实验一：基于 DNN 的手写体识别

一、实验简介

1.1 实验目的

通过实验了解深层神经网络（DNN）模型的结构以及前馈传播和反向传播训练过程，并使用其对 MNIST 数据集进行分类。

1.2 数据集介绍

数据集	类型	类别数量	图片数量	图片尺寸
MNIST	手写体数字	10	4000	20×20

此次实验使用的是已经经过处理后的 MNIST 数据集，他们被储存在 MATLAB 的矩阵中，ex1data.mat 包含了 4000 个用于训练的示例，每一行代表一个手写数字图像的训练实例，每一个训练示例是由 20*20 的像素点组成，每个像素点将由一个浮点数表示此点的灰度强度。此外，这个 20*20 的像素图像被展开成 400 维向量，因此一个训练实例可用一个 400 维的向量表示，.mat 表示数据已经被储存为矩阵格式。可以在 ex1_fp.m 程序中通过 load 命令将这些数据读取到程序中，加载后，我们将会在变量中看到这些矩阵，代码如下所示：

```
fprintf('Loading and Visualizing Data ...\n')
load('ex1data.mat');
m = size(X, 1);
```

每一个训练示例都变成了 X 中的一行，因此，X 是一个 4000*400 大小的矩阵：

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix} \tag{1.1}$$

训练集的第二部分是包含训练集的标签的 4000*1 的向量 y，为了更好地进

行 MATLAB 索引，数字“0”会被标记为 10，数字 1-9 将会被按照原本的顺序标记成 1-9。

1.3 实验环境

MATLAB

包含的代码文件：

ex1_fp.m 神经网络前馈传播主程序

ex1_bp.m 神经网络反向传播主程序

ex1data.mat 手写数字的训练集

ex1weights.mat 初始神经网络参数

displayData.m 实现数据集可视化的函数

fmincg.m 函数最小化程序

sigmoid.m 激活函数

computerNumericalGradient.m 梯度的数值计算

checkNNGradients.m 帮助检查梯度计算正确与否

debugInitializeWeights.m 初始化权重

□ sidmoidGradient.m 计算激活函数的导数

□ randInitializeWeights.m 随机初始化权重

□ nnCostFunction.m 神经网络成本函数

□ predict.m-神经网络预测函数

□ 标志着代码是待完成的

二．神经网络的前馈传播与预测

本部分的练习中，将构建一个前馈传播神经网络模型来识别手写数字。神经网络模型可以形成非线性假设的复杂模型来进行预测。完成本部分练习将实现：使用已经训练完成的神经网络参数，实现前馈传播并根据训练好的模型进行数字预测。

2.1 数据集可视化

本步骤将实现参数的载入以及上述数据集的可视化。首先随机挑选 X 矩阵中的任意 100 行数据也就是 100 个训练实例，并调用 displayData 函数将这些数据可视化，该函数将每一行映射为原来的 20*20 像素的灰度图像，并将这些图像一起显示。ex1_fp.m 的主程序中对应代码如下所示：

```
input_layer_size  = 400;
hidden_layer_size = 25;
num_labels = 10;
fprintf('Loading and Visualizing Data ...\n')
load('ex1data.mat');
m = size(X, 1);
sel = randperm(size(X, 1));
sel = sel(1:100);
displayData(X(sel, :));
fprintf('Program paused. Press enter to continue.\n');
pause;
```

其中，displayData.m 函数对应代码如下：

```
function [h, display_array] = displayData(X, example_width)
if ~exist('example_width', 'var') || isempty(example_width)
      example_width = round(sqrt(size(X, 2)));
end
colormap(gray);
[m n] = size(X);
example_height = (n / example_width);
display_rows = floor(sqrt(m));
display_cols = ceil(m / display_rows);
pad = 1;
display_array = - ones(pad + display_rows * (example_height + pad), ...
                       pad + display_cols * (example_width + pad));
```

```
curr_ex = 1;
for j = 1:display_rows
    for i = 1:display_cols
        if curr_ex > m,
            break;
        end
        max_val = max(abs(X(curr_ex, :)));
        display_array(pad + (j - 1) * (example_height + pad) + (1:example_height), ...
                      pad + (i - 1) * (example_width + pad) + (1:example_width)) = ...
                        reshape(X(curr_ex, :), example_height, example_width) / max_val;
        curr_ex = curr_ex + 1;
    end
    if curr_ex > m,
        break;
    end
end
h = imagesc(display_array, [-1 1]);
axis image off
drawnow;
end
```

