实验4. 神经网络

实验目的

1. 实现神经网络模型和算法，解决手写数字识别问题
2. 实现反向传播算法

实验数据

ex4data.mat-手写数字识别的数据集：该数据是著名的手写字符识别MNIST数据集的子集。包含手写的0至9的10组数字图像共5000幅，每幅图像尺寸为20*20，每组数字包含500幅的图像。

在ex5data.mat中，包含两组数据，一是5000幅图像，在X数组中存储，尺寸为5000*400。其中，每幅20*20的图像被拉伸为长度为400的向量。另一组数据是这5000幅图像中包含的数字，分别使用1~10去表示。

实验步骤

1. 准备数据

1.1 读取数据

使用scipy.io.loadmat()去装载MNIST手写数字图像数据集。读取后，使用data['X']和data['y']读取数据X和y。

为使类别标签1~10变换为图像中的数字0~9。因此，在读取后，通过data['y']-1完成变换。

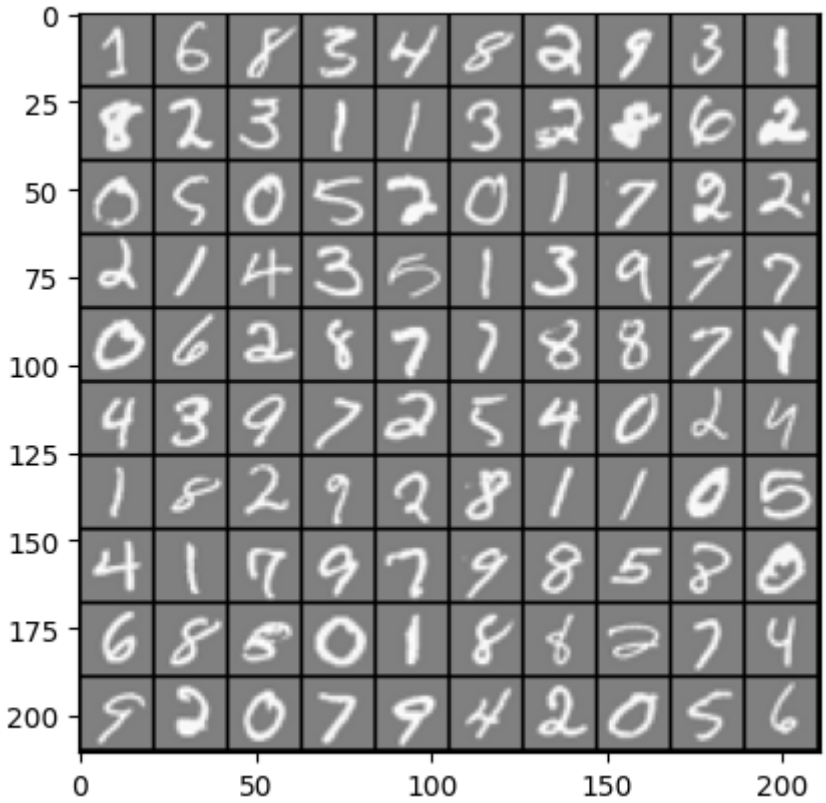
```
from scipy.io import loadmat

data = loadmat('ex4data.mat')
X = data['X']
```

1.2 可视化手写数字图像

调用如下代码，随机挑选100幅手写数字图像进行显示。图像的显示调用已经准备好的displayMNISTImage()，其参数是在使用np.random.permutation()打散5000幅图像后的前100幅。效果如下图。

```
rand_indices = np.random.permutation(np.array(range(len(X))))
sel = X[rand_indices[0:100]]
displayMNISTImage(sel)
```



1.3 数据预处理

为X添加一维x0=1。

为了适应网络的训练，在这一部分需要对标签y进行one-hot编码，编码后的结果仍存放于y。

原始的标签值是从1~10，在这里对标签进行了处理，使其范围变为0~9（注意0对应真实值1，1对应真实值2，以此类推，9对应真实值0），方便后续的one-hot编码。

