multi-classification

yfpeng

October 2022

1 算法实现

1.1 OVO

由于是以感知器为基础，我直接把感知器的代码搬过来，在此基础上，加上投票的操作:

```
#下面是直接把感知器的代码搬过来
def pla(train_data,max_itera,random_init=False):
    if random_init==True:
        weight=np.random.normal(0,1,train_data.shape[1]-1)
    else:
        weight=np.array([2.,2.,2.])
    for i in range(max_itera):
        error=False
        for j in range(train_data.shape[0]):
            if np.dot(weight,train_data[j][0:-1])*train_data[j][-1]<0:
                error=True
                weight+=train_data[j][-1]*train_data[j][0:-1]
        if error==False:
            print('线性可分，找到分类面')
            return weight
        elif i==max_itera-1:
            error_rate=0
            for k in range(train_data.shape[0]):
                if np.dot(weight,train_data[k][0:-1])*train_data[j][-1]<0:
                    error_rate+=1
            print('未能找到分类面，正确率为：',1-error_rate/train_data.shape[0])
            return weight
0.4s
```

图 1: 感知器算法

测试的时候则是需要考虑三个分类器，因此编写投票的函数:

```python
#分别在三个测试集上测试
def eval(test_set1,test_set2,test_set3,w12,w13,w23):
    error=0
    for j in range(test_set1.shape[0]):
        vote=np.array([0,0,0])
        label=np.array([0,0,0])
        if np.dot(w12,test_set1[j][0:-1])>0:
            vote[0]+=1
        else:
            vote[1]+=1
        if np.dot(w13,test_set1[j][0:-1])>0:
            vote[0]+=1
        else:
            vote[2]+=1
        if np.dot(w23,test_set1[j][0:-1])>0:
            vote[1]+=1
        else:
            vote[2]+=1

        if test_set1[j][-1]==1:
            label[0]=1
        else:
            label[1]=1

        if int(np.argmax(vote))!=int(np.argmax(label)):
            error+=1
```

图 2: 测试函数-投票

1.2 softmax

通过对矩阵运算的推导，而不是像 ppt 里面那样一个个去算，我总结出了计算 softmax 和梯度的公式，实现如下：

```python
def softmax(batch,w):
    S=batch@np.transpose(w)
    Y_hat=np.exp(S)
    sum=np.sum(Y_hat,axis=-1).reshape(-1,1)
    Y_hat=Y_hat/sum
    return Y_hat
```

图 3: softmax 计算

```python
#batch为N*dimention的张量,Y_hat为N*class_num的张量，label为N*1的向量
def gradient(batch,Y_hat,label):
    #求得loss
    Y=np.eye(Y_hat.shape[-1])[np.transpose(label)]
    Y=np.squeeze(Y)
    loss=-np.sum(Y*np.log(Y_hat))/Y.shape[0]

    #求得梯度
    y=Y_hat-Y
    grad=np.transpose(y)@batch/(batch.shape[0])


    return loss,grad
```

图 4: 梯度计算

```python
def train(train_set,test_set,labels,label_test,batch_size,epochs,lr,class_num):
    steps_per_epoch=train_set.shape[0]//batch_size
    w=np.random.normal(0,0.01,(class_num,train_set.shape[-1]))
    loss_all=[]
    acc_all=[]
    acc_all_train=[]
    for epoch in range(epochs):
        epoch_loss=0.0
        acc_on_train=0
        #测试集上的精度
        predict_on_test=softmax(test_set,w)
        prediction=np.argmax(predict_on_test,axis=-1).reshape(1,-1)
        gt=np.transpose(label_test)
        test_correct=len(np.where(gt==prediction)[0])
        acc_all.append(test_correct/test_set.shape[0])

        for step in range(steps_per_epoch):
            idx=np.random.choice(train_set.shape[0],batch_size)
            batch=train_set[idx,:]
            label=labels[idx,:]
            Y_hat=softmax(batch,w)
            loss,grad=gradient(batch,Y_hat,label)

            prediction_on_train=np.argmax(Y_hat,axis=-1).reshape(1,-1)
            gt_train=np.transpose(label)
            acc_on_train+=len(np.where(gt_train==prediction_on_train)[0])

            epoch_loss+=loss
            w=w-lr*grad
        loss_all.append(epoch_loss)
        acc_all_train.append(acc_on_train/train_set.shape[0])
```

