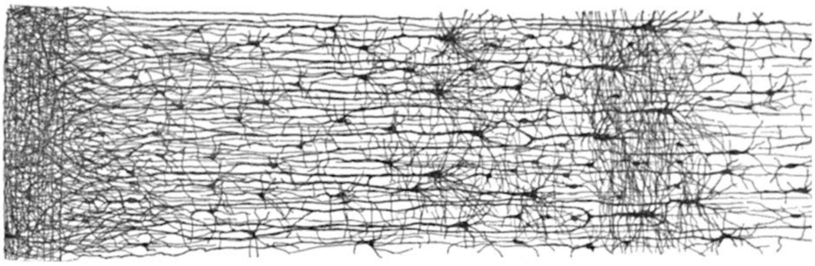

图 8.1　人类大脑皮层的神经分布

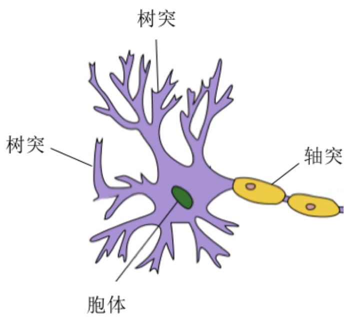

图 8.2　神经元结构

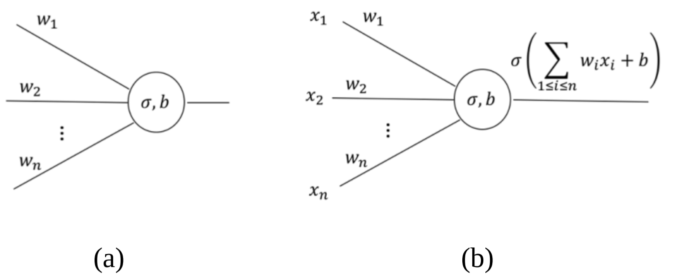

图 8.3　神经网络算法计算单元的基本结构

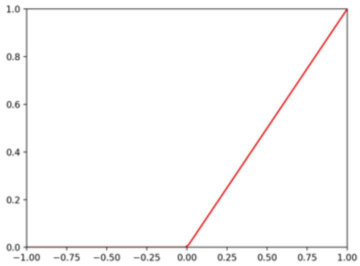

图 8.4　ReLU 激活函数

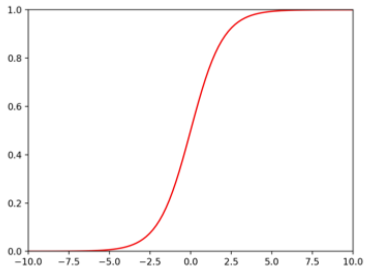

图 8.5　Sigmoid 激活函数

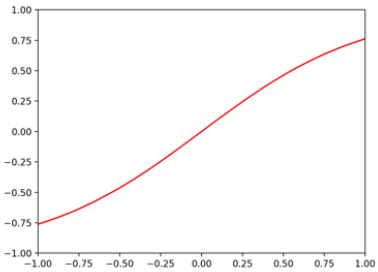

图 8.6　Tanh 激活函数

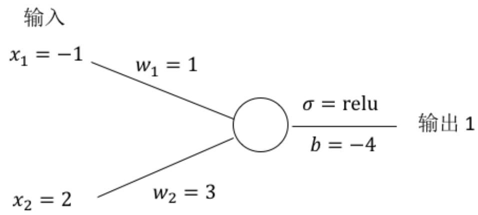

图 8.7　神经元计算示例

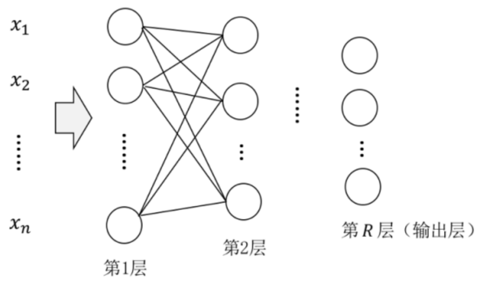

图 8.8　神经网络模型示意

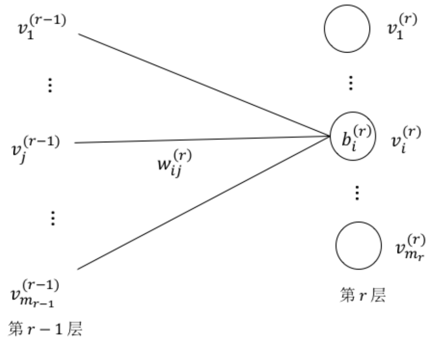

图 8.9　神经网络第 r 层的输入 $\boldsymbol{v}^{(r-1)}$与输出$\boldsymbol{v}^{(r)}$

神经网络模型

$\boldsymbol{v}^{(0)} = \boldsymbol{x}$

For $r = 1,2,\ldots,R$:

$\quad\quad \boldsymbol{s}^{(r)} = \boldsymbol{W}^{(r)}\boldsymbol{v}^{(r-1)} + \boldsymbol{b}^{(r)}$

$\quad\quad \boldsymbol{v}^{(r)} = \sigma^{(r)}\left(\boldsymbol{s}^{(r)}\right)$

Return $\boldsymbol{v}^{(R)}$