运行以上代码，可得到数据集的图像如图一所示：

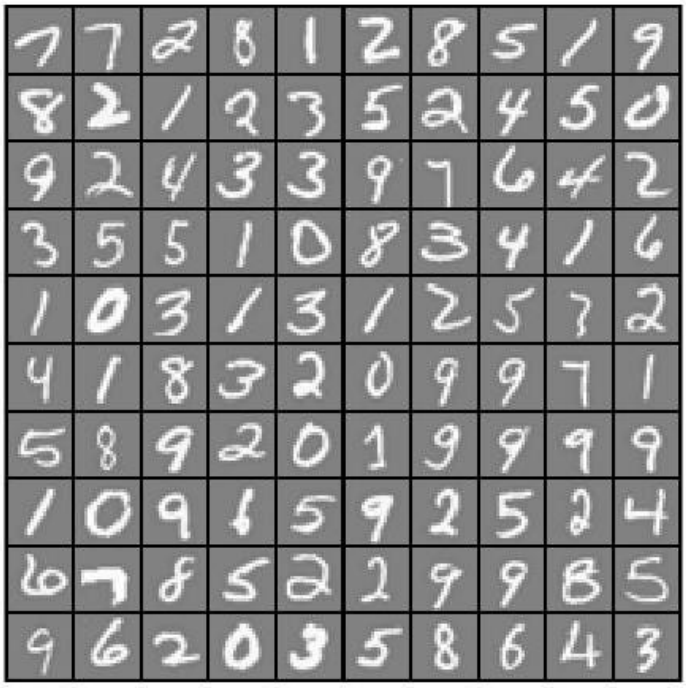

图 1 数据集训练示例

2.2 模型结构

神经网络结构如图 2 所示，由输入层，隐藏层和输出层组成。每一个圆圈代表一个神经元，一层由若干个神经元组成。层与层之间是全连接的，每一层神经元接受前一层的输入信号并与同层的参数对应相乘累加，最后经过激活函数得到该神经元的输出。

使用 load 指令载入上述的数据集以及已经训练好的参数$\left(\Theta^{(1)},\Theta^{(2)}\right)$，这些参数被存储在 ex1weights.mat 中，ex1_fp.m 将把这些参数装载到 Theta1 和 Theta2 中，作为每一层的网络参数。

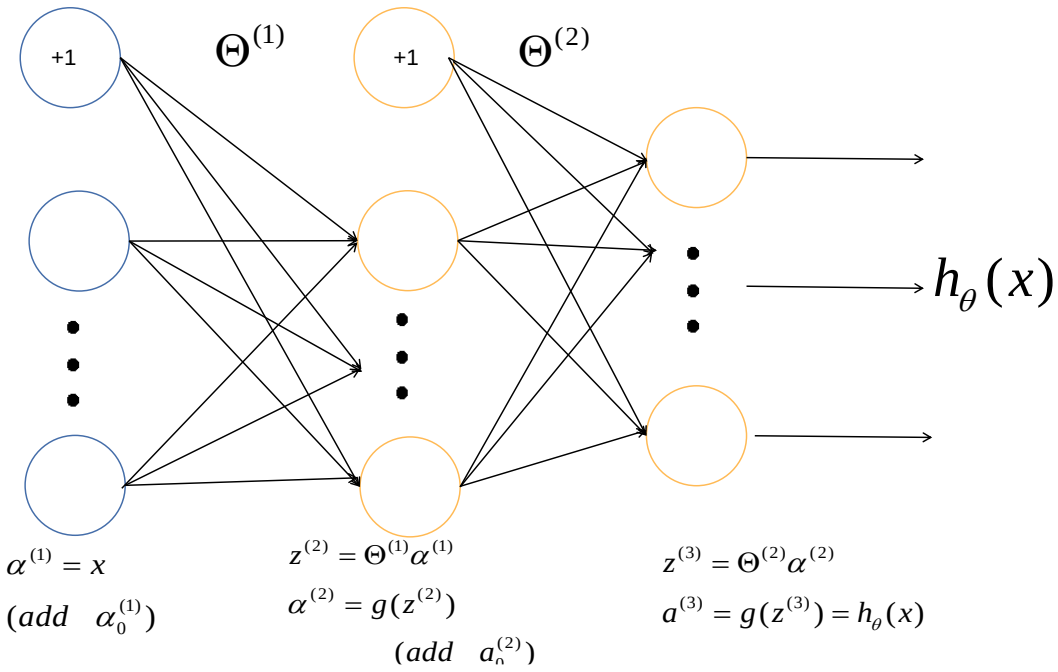

图 2 神经网络前向传播结构

此部分由 ex1_fp.m 实现，具体代码如下：

```
fprintf('\nLoading Saved Neural Network Parameters ...\n')
load('ex1weights.mat');
```

