearable. People can only see part of the image information each time. Using the method of people understanding an image as a reference, we can design a filter to extract the local features of an image and apply it to understand the whole image.



Convolution in CNN

- In other words, we use a filter to realize the local connection between input layer and hidden layer. Assume，a 10x10 filter is designed to imitate human eyes to feel the local image region. Then a hidden layer neuron is connected to a 10x10 area of the input layer through the filter. Hidden layer has 10^8 hidden neurons so the quantity of filter between hidden layer and input layer is 106. Each connection is corresponding to a 10x10 area of the input layer through a filter. Thus the quantity of connection weight parameters between input layer and hidden layer becomes $10x10x10^8 = 10^{10}$.

- While reducing the quantity of weights from $25x10^{12}$ to 10^{10} by changing full connection to local connection, the problem that the weight parameters is too many and the computational burden is too large is still not solved. Because there are intrinsic features in natural images. In other words, the statistic features for one part of an image are the same with those of other parts. This means the features learned from one part of an image can also be utilized for other parts. Therefore, for all the locations of the same image, we can use the same learning features. A filter is a matrix of weights and represents local features of an image.

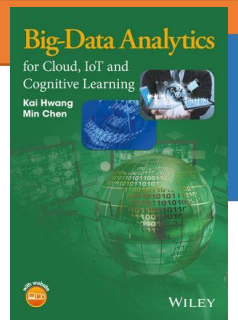


Convolution in CNN

- Then the same filter can be utilized for all the locations of an image naturally.

- By sharing weight, the quantity of weight parameter is reduced from $25x10^{10}$ to 100, which reduces the quantity of weight parameters and computational burden greatly.

- The concept of local connection and weight sharing that we use achieves convolution operation. The filter of 10*10 is exactly a convolution kernel. A convolution kernel just represents one kind of local feature of an image. When we need to represent more local features of an image, we can use multiple convolution kernels. The first convolution layer of Fig. 6.22 has 6 filters, then we can gain 6 feature maps of hidden layers by convolution operation.

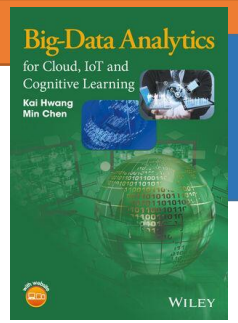


Convolution in CNN

Example 6.5: How Convolution Works for Building Convolutional Neural Network

- We can understand how to realize the convolution through the convolution operation for an image of 8x8 as shown in Fig. 6.23. The dimension of convolution kernel is 4x4; and the feature matrix is