图 8.10　神经网络模型描述

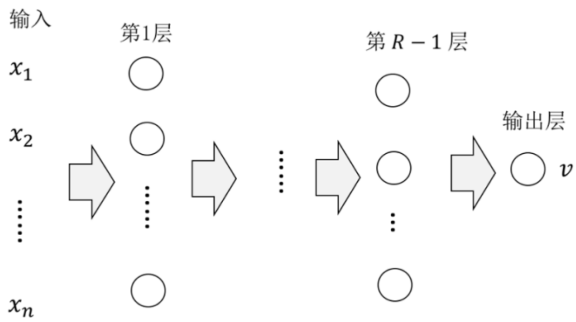

图 8.11 回归问题的神经网络模型

神经网络回归算法

样本空间 $X \subseteq \mathbb{R}^n$

输入： m 个训练数据 $S = \{(\boldsymbol{x}^{(1)}, y^{(1)}),\ (\boldsymbol{x}^{(2)}, y^{(2)}), \dots,\ (\boldsymbol{x}^{(m)}, y^{(m)})\}$

模型假设 H ：一个含有 n 个输入以及 1 个输出的取定的网络结构

任务：

$$\min_{h \in H} \frac{1}{m} \sum_{i=1}^{m} \left(h(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2$$

图 8.12　神经网络回归算法描述

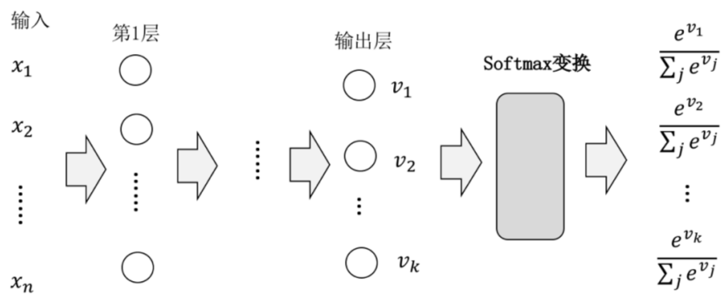

图 8.13 k 元分类问题的神经网络模型

神经网络分类算法

样本空间 $X \subseteq \mathbb{R}^n$，标签空间 $Y \subseteq \{0,1\}^k$

输入： m 个训练数据 $S = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \dots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$

模型假设 H：含有 n 个输入以及 k 个输出的取定的网络结构加上 Softmax 变换

任务：

$$\min_{h \in H} -\frac{1}{m}\sum_{i=1}^{m}\langle \boldsymbol{y}^{(i)}, \log h(\boldsymbol{x}^{(i)})\rangle$$