2.3 前馈传播与预测

参数加载完毕，接下来完成 predict.m 代码实现神经网络的前馈传播。

通过为每个训练实例计算$h_\theta\left(x^{(i)}\right)$并返回具有最大输出的标签$\left(h_\theta\left(x\right)\right)_k$作为预测结果。

predict.m 代码部分需要我们完成：

```
function p = predict(Theta1, Theta2, X)
m = size(X, 1);
num_labels = size(Theta2, 1);
% You need to return the following variables correctly


% ============================================
end
```

完成 predict.m 代码后，ex1_fp.m 将会调用此函数来预测输出。运行结果的准确率约为 97.5%。并且，控制台在程序启动后会输出所预测的图像。按下 ctrl+C 即可停止程序运行。

ex1_fp.m 的调用以及预测代码如下：

```
pred = predict(Theta1, Theta2, X);
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
fprintf('Program paused. Press enter to continue.\n');
pause;
rp = randperm(m);
for i = 1:10
    % Display
    fprintf('\nDisplaying Example Image\n');
    displayData(X(rp(i), :));
    pred = predict(Theta1, Theta2, X(rp(i),:));
    fprintf('\nNeural Network Prediction: %d (digit %d)\n', pred, mod(pred, 10));
    % Pause
    fprintf('Program paused. Press enter to continue.\n');
    pause;
end
```

三．神经网络与反向传播

本部分练习中，将完成对神经网络反向传播算法的学习。

3.1 数据集可视化

首先实现训练数据的可视化以及初始参数的载入，ex1_bp.m 对应代码如下：

```
clear ; close all; clc
input_layer_size  = 400;
hidden_layer_size = 25;
num_labels = 10;
% Load Training Data
fprintf('Loading and Visualizing Data ...\n')
load('ex1data.mat');
m = size(X, 1);
% Randomly select 100 data points to display
sel = randperm(size(X, 1));
sel = sel(1:100);
displayData(X(sel, :));
fprintf('Program paused. Press enter to continue.\n');
pause;
% neural network parameters.
fprintf('\nLoading Saved Neural Network Parameters ...\n')
% Load the weights into variables Theta1 and Theta2
load('ex1weights.mat');
% Unroll parameters
nn_params = [Theta1(:) ; Theta2(:)];
```

其中，displayData.m 和 ex1_fp 所调用函数相同。

3.2 前馈传播和成本函数

在此步骤中，仍将首先计算神经网络的成本函数和梯度并实现前馈传播，这将由 nnCostFunction.m 的函数代码来实现，它为每个训练实例计算损失函数并求和取平均值得到成本函数。在神经网络中，成本函数的公式如式 3.1 所示：

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)_k-\left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)_k\right] \tag{3.1}$$

$h_\theta(x^{(i)}))_k=\alpha_k^{(3)}$ 是第三层第 k 个神经元经过激活函数的输出。同时，为了更

好的训练神经网络，将输出的标签 1-10 进行编码为只包含 0 或 1 的向量，如式 3.2 所示：

$$y = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \cdots or \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix} \tag{3.2}$$

比如，如果对输入 $x^{(i)}$ 的分类结果是数字 5，那么输出 y 应该是 10 维向量，其中 y_5=1，其他元素等于 0。

nnCostFunction 的对应代码如下：

```
function [J grad] = nnCostFunction(nn_params, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, ...
                                   X, y, lambda)
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));
m = size(X, 1);
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));
% ====================== YOUR CODE HERE ======================

%convert labels to vectors

%unregularized cost function

end
```

完成这一步之后，ex1_bp.m 将会调用我们所编写的 nnCostFunction 并使用之前存储的神经网络参数 Theta1 和 Theta2，运行后，我们将看到，cost 大概是 0.288401。ex1_bp.m 调用函数代码如下：

```
fprintf('\nFeedforward Using Neural Network ...\n')
lambda = 0;
J = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, ...
                   num_labels, X, y, lambda);
fprintf(['Cost at parameters (loaded from ex1weights): %f '...
```

```
        '\n(this value should be about 0.288401)\n'], J);
fprintf('\nProgram paused. Press enter to continue.\n');
pause;
```

3.2.1 正则化成本函数

和多元分类实验相同，神经网络中的成本函数也需要正则化，如式 3.3 所示：

$$J(\theta)=\frac{1}{\mathrm{m}}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)_k-\left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)_k\right]+$$
$$\frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}\left(\Theta_{j,k}^{(1)}\right)^2+\sum_{j=1}^{10}\sum_{k=1}^{25}\left(\Theta_{j,k}^{(2)}\right)^2\right] \tag{3.3}$$

