《人工智能与深度学习》课程

实验报告

学号：04022212

姓名：钟 源

2022 年 3 月 12 日

实验一：深度神经网络应用示例与实验 DNN

一、实验目的

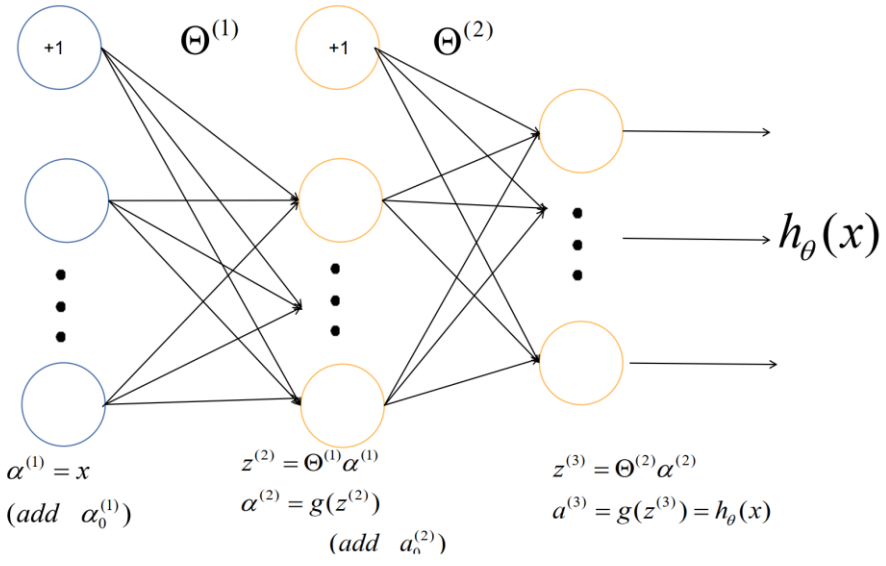
通过实验了解深层神经网络（DNN）模型的结构以及前馈传播和反向传播训练过程，并使用其对 MNIST 数据集进行分类。

二、实验内容

按照实验手册要求完成指定实验内容。

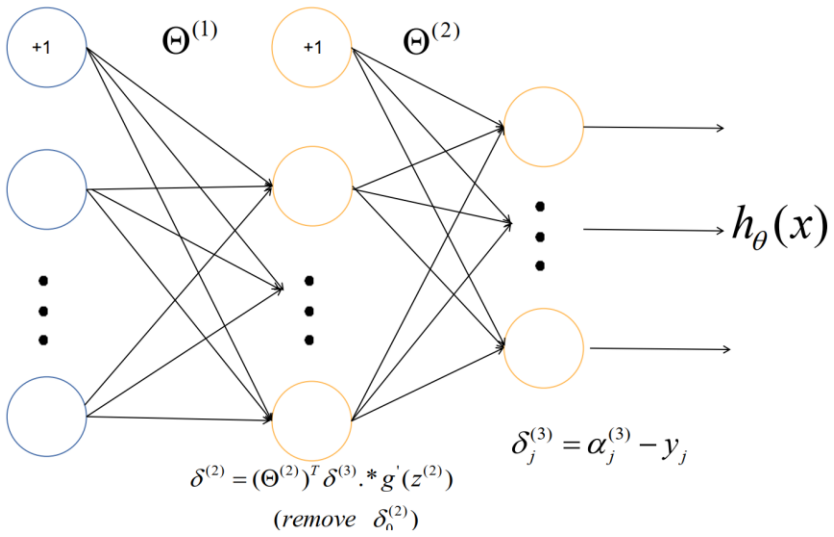
1.思路分析

根据实验手册，前向传播与预测模型示意图如下，其中第一层为输入层，第二、三层为隐藏层。



在程序中，prad 为预测结果，y 为实际结果，用 mean(double(pred == y))衡量预测准确率，预期结果为约 97.5%

反向传播与预测模型示意图如下，其中第一层为输入层，第二、三层为隐藏层。



模型在训练次数为 50 和正则化系数λ=1 时，其训练的准确性约为 95%（由于参数初始化的随机性，可能有 1%的偏差）。

2.完成代码

1）ex1_fp.m 中调用的 **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
     p = zeros(size(X, 1), 1);
     h1 = sigmoid([ones(m, 1) X] * Theta1');
     n = size(h1,1);
     h2 = sigmoid([ones(n,1) h1] * Theta2');
     [temp, p] = max(h2, [], 2);
     % =============================
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。

