neural network

yfpeng

October 2022

1 关键的算法

我们用每层网络的节点数，来确定网络架构，可以随意修改网络架构：

```python
#以下是网络创建
import torch.nn as nn
class IRIS_Net(nn.Module):
    def __init__(self,hidden_layer_dims,active_type):
        super(IRIS_Net,self).__init__()
        layer_num=len(hidden_layer_dims)
        layers=[]
        self.hidden_layer_dims=[4]+hidden_layer_dims+[3]
        self.layer_num=layer_num+1

        #添加网络层
        for i in range(self.layer_num):
            layers.append(nn.Linear(self.hidden_layer_dims[i],self.hidden_layer_dims[i+1]))
            if active_type=='relu' and i!=self.layer_num-1:
                layers.append(nn.ReLU())
            elif active_type=='tanh' and i!=self.layer_num-1:
                layers.append(nn.Tanh())

        self.layers= nn.Sequential(*layers)

    #这里之所以用logits是之前在某门课程中听说先logits与softmax分开比较好
    def forward(self,x):
        logits=self.layers(x)
        return logits
```

图 1: 对 IRIS 数据集分类的网络

以下是两个网络共用的训练函数：

```python
def train(model,epochs,batch_size,data,optimizer):
    dataloader=DataLoader(data,batch_size)
    acc_on_train=[]
    acc_on_test=[]
    loss_all=[]
    for epoch in range(epochs):
        correct_train =0
        correct_test=0
        loss_per_epoch = 0.0

        with torch.no_grad():
            test_example=data.test_set.to(device)
            test_label=data.test_label.to(device)
            logits_on_test=model(test_example)
            pred_on_test=torch.argmax(logits_on_test,-1)
            correct_test=int((pred_on_test== test_label).sum().int().cpu())
            acc_on_test.append(correct_test/test_label.shape[1])

        for batch, (X, y) in enumerate(dataloader):
            X=X.to(device)
            y=y.long().to(device)
            logits=model(X)
            loss=F.cross_entropy(logits,y)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_per_epoch+=float(loss.cpu())
            pred_on_train=torch.argmax(logits,-1)
            correct_train += int((pred_on_train== y).sum().int().cpu())

        acc_on_train.append(correct_train/data.train_set.shape[0])
        loss_all.append(loss_per_epoch)
```

图 2: 训练函数

根据 word 作业确定的网络架构:

```
#以下是Lenet
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet,self).__init__()
        self.conv1=nn.Conv2d(1,6,5,1,2)
        self.ac1=nn.Sigmoid()
        self.av1pool=nn.AvgPool2d(2,2)
        self.conv2=nn.Conv2d(6,16,5,1,0)
        self.ac2=nn.Sigmoid()
        self.av2pool=nn.AvgPool2d(2,2)
        self.flatten=nn.Flatten()
        self.fc1 = nn.Linear(400,120)
        self.ac3=nn.Sigmoid()
        self.fc2 = nn.Linear(120,84)
        self.ac4=nn.Sigmoid()
        self.fc3=nn.Linear(84,10)
        self._initialize()
    def forward(self,x):
        (variable) pool1_out: Any v1(x))
        pool1_out=self.av1pool(conv1_out)
        conv2_out=self.ac2(self.conv2(pool1_out))
        pool2_out=self.av2pool(conv2_out)
        flat=self.flatten(pool2_out)
        fc1_out=self.ac3(self.fc1(flat))
        fc2_out=self.ac4(self.fc2(fc1_out))
        logits=self.fc3(fc2_out)

        return logits

    def _initialize(self):#初始化权重
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.xavier_uniform_(m.weight)
                if m.bias is not None:
                    nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
```

