深 圳 大 学 实 验 报 告

课 程 名 称：机器学习

实验项目名称：基于 BP 神经网络的手写数字识别

学　　院：计算机与软件学院

专　　业：计算机科学与技术

指 导 教 师：贾森 徐萌

报告人：刘睿辰 学号：2018152051 班级：数计班

实 验 时 间：2021.4.5-2021.5.9

实验报告提交时间：2021.5.9

教务处制

一、 实验目标

1. 理解 BP 神经网络算法；
2. 实现 BP 神经网络在手写数字识别上的应用。

二、 实验环境

1. 操作系统：*Windows* 10；
2. 实验平台：*Matlab* 2019*a*。

三、 实验内容、步骤与结果

1. 神经网络的数学基础

1.1 非线性分类 Logistic 回归的弊端

我们以二分类问题为例，如图 1 所示的分类器。

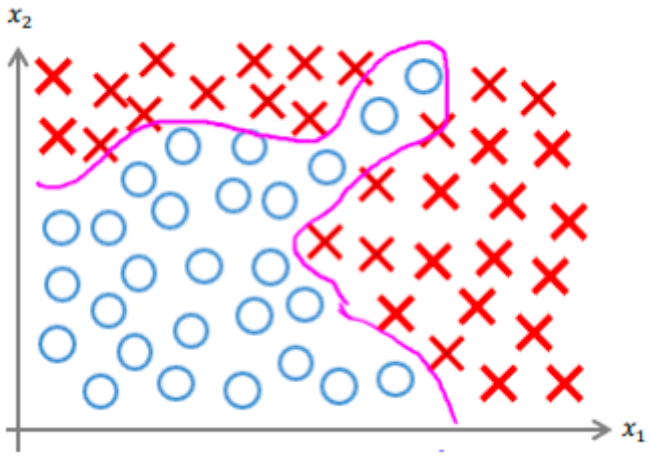


图 1. 二分类问题分类器

这个分类函数表示成x_1和x_2的多项式为

$$\theta_0+\theta_1x_1+\theta_2x_2+\theta_3x_1x_2+\theta_4x_1^2x_2+\theta_5x_1^3x_2+\theta_6x_1x_2^2+...$$

而这里仅仅考虑了两个自变量。事实上，对于分类问题，有时候自变量的个数非常多。如果是 100 个特征形成 100 个自变量，那么**只考虑二次多项式**的话，就会有$x_1^2,x_1x_2,x_1x_3,x_1x_4,...,x_1x_{100},x_2^2,x_2x_3,...,x_{100}^2$这个多个项，也就是 5000 多个项。如果是$n$项的话，总的项数大概是$\frac{1}{2}n^2$，复杂度为$O(n^2)$，计算量比较大。当然，如果只使用单变量的二次项像$x_1^2,x_2^2,...,x_{100}^2$组成一个多项式的话可以减少二次项个数，但我们得到的可能是一个椭圆形的图象，不可能拟合出像图 1 这样的分界线。

我们刚刚只用二次项来拟合分界函数的时候就有了 5000 个特征，那么如果将**三次项**加入的话，三次项个数的复杂度为$O(n^3)$，这样特征数就更多了，造成了特征空间的急剧

膨胀。

综上所述，使用 Logistic 回归来实现用高阶多项式做分类器进行分类并不是一个好做法。现实的机器学习问题中，特征数往往很多。例如一个50×50的图片共有 2500 个像素，每一个像素都代表一个特征，那么如果仅仅用二次项来拟合边界函数的时候就有 300 多万个特征，计算代价非常高

1.2 神经网络--前馈网络

模拟神经元的工作：如图 2 所示。

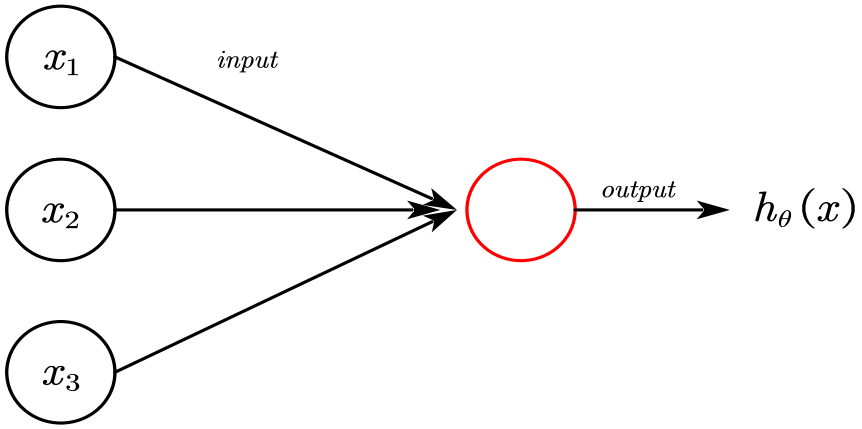


图 2. 神经元模型

在图 2 中，我们模拟了一个简单神经元的工作。x_1，x_2和x_3是所有特征，每一个特征应该都对应了一个权重θ_i，在所有的特征值都输入了之后，它们按照各自的权重得到线性组合，我们得到下式，这里m代表特征的个数。

$$\sum_{i=1}^{m}\theta_i x_i$$

得到了线性组合之后，神经元用 sigmoid 函数进行激活，所以神经元的输出通道就是

$$output = sigmoid\left(\sum_{i=1}^{m}\theta_i x_i\right)$$

那么在这里，神经元的输入就应该是一个样本，包含m个特征，也就是

$$\boldsymbol{x}=\begin{bmatrix}x_0\\x_1\\x_2\\\vdots\\x_m\end{bmatrix}$$

这里我们在x_1前面加了一个偏置项x_0。偏置项的作用在于提高函数的灵活性，提高了神经元的拟合能力。有了偏置项，sigmoid 函数可能并不在 0 点函数值达到 0.5，如图 3 所示。

