University of Fudan
School of Data Science

Homework #1

Student name: *Du Mengfei*

Course: *Netural Network and Deep Learning* – Professor: *Dr. Zhang*
Due date: *April 7th, 2023*

构建两层神经网络分类器

1 训练：激活函数, 反向传播, loss 以及梯度的计算, 学习率下降策略, L2 正则化, 优化器 SGD, 保存模型

2 参数查找：学习率，隐藏层大小，正则化强度

3 测试：导入模型，用经过参数查找后的模型进行测试，输出分类精度

4 数据集：MINIST

5 可视化：可视化训练和测试的 loss 曲线，测试的 accuracy 曲线，以及可视化每层的网络参数。

github link:https://github.com/mengfeidu/Two-Layers-Network-for-MNIST

1. 实现细节

网络层数 根据题意，我们构建了具有两层隐藏层的神经网络分类器 Model1:

Input Data	**Input**	(Batch size , 784)
Hidden Layer 0	**Input**	(Batch size , 784)
	Output	(Batch size , 784)
	Activate Function	Leaky Relu
Hidden Layer 1	**Input**	(Batch size , 784)
	Output	(Batch size , 100)
	Activate Function	Leaky Relu
Output Layer	**Input**	(Batch size , 100)
	Output	(Batch size , 10)

其中，两层隐藏层的输入输出维度分别为 (784,784) 与 (784,100)，最后通过一个输出层将 100 维的特征映射到 10 维的分类。
同时也构建了只具有两层参数层的神经网络 Model2，即具有一个隐藏层和一个输出层的网络，这里隐藏层的输入输出维度为 (784,784)，输出层的输入输出维度为 (784,10)。

激活函数 本次实验中我们选择 Leaky Relu 作为隐藏层的激活函数：

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

其中，$\alpha = 0.01$ 代码实现 (utils.py LeakyRelu())：

```
# Active functions
def LeakyRelu(input_data,gradient = False):
    if not gradient:
        output = copy.deepcopy(input_data)
        output[output <= 0] = 0.01 * output[output <= 0]
        return output
    else:
        output = copy.deepcopy(input_data)
        output[output > 0] = 1
        output[output <= 0] = 0.01
        return output
```

损失函数 本次实验中我们选择平方损失函数 (Mean Square Error, MSE) 作为神经网络分类器的损失函数

$$loss = \frac{1}{2}(\mathbf{y}_{pred} - \mathbf{y}_{true})^T(\mathbf{y}_{pred} - \mathbf{y}_{true})$$

代码实现 (utils.py class MSE)：

```
# Loss function
class Mse:
    def loss_value(self, input_data, target):
        self.input = input_data
        self.target = target
        self.batch_size , _ = input_data.shape
        return 0.5*np.sum((self.input - self.target)**2)/self.batch_size
    def gradient(self):
        g = self.input - self.target
        g /= self.batch_size
        return g
```

学习率下降策略 本实验选择连续下降调整策略，实现了两种衰减策略：

(1) 每轮 epoch 学习率的衰减公式为:

$$lr = \frac{1}{1 + lr\_decay \times i} \times lr\_start$$

其中，i 为当前的 epoch 轮次

(2) 指数衰减调整策略：

$$lr = lr\_start \times lr\_decay^i$$

其中，i 为当前的 epoch 轮次代码实现 (main.py train())：

```
    ## lr decay
    #(1)
    lr = lr_start * 1.0 / (1.0 + lr_decay * i)
    #(2)
    # lr = lr_start *  np.exp(lr_decay, i)
```

l2 正则化、Momentum 与 SGD 优化 在更新参数过程中，本实验选择 SGD 优化算法进行更新参数，同时使用 l2 正则化和 Momentum 技术优化参数过程。

根据课程关于 SGD 算法的内容:

"Gradient Descent"

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

以及 Momentum 机制, 我们计算了两次梯度更新的方向的差值, 并更新线性层的参数，具体代码见 utils.py 文件，class LinearLayer 的 update 函数。

模型保存与加载 实验中主要利用 numpy 包中的 save 和 load 函数实现模型的保存和加载。

2. 超参数设置

在确定超参数的过程中，我们在 MNIST 训练集选择 5000 样本组成的小数据集上进行搜索，检索范围为：

1 batch size ： $\{64,32,16,8,4\}$

2 学习率 lr ： $\{10^{-2},5\times10^{-3},1\times10^{-3},3\times10^{-4}\}$

3 学习率衰减因子: *lr_decay* ： $\{0.001,0.009\}$

4 l2 正则化: regularization ： $\{0.01,0.005,0.001\}$

Batch Size 经过性能对比，本实验选择 batch size 16 作为最优的 batch size.