对于y标签为0~9的数字进行One-Hot编码的过程如下：

1. 创建一个与原向量维度相同的零向量或矩阵，维度为10（对应0~9这10个数字）。
2. 对于原向量中的每个元素y素 y_i :

    a. 在对应的编码向量位置上将其置为1。

    b. 其他位置上将其置为0。

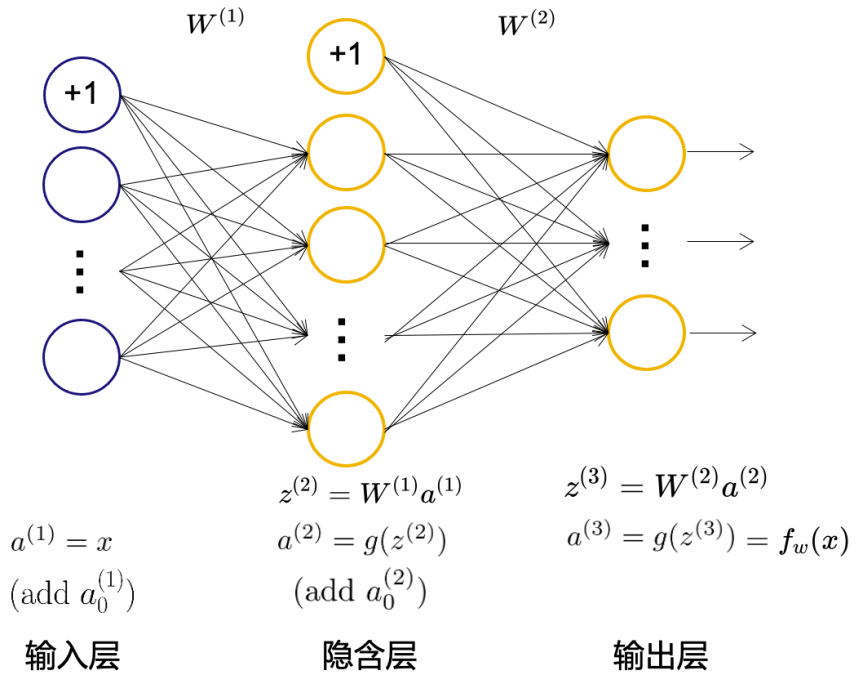
举例来说，如果y = 3，那么进行One-Hot编码后的向量就是[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]。这样，每个数字都会对应一个独立的编码向量。所以最终y会是一个10*5000的向量。

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

```
# 为X添加一维X0 = 1并转置，最后X的尺寸应为(401,5000)
# y扩展为(10,5000)的二维矩阵
m, n = data['X'].shape
X =
y =
```

2. 神经网络模型

本实验构建的神经网络包含三层，输入层、隐含层和输出层。输入层有400个单元（对应于20×20的图像像素，不包括1个偏置单元）。隐藏层有25个单元（不包括1个偏置单元），输出层有10个单元，对应于10个数字类别。其结构可见下面的示意图。



2.1 连接权重W的准备与拉伸

```
weights = loadmat('ex5weights.mat')
W1 = weights['Theta1']
W2 = weights['Theta2']

all_W = np.hstack((W1.flatten(), W2.flatten()))
```

2.2 神经网络的预测（前馈传递）

在这一部分你需要完成feedForwardProp(W1, W2, X)函数，实现前向传播，注意传入的X要求为一维数组（本函数实现输入单个样本输入时的前向计算）。

根据神经网络结构图下方的公式为神经元准备输入和输出，最后获得的a3即为整个网络的输出。

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoidGradient(z):
    return sigmoid(z) * (1 - sigmoid(z))

def feedForwProp(W1, W2, x):
    z2 =
    a2 =
    z3 =
    a3 =
    return a3
```

2.3 神经网络的代价函数（带正则项）

$$L(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log_2(f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})_k) + (1-y_k^{(i)})\log_2(1-f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})_k)\right] \tag{1}$$

$$+\frac{\lambda}{2m}\left[\sum_{u=1}^{400}\sum_{v=1}^{25}(W_{u,v}^{(1)})^2 + \sum_{u=1}^{25}\sum_{v=1}^{10}(W_{u,v}^{(2)})^2\right] \tag{2}$$