图 3: LeNet

2 实验结果及讨论

2.1 IRIS 实验结果

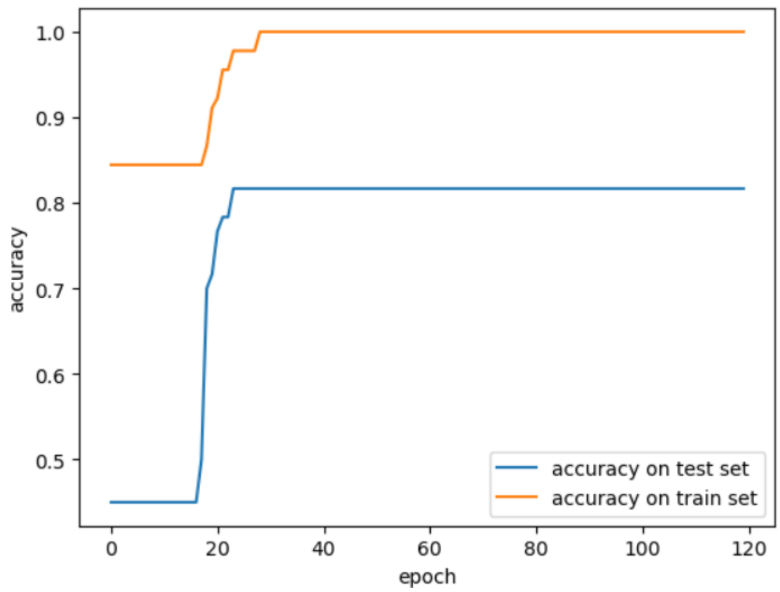

图 4: 正确率随 epoch 变化情况

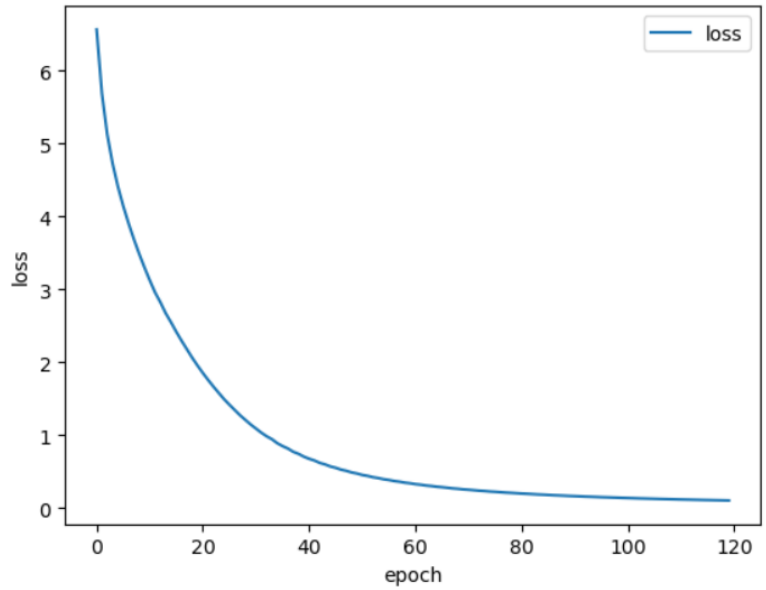

图 5: loss 随 epoch 变化情况

其收敛速度还是比较慢的，大概 18 个 epoch 才收敛，正确率只能达到 80% 左右

2.2 增加隐藏层数

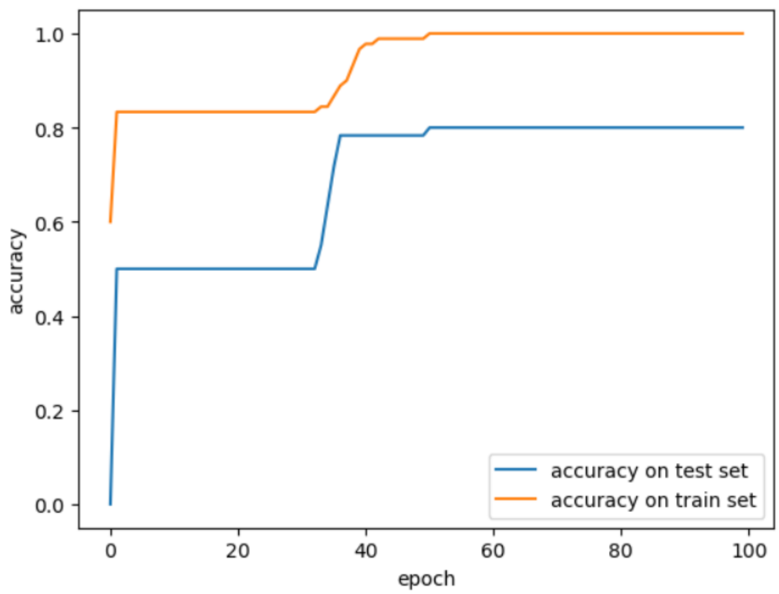

图 6: 正确率随 epoch 变化情况

