Softmax 实现手写数字识别

本次作业基于 Softmax 实现手写数字识别，根据作业一的要求完成前向计算和参数的更新，并通过调整超参数记录对分类性能的影响，此外在 SGD 中比较加入 momentum 和没有加入 momentun 对分类性能的影响。

在数据集方面，用的是 MNIST 数据集，包含 60,000 个训练样本和 10,000 个测试样本。MNIST 中的数字范围是 0 到 9，由此可知有 10 种类别（10 种标签），分类数量 K>2，更适合用 K>2 范畴 Multinoulli/categorical 分布的假设。

构造一个非线性的函数用描述样本 x 为 k 类的概率，该函数为 softmax 函数，在条件似然函数（独立性假设）的前提下，将最大化似然等同于最小化，即交叉熵误差函数。确认目标函数后，分别对 W 和 b 求偏导，计算梯度实现对参数 W 和 b 进行更新，上述为实现数字识别的基本思路。

一．数据处理方面

MNIST 数据集，包含 60,000 个训练样本和 10,000 个测试样本。MNIST 中的数字范围是 0 到 9。由于考虑到一次更新成本函数的计算机开销比较大，因此将数据集进划为更小的集子，利用 batch_size 参数为子集的样本数量，该方法用于随机梯度下降法中。数字范围是 0 到 9，由此可知有 10 种不同的标签，需要将其转化为 one-hot 的表达方式。代码如下，创建一个全 0 的（n，10）的矩阵，其中 n 为样本数量，将下标与标签对应的值改为 1。

```
def one_hot(self):
        # one-hot
        t = np.zeros((softmax.shape[1],softmax.shape[0]))
        t[range(0,softmax.shape[1]),labels] = 1
        #print("labels[0] =",labels[0])
        #print("t[0] =",t[0])
        t = t.T
        return
```

二．前向传播

假设 batch_size 为 100，在 forword 函数中：

Input：为（784，100）的二维矩阵，表示有 100 个样本，每个样本有 784 个参数。

W：为（784，10）的二维矩阵，初始化为 0 的权重。

b：为（1，10）的一维矩阵，初始化为 0 的偏置。

经过矩阵相乘和元素积可以求出 softmax 函数，公式如下：

$$\text{softmax}(X)=\frac{\exp(W^{(k)}.T\times X+b)}{\sum_{j=1}^{K}\exp(W^{(j)}.\times X+b)}\in(0,1)$$

```
z = np.dot(self.W.T,Input.T )+self.b.T
#print(z.shape) 10 100
exp = np.exp(z)
#print("exp.shape=",exp.shape) # exp.shape= (10, 100)
sum_exp = np.sum(exp,axis=0,keepdims = True)
#print("sum_exp.shape=",sum_exp.shape) # sum_exp.shape= (1, 100)
softmax = exp/sum_exp
```

在求出 softmax 函数后，经过交叉熵误差函数可求出 loss 值。

```
tmp = t*np.log(softmax)
# print("tmp.shape =",tmp.shape) #tmp.shape = (10, 100)
tmp_sum = np.sum(tmp,axis=0,keepdims = True)
#print("tmp_sum.shape =",tmp_sum.shape) # tmp_sum.shape = (1, 100)
#print("Input.shape[0] =",Input.shape[0]) # Input.shape[0] = 100
#计算损失值
loss = -np.sum(tmp_sum,axis=1)/Input.shape[0]
```

再根据求出来的 softmax 函数求出 accuracy，代码如下：

```
tmp2 = np.argmax(softmax,axis=0)
#print(tmp2.shape)
#print("tmp2 =",tmp2)
#计算准确率
#print(np.mean(tmp2==labels))
#result=np.sum(tmp2==labels)/len(labels)
#print("result =",result)
#两种方法都可以
acc = np.mean(tmp2==labels)
```

三、计算梯度

结合所求解出来的 softmax 函数的值，对 W 和 b 求偏导，公式如下：

$$\frac{\partial E}{\partial W}=\frac{1}{N}\sum_{n=1}^{N}\left(f(x^{(n)})-t^{n}\right)*x^{(n)}$$

