实验 3：神经网络

实验目的

1、实现神经网络解决手写数字识别问题
2、使用预设参数实现前向传播
3、实现反向传播算法

实验数据

1. ex3data1.txt-用于训练神经网络的数据集
2. theta1.txt & theta2.txt-用于前向计算的预设参数数据

实验步骤

1. 前向传播

手写数字自动识别技术在当今社会有着广泛的应用，如邮件信封上邮政编码的识别、银行支票上金额的识别等。本实验旨在通过实现神经网络，来识别0至9的手写数字。在该部分，你将搭建一个神经网络，并利用已给的参数进行前向传播。

1.1 读取数据

数据集ex3data1.txt中包含了5000个20×20像素的手写数字灰度图像的训练样本，txt文件中的每一行数据代表一个图像样本。每个图像都被展开成一个400维的向量，标签存储在最后一列，因此ex3data1.txt中的数据规模为5000*401。theta1.txt和theta2.txt中的数据为神经网络参数。

首先你需要做的是将文件中的数据进行读取，，具体要求如下：

1.使用loadtxt函数读取ex3data1.txt中的数据存于变量ex3data1

2.使用变量X储存ex3data1的前400列数据（图像数据）

3.使用变量y储存ex3data1的第401列数据（标签）

4.使用loadtxt函数读取theta1.txt中的数据存于变量theta1