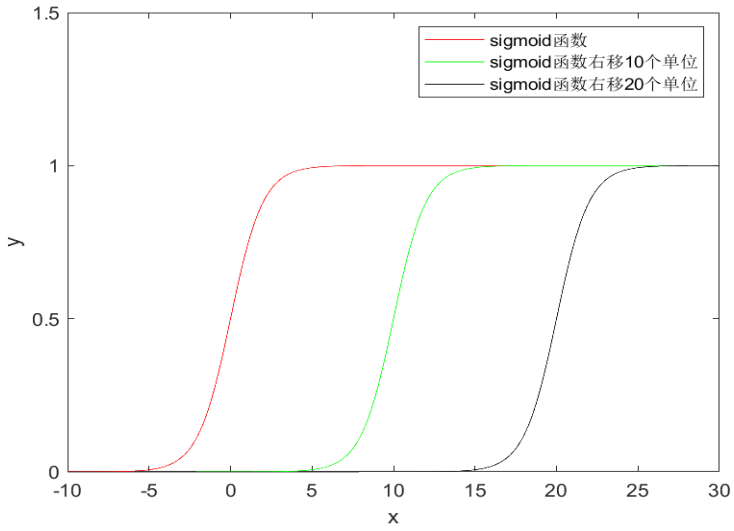


图 3. 偏置项的存在使得 sigmoid 函数左右移动

现在，我们来讨论神经元组成的神经网络，如图 4 所示。这是一个三层网络，包含了输入层、一个隐藏层以及输出层。

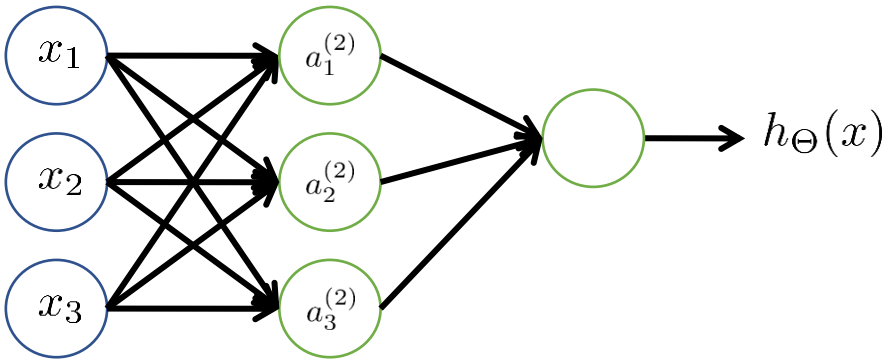


图 4. 神经网络结构

接下来我们需要计算隐藏层神经元的值。正如生物神经元一样，每一个神经元都要有输入以及传给下一个神经元的输出。关于隐藏层神经元的输入值，根据我们之前单个神经元的分析，我们用之前的方法来计算。由于神经元比较多，这里面我们有必要进行符号约定，如表 1 所示。

符号	含义
$a_i^{(j)}$	第 j 层的第 i 个神经元
θ^j	第 j 层到第 $j+1$ 层的权重值

表 1. 神经网络符号约定

然后对于每一个$a_i^{(j)}$我们都使用之前的方法来计算，可以得到以下式子。注意这里面添加了偏置项x_0.

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$

隐藏层的计算结束之后，我们就可以计算神经网络的输出值

$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

那么也就是说，从输入层到中间唯一的隐藏层，我们用以下矩阵算式来过渡

$$\boldsymbol{z}^{(2)} = \boldsymbol{\theta}^{(1)} \boldsymbol{x}$$

为了统一起见，我们一般将$\boldsymbol{x}$记作$\boldsymbol{a}^{(1)}$，再通过可以得到下式

$$\boldsymbol{a}^{(2)} = g(\boldsymbol{z}^{(2)}) = g(\boldsymbol{\theta}^{(1)} \boldsymbol{a}^{(1)})$$

注意$\boldsymbol{a}^{(2)}$要加偏置项。那么从中间隐藏层到输出层，我们得到过渡矩阵算式

$$\boldsymbol{z}^{(3)} = \boldsymbol{\theta}^{(2)} \boldsymbol{a}^{(2)}$$

然后通过 sigmoid 函数进行激活，得到

$$\boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)}) = g(\boldsymbol{\theta}^{(2)} \boldsymbol{a}^{(2)})$$

按照这个方式我们可以得到最终的输出值。不管是单隐层前馈网络（如图 5(a)所示）还是多隐层前馈网络（如图 5(b)所示），我们都可以结合神经元的输入与输出不断向前推进直到得到最终的结果。

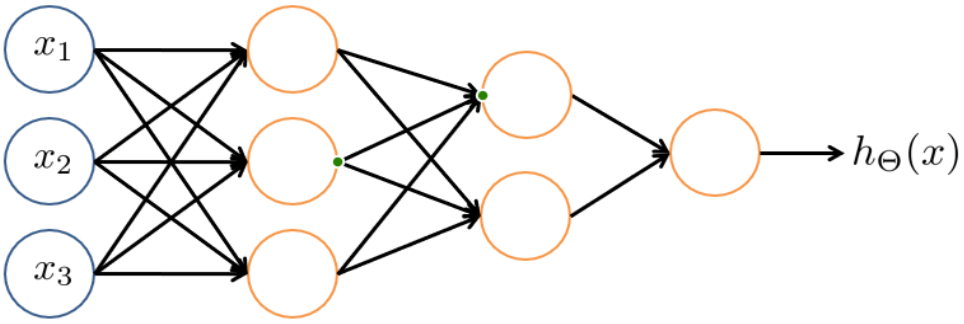


图 5(a) 单隐层前馈网络

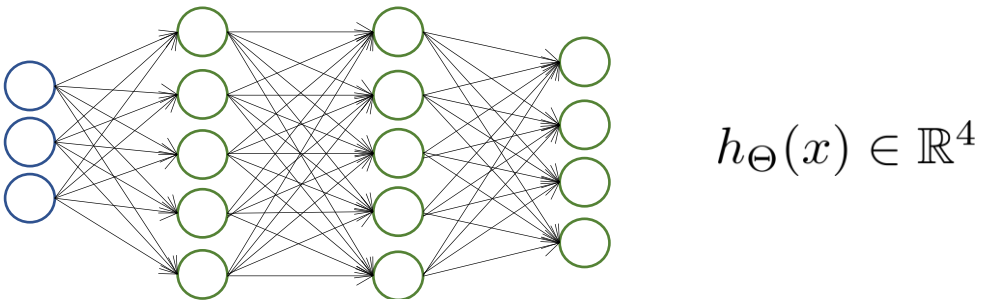


图 5(b) 多隐层前馈网络

图 5. 前馈网络种类

那么前馈网络如何完成多分类任务呢？
如果输出单元仅仅只有一个节点那就是二分类问题，如果输出单元有多个节点那就是多分类问题。例如图 5(b)中，输出节点有 4 个，那这里就是一个四分类问题。因为实际情况中，对于n 分类问题，一般不以$1,2,3,...,n$ 为判断标准进行分类，而是以

$$\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix},\begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix},\begin{bmatrix}0\\0\\1\\\vdots\\0\end{bmatrix},...,\begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$$

为分类标准。例如对于一个四分类问题，有如图 6 所示的分类结果。

$$h_\theta(x)=\underset{if\ class\ A}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}}\quad\underset{if\ class\ B}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}\quad\underset{if\ class\ C}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}\quad\underset{if\ class\ D}{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}$$

图 6. 神经网络对于多分类问题的目标输出

图 6 是我们以一个四分类问题为例，我们希望神经网络可以完美地输出这样的结果。但是实际上这是不可能的，我们希望可以尽量达到这样的效果，也就是通过计算代价函数，然后通过优化方法来降低代价的值，达到参数最优的目的。
在之前的 Logistic 回归中我们得到的代价函数表达式为

$$J(\boldsymbol{\theta})=\frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\ln\left(h_\theta(x^{(i)})\right)-(1-y^{(i)})\ln\left(1-h_\theta(x^{(i)})\right)\right)+\frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2$$

其中$\frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2$为正则项，目的是防止过拟合现象的发生。

类似地，我们得出神经网络的代价函数为

$$J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left(-y_k^{(i)}\ln\left(h_\theta(x^{(i)})\right)_k+(1-y_k^{(i)})\ln\left(1-\left(h_\theta(x^{(i)})\right)_k\right)\right)+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\theta_{ji}^{(l)})^2$$

