(a) (b)

图 9.1 两种视觉神经元对图像信号的接收方式

图 9.2　大脑获得完整图像信息的方式

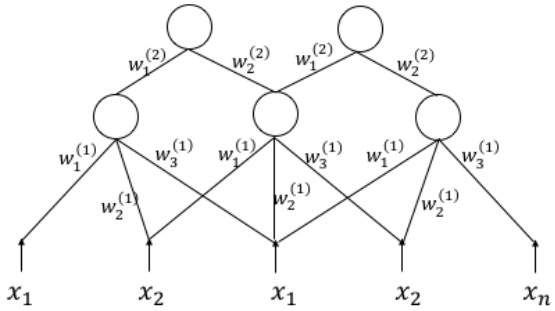


图 9.3 卷积神经网络示意图

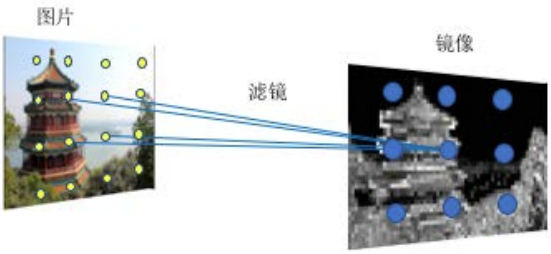


图 9.4 滤镜示意图

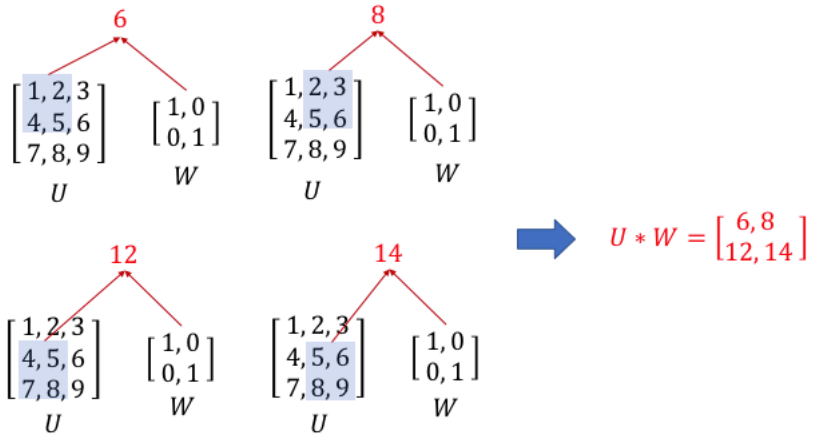


图 9.5　计算卷积 $\boldsymbol{U} * \boldsymbol{W}$

图 9.6　颐和园佛香阁

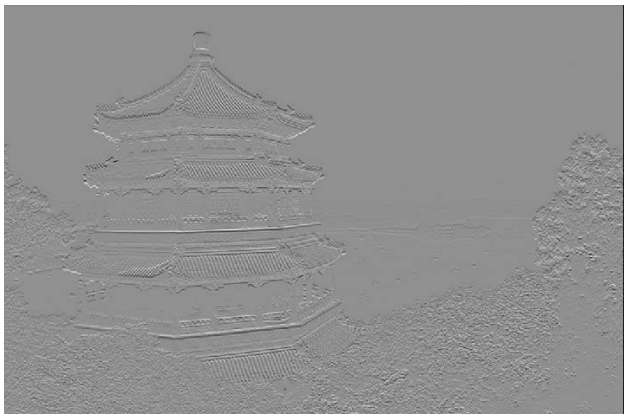


图 9.7　佛香阁的横向轮廓

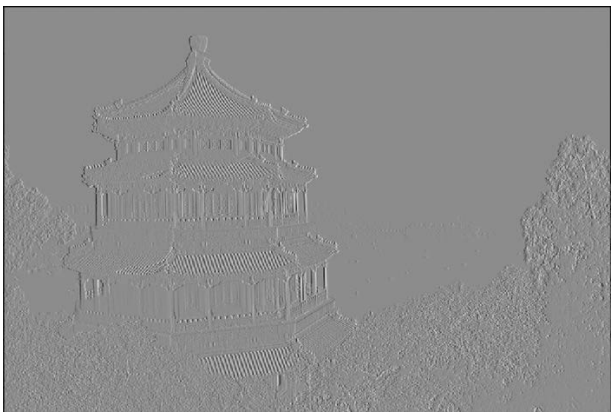


图 9.8　佛香阁的纵向轮廓

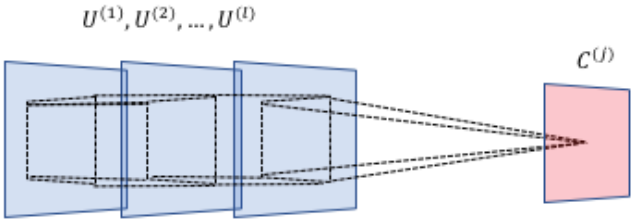


图 9.9 滤镜过滤 $\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}, \dots, \boldsymbol{U}^{(l)}$后镜像的叠加

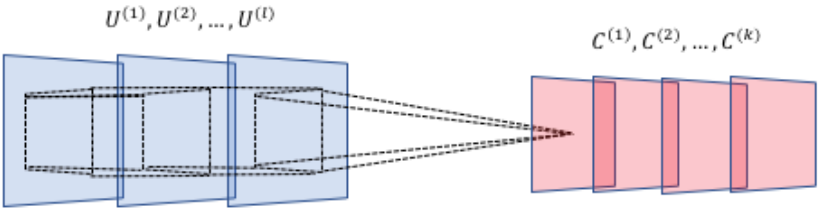


图 9.10　当前层输出的镜像 $\boldsymbol{C}^{(1)}, \boldsymbol{C}^{(2)}, \dots, \boldsymbol{C}^{(k)}$

