人工神经网络

GWL

（北京理工大学数学与统计学院，北京 100081）

目录

第 1 章 实验介绍

本实验通过总结神经网络原理，人工搭建了一个简单的三层神经网络，使用 MNIST 数据集，通过误差回传来更新权重。训练完成后，在测试集上测试神经网络，采用数字的识别准确率来评估神经网络的性能。将 0～9 这十个标签反向输入神经网络中，输出神经网络“心里”的数字图像。传入自己手写的 0～9 这十个数字，使用神经网络进行识别，并计算准确率。采用控制变量法，分别调整学习率、训练代数和隐藏层神经元个数，观察神经网络性能的变化情况。最后，对本实验做一个总结。

第 2 章 实验环境

Windows 10，Python 3.10.4

第 3 章 实验原理

3.1 整合函数与激活函数

对每一个神经元，设输入为 X ，整合函数为 g ，激活函数为 f ，输出为 Y ，则

$$Y = f(g(X))$$

在本实验中，整合函数选用的是加权求和函数，即

$$g(X) = \sum_{i=1}^{n} w_i x_i$$

激活函数选用的是 sigmoid 函数，即

$$f(X) = \frac{1}{1 + e^{-X}}$$

3.2 神经网络的结构

为简单起见，本实验使用的是只包含一个输入层，一个输出层和一个隐藏层的三层神经网络，将输入和输出都用矩阵表示。

设 I_{input} 为输入矩阵，W_{input_hidden} 为输入层与隐藏层之间的权重矩阵，X_{hidden} 为隐藏层的输入矩阵，O_{hidden} 为隐藏层的输出矩阵，W_{hidden_output} 为隐藏层与输出层之间的权重矩阵，X_{output} 表示输出层的输入矩阵，O_{output} 表示最终的输出。

$$
\begin{aligned}
X_{hidden} &= W_{input\_hidden} \cdot I_{input} \\
O_{hidden} &= \text{sigmoid}(X_{hidden}) \\
X_{output} &= W_{hidden\_output} \cdot O_{hidden} \\
O_{output} &= \text{sigmoid}(X_{output})
\end{aligned}
$$

3.3 更新权重

为了增加神经网络的准确率，我们需要将误差最小化。

定义误差为 $E = \frac{1}{2}\sum_{i=1}^{n}(y_i - d_i)^2$，输出层的误差为 E_{output}，隐藏层的误差为 E_{hidden}，则

$$
\begin{aligned}
E_{output} &= O_{output} - target \\
E_{hidden} &= W^T_{hidden_output} \cdot E_{output}
\end{aligned}
$$

设学习率为 η，则权重调整的方向应为梯度下降最快的方向，则

$$
w = w - \eta\frac{\partial e}{\partial w}
$$

代入求偏导得：

$$
\begin{aligned}
W_{hidden_output} &= W_{hidden_output} + \eta \times E_{output} \times O_{output} \times (1 - O_{output}) \\
W_{input_hidden} &= W_{input_hidden} + \eta \times E_{hidden} \times O_{hidden} \times (1 - O_{hidden})
\end{aligned}
$$

3.4 评价指标

在本实验中，采用神经网络在测试集中的预测准确率来衡量神经网络的性能大小，即

$$Performance = \frac{\text{正确预测数}}{\text{测试集总数}}$$

第 4 章 实验内容

4.1 初始化

初始化神经网络。

```
class neuralNetwork(object):

    def __init__(self, in_nodes=784, hide_nodes=520,
     ↪  out_nodes=10, learning_rate=0.1, epoch=5):
        self.in_nodes = in_nodes
        self.hide_nodes = hide_nodes
        self.out_nodes = out_nodes

        # 输入层到隐藏层的权重
        self.wih = np.random.normal(0.0, pow(self.in_nodes,
         ↪  -0.5), (self.hide_nodes, self.in_nodes))
        # 隐藏层到输出层的权重
        self.who = np.random.normal(0.0,
         ↪  pow(self.hide_nodes, -0.5), (self.out_nodes,
         ↪  self.hide_nodes))

        # 学习率
        self.learning_rate = learning_rate

        # 激活函数 sigmoid
        self.activation_function = lambda x:
         ↪  special.expit(x)
        self.inverse_activation_function = lambda x:
         ↪  special.logit(x)