2）ex1_bp.m 中调用的 **nnCostFunction.m** 代码部分需要我们完成：

```
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));

% Setup some useful variables
m = size(X, 1); % 样本数量

% You need to return the following variables correctly
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));

% ====================== YOUR CODE HERE ======================
X1=X*Theta1(:,2:401).';
X1=X1+repmat(Theta1(:,1).',4000,1);
G1=X1;
X1=1./(1+exp(-X1));
sigmoid_1=X1;
sigmoid_grad=X1.*(1-X1);
X2=X1*Theta2(:,2:26).';
X2=X2+repmat(Theta2(:,1).',4000,1);
X2=1./(1+exp(-X2));
G2=X2;

%convert labels to vectors
Y1=zeros(4000,10);
for i=1:4000
    switch(y(i))
        case 1
            Y1(i,:)=[1 0 0 0 0 0 0 0 0 0];
        case 2
            Y1(i,:)=[0 1 0 0 0 0 0 0 0 0];
        case 3
            Y1(i,:)=[0 0 1 0 0 0 0 0 0 0];
        case 4
            Y1(i,:)=[0 0 0 1 0 0 0 0 0 0];
        case 5
            Y1(i,:)=[0 0 0 0 1 0 0 0 0 0];
```

```
        case 6
            Y1(i,:)=[0 0 0 0 0 1 0 0 0 0];
        case 7
            Y1(i,:)=[0 0 0 0 0 0 1 0 0 0];
        case 8
            Y1(i,:)=[0 0 0 0 0 0 0 1 0 0];
        case 9
            Y1(i,:)=[0 0 0 0 0 0 0 0 1 0];
        case 10
            Y1(i,:)=[0 0 0 0 0 0 0 0 0 1];
    end
end
%unregularized cost function
if lambda==0
for i=1:4000
    for k=1:10
        J=J-Y1(i,k)*log(X2(i,k))-(1-Y1(i,k))*log(1-X2(i,k));
    end
end

%bias terms are not regularized
J=J/4000;
%add regularization term
else
for i=1:4000
    for k=1:10

    J=J-Y1(i,k)*log(X2(i,k))-(1-Y1(i,k))*log(1-X2(i,k));
    end
end
J=J/4000;
    S=0;
    for i=1:25
        for j=1:400
            S=S+Theta1(i,j+1)*Theta1(i,j+1);
        end
    end
    for i=1:10
        for j=1:25
            S=S+Theta2(i,j+1)*Theta2(i,j+1);
        end
    end
    S=S*lambda/2/4000;
    J=J+S;
```

```
end

%unregularized gradient
delta3=X2-Y1;
delta3=delta3.';
delta2=((Theta2(:,2:26).')*delta3).*(sigmoid_grad.');
Theta2_grad(:,2:end)=Theta2_grad(:,2:end)+(delta3)*(sigmoid_1);
Theta1_grad(:,2:end)=Theta1_grad(:,2:end)+(delta2)*(X);
Theta2_grad=Theta2_grad/4000;
Theta1_grad=Theta1_grad/4000;
%regularize gradient
if lambda~=0
   Theta2_grad(:,2:end)=Theta2_grad(:,2:end)+lambda/4000*Theta2(:,2:end);
   Theta1_grad(:,2:end)=Theta1_grad(:,2:end)+lambda/4000*Theta1(:,2:end);
end
%=========================================================================
% Unroll gradients
grad = [Theta1_grad(:) ; Theta2_grad(:)];
end
```

