# 实验一

## Softmax实现手写数字识别

2022年9月7日

# 1. SoftmaxCrossEntropyLoss

Forward函数用以计算cross-entropy损失和正确率。

```
input_size = len(Input)

score = np.dot(Input, self.W) + self.b   # 计算wx+b

shift_score = score - np.max(score, axis=1, keepdims=True)   # 减去最大值，避免上溢或下溢

softmax_score = np.exp(shift_score) / np.sum(np.exp(shift_score), axis=1, keepdims=True)    # 计算softmax

onehot_label = np.zeros_like(softmax_score)
onehot_label[range(input_size), labels] = 1        # 将label 转为 onehot编码

loss = np.sum(onehot_label * np.log(softmax_score)) / input_size    # 计算损失

acc = np.sum(labels == np.argmax(softmax_score, axis=1)) / input_size    # 计算正确率

# 保存用以计算dW,  dB
self.X = Input
self.labels = onehot_label
self.softmax_score = softmax_score

return loss, acc
```

softmax的计算公式为：

$$\psi(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

由于exp会出现指数增长，为了避免上下溢，可以减去最大值。

cross-entropy loss计算公式为：

$$E(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} E^{(n)}(\boldsymbol{\theta}), \quad E^{(n)}(\boldsymbol{\theta}) = -\sum_{i=1}^{K} t_i^{(n)} \ln h_i^{(n)}$$

gradient_computing函数用以计算dW和dB

```python
def gradient_computing(self):
    # 根据公式，计算dW， dB
    self.grad_W = -np.dot(self.X.T, self.labels - self.softmax_score) / len(self.X)
    self.grad_b = -np.sum(self.labels - self.softmax_score) / len(self.X)
```

$$\frac{\partial E}{\partial \boldsymbol{w}^{(k)}} = -\frac{1}{N}\sum_{n=1}^{N}\left(t_k^{(n)} - h_k(\boldsymbol{x}^{(n)})\right)\boldsymbol{x}^{(n)}$$

$$\frac{\partial E}{\partial b^{(k)}} = -\frac{1}{N}\sum_{n=1}^{N}\left(t_k^{(n)} - h_k(\boldsymbol{x}^{(n)})\right)$$

## 2. SGD

step函数用以更新W 和 B

```python
def step(self):
    """One updating step, update weights"""

    layer = self.model
    if layer.trainable:
        self.vw = self.momentum * self.vw + self.learning_rate * layer.grad_W
        self.vb = self.momentum * self.vb + self.learning_rate * layer.grad_b
        layer.W += -self.vw
        layer.b += -self.vb
```

当momentum为0时，就相当于不带动量的方法。

# 3. 实验结果

在epoch为10，batch_size 为100的情况下，不带动量的SGD，结果正确率为0.897

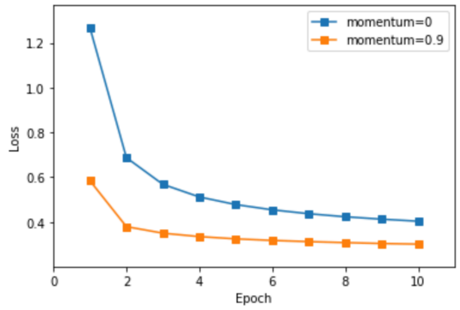
```
In [2]: # train without momentum
        cfg = {
            'data_root': 'data',
            'max_epoch': 10,
            'batch_size': 100,
            'learning_rate': 0.01,
            'momentum': 0,
            'display_freq': 50,
        }

        runner = Solver(cfg)
        loss1, acc1 = runner.train()

        Epoch [8]        Average training loss 0.4128    Average training accuracy 0.8879
        Epoch [8]        Average validation loss 0.3251  Average validation accuracy 0.9168

        Epoch [9][10]    Batch [0][550]  Training Loss 0.3346    Accuracy 0.9100

        Epoch [9][10]    Batch [50][550]         Training Loss 0.3598    Accuracy 0.9000
        Epoch [9][10]    Batch [100][550]        Training Loss 0.3013    Accuracy 0.9400
        Epoch [9][10]    Batch [150][550]        Training Loss 0.3768    Accuracy 0.9200
        Epoch [9][10]    Batch [200][550]        Training Loss 0.3453    Accuracy 0.9100
        Epoch [9][10]    Batch [250][550]        Training Loss 0.3652    Accuracy 0.9200
        Epoch [9][10]    Batch [300][550]        Training Loss 0.5387    Accuracy 0.8500
        Epoch [9][10]    Batch [350][550]        Training Loss 0.4102    Accuracy 0.9000
        Epoch [9][10]    Batch [400][550]        Training Loss 0.3020    Accuracy 0.9200
        Epoch [9][10]    Batch [450][550]        Training Loss 0.3696    Accuracy 0.9000
        Epoch [9][10]    Batch [500][550]        Training Loss 0.3250    Accuracy 0.9400

        Epoch [9]        Average training loss 0.4038    Average training accuracy 0.8899
        Epoch [9]        Average validation loss 0.3178  Average validation accuracy 0.9182

In [3]: test_loss, test_acc = runner.test()
        print('Final test accuracy {:.4f}\n'.format(test_acc))

        Final test accuracy 0.8997
```