```
def nnCostReg(all_W, x, y, lmd):
    W1 = np.reshape(all_W[0 : 25 * 401], (25, 401))
    W2 = np.reshape(all_W[25 * 401 : ], (10, 26))
    '''

    code here

    '''
    return cost
```

2.4 代价函数相对模型参数的梯度（基于反向传递算法）

在不考虑正则项的前提下，各层分配的误差和梯度如下：

$$\delta^{(3)} = a^{(3)} - y$$
$$\delta^{(2)} = \boldsymbol{W}^{(2)^T}\delta^{(3)} \cdot \left(\boldsymbol{a}^{(2)}(1-\boldsymbol{a}^{(2)})\right)$$
$$\frac{\partial L}{\partial W^{(2)}} = \delta^{(3)}\frac{\partial z^{(3)}}{\partial W^{(2)}} = \delta^{(3)}a^{(2)}$$
$$\frac{\partial L}{\partial W^{(1)}} = \delta^{(2)}a^{(1)}$$

正则项的梯度如下：

$$\frac{\partial L_{reg}}{\partial w} = \frac{\lambda}{m}w$$

最后加上正则项的梯度：

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\delta^{(2)}a^{(1)}}{m} + \frac{\lambda}{m}w_1$$
$$\frac{\partial L}{\partial W^{(2)}} = \frac{\delta^{(3)}a^{(2)}}{m} + \frac{\lambda}{m}w_2$$

```
def nnGradReg(all_W, x, y, lmd):
    W1 = np.reshape(all_W[0 : 25 * 401], (25, 401))
    W2 = np.reshape(all_W[25 * 401 : ], (10, 26))

    '''

    code here

    '''

    return np.hstack((W1Grad.flatten(), W2Grad.flatten() ))
```

2.5 最小化代价函数

训练神经网络的反向传递（Back Propogation）算法用于计算连接权重的梯度，尤其是隐含层节点前的连接权重。当可以获得代价函数相对于所有模型参数的梯度后，便可使用梯度下降法、共轭梯度法等优化算法。

2.5.1 初始化神经网络的参数

上面验证时我们使用了从ex5weights.mat读取的参数。在训练神经网络时，随机初始化的参数是很重要的，这里一种合适的初始化方式的代码已经给出，运行即可。

```
def randInitWeights(l_in, l_out):
    epsilon_init = 0.12
    return np.random.rand(l_out, l_in + 1) * 2 * epsilon_init - epsilon_init
```

2.5.2 训练网络

在本实验中，继续使用scipy.optimize.minimize()的共轭梯度法来最小化神经网络的代价函数。

```
result = op.minimize(nnCostReg, x0=initNNParams, args=(X, y, lmd),
method='CG', jac=nnGradReg)
W1 = np.reshape(result.x[0 : 401 * 25], (25, 401))
W2 = np.reshape(result.x[(401 * 25):], (10, 26))
```

2.6 使用神经网络预测手写数字图片

2.6.1 预测函数

根据前馈神经网络的输出结果预测该手写数字的标签。

```
def predict(W1, W2, x):
    a3 = feedForwProp(W1, W2, x)
    return np.argmax(a3, axis = 0)
```

2.6.2 预测单个手写数字的准确性

i为手写数字集的索引号，W1, W2为已经训练好的神经网络参数值。输入单个手写数字，predOne为输出的预测标签值，yOne_label为真实的标签值。

```
def predict_one(i):
    predOne = predict(W1, W2, X[:,[i]])
    yOne_label = (data['y'][i] - 1).flatten()
    return predOne,yOne_label
predOne,yOne_label = predict_one(100)
print("The result of the model predictions is: ",predOne)
print("The true label value is: ",yOne_label)
```

在这里我们测试编号为100的数据，该数据的真实标签值为9。

Cell输出：

The result of the model predictions is: [9]

The true label value is: [9]

2.7 获得神经网络的训练准确率

你的准确率应该在97.7%左右。

```
predRslt = predict(Theta1, Theta2, X)
y_label = (data['y'] - 1).flatten()
acc_rate = (predRslt == y_label).sum() / m
print('Training Set Accuracy: {:.2f}'.format(acc_rate * 100), '%')
```

3. 可视化隐含层

```
displayMNISTImage(W1[:, 1:])
```