这里L代表网络的总的层数（包括输入层与输出层），S_l代表第l层的节点数。由于L个网络层中间只有$L-1$个过渡，所以l从 1 到$L-1$变化。此外，每一个过渡矩阵$\boldsymbol{\theta}$的大小都应该是$S_{l+1}\times S_l$。

1.3 神经网络—误差逆传播(backpropagation)

前面在前馈网络中我们得到了代价函数的表达式，那么我们就想通过降低代价函数的值来达到优化的目的。接下来我们来看逆传播的过程。
在如图 7 所示的网络中，我们通过前馈可以得到一个输出值，过程如下

$$\boldsymbol{a}^{(1)}=\boldsymbol{x}$$

$$\boldsymbol{z}^{(2)}=\boldsymbol{\theta}^{(1)}\boldsymbol{a}^{(1)},\boldsymbol{a}^{(2)}=g(\boldsymbol{z}^{(2)})\ (\text{add bias unit}\ a_0^{(2)})$$

$$\boldsymbol{z}^{(3)}=\boldsymbol{\theta}^{(2)}\boldsymbol{a}^{(2)},\boldsymbol{a}^{(3)}=g(\boldsymbol{z}^{(3)})\ (\text{add bias unit}\ a_0^{(3)})$$

$$\boldsymbol{z}^{(4)}=\boldsymbol{\theta}^{(3)}\boldsymbol{a}^{(3)},h_\theta(x)=\boldsymbol{a}^{(4)}=g(\boldsymbol{z}^{(4)})$$

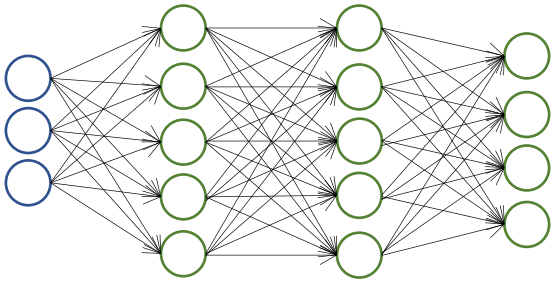


图 7. 四层网络

现假设神经网络输出了一个结果$\boldsymbol{a}^{(4)}$，为了衡量网络输出的结果是否准确，我们最直观的方法就是计算误差，而误差在这里我们就以差值作为衡量，误差以$\boldsymbol{\delta}^{(4)}$来表示，代表第$j$层的误差。我们以$\boldsymbol{y}$作为理想输出，那么我们可以得到

$$\boldsymbol{\delta}^{(4)}=\boldsymbol{a}^{(4)}-\boldsymbol{y}$$

根据反向传播的定义，我们来进行反推的过程，以图 8 所示网络为例。

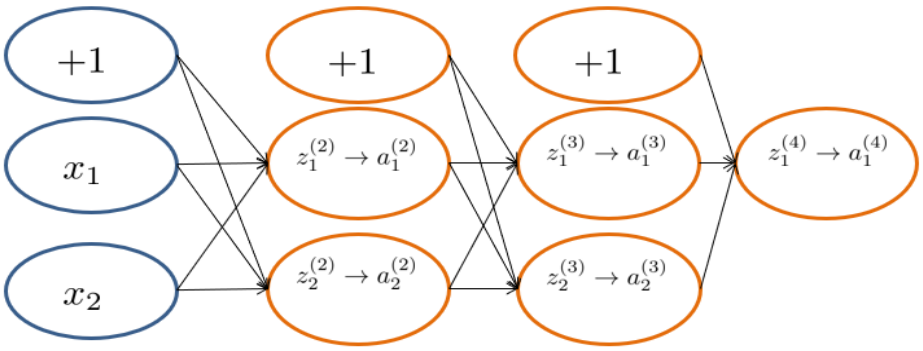


图 8. 逆传播过程示意图

现在我们得到了$\boldsymbol{\delta}^{(4)}$，我们希望通过$\boldsymbol{\delta}^{(4)}$得到第三层所有节点的误差值。我们可以知道$\boldsymbol{\delta}^{(4)}$实际上是由于第三层的误差造成的，如图 9 所示。

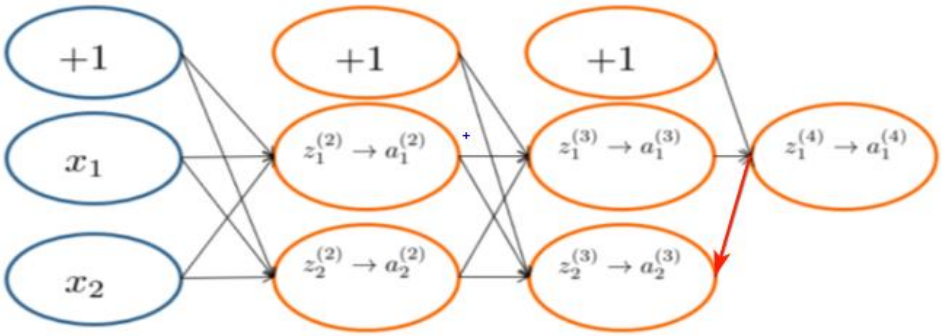


图 9. 逆传播过程 1

所以以$a_2^{(3)}$为例我们有

$$\delta_2^{(3)}=\theta_{12}^{(3)}\delta_1^{(4)}$$

得到了$a_2^{(3)}$这个节点的误差之后，我们可以用同样的方法得到$\delta_1^{(3)}$，然后结合这两个误差得到$a_2^{(2)}$的误差，如图 10 所示。

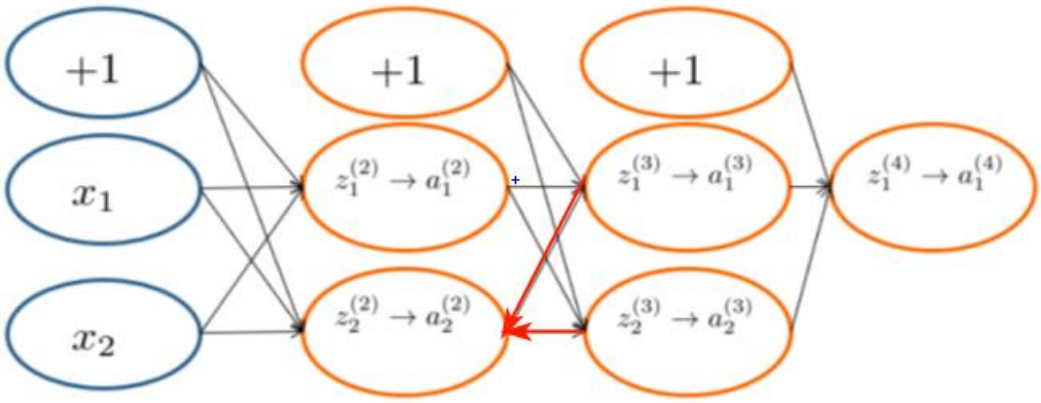


图 10. 逆传播过程 2

所以误差可以表示为

$$\delta_2^{(2)}=\theta_{12}^{(2)}\delta_1^{(3)}+\theta_{22}^{(2)}\delta_2^{(3)}$$