$$w = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$



Convolution in CNN

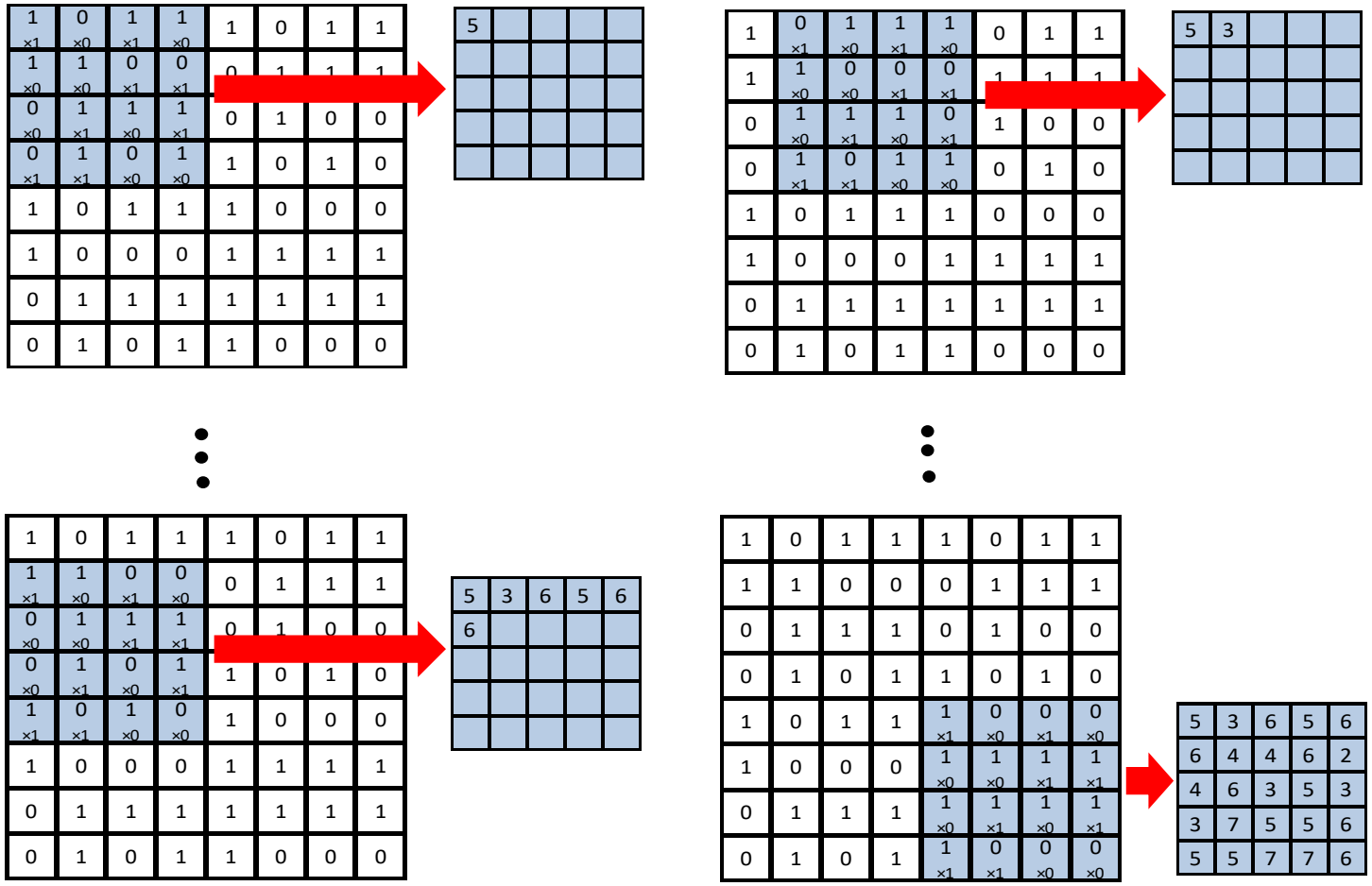
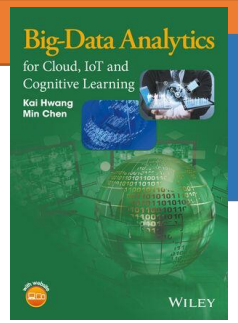


Figure 6.23: Schematic diagram for a convolutional ANN (CNN)
(Source:http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial)



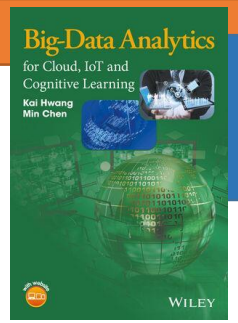
Convolution in CNN

- First we extract an image x_1 of 4*4 from the image of 8*8 for convolution operation with feature matrix. Here, we obtain the value y_1 for the first neuron in hidden layer by utilizing equation

$$y_i = w \cdot x_i$$

- The step size of convolution is set 1. We continue to extract image x_2 of 4*4, and obtain the value y_2 for the second neuron through convolution operation.

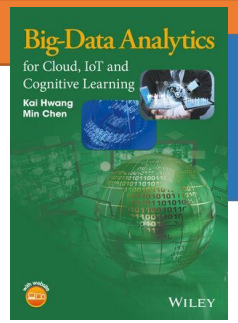
- We repeat the aforementioned steps until the traverse of the whole image is completed. After the calculation for all values of neuron in hidden layer is completed, a feature map corresponding to a convolution kernel is obtained.



Convolution in CNN

Usually, we calculate values of the output feature map in hidden layer utilizing activation function. The frequently used activation functions are:

- sigmod function [$\sigma(x) = \frac{1}{1+e^{-x}}$]
- hyperbolic tangent function [$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$]
- RELU function [$\mathrm{RELU}(x) = \max(0, x)$]



Convolution in CNN

Calculate the output feature map in convolution layer:

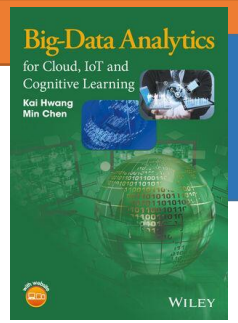
$$y_j = f\left(\sum_{i=1}^{n}(w_{ij} * x_i + b_i)\right)$$