在这里，神经网络被假设为只有三层：即一个输入层，一个隐藏层和一个输出层。同时，应该注意正则项的添加不包括偏差项，也就是 theta1 和 theta2 的第一列。

了解了正则项的构成后，就可以向之前编写好的 nnCostFunction.m 里的非正则化成本函数添加正则项。需要我们完成：

```
%bias terms are not regularized


%add regularization term
```

当完成以上部分后，我们就可以在 ex1_bp 主程序使用加载好的 Theta1 和 Theta2 参数并调用 nnCostFunction 计算正则化后的成本函数，运行后，我们将会看到此神经网络的成本大约是 0.408577。ex1_bp.m 调用代码如下：

```
fprintf('\nChecking Cost Function (w/ Regularization) ... \n')
lambda = 1;
J = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, ...
                   num_labels, X, y, lambda);
fprintf(['Cost at parameters (loaded from ex1weights): %f '...
         '\n(this value should be about 0.408577)\n'], J);
fprintf('Program paused. Press enter to continue.\n');
pause;
```

3.3 反向传播

在这部分练习，我们将实现反向传播算法来计算神经网络代价函数的梯度，在上述的正向传播过程中，输入数据利用已经训练好的神经网络参数经过输入层，隐含层，逐层处理最后经输出层输出，而如果在输出层得不到准确的输出值，则需要优化神经网络的参数以达到期望的输出，调整神经参数的过程就是反向传播过程，从输出层逐层求出目标函数对于神经网络参数也就是权值的偏导数，并以此作为修改权值的依据，通过优化器不断修改权值使得成本函数取得最小值，最终得到使得分类准确率最高的神经网络参数，这就是神经网络的学习过程。

具体做法：我们需要完成 nnCostFunction.m 计算成本函数对于参数的梯度，并返回合适的梯度值。之后，我们就能通过使用 fmincg 来最小化 $J(\Theta)$ 从而达到训练神经网络的目的。

3.3.1 计算激活函数的梯度

激活函数是非线性函数，其使用使得深层神经网络的表达能力更加强大，如若不使用激活函数，那么神经网络等同于线性回归。在此神经网络中，使用的激活函数是 sigmoid 函数，由式 3.4 所示，由于在利用求导链式法则计算成本函数对于参数的偏导时，会用到激活函数的导数，因此，我们需要先计算激活函数的梯度并返回。

$$g(z)=\frac{1}{1+e^{-z}} \tag{3.4}$$

故此激活函数的导数可求得为：

$$g'(z)=\frac{d}{dz}g(z)=g(z)(1-g(z)) \tag{3.5}$$

代码实现如下：

```
function d=sigmoidGradient(z)
g = zeros(size(z))
% ====================== YOUR CODE HERE ======================


end
```

```
function g = sigmoid(z)
```

```
g = 1.0 ./ (1.0 + exp(-z));
end
```

在 ex1_bp.m 中调用 sigmoidGradient 计算导数，代码如下：

```
fprintf('\nEvaluating sigmoid gradient...\n')

g = sigmoidGradient([1 -0.5 0 0.5 1]);
fprintf('Sigmoid gradient evaluated at [1 -0.5 0 0.5 1]:\n  ');
fprintf('%f ', g);
fprintf('\n\n');

fprintf('Program paused. Press enter to continue.\n');
pause;
```

3.3.2 初始化参数

要开始训练神经网络，我们首先要随机初始化网络参数，输入参数的规范化使得执行梯度下降算法进行优化时可以避免过多的迭代。一种有效初始化参数的方法是在[$-\varepsilon_{init}$，ε_{init}]的范围内随机选择数值作为神经网络的参数$\Theta^{(l)}$，初始化权重使其规定在正确的范围内可能可以避免梯度消失和爆炸问题，在这里，ε_{init}=0.12，这让选择参数的范围变小从而使得训练学习的过程更加高效。

此部分代码 randInitializeWeights.m 需要我们完成：

```
function W = randInitializeWeights(L_in, L_out)
W = zeros(L_out, 1 + L_in);
% Instructions: Initialize W randomly so that we break the symmetry while
%               training the neural network.
%
% Note: The first row of W corresponds to the parameters for the bias units

% Randomly initialize the weights to small values
end
```

在 ex1_bp.m 中调用这一函数，代码如下：

```
fprintf('\nInitializing Neural Network Parameters ...\n')
initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];
```

3.3.3 后向传播

这一步用于神经网络参数的优化，通过更新 theta1 和 theta2，使得成本函数取得最小值。因此，在这一步骤中，我们需要在 nnCostFuncTion 中继续计算成本函数对于每一层神经网络参数的梯度，并用此结果来更新各层的神经网络参数，从而最小化成本函数。后向传播的更新参数过程如图 3 所示：

