Week3：实现全连接神经网络

姓名：罗镜泉
学号：17341120

一、实验内容

1、实现全连接神经网络的前向传播和反向传播

2、实现交叉熵损失函数

3、实现带动量的SGD优化器

4、搭建三层全连接神经网络在MNIST数据集上训练和测试

二、实验过程

1、实现relu激活函数及其导数

```
def relu(x):
    y = x.clone()
    y[y<0] = 0
    return y
```

```
def relu_derivation(tensor):
    return torch.where(tensor > 0, torch.full_like(tensor, 1),
torch.full_like(tensor, 0))
```

2、实现softmax

对于向量$x=[x_1,x_2,...,x_n]$来说，softmax计算公式如下

$$softmax(x_i)=\frac{e^{x_i}}{\sum_i^n e^{x_j}} \tag{1}$$

但这条公式在实际运算中可能会由于指数爆炸造成溢出，修改后的公式为

$$m=\max_i\{x_i\} \tag{2}$$

$$softmax(xi)=\frac{e^{x_i}/e^m}{(\sum_1^n e^{x_j})/e^m}=\frac{e^{x_i-m}}{\sum_1^n e^{x_j-m}}$$

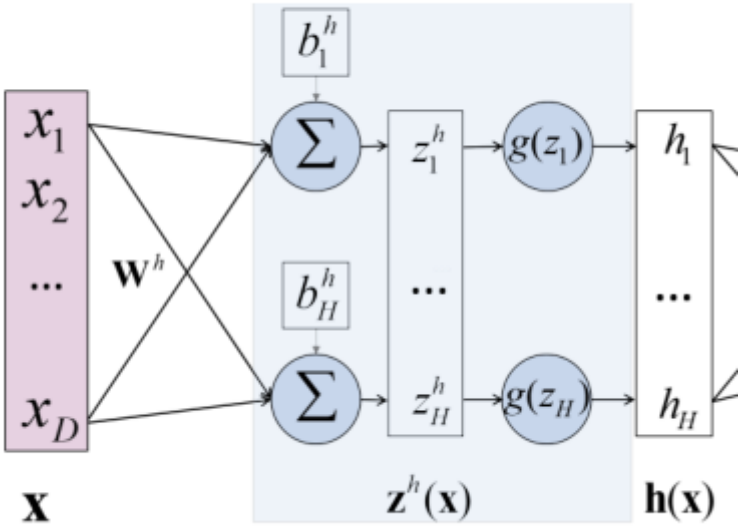
```
def softmax(tensor, dim=None):
    # dim指定进行softmax运算的维度，默认最后一维
    if dim is None:
        dim = tensor.dim() - 1
    # 找出最大值，令每个值减去最大值max{x}
    # 保证softmax指数幂位置小于等于0，  不会发生溢出
    # 实际上是在softmax分式上下同除以exp（max），所以计算结果不变
    max_values, _ = torch.max(tensor, dim=dim, keepdim=True)
    e = torch.exp(tensor-max_values)
    return e / torch.sum(e, dim=dim, keepdim=True)
```

3、实现crossEntropy

交叉熵计算中存在对数计算，可能出现下溢的情况，在实际实现中加上一个很小的数字保证不会下溢。

```
def crossEntropy(tensor1, tensor2, dim=None):
    # dim 指定运算的维度
    # 默认对最后一维进行交叉熵计算
    if dim is None:
        dim = tensor1.dim() - 1
    return - torch.sum(tensor1 * torch.log(tensor2 + 1e-10), dim=dim)
```

4、对单层全连接神经网络前向传播和方向传播的推导



使用mini-batch stochastic descent的方法，假设输入$input$维度为

$$[batch\_size, input\_size] \tag{3}$$

权重向量W^h的维度为

$$[input\_size, hidden\_size] \tag{4}$$

偏置量$bias$的维度为

$$[hidden\_size,] \tag{13}$$

输出$hidden_output$的计算公式为

$$hidden\_output = input * W^h + bias \tag{6}$$

那么输出$hidden_output$的维度为

$$[batch\_size, hidden\_size] \tag{7}$$

如果对于隐藏层输出经过relu之后张量的梯度为

$$h\_relu\_grad \tag{8}$$

对于隐藏层输出(未经过激活)$hidden\_output$ 的梯度为

$$hidden\_ouput\_grad = \begin{cases} h\_relu\_grad, & if\ input > 0; \\ 0, & if\ input < 0 \end{cases} \tag{9}$$

