

《运筹学》大作业

从非线性规划到深度学习

课题报告

林子坤 （自动化系 自 45 班 2014011541）

郑文举 （自动化系 自 45 班 2014011548）

目录

一、课题概述	4
二、程序属性	4
三、文件说明	4
四、基本任务	5
4.1 梯度下降法（任务 A1）	5
4.1.1 方法原理	5
4.1.2 程序算法	6
4.1.3 运行结果与讨论	6
4.2 基本三层神经网络（任务 A2）	7
4.2.1 方法原理	7
4.2.2 程序算法	12
4.3 BANKRUPTCY 数据集实验（任务 A3）	15
4.3.1 程序算法	15
4.3.2 运行结果与讨论	16
五、进阶任务	17
5.1 参数选择研究（任务 B1）	17
5.1.1 比较不同的 s_2 对实验的影响	17
5.1.2 比较不同的 α 对实验的影响	18
5.1.3 比较不同的 λ 对实验的影响	21
5.2 WINE QUALITY 数据集实验（任务 B2）	21
5.2.1 数据选取	22
5.2.2 参数估计	22
5.2.3 实验测试与讨论	22
5.3 MNIST 手写字数据集实验（任务 B3）	24
5.3.1 数据选取	24
5.3.2 参数估计	24
5.3.3 实验测试与讨论	25
5.4 RELU 激活函数梯度推导与实验（任务 B4）	26
5.4.1 方法原理	26

5.4.2 程序算法	30
5.4.3 实验测试与讨论	32
六、加分任务——深层网络（任务 C1）	33
6.1 全连接层梯度推导与实验	33
6.1.1 方法原理	33
6.1.2 程序算法	36
6.1.3 仅有全连接层的多层神经网络实验测试与讨论	37
6.2 卷积层梯度推导与实验	39
6.2.1 方法原理	39
6.2.2 程序算法	41
6.2.3 含有卷积层的多层神经网络实验测试与讨论	42
七、人员分工说明	43
八、遇到的问题与解决方法	44
九、总结与感受	45
参考文献	46

从非线性规划到深度学习

课题报告

林子坤 （自动化系 自 45 班 2014011541）

郑文举 （自动化系 自 45 班 2014011548）

[摘 要] “深度学习”在近年的人工智能领域掀起了潮流，其学习的有效性和高准确率使得人们对“深度学习”的方法越来越青睐。在本次任务中，我们将揭开深度学习框架的“神秘面纱”，通过对目标函数梯度的求取和运筹学中的梯度下降法，亲手构建一个层数可变、连接函数可变、隐层形式可变的完整的神经网络，并通过多个数据集的测试验证网络结构的正确性和有效性，了解神经网络在人工智能领域的效果、作用与魅力。

[关键词] 非线性规划 深度学习 梯度下降法 求取梯度 向量化编程 参数调整

一、课题概述

本次课题使用了运筹学课程中的非线性规划方法，按照任务要求逐步推导并实现基本三层神经网络、连接函数可变的神经网络、仅含全连接层的多层神经网络，最终实现包含全连接层、卷积层的连接函数可变的神经网络，并在提供的三个数据集上分别进行测试分析，通过不断提高神经网络可行性和准确率，探索参数选择方法。

二、程序属性

本次课题使用 MATLAB R2017a 进行编写，并且未使用任何现有的框架或代码包。所有代码可以在任何安装有 MATLAB 程序的计算机中运行。

三、文件说明

为了保持代码结构与助教老师所给的示例代码基本相同，测试 A1~B4 任务的可执行程序为“**ex1.m**”，其中沿用了示例代码中的分节做法。

由于 C1 任务最终的含卷积层的神经网络参数调节与输入方式较为复杂，且运行一次所需时间十分长，因此为其专门编写一个可执行程序进行测试，程序名为“**finalTestFunction.m**”，执行后按照程序要求输入参数即可。

所在文件夹	文件名（按字母顺序排列）	内容说明
src	BankData.mat	经过处理后的 A3 任务数据集
	costFunctionA2.m	A2 任务核心代码： 代价函数与梯度计算（对应 4.2.2(1)）

		中所述算法)
	costFunctionA2_com.m	A2 任务核心代码: 代价函数与梯度计算的第二种方法 (对应 4.2.2(2)中所述算法)
	costFunctionB4.m	B4 任务核心代码: 含有 ReLU 层的神经网络代价函数与梯度计算
	costFunctionC1.m	C1 任务核心代码: 仅含全连接层的多层神经网络代价函数与梯度计算
	defaultCostFunction.m	A1 任务示例代码, 对其进行了一定修改 (在 4.1.2(1)中进行说明)
	ex1.m	可执行程序 1: 测试 A1~B4 任务
	finalTestFunction.m	可执行程序 2: 测试 C1 任务最终的神经网络
	hypothesisA2.m	A2 任务核心代码: 函数值计算
	hypothesisB4.m	B4 任务核心代码: 函数值计算
	hypothesisC1.m	C1 任务核心代码: 函数值计算
	lrCostFunction.m	示例代码, 未使用到
	lrHypothesis.m	示例代码, 未使用到
	mnist.mat	B3 任务数据集
	myfminuncA1.m	A1 任务核心代码: 基本梯度下降法
	myfminuncB1.m	B1 任务使用过的代码: 自编可变学习率梯度下降算法 (在 5.1.2(1)中进行说明)
	myfminuncBoldDriver.m	B1 任务和之后测试任务使用的代码: 使用 BoldDriver 算法进行动态学习率变化的梯度下降算法 (在 5.1.2(2)中进行说明)
	relu.m	B4 任务调用函数: ReLU 函数值的计算
	relu_d.m	B4 任务调用函数: ReLU 函数导数值的计算
	sigmoid.m	A2 及之后任务调用函数: Sigmoid 函数值计算
	winequality-red.csv	B2 任务数据集
output	Bankruptcy 文件夹	A3 任务实验中所设置的参数和训练所得的 W , b 值和按列优先的顺序排成列向量保存的文件
	Winequality-red 文件夹	B2 任务实验中所设置的参数和训练所得的 W , b 值和按列优先的顺序排成列向量保存的文件
	MNIST 文件夹	B3 任务实验中所设置的参数和训练所得的 W , b 值和按列优先的顺序排成列向量保存的文件

四、基本任务

4.1 梯度下降法 (任务 A1)

4.1.1 方法原理

梯度下降法的公式为:

$$\theta := \theta - \alpha \nabla J$$

其中, J 为需要找到最优点的 n 元函数; θ 为历次迭代的梯度, 为 n 维向量; α 为学习率, ∇J 为函数 J 的梯度, 计算公式如下:

$$\nabla J = \begin{bmatrix} \frac{\partial}{\partial \theta_1} J \\ \vdots \\ \frac{\partial}{\partial \theta_n} J \end{bmatrix}$$

4.1.2 程序算法

(1) 编写默认函数, 输出函数值及其梯度

由于 `defaultCostFunction.m` 中的函数并不符合梯度下降法的原理与形式, 因此对其进行一定修改。

待求解的二元函数 $f(\theta_1, \theta_2) = x_1 \theta_1^2 + x_2 \theta_2^2 + y$ 的梯度计算如下:

$$\nabla f(\theta_1, \theta_2) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} f(\theta_1, \theta_2) \\ \frac{\partial}{\partial \theta_2} f(\theta_1, \theta_2) \end{bmatrix} = \begin{bmatrix} 2x_1 \theta_1 \\ 2x_2 \theta_2 \end{bmatrix}$$

根据示例代码, $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$, 为一 2 维列向量; $X = [x_1, x_2]$, 为一 2 维行向量。使用 θ 和 X 表示函数 $f(\theta_1, \theta_2)$ 和 $\nabla f(\theta_1, \theta_2)$:

$$f(\theta_1, \theta_2) = x_1 \theta_1^2 + x_2 \theta_2^2 + y = X \begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} + y = X \theta^2 + y$$

$$\nabla f(\theta_1, \theta_2) = \begin{bmatrix} 2x_1 \theta_1 \\ 2x_2 \theta_2 \end{bmatrix} = (2X .* \theta^T)^T$$

其中, “ \cdot^2 ”、“ $.*$ ”沿用了 MATLAB 中的定义, 代表对矩阵内每个数分别进行平方与乘法运算而非矩阵运算。

(2) 编写梯度下降法函数, 输出最优解并记录历史函数值

梯度下降法函数直接调用示例代码 `myfminuncA1.m`, 将梯度下降法的公式 $\theta := \theta - \alpha \nabla J$ 填入待改写的代码行中。

记录历史函数值的功能在示例代码中已实现, 实现方法是开辟一个维数为最大迭代次数的向量, 在每次迭代结束时记录函数值 J 。

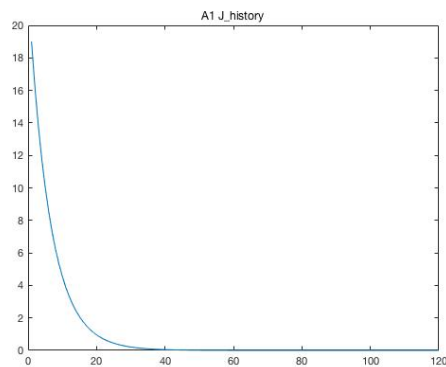
4.1.3 运行结果与讨论

很容易看出, 二元函数 $f(\theta_1, \theta_2) = x_1 \theta_1^2 + x_2 \theta_2^2 + y$ 的最优解和最优值分别为:

$$\begin{cases} \theta^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ f^* = 0 \end{cases}$$

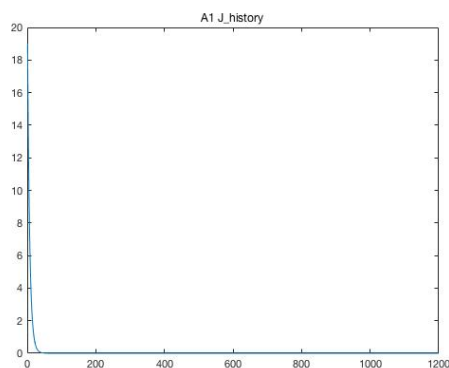
运行“A1:实现梯度下降法, 并通过二次函数测试”函数节, 获得以下输出结果与图像。

A1: optimal_theta=[0.000596,0.000090]



从结果中可以看出，图像收敛效果良好，函数值逐渐收敛至 0。最优解误差小于 10^{-3} 量级。为了减小误差，可以选择增加迭代次数。例如以下为增加最大迭代次数为 1200 次的输出结果与图像。

A1: optimal_theta=[0.000000,0.000000]

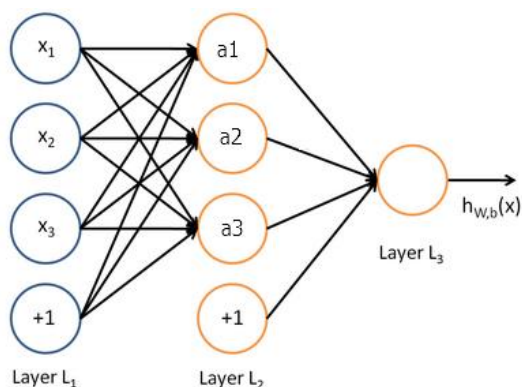


可以看出，最优解误差已经小于 10^{-6} 量级，说明增加最大迭代次数能够获得更小的误差。

4.2 基本三层神经网络（任务 A2）

4.2.1 方法原理

所谓神经网络就是将许多个单一“神经元”联结在一起，这样，一个“神经元”的输出就可以是另一个“神经元”的输入。例如，下图就是一个简单的神经网络：



其中，标上“+1”的圆圈被称为偏置节点，也就是截距项。神经网络最左边的一层叫做输入层，最右的一层叫做输出层（本例中，输出层只有一个节点）。中间所有节点组成的一层叫做隐藏层。同时可以看到，以上神经网络的例子中有 3 个输入单元（偏置单元不计在内），3 个隐藏单元及一个输出单元。

根据以上网络结构与作业要求，定义以下变量名及其格式：

变量名	程序中名称	维数	含义
$X_{m \times n}$	X	$m \times n$	输入样本矩阵，一行一个样本。 m 代表样本数， n 代表特征维数
$W^{(1)}$	W1	$n \times s_2$	Layer L1 和 Layer L2 之间的连接参数。 s_2 代表隐层维数。
$b^{(1)}$	b1	$1 \times s_2$	Layer L1 和 Layer L2 之间的偏置参数
$W^{(2)}$	W2	$s_2 \times 1$	Layer L2 和 Layer L3 之间的连接参数
$b^{(2)}$	b2	1×1	Layer L2 和 Layer L3 之间的偏置参数
$a^{(1)}$	a1	$m \times n$	输入层
$a^{(2)}$	a2	$m \times s_2$	隐层
h	h	$m \times 1$	输出层

为了实现神经网络的反向传播算法和梯度下降法，依次进行如下步骤。

(1) 计算神经网络的输出结果

为了计算神经网络的隐层和输出层的输出结果，编写函数 hypothesisA2.m，进行逐层求解。

根据作业要求给出的神经网络公式

$$h_{W,b}(x^{(i)}) = f(f(x^{(i)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)})$$

对 m 个样本使用向量化表示方式，令

$$X_{m \times n} = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

以上公式可表示为：

$$h_{W,b}(X_{m \times n}) = f(f(X_{m \times n}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)})$$

可以得出，隐层输出结果为：

$$a^{(2)} = f(X_{m \times n}W^{(1)} + b^{(1)})$$

输出层输出结果可表示为：

$$h_{W,b}(X_{m \times n}) = f(a^{(2)}W^{(2)} + b^{(2)})$$

(2) 计算目标函数的梯度

根据作业要求，为使得我们构造的 $h_{\Theta}(X_{m \times n})$ 达到最佳逼近效果，我们需要最小化以下

目标函数：

$$\begin{aligned} \min J(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}) \\ = \frac{1}{m} \sum_{i=1}^m \left[-y^{(i)} \log(h_{W,b}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{W,b}(x^{(i)})) \right] \\ + \frac{\lambda}{2m} (\|W^{(1)}\|^2 + \|W^{(2)}\|^2) \end{aligned}$$

对于这一优化问题来说， $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$ 是优化变量， $x^{(i)}, y^{(i)}$ 是由训练数据集决定的常数， λ 是事先人为确定的常数。式中 \log 表示以 e 为底的自然对数， $\|\cdot\|$ 表示求矩阵所有元素的平方和。