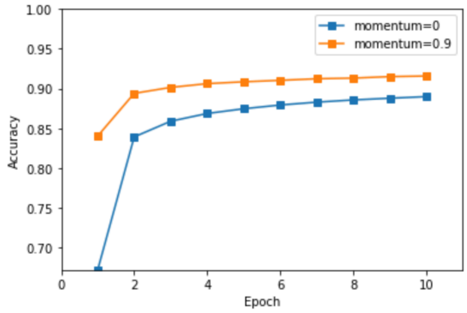
而动量为0.9的SGD，预测结果正确率为0.92

```
In [4]: # train with momentum
        cfg = {
            'data_root': 'data',
            'max_epoch': 10,
            'batch_size': 100,
            'learning_rate': 0.01,
            'momentum': 0.9,
            'display_freq': 50,
        }

        runner = Solver(cfg)
        loss2, acc2 = runner.train()

        Epoch [8]        Average training loss 0.3043    Average training accuracy 0.9151
        Epoch [8]        Average validation loss 0.2450  Average validation accuracy 0.9356

        Epoch [9][10]    Batch [0][550]  Training Loss 0.2418    Accuracy 0.9100

        Epoch [9][10]    Batch [50][550]         Training Loss 0.3256    Accuracy 0.9200
        Epoch [9][10]    Batch [100][550]        Training Loss 0.3698    Accuracy 0.8900
        Epoch [9][10]    Batch [150][550]        Training Loss 0.2077    Accuracy 0.9500
        Epoch [9][10]    Batch [200][550]        Training Loss 0.2043    Accuracy 0.9400
        Epoch [9][10]    Batch [250][550]        Training Loss 0.2886    Accuracy 0.9300
        Epoch [9][10]    Batch [300][550]        Training Loss 0.5007    Accuracy 0.8500
        Epoch [9][10]    Batch [350][550]        Training Loss 0.2290    Accuracy 0.8900
        Epoch [9][10]    Batch [400][550]        Training Loss 0.3885    Accuracy 0.9000
        Epoch [9][10]    Batch [450][550]        Training Loss 0.4523    Accuracy 0.8600
        Epoch [9][10]    Batch [500][550]        Training Loss 0.3192    Accuracy 0.9200

        Epoch [9]        Average training loss 0.3014    Average training accuracy 0.9158
        Epoch [9]        Average validation loss 0.2422  Average validation accuracy 0.9348

In [5]: test_loss, test_acc = runner.test()
        print('Final test accuracy {:.4f}\n'.format(test_acc))

        Final test accuracy 0.9208
```

画出训练损失和准确率曲线，如下：

```
In [6]: plot_loss_and_acc({
            "momentum=0": [loss1, acc1],
            "momentum=0.9": [loss2, acc2]
        })
```




可以看出，带动量的训练损失收敛速度较快，而且准确率也较高。

# 4. 参数调整

对比学习率分别为：[0.001, 0.005, 0.01, 0.05, 0.1, 0.5]的准确率。

```
In [8]: cfg = {
            'data_root': 'data',
            'max_epoch': 10,
            'batch_size': 100,
            'learning_rate': 0.01,
            'momentum': 0.9,
            'display_freq': 50,
        }
        learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1, 0.5]
        accuracys = []
        for lr in learning_rates:
            cfg['learning_rate'] = lr
            runner = Solver(cfg)
            runner.train()
            test_loss, test_acc = runner.test()
            accuracys.append(test_acc)

...

In [10]: accuracys

Out[10]: [0.8976999999999999,
          0.9169000000000002,
          0.9201,
          0.9196999999999999,
          0.9174,
          0.9013]
```

