A.5 神经网络的数学推导和代码实现

A.5.1 数学推导

根据 2.4.2 节，假设训练集为$\{(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}),(\boldsymbol{x}^{(2)},\boldsymbol{y}^{(2)}),\ldots,(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)}),\ldots(\boldsymbol{x}^{(N)},\boldsymbol{y}^{(N)})\}$，即共有 N 个训练样本，神经网络模型对这 N 个训练样本的输出为$\{\boldsymbol{a}^{(1)},\boldsymbol{a}^{(2)},\ldots,\boldsymbol{a}^{(N)}\}$（为了书写方便，输出层的输出省略了上标 L），每一个目标输出$y^{(i)}=(y_1^{(i)},y_2^{(i)},\ldots y_{n^L}^{(i)})^T$。则对于某个数据$(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})$来说，其代价函数定义为：

$$
\begin{aligned}
E_i &= \frac{1}{2}||\boldsymbol{y}^{(i)}-\boldsymbol{a}^{(i)}|| \\
&= \frac{1}{2}\sum_{k=1}^{n^{(L)}}(y_k^{(i)}-a_k^{(i)})^2
\end{aligned}
\tag{A.45}
$$

模型在训练数据上的总体代价可表示为：

$$E_t=\frac{1}{N}\sum_{i=1}^{N}E_i \tag{A.46}$$

我们的目标就是不断调整权重和偏差使总体代价最小。根据梯度下降算法，可以用如下公式来更新参数。

$$
\begin{aligned}
\boldsymbol{W}^l &= \boldsymbol{W}^l-\eta\frac{\partial E_t}{\partial \boldsymbol{W}^l} \\
&= \boldsymbol{W}^l-\frac{\eta}{N}\sum_{i=1}^{N}\frac{\partial E_i}{\partial \boldsymbol{W}^l}
\end{aligned}
\tag{A.47}
$$

$$
\begin{aligned}
\boldsymbol{b}^l &= \boldsymbol{b}^l-\eta\frac{\partial E_t}{\partial \boldsymbol{b}^l} \\
&= \boldsymbol{b}^l-\frac{\eta}{N}\sum_{i=1}^{N}\frac{\partial E_i}{\partial \boldsymbol{b}^l}
\end{aligned}
\tag{A.48}
$$

由(A.47)和(A.48)可知，只需计算每一个训练数据的代价函数E_i对参数的偏导数$\frac{\partial E_i}{\partial \boldsymbol{W}^l}$和$\frac{\partial E_i}{\partial \boldsymbol{b}^l}$，即可得到参数更新公式。这里考虑每次只根据一个数据$(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})$进行参数调整，通过循环（遍历每一个数据）完成基于所有数据的一轮参数更新。下面给出参数更新方法的推导过程，为叙述方便，用 E、a 和 y 来代替上述的E_i、$a^{(i)}$和 $y^{(i)}$。

首先计算输出层权重的梯度。由求导链式法则，结合式(A.45)，对输出层权重参数求偏导，有：

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ij}^L} &= \frac{\partial E}{\partial a_i^L}\frac{\partial a_i^L}{\partial w_{ij}^L} \\
&= \frac{1}{2}*2(y_i-a_i^L)(-\frac{\partial a_i^L}{\partial w_{ij}^L}) \\
&= -\frac{1}{2}*2(y_i-a_i^L)\frac{\partial a_i^L}{\partial z_i^L}\frac{\partial z_i^L}{\partial w_{ij}^L} \\
&= -(y_i-a_i^L)g^{L\prime}(z_i^L)\ \frac{\partial z_i^L}{\partial w_{ij}^L} \\
&= -(y_i-a_i^L)g^{L\prime}(z_i^L)a_j^{L-1}
\end{aligned}
\tag{A.49}
$$

如果把$\frac{\partial E}{\partial z_i^l}$记做$\delta_i^l$，即：

$$\delta_i^l=\frac{\partial E}{\partial z_i^l} \tag{A.50}$$

则$\frac{\partial E}{\partial w_{ij}^L}$可以写为：

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ij}^L} &= \frac{\partial E}{\partial z_i^L}\frac{\partial z_i^L}{\partial w_{ij}^L} \\
&= \delta_i^L a_j^{L-1}
\end{aligned}
\tag{A.51}
$$

在后续的推导中我们会看到第 l 层的$\boldsymbol{\delta}^l$可以由第 l+1 层的$\boldsymbol{\delta}^{l+1}$得到，这是一个非常重要的性质。此时，我们可以得到输出层权重矩阵更新的两个公式：

$$\delta_i^L=-(y_i-a_i^L)g^{L\prime}(z_i^L),(1\le i\le n^L) \tag{A.52}$$

$$\frac{\partial E}{\partial w_{ij}^L}=\delta_i^L a_j^{L-1},(1\le i\le n^L,1\le j\le n^{L-1}) \tag{A.53}$$