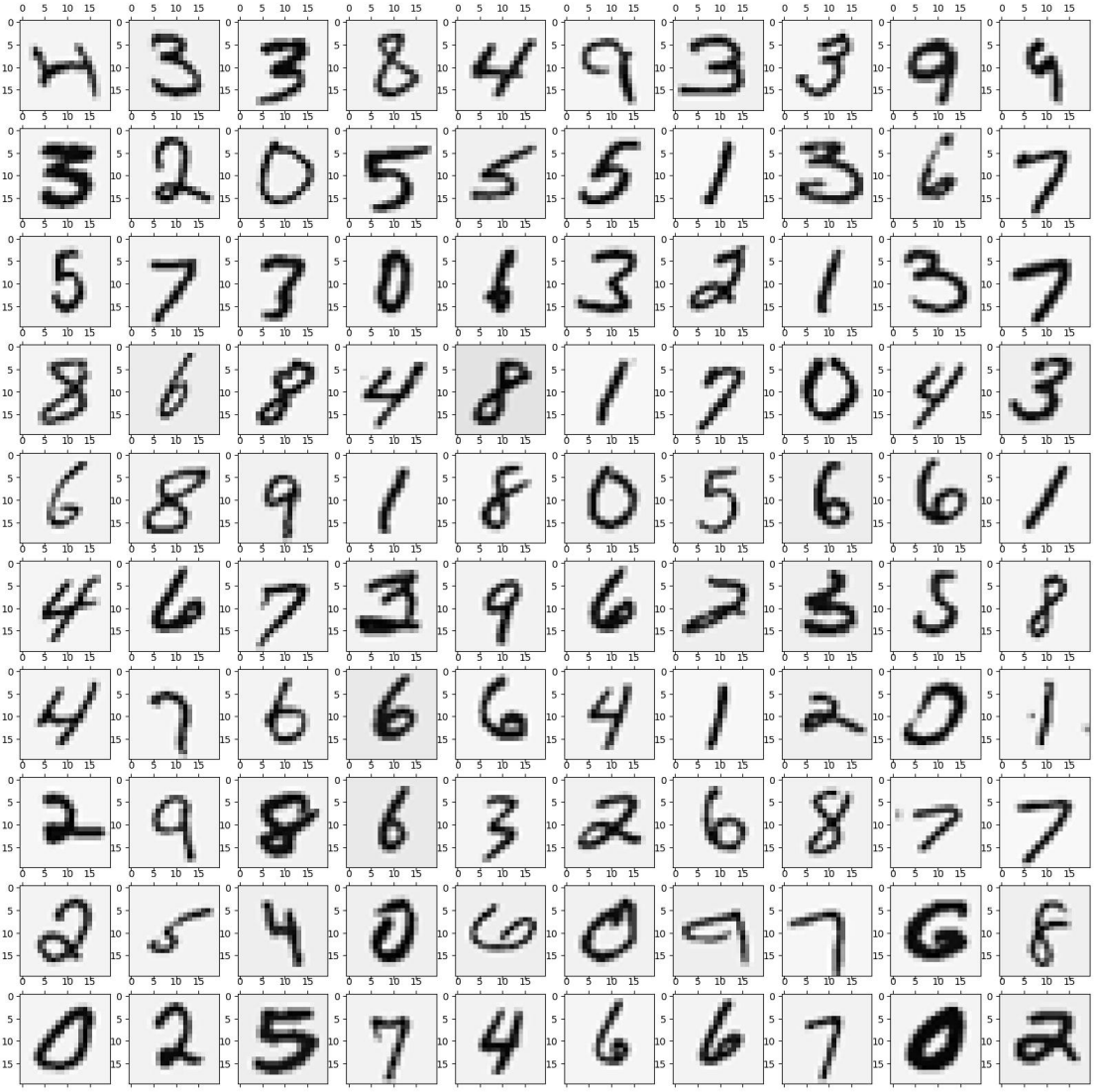
5.使用loadtxt函数读取theta2.txt中的数据存于变量theta2

cell正确输出：(5000, 400) (5000, 1) (25, 401) (10, 26)

1.2 可视化数据

在5000张图片中随机挑选100张进行可视化，这部分代码已经完成，你可以运行查看手写数字可视化的结果。

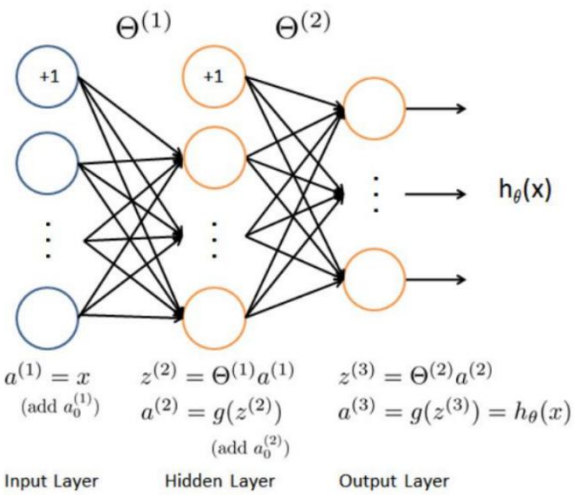
cell正确输出：



#随机挑选展示图片不同

1.3 神经网络前向传播

本实验使用的神经网络模型包含三层：一个输入层，一个隐藏层，以及一个输出层。



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。

在这一部分你需要完成feedforward_propagation(X,Theta1,Theta2)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）。

完成前向传播函数后运行准确率预测代码，你的模型应达到约97.5%的准确率。

2. 反向传播

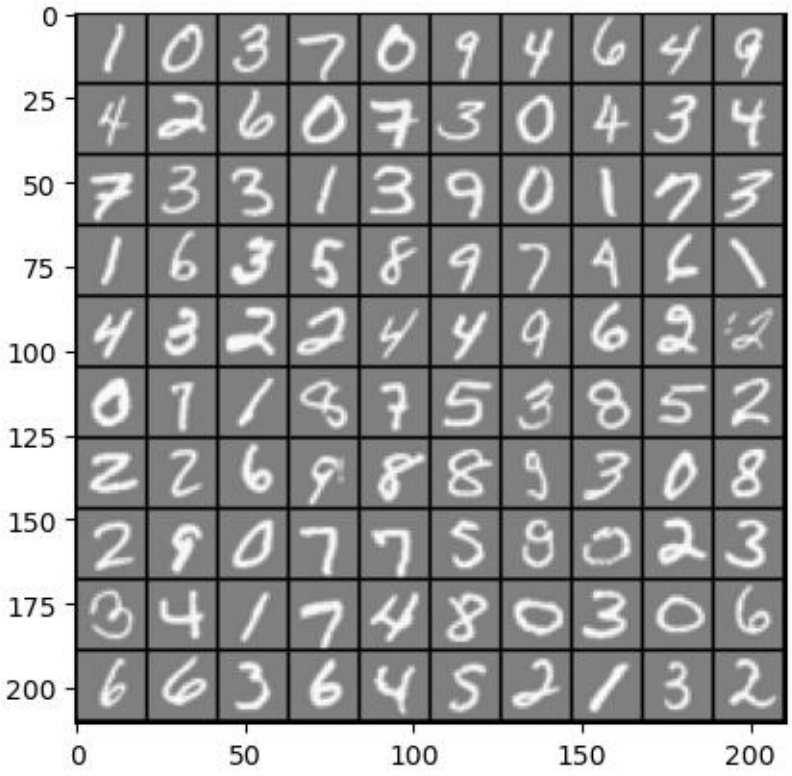
反向传播算法是一种监督学习下的参数优化算法，在本部分你将通过实现反向传播算法，改进神经网络对0至9的手写数字的识别能力。