```

```
        # 训练代数
        self.epoch = epoch

        # 训练数据
        self.train_data_list = []

        # 测试数据
        self.test_data_list = []
```

4.2 导入数据

从数据集中导入训练集和测试集。

```
# 导入数据
def load_data(self, train_set_path, test_set_path):
    with open(train_set_path, 'r') as f:
        train_data_list = f.readlines()

    with open(test_set_path, 'r') as f:
        test_data_list = f.readlines()

    self.train_data_list = train_data_list
    self.test_data_list = test_data_list
    return len(train_data_list), len(test_data_list)
```

4.3 训练与测试

训练神经网络时使用的默认参数为：学习率 learning rate = 0.1、训练代数 epoch = 5、隐藏层神经元数量 hide nodes = 520。

```
# 训练数据
def train(self, input_list, target_list):
    inputs = np.array(input_list, ndmin=2).T
    targets = np.array(target_list, ndmin=2).T

    hide_inputs = np.dot(self.wih, inputs)
    hide_outputs = self.activation_function(hide_inputs)

```

```
    final_inputs = np.dot(self.who, hide_outputs)
    final_outputs = self.activation_function(final_inputs)

    output_errors = targets - final_outputs
    hide_errors = np.dot(self.who.T, output_errors)

    # 更新权重
    self.who += self.learning_rate * np.dot((output_errors *
     ↪  final_outputs * (1.0 - final_outputs)),
     ↪  np.transpose(hide_outputs))
    self.wih += self.learning_rate * np.dot((hide_errors *
     ↪  hide_outputs * (1.0 - hide_outputs)),
     ↪  np.transpose(inputs))

# 预测数据
def predict(self, input_list):
    inputs = np.array(input_list, ndmin=2).T
    hide_inputs = np.dot(self.wih, inputs)
    hide_outputs = self.activation_function(hide_inputs)
    final_inputs = np.dot(self.who, hide_outputs)
    final_outputs = self.activation_function(final_inputs)
    return final_outputs


def train_data(self):
    # 训练数据
    for e in range(self.epoch):
        for record in self.train_data_list:
            all_values = record.split(',')
            inputs = (np.asfarray(all_values[1:]) / 255.0 *
             ↪  0.99) + 0.01
            target = np.zeros(self.out_nodes) + 0.01
            target[int(all_values[0])] = 0.99
            self.train(inputs, target)
            print("\r", np.random.normal(), end="\r")
    print("                                    \r训练完成！ ")

```

```
def test_data(self):
    scorecard = []
    right = 0
    total = 0
    for record in self.test_data_list:
        all_values = record.split(',')
        correct_label = int(all_values[0])
        inputs = (np.asfarray(all_values[1:]) / 255.0 *
        ↪  0.99) + 0.01
        outputs = self.predict(inputs)
        label = np.argmax(outputs)
        right += correct_label == label
        total += 1

    performance = right / total
    print("\rPerformance =", performance)
    return performance

def train_and_test(self):
    self.train_data()
    performance = self.test_data()
    return performance
```