为了用梯度下降法求解这一优化问题，推导目标函数的梯度获得(1)(2)两式：

$$\begin{aligned} \frac{\partial J}{\partial W^{(i)}} \\ = \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(i)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(i)}} \right] \\ + \frac{\lambda}{m} W^{(i)} \quad \dots\dots\dots (1) \end{aligned}$$

$$\frac{\partial J}{\partial b^{(i)}} = \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(i)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(i)}} \right] \quad (2)$$

由于该神经网络两层之间的传递函数均为 sigmoid 函数，因此根据 sigmoid 函数的性质：

$$f'(x) = f(x)(1 - f(x))$$

求解 $h_{W,b}$ 函数关于 $W^{(i)}, b^{(i)}$ 的偏导数过程如下：

注：为了表示方便并节省书写空间，以下计算过程中的 $h_{W,b}$ 均为 $h_{W,b}(x^{(j)})$ 的略写， $x^{(j)}$

沿用(1)(2)两式中的定义，表示第 j 个输入样本， $a_j^{(2)} = f(x^{(j)} W^{(1)} + b^{(1)})$ ，实际上是程序中间步骤中的变量 $a^{(2)}$ 的第 j 行。

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial W^{(1)}} &= \frac{\partial h_{W,b}}{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial W^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial f(x^{(j)}W^{(1)} + b^{(1)})}{\partial W^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial f(x^{(j)}W^{(1)} + b^{(1)})}{\partial (x^{(j)}W^{(1)} + b^{(1)})} \cdot \frac{\partial (x^{(j)}W^{(1)} + b^{(1)})}{\partial W^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \dots \dots \dots (3)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial W^{(2)}} &= \frac{\partial h_{W,b}}{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial W^{(2)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \dots \dots \dots (4)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial b^{(1)}} &= \frac{\partial h_{W,b}}{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial f(x^{(j)}W^{(1)} + b^{(1)})}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial f(x^{(j)}W^{(1)} + b^{(1)})}{\partial (x^{(j)}W^{(1)} + b^{(1)})} \cdot \frac{\partial (x^{(j)}W^{(1)} + b^{(1)})}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \dots \dots \dots (5)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial b^{(2)}} &= \frac{\partial h_{W,b}}{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial b^{(2)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \dots \dots \dots (6)
\end{aligned}$$

将(3)式代入(1)式，求解 J 函数关于 $W^{(1)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial W^{(1)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(1)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(1)}} \right] \\
&\quad + \frac{\lambda}{m} W^{(1)}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \right] \right\} \\
&\quad + \frac{\lambda}{m} W^{(1)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \cdot h_{W,b}(x^{(j)}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \right] \right\} + \frac{\lambda}{m} W^{(1)} \\
&= \frac{1}{m} \sum_{j=1}^m \left[\left(-y^{(j)} + h_{W,b}(x^{(j)}) \right) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \cdot x^{(j)} \right] + \frac{\lambda}{m} W^{(1)}
\end{aligned}$$

将(4)式代入(1)式，求解 J 函数关于 $W^{(2)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial W^{(2)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(2)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(2)}} \right] \\
&\quad + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \right] + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \right] \right\} \\
&\quad + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot a_j^{(2)} \right] + \left[(1 - y^{(j)}) \cdot h_{W,b}(x^{(j)}) \cdot a_j^{(2)} \right] \right\} + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left[\left(-y^{(j)} + h_{W,b}(x^{(j)}) \right) \cdot a_j^{(2)} \right] + \frac{\lambda}{m} \cdot W^{(2)}
\end{aligned}$$

将(5)式代入(2)式，求解 J 函数关于 $b^{(1)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial b^{(1)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(1)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(1)}} \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot a_j^{(2)}(1 - a_j^{(2)}) \right] \right\}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot W^{(2)} \cdot a_j^{(2)} (1 - a_j^{(2)}) \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) h_{W,b}(x^{(j)}) \cdot W^{(2)} \cdot a_j^{(2)} (1 - a_j^{(2)}) \right] \right\} \\
&= \frac{1}{m} \sum_{j=1}^m \left[(-y^{(j)} + h_{W,b}(x^{(j)})) \cdot W^{(2)} \cdot a_j^{(2)} (1 - a_j^{(2)}) \right]
\end{aligned}$$

将(6)式代入(2)式，求解 J 函数关于 $b^{(2)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial b^{(2)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(2)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(2)}} \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \right] + \left[(1 - y^{(j)}) h_{W,b}(x^{(j)}) \right] \right\} \\
&= \frac{1}{m} \sum_{j=1}^m (-y^{(j)} + h_{W,b}(x^{(j)}))
\end{aligned}$$

4.2.2 程序算法

对于以上的推导结果，我们两位同学在编写程序时分别产生了两个不同的想法。其中一个想法是按照 $\frac{\partial J}{\partial W^{(1)}}$ 、 $\frac{\partial J}{\partial W^{(2)}}$ 、 $\frac{\partial J}{\partial b^{(1)}}$ 、 $\frac{\partial J}{\partial b^{(2)}}$ 的推导最终结果进行编程，这将使得代码的步骤更为清晰，且每步内的步骤也能很简洁；而另一个想法是将 $\frac{\partial h_{W,b}}{\partial W^{(1)}}$ 、 $\frac{\partial h_{W,b}}{\partial W^{(2)}}$ 、 $\frac{\partial h_{W,b}}{\partial b^{(1)}}$ 、 $\frac{\partial h_{W,b}}{\partial b^{(2)}}$ 作为中间变量，虽然表达式会较为复杂，但是这对反向传导算法的推广很有帮助。

为了能够兼顾理论推导结果和灵活性，在所附程序中包含有两种计算方法的函数，分别进行算法说明如下。

(1) 按照推导最终结果进行编程

根据参考资料[1]中反向传导算法的思路，定义变量

$$\delta_3^{(j)} = -(y^{(j)} - h_{W,b}(x^{(j)}))$$

$$\delta_2^{(j)} = \delta_3^{(j)} \cdot W^{(2)} \cdot a^{(2)} (1 - a^{(2)})$$

它们的实际意义是每层的“残差”。

再定义变量

$$\Delta W_j^{(1)} = \delta_2^{(j)} \cdot x^{(j)}$$

$$\Delta W_j^{(2)} = \delta_3^{(j)} \cdot a_j^{(2)}$$

$$\Delta b^{(1)} = \sum_{j=1}^m \delta_2^{(j)}$$

$$\Delta b^{(2)} = \sum_{j=1}^m \delta_3^{(j)}$$

则上式化为：

$$\frac{\partial J}{\partial W^{(1)}} = \frac{1}{m} \sum_{j=1}^m \Delta W_j^{(1)} + \frac{\lambda}{m} W^{(1)}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{1}{m} \sum_{j=1}^m \Delta W_j^{(2)} + \frac{\lambda}{m} W^{(2)}$$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{1}{m} \Delta b^{(1)}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{1}{m} \Delta b^{(2)}$$

为了减少循环语句的使用，将原始的数据矩阵直接用于最终的求解计算，本人将以上各样本进行向量化，令：

$$\Delta W^{(1)} = \delta_2 * X_{m \times n}$$

$$\Delta W^{(2)} = \delta_3 * a^{(2)}$$

在此处用*代表向量乘积运算符，也称作阿达马乘积。

得到的表达式如下所示。

$$\frac{\partial J}{\partial W^{(1)}} = \frac{1}{m} \Delta W^{(1)} + \frac{\lambda}{m} W^{(1)}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{1}{m} \Delta W^{(2)} + \frac{\lambda}{m} W^{(2)}$$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{1}{m} \Delta b^{(1)}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{1}{m} \Delta b^{(2)}$$

这便是程序 `costFunctionA2.m` 的求解过程。关键实现代码如下：

```

%%%%%%%%%%以下为求取theta下的目标函数梯度grad的步骤%%%%%%%%%%
%1、根据反向传播算法，求取输出层的delta参数(注:f'(a3)=a3.*(1.-a3))
delta3=-(y-h); %求取delta3，维数m*1
delta2=(delta3*W2').*a2.*(1.-a2); %求取delta2，维数m*s2

%2、根据反向传播算法，计算需要的△W1、△W2、△w1、△w2
DeltaW1=a1'*delta2; %求取DeltaW1，维数n*s2
DeltaW2=a2'*delta3; %求取DeltaW2，维数s2*1
Deltab1=sum(delta2,1); %求取Deltab1
Deltab2=sum(delta3,1); %求取Deltab2

%3、计算梯度grad
grad_W1=1/m*DeltaW1+lambda/m*W1;
grad_W2=1/m*DeltaW2+lambda/m*W2;
grad_b1=1/m*Deltab1;
grad_b2=1/m*Deltab2;

```

(2) 将 $\frac{\partial h_{W,b}}{\partial W^{(1)}}$ 、 $\frac{\partial h_{W,b}}{\partial W^{(2)}}$ 、 $\frac{\partial h_{W,b}}{\partial b^{(1)}}$ 、 $\frac{\partial h_{W,b}}{\partial b^{(2)}}$ 作为程序的中间变量

根据(1)(2)式：

$$\frac{\partial J}{\partial W^{(i)}} = \frac{1}{m} \sum_{j=1}^m \left[\left(-\frac{y^{(j)}}{h_{W,b}(x^{(j)})} + \frac{1-y^{(j)}}{1-h_{W,b}(x^{(j)})} \right) \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(i)}} \right] + \frac{\lambda}{m} W^{(i)}$$

$$\frac{\partial J}{\partial b^{(i)}} = \frac{1}{m} \sum_{j=1}^m \left[\left(-\frac{y^{(j)}}{h_{W,b}(x^{(j)})} + \frac{1-y^{(j)}}{1-h_{W,b}(x^{(j)})} \right) \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(i)}} \right]$$

定义以下变量

$$p(x^{(j)}) = -\frac{y^{(j)}}{h_{W,b}(x^{(j)})} + \frac{1-y^{(j)}}{1-h_{W,b}(x^{(j)})}$$

$$q_j^{W^{(i)}} = p(x^{(j)}) \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(i)}}$$

$$q_j^{b^{(i)}} = p(x^{(j)}) \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(i)}}$$

则上式化为：

$$\frac{\partial J}{\partial W^{(i)}} = \frac{1}{m} q_j^{W^{(i)}} + \frac{\lambda}{m} W^{(i)}$$

$$\frac{\partial J}{\partial b^{(i)}} = \frac{1}{m} q_j^{b^{(i)}}$$

为了减少循环语句的使用，将原始的数据矩阵直接用于最终的求解计算，本人将以上各样本进行向量化，令：

$$p(X_{m \times n}) = -\frac{y}{h_{W,b}(X_{m \times n})} + \frac{1-y}{1-h_{W,b}(X_{m \times n})}$$

$$q_{W^{(i)}} = p(X_{m \times n}) \frac{\partial h_{W,b}(X_{m \times n})}{\partial W^{(i)}}$$

$$q_{b^{(i)}} = p(X_{m \times n}) \frac{\partial h_{W,b}(X_{m \times n})}{\partial b^{(i)}}$$

得到的表达式如下所示。

$$\frac{\partial J}{\partial W^{(1)}} = \frac{1}{m} q_{W^{(1)}} + \frac{\lambda}{m} W^{(1)}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{1}{m} q_{W^{(2)}} + \frac{\lambda}{m} W^{(2)}$$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{1}{m} q_{b^{(1)}}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{1}{m} q_{b^{(2)}}$$

这便是程序 `costFunctionA2_com.m` 的求解过程。关键实现代码如下：

```
%%%%%%%%%%%%%以下为求取theta下的目标函数梯度grad的步骤%%%%%%%%%%%%%
%1、求取目标梯度的公共项p
p=-y./h+ (1.-y)./(1.-h);

%2、求取梯度每个元素的附加项
qW1=a1'*((p.*h.*(1.-h))*W2'.*(a2.*(1.-a2)));
qW2=a2'*(p.*h.*(1.-h));
qb1=sum((p.*h.*(1.-h))*W2'.*(a2.*(1.-a2)));
qb2=sum(p.*h.*(1.-h));

%3、计算梯度grad
grad_W1=1/m*qW1+lambda/m*W1;
grad_W2=1/m*qW2+lambda/m*W2;
grad_b1=1/m*qb1;
grad_b2=1/m*qb2;
```

4.3 Bankruptcy 数据集实验（任务 A3）

4.3.1 程序算法

(1) 数据处理

读取数据后我们发现，Bankruptcy 数据集的前半部分的结果是 NB，后半部分的结果是 B。因此在挑选训练集和测试集时，不能像 demo 代码中取前 80% 做训练数据、后 20% 做测试数据，而应该随机挑选。因此我们使用了 `randperm` 函数，能够生成长度为样本个数的向量，并随机取出其中的 80% 置为 1，另外 20% 置为 0。这样，我们便可以将样本数据与向量进行一一对照，去除 80% 的训练集和 20% 的测试集。

为了进行数值计算，我们按照作业要求中的建议，令 $P=1$ 、 $A=0$ 、 $N=-1$ ，以及 $B=1$ 、 $NB=0$ 。

(2) 参数设定

对于神经网络来说，如果将所有的参数都初始化为零，会导致在进行每一步训练的过程中，连接到每一个神经元的连接权重相同，导致了神经网络在训练的过程中，只能学习到一种特征，即：

$$a_1^{(2)} = a_2^{(2)} = \dots a_{s_2}^{(2)}$$

那么其效果势必将受到影响。而如果每一个神经元与前一层的连接权重不一样的话，神经网络就能够学习到不同的特征。

为了使得对称失效，对网络参数的初始值进行随机初始化。在最初的尝试中，我们尝试使用 `normrnd` 函数生成在 0 附近的一系列随机初始值，即：

```
initial_theta = normrnd(0,0.0001,(n+1)*s2+s2+1,1);
```

在之后查阅一定的资料后，我们又发现一种做法，随机数按照以下经验公式进行生成：

$$r = \frac{\sqrt{6}}{\sqrt{n + s_2 + 1}}$$

$$W_{n \times s_2}^{(1)} = 2r \cdot \text{rand}(n, s_2) - r$$

$$W_{s_2 \times 1}^{(2)} = 2r \cdot \text{rand}(s_2, 1) - r$$

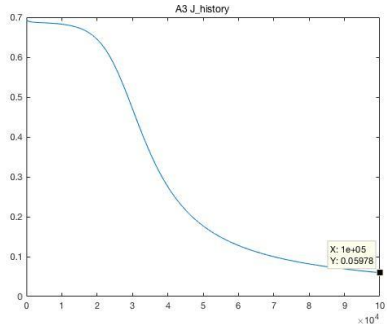
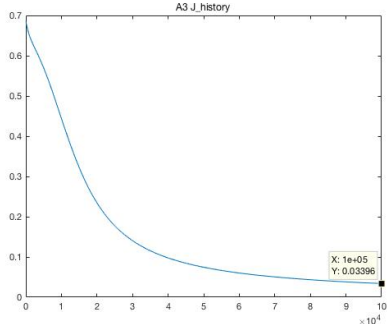
$$b^{(1)} = 0$$

$$b^{(2)} = 0$$

在接下来的运行中，我们将分别对这两种参数初始化的方式进行尝试，并分析运行结果

4.3.2 运行结果与讨论

设置隐层结点个数 $s_2 = 5$ ，最大迭代次数为 100000 次， $\alpha = 0.001$ ， $\lambda = 0.001$ ，使用两种参数初始化方式进行尝试，运行结果如下所示：