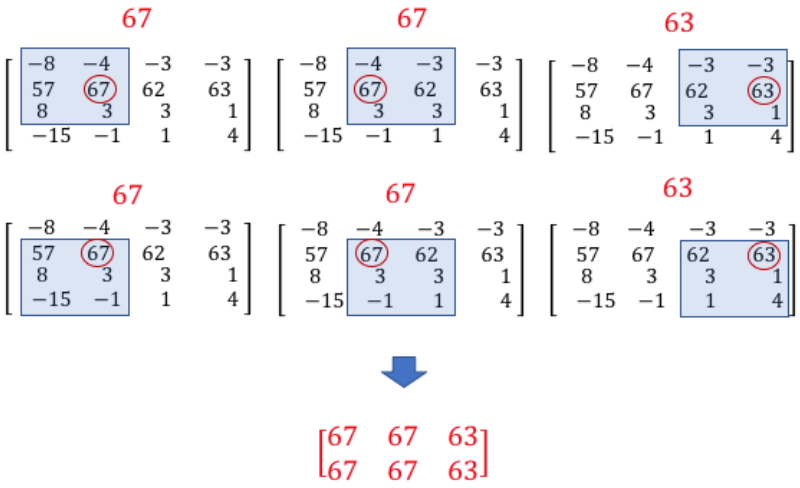


图 9.11　最大值池化过程

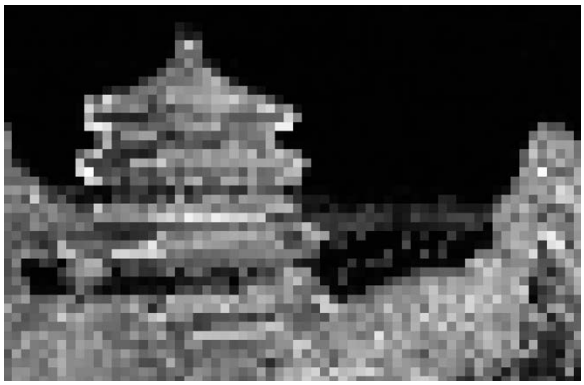


图 9.12　最大值池化后佛香阁的横向边界轮廓

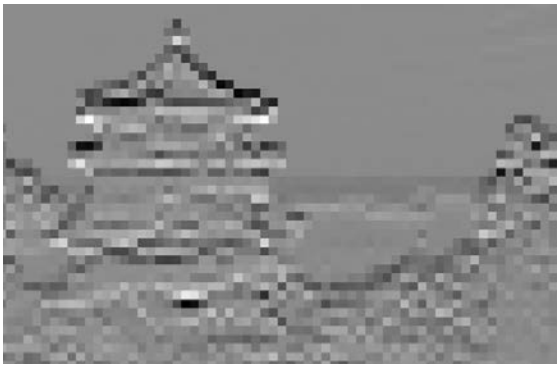


图 9.13　平均值池化后佛香阁的横向边界轮廓

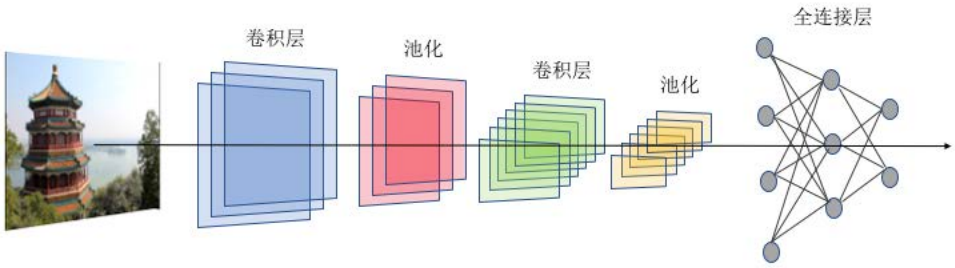


图 9.14　卷积神经网络的结构

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image

china = load_sample_image("china.jpg")
height, width, channels = china.shape
images = china.reshape(1, height, width, channels)
plt.figure(9.7)
plt.imshow(china)

X = tf.placeholder(tf.float32, shape = [1, height, width, channels])
filters = np.zeros(shape = (2, 1, 3, 1), dtype = np.float32)
filters[0, :, :, :] = -1
filters[1, :, :, :] = 1
convolution = tf.nn.conv2d(X, filters, strides = [1,1,1,1], padding = 'VALID')
max_pool = tf.nn.max_pool(convolution, ksize = [1,10,10,1],
                                    strides = [1,10,10,1], padding = 'SAME')

with tf.Session() as sess:
    conv_output = sess.run(convolution, feed_dict = {X : images})
    pool_output = sess.run(max_pool, feed_dict = {X : images})
    plt.figure(9.8)
    plt.imshow(conv_output[0, :, :, 0], cmap = "gray")
    plt.figure(9.12)
    plt.imshow(pool_output[0, :, :, 0], cmap = "gray")
```

图 9.15　卷积层与池化

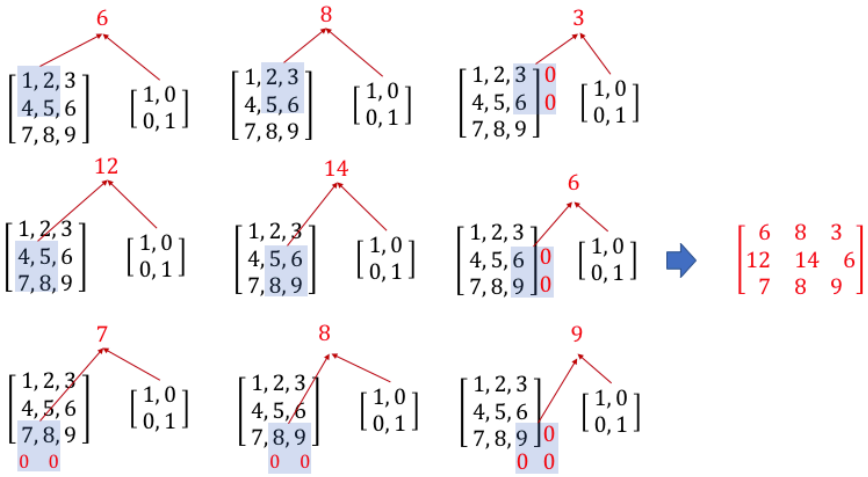


图 9.16　用填充全 0 的方法计算卷积

图 9.17　训练数据集中的图片采样

machine_learning.cnn.cats_and_dogs.data_reader