经过测试，神经网络在测试集上的准确率为 97.61%，说明该神经网络对该数据集的识别能力很高。

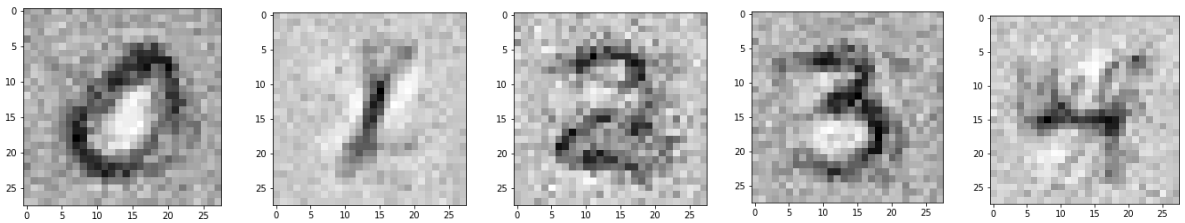
4.4 神经网络反向传播

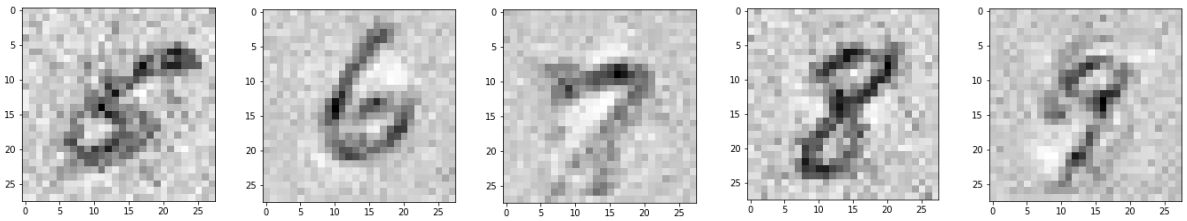
将 0 ~ 9 的标签反向输入到训练好的神经网络中，输出图像，就可以得到神经网络“所认为”的 0 ~ 9 是什么样子。

```
# 神经网络反向传播
def backpredict(self, targets_list):
    final_outputs = np.array(targets_list, ndmin=2).T
    final_inputs =
     ↪  self.inverse_activation_function(final_outputs)

```

```
    # 归一化处理到区间 [0.01, 0.99]
    hidden_outputs = np.dot(self.who.T, final_inputs)
    hidden_outputs -= np.min(hidden_outputs)
    hidden_outputs /= np.max(hidden_outputs)
    hidden_outputs *= 0.98
    hidden_outputs += 0.01

    hidden_inputs =
     ↪  self.inverse_activation_function(hidden_outputs)
    inputs = np.dot(self.wih.T, hidden_inputs)
    inputs -= np.min(inputs)
    inputs /= np.max(inputs)
    inputs *= 0.98
    inputs += 0.01

    return inputs

# 神经网络内部
def numbers_in_network(self):
    for i in range(self.out_nodes):
        targets = np.zeros(self.out_nodes) + 0.01
        targets[i] = 0.99
        # print(targets)
        image_data = self.backpredict(targets)
        plt.imshow(image_data.reshape(28, 28), cmap="Greys",
         ↪  interpolation="None")
        plt.savefig("./figures/{}_in_network.png".format(i))
```

保存图片，得到以下结果：

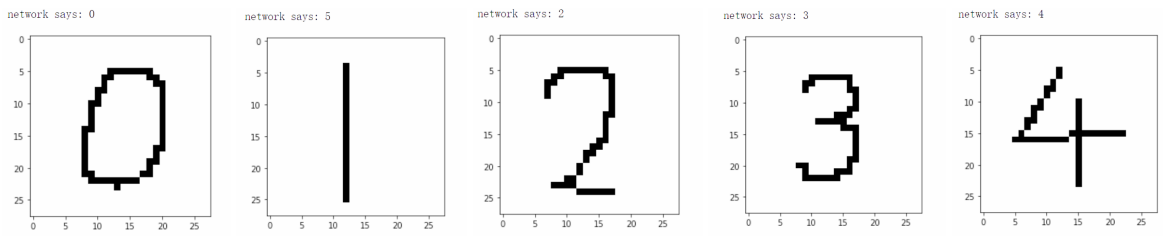




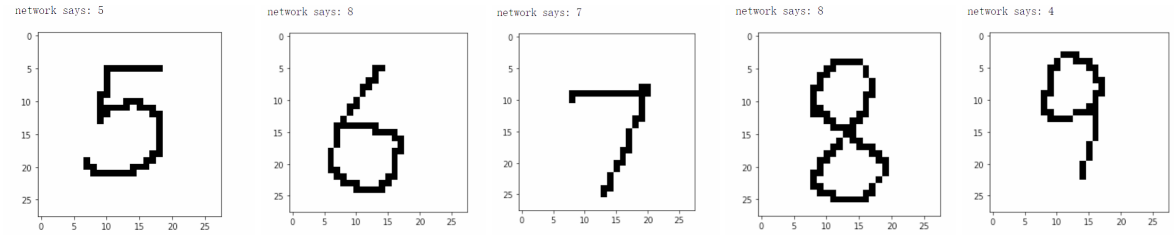
4.5 识别自己的手写数字

分别输入自己的手写数字 0 ~ 9，计算准确率。

```python
# 识别手写数字
def identify_numbers(self, cnt=10):
    right = 0
    for i in range(cnt):
        img_array = imageio.imread("figures/手写-{}.png".format(i), as_gray=True)
        img_data = 255.0 - img_array.reshape(28 * 28)
        img_data = (img_data / 255.0 * 0.99) + 0.01

        plt.imshow(img_data.reshape(28, 28), cmap="Greys", interpolation="None")
        outputs = self.predict(img_data)

        label = np.argmax(outputs)
        print("network says: {}, actually is {}".format(label, i))
        right += label == i

    print("Accuracy:", right / cnt)
```