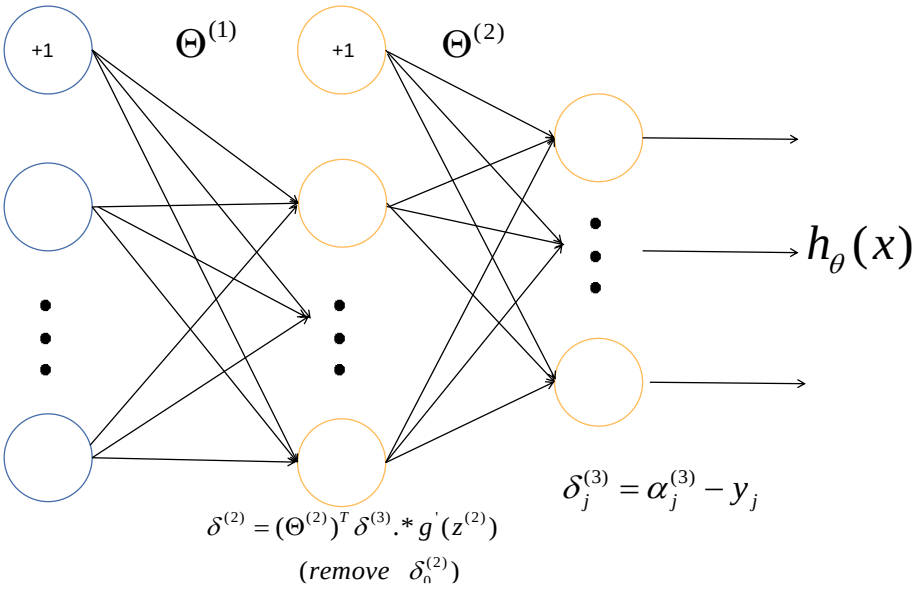

图 3 后向传播参数的更新过程

由于后向传播需要用到神经网络各层的参数以及输出结果，因此，我们将首先进行神经网络的前向传播计算整个网络的参数包括各层神经元的输出值，并储存在内存中。完成此步后，利用导数的链式求导法则逐层求解成本函数对于参数的偏导，并利用梯度下降法，完成参数的更新，具体步骤如下：

1. 首先，为第 t 个训练实例 $x^{(t)}$ 计算$\left(\alpha^{(1)}\right)$，并执行前向传播，如图 3 所示，依次计算第二层和第三层的$\left(z^{(2)},\alpha^{(2)},z^{(3)},\alpha^{(3)}\right)$

2. 由于第三层的神经网络参数和成本函数联系更加紧密，因此从第三层的导数求导开始。对于第三层的每个神经元的输出，利用导数的链式求导法则计算成本函数对于 w[3]的偏导，并使：

$$\delta_k^{(3)} = \left(\alpha_k^{(3)} - y_k\right) \tag{3.6}$$

$\alpha_k^{(3)}$ 是经过激活函数的输出， y_k 是真实的标签向量。

3. 对于第 2 层，使得：

$$\delta^{(2)}=\left(\Theta^{(2)}\right)^{T}\delta^{(3)}.*g'(z^{(2)}) \tag{3.7}$$

4. 使用式 3.8 计算每个层积累的梯度，需要注意的是，我们应该跳过 $\delta_0^{(2)}$，在 MATLAB 中，对应着 delta2=delta2(2:end)。

$$\Delta^{(l)}=\Delta^{(l)}+\delta^{(l+1)}(\alpha^{(l)})^{T} \tag{3.8}$$

5. 将上步计算得到的累积梯度除以 m，m 是训练实例的个数，从而得到此神经网络的代价函数的（非正则化）梯度。

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)=D_{ij}^{(l)}=\frac{1}{m}\Delta_{ij}^{(l)} \tag{3.9}$$

此部分代码将添加在 nnCostFunction 中，代码需要我们完成：

```
%unregularized gradient
```

在实现了反向传播算法之后，ex1_bp.m 将调用梯度检查算法，梯度检查算法是将我们计算得到的梯度和理论计算的梯度相比较，梯度检查将增加我们的信心，调用代码如下：

```
fprintf('\nChecking Backpropagation... \n');
checkNNGradients;
fprintf('\nProgram paused. Press enter to continue.\n');
pause;
```

3.3.4 梯度检查

在此神经网络中，我们需要最小化代价函数 $J(\Theta)$，为了更好的对参数进行梯度检查，我们可以将 $\Theta^{(1)}$，$\Theta^{(2)}$ 展开成一个长向量，从而，成本函数变成了 $J(\theta)$，并使用如下的梯度检查程序检查梯度计算的正确与否。