对于权重向量W^h来说梯度为

$$W_grad = input^T * hidden\_output\_grad \tag{10}$$

对于$bias$的梯度为

$$bias\_grad = W\_ones * hidden\_ouput\_grad \tag{11}$$

其中W_ones是一个全为一的行向量，维度为$[batch_size]$，即对$didden_output_grad$进行列求和。

对于input的梯度为

$$input\_grad = hidden\_output\_grad * (W^h)^T \tag{12}$$

```
    # 部分代码
    def backward(self):
        # 记录上一次迭代的梯度
        self.w1_t, self.b1_t = self.w1_grad, self.b1_grad
        self.w2_t, self.b2_t = self.w2_grad, self.b2_grad
        self.w3_t, self.b3_t = self.w3_grad, self.b3_grad

        output_grad = (softmax(self.output)  - self.onehot_label) /
self.output.shape[0]

        # 第三层（输出层）权重梯度
        # w的梯度
        self.w3_grad = torch.mm(self.h2_relu.permute(1, 0), output_grad)
        # b的梯度
        self.b3_grad = torch.sum(output_grad, dim=0)
        # 输入（relu之后张量）的梯度
        h2_relu_grad = torch.mm(output_grad, self.w3.permute(1, 0))
        # 输入（relu之前张量）的梯度
        h2_grad = h2_relu_grad.clone()
        h2_grad[self.h2 < 0] = 0

        # 第二层隐藏层梯度
        self.w2_grad = torch.mm(self.h1_relu.permute(1, 0), h2_grad)
        self.b2_grad = torch.sum(h2_grad, dim=0)

        h1_relu_grad = torch.mm(h2_grad, self.w2.permute(1, 0))
        h1_grad = h1_relu_grad.clone()
        h1_grad[self.h1 < 0] = 0

        # 第一层隐藏层梯度
        self.w1_grad = torch.mm(self.input.permute(1, 0), h1_grad)
        self.b1_grad = torch.sum(h1_grad, dim=0)
```

5、loss对输出层输出的梯度

在老师的课件中有完整的推导，输出层输出经过softmax之后与label计算交叉熵取平均值作为loss， 这时候对于输出层输出的梯度为：输出经过softmax之后的结果减去label， 再除以batch_size。如果不除以batch_size会是梯度累积，learning rate要相应调小。

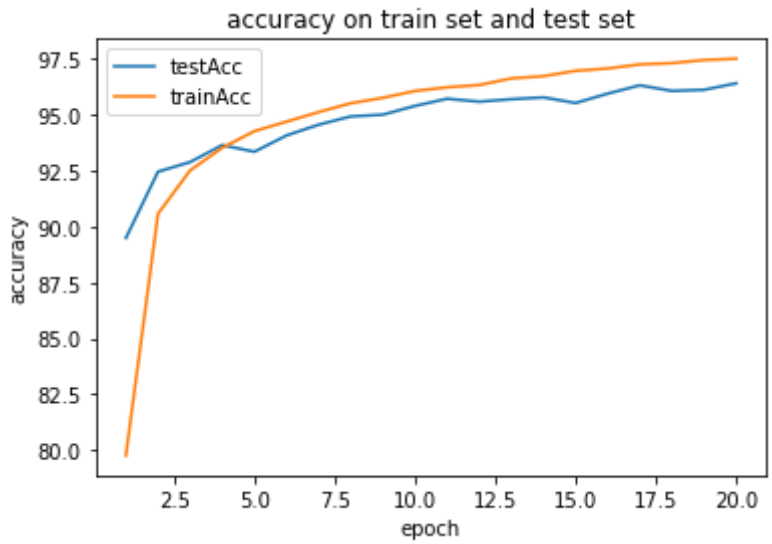
6、带动量的SGD优化器

动量的引入能够加速收敛，减小震荡。

```
    def SGD(self, lrate, gama):
        self.w1 -= gama * lrate * self.w1_t + lrate * self.w1_grad
        self.b1 -= gama * lrate * self.b1_t +lrate * self.b1_grad
        self.w2 -= gama *  lrate * self.w2_t + lrate * self.w2_grad
        self.b2 -= gama * lrate * self.b2_t + lrate * self.b2_grad
        self.w3 -= gama *  lrate * self.w3_t + lrate * self.w3_grad
        self.b3 -= gama * lrate * self.b3_t + lrate * self.b3_grad
```

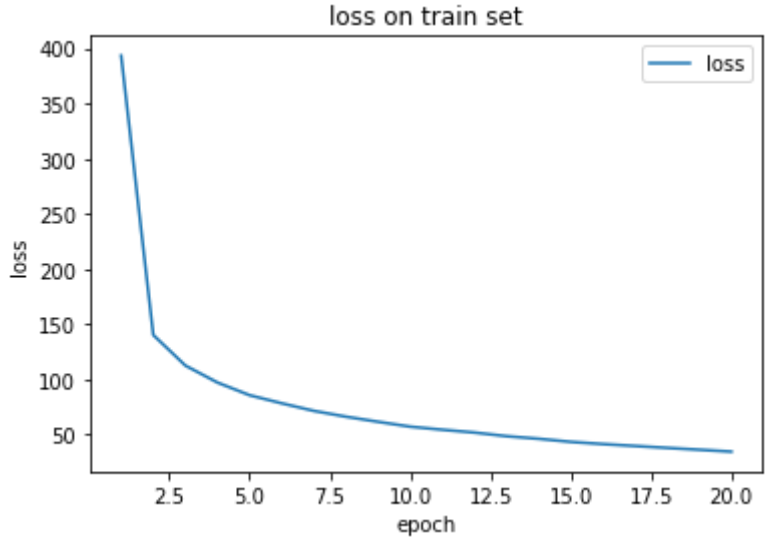
三、实验结果

1、准确率



准确率在经过20个epoch之后，在训练集上达到97.5%左右，在测试集上达到96%左右。

2、损失值



损失一直呈现下降的趋势，也没有出现震荡，最终达到34左右，收敛的趋势已经很小了。