对我们自己的 10 个手写数字，神经网络的识别准确率为 70%，基本满足要求。



第 5 章 模型改进

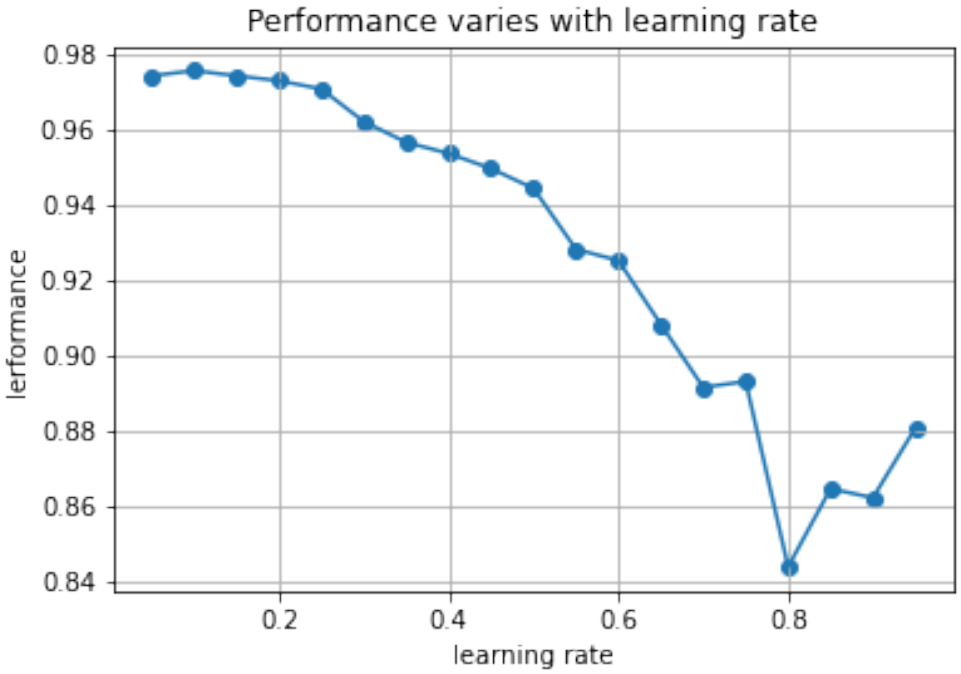
在我们的神经网络中，可变参数有三个：学习率（learning rate）、训练代数（epoch）、隐藏层神经元数量（hide nodes）。我们采用控制变量法，分别改变这三个参数，来获得神经网络的性能的变化趋势。

5.1 学习率

在不改变其他参数的情况下，调整学习率，绘制出神经网络性能随学习率的变化趋势。

```
def different_learning_rate(self):
    x, y = [], []
    origin_learning_rate = self.learning_rate
    origin_wih = self.wih
    origin_who = self.who

    self.wih = np.random.normal(0.0, pow(self.in_nodes,
     ↪  -0.5), (self.hide_nodes, self.in_nodes))
    self.who = np.random.normal(0.0, pow(self.hide_nodes,
     ↪  -0.5), (self.out_nodes, self.hide_nodes))

    self.learning_rate = 0.05
    while self.learning_rate < 1:
        x.append(self.learning_rate)
        y.append(self.train_and_test())
        self.learning_rate += 0.05
    print(x, "\n", y)
    self.draw_picture(x, y, "learning rate", "lerformance",
     ↪  "Performance varies with learning rate")
    self.learning_rate = origin_learning_rate
```

```
18      self.wih = origin_wih
19      self.who = origin_who
```