The quantity of neurons in hidden layer:

$$n_y = \left(\left|\frac{n_{l-1} - n_k}{s}\right| + 1\right) \times \left(\left|\frac{m_{l-1} - m_k}{s}\right| + 1\right) \times m$$

The quantity of neurons in hidden layer:

$$n_l = \left(\left|\frac{8-4}{1}\right| + 1\right) \times \left(\left|\frac{8-4}{1}\right| + 1\right) \times 1 = 5 \times 5$$



Pooling in CNN

Common images have an attribute of static nature. The features that are useful in one image area are very likely to be applicable in another area. Therefore, in order to describe large images, we can conduct aggregation statistics for features at different locations. For instance, people can calculate the mean value (or maximum value) in an area of image. The statistical features obtained by such aggregation cannot only reduce dimensionality but also improve the results (by preventing over-fitting). The operation of such aggregation is called pooling. As per different computational methods, it is divided into mean pooling and maximum pooling. Fig. 6.24 shows pooling operation of 3×3 for an image of 6×6; the image is divided into 4 areas that do not overlap with each other. Fig. 6.24 shows the result after maximum pooling in one area. The feature graph after pooling is 2×2.

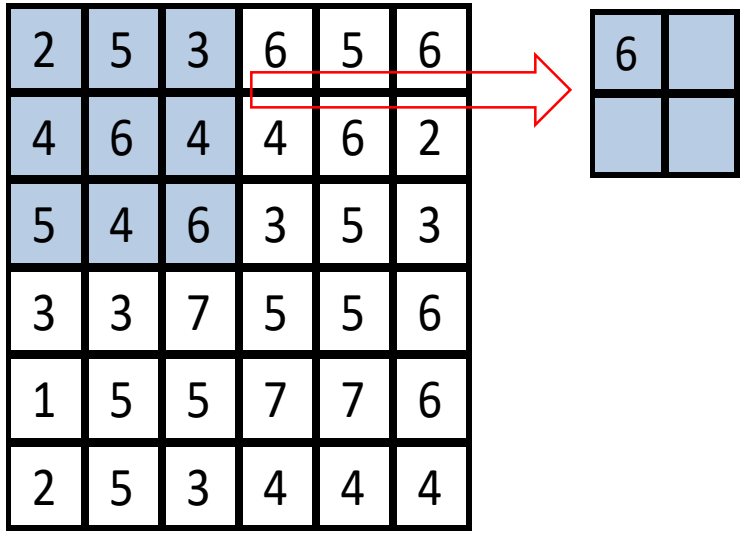
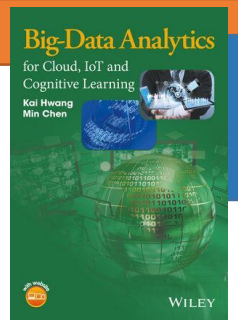


Figure 6.24: The concept of pooling from the 3 x 3 grid to a 2 x 2 grid (source: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial)



Pooling in CNN

Example 6.6: Convolution and pooling for CNN

In recent years, CNN has been widely used in digital image processing with the rapid development of CNN. For example, using this DeepID convolutional neural network, the correct recognition rate of the human face can reach a maximum of 99.15%. This technique can play an important role in the search for missing people and the prevention of terrorist crime. Fig. 6.25 shows CNN used.

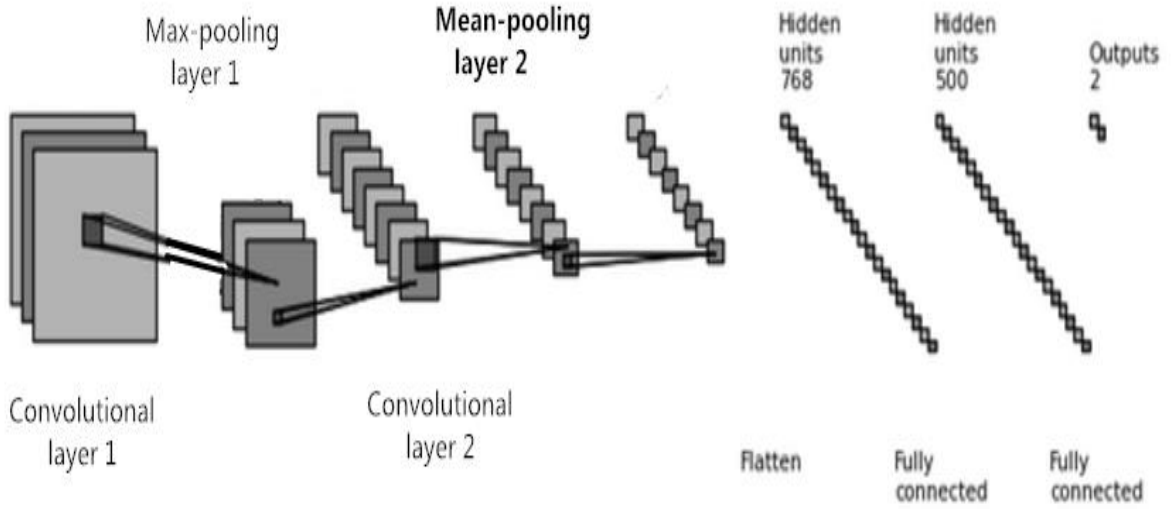
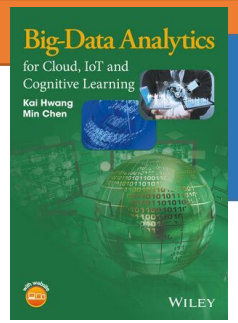


