实验二实验报告

1.实验介绍：

本实验使用MLP（多层感知器）回归实现MNIST的手写数字识别功能，主要工作是完成MLP分类器的代码实现。主要部分有：FC全连接层，Sigmoid层，ReLU层，Softmax回归，EuclideanLossLayer以及随机梯度下降算法（SGD）的实现。

2.代码分析：

SGD优化器代码如下：采用了随机梯度下降

```
import numpy as np

class SGD():
    def __init__(self, learningRate, weightDecay):
        self.learningRate = learningRate
        self.weightDecay = weightDecay

    # One backpropagation step, update weights layer by layer
    def step(self, model):
        layers = model.layerList
        for layer in layers:
            if layer.trainable:
                layer.diff_W = - self.learningRate * layer.grad_W -
self.weightDecay*self.learningRate * layer.W
                layer.diff_b = - self.learningRate * layer.grad_b
                # Weight update
                layer.W += layer.diff_W
                layer.b += layer.diff_b
```

全连接层fully connect layer主要代码如下（前馈和反馈）：

```
def forward(self, Input):
    self.input = Input
    return np.dot(self.input , self.W)  + self.b


def backward(self, delta):
    #self.grad_W = np.dot((self.inp).T , delta)
    self.grad_W = np.dot(np.transpose(self.input) , delta)
    self.grad_b = np.mean(delta , axis=0)
    return np.dot(delta , np.transpose(self.W))
```

Euclidean Loss Layer主要代码：

```
    def forward(self, logit, gt):
        """
          Inputs: (minibatch)
          - logit: forward results from the last FCLayer, shape(batch_size, 10)
          - gt: the ground truth label, shape(batch_size, 10)
        """
        self.grad = logit
        self.g = gt
        self.loss = 0.5*np.sum((logit-gt)*(logit-gt))/ (batch_size * 10)
        self.acc=np.sum(np.argmax(logit,axis=1)-np.argmax(gt,axis=1)==0)/batch_size
        return self.loss

    def backward(self):
        self.grad -= self.g
        return self.grad
```

Softmax Cross-Entropy Loss Layer主要代码:

```
def forward(self, logit, gt):
        """
          Inputs: (minibatch)
          - logit: forward results from the last FCLayer, shape(batch_size, 10)
          - gt: the ground truth label, shape(batch_size, 10)
        """
        self.log = logit
        self.g = gt

        logit=np.exp(logit)
        for i in range(logit.shape[0]):
                logit[i]=logit[i]/np.sum(logit[i])
        self.loss=np.sum(np.log(logit)*gt)/(-batch_size * 10)
        self.acc=np.sum(np.argmax(logit,axis=1)-np.argmax(gt,axis=1)==0)/batch_size
        return self.loss


    def backward(self):
        self.log -= self.g
        return self.log
```

ReLu激活层主要代码如下（前馈和反馈）：

```
    def forward(self, Input):
        self.input = Input
        return np.maximum(Input, 0)

    def backward(self, delta):

        return delta*(self.input>=0)
```

Sigmoid激活层主要代码如下（前馈和反馈）：

```
    def forward(self, Input):
        self.input = Input
        return  (1/ (1+ np.exp (-Input)))

    def backward(self, delta):
        return delta*((1/(1+np.exp(-self.input)))*(1-1/(1+np.exp(-self.input))))
```

3.结果分析：

本实验中超参数的选择如下：

batch_size = 100 总epoch次数为max_epoch = 20 init_std = 0.01 SGD学习率为 learning_rate_SGD = 0.001 weight_decay = 0.5 disp_freq = 50

本此实验一共训练了6个不同的模型，下面分别介绍

1.1 MLP with Euclidean Loss and Sigmoid Activation Function

具有一个hidden layer,采用Sigmoid激活函数并使用Euclidean Loss Function，结果如下：

```
1.1
Epoch [19]   Average training loss 0.0152    Average training accuracy 0.8792
Epoch [19]   Average validation loss 0.0130  Average validation accuracy 0.9190

Testing...
The test accuracy is 0.9003.
```