项目	实验	normrnd 函数法	经验公式法
第 1 次的 J_history 图像（第 2 次、第 3 次实验图像类似，略）			
最后一次迭代后的	第 1 次	0.05978	0.03396
	第 2 次	0.06029	0.02929

J 值	第 3 次	0.06102	0.03491
训练错误率	第 1 次	0.5%	0.5%
	第 2 次	0.5%	0.0%
	第 3 次	0.5%	0.0%
测试错误率	第 1 次	0.0%	0.0%
	第 2 次	0.0%	0.0%
	第 3 次	0.0%	0.0%

由此可见，使用经验公式能够减少一些 J 值下降较小的初始迭代次数，在一开始便进入较快的梯度下降过程。

综上所述：在 Bankruptcy 数据集下，我们获得的最好结果是：训练错误率和测试错误率均为 0.0%。

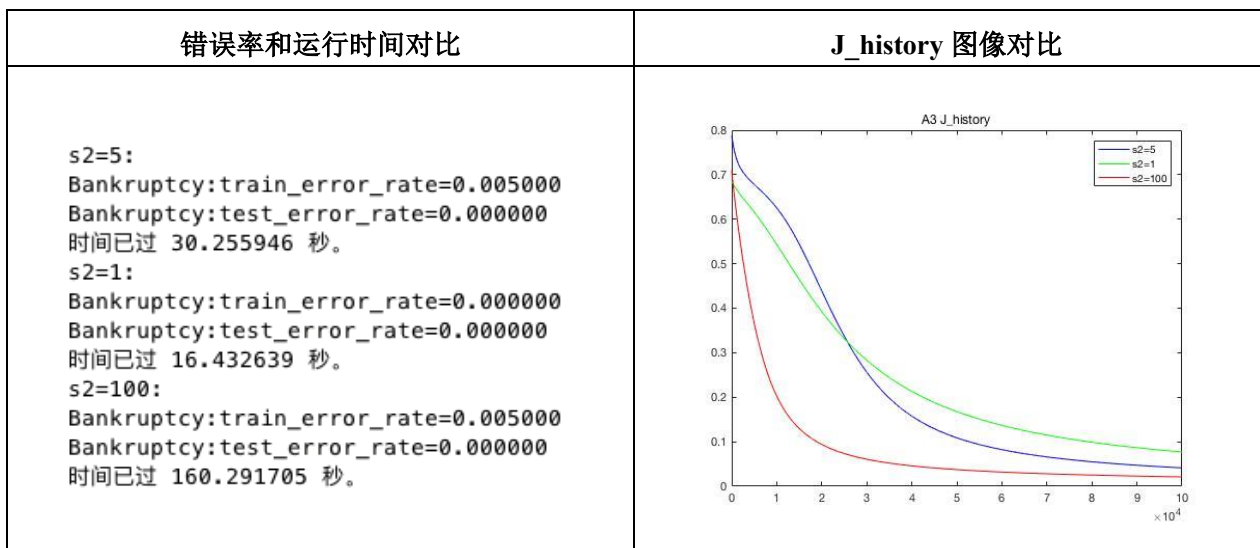
五、进阶任务

5.1 参数选择研究（任务 B1）

5.1.1 比较不同的 s_2 对实验的影响

(1) 实验比较与讨论

最大迭代次数为 100000 次， $\alpha = 0.001$ ， $\lambda = 0.001$ 。取 $s_2 = 1$ 和 $s_2 = 100$ 作为参照进行训练，获得结果如下所示。



由此可见，随着隐藏节点数 s_2 的增加，迭代过程中的梯度下降速度明显加快。在最大迭代次数有限的情况下，增加隐藏节点数 s_2 能够有效改善迭代效果，获得相对更好的结果。但是从实际训练过程中也可以发现，随着隐藏节点数 s_2 的增加，训练时间也会大大延长。

(2) 理论分析与算法实现

为了既能在可接受的时间内完成训练，又能够获得最好的训练效果（最小的错误率），我们需要选择一个较为合适的值。查阅了一些资料后，我发现了以下求取隐藏节点数的经验公式。

$$s_2 = p - 1 \quad (p \text{ 为输入层节点数})$$

$$s_2 = \log_2 p \quad (p \text{ 为输入层节点数，计算后四舍五入})$$

$$s_2 = \sqrt{n + m} + \alpha \quad (n \text{ 为输入层节点数，} m \text{ 为输出层节点数，} \alpha = 1 \sim 10)$$

$$s_2 = \sqrt{0.43mn + 2.54m + 0.77n + 0.35 + 0.12n^2} + 0.51 \quad (n, m \text{ 意义同上})$$

同时还必须保证 $s_2 < N - 1$ ，其中 N 为样本数。否则，网络模型的系统误差与训练样本的特性无关而趋于零，即建立的网络模型没有泛化能力，也没有任何实用价值。

分别带入数据，可以得到 4 个 s_2 的值分别为：

$$s_2 = 5, \quad s_2 = 3, \quad s_2 = 2 \sim 12, \quad s_2 = 4$$

由以上经验数据可得，取 s_2 为 4 或 5 较为合理。

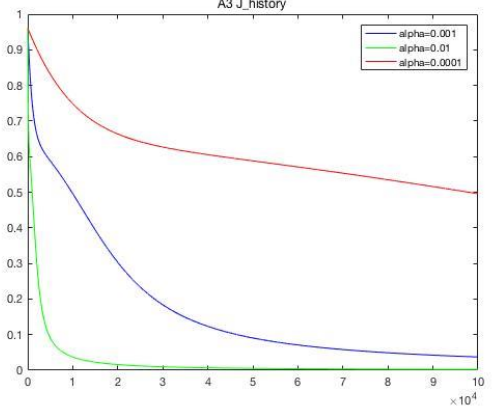
(3) 结论

在合理范围内，随着 s_2 的增加，迭代过程中的梯度下降速度明显加快，迭代效果变好，但训练时间也会随之增加。

5.1.2 比较不同的 α 对实验的影响

(1) 实验比较与讨论

最大迭代次数为 100000 次， $s_2 = 5$ ， $\lambda = 0.001$ 。取 $\alpha = 0.01$ 和 $\alpha = 0.0001$ 作为参照进行训练，获得结果如下所示。

错误率和运行时间对比	J_history 图像对比
<pre>alpha=0.001: Bankruptcy:train_error_rate=0.005000 Bankruptcy:test_error_rate=0.000000 时间已过 19.634956 秒。 alpha=0.01: Bankruptcy:train_error_rate=0.000000 Bankruptcy:test_error_rate=0.020000 时间已过 17.806676 秒。 alpha=0.0001: Bankruptcy:train_error_rate=0.035000 Bankruptcy:test_error_rate=0.000000 时间已过 17.440442 秒。</pre>	

由此可见，随着学习率 α 的增加，迭代过程中的梯度下降速度明显加快。例如图中所示的 $\alpha = 0.01$ 曲线基本上在第 30000 次迭代后的 J 值已经十分接近于 0 了，而 $\alpha = 0.0001$ 曲线在第 100000 次迭代后的 J 值依然在 0.5 以上，且训练错误率达到了 3.5%。这说明较大的 α 值有助于在较小的迭代次数内将 J 值收敛到一定范围内；但是在进一步的研究中我们也发现，过高的学习率也会导致振荡现象的产生。

(2) 理论分析与算法实现

运用梯度下降算法进行优化时，权重的更新规则 $\theta := \theta - \alpha \nabla J$ 中，在梯度项前会乘以一个系数，这个系数就叫学习速率 α 。

正如上述实验中可以看到，如果学习率 α 较大，从高点“下山”的速度会加快，但是在最优解或局部最优解附近，过大的学习率会导致函数直接冲过最优解（“步子太大，迈过山谷”），而在其附近进行来回振荡。如果调小学习率 α ，能够解决上面所说的振荡问题，但是收敛到一定范围所要求的迭代次数和运行时间都将会大幅度上升。

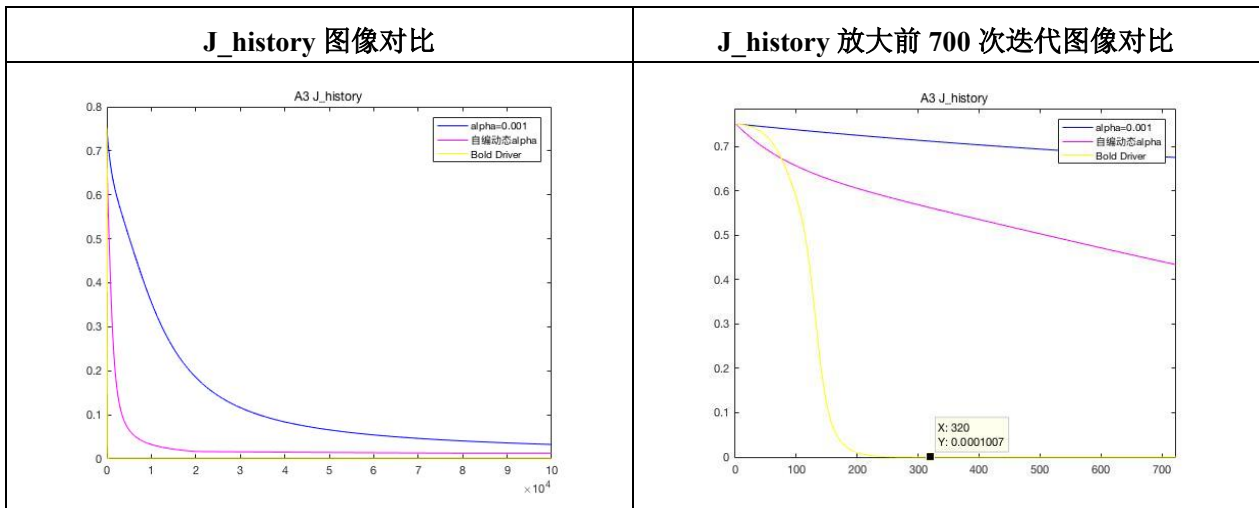
为了既能在可接受的时间内完成训练，又能够获得最好的训练效果（最小的错误率），我们需要选择一个较为合适的值。

除了确定一个固定的学习率外，可变学习率也是一种被广泛采用的方法。我们最初产生的想法是：在前 20% 的迭代中采用较高的学习率，使得函数能够尽快“下山”；在后 20% 的迭代中采用较低的学习率，使得函数在接近“谷底”时仔细地寻找最优解。

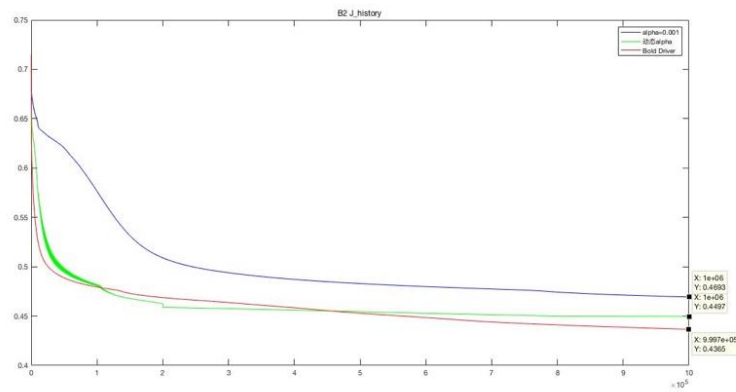
除此之外，我们还查阅到另外一种改变学习率的方法——“Bold Driver”。在这种方法下，我们在每次迭代的最后，使用估计的模型参数检查误差函数的值。如果相对于上一次迭代，错误率减少了，就可以以 5% 的幅度增大学习率；如果相对于上一次迭代，错误率增大了（意味着跳过了最优值），那么减少学习率到之前的 50%。这种方法对初始 α 值的准确性要求变得更低了。因为即使初始 α 值比最好的 α 值小了 10 倍，在 $\log_{1.05} 10 \approx 47$ 次内也能完成调整；如果初始 α 值比最好的 α 值大了 10 倍，在 $\log_{0.5} \frac{1}{10} \approx 3$ 次内便能完成调整。使用这种方法我们便无需在参数调整时对 α 值进行多次调整。此方法的关键实现代码如下：

```
if i>=2 && J<=J_history(i-1)
    alpha=1.05*alpha;
end
if i>=2 && J>J_history(i-1)
    alpha=0.5*alpha;
end
```

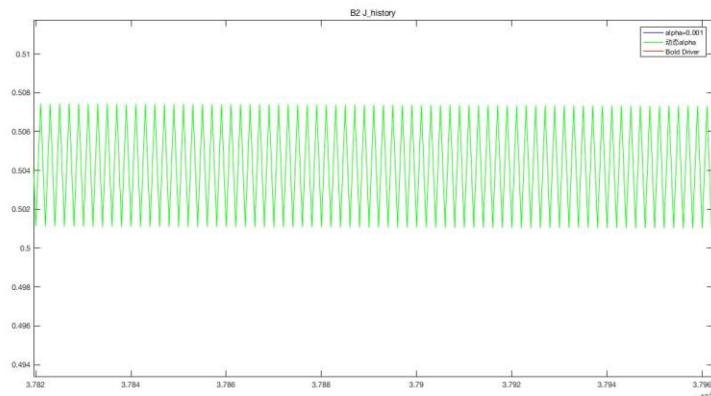
为了对以上方法有更加直观的认识，我们对这三种方法分别进行三次的训练比较，获得结果如下所示。



可以看出，由于 Bankruptcy 数据集特征良好，“下山”过程几乎畅通无阻，使用“Bold Driver”方法在 320 次迭代之后便将 J 值下降到 10^{-4} 量级范围内。但是对于特征并没有这么好的数据集会有什么结果呢？这将在我们的 B2 任务“Wine Quality 数据集实验”报告中进行阐述，运行结果如下图。