完成了非正则化的成本函数之后， 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。

完成了非正则化的成本函数后，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。

完成了反向传播算法之后，ex1_bp.m 将调用梯度检查算法，梯度检查算法会将我们计算得到的梯度和理论计算的梯度相比较。

当完成了整个 nnCostFunction.m 之后，ex1_bp.m 将调用检查梯度函数并输出正则化成本函数值。

3）ex1_bp.m 中调用的 **sigmoidGradient.m** 代码部分需要我们完成：

```
function g = sigmoidGradient(z)
g = zeros(size(z));
% ====================== YOUR CODE HERE ======================
for i=1:size(z,2)
    g(i)=sigmoid(z(i)).*(1- sigmoid(z(i)));
end
% =============================================================
End
```

完成该函数后，可以得到激活函数的梯度。

4）ex1_bp.m 中调用的 **randInitializeWeights.m** 代码部分需要我们完成：

```
function W = randInitializeWeights(L_in, L_out)
    W = zeros(L_out, 1 + L_in);
    % ====================== YOUR CODE HERE ======================
    W = 0.12*(2*rand(size(W))-1);
    ============================================================
end
```

3.结果分析

1）在前馈传播神经网络模型中，模型的预测准确率为 97.575%，与预期结果相符，如下图所示：

```
Training Set Accuracy: 97.575000
```

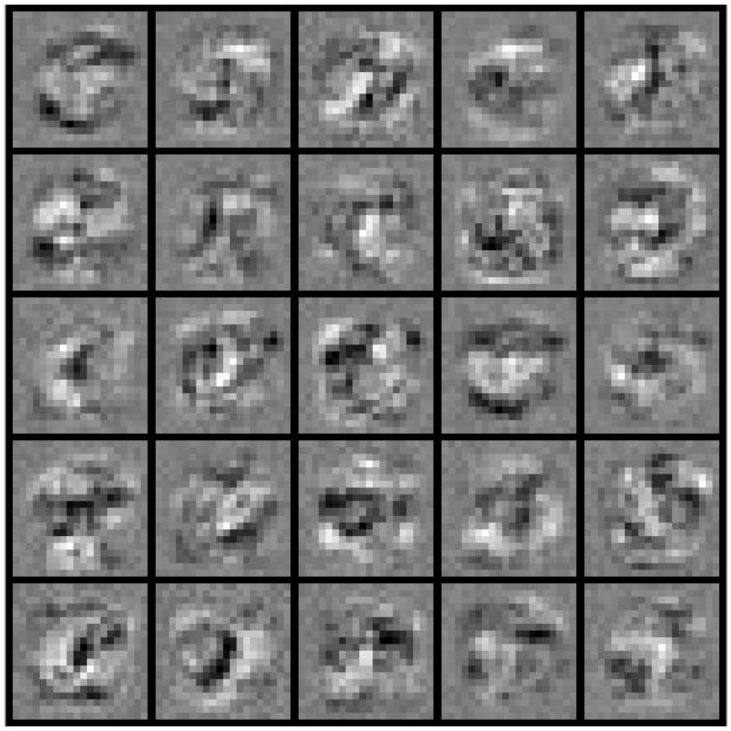
随机运行例子的结果如下图所示：

```
Neural Network Prediction: 10 (digit 0)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 5 (digit 5)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 3 (digit 3)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 7 (digit 7)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 9 (digit 9)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 5 (digit 5)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 5 (digit 5)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 7 (digit 7)
------------Program paused. Press enter to continue.---------------
Displaying Example Image
Neural Network Prediction: 4 (digit 4)
------------Program paused. Press enter to continue.---------------
```

2）在反向传播神经网络模型中，模型的预测准确率为 96.775%，与预期结果相符，如下图所示：

```
Training Set Accuracy: 96.775000
```