```
import numpy as np
import cv2
import os
import glob

def load_data(data_path, image_size, classes):
    images = []
    labels = []
    for c in classes:
        index = classes.index(c)
        path = os.path.join(data_path, c+'*.jpg')
        files = glob.glob(path)
        for file in files:
            image = cv2.imread(file)
            image = cv2.resize(image, (image_size, image_size), 0, 0, cv2.INTER_LINEAR)
            image = np.multiply(image, 1.0 / 255.0)
            images.append(image)
            label = np.zeros(len(classes))
            label[index] = 1
            labels.append(label)
    return np.array(images), np.array(labels)
```

图 9.18　读入猫和狗的图片

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from machine_learning.cnn.cats_and_dogs.data_reader import load_data

n_classes = 2
img_size = 128
n_channels = 3
X = tf.placeholder(tf.float32, shape = [None, img_size, img_size, n_channels])
y = tf.placeholder(tf.float32, shape = [None, n_classes])
conv1 = tf.layers.conv2d(X, filters = 32, kernel_size = [3,3],
                                    strides = [1,1], padding = 'SAME')
conv1_pool = tf.nn.max_pool(conv1, ksize = [1,2,2,1],
                                    strides = [1,2,2,1], padding = 'SAME')
conv2 = tf.layers.conv2d(conv1_pool, filters = 32, kernel_size = [3, 3],
                                    strides = [1,1], padding = 'SAME')
conv2_pool = tf.nn.max_pool(conv2, ksize = [1,2,2,1],
                                    strides = [1,2,2,1], padding = 'SAME')
conv3 = tf.layers.conv2d(conv2_pool, filters = 64, kernel_size = [3,3],
                                    strides=[1,1], padding = 'SAME')
conv3_pool = tf.nn.max_pool(conv3, ksize = [1,2,2,1],
                                    strides = [1,2,2,1], padding = 'SAME')
n_features = conv3_pool.get_shape()[1:4].num_elements()
conv_flat = tf.reshape(conv3_pool, [-1, n_features])
fc1 = tf.layers.dense(conv_flat, 128, activation=tf.nn.relu)
fc2 = tf.layers.dense(fc1, n_classes)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=fc2, labels=y)
cost = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)
y_pred = tf.nn.softmax(fc2)
correct_prediction = tf.equal(tf.argmax(y_pred, axis = 1), tf.argmax(y, axis = 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

saver = tf.train.Saver()
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    data_path = './ cats_and_dogs'
```

```
37      images, labels = load_data(data_path, img_size, ['dog','cat'])
38      image_train, image_test, label_train, label_test = train_test_split(
39              images, labels, test_size=0.1, random_state=0)
40      n_epoches = 20
41      batch_size = 32
42      best_accuracy = 0
43      for epoch in range(n_epoches):
44          for batch in range(len(image_train) // batch_size):
45              start = batch * batch_size
46              end = (batch + 1) * batch_size
47              X_batch = image_train[start:end]
48              y_batch = label_train[start:end]
49              sess.run(optimizer, feed_dict = {X:X_batch, y:y_batch})
50          test_accuracy = accuracy.eval(feed_dict = {X:image_test, y:label_test})
51          print("epoch {}: test accuracy = {}".format(epoch, test_accuracy))
52          if test_accuracy > best_accuracy:
53              saver.save(sess, "./cats_and_dogs_model/cats_and_dogs.ckpt")
54              best_accuracy = test_accuracy
```