从上面的图中我们可以看到，在 100 万次迭代后，“Bold Driver”方法既能保证最终的 J 值获得最多的下降，又能避免振荡现象的产生。我们可以观察到，如果不使用“Bold Driver”方法，且设定了一个较大的 α 值，会出现较为明显的振荡现象（绿线中的较粗部分），放大绿线中第 37820 至第 37960 次迭代的图像如下图，可以看出此时在 $\alpha = 0.01$ 的学习率下出现的多次往复振荡。



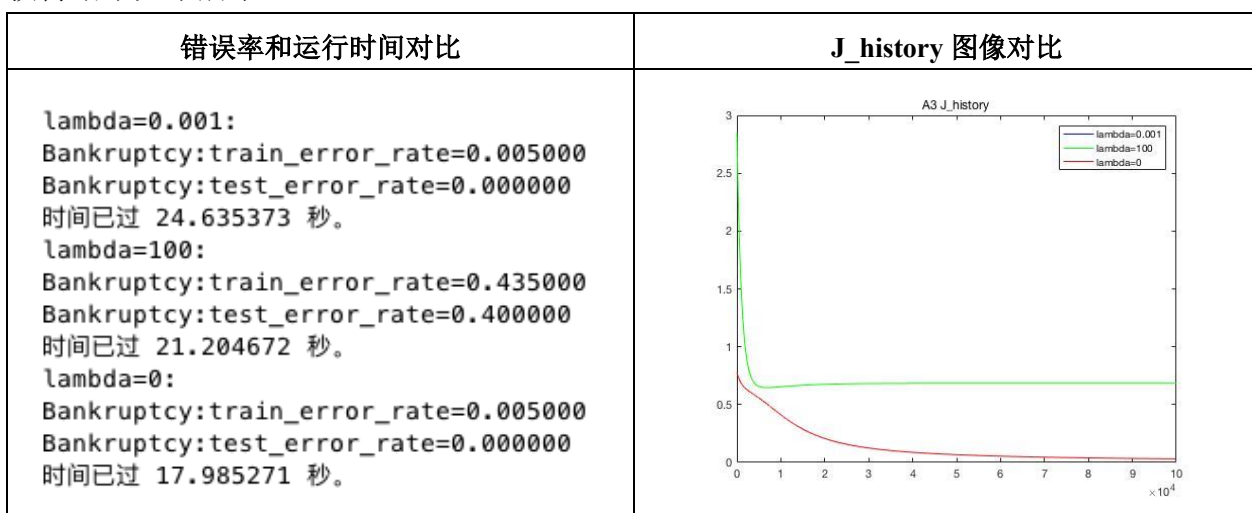
(3) 结论

随着 α 的增加，迭代过程中的梯度下降速度明显加快，所需要的迭代次数下降，但可能会导致振荡现象。为了调和这个矛盾，我们可以选择合适的 α 值，或是根据 J 值的增减对 α 值进行动态变化。

5.1.3 比较不同的 λ 对实验的影响

(1) 实验比较与讨论

最大迭代次数为 100000 次， $s_2 = 5$ ， $\alpha = 0.001$ 。取 $\lambda = 100$ 和 $\lambda = 0$ 作为参照进行训练，获得结果如下所示。



可以看出，在一定范围内 λ 值对实验结果的影响较小。但是如果 λ 高到了一定值，其对 J 值的影响变得很大，那么最终求得的 J 值最优值并不能带来较低的正确率。

(2) 理论分析与算法实现

正则化的作用是：过拟合的时候，拟合函数需要顾忌每一个点，最终形成的拟合函数波动很大。在某些很小的区间里，函数值的变化很剧烈。这就意味着函数在某些小区间里的导数值（绝对值）非常大，由于自变量值可大可小，所以只有系数足够大，才能保证导数值很大。而正则化是通过约束参数的范数使拟合函数的系数不要太大，所以可以在一定程度上减少过拟合情况。

按照网上的经验与教程，正则项系数 λ 的初始值应该设置为多少，好像也没有一个比较好的准则。建议一开始将正则项系数 λ 设置为 0，在随后的训练中测试出一个较好的数值。

(3) 结论

在一定范围内 λ 值对实验结果的影响较小。但是如果 λ 高到了一定值，其对 J 值的影响变得很大，训练错误率和测试错误率均会开始上升。

5.2 Wine Quality 数据集实验（任务 B2）

注：本次实验选用了 Wine Quality 数据集中的 red 数据集。

5.2.1 数据选取

在本次实验中，选取 quality 为“4、5”的作为一组，quality 为“6、7”的作为一组。

5.2.2 参数估计

正如前文中所提到的那样，Wine Quality 数据集的特征并不如 Bankruptcy 数据集那么好，在前期的初测试中发现其 J 值收敛速度很慢，且训练错误率都经常高于 25%。为了对其进行较好的训练，我们需要事先对其初始化参数进行估计。

(1) 隐藏节点数 s_2

根据 5.1.1 节中给出的几个经验公式，我们对隐藏节点数进行计算，得到几个 s_2 值分别为：

$$s_2 = 10, \quad s_2 = 3, \quad s_2 = 3 \sim 13, \quad s_2 = 6$$

由以上经验数据可得，取 s_2 为 6 至 10 较为合理。

(2) 正则项系数 λ

由于该数据集特征较不明显，始终处于“欠拟合”而非“过拟合”状态，因此将其正则项系数 λ 先设置为 0 进行测试。

5.2.3 实验测试与讨论

由于训练集和测试集均为随机选取的，因此每组实验都进行 2 次，尽可能减少较极端实验数据对结果的影响。

(1) 调整隐藏节点数 s_2

设置最大迭代次数为 1000000 次， $\alpha = 0.001$ ， $\lambda = 0$ ，调整隐藏节点数 s_2 ，获得结果如下表所示。

隐藏节点数 s_2	3	10	12	15	20
训练错误率	0.247850	0.232995	0.218139	0.231431	0.192338
	0.227522	0.229085	0.240031	0.192338	0.196247
测试错误率	0.256250	0.231250	0.271875	0.250000	0.290625
	0.231250	0.265625	0.234375	0.278125	0.265625

可以看出，当隐藏节点数 s_2 在 10 个左右时，神经网络的训练效果最好；更多的节点虽然能够继续降低训练错误率，但是测试错误率发生上升或者不能更好地改善神经网络的性能。经分析可能是产生了“过拟合”现象所致。在之后的实验中，为了避免“过拟合”现象的出现，并节约训练时间，将隐藏节点数均设置为 10。

(2) 调整正则项系数 λ

设置最大迭代次数为 1000000 次， $s_2 = 10$ ， $\alpha = 0.001$ 。调整 λ 值，获得结果如下表所示。

正则项系数 λ	0	0.001	0.01	0.1
训练错误率	0.232995	0.244722	0.220485	0.214230
	0.229085	0.229867	0.225958	0.215794
测试错误率	0.231250	0.278125	0.284375	0.293750
	0.265625	0.237500	0.240625	0.284375

可以看出，较小的 λ 值能够保证训练和测试的错误率都能够在较小的水平上下浮动。而当 λ 值达到 1 甚至超过 1 时，训练错误率和测试错误率均会开始上升。由于在 $\lambda = 0$ 的条件下获得的平均结果最好，因此在今后的实验中，均令 $\lambda = 0$ 。

(3) 调整最大迭代次数

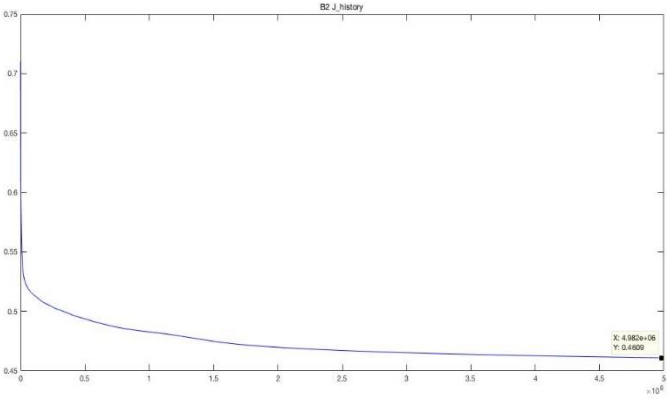
设置 $s_2 = 10$ ， $\alpha = 0.001$ ， $\lambda = 0$ 。调整最大迭代次数，获得结果如下表所示。

最大迭代次数	100000	500000	1000000	5000000	10000000
训练错误率	0.240813	0.254105	0.232995	0.233776	0.229085
	0.256450	0.238468	0.229085	0.220485	0.220485
测试错误率	0.278125	0.221875	0.221875	0.221875	0.221875
	0.240625	0.268750	0.265625	0.253125	0.259375

可以看出，将最大迭代次数设置为 5000000 次获得的平均效果最好。如果迭代次数再增加，训练错误率的降低说明神经网络模型已经可以基本拟合输入样本，但是这并不代表测试效果会变得更好，相反地，测试效果可能还会变差或者没有改变。为了最优化测试效果并节省训练时间，最终确定训练的最大迭代次数为 5000000 次。

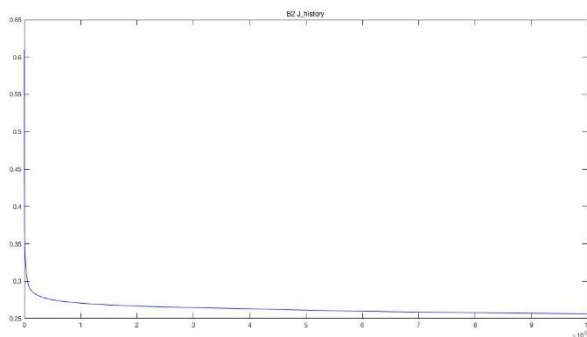
(4) 最终训练结果

设置最大迭代次数为 5000000 次， $s_2 = 10$ ， $\alpha = 0.001$ ， $\lambda = 0$ ，经过 1 小时 31 分钟的训练后，获得结果如下所示。

训练错误率	0.229867
测试错误率	0.262500
迭代过程与结果	J_history 图像  <pre> 已进行1000000次迭代,J=0.482673 时间已过 1139.922517 秒。 已进行2000000次迭代,J=0.469757 时间已过 949.632664 秒。 已进行3000000次迭代,J=0.465189 时间已过 1042.804815 秒。 已进行4000000次迭代,J=0.462654 时间已过 1075.905982 秒。 已进行5000000次迭代,J=0.460905 时间已过 1198.713273 秒。 Winequality-red:train_error_rate=0.229867 Winequality-red:test_error_rate=0.262500 </pre>

经过多次尝试（包括参数调整与重复训练），样本的测试错误率还是维持在 20%以上。经过以上的分析，我们认为该样本的特征可能不如前一个实验中的样本特征那么明显，也可能需要使用其他结构的神经网络或是其他方式才能够较好地进行分类。

值得一提的是，我们发现如果我们将类别“7”单独分为一组，“4”、“5”、“6”单独分为一组，收敛的速度变得更快，最终的准确率将会更高。例如，我们设置最大迭代次数为 1000000 次， $s_2 = 10$ ， $\alpha = 0.001$ ， $\lambda = 0$ ，经过 22 分 14 秒的训练后，获得结果如下所示。

训练错误率	0.111024
测试错误率	0.121875
迭代过程与结果	J_history 图像
<p>已进行1000000次迭代, J=0.256474 时间已过 1183.415149 秒。 Winequality-red:train_error_rate=0.111024 Winequality-red:test_error_rate=0.121875</p>	

经过分析，我们认为出现以上现象的原因是——类别“7”仅占有所有样本的约七分之一，因此在训练时，分类器将会以更大倾向将测试样本分为“4、5、6”所在分类，从而提高了正确率。

5.3 MNIST 手写字数据集实验（任务 B3）

5.3.1 数据选取

在本次实验中，选取数字“4”、“6”、“8”、“9”、“0”作为一组，剩余作为另外一组。分组依据是该数字是否存在类似“洞”的密闭空间。

5.3.2 参数估计

(1) 隐藏节点数 s_2

根据 5.1.1 节中给出的几个经验公式，我们对隐藏节点数进行计算，得到几个 s_2 值分别为：

$$s_2 = 399, \quad s_2 = 9, \quad s_2 = 20 \sim 30, \quad s_2 = 140$$

4 个经验公式所给出的隐藏节点数差距甚远，因此为了更好的实验结果，我们还需要在实验测试中进行不断尝试摸索。

(2) 正则项系数 λ

如前文所述，正则项系数 λ 的初始值应该设置为多少并没有一个比较好的准则。因此将

正则项系数 λ 设置为 0，在随后的训练中测试出一个较好的数值。

(3) 停止迭代阈值

和上个数据集不同，MNIST 数据集的 J 值下降很快，测试过程中的 J 值很可能因下降过快，小于程序可接受范围而引起程序错误。因此我们并不让程序完全跑完全部的最大迭代次数，而是设置了停止迭代阈值，当 J 值小于该阈值时自动停止程序。这也能减少一些无用的迭代次数。

5.3.3 实验测试与讨论

由于训练集和测试集均为随机选取的，因此每次实验都进行 2 次取平均结果，尽可能减少较极端实验数据对结果的影响。

(1) 调整隐藏节点数 s_2

设置最大迭代次数为 100000 次， $\alpha = 0.001$ ， $\lambda = 0$ ，停止迭代阈值为 0.000005。调整 λ 值，获得结果如下表所示。

隐藏节点数 s_2	10	30	100	150	200	399
训练错误率	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
测试错误率	3.65%	3.35%	3.45%	2.40%	2.70%	2.90%

可以看出，当隐藏节点数 s_2 在 150 左右时，神经网络的训练效果最好。在之后的实验中，将隐藏节点数均设置为 150。

(2) 调整正则项系数 λ

设置最大迭代次数为 100000 次， $s_2 = 150$ ， $\alpha = 0.001$ ，停止迭代阈值为 0.000005。调整 λ 值，获得结果如下表所示。

正则项系数 λ	0	0.001	0.005	0.01	0.1	1	10
训练错误率	0.00%	0.00%	0.00%	0.00%	0.33%	4.55%	49.73%
测试错误率	2.50%	2.40%	3.00%	2.30%	2.90%	6.00%	51.10%

可以看出，较小的 λ 值能够保证训练和测试的错误率都能够在较小的水平上下浮动。而当 λ 值达到 1 甚至超过 1 时，训练错误率和测试错误率均会开始上升。在今后的实验中，均令 $\lambda = 0.001$ 。

(3) 调整停止迭代阈值

设置最大迭代次数为 100000 次， $s_2 = 150$ ， $\alpha = 0.001$ ， $\lambda = 0.001$ 。调整停止迭代阈值，获得结果如下表所示。