假设 现在有函数 $f_i(\theta)$ 可以计算 $\frac{\partial J(\theta)}{\partial\theta_i}$，现在需要检查 $f_i(\theta)$ 计算的导数是否正确，以第 i 个元素为例：

$$\theta^{(i+)} = \theta + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \varepsilon \\ \vdots \\ 0 \end{bmatrix}, \quad \theta^{(i-)} = \theta - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \varepsilon \\ \vdots \\ 0 \end{bmatrix} \tag{3.10}$$

要想检查 $f_i(\theta)$，对于每个元素 i 导数计算的正确与否，则观察：

$$f_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\varepsilon} \tag{3.11}$$

此部分将由函数 checkNNGradients.m 实现，代码具体部分如下

```
function checkNNGradients(lambda)
if ~exist('lambda', 'var') || isempty(lambda)
    lambda = 0;
end
input_layer_size = 3;
hidden_layer_size = 5;
num_labels = 3;
m = 5;
Theta1 = debugInitializeWeights(hidden_layer_size, input_layer_size);
Theta2 = debugInitializeWeights(num_labels, hidden_layer_size);
X  = debugInitializeWeights(m, input_layer_size - 1);
y  = 1 + mod(1:m, num_labels)';
nn_params = [Theta1(:) ; Theta2(:)];
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);
disp([numgrad grad]);
fprintf(['The above two columns you get should be very similar.\n' ...
         '(Left-Your Numerical Gradient, Right-Analytical Gradient)\n\n']);
diff = norm(numgrad-grad)/norm(numgrad+grad);
fprintf(['If your backpropagation implementation is correct, then \n' ...
         'the relative difference will be small (less than 1e-9). \n' ...
         '\nRelative Difference: %g\n'], diff);
end
```

其中，用到的 computerNumericalGradient.m 代码如下：

```
function numgrad = computeNumericalGradient(J, theta)
```

```
numgrad = zeros(size(theta));
perturb = zeros(size(theta));
e = 1e-4;
for p = 1:numel(theta)
      perturb(p) = e;
      loss1 = J(theta - perturb);
      loss2 = J(theta + perturb);
      numgrad(p) = (loss2 - loss1) / (2*e);
      perturb(p) = 0;
end
end
```

其中，用到的 debugInitializeWeights.m 代码如下：

```
function W = debugInitializeWeights(fan_out, fan_in)
W = zeros(fan_out, 1 + fan_in);
W = reshape(sin(1:numel(W)), size(W)) / 10;
end
```

完成此步后，ex1_bp.m 将会调用已被提供的 checkNNGradients.m 检查梯度。具体实现是它将创建一个小的神经网络去检查后向传播计算的梯度正确与否，如果反向传播是正确的，我们应该看到运行出来的两个梯度应该得到非常相似的值。

3.3.5 正则化神经网络

在成功地实现了反向传播算法之后，我们将向梯度添加正则化项。具体地说，当我们用后向传播算法计算完 $\Delta_{ij}^{(l)}$，我们应该用实现向之前计算得到的梯度增加正则项如式 3.12 所示，同时注意不应对于偏差项也就是 Θ（L）的第一列增加正则项。

$$\begin{aligned} &\frac{\partial}{\partial\Theta_{ij}^{(l)}}J\left(\Theta\right)=D_{ij}^{(l)}=\frac{1}{m}\Delta_{ij}^{(l)} && \text{for j=0} \\ &\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)=D_{ij}^{(l)}=\frac{1}{m}\Delta_{ij}^{(l)}+\frac{\lambda}{m}\Theta_{ij}^{(l)} && \text{for j}\geq 1 \end{aligned} \tag{3.12}$$

需要增加正则项的参数具体如式所示：

$$\Theta^{(l)}=\begin{bmatrix}\Theta_{1,0}^{(i)} & \Theta_{1,1}^{(l)} & \cdots \\ \Theta_{2,0}^{(i)} & \Theta_{2,1}^{(l)} & \\ \vdots & & \ddots\end{bmatrix} \tag{3.13}$$

为梯度增加正则项在 nnCostFunction.m 中添加实现，代码需要我们完成：

```
%regularize gradient


% Unroll gradients
grad = [Theta1_grad(:) ; Theta2_grad(:)];
end
```

ex1_bp.m 调用检查梯度函数并输出正则化成本函数值，代码如下：

```
fprintf('\nChecking Backpropagation (w/ Regularization) ... \n')
lambda = 3;
checkNNGradients(lambda);
debug_J  = nnCostFunction(nn_params, input_layer_size, ...
                          hidden_layer_size, num_labels, X, y, lambda);
fprintf(['\n\nCost at (fixed) debugging parameters (w/ lambda = 10): %f ' ...
         '\n(this value should be about 0.648928)\n\n'], debug_J);
fprintf('Program paused. Press enter to continue.\n');
pause;
```