按照这个方法，我们得到各层误差表达式为

$$\boldsymbol{\delta}^{(3)}=\left(\boldsymbol{\theta}^{(3)}\right)^T\boldsymbol{\delta}^{(4)}.*g'\left(\boldsymbol{z}^{(3)}\right)$$

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

由于 sigmoid 函数有一个良好的性质，即

$$f'(x)=f(x)\left(1-f(x)\right)$$

所以上式可以替换为

$$\boldsymbol{\delta}^{(3)}=\left(\boldsymbol{\theta}^{(3)}\right)^T\boldsymbol{\delta}^{(4)}.*\boldsymbol{a}^{(3)}.*\left(1-\boldsymbol{a}^{(3)}\right)$$

$$\boldsymbol{\delta}^{(2)}=\left(\boldsymbol{\theta}^{(2)}\right)^T\boldsymbol{\delta}^{(3)}.*\boldsymbol{a}^{(2)}.*\left(1-\boldsymbol{a}^{(2)}\right)$$

得到了各层误差的计算方法之后，我们应该用误差来校正参数来达到训练神经网络的目的。这里我们依然选用梯度下降的方法。这里面我们看到$J(\boldsymbol{\theta})$是一个关于很多变量的函数，所以我们要求得偏导数才能进行梯度下降法。

我们在这里面用误差来求得梯度。这里面我们用$D_{ij}^{(l)}$这个矩阵来代表梯度$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\boldsymbol{\theta})$，而考虑到正则化的问题，我们首先构造矩阵$\Delta_{ij}^{(l)}$。对于训练集$\{(\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}),...,(\boldsymbol{x}^{(m)},\boldsymbol{y}^{(m)})\}$，我们进行一次循环操作，针对每一个样本，都进行前馈得到$\boldsymbol{a}^{(l)},l=2,3,...,L$，然后计算

$\boldsymbol{\delta}^{(L)}$，根据$\boldsymbol{\delta}^{(L)}$来计算$\boldsymbol{\delta}^{(L-1)},\boldsymbol{\delta}^{(L-2)},...,\boldsymbol{\delta}^{(2)}$从而得到各层的误差值。然后根据误差值构造梯度矩阵$\Delta_{ij}^{(l)}$，我们有

$$\Delta_{ij}^{(l)}:=\Delta_{ij}^{(l)}+a_j^{(l)}\delta_i^{(l+1)}$$

关于这个公式，我们可以这样理解：$\delta_i^{(l+1)}$产生的原因正是由于$\theta_{i1}^{(l)},\theta_{i2}^{(l)},...,\theta_{i,S_l}^{(l)}$的误差造成的。而为了校正$\theta_{ij}^{(l)}$的值，我们需要将已知的误差加回去来得到更准确的结果。当然，该式子也可表示成矩阵乘法为

$$\boldsymbol{\Delta}^{(l)}:=\boldsymbol{\Delta}^{(l)}+\boldsymbol{\delta}^{(l+1)}\left(\boldsymbol{a}^{(l)}\right)^T$$

循环结束之后，我们要用$\boldsymbol{\Delta}^{(l)}$矩阵来得到$\boldsymbol{D}$矩阵。考虑到正则化的问题，对于$\boldsymbol{a}^{(l)},l=2,3,...,L$中的偏置项无需进行正则化，而其余项都需要进行正则化，所以有

$$D_{ij}^{(l)}:=\begin{cases}\dfrac{1}{m}\Delta_{ij}^{(l)}+\dfrac{\lambda}{m}\theta_{ij}^{(l)} & \text{if } j\neq 0\\ \dfrac{1}{m}\Delta_{ij}^{(l)} & \text{if } j=0\end{cases}$$

得到了梯度之后就可以用梯度下降法来计算$\theta_{ij}^{(l)}$，达到参数优化的目的。

值得注意的是，我们这种方法是需要检验的，也就是说我们计算出来的梯度需要被检验。事实上，反向传播算法很难调试得到正确结果，尤其是当实现程序存在很多难于发现的 bug 时。这些错误也会得到一个看似十分合理的结果，但实际上比正确代码的结果要差。因此，仅从计算结果上来看，我们很难发现代码中有什么东西遗漏了。所以我们有必要进行**梯度检验**。

梯度检验的思想：以$\theta\in\mathbb{R}^2$为例，如图 11 所示。

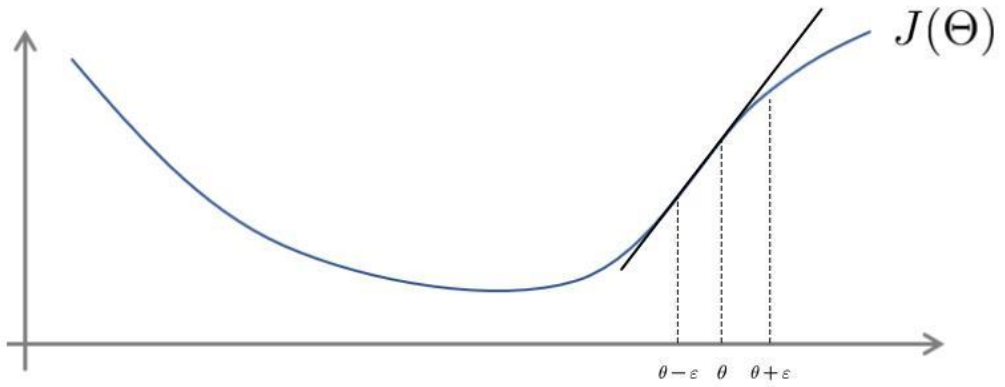


图 11. 梯度检验

根据双侧差分的方法，当ε很小的时候，θ处的导数值可以近似为

$$\frac{\mathrm{d}J(\boldsymbol{\theta})}{\mathrm{d}\theta}=\frac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon}$$

我们扩展到$\boldsymbol{\theta} \in \mathbb{R}^n$的情况，可以得到

$$\frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) = \frac{J(\theta_1 + \varepsilon, \theta_2, \theta_3, ..., \theta_n) - J(\theta_1 - \varepsilon, \theta_2, \theta_3, ..., \theta_n)}{2\varepsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\boldsymbol{\theta}) = \frac{J(\theta_1, \theta_2 + \varepsilon, \theta_3, ..., \theta_n) - J(\theta_1, \theta_2 - \varepsilon, \theta_3, ..., \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) = \frac{J(\theta_1, \theta_2, \theta_3, ..., \theta_n + \varepsilon) - J(\theta_1, \theta_2, \theta_3, ..., \theta_n - \varepsilon)}{2\varepsilon}$$