将其表示成矩阵（向量）形式，则(A.52)和(A.53)可重写为：

$$\boldsymbol{\delta}^L=-(\boldsymbol{y}-\boldsymbol{a}^L)\odot g^{L\prime}(\boldsymbol{z}^L) \tag{A.54}$$

$$\nabla_{\boldsymbol{W}^L}E=\boldsymbol{\delta}^L(\boldsymbol{a}^{L-1})^T \tag{A.55}$$

其中，$\odot$为哈达玛积，表示同型矩阵对应项相乘得到新的矩阵；$\boldsymbol{\delta}^L$是一个n^L维列向量；$\nabla_{\boldsymbol{W}^L}E$是一个$n^L$行$n^{L-1}$列的矩阵；$g^{L\prime}(\boldsymbol{z}^L)$表示$g^L(\boldsymbol{z}^L)$的导数，是一个$n^L$维列向量。

同理，可对隐藏层神经元的权重参数求偏导得到这些权重参数的更新公式。利用δ_i^l的定义，有：

$$\frac{\partial E}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1}, (2 \le l \le L-1) \tag{A.56}$$

其中，δ_i^l的推导如下：

$$\begin{aligned}\delta_i^l &= \frac{\partial E}{\partial z_i^l} \\ &= \sum_{j=1}^{n_{l+1}} \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l} \\ &= \sum_{j=1}^{n_{l+1}} \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial z_i^l}\end{aligned} \tag{A.57}$$

由于$z_j^{l+1} = \sum_{i=1}^{n^l} w_{ji}^{l+1} a_i^l + b_j^{l+1} = \sum_{i=1}^{n^l} w_{ji}^{l+1} g^l(z_i^l) + b_j^{l+1}$，因此有$\frac{\partial z_j^{l+1}}{\partial z_i^l} = \frac{\partial z_j^{l+1}}{\partial a_i^l}\frac{\partial a_i^l}{\partial z_i^l} = w_{ji}^{l+1} g^{l\prime}(z_i^l)$。将其代入(A.57)，则有：

$$\begin{aligned}\delta_i^l &= \sum_{j=1}^{n_{l+1}} \delta_j^{l+1} w_{ji}^{l+1} g^{l\prime}(z_i^l) \\ &= (\sum_{j=1}^{n_{l+1}} \delta_j^{l+1} w_{ji}^{l+1}) g^{l\prime}(z_i^l)\end{aligned} \tag{A.58}$$

其中，$g^{l\prime}(z_i^l)$是$g^l(z_i^l)$的倒数。(A.58)是反向传播算法中最核心的公式，其利用第 l+1 层的δ^{l+1}来计算第 l 层的δ^l，将它表示为矩阵形式：

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot g^{l\prime}(\boldsymbol{z}^l) \tag{A.59}$$

读者可通过分析(A.59)中各矩阵和向量的维度，对从(A.58)至(A.59)的转换的正确性进行校验。

采用同样的推导过程，可得到输出层和隐藏层的偏置参数偏导结果：

$$\begin{aligned}\frac{\partial E}{\partial b_i^l} &= \frac{\partial E}{\partial z_i^l}\frac{\partial z_i^l}{\partial b_i^l} \\ &= \delta_i^l\end{aligned} \tag{A.60}$$

对应的矩阵（向量）形式为：

$$\nabla_{\boldsymbol{b}^l} E = \boldsymbol{\delta}^l \tag{A.61}$$

最后，将上述所求参数代入式(A.47)和(A.48)即可得到权重和偏置参数的更新公式：

$$\boldsymbol{W}^l = \boldsymbol{W}^l - \eta \boldsymbol{\delta}^l (a^{l-1})^T \tag{A.62}$$

$$\boldsymbol{b}^l = \boldsymbol{b}^l - \eta \boldsymbol{\delta}^l \tag{A.63}$$

$$\boldsymbol{\delta}^L = -(\boldsymbol{y} - \boldsymbol{a}^L) \odot g^{L\prime}(\boldsymbol{z}^L) \tag{A.64}$$

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot g^{l\prime}(\boldsymbol{z}^l), l = 2,3,\dots,\mathrm{L}-1 \tag{A.65}$$

其中，η是学习率。学习率越大，模型收敛越快。但需要注意，学习率过大有可能无法达到局部极小值。在实际中，通常初始设置一个较大的学习率，再设置一个衰减值，每迭代若干轮通过衰减值降低学习率，从而使得模型在初期能够快速调整参数、而后期能够对参数进行微调以达到局部极小值。