隐藏层的可视化如下：

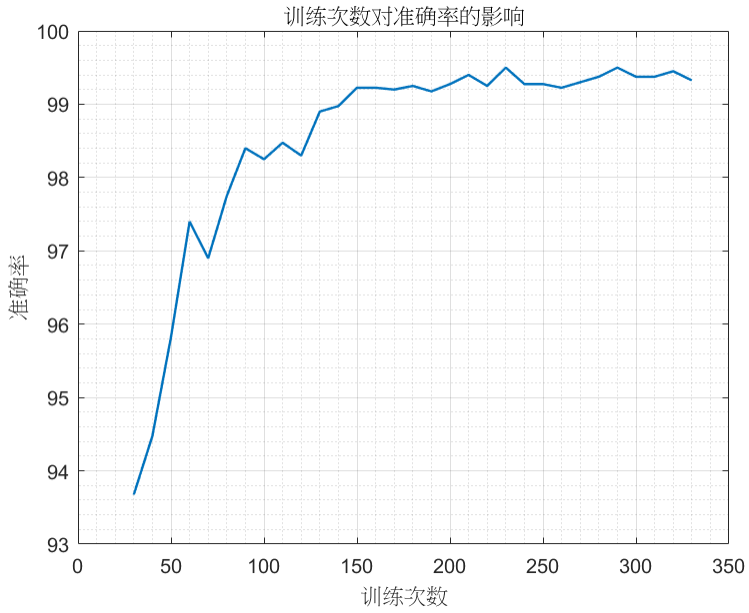


三、提高训练

基于实验手册的内容，尝试对现有实验做修改和调整，例如损失函数、激活函数等。比较不同神经网络带来的性能影响，分析猜测其背后的原因。

1. 调整训练次数

首先对训练次数作改变，修语句改 `options = optimset('MaxIter', 50);`中的次数为 30-330，可见准确率变化如下：

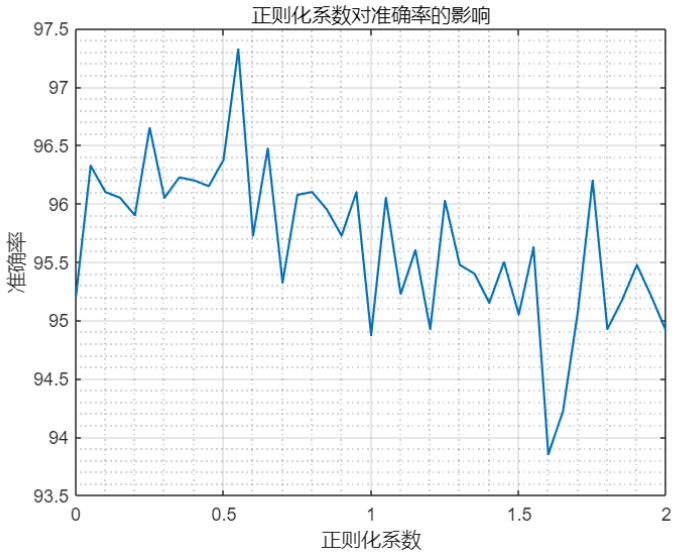


随着训练次数增加，观察准确率的变化情况，不难发现，准确率的变化趋势呈上升趋势。且由于初始参数随机生成的原因，每次准确率会有一些一些波动，

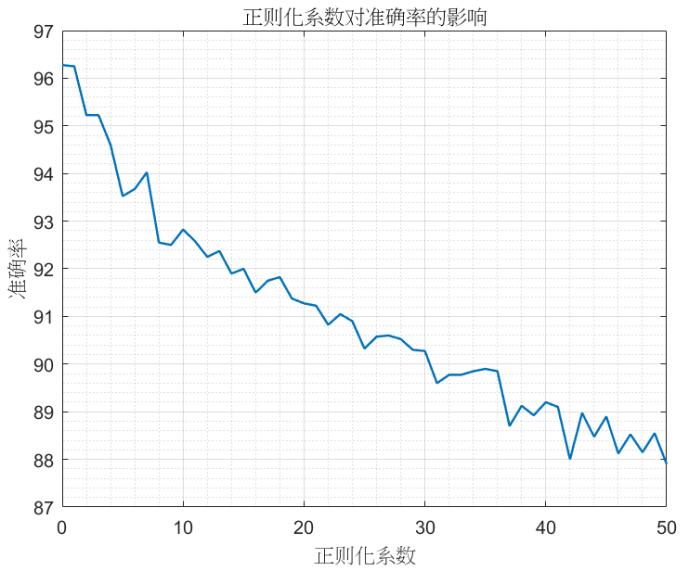