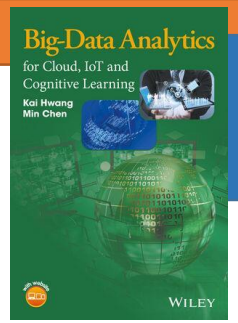
Figure 6.25 Schematic diagrams of Convolutional Neural Network



Pooling in CNN

If the given input image is as shown in Fig. 6.26(a), the image size is 8 x 6. We assume the size of convolution kernels is 3 x 3 and the size of one feature graph in convolutional layer 1 is ((6-3)+1) x ((8-3)+1)=4 x 6. Assuming we use 3 filters, the corresponding weight matrices are:

$$w_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, w_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, w_3 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$



Pooling in CNN

4	3	1	6	8	0	0	0
0	0	0	5	3	2	1	0
0	0	0	4	5	7	9	1
0	0	0	0	0	0	0	0
1	2	6	2	1	6	3	1
0	0	0	0	0	0	1	1

(a) Input image

5	13	14	17	22	8
0	5	3	7	4	2
7	8	12	19	18	15
0	0	0	0	1	1

(b) After first convolution

-5	3	4	7	12	8
-10	-5	-7	-3	-6	-8
-3	-2	2	9	8	5
-10	-10	-10	-10	-9	-9

(c) After deviation

0	3	4	7	12	8
0	0	0	0	0	0
0	0	2	9	8	5
0	0	0	0	0	0

(d) Feature map at layer 1

0	0	3	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

(e) After second convolution

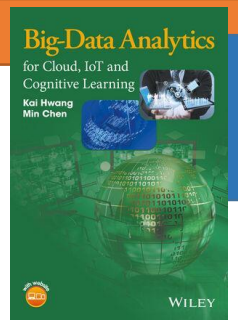
0	0	3	0	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0

(f) After another convolution

1			2			3		
3	7	12	0	3	0	0	3	0
0	9	8	0	0	0	0	1	0

(g) Feature may after maximum polling

Figure 6.26: Successive convolutional steps in a CNN

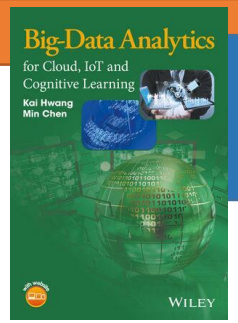


Pooling in CNN

Assume deviation b= – 10, the activation function RELU(x) = max(0, x).

In order to obtain the feature graphs of convolutional layer 1, we need the following operations:

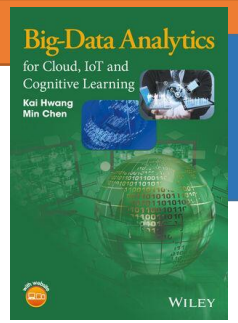
- **Step 1**: Use w_1 to perform convolution of input data. The result is showed in Fig. 6.26(b).
- **Step 2**: Fig. 6.26(c) shows the result with added deviation.
- **Step 3**: After activation, we obtain the feature map 1 of convolutional layer 1, as showed in Fig. 6.26(d).
- **Step 4**: Repeating the above steps over weight matrix w_2 and w_3, we obtain the feature maps 2 and 3 of convolutional layer 1, as showed in Figures 6.26(e) and (f).