3.4 训练网络更新参数

当我们成功地实现了神经网络代价函数以及梯度计算之后，练习的下一步就是利用 fmincg.m 函数作为优化算法来训练学习更新参数。目的是使得成本函数达到最小值（梯度下降法）。

fmincg.m 函数代码如下：

```
function [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
    length = options.MaxIter;
else
    length = 100;
end
RHO = 0.01;
SIG = 0.5;
INT = 0.1;
EXT = 3.0;
MAX = 20;
```

```
RATIO = 100;
argstr = ['feval(f, X'];
for i = 1:(nargin - 3)
  argstr = [argstr, ',P', int2str(i)];
end
argstr = [argstr, ')'];
if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
S=['Iteration '];
i = 0;
ls_failed = 0;
fX = [];
[f1 df1] = eval(argstr);
i = i + (length<0);
s = -df1;
d1 = -s'*s;
z1 = red/(1-d1);

while i < abs(length)
  i = i + (length>0);

  X0 = X; f0 = f1; df0 = df1;
  X = X + z1*s;
  [f2 df2] = eval(argstr);
  i = i + (length<0);
  d2 = df2'*s;
  f3 = f1; d3 = d1; z3 = -z1;
  if length>0, M = MAX; else M = min(MAX, -length-i); end
  success = 0; limit = -1;
  while 1
    while ((f2 > f1+z1*RHO*d1) || (d2 > -SIG*d1)) && (M > 0)
      limit = z1;
      if f2 > f1
        z2 = z3 - (0.5*d3*z3*z3)/(d3*z3+f2-f3);
      else
        A = 6*(f2-f3)/z3+3*(d2+d3);
        B = 3*(f3-f2)-z3*(d3+2*d2);
        z2 = (sqrt(B*B-A*d2*z3*z3)-B)/A;
      end
      if isnan(z2) || isinf(z2)
        z2 = z3/2;
      end
      z2 = max(min(z2, INT*z3),(1-INT)*z3);
      z1 = z1 + z2;
      X = X + z2*s;
```

```
        [f2 df2] = eval(argstr);
        M = M - 1; i = i + (length<0);
        d2 = df2'*s;
        z3 = z3-z2;
    end
    if f2 > f1+z1*RHO*d1 || d2 > -SIG*d1
        break;
    elseif d2 > SIG*d1
        success = 1; break;
    elseif M == 0
        break;
    end
    A = 6*(f2-f3)/z3+3*(d2+d3);
    B = 3*(f3-f2)-z3*(d3+2*d2);
    z2 = -d2*z3*z3/(B+sqrt(B*B-A*d2*z3*z3));
    if ~isreal(z2) || isnan(z2) || isinf(z2) || z2 < 0
            if limit < -0.5
            z2 = z1 * (EXT-1);
        else
            z2 = (limit-z1)/2;
        end
    elseif (limit > -0.5) && (z2+z1 > limit)
        z2 = (limit-z1)/2;
    elseif (limit < -0.5) && (z2+z1 > z1*EXT)
        z2 = z1*(EXT-1.0);
    elseif z2 < -z3*INT
        z2 = -z3*INT;
    elseif (limit > -0.5) && (z2 < (limit-z1)*(1.0-INT))
        z2 = (limit-z1)*(1.0-INT);
    end
    f3 = f2; d3 = d2; z3 = -z2;
    z1 = z1 + z2; X = X + z2*s;
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);
    d2 = df2'*s;
end
if success
    f1 = f2; fX = [fX' f1]';
    fprintf('%s %4i | Cost: %4.6e\r', S, i, f1);
    s = (df2'*df2-df1'*df2)/(df1'*df1)*s - df2;
    tmp = df1; df1 = df2; df2 = tmp;
    d2 = df1'*s;
    if d2 > 0
        s = -df1;
```

```
        d2 = -s'*s;
      end
      z1 = z1 * min(RATIO, d1/(d2-realmin));
      d1 = d2;
      ls_failed = 0;
    else
      X = X0; f1 = f0; df1 = df0;
      if ls_failed || i > abs(length)
        break;
      end
      tmp = df1; df1 = df2; df2 = tmp;
      s = -df1;
      d1 = -s'*s;
      z1 = 1/(1-d1);
      ls_failed = 1;
    end
    if exist('OCTAVE_VERSION')
      fflush(stdout);
    end
end
fprintf('\n');
```

ex1_bp.m 将调用 fmincg 函数来训练神经网络

```
fprintf('\nTraining Neural Network... \n')
options = optimset('MaxIter', 50);
lambda = 1;
costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X, y, lambda);
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));
fprintf('Program paused. Press enter to continue.\n');
pause;
```