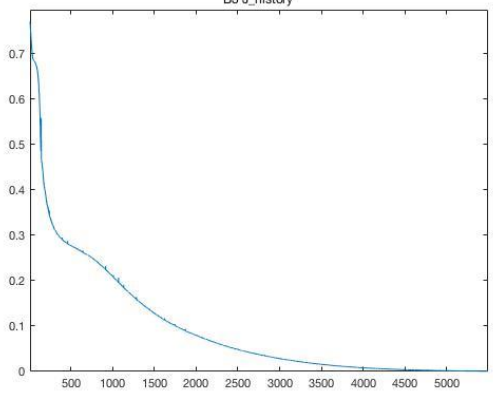
停止迭代阈值	0.000001	0.000005	0.00005	0.0005	0.005	0.05
训练错误率	0.00%	0.00%	0.00%	0.00%	0.025%	1.30%
测试错误率	2.90%	2.40%	2.90%	3.05%	3.80%	3.80%

可以看出，如果让 J 值小于一个较小的阈值就停止程序，训练出来的模型基本能够完

全拟合输入样本，但所需要的迭代次数将会增加；如果把这个阈值稍微调大，出现的训练错误率说明训练出来的模型并不能完全拟合输入样本。最终，我们将停止迭代阈值设在了 0.000005。

(4) 最终训练结果

我们设置最大迭代次数为 100000 次， $s_2 = 150$ ， $\alpha = 0.001$ ， $\lambda = 0.001$ ，停止迭代阈值为 0.000005，经过 9 分 23 秒的训练后，获得结果如下所示。

训练错误率	0.00%
测试错误率	2.40%
迭代过程与结果	J_history 图像
<p>已进行1000次迭代, J=0.209690 时间已过 97.652726 秒。</p> <p>已进行2000次迭代, J=0.079581 时间已过 93.384525 秒。</p> <p>已进行3000次迭代, J=0.028459 时间已过 90.337431 秒。</p> <p>已进行4000次迭代, J=0.008106 时间已过 93.275293 秒。</p> <p>已进行5000次迭代, J=0.001674 时间已过 102.568512 秒。</p> <p>在第5846次迭代中, J=0.000005, 小于阈值, 自动终止迭代 时间已过 76.264258 秒。</p> <p>MNIST:train_error_rate=0.000000 MNIST:test_error_rate=0.024000 时间已过 0.215279 秒。</p>	

训练错误率为 0，说明训练过程已经十分完全，神经网络结构能够基本完全拟合输入数据；测试错误率为 2.4%，也是一个较低的数值，说明训练效果良好。

5.4 ReLU 激活函数梯度推导与实验（任务 B4）

5.4.1 方法原理

近年来，ReLU 变的越来越受欢迎。它的数学表达式如下：

$$f(x) = \max(0, x)$$

很显然，当 ReLU 激活函数的输入信号小于 0 时，输出都是 0；当输入信号大于 0 时，输出与输入相同。

在不少先前的实验中，人们发现使用 ReLU 函数的收敛速度会远远快于 Sigmoid 函数。这是因为 ReLU 函数类似于线性函数，且是“不饱和”的，相比于 Sigmoid 函数或 tanh 函数，ReLU 激活函数只需要一个阈值就可以得到激活值，而不需要进行一系列复杂的运算。但是 ReLU 函数也存在缺点，那就是训练的时候很“脆弱”（举个例子：一个非常大的梯度流过 ReLU 神经元，更新过参数之后，这个神经元再也不会对任何数据有激活现象了。如果这个情况发生了，那么这个神经元的梯度就永远都会是 0），因此在实际运行中不能使用过大的学习率。

该部分的方法原理与 4.2.1 节所述的基本三层神经网络基本相同。为了实现神经网络的反向传播算法和梯度下降法，依次进行如下步骤。

(1) 计算神经网络的输出结果

根据作业要求给出的神经网络公式

$$h_{W,b}(x^{(i)}) = f_{sigmoid}(f_{relu}(x^{(i)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)})$$

对 m 个样本使用向量化表示方式，令

$$X_{m \times n} = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

以上公式可表示为：

$$h_{W,b}(X) = f_{sigmoid}(f_{relu}(X_{m \times n}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)})$$

可以得出，隐层输出结果为：

$$a^{(2)} = f_{relu}(X_{m \times n}W^{(1)} + b^{(1)})$$

输出层输出结果可表示为：

$$h_{W,b}(X_{m \times n}) = f_{sigmoid}(a^{(2)}W^{(2)} + b^{(2)})$$

(2) 计算目标函数的梯度

输入层到隐藏层激活函数变为 ReLU 函数，其定义与导函数如下：

$$\begin{cases} f_{relu}(x) = \max(0, x) \\ f'_{relu}(x) = I_{(x>0)} \end{cases}$$

在这里不考虑原点不可导的情况。

由于神经网络的隐层和输出层之间的传递函数为 sigmoid 函数，因此根据 sigmoid 函数的性质：

$$f'(x) = f(x)(1 - f(x))$$

求解 $h_{W,b}$ 函数关于 $W^{(l)}$ 、 $b^{(l)}$ 的偏导数过程如下：

注：为了表示方便并节省书写空间，以下计算过程中的 $h_{W,b}$ 均为 $h_{W,b}(x^{(j)})$ 的略写， $x^{(j)}$

沿用(1)(2)两式中的定义，表示第 j 个输入样本， $a_j^{(2)} = f(x^{(j)}W^{(1)} + b^{(1)})$ ，实际上是程序中间步骤中的变量 $a^{(2)}$ 的第 j 行。

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial W^{(1)}} &= \frac{\partial h_{W,b}}{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial W^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ \frac{\partial f_{relu}(x^{(j)}W^{(1)} + b^{(1)})}{\partial W^{(1)}} \cdot W^{(2)} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ \frac{\partial f_{relu}(x^{(j)}W^{(1)} + b^{(1)})}{\partial (x^{(j)}W^{(1)} + b^{(1)})} \cdot \frac{\partial (x^{(j)}W^{(1)} + b^{(1)})}{\partial W^{(1)}} \cdot W^{(2)} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)} > 0)} \cdot x^{(j)} \cdot W^{(2)} \dots \dots \dots (7)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial W^{(2)}} &= \frac{\partial h_{W,b}}{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial W^{(2)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \dots \dots \dots (8)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial b^{(1)}} &= \frac{\partial h_{W,b}}{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot \left\{ W^{(2)} \cdot \frac{\partial f_{relu}(x^{(j)}W^{(1)} + b^{(1)})}{\partial (x^{(j)}W^{(1)} + b^{(1)})} \cdot \frac{\partial (x^{(j)}W^{(1)} + b^{(1)})}{\partial b^{(1)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)} + b^{(1)} > 0)} \dots \dots \dots (9)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial h_{W,b}}{\partial b^{(2)}} &= \frac{\partial h_{W,b}}{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]} \cdot \left\{ \frac{\partial [f_{relu}(x^{(j)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}]}{\partial b^{(2)}} \right\} \\
&= h_{W,b}(1 - h_{W,b}) \dots \dots \dots (10)
\end{aligned}$$

将(7)式代入(1)式，求解J函数关于 $W^{(1)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial W^{(1)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(1)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(1)}} \right] \\
&\quad + \frac{\lambda}{m} W^{(1)}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \cdot x^{(j)} \cdot W^{(2)} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \cdot x^{(j)} \cdot W^{(2)} \right] \right\} \\
&\quad + \frac{\lambda}{m} W^{(1)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \cdot x^{(j)} \cdot W^{(2)} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \cdot h_{W,b}(x^{(j)}) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \cdot x^{(j)} \cdot W^{(2)} \right] \right\} + \frac{\lambda}{m} W^{(1)} \\
&= \frac{1}{m} \sum_{j=1}^m \left[\left(-y^{(j)} + h_{W,b}(x^{(j)}) \right) \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \cdot x^{(j)} \cdot W^{(2)} \right] + \frac{\lambda}{m} W^{(1)}
\end{aligned}$$

将(8)式代入(1)式，求解 J 函数关于 $W^{(2)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial W^{(2)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(2)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial W^{(2)}} \right] \\
&\quad + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \right] + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}} \cdot h_{W,b}(1 - h_{W,b}) \cdot a_j^{(2)} \right] \right\} \\
&\quad + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot a_j^{(2)} \right] + \left[(1 - y^{(j)}) \cdot h_{W,b}(x^{(j)}) \cdot a_j^{(2)} \right] \right\} + \frac{\lambda}{m} W^{(2)} \\
&= \frac{1}{m} \sum_{j=1}^m \left[\left(-y^{(j)} + h_{W,b}(x^{(j)}) \right) \cdot a_j^{(2)} \right] + \frac{\lambda}{m} \cdot W^{(2)}
\end{aligned}$$

将(9)式代入(2)式，求解 J 函数关于 $b^{(1)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial b^{(1)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(1)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(1)}} \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)} + b^{(1)}) > 0} \right] \right\}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)}+b^{(1)})>0} \right] \right. \\
&\quad \left. + \left[(1 - y^{(j)})h_{W,b}(x^{(j)}) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)}+b^{(1)})>0} \right] \right\} \\
&= \frac{1}{m} \sum_{j=1}^m \left[(-y^{(j)} + h_{W,b}(x^{(j)})) \cdot W^{(2)} \cdot I_{(x^{(j)}W^{(1)}+b^{(1)})>0} \right]
\end{aligned}$$

将(10)式代入(2)式，求解 J 函数关于 $b^{(2)}$ 的偏导数过程如下：

$$\begin{aligned}
\frac{\partial J}{\partial b^{(2)}} &= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(2)}} + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot \frac{\partial h_{W,b}(x^{(j)})}{\partial b^{(2)}} \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left[-y^{(j)} \cdot \frac{1}{h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) + (1 - y^{(j)}) \frac{1}{1 - h_{W,b}(x^{(j)})} \cdot h_{W,b}(1 - h_{W,b}) \right] \\
&= \frac{1}{m} \sum_{j=1}^m \left\{ \left[-y^{(j)} \cdot (1 - h_{W,b}(x^{(j)})) \right] + \left[(1 - y^{(j)})h_{W,b}(x^{(j)}) \right] \right\} \\
&= \frac{1}{m} \sum_{j=1}^m (-y^{(j)} + h_{W,b}(x^{(j)}))
\end{aligned}$$

5.4.2 程序算法

在进行 ReLU 激活函数的代价函数编写时，我们发现其与 Sigmoid 函数的代价函数之间仅有一项导数的差距，因此编程步骤与 4.2.2 小节所述十分类似。

根据参考资料[1]中反向传导算法的思路，定义变量

$$\delta_3^{(j)} = -(y^{(j)} - h_{W,b}(x^{(j)}))$$

$$\delta_2^{(j)} = \delta_3^{(j)} \cdot W^{(2)} \cdot I_{(a^{(2)})>0}$$

它们的实际意义是每层的“残差”。

再定义变量

$$\Delta W_j^{(1)} = \delta_2^{(j)} \cdot x^{(j)}$$

$$\Delta W_j^{(2)} = \delta_3^{(j)} \cdot a_j^{(2)}$$

$$\Delta b^{(1)} = \sum_{j=1}^m \delta_2^{(j)}$$

$$\Delta b^{(2)} = \sum_{j=1}^m \delta_3^{(j)}$$

则上式化为：

$$\frac{\partial J}{\partial W^{(1)}} = \frac{1}{m} \sum_{j=1}^m \Delta W_j^{(1)} + \frac{\lambda}{m} W^{(1)}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{1}{m} \sum_{j=1}^m \Delta W_j^{(2)} + \frac{\lambda}{m} W^{(2)}$$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{1}{m} \Delta b^{(1)}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{1}{m} \Delta b^{(2)}$$

为了减少循环语句的使用，将原始的数据矩阵直接用于最终的求解计算，本人将以上各样本进行向量化，令：

$$\Delta W^{(1)} = \delta_2 * X_{m \times n}$$

$$\Delta W^{(2)} = \delta_3 * a^{(2)}$$

在此处用*代表向量乘积运算符，也称作阿达马乘积。

得到的表达式如下所示。

$$\frac{\partial J}{\partial W^{(1)}} = \frac{1}{m} \Delta W^{(1)} + \frac{\lambda}{m} W^{(1)}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{1}{m} \Delta W^{(2)} + \frac{\lambda}{m} W^{(2)}$$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{1}{m} \Delta b^{(1)}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{1}{m} \Delta b^{(2)}$$

这便是程序 `costFunctionB4.m` 的求解过程。求取梯度的关键实现代码与 4.2.2 小节中任务 A2 完全相同，在此不再赘述。在求取梯度的过程中，需要求取 ReLU 函数的数值及其导数，关键实现代码如下：

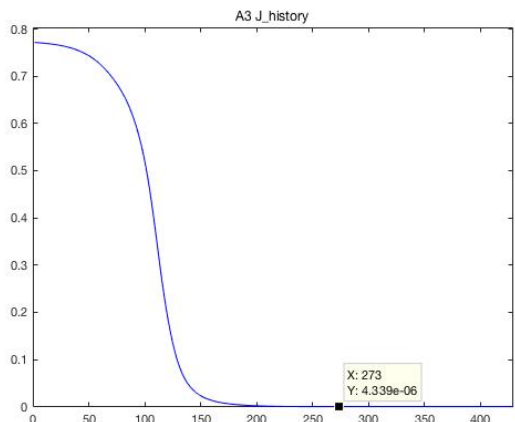
求取函数值	<pre>function [y] = relu(X) %UNTITLED3 此处显示有关此函数的摘要 % 此处显示详细说明 y = max(0,X); end</pre>	<pre>function [y] = relu_d(X) %UNTITLED 此处显示有关此函数的摘要 % 此处显示详细说明 % X 为列向量 y=zeros(size(X)); n1 = size(X,1); n2 = size(X,2); for i=1:n1 for j=1:n2 if(X(i,j)>0) y(i,j)=1; end end end end</pre>
-------	--	--

5.4.3 实验测试与讨论

为了验证 ReLU 函数的正确编写，并比较该结构神经网络与最初的神经网络结构的性能差异，我们分别使用三种数据集及其测试出的参数对该结构神经网络分别进行两次实验测试。获得结果如后文所述。

(1) 使用 Bankruptcy 数据集进行测试

设置最大迭代次数为 100000 次， $s_2 = 5$ ， $\alpha = 0.001$ ， $\lambda = 0.001$ ，经过不到 1 秒的训练后，获得结果如下所示。

训练错误率	第一次：0.000000	第二次：0.000000
测试错误率	第一次：0.000000	第二次：0.000000
效果最好一次迭代过程	效果最好一次 J_history 图像	
<p>在第273次迭代中，J=0.000004，小于阈值，自动终止迭代 时间已过 0.094133 秒。 Bankruptcy:train_error_rate=0.000000 Bankruptcy:test_error_rate=0.000000</p>		