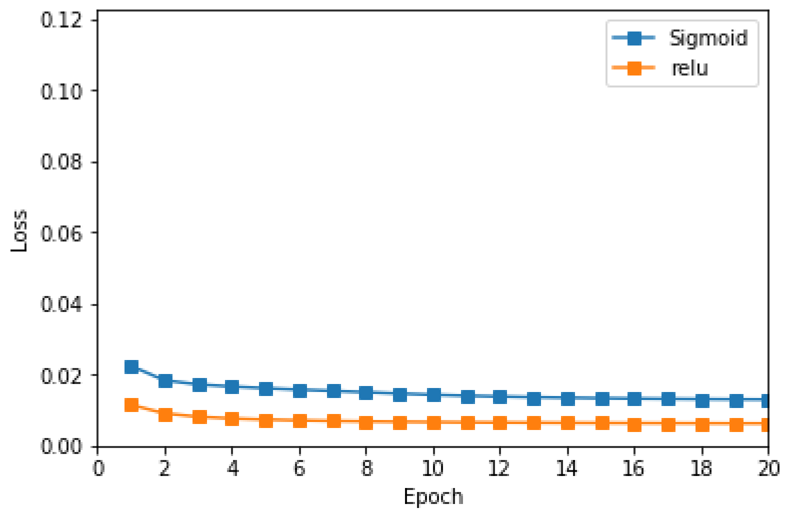
1.2 MLP with Euclidean Loss and ReLU Activation Function

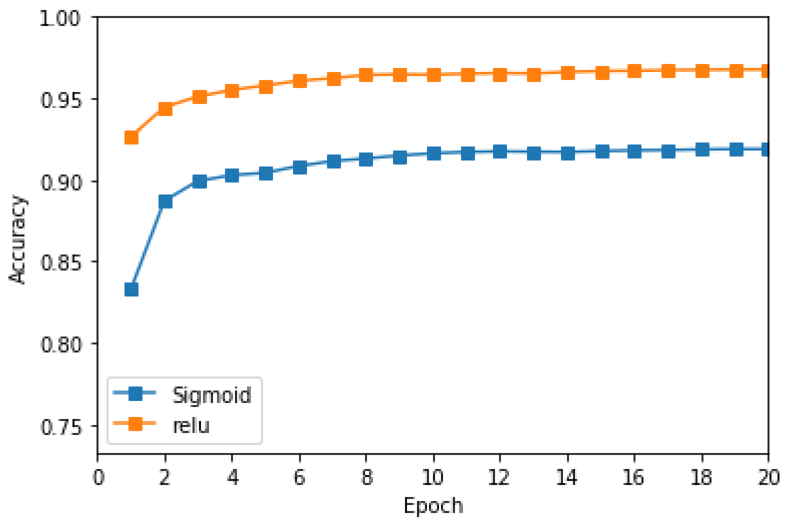
具有一个hidden layer,采用ReLu激活函数并使用Euclidean Loss Function，结果如下：

```
1.2
Epoch [19]   Average training loss 0.0071    Average training accuracy 0.9520
Epoch [19]   Average validation loss 0.0062  Average validation accuracy 0.9676


Testing...
The test accuracy is 0.9531.
```

以上两个模型的运行结果如图，可以发现大约在第三个epoch模型收敛，使用relu激活函数的效果更好，准确率达到0.95左右，sigmoid激活函数模型准确率在0.9左右。验证集和测试集准确度相差不大，未出现过拟合情况。





2.1 MLP with Softmax Cross-Entropy Loss and Sigmoid Activation Function

具有一个hidden layer,采用Sigmoid激活函数并使用SoftMax的交叉熵损失函数，结果如下：

```
2.1

Epoch [19]   Average training loss 0.1767    Average training accuracy 0.8823
Epoch [19]   Average validation loss 0.1737  Average validation accuracy 0.9218

Testing...
The test accuracy is 0.9102.
```

2.2 MLP with Softmax Cross-Entropy Loss and ReLU Activation Function

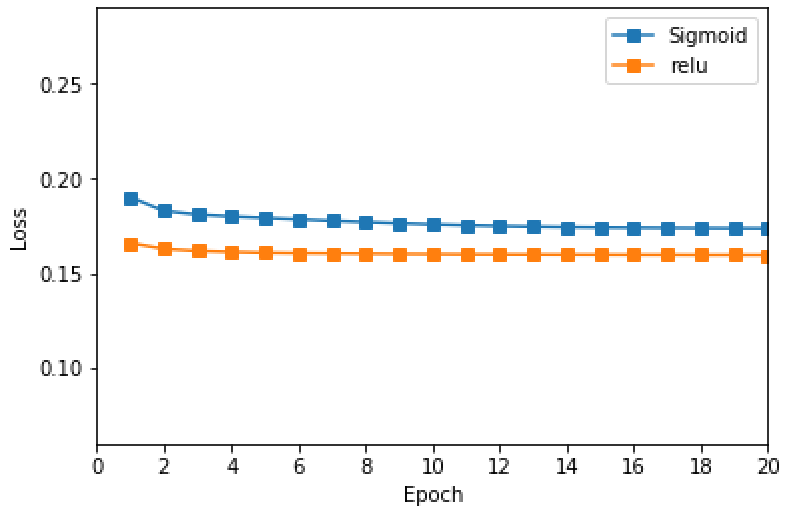
具有一个hidden layer,采用Sigmoid激活函数并使用SoftMax的交叉熵损失函数，结果如下