求得了偏导数之后，我们需要和之前计算的矩阵$\boldsymbol{D}$作比较，如果差别不大则可以认为梯度检验通过。梯度检验通过之后可以说名逆传播是正确的。

那么在最后，我们要解决参数初始化的问题，也就是我们的$\boldsymbol{\theta}$矩阵的初始值取多少。

尽管在 Logistic 回归中我们将参数全部初始化为 0，但是在神经网络中，如果参数全部初始化为 0，那么中间隐藏层单元输入的数值全部相等，相当于只学习一种特征，这是一种高度冗余的现象。所以我们应该换一种参数初始化的方式，这里面我们引入**随机初始化**的方法。

随机初始化：对于每一个$\theta_{ij}^{(l)}$，我们将其赋值为$[-\varepsilon, \varepsilon]$之间的一个随机值，当然这里面的$\varepsilon$与梯度检验中的$\varepsilon$没有任何关系，只是用同样的记号来进行表示。而所有的$\theta_{ij}^{(l)}$也是均匀地取该区间内得随机值。关于ε的确定，一个有效的方法就是根据神经网络中各层的节点个数来确定ε的值。我们有

$$\varepsilon_{init} = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$$

其中L_{in}和L_{out}为$\boldsymbol{\theta}^{(l)}$相邻网络层的节点个数。

1.4 总结

根据前面的描述，我们总结训练神经网络步骤如下：

1） 随机初始化参数$\theta_{ij}^{(l)}$；

2） 用前馈的方法，针对训练集$\boldsymbol{x}^{(i)}$求得输出$h_\theta(\boldsymbol{x}^{(i)})$；

3） 计算代价函数$J(\boldsymbol{\theta})$；

4） 通过逆传播计算梯度矩阵$\boldsymbol{D}$；

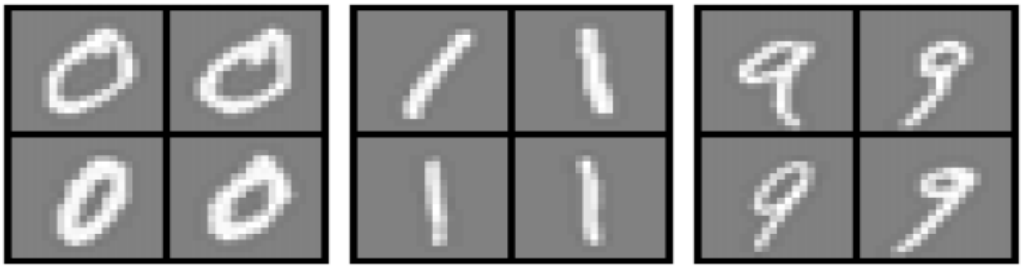
5） 梯度检验，如果数值计算方法得出的梯度和逆传播计算的梯度近似，那么梯度检验通过。梯度检验通过之后需要关闭梯度检验，因为它比较耗时，检验通过之后无需再进行检验；

6） 用梯度下降法进行优化，使代价函数$J(\boldsymbol{\theta})$达到局部最低点或全局最低点。

2. 神经网络应用—手写数字识别

在这个项目中，我们得到一组训练集。在这个训练集中，一个是输入的图片，它们被保存成矩阵的形式，并且将该矩阵按列展开形成一个行向量。这里面的每一个训练样本都是一个20×20的图片，所以展开之后每一个图片都是一个1×400的行向量。我们共有5000个训练样本，也就是5000行这样的向量，它们每一个行向量形成的图片都显示了阿拉伯数字0-9.

同样地，对于每一个图片，训练集中都给出了“正确答案”，也就是告诉神经网络这张图片代表哪一个阿拉伯数字。每一个阿拉伯数字都有500个训练样本，一共就是500×10＝5000个结果。例如，第1-500个就是代表了数字0；第501-1000个就是数字1；…；第4501-5000个就是数字9，如图12所示。而在“正确答案”中，第1-500个被标记为10，第501-1000个被标记为1，…，第4501-5000个被标记为9.



(a) digit 0　(b) digit 1　(c) digit 9

图 12. 不同数字的可视化

在这个问题中，我们采用三层网络结构，即输入层、中间隐藏层以及输出层。所以我们要想设置参数，就要设置$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$。这里面我们依然选择随机初始化的方法。在随机初始化之前，我们先得到了一组数据$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$，用来验证我们的算法实现是否正确。

我们先使用给定的数据$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$来进行前馈。

在进行前馈之前，我们先来分析一下我们的网络结构。输入层都有20×20＝400个单元，所以第一层输入有400个，加上偏置项的话就应该是401个。

关于中间隐藏层神经元的个数，Hornik et al证明，只需一个包含足够多神经元的隐藏层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数。然而如何设置隐藏层神经元的个数仍然是一个悬而未决的问题，在实际应用中我们一般使用“试错法”(trial-by-error)来进行调整。这里面我们设置中间隐藏层神经元个数为25，加上偏置项就是26个。

关于输出层单元个数，很明显这是一个多分类问题，我们想要进行的分类就是数字识别0-9，也就是分为10个类，所以根据之前的描述，我们输出层单元个数就为10，也可以说成是一个10维列向量。

网络结构分析完毕之后，关于权重$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$的维度也可以确定，$\boldsymbol{\theta}^{(1)}$为25×401大小的矩阵，$\boldsymbol{\theta}^{(2)}$为10×26大小的矩阵。然后运用我们之前提及的前馈的方法，得到一个输出的结果$h_\theta(x)$，它应该是一个10维列向量。代码以及相应的矩阵计算如图13所示。

```
1  X = [ones(size(X,1),1) X];   %5000*401    a^(1) = x
2  z2 = X * Theta1';   %5000*25   z^(2) = θ^(1) a^(1)
3
4  a2 = sigmoid(z2);   %a2 = g(z2),5000*25 a^(2) = g(z^(2))
5
6  a2 = [ones(size(a2,1),1) a2];   %add bias
7
8  z3 = a2 * Theta2';   %5000*10 z^(3) = θ^(2) a^(2)
9
10 h = sigmoid(z3);   %h_theta,5000*10 a^(3) = g(z^(3))
```

图 13. 网络前馈

我们先来明确一下矩阵的维度。事实上，$\boldsymbol{X}$ 矩阵每一行都代表输入加上偏置项的 401 个输入，样本容量为 5000，所以是一个5000×401 的矩阵。然后，经过$\boldsymbol{\theta}^{(1)}$ 的过渡之后来到了隐藏层，这里实际上是 25 个神经元（不算偏置项），所以$\boldsymbol{z}^{(2)}$ 是5000×25 的矩阵，而$\boldsymbol{a}^{(2)}$ 是加上偏置项之后的5000×26 的矩阵。最后经过$\boldsymbol{\theta}^{(2)}$ 的过渡之后，最后一层应该有 10 个神经元，所以$\boldsymbol{z}^{(3)}$ 和$\boldsymbol{a}^{(3)}$ 都是5000×10 的矩阵。

最后得到的h 实际上就是网络输出结果。接下来，我们根据之前的步骤，该要计算代价函数的值了。代价函数结合之前的公式

$$J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left(-y_k^{(i)}\ln\left(h_\theta(x^{(i)})\right)_k+(1-y_k^{(i)})\ln\left(1-\left(h_\theta(x^{(i)})\right)_k\right)\right)+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\theta_{ji}^{(l)})^2$$