(2) 使用 Wine Quality 数据集进行测试

设置最大迭代次数为 5000000 次， $s_2 = 10$ ， $\alpha = 0.001$ ， $\lambda = 0$ ，经过 1 小时 30 分钟的训练后，获得结果如下所示。

训练错误率	第一次：0.247068	第二次：0.247068
测试错误率	第一次：0.268750	第二次：0.262500

效果最好一次迭代过程	效果最好一次 J_history 图像
<p>已进行1000000次迭代,J=0.519298 时间已过 1049.871354 秒。 已进行2000000次迭代,J=0.519298 时间已过 1031.995402 秒。 已进行3000000次迭代,J=0.519298 时间已过 1052.980369 秒。</p> <p>已进行4000000次迭代,J=0.519298 时间已过 1151.746338 秒。 已进行5000000次迭代,J=0.519298 时间已过 1067.663318 秒。 Winequality-red:train_error_rate=0.247068 Winequality-red:test_error_rate=0.262500</p>	

(3) 使用 MNIST 手写字数据集进行测试

设置最大迭代次数为 100000 次， $s_2 = 150$ ， $\alpha = 0.001$ ， $\lambda = 0.001$ ，经过 5 分 33 秒的训练后，获得结果如下所示。

训练错误率	第一次：0.000000	第二次：0.000000
测试错误率	第一次：0.036000	第二次：0.028000
效果最好一次迭代过程	效果最好一次 J_history 图像	
<p>在第2893次迭代中，J=NaN，小于阈值，自动终止迭代 时间已过 332.759905 秒。 MNIST:train_error_rate=0.000000 MNIST:test_error_rate=0.028000 时间已过 0.170888 秒。</p>		

从以上结果可以看出，将输入层与隐层之间的传递函数改为 ReLU 函数对正确率的影响看起来并不十分明显，但总体看来测试错误率有一些下降。

六、加分任务——深层网络（任务 C1）

6.1 全连接层梯度推导与实验

6.1.1 方法原理

以上讨论的都是三层网络，如果网络结构多于三层，那么 $h_{W,b}(x^{(l)})$ 就变为以下形式：

$$h_{W,b}(x^{(l)}) = f_{L-1}(f_2(f_1(x^{(l)}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)})W^{(L-1)} + b^{(L-1)})$$

根据以上网络结构与作业要求，重新定义以下变量名及其格式：

变量名	程序中名称	维数	含义
$X_{m \times n}$	X	$m \times n$	输入样本矩阵。 m 代表样本数， n 代表特征维数
L	/	/	神经网络层数
c_1	c1	1	隐藏层层数 ($c_1 = L - 2$)
ns	ns	$c_1 \times 1$	隐含层节点数信息矩阵，第一个元素代表第一个隐藏层的节点个数，以此类推
$W^{(j)}$	$W\{j\}$	$ns_{(j-1)} \times ns_{(j)}$	第 $(j-1)$ 层和第 j 层之间的连接参数
$b^{(j)}$	$B\{j\}$	$1 \times ns_{(j)}$	第 $(j-1)$ 层和第 j 层之间的偏置参数
f_{j-1}	/	/	第 $(j-1)$ 层和第 j 层之间的激活函数
δ^j	$\Delta\{j\}$	$m \times ns_{(j)}$	第 j 层的残差
$a^{(1)}$	$a\{1\}$	$m \times n$	输入层
$a^{(j)}$	$a\{j\}$	$m \times ns_{(j)}$	第 j 层的输出
h	h	$m \times 1$	输出层： $h = a^{(c_1+1)}$

注：为了定义方便，人为规定输入层节点为： $ns(0) = n$ ；输出层节点为： $ns(c_1 + 1) = 1$ 。

(1) 计算神经网络的输出结果

对 m 个样本使用向量化表示方式，令

$$X_{m \times n} = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

以上公式可表示为：

$$h_{W,b}(X_{m \times n}) = f_{L-1}(\dots f_2(f_1(X_{m \times n}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}) \dots W^{(L-1)} + b^{(L-1)})$$

可以得出，隐层第 j 层输出结果为：

$$a^{(j)} = f_{j-1}(\dots f_2(f_1(X_{m \times n}W^{(1)} + b^{(1)})W^{(2)} + b^{(2)}) \dots W^{(j-1)} + b^{(j-1)})$$

输出层输出结果可表示为：

$$h_{W,b}(X_{m \times n}) = f(a^{(L-1)}W^{(L-1)} + b^{(L-1)})$$

(2) 计算目标函数的梯度

根据本次作业要求，由于本次实验为二分类实验，故输出层节点只有一个。采用逻辑回归中的误差函数（代价函数），其在多个隐藏层情况下的定义为：

$$J(\vec{W}, \vec{b}) = \frac{1}{m} \sum_{i=1}^m \left[-y^{(i)} \log(h_{W,b}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{W,b}(x^{(i)})) \right] \\ + \frac{\lambda}{2m} \sum_{k=1}^{L-1} (\|W^{(k)}\|^2)$$

为使得我们构造的 $h_{\theta}(X_{m \times n})$ 达到最佳逼近效果，我们需要最小化以上目标函数。

和三层神经网络的结构相似，我们可以得到多层神经网络的前向传播公式：

$$a^{(1)} = X_{m \times n} \\ z^{(j)} = a^{(j-1)} \cdot W^{(j-1)} + b^{(j-1)} \\ a^{(j)} = f_j(z^{(j)})$$

为了方便推导，记：

$$\text{cost}_i(\vec{W}, \vec{b}) = -y \log(h_{W,b}(X_{m \times n})) - (1 - y) \log(1 - h_{W,b}(X_{m \times n})) \\ \theta = (\vec{W}, \vec{b})$$

根据参考资料[1]中反向传导算法的思路，定义残差变量。

输出层残差：

$$\delta^{(L)} = \frac{\partial \text{cost}_i}{\partial z^{(c_1+1)}} = \frac{\partial}{\partial z} \left(-y \log(f_{c_1+1}(z)) - (1 - y) \log(1 - f_{c_1+1}(z)) \right) \\ = \begin{cases} -y + h, & f_{c_1+1} = \text{Sigmoid} \\ -\frac{y}{h} \cdot I_{(h>0)} - \frac{1-y}{1-h} \cdot I_{(1-h>0)}, & f_{c_1+1} = \text{ReLU} \end{cases}$$

下面通过公式推导反向传播算法：

对于第 k 层第 i 行第 j 列的待训练参数 $\theta^{(k)}_{ij}$ ：

由复合函数的链式求导法则，可以得到：

$$\frac{\partial J}{\partial \theta^{(k)}_{ij}} = \frac{\partial J}{\partial a^{(k+1)}_i} \times \frac{\partial a^{(k+1)}_i}{\partial z^{(k+1)}_i} \times \frac{\partial z^{(k+1)}_i}{\partial \theta^{(k)}_{ij}} =: \textcircled{1} \times \textcircled{2} \times \textcircled{3}$$

分别计算 $\frac{\partial J}{\partial a^{(k+1)}_i}$ 、 $\frac{\partial a^{(k+1)}_i}{\partial z^{(k+1)}_i}$ 、 $\frac{\partial z^{(k+1)}_i}{\partial \theta^{(k)}_{ij}}$ 式。

$$\frac{\partial J}{\partial a^{(k+1)}_i} = \frac{\partial J}{\partial z^{(k+2)}_1} \times \frac{\partial z^{(k+2)}_1}{\partial a^{(k+1)}_i} + \frac{\partial J}{\partial z^{(k+2)}_2} \times \frac{\partial z^{(k+2)}_2}{\partial a^{(k+1)}_i} + \dots + \frac{\partial J}{\partial z^{(k+2)}_{ns(k+2)}} \times \frac{\partial z^{(k+2)}_{ns(k+2)}}{\partial a^{(k+1)}_i} \\ = \sum_{m=1}^{ns(k+2)} \frac{\partial J}{\partial z^{(k+2)}_m} \times \frac{\partial z^{(k+2)}_m}{\partial a^{(k+1)}_i} = \sum_{m=1}^{ns(k+2)} \delta^{(k+2)}_m \times \frac{\partial z^{(k+2)}_m}{\partial a^{(k+1)}_i}$$

由前向传播公式可知：

$$z^{(k+2)} = a^{(k+1)} \cdot W^{(k+1)} + b^{(k+1)}$$

$$\frac{\partial z^{(k+2)}_m}{\partial a^{(k+1)}_i} = \theta^{(k+1)}_{mi}$$

因此，

$$\frac{\partial J}{\partial a^{(k+1)}_i} = \sum_{m=1}^{ns(k+2)} \delta^{(k+2)}_m \times \theta^{(k+1)}_{mi}$$

$$\frac{\partial a^{(k+1)}_i}{\partial z^{(k+1)}_i} = \frac{\partial}{\partial z^{(k+1)}_i} f(z^{(k+1)}_i) = \begin{cases} z^{(k+1)}_i \times (1 - z^{(k+1)}_i), & f_{k+1} = Sigmoid \\ I_{(z^{(k+1)}_i > 0)}, & f_{k+1} = ReLU \end{cases}$$

最后来看 $\frac{\partial z^{(k+1)}_i}{\partial \theta^{(k)}_{ij}}$,

$$\frac{\partial z^{(k+1)}_i}{\partial \theta^{(k)}_{ij}} = \frac{\partial}{\partial \theta^{(k)}_{ij}} (\theta^{(k)}_{i0} \times a^{(k)}_0 + \theta^{(k)}_{i1} \times a^{(k)}_1 + \dots + \theta^{(k)}_{i,ns(k)} \times a^{(k)}_{ns(k)}) = a^{(k)}_j$$

因此，综合三个偏导数，得到：

$$\begin{aligned} \frac{\partial J}{\partial \theta^{(k)}_{ij}} &= \frac{\partial J}{\partial a^{(k+1)}_i} \times \frac{\partial a^{(k+1)}_i}{\partial z^{(k+1)}_i} \times \frac{\partial z^{(k+1)}_i}{\partial \theta^{(k)}_{ij}} \\ &= \sum_{m=1}^{ns(k+2)} \delta^{(k+2)}_m \times \theta^{(k+1)}_{mi} \times \frac{\partial}{\partial z^{(k+1)}_i} f_{k+1}(z^{(k+1)}_i) \times a^{(k)}_j \end{aligned}$$

根据残差定义，我们知道：

$$\delta^{(k+1)}_i = \sum_{m=1}^{ns(k+2)} \delta^{(k+2)}_m \times \theta^{(k+1)}_{mi} \times \frac{\partial}{\partial z^{(k+1)}_i} f_{k+1}(z^{(k+1)}_i)$$

表示成矩阵形式即为：

$$\delta^{(k+1)} = (\theta^{(k+1)})^T \cdot \delta^{(k+2)} \cdot \frac{\partial}{\partial a^{(k+1)}_i} f_{k+1}(a^{(k+1)}_i)$$

因此，得到代价函数（交叉信息熵）的导数的矩阵形式：

$$\frac{\partial J}{\partial \theta^{(k)}} = \delta^{(k+1)} \times (a^{(k)})^T$$

6.1.2 程序算法

由于各个层的参数矩阵维数不确定，故在编程的时候采用元胞数组。计算梯度时，按照推导过程先计算残差 $\delta\{j\}$ ，进而计算各个待训练参数 (\vec{W}, \vec{b}) 的梯度。

根据反向传播算法，向后传播，计算第 j 层的残差， $j = c_1 + 1, c_1, \dots, 2$

$$\delta^{(j)} = \delta^{(j+1)} \times W^{(j)} \times f'_{j-1}(a^{(j)})$$

考虑第 j 层

$$\frac{\partial \text{cost}_i}{\partial \theta} = \frac{\partial \text{cost}_i}{\partial z^{(j)}} \cdot \frac{\partial z^{(j)}}{\partial \theta} = \delta^{(j)} \cdot a^{(j-1)}$$

进而计算各个待训练参数 (\vec{W}, \vec{b}) 的梯度

$$\frac{\partial}{\partial W^{(j)}} J(\vec{W}, \vec{b}) = \frac{1}{m} \times \delta^{(j+1)} \times a^{(j)} + \frac{\lambda}{2m} W^{(j)}$$

$$\frac{\partial}{\partial b^{(j)}} J(\vec{W}, \vec{b}) = \frac{1}{m} \sum_{j=2}^L \delta^{(j+1)}$$

关键实现代码如下：

计算残差	计算梯度
<pre> %% 1: 计算残差 delta = cell(c1 + 2, 1); for j=c1+2:-1:2 % 最后一层 if(j==c1+2) if(choise1(j-1)==0) % 最后一层传递函数为 sigmoid % delta[c1+2] = -y*log(h)-(1-y)*log(1-h); delta[c1+2] = h - y; else delta[c1+2] = -y ./ h .* relu_d(h) + (1-y) ./ (1-h) .* relu_d(1-h); end % 反向传播 else if(choise1(j-1)==0) % 最后一层激活函数为 sigmoid if(choise2(j)==1) delta{j} = delta{j+1}*(repmat(W{j}', ns(j-1), 1)') .* a{j} .* (1.-a{j}); else delta{j} = delta{j+1}*(W{j}') .* a{j} .* (1.-a{j}); end else % 最后一层激活函数为 relu if(choise2(j)==1) delta{j} = delta{j+1}*repmat(W{j}', ns(j-1), 1)' .* relu_d(a{j}); else delta{j} = delta{j+1}*W{j}' .* relu_d(a{j}); end end end end end </pre>	<pre> %% 2: 计算梯度 grad_W = cell(c1+1, 1); grad_b = cell(c1+1, 1); for j=1:c1+1 if(choise2(j)==1&& j==1) grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m * repmat(W{j}, 1, n)'; grad_W{j} = grad_W{j} (:, 1); elseif(choise2(j)==1) grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m*repmat(W{j}, 1, ns(j-1))'; grad_W{j} = grad_W{j} (:, 1); else grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m*W{j}; end grad_b{j} = 1/m* sum(delta{j+1}, 1); end %%把W, b 的偏导数排列成梯度列向量grad grad = [grad_W(1) (:); grad_b(1) (:)]; for j=2:c1+1 grad = [grad (:); grad_W(j) (:); grad_b(j) (:)]; end if(s~=size(grad)) bu = zeros(s-size(grad, 1), 1); grad = [grad (:); bu]; %error(' costFunction error!'); end </pre>