这里仅考虑了每次根据一个样本进行参数更新的情况，如果希望一次能够根据多个样本进行参数更新，则只需将(A.64)改写为：

$$\boldsymbol{\delta}^L = -\frac{1}{|S|}\sum_{s\in S}(\boldsymbol{y}^s - \boldsymbol{a}^{s,L}) \odot g^{L\prime}(\boldsymbol{z}^L) \tag{A.66}$$

其中，S 是当前参数更新所使用的样本集合。

A.5.2 代码实现

BP 网络的实现代码中用到了 SciPy 模块，以 sigmoid 函数作为隐层和输出层神经元的激活函数。在 Jupyter Notebook 中依次输入以下代码并运行。

1）导入包

代码清单 A-1 导入库

```
import numpy as np
import scipy.special
import matplotlib.pyplot as plt
from pylab import mpl
```

2）创建神经网络类 NeuralNetwork

输入以下代码创建 NeuralNetwork 类（实现中每个神经元没有设置偏置参数，读者可尝试修改下面代码添加偏置参数）。

代码清单 A-2 创建神经网络类

```
# 创建神经网络类，以便于实例化成不同的实例
class NeuralNetwork:
    # 初始函数
    def __init__(self, input_nodes,hidden_nodes,output_nodes,learning_rate):
        # 初始化输入层、隐藏层、输出层的节点个数
        self.inodes=input_nodes
        self.hnodes=hidden_nodes
        self.onodes=output_nodes
        # 初始化输入层与隐藏层之间的初始权重参数
        self.wih=np.random.normal(0.0,pow(self.hnodes,-0.5),(self.hnodes,self.inodes))
        # 初始化隐藏层与输出层之间的初始权重参数
        self.who=np.random.normal(0.0,pow(self.hnodes,-0.5),(self.onodes,self.hnodes))
        # 初始化学习率
        self.lr=learning_rate
        # 定义激活函数为 sigmoid
        self.activation_function=lambda x: scipy.special.expit(x)
    # 训练函数
    def train(self,input_list,target_list):
        # 将数据的输入和标签转化为列向量
        inputs = np.array(input_list, ndmin=2).T
        targets = np.array(target_list, ndmin=2).T
        # 前向传播过程，隐藏层输入为权重矩阵和输入矩阵做点积
        hidden_inputs = np.dot(self.wih, inputs)
        # 隐藏层接收的输入经激活函数处理得到隐藏层输出，此处未考虑偏置
        hidden_outputs = self.activation_function(hidden_inputs)
        # 同理前向传播得到最终输出层
        final_inputs = np.dot(self.who, hidden_outputs)
        final_outputs = self.activation_function(final_inputs)
        # 预测与实际相减得到偏差矩阵
        output_errors = targets - final_outputs
        # 根据(A.62)计算得到 delta
        delta = output_errors * final_outputs * (1 - final_outputs)
        # 根据(A.60)更新隐藏层到输出层的权重矩阵
        self.who += self.lr * np.dot(delta, np.transpose(hidden_outputs))
        # 根据(A.60)和(A.63)更新输入层到隐藏层的权重矩阵
        self.wih += self.lr * np.dot((np.dot(self.who.T, delta) * hidden_outputs * (1 - hidden_outputs)),
(np.transpose(inputs)))

    # predict 功能，预测新样本的种类
    def predict(self,inputs_list):
        # 转换输入矩阵为列向量
        inputs=np.array(inputs_list,ndmin=2).T
        # 前向传播得到最终输出结果
        hidden_inputs=np.dot(self.wih,inputs)
        hidden_outputs=self.activation_function(hidden_inputs)
        final_inputs=np.dot(self.who,hidden_outputs)
        final_outputs=self.activation_function(final_inputs)
        return final_outputs

    # 得分函数，在测试集上进行一次测试
    def score(self, inputs, targets):
        # 通过类方法 query 输出 test 数据集中的每一个样本的目标值和预测值进行对比。
```

```
        scorecord = []
        for i in range(len(inputs)):
            # 每个数据的目标值
            correct_label = np.argmax(targets[i])
            # 每个数据的预测值
            outputs = self.predict(inputs[i])
            label = np.argmax(outputs)
            # 预测正确将 1 加入到 scorecord 数组，错误加 0
            if (label == correct_label):
                scorecord.append(1)
            else:
                scorecord.append(0)
        # 将列表转化为 array
        scorecord_array = np.asarray(scorecord)
        # 返回精度
        return scorecord_array.sum() / scorecord_array.size
```