来进行计算。注意，这里面我们需要设置所谓“理想输出”，也就是如图 6 所示的神经网络对于多分类问题的目标输出

$$\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix},\begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix},\begin{bmatrix}0\\0\\1\\\vdots\\0\end{bmatrix},\ldots,\begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$$

这里我们采用一种设置方式，我们特别注意到给出的$\boldsymbol{y}$ 中，数字“0”的标记是 10，所以我们就设置数字“0”的标准输出为

$$\begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}_{10\times1}$$

也就是最后一位置 1 其余都置 0.类似地，数字“1”设置列向量第一个数字为 1 其余为 0，数字“2”设置列向量第二个数字为 1 其余为 0，依此类推，代码如图 14 所示。

```
y_recoded = zeros(size(X,1),num_labels);

for i = 1:size(X,1)
   y_recoded(i,y(i,1)) = 1;
end
```

图 14. 设置标准输出

设置好“标准输出”之后我们将这些向量代入到$J(\boldsymbol{\theta})$ 中，这里$K=10$ 而$m=5000$。不带有正则化的代价函数的计算方法如图 15 所示。

```
sum = 0;
for i = 1:size(X,1)
    res1 = y_recoded(i,:) * log(h(i,:)');
    res1 = -res1;
    res2 = (1 - y_recoded(i,:)) * log(1 - h(i,:)');
    res2 = -res2;

    sum = sum + res1 + res2;
end

J = sum/size(X,1);
```

图 15. 计算代价函数（非正则化）

然后我们计算一下后面的正则项，由于是三层网络，所以我们需要计算两次，分别是$\boldsymbol{\theta}^{(1)}$中各项的和以及$\boldsymbol{\theta}^{(2)}$中各项的和。代码如图 16 所示。

```
sum = 0;
for i = 1:size(Theta1,1)
    for j = 2:size(Theta1,2)
        sum = sum + Theta1(i,j) * Theta1(i,j);
    end
end
for i = 1:size(Theta2,1)
    for j = 2:size(Theta2,2)
        sum = sum + Theta2(i,j) * Theta2(i,j);
    end
end
sum = sum * (lambda/(2*m));

J = J + sum;
```

图 16. 计算代价函数正则项

这部分代码完成之后，我们在主函数中运行这部分代码来计算损失，结果如图 17 所示，我们可以认为我们计算的结果是正确的。

```
Feedforward Using Neural Network ...
Cost at parameters (loaded from ex4weights): 0.287629
(this value should be about 0.287629)

Program paused. Press enter to continue.

Checking Cost Function (w/ Regularization) ...
Cost at parameters (loaded from ex4weights): 0.383770
(this value should be about 0.383770)
Program paused. Press enter to continue.
```

图 17. 代价函数计算正确

接下来我们需要进行逆传播来校正参数。在此之前，由于逆传播的误差计算需要使用 sigmoid 函数的导数，再结合 sigmoid 函数的导数的特殊性质，我们先来编写 sigmoid 函数的导数的代码，如图 18 所示。

```
g = sigmoid(z).*(1-sigmoid(z));
```

图 18. 计算 sigmoid 函数的导数

检查以下这个函数，以给出的样例进行测试，结果如图 19 所示。

```
Evaluating sigmoid gradient...
Sigmoid gradient evaluated at [-1 -0.5 0 0.5 1]:
  0.196612 0.235004 0.250000 0.235004 0.196612
```

图 19. sigmoid 函数的导数值（特殊点）

以零点为例，sigmoid 函数在 0 点取值为 $\frac{1}{2}$，所以该点的导数值为 $\frac{1}{2}\times\frac{1}{2}=\frac{1}{4}=0.25$ 与

图 19 中结果对应。此外，经过验算可以得知$g'(x)$是偶函数，图 19 所示的结果符合偶函数性质，所以结果是正确的。

接下来编写逆传播的代码。各部分的代码的目的如图 20 所示。

我们先来对各矩阵的维度来作说明。事实上矩阵维度的确定非常重要，因为有时候如果矩阵维度发生了错误会导致整个程序出现 bug。

首先，对于每一个训练样本，$\boldsymbol{\delta}^{(3)}$都代表了输出层的误差值，所以$\boldsymbol{\delta}^{(3)}$是一个 10 维列向量，当然在这里面$\boldsymbol{\delta}^{(3)}$被保存成了行向量，因为我们有 5000 个训练样本，所以$delta3$是一个5000×10的矩阵。而由于之前描述的$\boldsymbol{\theta}^{(2)}$是一个$10\times26$的矩阵，所以我们需要对它进行转置。在图 20 的第三行代码中我们对矩阵进行了截取，因为我们应该忽略偏置项的误差。而最终的$\boldsymbol{\delta}^{(2)}$就是一个$25\times5000$的矩阵，行数代表了样本容量 5000，列数 25 代表了第二层网络的神经元个数。

现在我们得到了误差矩阵$\boldsymbol{\delta}^{(2)}$和$\boldsymbol{\delta}^{(3)}$，然后计算梯度矩阵即可。

整个过程代码如图 20 所示。

```
1  delta3 = h - y_recoded;      % 5000*10  δ(3)=a(3)-y
2  delta2 = Theta2'* delta3';   % Theta2':26*10 delta3':10*5000,26*5000
3  delta2 = delta2(2:end,:);       % 25*5000
4  delta2 = delta2.*sigmoidGradient(z2)';   %(25*5000).*(25*5000)
5                                              δ(2)=(θ(2))^T δ(3).*g'(z(2))
6  Delta1 = zeros(size(Theta1));  % 25*401
7  Delta2 = zeros(size(Theta2));  % 10*26
8
9  Delta1 = Delta1 + delta2 * X; % (25*5000) * (5000*401) = 25*401
10 Delta2 = Delta2 + delta3' * a2; % (10*5000) * (5000*26) = 10*26
11                                        Δ(l):=Δ(l)+δ(l+1)(a(l))^T
12 Theta1_grad = Delta1 / m; % 25*401
13 Theta2_grad = Delta2 / m; % 10*26
```

图 20. 逆传播过程的代码

这里没有进行正则化。如果考虑正则化，我们除了偏置项不需要进行正则化之外，其余的梯度都要进行正则化。所以关于$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$我们进行截取，然后与$\boldsymbol{\Delta}$相加即可。代码如图 21 所示。

```
1  Reg1 = lambda/m.*Theta1(:,2:end);  % 25*400
2  Reg2 = lambda/m.*Theta2(:,2:end);  % 10*25
3  Reg1 = [zeros(size(Theta1,1),1),Reg1]; %j>=1
4  Reg2 = [zeros(size(Theta2,1),1),Reg2]; %j>=1
5  Theta1_grad = Theta1_grad + Reg1;
6  Theta2_grad = Theta2_grad + Reg2;
```

图 21. 梯度正则化

在前馈、逆传播的代码完成之后，我们就可以进行网络的训练了。按照之前所说我们需要进行参数的初始化。参数初始化我们默认以下式来进行初始化。

$$\varepsilon_{init}=\frac{\sqrt{6}}{\sqrt{L_{in}+L_{out}}}$$

此外，我们要注意，输入层中往往添加一个偏置项，所以输入层单元个数我们要在原来的个数的基础上加一。代码如图 22 所示。

```
E_init = sqrt(6/(L_in + L_out));

W = rand(L_out,1 + L_in) .* (2 * E_init) - E_init;
```