但不影响变化的整体趋势。

2. 调整正则化系数

接下来固定训练次数为 50，调整算法中的正则化系数λ，可以看到，当λ在 0 至 2 范围内变化时，准确率的变化情况如下图所示：



虽然准确率波动情况较大，但可以看出有隐约的下降趋势，猜测随着正则化系数λ变大，准确率呈现波动下降的趋势。为了验证这个猜想，我们扩大λ的取值范围（取 0-50），进行验证：

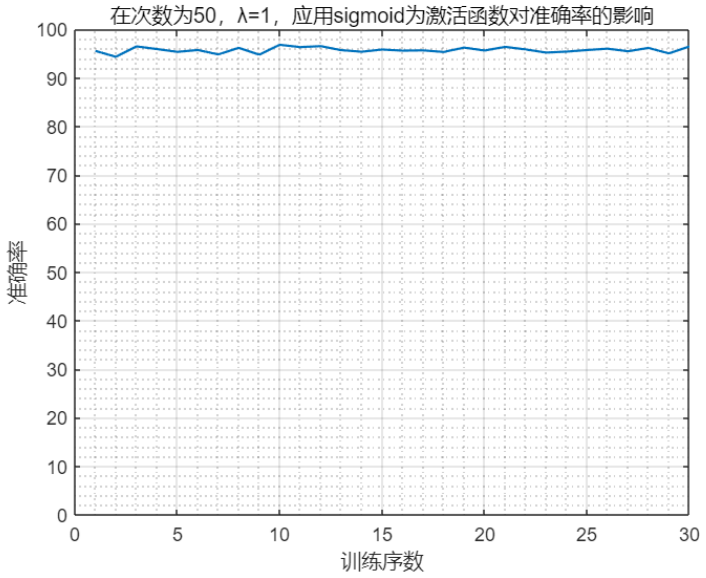


上图清晰地反映准确率随λ增大而波动下降的趋势，这是因为随着正则化系数的增大，训练容易产生欠拟合的情况。正则化系数的引入是为了防止过拟合，虽然较大的正则化系数能够使系统变得简单，但是会增加欠拟合的风险。一般正则化系数不会超过 1，此处为了探究系数的影响，故采用了比较夸张的数据。