其中 f(x)为一个是(10，1)的矩阵，t 为一个（10，1）的矩阵，x 为（1，784）的矩阵，经过矩阵相乘得到是（10，784）的矩阵，共有 N 个（10，784）的矩阵进行求和操作，再除于 100，可得到对 W 的偏导。

在代码中，seft.softmax 是（10，100）的矩阵，self.t 是（10，100）的矩阵，两者求和得到（10，100）矩阵，再与 self.input 是（100，784）的矩阵进行矩阵相乘，即完成了求和操作，直接除于样本 100 即可，代码如下：

```
self.grad_W = np.dot((self.softmax-self.t),self.input)/self.input.shape[0]
self.grad_b = np.average((self.softmax-self.t),axis=1)
#print("self.grad_W.shape =",self.grad_W.shape) #self.grad_W.shape = (10, 784)
#print("self.grad_b.shape =",self.grad_b.shape)
```

对 b 求偏导同理。

四、更新 w 和 b

计算完梯度求出对 W 和对 b 的偏导后，即可更新 W 和 b，这里引入了 momentum 动量法，公式如下：

$$v_t = \gamma v_{t-1} + \alpha \nabla b$$

$$\theta_{new} = \theta - v_t$$

更新代码如下：

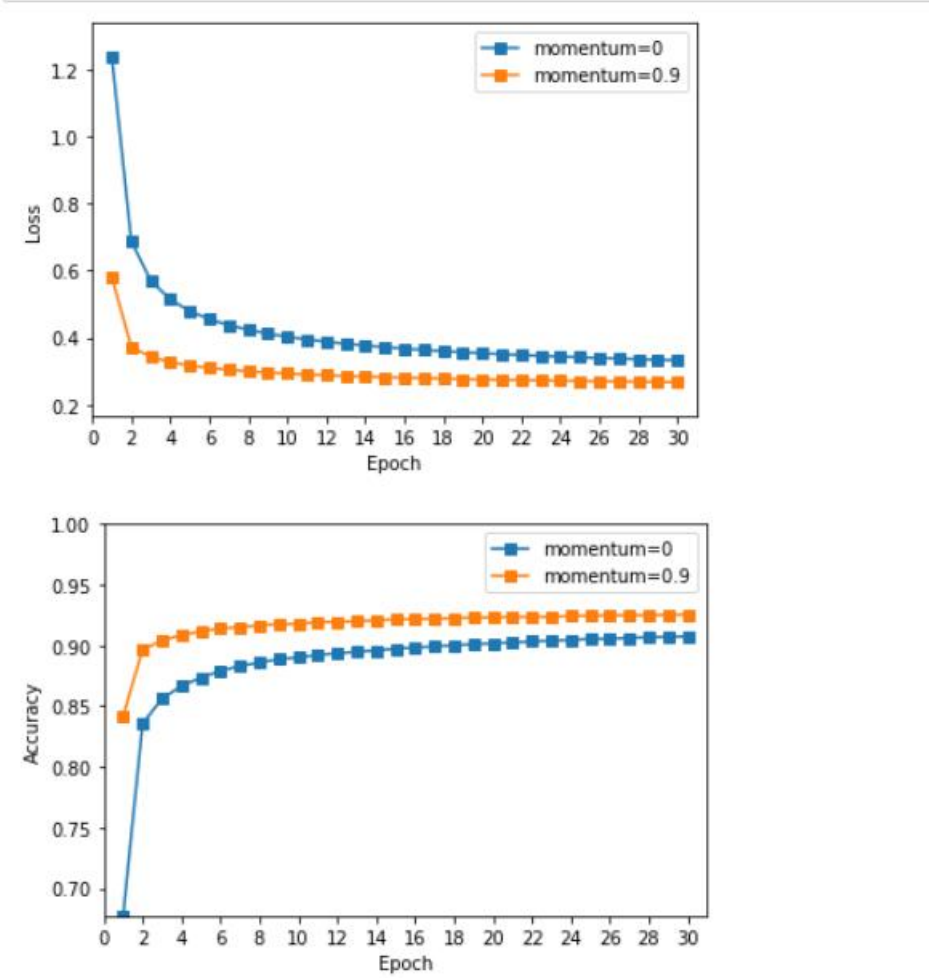
```
self.w_v = self.momentum * self.w_v + self.learning_rate * layer.grad_W
self.b_v = self.momentum * self.b_v + self.learning_rate * layer.grad_b
  print("self.w_v.shape =",self.w_v.shape)
  print("self.b_v.shape =",self.b_v.shape)
  print("layer.W.shape =",layer.W.shape)
  print("layer.b.shape =",layer.b.shape)
layer.W += -self.w_v.T
layer.b += -self.b_v
```