图 22. 参数初始化

我们先来计算一下梯度。先按照数值方法来计算$J(\boldsymbol{\theta})$的梯度，我们以$\varepsilon=10^{-4}$来进行双侧差分得到数值梯度。

为了得到梯度，我们使用 MATLAB 的一种特殊用法，即句柄。这里由于在前面的程序中我们已经编写了计算代价函数的程序，所以这里面我们直接调用即可。我们的代价函数计算在$nnCostFunction$中，所以我们通过句柄的方式调用这个函数，第 2 行中的p代表接受输入。这里相当于我们就可以直接得到$J(\boldsymbol{\theta})$的值。调用方法如图 23 所示。

然后我们需要完成数值方法计算梯度的代码。事实上数值方法计算梯度就相当于计算函数的斜率。注意，如果是对θ_1来求偏导数，我们就设置一个列向量，在θ_1对应的位置置ε，其余位置置 0，然后$\boldsymbol{\theta}$矩阵对这个列向量加减即可，相当于对应位置来计算斜率。

```
% Short hand for cost function
costFunc = @(p) nnCostFunction(p, input_layer_size, ...
                               hidden_layer_size, ...
                               num_labels, X, y, lambda);

[cost, grad] = costFunc(nn_params);
numgrad = computeNumericalGradient(costFunc, nn_params);
```

图 23. 句柄调用

以对θ_1的偏导数为例，计算方法为

$$\frac{\partial}{\partial\theta_1}J(\boldsymbol{\theta})=\frac{J\left(\begin{bmatrix}\theta_1\\\theta_2\\\vdots\\\theta_n\end{bmatrix}+\begin{bmatrix}\varepsilon\\0\\\vdots\\0\end{bmatrix}\right)-J\left(\begin{bmatrix}\theta_1\\\theta_2\\\vdots\\\theta_n\end{bmatrix}-\begin{bmatrix}\varepsilon\\0\\\vdots\\0\end{bmatrix}\right)}{2\varepsilon}$$

按照这个方法计算偏导数的代码如图 24 所示。

```
numgrad = zeros(size(theta));
perturb = zeros(size(theta));
e = 1e-4;
for p = 1:numel(theta)
    % Set perturbation vector
    perturb(p) = e;
    loss1 = J(theta - perturb);
    loss2 = J(theta + perturb);
    % Compute Numerical Gradient
    numgrad(p) = (loss2 - loss1) / (2*e);
    perturb(p) = 0;
end
```

图 24. 数值方法计算偏导数

我们需要特别注意，对于神经元个数比较多的网络，如果按照我们的方法来进行梯度检验，那么将会相当耗时。所以我们要采用一种更加简便的方式来缩短时间。一种策略就是用更简单的网络来代替我们的神经网络来做梯度检验。事实上，梯度检验的目的是检验逆传播是否准确。对于我们写好的逆传播，在任何网络上工作都是一样的。所以我们可以使用一种结构更加简单的网络来代替检验。这里面我们使用输入个数为 3，隐藏层个数为 5，输出层神经元个数为 3 的简单网络来测试我们的梯度。遂于替代的网络，要计算的梯度的个数就是$(3+1)\times 5+(5+1)\times 3=38$个，数量相对较少，耗时更短。

现在，我们来对比两种方法计算的梯度之间的差距。对于逆传播，计算出来的结果保存在图 23 所示的第 6 行的$grad$中。然后将图 23 中的$numgrad$和$grad$分别打印出来进行对比，如图 25(a)所示。然后计算相对误差，可以得到结果为2.5015×10^{-11}，这个值已经低于10^{-9}，梯度检验通过。

而对于正则化的梯度检验也是类似的。比较数值方法计算的结果与逆传播方法计算出来的结果，如图 25(b)所示。经过计算，相对误差为2.34649×10^{-11}，低于10^{-9}，所以正则化之后的梯度检验通过。

仔细对比图 25(a)和图 25(b)可以发现，图 25(b)中的数值更大一些，这正是因为添加了正则项。进一步分析可以得到，1-5 行二者是相同的，因为这一部分是第一层的偏置项与第二层之间的权重共 5 个，没有进行正则化；21-23 行二者也是相同的，因为这一部分是第二层的偏置项与第三层之间的权重共 3 个，也没有进行正则化。

1	-0.009278252348643	-0.009278252357989
2	0.008899119587902	0.008899119595670
3	-0.008360107610628	-0.008360107617465
4	0.007628135503257	0.007628135510830
5	-0.006747983696265	-0.006747983700319
6	-0.000003049791530	-0.000003049789135
7	0.000014286942740	0.000014286944253
8	-0.000025938307058	-0.000025938310002
9	0.000036988323515	0.000036988323443
10	-0.000046875978654	-0.000046875976885
11	-0.000175060084207	-0.000175060082341
12	0.000233146359996	0.000233146356525
13	-0.000287468728821	-0.000287468729345
14	0.000335320347045	0.000335320347216
15	-0.000376215583131	-0.000376215586743
16	-0.000096266059568	-0.000096266061953
17	0.000117982665859	0.000117982665796
18	-0.000137149704926	-0.000137149706034
19	0.000153247077250	0.000153247081600
20	-0.000166560294446	-0.000166560294197
21	0.314544970041464	0.314544970053852
22	0.111056588207870	0.111056588217144
23	0.097400696970062	0.097400696964439
24	0.164090818794982	0.164090818795005
25	0.057573649347997	0.057573649349412
26	0.050457585483166	0.050457585486229
27	0.164567932285919	0.164567932287890
28	0.057786737850396	0.057786737849711
29	0.050753017291072	0.050753017290885
30	0.158339333882207	0.158339333883892
31	0.055923529598267	0.055923529600981
32	0.049162084116983	0.049162084115012
33	0.151127527463490	0.151127527466251
34	0.053696700912376	0.053696700910282
35	0.047145624857414	0.047145624852528
36	0.149568334717465	0.149568334716819
37	0.053154205243988	0.053154205242622
38	0.046559718624994	0.046559718625937