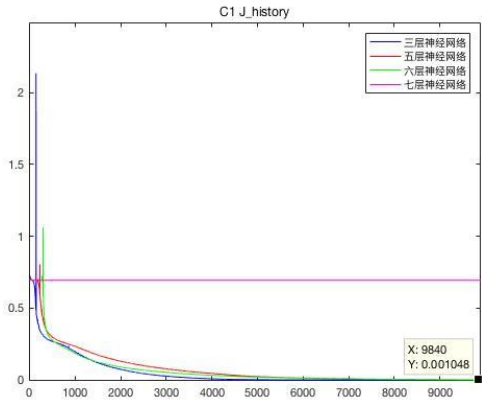
6.1.3 仅有全连接层的多层神经网络实验测试与讨论

为了验证多层神经网络的正确编写，并比较该结构神经网络与最初的神经网络结构的性能差异，我们分别使用 MNIST 数据集及其测试出的参数对该结构神经网络分别进行两次实验测试。获得结果如后文所述。

(1) 改变神经网络层数

设置最大迭代次数为 30000 次， $\alpha = 0.001$ ， $\lambda = 0.001$ 。对三层神经网络： $s_2 = 150$ ，连接函数均为 Sigmoid 函数；对五层神经网络：隐层节点数分别为 50、40、20，连接函数分别为 Sigmoid、Sigmoid、ReLU、Sigmoid 函数；对六层神经网络：隐层节点数分别为 40、20、10、5，连接函数分别为 Sigmoid、ReLU、Sigmoid、ReLU、Sigmoid 函数。获得结果如下所示。

	训练错误率	测试错误率
三层神经网络（含 1 隐层）	0.000000 0.000000	0.031000 0.036000
五层神经网络（含 3 隐层）	0.000000 0.000000	0.036000 0.045000
六层神经网络（含 4 隐层）	0.000000 0.000000	0.043000 0.048000
效果最好一次迭代过程	效果最好一次 J_history 图像	
<p>在第5573次迭代中, J=0.000005, 小于阈值, 自动终止迭代 时间已过 609.875305 秒。 MNIST:train_error_rate=0.000000 MNIST:test_error_rate=0.031000 时间已过 0.178908 秒。 =====以上是MNIST对照组, 以下是使用五层神经网络的实验组===== 3层隐层, 各50、40、20结点, 连接函数Sigmoid、Sigmoid、ReLU、Sigmoid: 已进行10000次迭代, J=0.000287 时间已过 540.330093 秒。 在第12856次迭代中, J=0.000005, 小于阈值, 自动终止迭代 时间已过 146.405204 秒。 MNIST:train_error_rate=0.000000 MNIST:test_error_rate=0.036000 时间已过 0.058508 秒。 =====以下是使用六层神经网络的实验组===== 4层隐层, 各40、20、10、5结点, 连接函数Sigmoid、ReLU、Sigmoid、ReLU、Sigmoid: 已进行10000次迭代, J=0.000956 时间已过 414.166124 秒。 已进行20000次迭代, J=0.000012 时间已过 412.942651 秒。 在第24560次迭代中, J=0.000005, 小于阈值, 自动终止迭代 时间已过 188.216839 秒。 MNIST:train_error_rate=0.000000 MNIST:test_error_rate=0.043000 时间已过 0.044136 秒。</p>		

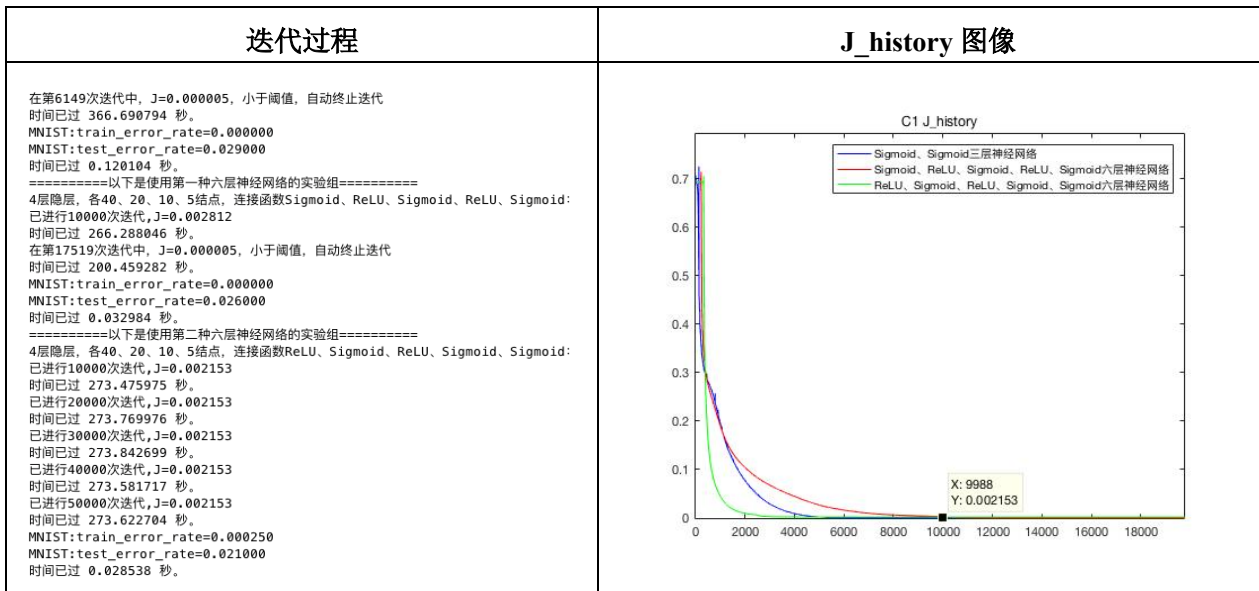


从结果中我们可以看出, 在该数据集和参数下, 三、五、六层神经网络的训练结果良好, 在不到 10000 次的迭代中已经能将 J 值收敛到 10^{-3} 量级上下, 且训练错误率均为 0%, 测试错误率也都小于 5%, 是一个比较好的水平。随着神经网络层数的增加, 获得相同效果每层所需要的隐藏节点数也将相对减少, 且需要避免“过拟合”现象的产生（例如五层、六层神经网络的测试正确率并不如简单的三层神经网络的测试正确率, 很可能是发生了“过拟合”现象）。但是随着网络复杂性的提升, 神经网络收敛概率也相对下降。例如图中所示的七层神经网络最终因没法收敛而训练失败。这启发我们在进行神经网络训练时需要根据数据集选取适合的网络结构。

(2) 改变神经网络连接函数

对于六层神经网络, 设置最大迭代次数为 50000 次, $\alpha = 0.001$, $\lambda = 0.001$, 隐层节点数分别为 40、20、10、5。分别训练两类连接函数组合, 分别为“Sigmoid、ReLU、Sigmoid、ReLU、Sigmoid 函数组合”和“ReLU、Sigmoid、ReLU、Sigmoid、Sigmoid 函数组合”。为了进行对照, 同样训练一个三层神经网络作为参照组。获得结果如下所示。

	训练错误率	测试错误率
三层神经网络（含 1 隐层）	0.000000	0.029000
Sigmoid、ReLU、Sigmoid、ReLU、Sigmoid 函数组合	0.000000	0.026000
ReLU、Sigmoid、ReLU、Sigmoid、Sigmoid 函数组合	0.000250	0.021000



从结果中我们可以看出, 在该数据集和参数下, “ReLU、Sigmoid、ReLU、Sigmoid、Sigmoid 函数组合” 从收敛速度和测试准确率上来说都略胜一筹。

通过以上实验测试, 我们既验证了仅有全连接层的多层神经网络函数编写正确, 能够实现改变网络层数和改变连接函数的功能, 又对不同网络层数和不同连接函数组合的神经网络进行了比较分析, 并为我们下一步构造含有卷积层的神经网络带来了启发。

6.2 卷积层梯度推导与实验

6.2.1 方法原理

在全连接层实验中, 我们看到待训练的参数随着层数以及各层节点数的增加而增加, 对于相邻的两层来说, 假设他们的层节点数各有 $ns(j-1), ns(j)$ 个, 则连接这两层的待训练参数为:

$$W_{ik} \quad i = 1, \dots, ns(j-1), k = 1, \dots, ns(j)$$

共有 $ns(j-1) \times ns(j)$ 个。可以想象, 层数以及各层节点数增加到一定程度, 待训练参数 W 必然呈爆炸增长的趋势。

为了减少待训练参数, 提高网络的训练速度以及效率, 卷积神经网络应用而生。在本例中采用一维卷积的形式。

同样考虑相邻两层之间的传递参数 (偏移参数 b 暂不考虑)。在卷积层运算过程中, 我们将 $ns(j-1) \times ns(j)$ 个参数化简为 $ns(j)$ 个参数。(之所以选取 $ns(j)$ 个参数是出于矩阵求导维度的考虑, 在后续的证明中会有体现。)

从全连接层的推导来看, 其前向传播公式为:

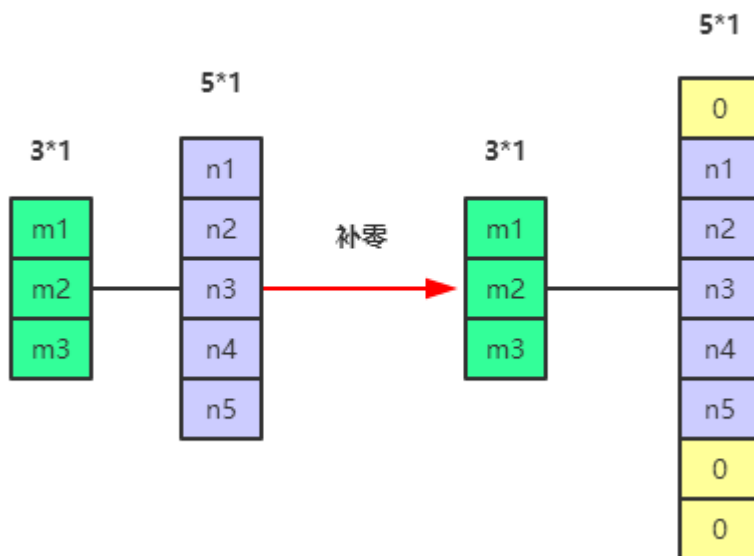
$$\begin{cases} z^{(j)}_i = \sum_{k=1}^{ns(j-1)} a^{(j-1)}_k \times W^{(j-1)}_{ik} \\ a^{(j)}_i = f_{sigmoid \text{ or } relu}(z^{(j)}_i) \end{cases}$$

而在卷积层，其前向传播公式为：

$$\begin{cases} z^{(j)}_i = (a^{(j-1)} * W^{(j-1)})_i \\ a^{(j)}_i = f_{\text{sigmoid or reLU}}(z^{(j)}_i) \end{cases}$$

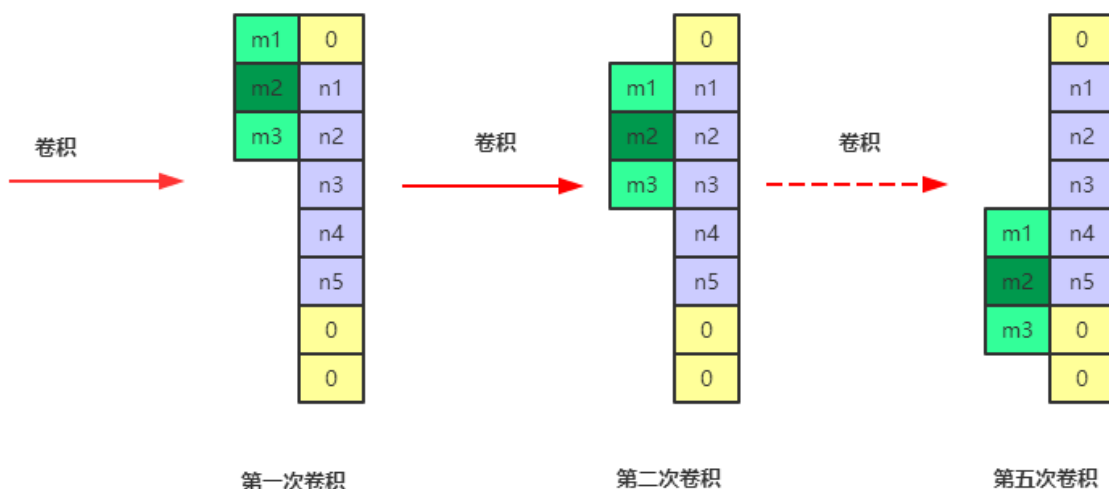
我们用图例来说明定义中的卷积形式：

首先假设神经网络第 j 层传入的每一个数据维度为 $ns(j-1)$ ，而第 j 层有 $ns(j)$ 个节点，不妨设其全连接层参数维度为 3×5 ，则首先对卷积核补零：



若传来的参数维度的一半为： $centre = \frac{1}{2}ns(j-1)$ ，则对 $ns(j)$ 层前面补 $centre-1$ 个零，后面补 $ns(j-1)-centre$ 个零。

接着依次进行卷积运算：



第一次卷积的结果为： $m1 \times 0 + m2 \times 1 + m3 \times 2$ ，此后类推。

由此可见：

$$z^{(j)}_i = (a^{(j-1)} * W^{(j-1)})_i = a^{(j-1)} \times W^{*(j-1)}(i: i + ns(j-1) - 1)$$

其中 $W^{*(j-1)}(i: i + ns(j-1) - 1)$ 为补零后的矩阵中第 i 个元素到第 $i + ns(j-1) - 1$ 个元素组成的向量。

下面进行卷积层参数的梯度计算。

首先，全连接层的 BP 算法中，残差定义为：

$$\delta^{(j)} = \delta^{(j+1)} \times W^{(j)} \times f'_{j-1}(a^{(j)})$$

通过残差可以计算各参数的梯度为：

$$\begin{aligned} \frac{\partial}{\partial W^{(j)}} J(\vec{W}, \vec{b}) &= \frac{1}{m} \times \delta^{(j+1)} \times a^{(j)} + \frac{\lambda}{2m} W^{(j)} \\ \frac{\partial}{\partial b^{(j)}} J(\vec{W}, \vec{b}) &= \frac{1}{m} \sum_{j=2}^L \delta^{(j+1)} \end{aligned}$$

对于卷积层，同样的，根据 BP 算法，要想求得层 j 的每个神经元对应的权值的权值更新，就需要先求层 j 的每一个神经节点的残差。

为了有效计算层 j 的残差 $\delta^{(j)}$ ，我们需要上采样 upsample 这个下采样 downsample 层对应的残差 $\delta^{(j+1)}$ ，这样才使得这个残差维度与卷积层的层节点数大小一致，然后再将层 j 的残差的激活值的偏导数与从第 $j+1$ 层的上采样得到的逐元素相乘。