Training Convolutional Neural Networks

In fact, the learning of convolutional neural network is to obtain the mapping relation between input and output through large quantity of learning of data with tags. But this mapping relation between input and output is hard to express with precise mathematical expression. When the training of such mapping relation is completed, the corresponding output data can be obtained through the mapping by inputting data into the network.

The training of convolutional neural network is similar to BP network. It is divided into two stages, one is forward propagation to calculate output and the other is backward propagation for parameter adjusting with error. Algorithm for forward propagation is shown in Algorithm 6.4. During the course of forward propagation, we input sample data x into convolutional neural network, conduct operation on input data layer by layer, make the output data of previous layer as the input data of next layer, and finally obtain output result of y'.



Training Convolutional Neural Networks

During backward propagation, we calculate the error between right tag y of input data x and the output result y' , transmit the error and adjust parameters layer by layer from output layer until it reaches the input layer.

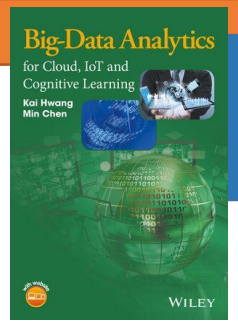
Detailed steps are as follows:

- **Step 1:** y stands for tag of sample data, y' stands for output through calculation, calculate error: $O = y - y'$, Calculate cost function: $E = \frac{1}{2}\sum_{N}\sum_{K}(y - y')^2$, N stands for number of samples, K stands for classes.

- **Step 2**: Calculate derivative of error E to weight w: $\frac{\partial E}{\partial w}$, layer by layer from output to input layer, Renew the weight by using $w = w + \Delta w = w + \lambda \frac{\partial E}{\partial w}$

- **Step 3:** Utilize Algorithm 6.4 from line 2) to line 15) to calculate y' .

- **Step 4:** Judge whether the terminal condition of backward propagation is achieved, if not, conduct Step 1, or end the training.



Training Convolutional Neural Networks

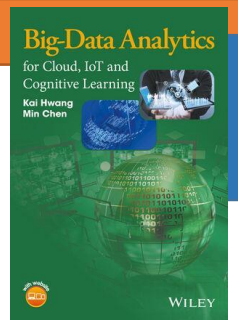
Algorithm 6.4 : Forward propagation in CNN

Input: x: sample data

Output: y': output through calculation

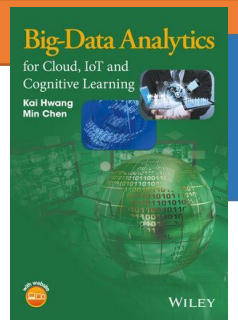
Procedure:

1) Conduct initialization to all parameters in n layers
2) p_1=x
3) for i=1,2,…n
4) if i is convolution layer
5) Conduct convolution operation for p_i, and output q_i
6) i++;
7) p_i=q_{i-1}
8) endif
9) if i is pooling layer
10) Conduct pooling operation for p_i, and output q_i
11) i++;
12) p_i=q_{i-1}
13) endif
14) endfor
15) Output y' through calculation



Other Deep Learning Neural Networks

Deep learning architecture includes input layer, output layer and several hidden layers. Deep learning architecture means deep neutral network which has forward propagation and reverse learning. There are many types of deep learning architecture. Most of these architectures are used to change the common architecture. Here we divide deep learning architecture into 3 types according to the connectionist models of neutrons: *fully connected*, *locally connected*, and many other deep learning networks as shown in Fig. 6.27.