在实验中，对比使用 momentum 动量法和不使用 momentum 动量法的结果，一般将其设置为 0.9 和 0。

五、实验结果

1.比较使用 momentum 和不使用 momentum 的实验结果：

其他参数为：'max_epoch': 30,'batch_size': 100,'learning_rate': 0.01 的情况下来进行对比，结果如图所示：

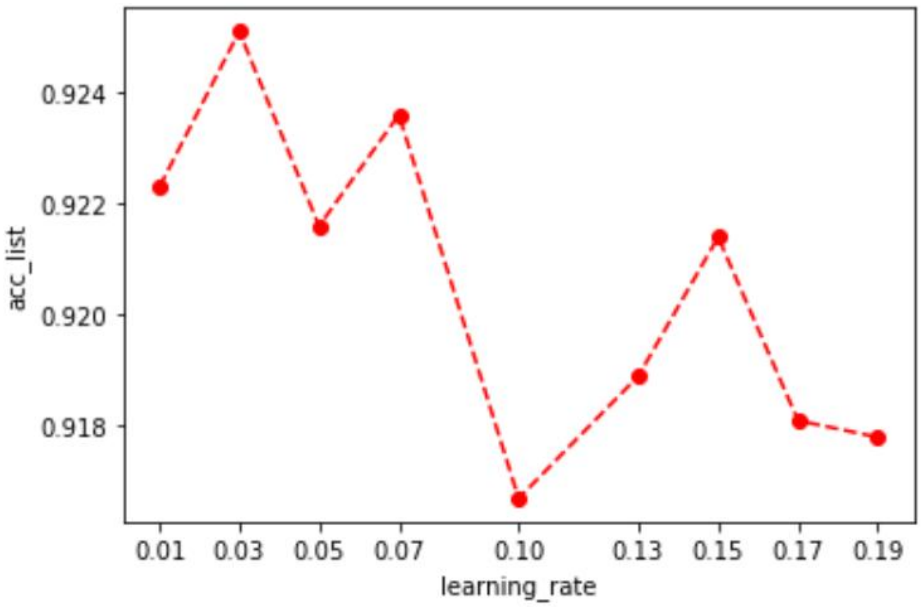


通过对比可以发现使用 momentum 后准确度大于没有使用 momentum 的准确度，而使用 momentum 后 loss 小于没有使用的 loss。两种方法训练时间分别为：40s 和 41s，在训练次数比较少时，并不

差距不大，如果将训练次数加到 100 次，两种方法的时间分别为 141s 和 144s，相差了 3s。momentum 算法思想：参数更新时在一定程度上保留之前更新的方向，同时又利用当前 batch 的梯度微调最终的更新方向，简言之就是通过积累之前的动量来加速当前的梯度，因此使用 momentum 后，收敛比较快，所需要的时间比较短。