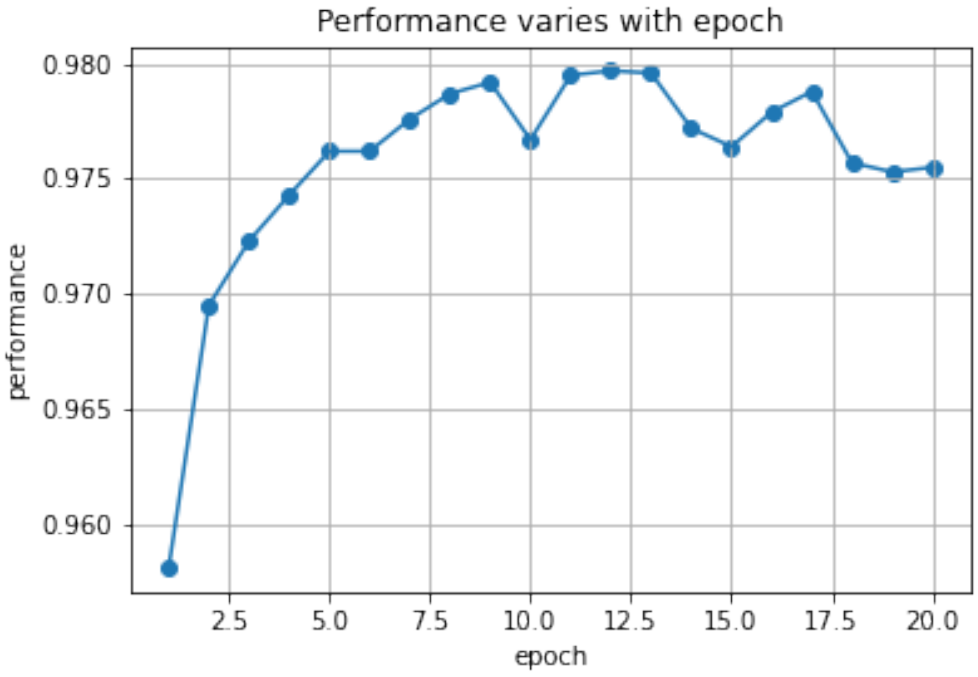
从图中我们可以看出，学习率较小时神经网络性能最好，随着学习率的增大，神经网络的性能呈现下降趋势，综合来看，学习率设置为 0.1 到 0.2 之间即可。

5.2 训练代数

在不改变其他参数的情况下，调整训练代数，绘制出神经网络性能随训练代数的变化趋势。

```
1  def different_epoch(self):
2      x, y = [], []
3      origin_epoch = self.epoch
4      origin_wih = self.wih
5      origin_who = self.who
6
7      self.wih = np.random.normal(0.0, pow(self.in_nodes,
       ↪  -0.5), (self.hide_nodes, self.in_nodes))
8      self.who = np.random.normal(0.0, pow(self.hide_nodes,
       ↪  -0.5), (self.out_nodes, self.hide_nodes))
```

```
9
10      self.epoch = 1
11      while self.epoch < 21:
12          x.append(self.epoch)
13          y.append(self.train_and_test())
14          self.epoch += 1
15      print(x, "\n", y)
16      self.draw_picture(x, y, "epoch", "performance",
        ↪  "Performance varies with epoch")
17      self.epoch = origin_epoch
18      self.wih = origin_wih
19      self.who = origin_who
```