图 8.14　神经网络分类算法描述

神经网络的随机梯度下降算法

For each r, i, j:

$$w_{ij}^{(r)} = \text{random number}$$

$$b_i^{(r)} = 0$$

For $t = 1, \ldots, N$:

Sample $(\boldsymbol{x}, \boldsymbol{y}) \sim S$

$$\boldsymbol{v}^{(R)} = h_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x})$$

$$\left(\frac{\partial L}{\partial w_{ij}^{(r)}}, \frac{\partial L}{\partial b_i^{(r)}}\right)_{r,i,j} = \text{BackProp}(\boldsymbol{v}^{(R)}, \boldsymbol{y})$$

For each r, i, j:

$$w_{ij}^{(r)} \leftarrow w_{ij}^{(r)} - \eta \frac{\partial L}{\partial w_{ij}^{(r)}}$$

$$b_i^{(r)} \leftarrow b_i^{(r)} - \eta \frac{\partial L}{\partial b_i^{(r)}}$$

Return $h_{\boldsymbol{W},\boldsymbol{b}}$

图 8.15 神经网络的随机梯度下降算法

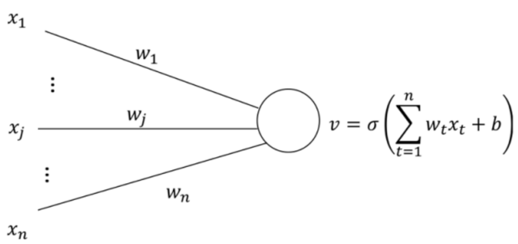

图 8.16　例 8.6 回归问题的神经网络

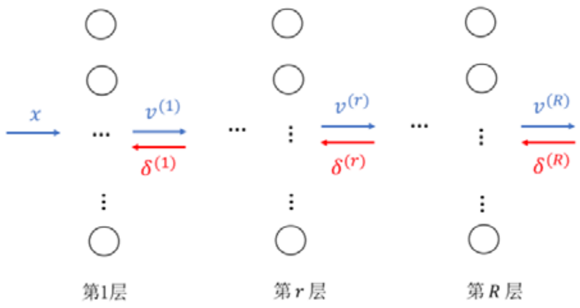

图 8.17　反向传播递推过程示意图

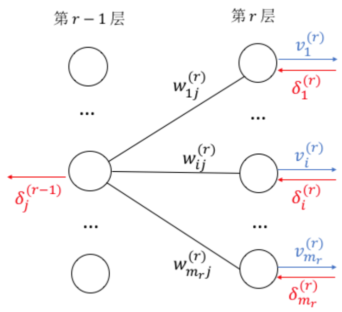

图 8.18　从 $\boldsymbol{\delta}^{(r)}$ 推导 $\boldsymbol{\delta}^{(r-1)}$

反向传播算法

BackProp$\left(\boldsymbol{v}^{(R)}, \boldsymbol{y}\right)$:

$$\boldsymbol{\delta}^{(R)} = \frac{\partial L}{\partial \boldsymbol{v}^{(R)}}$$

For $r = R, R-1, \ldots, 1$:

$$\boldsymbol{d}^{(r)} = \boldsymbol{\delta}^{(r)} \circ \sigma'\left(\boldsymbol{s}^{(r)}\right)$$

$$\frac{\partial L}{\partial \boldsymbol{W}^{(r)}} = \boldsymbol{d}^{(r)} \boldsymbol{v}^{(r-1)^T}$$

$$\frac{\partial L}{\partial \boldsymbol{b}^{(r)}} = \boldsymbol{d}^{(r)}$$

$$\boldsymbol{\delta}^{(r-1)} = \boldsymbol{W}^{(r)^T} \boldsymbol{d}^{(r)}$$

Return $\frac{\partial L}{\partial \boldsymbol{W}^{(r)}}, \frac{\partial L}{\partial \boldsymbol{b}^{(r)}}$, $r = 1, 2, \ldots, R$