图 9.19　猫和狗图片识别问题的卷积神经网络

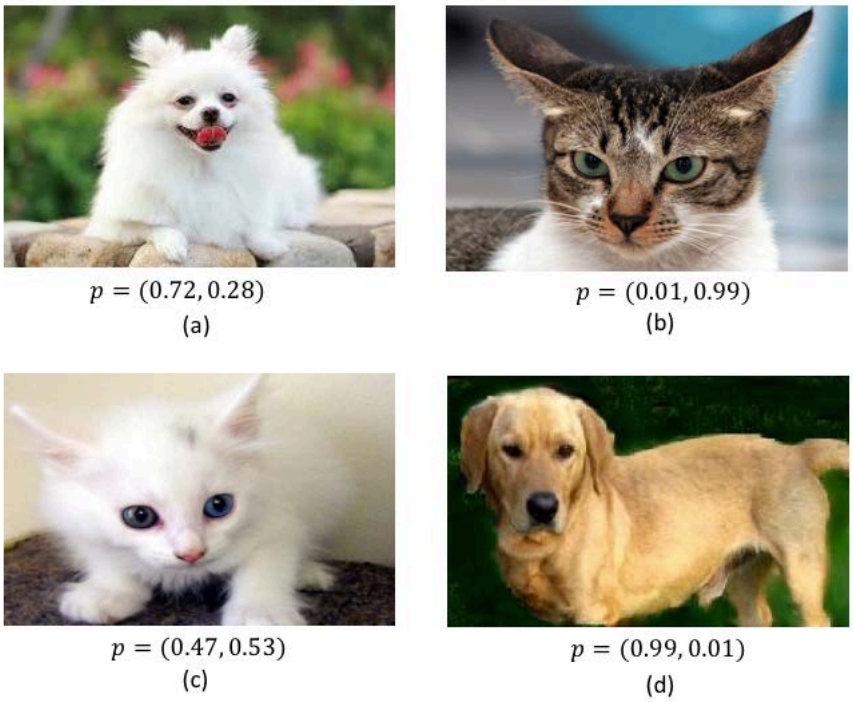
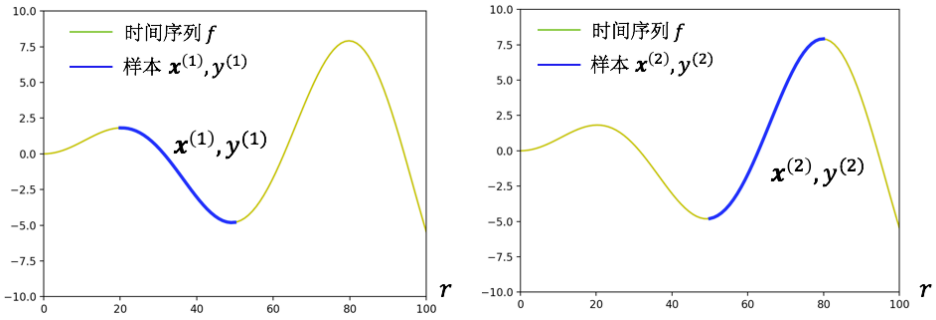


图 9.20　卷积神经网络对 4 个测试数据的概率预测



(a) (b)