Other Deep Learning Neural Networks

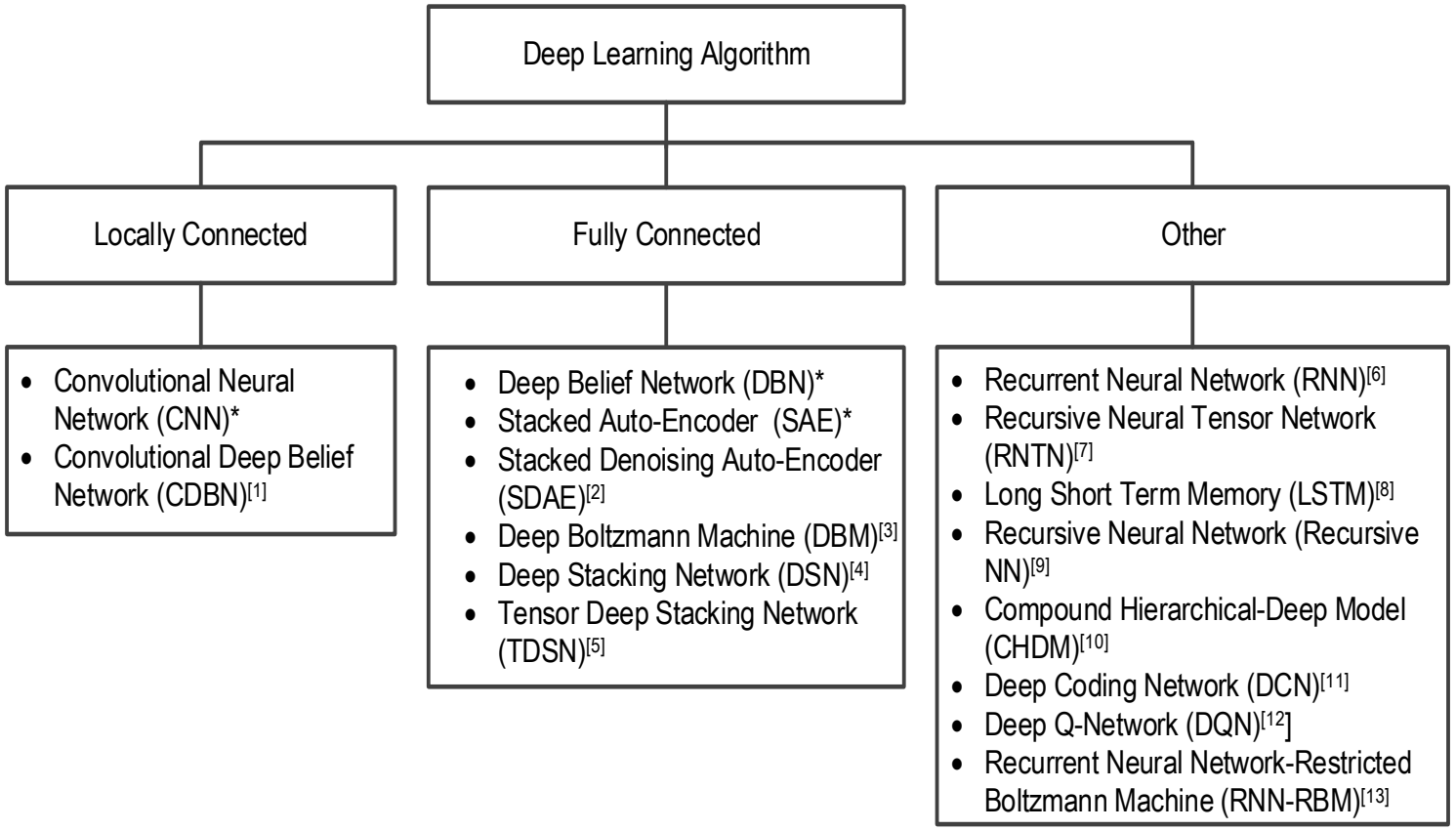
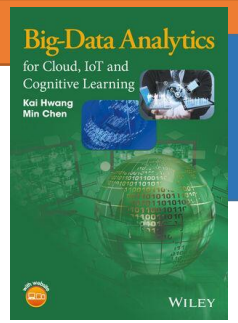


Figure 6.27: The architectural classification of various deep learning models. (Those marked with * are covered in this book. The superscripts correspond to the references of different neural networks)

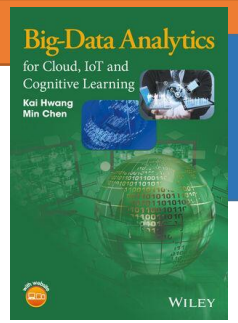


Connectiveness of Deep Learning Networks

- ***Fully connected networks:*** In the traditional neutral network, the connections between layers from input layer, hidden layer to output layer are fully connected. One neutron in the previous layer connects with every neuron in the next layer. Fully connected deep learning architectures include Deep Belief Networks (DBN), Deep Boltzmann Machine (DBM), Stacked Auto-Encoder (SAE), Stacked Denoising Auto-Encoder (SDAE), Deep Stacking Network (DSN) and Tensor Deep Stacking Network (TDSN).

- ***Locally connected Networks:*** Locally connected deep learning architecture means the connection mode between input layer and output layer is locally connected. This kind of deep learning architecture takes *Convolutional Neural Network* (CNN) as the representative class. It uses the concept of partial connection and weight sharing of convolutional operation to describe the overall by local features. Thus, it reduces the number of weight greatly. *Convolutional Deep Belief Networks* (CDBN) are also locally connected.