图 8.19　反向传播算法

machine_learning.lib.nn_activators

```
class IdentityActivator:
    def value(self, s):
        return s

    def derivative(self, s):
        return 1

class ReLUActivator:
    def value(self, s):
        return np.maximum(0, s)

    def derivative(self, s):
        return (s > 0).astype(np.int)
```

图 8.20　激活函数

machine_learning.lib.nn_layers

```
import numpy as np

class Layer:
    def __init__(self, n_input, n_output, activator = IdentityActivator()):
        self.activator = activator
        self.b = np.zeros((n_output, 1))
        r = np.sqrt(6.0 / (n_input + n_output))
        self.W = np.random.uniform(-r, r, (n_output, n_input))
        self.outputs = np.zeros((n_output, 1))

    def forward(self, inputs):
        self.inputs = inputs
        self.sums = self.W.dot(inputs)+ self.b
        self.outputs = self.activator.value(self.sums)

    def back_propagation(self, delta_in, learning_rate):
        d = self.activator.derivative(self.sums) * delta_in
        self.delta_out = self.W.T.dot(d)
        self.W_grad = d.dot(self.inputs.T)
        self.b_grad = d
        self.W -= learning_rate * self.W_grad
        self.b -= learning_rate * self.b_grad
```

图 8.21　神经元层的算法实现

machine_learning.lib.nn_loss

```
import numpy as np

class MSE:
    def value(self, y, v):
        return (v - y) ** 2

    def derivative(self, y, v):
        return 2 * (v - y)

def softmax(v):
    e = np.exp(v)
    s = e.sum(axis=0)
    for i in range(len(s)):
        e[i] /= s[i]
    return e

class SoftmaxCrossEntropy:
    def value(self, y, v):
        p = softmax(v)
        return - (y * np.log(p)).sum()

    def derivative(self, y, v):
        p = softmax(v)
        return p - y
```

图 8.22　神经网络模型的损失函数

machine_learning.lib.neural_network

```python
import numpy as np

class NeuralNetwork:
    def __init__(self, layers, loss):
        self.layers = layers
        self.loss = loss

    def forward(self, x):
        layers = self.layers
        inputs = x
        for layer in layers:
            layer.forward(inputs)
            inputs = layer.outputs
        return inputs

    def back_propagation(self, y, outputs, learning_rate):
        delta_in = self.loss.derivative(y, outputs)
        for layer in self.layers[::-1]:
            layer.back_propagation(delta_in, learning_rate)
            delta_in = layer.delta_out

    def fit(self, X, y, N, learning_rate):
        for t in range(N):
            i = np.random.randint(0, len(X))
            outputs = self.forward(X[i].reshape(-1,1))
            self.back_propagation(y[i].reshape(-1,1), outputs, learning_rate)

    def predict(self, X):
        y = []
        for i in range(len(X)):
            v = self.forward(X[i].reshape(-1,1)).reshape(-1)
            y.append(v)
        return np.array(y)
```

图 8.23　神经网络算法

```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import machine_learning.lib.nn_activators as nn_activators
import machine_learning.lib.nn_layers as nn_layers
import machine_learning.lib.nn_loss as nn_loss
from machine_learning.lib.neural_network import NeuralNetwork
from sklearn.metrics import r2_score

def process_features(X):
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    scaler = MinMaxScaler(feature_range=(-1,1))
    X = scaler.fit_transform(X)
    return X

def create_layers():
    n_inputs = 8
    n_hidden1 = 100
    n_hidden2 = 50
    n_outputs = 1
    layers = []
    relu = nn_activators.ReLUActivator()
    layers.append(nn_layers.Layer(n_inputs, n_hidden1, activator = relu))
    layers.append(nn_layers.Layer(n_hidden1, n_hidden2, activator = relu))
    layers.append(nn_layers.Layer(n_hidden2, n_outputs))
    return layers

housing = fetch_california_housing()
X = housing.data
y = housing.target.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
X_train = process_features(X_train)
X_test = process_features(X_test)

layers = create_layers()
```

```
loss = nn_loss.MSE()
model = NeuralNetwork(layers, loss)
model.fit(X_train, y_train, 100000, 0.01)
y_pred = model.predict(X_test)
print(r2_score(y_test, y_pred))
```