图 9.21　起点是 20 和50的两个长度为 30 的时间序列

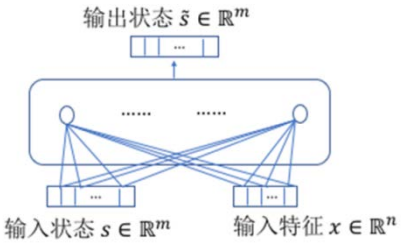


图 9.22　记忆单元的示意图

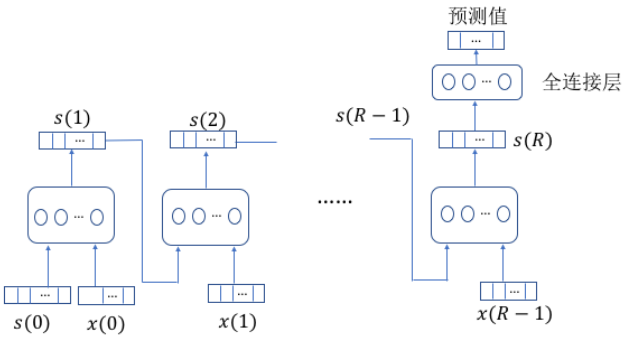


图 9.23 循环神经网络的结构示意图

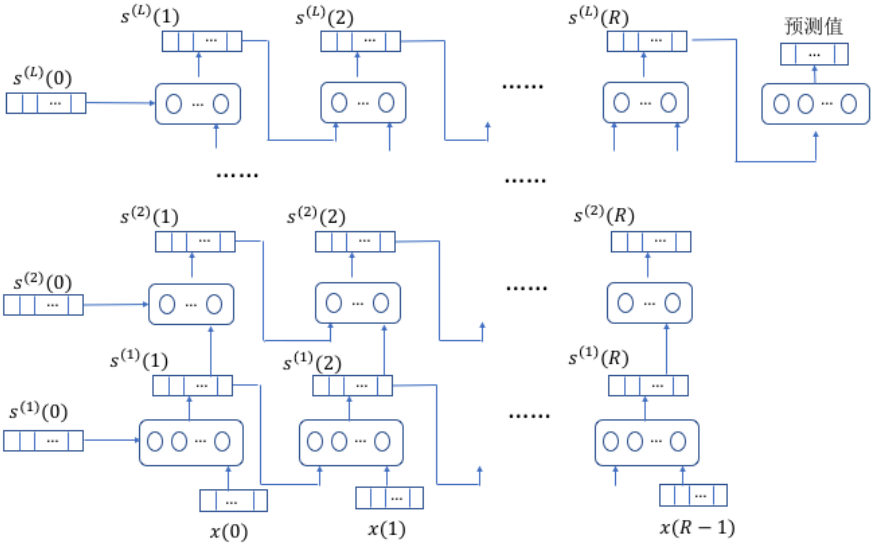


图 9.24　多层循环神经网络的结构

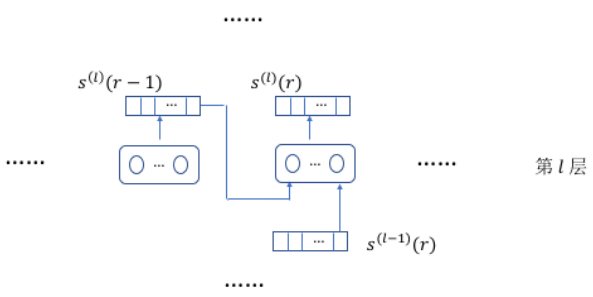


图 9.25　多层循环神经网络的第 l 层示意图

```
import tensorflow as tf
import numpy as np

def time_series(r):
    return r / 10.0 * np.sin(r / 10.0)

def get_samples(n_samples, r_max, R):
    r0 = np.random.rand(n_samples, 1) * (r_max – R)
    r = r0 + np.arange(0, R + 1)
    f = time_series(r)
    x = f[:, 0:R].reshape(-1, R, 1)
    y = f[:, R].reshape(-1, 1)
    return x, y

n_inputs = 1
n_outputs = 1
X = tf.placeholder(tf.float32, [None, 30, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])
cell = tf.contrib.rnn.BasicRNNCell(num_units = 50, activation = tf.nn.relu)
states, final_state = tf.nn.dynamic_rnn(cell, X, dtype = tf.float32)
preds = tf.layers.dense(final_state, n_outputs)
loss = tf.reduce_mean(tf.square(preds - y))
optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
training_op = optimizer.minimize(loss)

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    R, rmax = 30, 100
    X_train, y_train = get_samples(100, r_max, R)
    X_test, y_test = get_samples(100, r_max, R)
    for iteration in range(1000):
        sess.run(training_op, feed_dict = {X: X_train, y: y_train})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict = {X: X_test, y: y_test})
            print("iteration {} : MSE = {}".format(iteration, mse))
```