可以看出，随着学习率增加，准确率先增加，再降低。因为学习率太小，则收敛得慢。学习率太大，则损失会震荡，也无法收敛。

对比batch_size分别为：[1, 20, 50, 100, 200]的准确率。

```
In [24]: cfg = {
             'data_root': 'data',
             'max_epoch': 10,
             'batch_size': 100,
             'learning_rate': 0.01,
             'momentum': 0.9,
             'display_freq': 50,
         }
         batch_sizes = [1, 20, 50, 100, 200]
         accuracys = []
         for bz in batch_sizes:
             cfg['batch_size'] = bz
             runner = Solver(cfg)
             runner.train()
             test_loss, test_acc = runner.test()
             accuracys.append(test_acc)

                                                                 ...

In [25]: accuracys

Out[25]: [0.89, 0.9221, 0.9229, 0.9194000000000001, 0.9167000000000001]
```

batch_size决定了梯度下降的方向，当batch_size为1时，即相当于随机梯度下降算法，梯度变化波动大，损失不容易收敛。增加batch_size是，训练的时间会增加，但是准确率也会上升。增加到一定值后，再增加也不会变得更准确了。

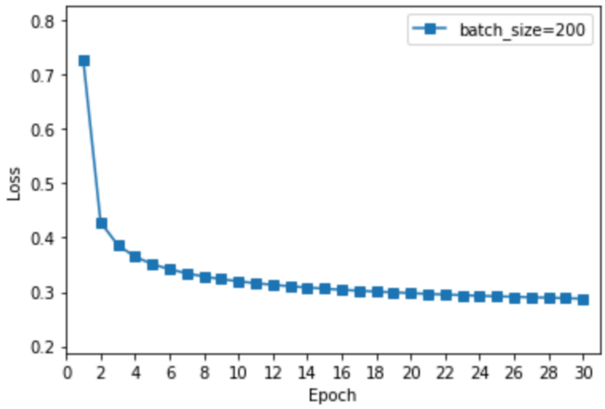
实验中batch_size为200时，准确率反而降低，原因是batch_size增加后，每个epoch迭代的次数降低了，因此10次epoch可能无法收敛，需要增加epoch次数。

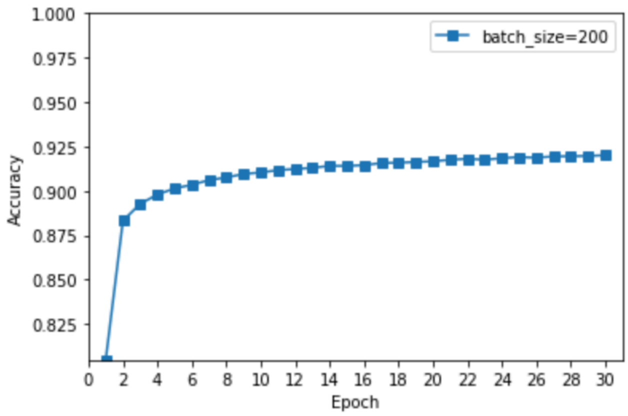
```
In [32]: cfg = {
             'data_root': 'data',
             'max_epoch': 30,
             'batch_size': 200,
             'learning_rate': 0.01,
             'momentum': 0.9,
             'display_freq': 50,
         }
         runner = Solver(cfg)
         loss3, acc3 = runner.train()
         test_loss, test_acc = runner.test()

                                                                 ...

In [35]: test_acc

Out[35]: 0.9228000000000001
```

当max_epoch为30时，准确率变成92%

画出训练损失和准确率曲线，可以看出，当batch_size=200，epoch为10时，损失函数曲线还未完全收敛。

```
In [36]: plot_loss_and_acc({
             "batch_size=200": [loss3, acc3],
         })
```




因此，在增加batch_size时，也要适当增大epoch。

# 5. 实验总结

1. 带动量的sgd方法可以加快收敛的速度。

2. 学习率过小，则收敛得慢。学习率太大，则损失会震荡，也无法收敛。

3. batch_size过小，梯度随机性较大，无法收敛。batch_size过大，需要增加epoch次数才能收敛，训练时间也很长。