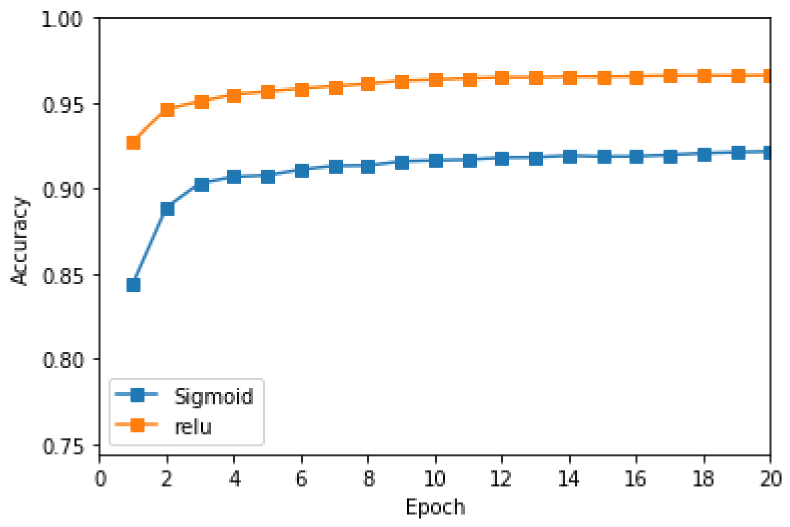
```
2.2

Epoch [19]   Average training loss 0.1622    Average training accuracy 0.9524
Epoch [19]   Average validation loss 0.1596  Average validation accuracy 0.9662

Testing...
The test accuracy is 0.9534.
```

以上两个模型的运行结果如图，可以发现大约在第三个epoch模型收敛，使用relu激活函数的效果更好，准确率达到0.95左右，sigmoid激活函数模型准确率在0.91左右。验证集和测试集准确度相差不大，未出现过拟合情况。





3.两层神经网络

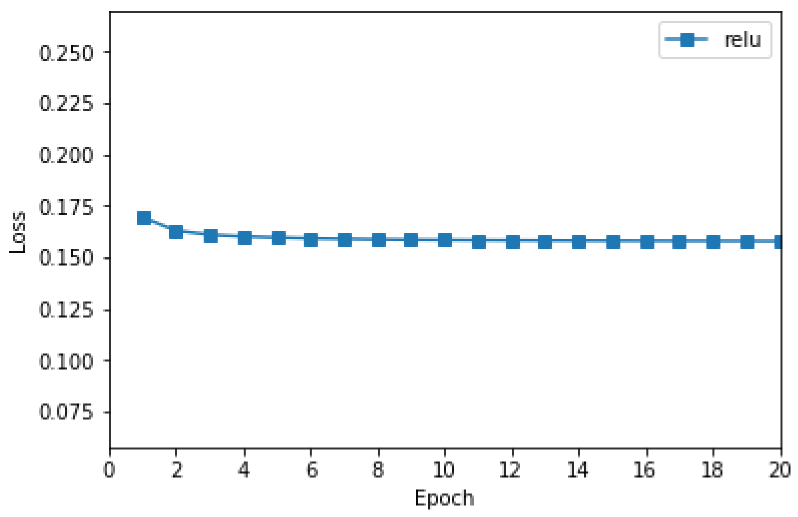
结果如下：

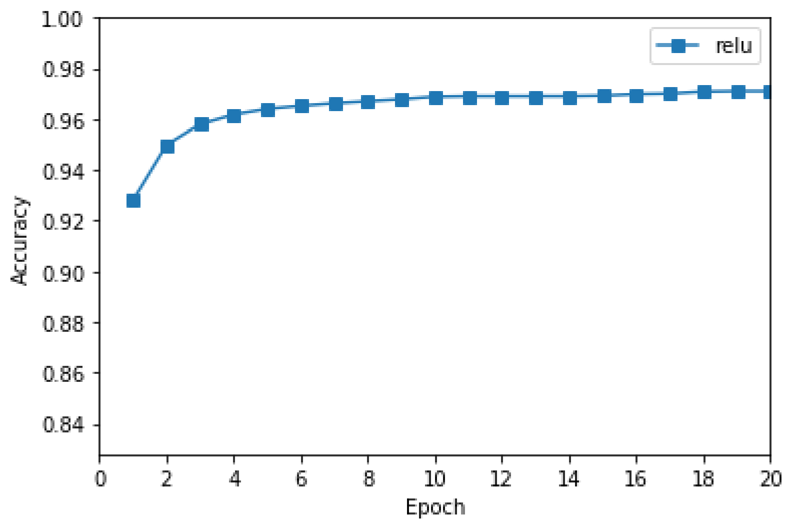
```
3.

Epoch [19]   Average training loss 0.1603    Average training accuracy 0.9590
Epoch [19]   Average validation loss 0.1577  Average validation accuracy 0.9710

Testing...
The test accuracy is 0.9582.
```