图 9.26　时间序列自回归问题的循环神经网络算法

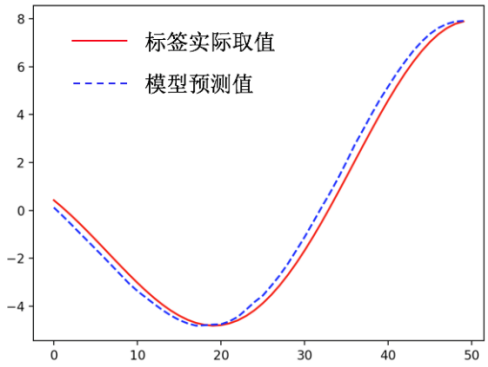


图 9.27　样本时间序列标签值与预测值

```
import tensorflow as tf
import numpy as np

def time_series(r):
    return r / 10.0 * np.sin(r / 10.0)

def get_samples(n_samples, r_max, R):
    r0 = np.random.rand(n_samples, 1) * (r_max - R)
    r = r0 + np.arange(0, R + 1)
    f = time_series(r)
    x = f[:, 0:R].reshape(-1, R, 1)
    y = f[:, R].reshape(-1, 1)
    return x, y

n_inputs = 1
n_outputs = 1
X = tf.placeholder(tf.float32, [None, R, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])
layers = []
layer_1 = tf.contrib.rnn.BasicRNNCell(num_units = 50, activation = tf.nn.relu)
layer_2 = tf.contrib.rnn.BasicRNNCell(num_units = 20, activation = tf.nn.relu)
layers.append(layer_1)
layers.append(layer_2)
multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
states, final_state = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype = tf.float32)
preds = tf.layers.dense(final_state[-1], n_outputs)
loss = tf.reduce_mean(tf.square(preds - y))
optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
training_op = optimizer.minimize(loss)

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    R, r_max = 30, 100
    X_train, y_train = get_samples(100, r_max, R)
    X_test, y_test = get_samples(100, r_max, R)
    for iteration in range(1000):
        sess.run(training_op, feed_dict={X: X_train, y: y_train})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_test, y: y_test})
            print("iteration {} : MSE = {}".format(iteration, mse))
```