3.5 隐藏层可视化

为了理解神经网络学习内容，我们可以通过将隐藏层捕捉到的来自前一层的信息可视化。具体方法：给定一个特定的隐藏层神经元，可一种可视化神经元计算内容的方法是找到一个输入 x,使其激活值 $a(l)$接近 1.对于此神经网络来说，则

Θ_1(1)的第 i 行应是一个 401 维的向量，此向量代表针对训练实例训练出来的参数，如果我们忽视偏差项，那我们将得到 400 维的向量，此向量代表每个输入像素到隐藏单元的权重。

Displaydata.m 将实现这一点，通过将 400 维向量重塑为 20*20 图像并显示它。代码如下（和 ex1_fp 所用代码相同）：

```
function [h, display_array] = displayData(X, example_width)
if ~exist('example_width', 'var') || isempty(example_width)
    example_width = round(sqrt(size(X, 2)));
end
colormap(gray);
[m n] = size(X);
example_height = (n / example_width);
display_rows = floor(sqrt(m));
display_cols = ceil(m / display_rows);
pad = 1;
display_array = - ones(pad + display_rows * (example_height + pad), ...
                       pad + display_cols * (example_width + pad));
curr_ex = 1;
for j = 1:display_rows
    for i = 1:display_cols
        if curr_ex > m,
            break;
        end
        max_val = max(abs(X(curr_ex, :)));
        display_array(pad + (j - 1) * (example_height + pad) + (1:example_height), ...
                      pad + (i - 1) * (example_width + pad) + (1:example_width)) = ...
                        reshape(X(curr_ex, :), example_height, example_width) / max_val;
        curr_ex = curr_ex + 1;
    end
    if curr_ex > m,
        break;
    end
end
h = imagesc(display_array, [-1 1]);
axis image off
drawnow;
end
```

ex1_bp.m 将通过调用 displaydata 函数实现对隐藏层捕捉到的信息具象化。代码如下所示：

```
fprintf('\nVisualizing Neural Network... \n')
```

```
displayData(Theta1(:, 2:end));
fprintf('\nProgram paused. Press enter to continue.\n');
pause;
```

结果如图 4 所示：

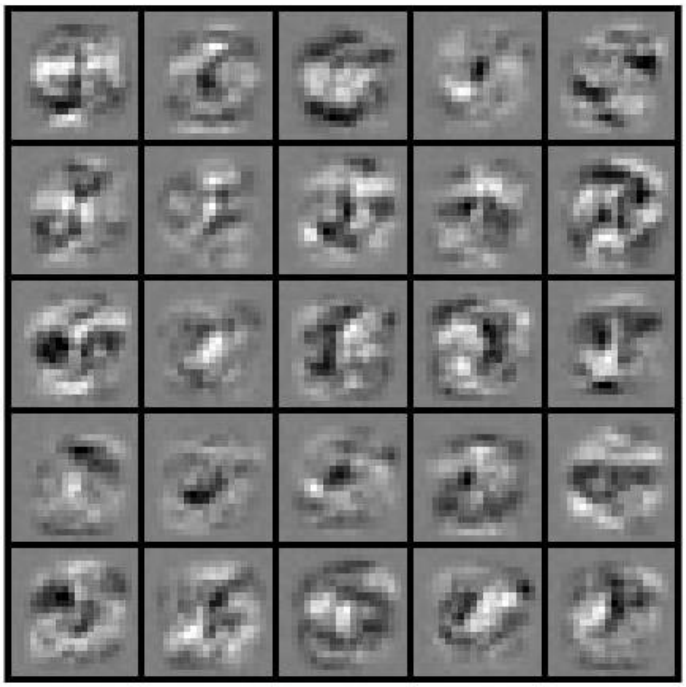

图 4 隐藏层的可视化

3.6 测试模型准确率

主程序将在训练结束后计算训练的准确率。你应该看到训练的准确性约为 95%（由于参数初始化的随机性，可能有 1%的偏差）。通过对神经网络进行更深层的迭代训练，可以获得更高的训练精度。我们可以通过改变 setMaxlterto 以及正则化参数 λ 的值，观察训练精度的变化。有了正确的学习设置，有可能使神经网络完美拟合训练集。ex1_bp.m 将会调用 predict 函数进行预测，代码如下：

```
pred = predict(Theta1, Theta2, X);
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

其中，predict 函数代码和 ex1_fp 使用代码相同。