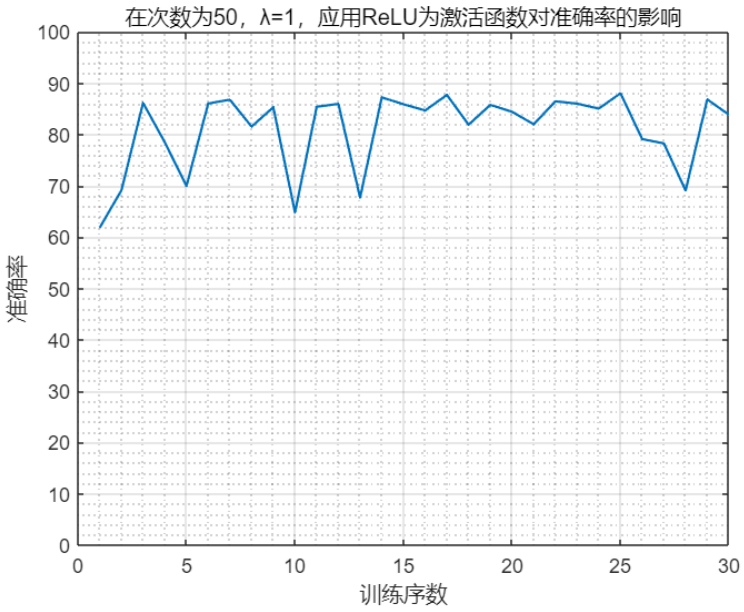
3. 改变激活函数

接下来我们将保持训练次数为 50 次，正则化系数为 1 ，损失函数为交叉熵的情况下，使用不同的激活函数，重复训练 30 次，观察模型的识别准确率的变化情况。

1）使用 sigmoid 函数作为激活函数时，模型准确率变化如下，以用作其他激活函数的对照组。

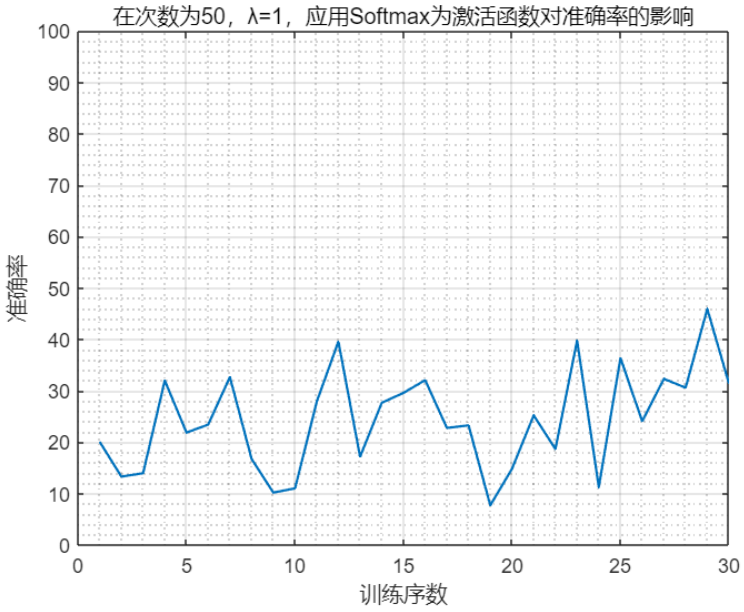


2）使用 ReLU 函数作为激活函数时，模型准确率变化如下图所示：



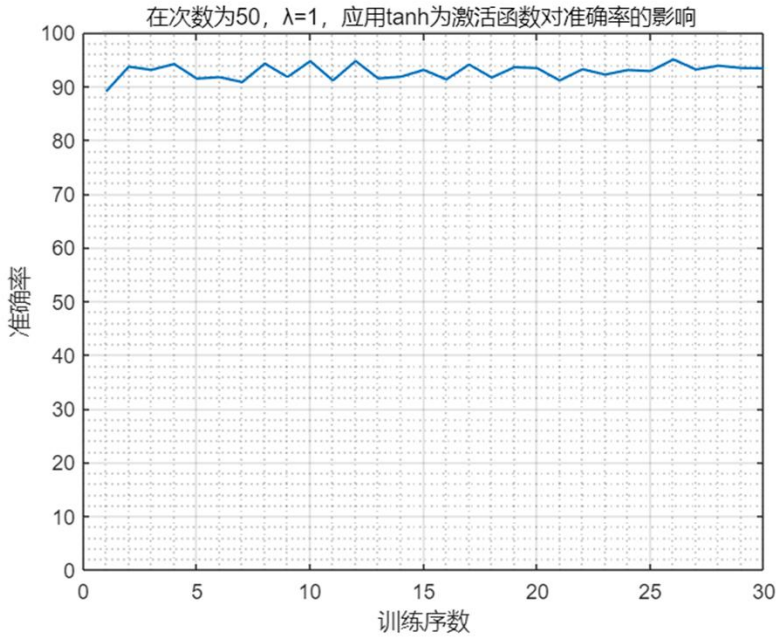
可以看到准确率在 80%左右，且波动较大，说明 ReLU 函数作为激活函数，效果上不如 sigmoid 函数，会较大程度影响模型准确率。