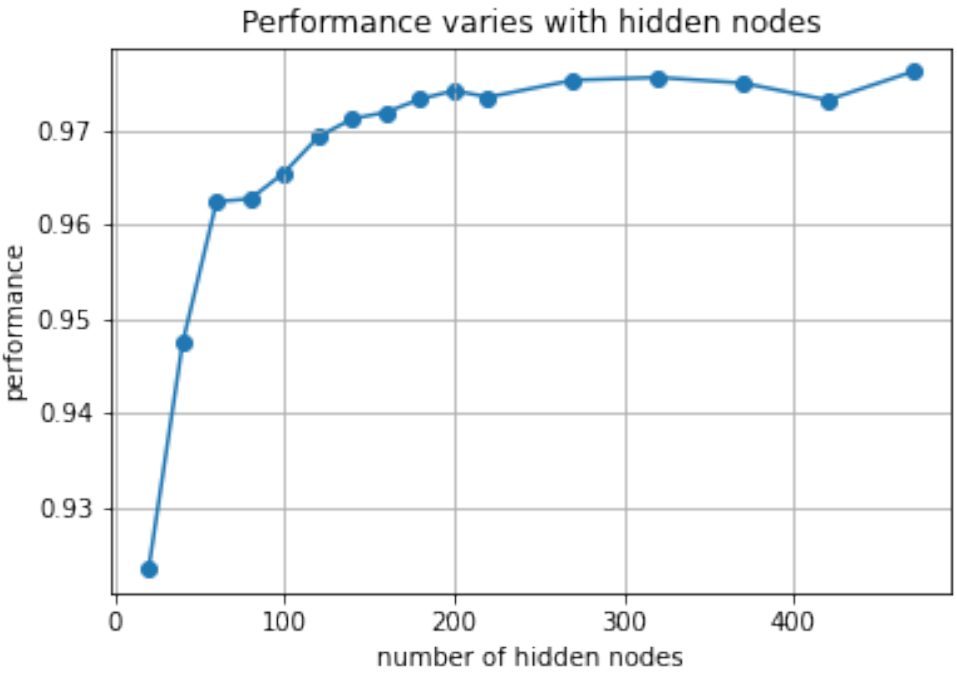
从图中可以看出，训练代数较小时，神经网络的性能较差（但准确率也超过了 95%），在训练代数超过 5 之后，性能已经较好，准确率在 97.5% 到 98.0% 之间波动，考虑到训练代数越多，训练时间越长，所以将训练代数设置为 5 到 8 之间比较合适。

5.3 隐藏层神经元数量

在不改变其他参数的情况下，调整隐藏层神经元数量，绘制出神经网络性能随隐藏层神经元数量的变化趋势。

```
def different_hide_nodes(self):
    x, y = [], []
    origin_hide_nodes = self.hide_nodes
    origin_wih = self.wih
    origin_who = self.who

    self.wih = np.random.normal(0.0, pow(self.in_nodes,
     ↪  -0.5), (self.hide_nodes, self.in_nodes))
    self.who = np.random.normal(0.0, pow(self.hide_nodes,
     ↪  -0.5), (self.out_nodes, self.hide_nodes))

    self.hide_nodes = 20
    while self.hide_nodes < 520:
        x.append(self.hide_nodes)
        y.append(self.train_and_test())
        if self.hide_nodes <= 200:
            self.hide_nodes += 20
        else:
            self.hide_nodes += 50
    print(x, "\n", y)
    self.draw_picture(x, y, "number of hidden nodes",
     ↪  "performance", "Performance varies with hidden
     ↪  nodes")
    self.hide_nodes = origin_hide_nodes
    self.wih = origin_wih
    self.who = origin_who
```

从图中可以看出，随着隐藏层神经元数量的增多，神经网络性能先大幅上涨，在超过 200 后性能变化不大，所以在训练时把隐藏层神经元个数设置为 180 到 200 之间比较合适。