图 9.28　时间序列自回归问题的多层循环神经网络算法

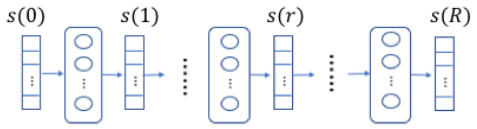


图 9.29　循环神经网络中的反向传播算法

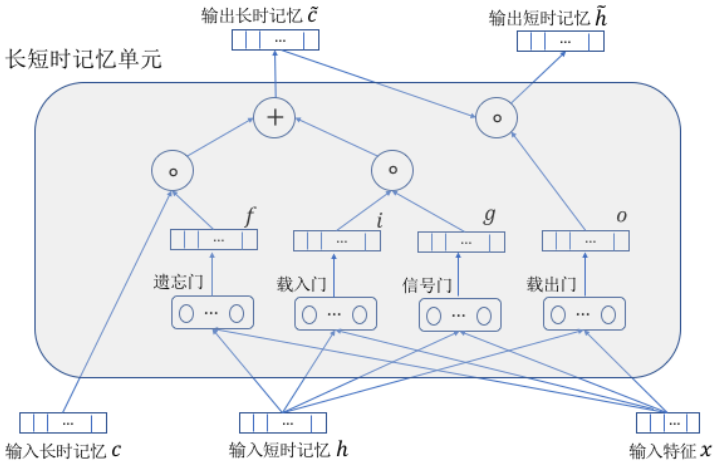


图 9.30　长短时记忆单元的内部结构

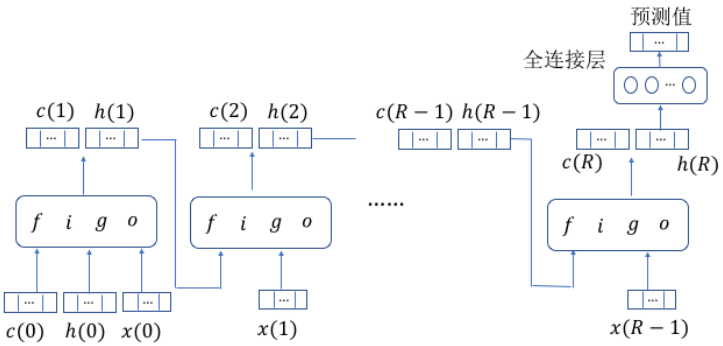


图 9.31　含有长短时记忆单元的循环神经网络示意图

```
import tensorflow as tf
import numpy as np

def time_series(r):
    return r / 10.0 * np.sin(r / 10.0)

def get_samples(n_samples, r_max, R):
    r0 = np.random.rand(n_samples, 1) * (r_max - R)
    r = r0 + np.arange(0.0, R + 1)
    f = time_series(r)
    x = f[:, 0:R].reshape(-1, R, 1)
    y = f[:, R].reshape(-1, 1)
    return x, y

n_inputs = 1
n_outputs = 1
num_units = 50
X = tf.placeholder(tf.float32, [None, R, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])
lstm_cell = tf.contrib.rnn.BasicLSTMCell(num_units)
states, final_state = tf.nn.dynamic_rnn(lstm_cell, X, dtype = tf.float32)
preds = tf.layers.dense(final_state.h, n_outputs)
loss = tf.reduce_mean(tf.square(preds - y))
optimizer = tf.train.AdamOptimizer(learning_rate = 0.001)
training_op = optimizer.minimize(loss)

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    R, r_max = 30, 100
    X_train, y_train = get_samples(100, r_max, R)
    X_test, y_test = get_samples(100, r_max, R)
    for iteration in range(1000):
        sess.run(training_op, feed_dict={X: X_train, y: y_train})
        if iteration % 100 == 0:
            mse = loss.eval(feed_dict={X: X_test, y: y_test})
            print("iteration {} : MSE = {}".format(iteration, mse))
```

图 9.32 含有长短时记忆单元的循环神经网络算法