- ***Other:*** This class includes Recurrent Neural Network (RNN), Recursive Neural Tensor Network (RNTN) and Long Short-Term memory (LSTM), etc. Other related networks include the Recurrent Neural Network-Restricted Boltzmann Machine (RNN-RBM), Deep Q-Network (DQN), Compound Hierarchical-DeepModels (CHDM) and Deep Coding Network (DPCN), etc. Conventional ANNs, locally connected or fully connected, may have limited applicability, because they perform badly or become powerless when dealing with data streams.



Recursive Neural Networks (RNNs)

A RNN corresponds to an ANN with three layers at each time point, so the training of this is similar to traditional ANN. However, a RNN considers current output of a data stream also related to the previous output. That means information processing at current time needs to consider the output from last time. Training a single layer RNN for hundred time steps is equivalent to training a feed forward network with hundreds of layers as shown in Fig. 6.28. When an RNN processes a sequence of data, the previous output will feedback as part of the input data.

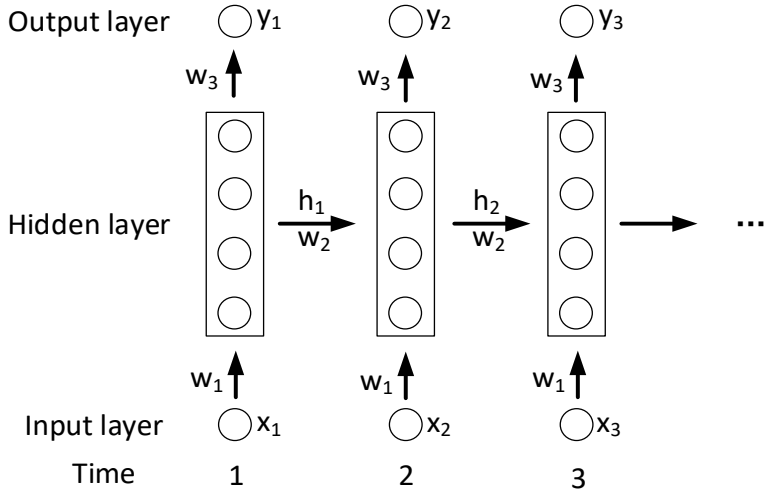
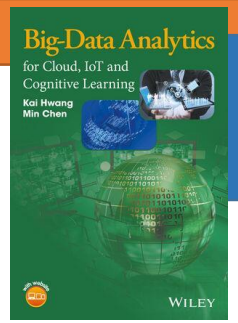
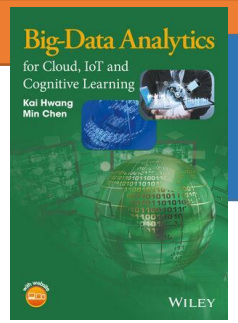


Figure 6.28 The structure of recursive neural networks (RNN)