即，

$$\delta^{(j)} = \delta^{(j+1)} \times up(W^{(j)}) \times f'_{j-1}(a^{(j)})$$

其中 $up(\sim)$ 函数表示上采样的过程，若上层的层节点数为 $ns(j-1)$ ，则其简单的将 $W^{(j)}$ 水平方向上拷贝 $ns(j-1)$ 次。这样的结果是因为该卷积层的卷积核恰好为该层节点数。

得到了残差，我们发现卷积层残差的维度与全连接层某一个样本的残差的维度一致，故在全连接层的 BP 算法中的递推公式仍然适用。

6.2.2 程序算法

类似全连接层，卷积层的算法也是先计算残差 $\delta^{(j)}$ ，然后根据残差和 BP 算法的递推公式进行梯度计算。关键实现代码如下：

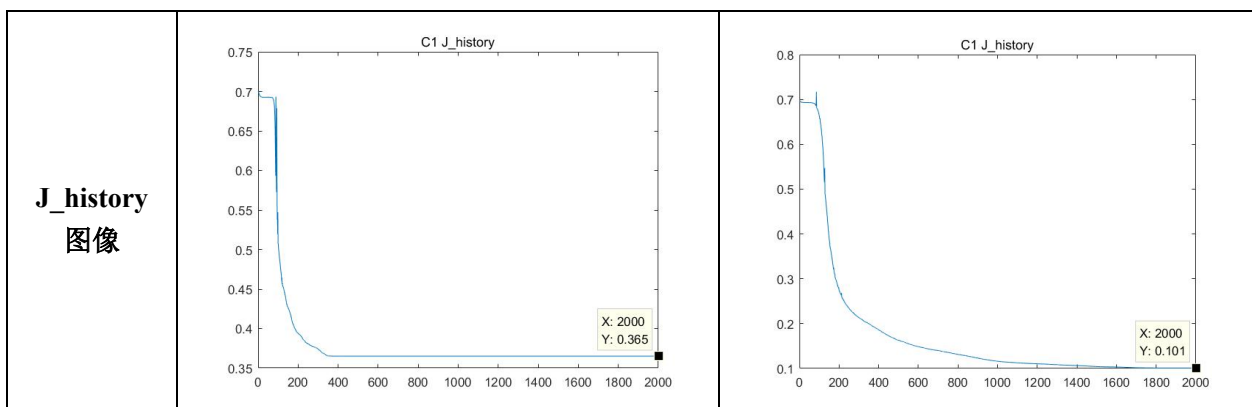
计算残差	计算梯度
<pre> %% 1: 计算残差 delta = cell(c1 + 2, 1); for j=c1+2:-1:2 % 最后一层 if(j==c1+2) if(choose1(j-1)==0) % 最后一层传递函数为 sigmoid % delta(c1+2) = -y*log(h)-(1-y)*log(1-h); delta(c1+2) = h - y; else delta(c1+2) = -y ./ h .* relu_d(h) + (1-y) ./ (1-h) .* relu_d(1-h); end % 反向传播 else if(choose1(j-1)==0) % 最后一层激活函数为 sigmoid if(choose2(j)==1) delta{j} = delta{j+1}*(repmat(W{j}', ns(j-1), 1)') .* a{j} .* (1.-a{j}); else delta{j} = delta{j+1}*(W{j}') .* a{j} .* (1.-a{j}); end else % 最后一层激活函数为 relu if(choose2(j)==1) delta{j} = delta{j+1}*repmat(W{j}', ns(j-1), 1)' .* relu_d(a{j}); else delta{j} = delta{j+1}*W{j}' .* relu_d(a{j}); end end end end end </pre>	<pre> %% 2: 计算梯度 grad_W = cell(c1+1, 1); grad_b = cell(c1+1, 1); for j=1:c1+1 if(choose2(j)==1&&j==1) grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m * repmat(W{j}, 1, n)'; grad_W{j} = grad_W{j}(:, 1); elseif(choose2(j)==1) grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m*repmat(W{j}, 1, ns(j-1))'; grad_W{j} = grad_W{j}(:, 1); else grad_W{j} = 1/m*a{j}'*delta{j+1}+lambda/m*W{j}; end grad_b{j} = 1/m* sum(delta{j+1}, 1); end % 把W, b 的偏导数排列成梯度列向量grad grad = [grad_W{1}(:); grad_b{1}(:)]; for j=2:c1+1 grad = [grad(:); grad_W{j}(:); grad_b{j}(:)]; end if(s~=size(grad)) bu = zeros(s-size(grad, 1), 1); grad = [grad(:); bu]; %error('costFunction error!'); end </pre>

6.2.3 含有卷积层的多层神经网络实验测试与讨论

为了验证含有卷积层的多层神经网络的正确编写，并测试其性能，我们对该结构的神经网络进行了实验测试。

设置最大迭代次数为 10000 次， $\alpha = 0.1$ ， $\lambda = 0$ 。神经网络隐层先后为一层节点数为 10 的全连接层和一层节点数为 12 的隐层；连接函数分别为 ReLU、Sigmoid、Sigmoid。

	第一次训练	第二次训练
训练错误率	0.149000	0.021000
测试错误率	0.186000	0.057000
迭代过程	<p>=====含卷积层的神经网络：=====</p> <p>已进行100次迭代, J=0. 506625 时间已过 6. 436778 秒。</p> <p>已进行200次迭代, J=0. 393892 时间已过 6. 241023 秒。</p> <p>已进行300次迭代, J=0. 374263 时间已过 6. 228461 秒。</p> <p>已进行400次迭代, J=0. 365055 时间已过 6. 382073 秒。</p> <p>已进行500次迭代, J=0. 365046 时间已过 6. 537192 秒。</p> <p>已进行800次迭代, J=0. 365046 时间已过 19. 379713 秒。</p> <p>已进行1200次迭代, J=0. 365046 时间已过 25. 249687 秒。</p> <p>已进行1600次迭代, J=0. 365046 时间已过 24. 491833 秒。</p> <p>已进行2000次迭代, J=0. 365046 时间已过 24. 538734 秒。</p> <p>MNIST:train_error_rate=0. 149000 MNIST:test_error_rate=0. 186000 时间已过 0. 635399 秒。</p>	<p>=====含卷积层的神经网络：=====</p> <p>已进行100次迭代, J=0. 663840 时间已过 6. 152440 秒。</p> <p>已进行200次迭代, J=0. 274756 时间已过 6. 111232 秒。</p> <p>已进行300次迭代, J=0. 213439 时间已过 6. 127163 秒。</p> <p>已进行400次迭代, J=0. 186457 时间已过 6. 056405 秒。</p> <p>已进行500次迭代, J=0. 162928 时间已过 6. 118569 秒。</p> <p>已进行800次迭代, J=0. 131301 时间已过 18. 387525 秒。</p> <p>已进行1200次迭代, J=0. 110408 时间已过 24. 485561 秒。</p> <p>已进行1600次迭代, J=0. 103259 时间已过 24. 400529 秒。</p> <p>已进行2000次迭代, J=0. 100971 时间已过 24. 354120 秒。</p> <p>MNIST:train_error_rate=0. 021000 MNIST:test_error_rate=0. 057000 时间已过 0. 114208 秒。</p>



从训练过程和结果中我们可以发现，随着网络复杂性的提升，尤其是加入卷积层之后，神经网络收敛概率也相对下降。在该数据集和参数下，含有卷积层的神经网络较难收敛；但一旦收敛，虽然我们设的层数并不算多，但是它的训练结果还是较好的。

通过以上实验测试，我们既验证了含有卷积层的多层神经网络函数编写正确，既能够实现改变网络层数和改变连接函数的功能，又能够实现对隐层种类（卷积层和全连接层）的改变。一个功能较为完善的多层神经网络可以说是成功完成。

七、人员分工说明

从开始进行课题到结束大概耗费包含工作日的一周左右的时间，能以如此快的速度完成规模如此庞大的课题也令我们自己感到不可思议。究其原因我们认为可以归功于两位队友的默契配合和合理分工。两人总工作量比例几乎相同，且在每个作业要求中的任务都有着自己的贡献。以下是分项目说明。

任务	林子坤负责工作	郑文举负责工作
A1 梯度下降法代码实现	代码编写、代码改进、报告撰写	代码审核
A2 基本三层神经网络的梯度推导	资料查阅、梯度推导、代码编写、报告撰写	资料查阅、梯度推导、代码审核纠错改进
A3 Bankruptcy 数据集实验	参与代码编写、参与实验测试、报告撰写	数据集格式处理、参与代码编写、参与实验测试
B1 参数选择研究	资料查阅、参与实验测试、报告撰写	资料查阅、参与实验测试
B2 Wine Quality 数据集实验	参与代码编写、参与实验测试、报告撰写	数据集格式处理、参与代码编写、参与实验测试
B3 MNIST 手写字数据集实验	参与代码编写、参与实验测试、报告撰写	数据集格式处理、参与代码编写、参与实验测试
B4 ReLU 激活函数梯度推导与实验	资料查阅、代码审核纠错改进、参与实验测试、参与报告实验部分撰写	资料查阅、梯度推导、代码编写、参与实验测试、参与报告原理部分撰写
C1 深层网络梯度推导与实验	资料查阅、代码审核纠错改进、参与实验测试、参与报告实验部分撰写，最终报告汇总	资料查阅、梯度推导、代码编写、参与实验测试、参与报告原理部分撰写

八、遇到的问题与解决方法

(1) 矩阵求导相关：标量对矩阵的导数

在刚开始接触矩阵求导的时候，我们只是简单的看一看老师给的资料，觉得和普通的标量求导没什么太大的区别，就没怎么在意。接着，我们在程序中直接使用的时候发现了一大堆矩阵相乘而维数不匹配问题，为了图省事，一开始我们只是为了凑矩阵乘法所要求的维度对矩阵进行“随意”的转置，先让程序跑起来，尽管之后的程序在准确率上维持一个很高的水平，但秉承着严谨、一丝不苟学术态度的我们在交流讨论之后，对整个程序单一的待训练参数进行逐步跟踪，最终发现是在矩阵求导这里出现了问题！

对于一个矩阵表示的标量表达式 $\alpha = y^T A x$ ，其偏导数应该为 $\frac{\partial \alpha}{\partial x} = y^T A$ ，但是其对于 y 的偏导数可不是简单的 Ax ，还要进行一个转置，即 $\frac{\partial \alpha}{\partial y} = x^T A^T$ 。

(2) 局部最优解和全局最优解及初值选择

在本次作业中的实验过程中，我们进行的不少实验结果的代价函数并没有收敛到全局最优解，而是陷入了局部最优解的邻域内。造成这样的原因是最速下降法在局部最优或者全局最优的邻域内的搜索效率非常之低，为了改善这样的情况，一方面，我们调整了算法的学习率，一方面增加了算法的迭代次数，这样虽然得到了较好的结果，却也是付出了相当的时间代价。通过查阅文献，我们发现了另一种并行的改善途径，就是修改初值。起初我们的初值选取即为零，这样的对称初值对于算法来说很容易陷入局部最优，为了使得对称失效，我们采用正态分布生成随机数的方法，大大提高了算法的执行效率。

(3) 最速下降法的局限性和改善

在课堂上，王老师反复强调最速下降法在局部最优或者全局最优的样本点的搜索效率较低，这样就导致了两种后果：其一，陷入局部最优的邻域“不可自拔”；其二，在全局最优的邻域内短时间无法收敛到最优。我们认为，造成这样的结果的一个重要原因是学习率是事先人为给定的。为了使程序尽可能少的依赖这些非核心参数，提高程序的运行效率，我们采用一种动态改变学习率的方法——“Bold Driver”！名为改变学习率，实际上也是对某一次迭代学习的梯度进行调整：如果相对于上一次迭代，错误率减少了，就可以以 5% 的幅度增大学习率；如果相对于上一次迭代，错误率增大了（意味着跳过了最优值），那么减少学习率到之前的 50%。

(4) 参数选择与“Great Search”

对于参数选择，在查阅了相关的文献之后，我们发现，很多时候参数选择及调整都是遵循着一定的“经验公式”，没有办法给出理论上的最优参数。于是，在进行“WINE QUALITY”实验的时候，我们怎么调参数得到的错误率始终维持在一定范围（25%左右），与之前“BANKRUPTCY”实验结果相差甚远，这让我们相当苦恼。后来我们采用“广撒网”的“Great Search”策略，将各个参数都在几个值之间跑一遍，选择实验最优数据作为最后

的参数选择。

九、总结与感受

公式理论推导相关：在进行理论推导的过程中，我们更加深刻地理解到了“细节决定成败”这一至理名言！公式的理论推导是整个算法的核心所在，哪怕是一个小地方出错（比如一个变量的正负号），都会出现一连串难以解释的现象。通过这次大作业的练习，我们逐步养成了严谨、一丝不苟的学术态度，这将是以后生活中一笔不可多得的财富。

——郑文举

这是一次令人激动的课题经历。因为在此之前，我们对“深度学习”的理解还完全停留在理论中，而通过本次课题，我们揭开了深度学习和神经网络的神秘面纱，且通过实践对深度学习的理论有了更加深入的了解和认识。当我们看到在我们改进参数后，神经网络的收敛变快了、准确率提升了，我们的激动之情难以言表。

在最后，感谢老师和助教为我们精心设计的大作业题，让我们有机会接触到并深入探索“深度学习”这个领域，在训练数据集的过程中更加深入地理解理论；感谢我队友郑文举同学，正是他超强的数学功底和默契的配合才能让我们能够在如此短的时间内完成从公式推导到代码编写到报告撰写的过程；同时还要感谢姚禹光同学提供的性能超棒的个人电脑，在我的电脑同时开两个 MATLAB 跑到快瘫痪的时候，我还能够同时使用他的电脑进行报告的撰写和代码的修改（顺便打游戏）。

当然，在这一周时间内，最应该感谢的就是我们的计算机了。感谢它们在连续几天 24 小时的连续工作，且并没有出什么严重的差错，这才有了我们前文所列举出的丰富的实验记录与实验数据。

——林子坤

参考文献

- [1] <http://deeplearning.stanford.edu/wiki/index.php/%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C>
- [2] <https://wenku.baidu.com/view/7195de8283d049649b66588e.html>
- [3] <http://www.docin.com/p-807765618.html>
- [4] <http://www.bubuko.com/infodetail-675460.html>
- [5] <http://www.willamette.edu/~gorr/classes/cs449/momrate.html>
- [6] <https://www.zhihu.com/question/20700829>
- [7] <http://blog.csdn.net/u013146742/article/details/51986575>