2. 调整学习率观察对准确度的影响。

其他参数为：'max_epoch': 30,'batch_size': 100,'momentum': 0.9 的条件下。

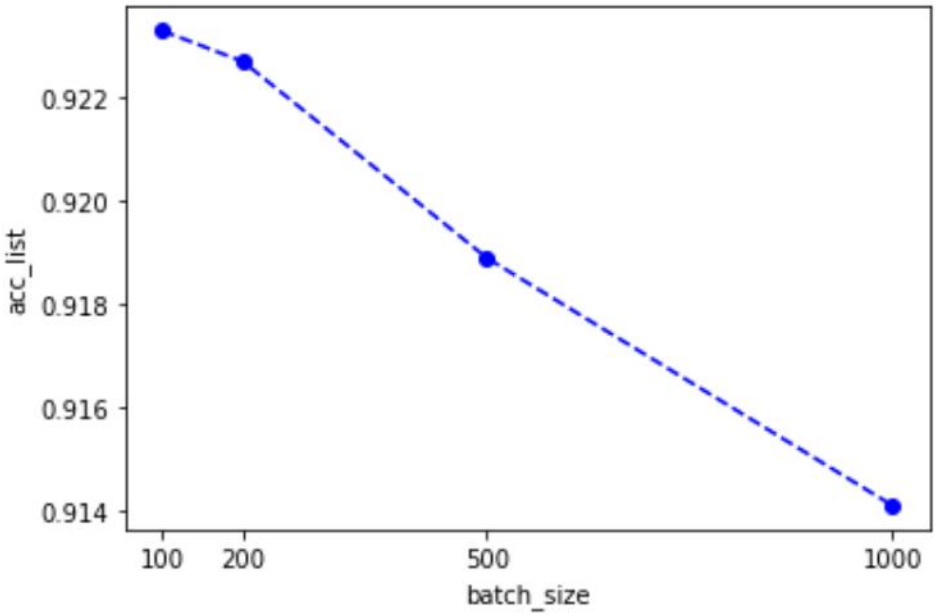


由此可见，并不是学习率越大越好，虽然学习率增大会加速梯度下降的速度，但可能错过最优解，图中学习率为 0.03，准确度最高。

3. 调整 batch_size 观察对准确度的影响。

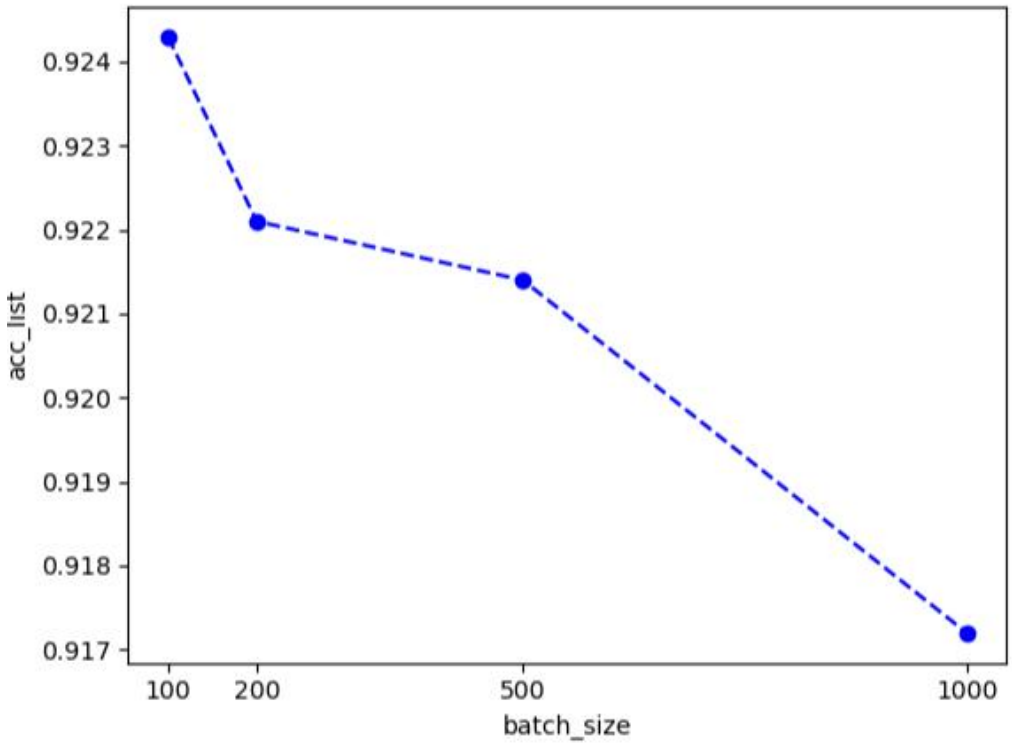
其他参数为：'max_epoch': 30,'learning_rate': 0.01,'momentum': 0.9 的条件下。

batch_size 分别为 100，200，500，1000，结果如下图所示：



有图可知，batch_size 越大，准确度越低。

将调整'max_epoch': 50，如图所示：



虽然增大 max_epoch 的数量，但是准确度依然递减。通过对比两图，可以发现增大 max_epoch 的数量，4 个测试点的准确度都有所提升。