图 25(a). 梯度检验（非正则化）

1	-0.009278252348643	-0.009278252357989
2	0.008899119587902	0.008899119595670
3	-0.008360107610628	-0.008360107617465
4	0.007628135503257	0.007628135510830
5	-0.006747983696265	-0.006747983700319
6	-0.016767979682530	-0.016767979681071
7	0.039433482865725	0.039433482867381
8	0.059335556490403	0.059335556487401
9	0.024764097437124	0.024764097437949
10	-0.032688142630466	-0.032688142630247
11	-0.060174472475971	-0.060174472475383
12	-0.031961228719180	-0.031961228723501
13	0.024922553480966	0.024922553480253
14	0.059771761686811	0.059771761688908
15	0.038641054824762	0.038641054822684
16	-0.017370465059674	-0.017370465061857
17	-0.057565866846687	-0.057565866846978
18	-0.045196384512725	-0.045196384512335
19	0.009145879655836	0.009145879661377
20	0.054610154749390	0.054610154749461
21	0.314544970041464	0.314544970053852
22	0.111056588207870	0.111056588217144
23	0.097400696970062	0.097400696964439
24	0.118682669076886	0.118682669076529
25	0.000038192866647	0.000038192869624
26	0.033692655592166	0.033692655594293
27	0.203987128208905	0.203987128211017
28	0.117148232647857	0.117148232647114
29	0.075480126404681	0.075480126405391
30	0.125698067230395	0.125698067230530
31	-0.004075882793497	-0.004075882792062
32	0.016967709037807	0.016967709034985
33	0.176337549673278	0.176337549675849
34	0.113133142254362	0.113133142251975
35	0.086162895265307	0.086162895261955
36	0.132294135717359	0.132294135716915
37	-0.004529644268558	-0.004529644270151
38	0.001500483817196	0.001500483819636

图 25(b). 梯度检验（正则化）

图 25. 梯度检验

在训练神经网络之前的最后一步，我们检验一下代价函数计算，结果如图 26 所示，结果计算正确。

```
Cost at (fixed) debugging parameters (w/ lambda = 10): 0.576051
(this value should be about 0.576051)
```

图 26. 代价函数计算

准备好所有的代码之后，就可以进行网络的训练了。我们先设置迭代次数为 50，正则化参数$\lambda=1$，用我们随机初始化的权重作为初始数据，用*fmincg*优化函数来进行优化得到一组$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$，这就是经过本次训练之后得到的神经网络权重。代码如图 27 所示。

```
% set the number of iteration
options = optimset('MaxIter', 50);

% regularization parameter
lambda = 1;

% Create "short hand" for the cost function to be minimized
costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X, y, lambda);

[nn_params, cost] = fmincg(costFunction,initial_nn_params,options);
```

图 27. 神经网络训练（lambda=1，迭代次数为 50）

当然，其中的一个实现细节就是我们还需要对第 13 行得到的*nn_params*进行方块化处理。我们在前面的*nnCostFunction*中得到的是一个列向量，是将矩阵按列展开得到的。那么这里面我们就要进行逆过程来将其方块化。在 MATLAB 中我们可以使用 reshape 函数来改变矩阵维度，由于 reshape 函数也是按列展开的，所以这个过程并不复杂。

得到了$\boldsymbol{\theta}^{(1)}$和$\boldsymbol{\theta}^{(2)}$之后就可以来进行预测了。我们拿到测试集，根据前馈的过程来得到输出，并于测试集的结果进行对比就可以得到准确率。

这里面要注意，由于我们得到的结果是一个列向量，列向量第几位是 1 就代表这个数字是多少。再 MATLAB 中，对于矩阵$\boldsymbol{A}$，$[dummy,p]=max(\boldsymbol{A},[],2)$中，*dummy*返回每一行的最大值，$p$返回这个最大值在这个行向量中的游标。对于我们的问题，每一个样本都是一个行向量，最大的数字应该是 1 或者接近 1，而这个最大值所在的位置就是我们预测的结果。代码如图 28 所示。

```
m = size(X, 1);
num_labels = size(Theta2, 1);

% You need to return the following variables correctly
p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X] * Theta1');
h2 = sigmoid([ones(m, 1) h1] * Theta2');
[dummy, p] = max(h2, [], 2);
```

图 28. 神经网络得到预测结果

在检查准确率之前，由于神经网络工作复杂，隐藏层神经元就比较神秘，所以我们可以想办法来看看隐藏层神经元的工作。为此我们需要一个输入$\boldsymbol{x}$，使得中间隐藏层的一些神经元的值为 1，这就代表某些特征被激活了。然后我们用可视化的方法来将中间隐藏层打印出来，如图 29 所示。

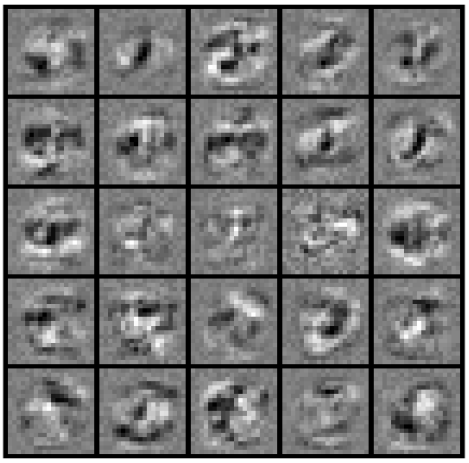


图 29. 神经网络隐藏层可视化

隐藏层神经元不算偏置单元的话就是 25 个，所以这里面展示了 25 个神经元的可视化，每一个神经元都是20×20的图片。

最后，我们通过图 27 所示的代码，得到了 5000 个样本中每一个样本的预测值，然后与原来的$\boldsymbol{y}$列向量进行比对，得到准确率。通过计算，可以本次准确率为 95.76%，如图 30 所示。需要注意的是，这个数值并不是固定的，每一次运行结果都略有不同，因为神经网络每次的训练都是不同的，这里的原因是由于参数随机初始化，每次初始参数都是不同的，这就导致后面的$h_\theta(\boldsymbol{x})$，$\boldsymbol{\delta}$以及$\boldsymbol{\Delta}$的不同，随之最终的$\boldsymbol{\theta}$也不会相同，最终导致预测结果存在偏差。

```
Training Set Accuracy: 95.760000
```

图 30. 神经网络训练结果（lambda=1，迭代次数为 50）

当然，正则化参数和迭代次数也会影响网络的预测能力。从正则化参数来说，它可以影响网络的拟合能力；迭代次数可以影响网络是否训练充分。例如，当迭代次数为 50 保持不变，$\lambda=0$的时候，训练结果为 96.5%；$\lambda=100$的时候训练结果为 83.84%，很明显地出现了欠拟合现象。当正则化参数$\lambda=1$时，迭代次数为 20 的时候，训练结果为 90.66%，准确率降低；迭代次数为 100 时，训练结果为 98.10%；迭代次数为 300 时，训练结果更是达到了 99.42%。而当$\lambda=0$（过拟合）的时候，迭代次数设置为 300，就可以达到 100%的结果。但是此时发生过拟合现象，只能说网络对于给定数据集工作良好，但是预测能力肯定不佳。

综上所述，正则化参数和迭代次数都明显影响了网络的预测能力。

四、 实验总结与体会

本次实验实现了神经网络来实现多分类问题（手写数字识别）。

本次实验是机器学习的第二次实验，开始对神经网络有了一个初步的印象。神经网络是解决多分类问题的一个重要方法，尽管原理复杂、实现起来难度比较高，但是从预测结果上看，确实是一个预测能力比较强的算法，这也是神经网络在人工智能中应用比较多的一个重要原因。事实上，对于神经网络的一些具体的数学原理还需要进一步的理解与熟悉，但是从大体上本次实验还是对神经网络有了一个初步的认知。

指导教师批阅意见：

成绩评定：

指导教师签字：　贾森 徐萌

2020 年 10 月　日

备注：