图 5: 训练函数

以上算法运行效率相当高，大概 6s 就把 mnist 跑完了

2 实验结果及讨论

2.1 OVO

OVO 不是一个太困难的数据集，但是权重我采用随机初始化，有一定偶然性，如果发现没法收敛的话，可以多运行几次，或者增加 pla 算法的迭代次数，以下是结果：

```
测试集上的正确率为:96.66666666666667%
```

图 6: ovo 结果

这个不是完全意义下的线性可分，正确率无法达到 100% 是可以理解的

2.2 softmax

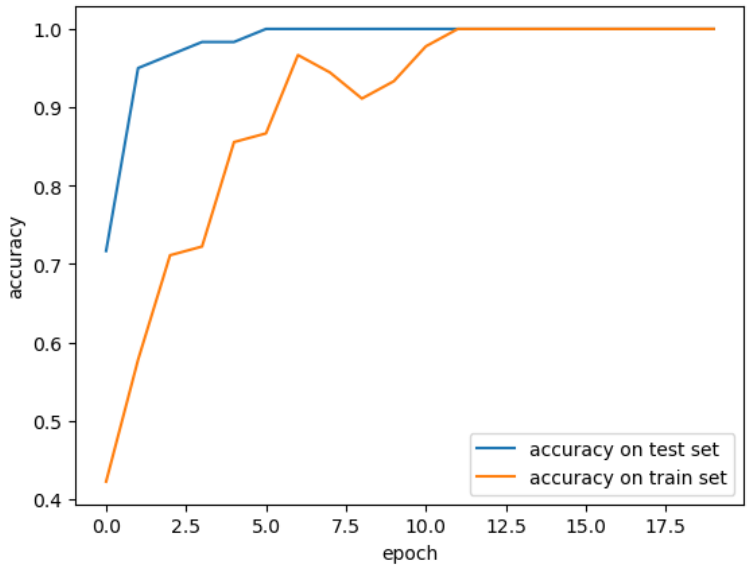

图 7: 准确率的变化

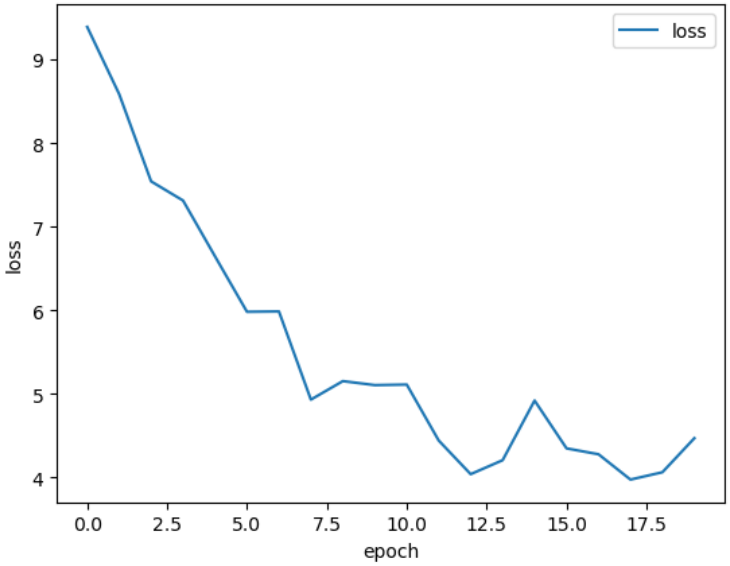

图 8: loss 的变化