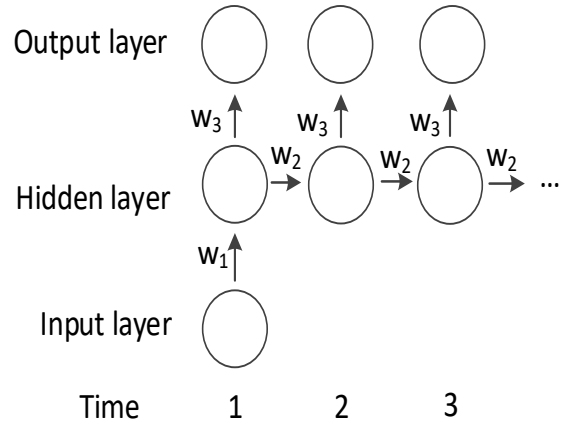


Input and Output Relationships

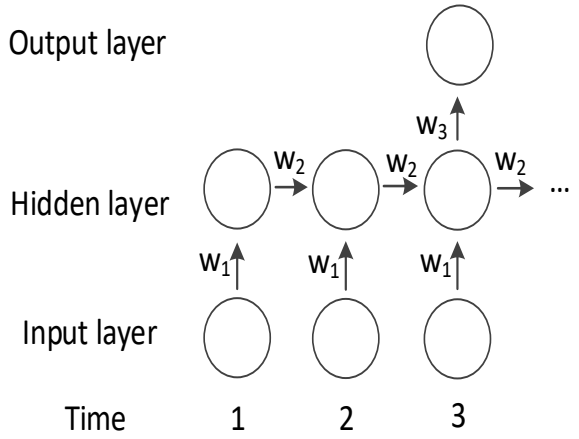
- RNN receives a sequence of input and also produces a sequence of output. According to different applications, there are different forms of input or output pairs, which are shown in Fig. 6.30 in four cases. In Fig. 6.30(a), the I/O structure is specially appealing to image capturing application. Fig. 6.30(b) shows the I/O structure for having multiple inputs with a single output, which matches with document classification.
- In Fig. 6.30(c), both input and output are shown as sequential. This is practiced in RNN for video streaming applications, frame by frame. This architecture is also suitable for statistical prediction of the future situation. In Fig. 6.30(d), we input the known data at time 1 and time 2, and the prediction starts at time 3. After inputting the data at time 3, we get the output result 1. This implies that we have predicted the data at the next time as 1. In the same way, we get the output result 2 after inputting the data 1 at time 4 and predicting the data at time 5 as 2.



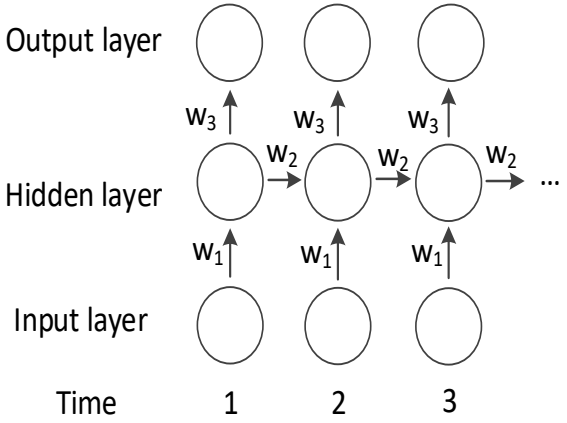
Input and Output Relationships



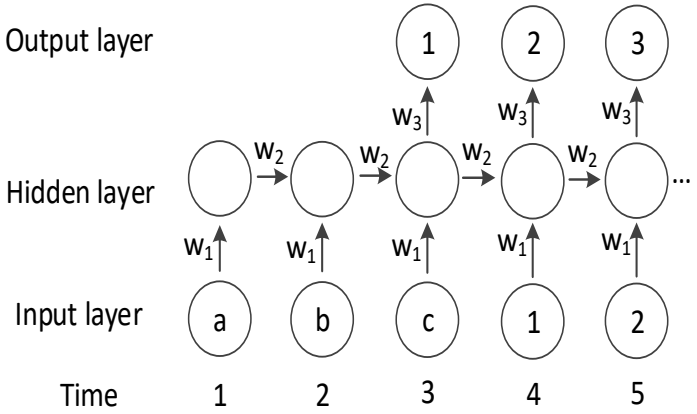
(a) Image capturing



(b) Document classification

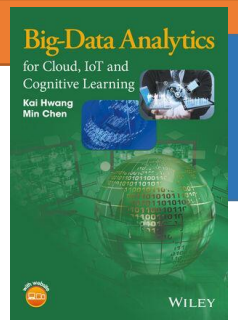


(c) Classify video frame by frame



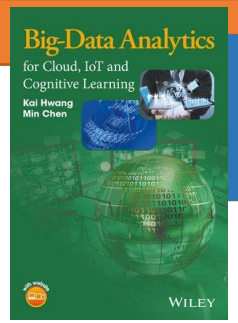
(d) Statistical prediction

Figure 6.30 Different forms of input or output applied in different deep learning applications



Other Deep Learning Neural Networks

- Convolutional Deep Belief Networks (CDBN)
- Deep Q-Networks (DQN)
- Deep Boltzmann Machines (DBM)
- *Stacked Denoising Auto-Encoder* (SDAE)
- Deep Stacking Network (DSN)
- Tensor Deep Stacking Network (TDSN)
- Long Short Term Memory (LSTM)
- Deep Coding Networks (DPCN)
- Compound Hierarchical-Deep Model(CHDM)
- Recurrent Neural Network-Restricted Boltzmann Machine (RNN-RBM)


Big-Data Analytics
for Cloud, IoT and
Cognitive Learning
Kai Hwang
Min Chen
WILEY

Thank You !