学习率 本实验选择初始学习率 *lr_start* 为 0.001。当学习率大于 0.005 时，观察到存在梯度爆炸的现象。

l2 正则化 本实验选择 l2 正则化参数为 0.001。实验结果显示，过高的正则化参数会明显降低模型的性能。

学习率衰减 本实验选择学习率衰减因子 *lr_decay* 为 0.001。

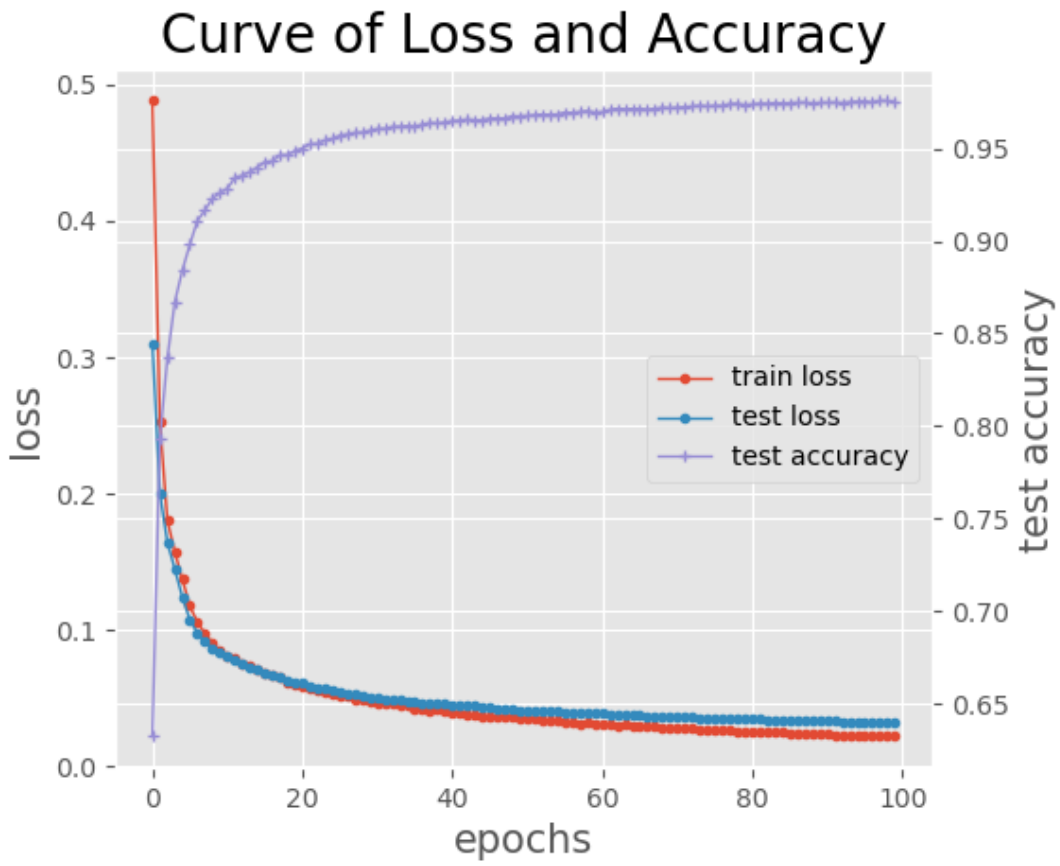
3. 实验结果

模型	Test Loss	Test Acc
Model1	**0.032**	**0.976**
Model2	0.055	0.960

从中也可以看出，具有更多参数的 Model1 的性能更好。最终模型在测试集上的准确率为 97.6%。

4. 训练可视化