2.1 读取并可视化数据

代码已给出，可以注意到该部分读取的数据文件格式为.mat，使用的函数是scipy库中的loadmat()。mat文件为二进制形式存储，相较于txt的纯文本，具有更高的读写速度，具备空间、结构等方面优势。

读取的5000个样本特征存于数组x中，标签存放于y，并对标签进行了处理，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。

同时该部分实现了随机现实100张图片，对手写数字的数据进行了可视化。



2.2 模型表示

反向传播部分使用的神经网络与前向传播相同，分为输入层，1个隐藏层和输出层。其中输入层有400个输入单元，隐藏层有25个神经元，输出层为10个输出单元。该部分读取了前向传播中用到的已知参数用于后续验证。

你首先需要对x进行进行处理，扩增x的维度（+1）方便偏置项运算，这个操作你应该比较熟悉了。

另外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。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_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。

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

2.3 正则化的神经网络

该部分你要完成用于训练网络的主要函数。包括前向传播函数、代价计算函数和梯度计算函数。需要使用到的sigmoid函数以及其导数函数sigmoidGradient()已经给出。

1. 前向传播函数feedForwardProp(Theta1,Theta2,x)，与前次实验的实现思路相同，但注意本次feedForwardProp()函数接收的x为二维数组，即传入5000*400特征矩阵。

2. 损失计算函数nnCostReg(all_theta,x,y,lmd)，需要注意传入的第一个参数all_theta为一个一维数组（为适应后面自动最小化函数对回调函数的参数要求）。带正则项的损失函数公式如下：

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

其中K=10，表示输出单元的数量，$y_k^{(i)}$表示第i个样本标签（one-hot后为十维向量）的第k个值，$h_\theta(x^{(i)})_k$表示传入第i个样本后输出层第k个输出单元输出值。

3. 梯度计算函数nnGradReg(all_theta,x,y,lmd)，第一个参数仍为一维数组。回想一下反向传播算法背后的原理。给定一个训练示例，我们将首先运行一个前向传播来计算整个网络的所有激活值。紧接着我们需要通过计算损失函数对参数的偏导得到梯度，并进行梯度下降等算法优化参数。

求损失对参数偏导的过程中，我们会需要用到一个中间量$\delta_i^{(l)}$，这个量的含义是计算了第l层的第i个神经元对损失分别负责多少，定义式为：

$$\delta^{(l)} = \frac{\partial J(\theta)}{\partial z^{(l)}} = \frac{\partial J(\theta)}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}$$

求得每层的$\delta^{(l)}$向量，方便我们对θ_{ij}求导。经过链式求导后隐藏层和输出层的$\delta^{(l)}$具体计算公式如下：

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

算得 $\delta^{(l)}$（for l=2,3）后即可利用如下思路进行梯度计算：

Backpropagation算法中的梯度计算

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

1. Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).
2. For $i = 1$ to m
   Set $a^{(1)} = x^{(i)}$
   Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$
   Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
   Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
   $\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ 为每个样本累计误差delta
3. $$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \begin{cases} \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)} & \text{if } j = 0 \end{cases}$$

其中，$\Delta_{ij}^{(l)}$表示第l层的参数 θ_{ij}的梯度值。注意最后返回的梯度grad要求为一维数组。

完成上面函数的定义后，即可运行下方代码块，该部分代码使用0和1两个lambda、预先读取的参数和你完成的代价函数、梯度函数，正确输出如下所示：

```
Feedforward Using Neural Network ...
Cost at parameters (loaded from ex4weights):  0.2876291651613189
(this value should be about 0.287629)
Checking Cost Function (with Regularization) ...
Cost at parameters (loaded from ex4weights):  0.38376985909092365
(this value should be about 0.383770)
Evaluating sigmoid gradient...
Sigmoid gradient evaluated at [1 -0.5 0 0.5 1]:    [0.19661193
0.23500371 0.25       0.23500371 0.19661193]
```

2.4 初始化参数

上面验证时我们使用了从weights.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。

2.5 训练网络和模型评估

在该部分我们需要使用自动最小化算法求取网络最优参数，该部分示例代码中使用到了op.fmin_cg()函数，该函数和我们之前使用的op.minimize()类似，前者可视为后者的一个特例，在处理大规模计算问题时有优势，二者参数要求类似，fmin_cg()函数的fprime参数即梯度计算函数。

你的模型准确率应约为97.86%。