上面为了体现过程，我把学习率调的很小，否则一个 epoch 就收敛了。softmax 之所以可以快速收敛，可能还是归结于它对非线性的建模能力吧

2.3 mnist

因为训练过程我是通过矩阵运算实现的，因此大概只花了 6s, 就完成了训练过程，结果如下：

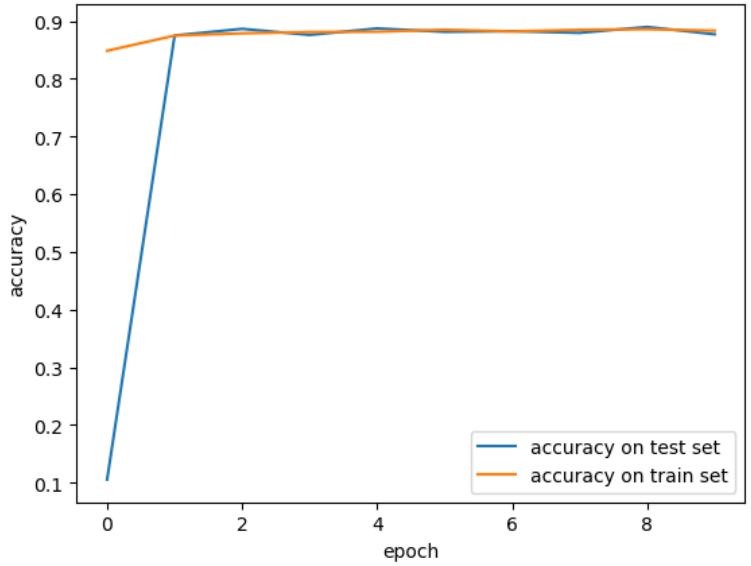

图 9: 准确率的变化

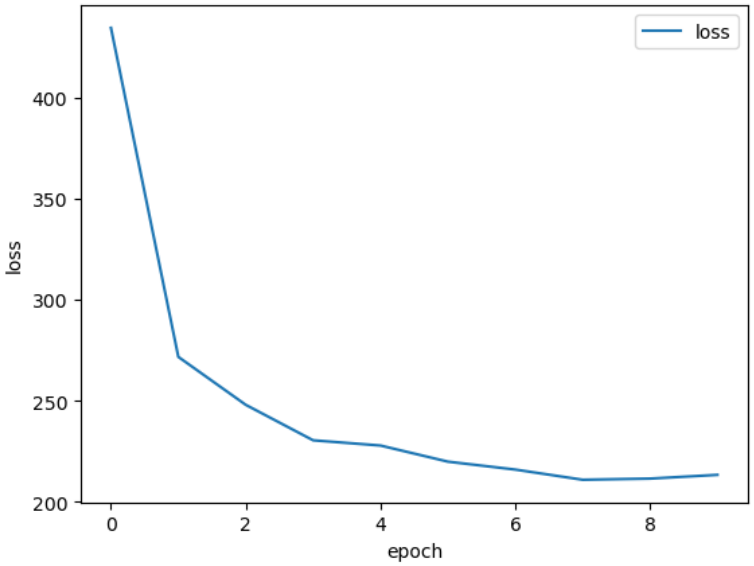

图 10: loss 的变化

比较奇怪的一点是我在采用归一化操作的时候，发现有些维度方差是

0，有点难理解。因此，我只是把均值做了归一化，没有对方差做归一化

2.4 展示

以下展示的是测试集里面随机抽取的 10 个样本，左边为 prediction, 右边为 ground-truth

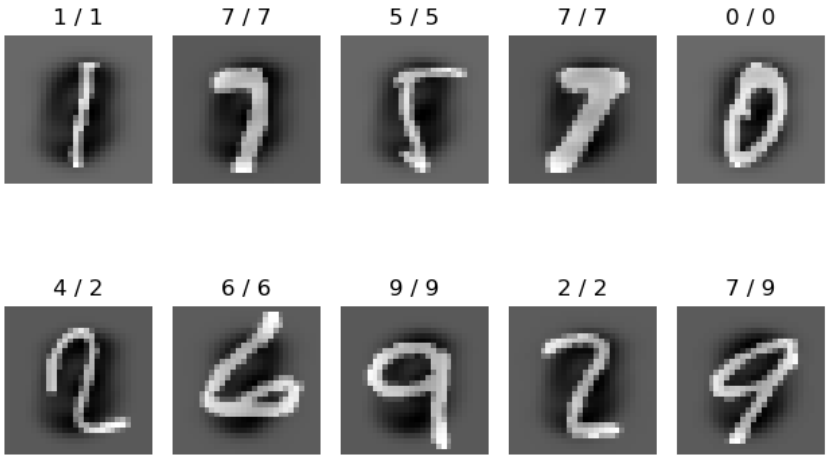

图 11: 效果展示