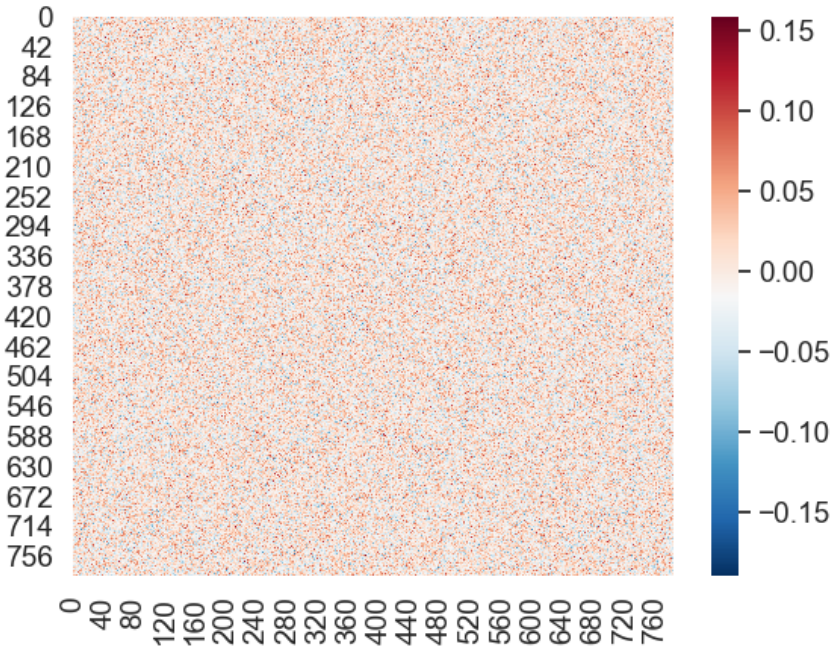
我们对最优模型训练过程中的训练集损失、测试集损失、测试机准确率进行可视化如下：



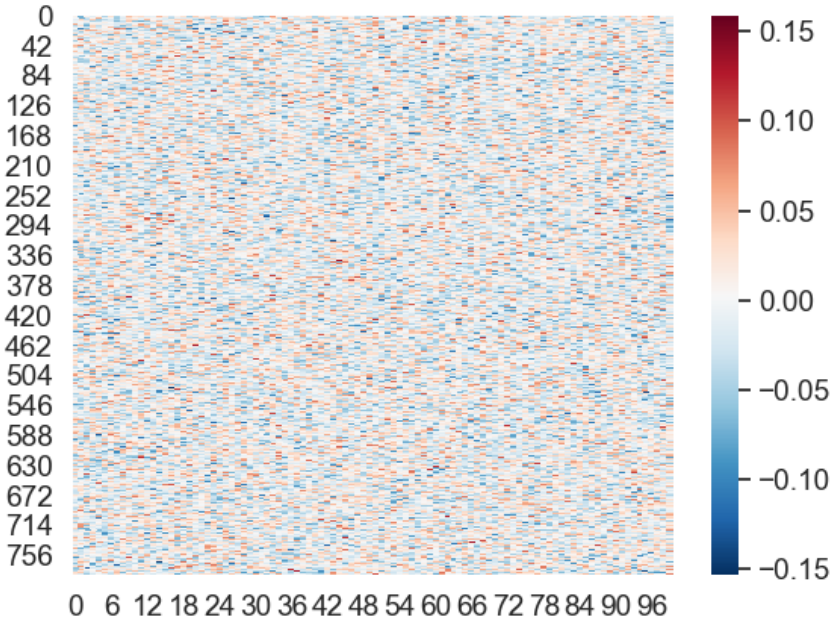
5. 神经网络参数可视化

我们对 Model1 的每一层网络的参数绘制对应的热力图如下：

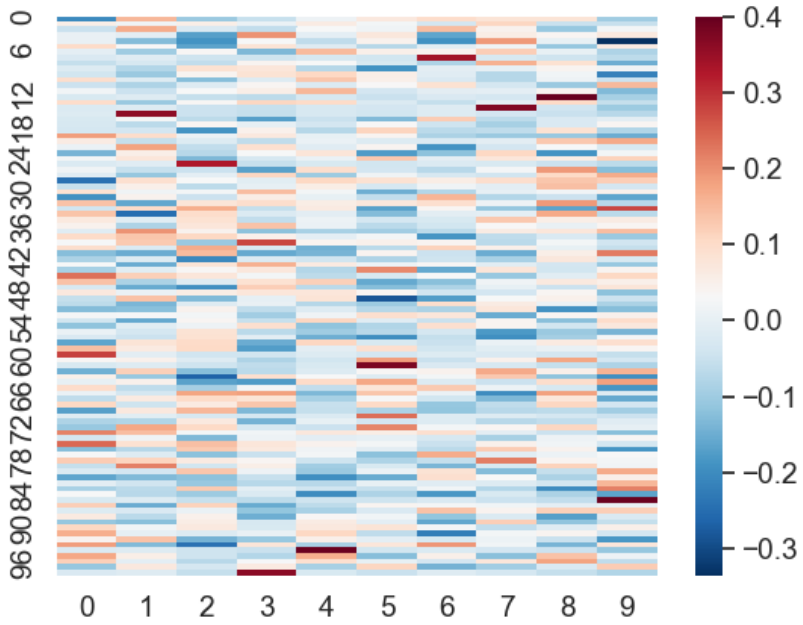
HiddenLayer 0 (784,784)



HiddenLayer 1 (784,100)



OutputLayer (100,10)



可以看出，输出层对应每个类别的参数向量值的分布已经具备了比较明确的区别，代表网络对于手写数字的输入具有一定的分类能力。