该模型加入了两个分别为256节点和64节点的 激活函数为ReLu的隐藏层，采用softmax交叉熵，结果如下，可以看到收敛较快准确率约96%，未出现过拟合的情况





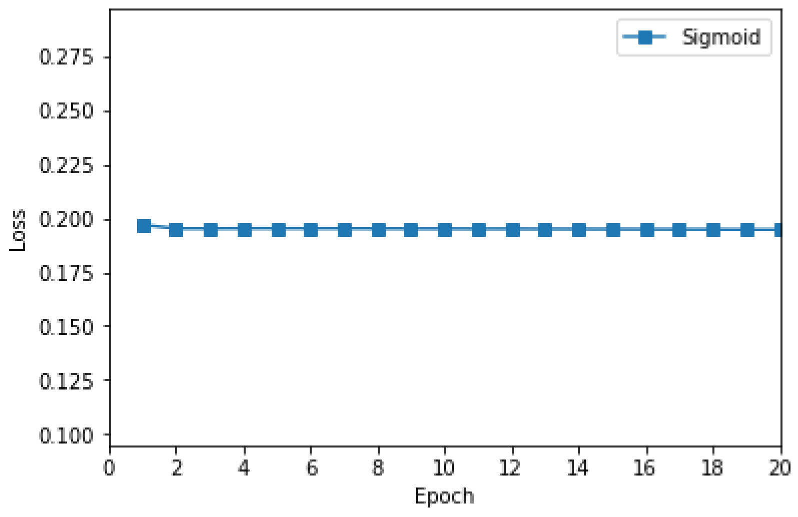
4.两层神经网络

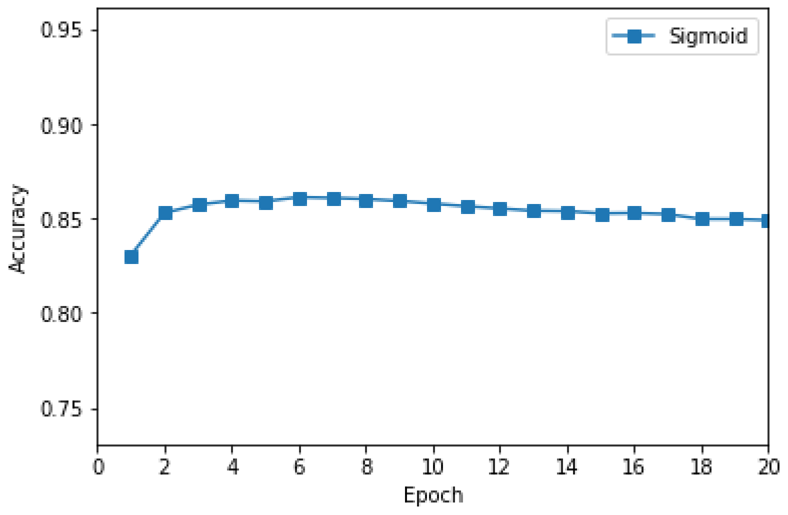
结果如下：

```
Epoch [19]   Average training loss 0.1965    Average training accuracy 0.7825
Epoch [19]   Average validation loss 0.1948  Average validation accuracy 0.8492

Testing...
The test accuracy is 0.8261.
```

以图表形式表示：





准确率约为82%，测试集准确率和验证集准确率基本相同，未发生过拟合现象。但整体效果较为不理想，后期梯度基本不再更新，可能产生了梯度消失的问题。

综上，6个模型的效果从好到坏大致为 3>2.2≈1.2>2.1≈1.1>4，总体上ReLu激活函数效果比Sigmoid函数要好，且在层数较多时差距越大，softmax交叉熵层比欧几里得loss效果略好，而增加了层数效果也有相应的提升，但采用sigmoid激活函数时效果下降，可能是发生了梯度消失的问题。