3）神经网络训练和测试

输入以下代码训练和测试神经网络。

代码清单 A-3 训练和测试神经网络

```
# 手写数字为 28*28 大小，所以在变成一维数据之后，需要有这么多的输入点，隐藏层神经元可以自行定义；输出层神经元为分类的总个数
input_nodes = 784
hidden_nodes = 50
output_nodes = 10
# 定义学习率
learning_rate = 0.1
# 进行 epochs 设定
epochs=50

def splitdata(datalist):
    inputs_list = []
    targets_list = []
    for record in datalist:
        # 将输入去掉','转化为向量
        all_values=record.split(',')
        # 对数据进行归一化操作转化为 0 到 1 间 float 类型的数字
        inputs=np.asfarray(all_values[1:])/255
        # 定义并初始化标签向量
        targets=np.zeros(output_nodes)
        # 将 targets 数组的第标签个分量的输出置为 1，即编码成 One-Hot 形式
        targets[int(all_values[0])]=1
        inputs_list.append(inputs)
        targets_list.append(targets)
    return inputs_list, targets_list

# 打开训练数据集
train_data_file=open('./mnist_dataset_csv/mnist_train.csv','r')
# 得到数据，一行代表一个输入
train_data_list=train_data_file.readlines()
train_data_file.close()
```

```
train_inputs,train_targets = splitdata(train_data_list)

# 打开测试数据集
test_data_file = open('./mnist_dataset_csv/mnist_test.csv', 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
test_inputs,test_targets = splitdata(test_data_list)

# 用我们的类创建一个神经网络实例
nn=NeuralNetwork(input_nodes, hidden_nodes, output_nodes, learning_rate)
# 定义分数矩阵，方便后续画图
train_scores=[]
test_scores=[]
for e in range(epochs):
    print('第%d 次迭代...'%(e+1))
    # 对每个数据进行一次训练
    for i in range(len(train_inputs)):
        # 训练网络更新权重值
        nn.train(train_inputs[i],train_targets[i])
    # 将分数加到分数数组
    train_scores.append(nn.score(train_inputs,train_targets))
    test_scores.append(nn.score(test_inputs,test_targets))
    print('训练准确率：%f'%train_scores[e])
    print('测试准确率：%f'%test_scores[e])
```

上面代码运行后，将开始进行网络训练，迭代 50 轮。输出结果如图 A-3 所示：

图 A-3 运行结果

4）结果的图形化展示

代码清单 A-4 结果展示

```
#优化 matplotlib 汉字显示乱码的问题
mpl.rcParams['font.sans-serif'] = ['FangSong']
mpl.rcParams['axes.unicode_minus'] = False

plt.figure(figsize=(10,4))
plt.xlabel('迭代轮数')   # x 轴标签
plt.ylabel('准确率')   # y 轴标签
plt.plot(range(1,51),train_scores,c='red',label='训练准确率')
plt.plot(range(1,51),test_scores,c='blue',label='测试准确率')
plt.legend(loc='best')
plt.grid(True)   # 产生网格
```

```
plt.show()   # 显示图像
```

代码中，隐藏节点数设置为 50，训练准确率和测试准确率随迭代轮数的变化如图 A-4 所示。

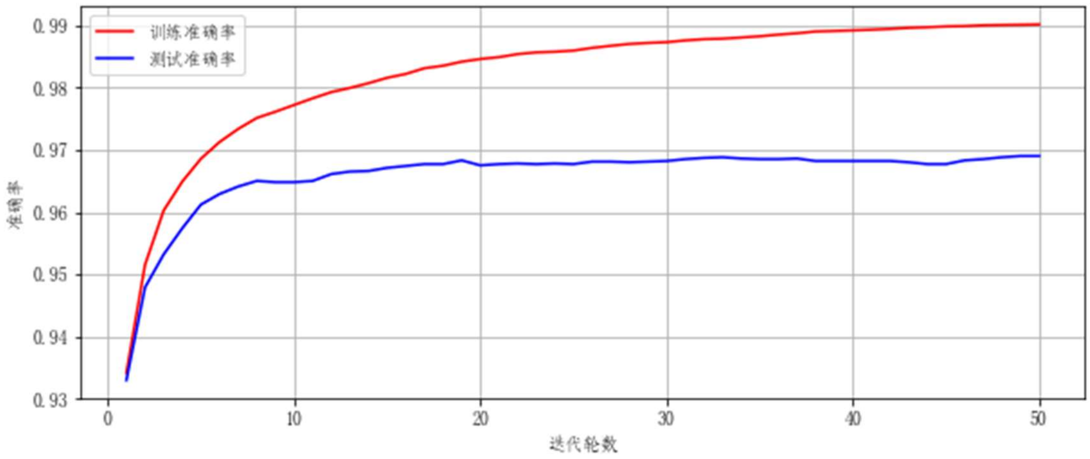

图 A-4 准确率随训练轮数的变化