第 6 章 实验总结

本实验所搭建的神经网络只是一个最简单的三层全连接神经网络，后续还可以改变网络结构，搭建出识别率更高，性能更好的网络。

在实际手写数字的识别中，评估神经网络所用的图片数量较少，在后期评估时应该适当增加数量，以便更好地体现神经网络的性能。

在参数调整时，本实验仅采取了单一变量的原则，实际上，各个参数之间应该存在着相互的影响，在参数调整时应当综合考虑各个参数。

综上，本实验还有很多进步的空间，可以在后续实验中进行改进。

第 7 章 完整代码

```
1 import imageio
2 import numpy as np
3 from matplotlib import pyplot as plt
4 from scipy import special
5
6
```

```
7  class neuralNetwork(object):
8
9      def __init__(self, in_nodes=784, hide_nodes=520,
       ↪  out_nodes=10, learning_rate=0.1, epoch=5):
10         self.in_nodes = in_nodes
11         self.hide_nodes = hide_nodes
12         self.out_nodes = out_nodes
13
14         # 输入层到隐藏层的权重
15         self.wih = np.random.normal(0.0, pow(self.in_nodes,
           ↪  -0.5), (self.hide_nodes, self.in_nodes))
16         # 隐藏层到输出层的权重
17         self.who = np.random.normal(0.0,
           ↪  pow(self.hide_nodes, -0.5), (self.out_nodes,
           ↪  self.hide_nodes))
18
19         # 学习率
20         self.learning_rate = learning_rate
21
22         # 激活函数 sigmoid
23         self.activation_function = lambda x:
           ↪  special.expit(x)
24         self.inverse_activation_function = lambda x:
           ↪  special.logit(x)
25
26         # 训练代数
27         self.epoch = epoch
28
29         # 训练数据
30         self.train_data_list = []
31
32         # 测试数据
33         self.test_data_list = []
34
35     # 导入数据
36     def load_data(self, train_set_path, test_set_path):
37         with open(train_set_path, 'r') as f:
```

```
            train_data_list = f.readlines()

        with open(test_set_path, 'r') as f:
            test_data_list = f.readlines()

        self.train_data_list = train_data_list
        self.test_data_list = test_data_list
        return len(train_data_list), len(test_data_list)

    # 训练数据
    def train(self, input_list, target_list):
        inputs = np.array(input_list, ndmin=2).T
        targets = np.array(target_list, ndmin=2).T

        hide_inputs = np.dot(self.wih, inputs)
        hide_outputs = self.activation_function(hide_inputs)

        final_inputs = np.dot(self.who, hide_outputs)
        final_outputs =
         ↪  self.activation_function(final_inputs)

        output_errors = targets - final_outputs
        hide_errors = np.dot(self.who.T, output_errors)

        # 更新权重
        self.who += self.learning_rate *
         ↪  np.dot((output_errors * final_outputs * (1.0 -
         ↪  final_outputs)), np.transpose(hide_outputs))
        self.wih += self.learning_rate * np.dot((hide_errors
         ↪  * hide_outputs * (1.0 - hide_outputs)),
         ↪  np.transpose(inputs))

    # 预测数据
    def predict(self, input_list):
        inputs = np.array(input_list, ndmin=2).T
        hide_inputs = np.dot(self.wih, inputs)
        hide_outputs = self.activation_function(hide_inputs)
```

```
70          final_inputs = np.dot(self.who, hide_outputs)
71          final_outputs =
             ↪  self.activation_function(final_inputs)
72          return final_outputs
73
74
75      def train_data(self):
76          # 训练数据
77          for e in range(self.epoch):
78              for record in self.train_data_list:
79                  all_values = record.split(',')
80                  inputs = (np.asfarray(all_values[1:]) /
                     ↪  255.0 * 0.99) + 0.01
81                  target = np.zeros(self.out_nodes) + 0.01
82                  target[int(all_values[0])] = 0.99
83                  self.train(inputs, target)
84                  print("\r", np.random.normal(), end="\r")
85          print("                                        \r训练完成!
             ↪  ")
86
87      def test_data(self):
88          scorecard = []
89          right = 0
90          total = 0
91          for record in self.test_data_list:
92              all_values = record.split(',')
93              correct_label = int(all_values[0])
94              inputs = (np.asfarray(all_values[1:]) / 255.0 *
                 ↪  0.99) + 0.01
95              outputs = self.predict(inputs)
96              label = np.argmax(outputs)
97              right += correct_label == label
98              total += 1
99
100         performance = right / total
101         print("\rPerformance =", performance)
102         return performance
```

```
103
104     def train_and_test(self):
105         self.train_data()
106         performance = self.test_data()
107         return performance
108
109     # 神经网络反向传播
110     def backpredict(self, targets_list):
111         final_outputs = np.array(targets_list, ndmin=2).T
112         final_inputs =
             ↪  self.inverse_activation_function(final_outputs)
113
114         # 归一化处理到区间 [0.01, 0.99]
115         hidden_outputs = np.dot(self.who.T, final_inputs)
116         hidden_outputs -= np.min(hidden_outputs)
117         hidden_outputs /= np.max(hidden_outputs)
118         hidden_outputs *= 0.98