图 8.24 房价预测的神经网络算法

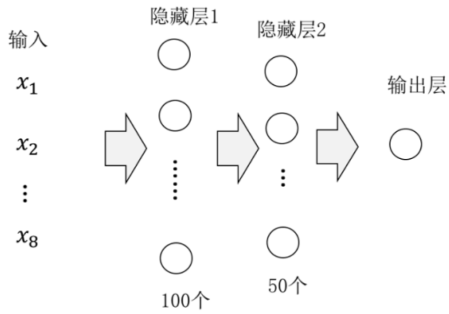

图 8.25 房价预测的神经网络模型

```python
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import machine_learning.lib.nn_activators as nn_activators
import machine_learning.lib.nn_layers as nn_layer
import machine_learning.lib.nn_loss as nn_loss
from machine_learning.lib.neural_network import NeuralNetwork
from sklearn.metrics import accuracy_score

def create_layers():
    n_features = 28 * 28
    n_hidden1 = 300
    n_hidden2 = 100
    n_classes = 10
    layers = []
    relu = nn.ReLUActivator()
    layers.append(nn.Layer(n_features, n_hidden1, activator = relu))
    layers.append(nn.Layer(n_hidden1, n_hidden2, activator = relu))
    layers.append(nn.Layer(n_hidden2, n_classes))
    return layers

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
X_train, y_train = mnist.train.images, mnist.train.labels
X_test, y_test = mnist.test.images, mnist.test.labels

layers = create_layers()
loss = nn.SoftmaxCrossEntropy()
model = nn.NeuralNetwork(layers, loss)
model.fit(X_train, y_train, 50000, 0.01)
v = model.predict(X_test)
proba = nn.softmax(v)
y_pred = np.argmax(proba, axis=1)
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
print("accuracy = {}".format(accuracy))
```

图 8.26　手写数字识别问题的神经网络算法

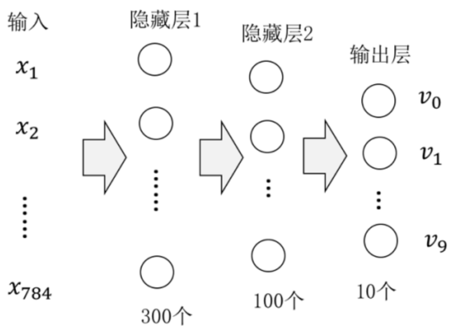

图 8.27　手写数字识别问题的神经网络模型

```python
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data

n_inputs = 28 * 28
n_hidden1 = 300
n_hidden2 = 100
n_classes = 10
X = tf.placeholder(tf.float32, shape=(None, n_inputs))
y = tf.placeholder(tf.int64, shape=(None, n_classes))
hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu)
hidden2 = tf.layers.dense(hidden1, n_hidden2, activation=tf.nn.relu)
outputs = tf.layers.dense(hidden2, n_classes)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=outputs)
loss = tf.reduce_mean(cross_entropy)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(outputs, 1))
accuracy_score = tf.reduce_mean(tf.cast(correct, tf.float32))

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    X_train, y_train = mnist.train.images, mnist.train.labels
    X_test, y_test = mnist.test.images, mnist.test.labels
    for t in range(50000):
        i = np.random.randint(0, len(X_train))
        X_i = X_train[i].reshape(1, -1)
        y_i = y_train[i].reshape(1, -1)
        sess.run(train_op, feed_dict = {X : X_i, y : y_i})
    accuracy = accuracy_score.eval(feed_dict = {X : X_test, y : y_test})
    print("accuracy = {}".format(accuracy))
```

图 8.28　TensorFlow 手写数字识别的神经网络