3）使用 Softmax 函数作为激活函数时，模型准确率变化如下图所示：

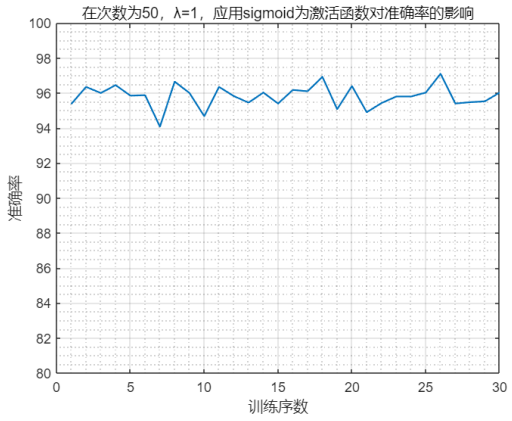


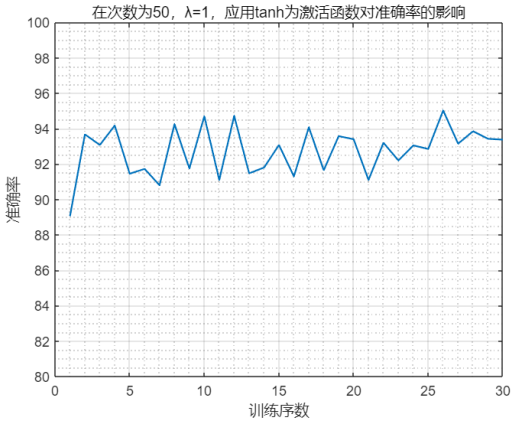
可以看到准确率在 10%-40 范围内波动，准确率较低（随机预测的准确率时 10%），且波动幅度非常大，说明 Softmax 函数不适合作为反向传播神经网络模型的激活函数。

4）使用 tanh 函数作为激活函数时，模型准确率变化如下图所示：



可以看到模型准确率与使用 sigmoid 函数作为激活函数时相近，为了更详细地对比两者，我们限制 y 轴为 80-100（%），结果如下：





可以看到相比于 sigmoid 函数，tanh 函数的准确率平均要低一些，在 92.5%左右，且波动更大，稳定性稍差。

综上，可见反向传播神经网络模型适合选用 sigmoid 函数和 tanh 函数作为激活函数，准确率都能达到 90%以上，其中 sigmoid 函数表现更好更稳定另外，激活函数可以选取 ReLU 函数，但表现较差；不建议选取 Softmax 函数。

四、实验心得

实验过程中遇到的什么问题。尝试使用什么方法去解决。通过实验获得了什么感悟与理解。

1.一开始最让我头疼的就是维度不匹配的问题，每次报错，基本都是维度对不上。后来仔细检查，才发现有时候权重矩阵里有偏置项，而输入数据没有；有时候又需要转置矩阵才能让维度匹配。为了搞清楚这些问题，我对照着公式，一个一个检查矩阵的维度，确保输入、权重和输出能正确对应。这让我明白，细节真的很重要，一个小小的维度错误，就能让整个程序崩溃。

2.在尝试不同的激活函数时，我发现并不是所有激活函数都能和交叉熵损失函数很好地搭配。比如 ReLU 函数，其输出范围包含 0，而交叉熵损失函数里有对数运算，对 0 进行对数运算就会出错。这让我意识到，选择激活函数和损失函数时，必须考虑它们的数学特性是否匹配。

3. 通过这次实验，我对前馈网络和反向传播算法有了更深的理解。前馈网络中，数据一层一层传递，每层通过激活函数引入非线性，让网络能学习复杂的特征。反向传播则是通过计算损失函数对权重的梯度，逐步调整权重，让损失最小化。这让我更清楚地看到神经网络是怎么学习的，以及如何调整网络结构和参数来提升性能。

4.在提高训练效果的部分，我用了控制变量法，逐一调整训练次数、正则化系数和激活函数等参数，看它们对模型性能的影响。但这种方法只能看出单一因素的作用，可能忽略了不同因素之间的相互影响。以后可以尝试同时调整多个因

素，比如在改变训练次数的同时调整正则化系数，或者在更换激活函数的同时优化网络结构，可能会有意想不到的收获。

这次实验让我看到深度学习在图像分类等领域的巨大潜力，但也意识到它面临的挑战，比如模型的可解释性、计算资源需求和过拟合问题。我希望以后能继续深入学习深度学习的理论和实践，探索更高效的算法和模型结构，用它们解决更多的实际问题。