119         hidden_outputs += 0.01
120
121         hidden_inputs =
             ↪  self.inverse_activation_function(hidden_outputs)
122         inputs = np.dot(self.wih.T, hidden_inputs)
123         inputs -= np.min(inputs)
124         inputs /= np.max(inputs)
125         inputs *= 0.98
126         inputs += 0.01
127
128         return inputs
129
130     # 神经网络内部
131     def numbers_in_network(self):
132         for i in range(self.out_nodes):
133             targets = np.zeros(self.out_nodes) + 0.01
134             targets[i] = 0.99
135             # print(targets)
136             image_data = self.backpredict(targets)
137             plt.imshow(image_data.reshape(28, 28),
                 ↪  cmap="Greys", interpolation="None")
```

```
138             plt.savefig("./figures/{}_in_network.png".format(i))
139
140     # 识别手写数字
141     def identify_numbers(self, cnt=10):
142         right = 0
143         for i in range(cnt):
144             img_array = imageio.imread("figures/手
                ↪ 写-{}.png".format(i), as_gray=True)
145             img_data = 255.0 - img_array.reshape(28 * 28)
146             img_data = (img_data / 255.0 * 0.99) + 0.01
147
148             plt.imshow(img_data.reshape(28, 28),
                ↪ cmap="Greys", interpolation="None")
149             outputs = self.predict(img_data)
150
151             label = np.argmax(outputs)
152             print("network says: {}, actually is
                ↪ {}".format(label, i))
153             right += label == i
154
155         print("Accuracy:", right / cnt)
156
157     def draw_picture(self, x=[], y=[], x_lable='x',
        ↪ y_lable='y', title='title'):
158         plt.scatter(x, y)
159         plt.plot(x, y)
160         plt.xlabel(x_lable)
161         plt.ylabel(y_lable)
162         plt.title(title)
163         plt.grid()
164         plt.savefig("./figures/" + title + ".png")
165         plt.show()
166         plt.close()
167
168     def different_learning_rate(self):
169         x, y = [], []
170         origin_learning_rate = self.learning_rate
```

```
171         origin_wih = self.wih
172         origin_who = self.who
173 
174         self.wih = np.random.normal(0.0, pow(self.in_nodes,
            ↪ -0.5), (self.hide_nodes, self.in_nodes))
175         self.who = np.random.normal(0.0,
            ↪ pow(self.hide_nodes, -0.5), (self.out_nodes,
            ↪ self.hide_nodes))
176 
177         self.learning_rate = 0.05
178         while self.learning_rate < 1:
179             x.append(self.learning_rate)
180             y.append(self.train_and_test())
181             self.learning_rate += 0.05
182         print(x, "\n", y)
183         self.draw_picture(x, y, "learning rate",
            ↪ "lerformance", "Performance varies with learning
            ↪ rate")
184         self.learning_rate = origin_learning_rate
185         self.wih = origin_wih
186         self.who = origin_who
187 
188     def different_epoch(self):
189         x, y = [], []
190         origin_epoch = self.epoch
191         origin_wih = self.wih
192         origin_who = self.who
193 
194         self.wih = np.random.normal(0.0, pow(self.in_nodes,
            ↪ -0.5), (self.hide_nodes, self.in_nodes))
195         self.who = np.random.normal(0.0,
            ↪ pow(self.hide_nodes, -0.5), (self.out_nodes,
            ↪ self.hide_nodes))
196 
197         self.epoch = 1
198         while self.epoch < 21:
199             x.append(self.epoch)
```

```
200             y.append(self.train_and_test())
201             self.epoch += 1
202         print(x, "\n", y)
203         self.draw_picture(x, y, "epoch", "performance",
            ↪ "Performance varies with epoch")
204         self.epoch = origin_epoch
205         self.wih = origin_wih
206         self.who = origin_who
207 
208     def different_hide_nodes(self):
209         x, y = [], []
210         origin_hide_nodes = self.hide_nodes
211         origin_wih = self.wih
212         origin_who = self.who
213 
214         self.wih = np.random.normal(0.0, pow(self.in_nodes,
            ↪ -0.5), (self.hide_nodes, self.in_nodes))
215         self.who = np.random.normal(0.0,
            ↪ pow(self.hide_nodes, -0.5), (self.out_nodes,
            ↪ self.hide_nodes))
216 
217         self.hide_nodes = 20
218         while self.hide_nodes < 520:
219             x.append(self.hide_nodes)
220             y.append(self.train_and_test())
221             if self.hide_nodes <= 200:
222                 self.hide_nodes += 20
223             else:
224                 self.hide_nodes += 50
225         print(x, "\n", y)
226         self.draw_picture(x, y, "number of hidden nodes",
            ↪ "performance", "Performance varies with hidden
            ↪ nodes")
227         self.hide_nodes = origin_hide_nodes
228         self.wih = origin_wih
229         self.who = origin_who
230 
```