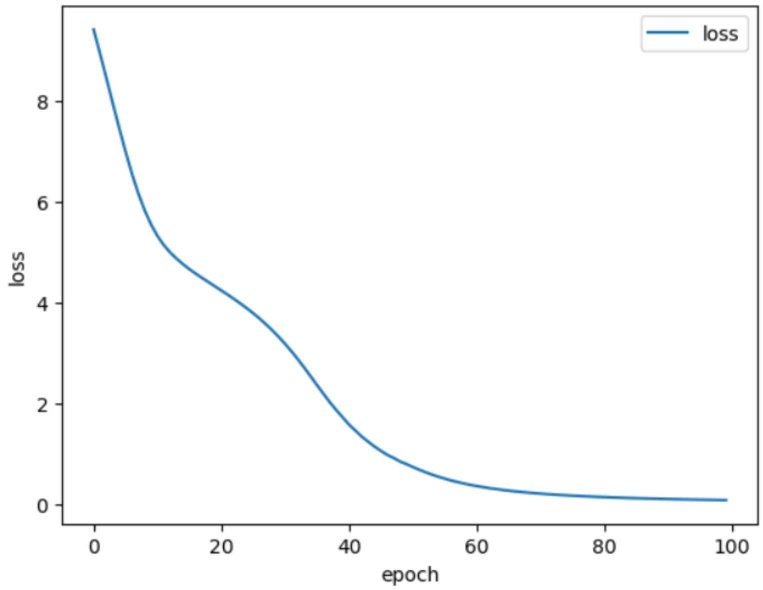

图 7: loss 随 epoch 变化情况

我们发现前几个 epoch 收敛非常快，但是中间有相当长一段时间不动，可能是陷入局部极值点

2.3 增加节点

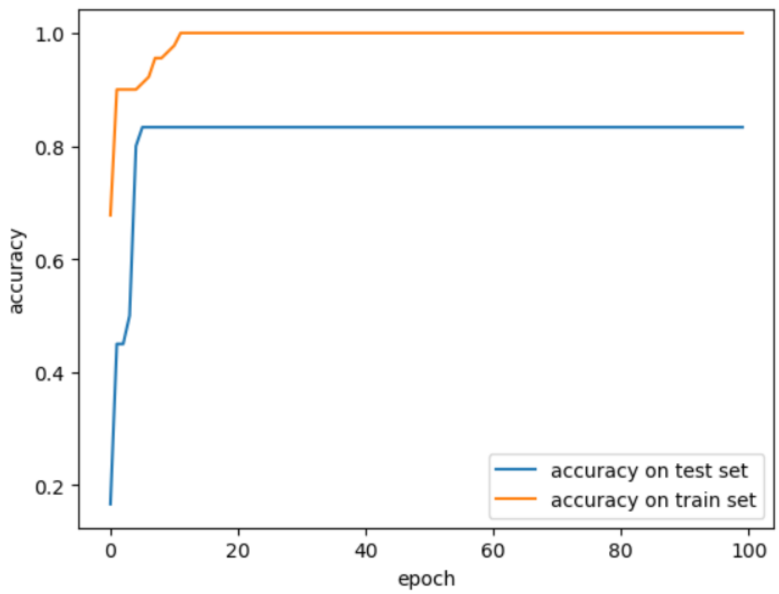

图 8: 正确率随 epoch 变化情况

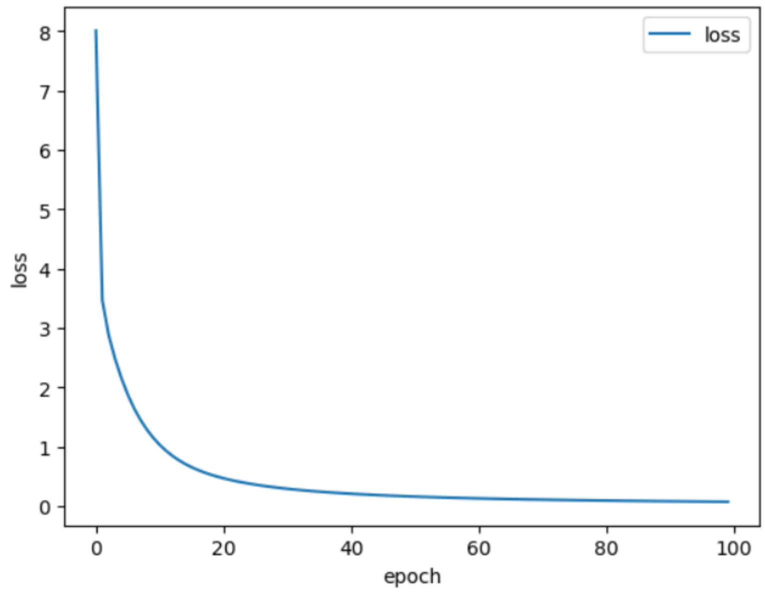

图 9: loss 随 epoch 变化情况

我们发现，增加节点的效果几乎是最好的，收敛快，正确率也高，这也许是数据维度比较低，信息比较少的原因

2.4 改变学习率

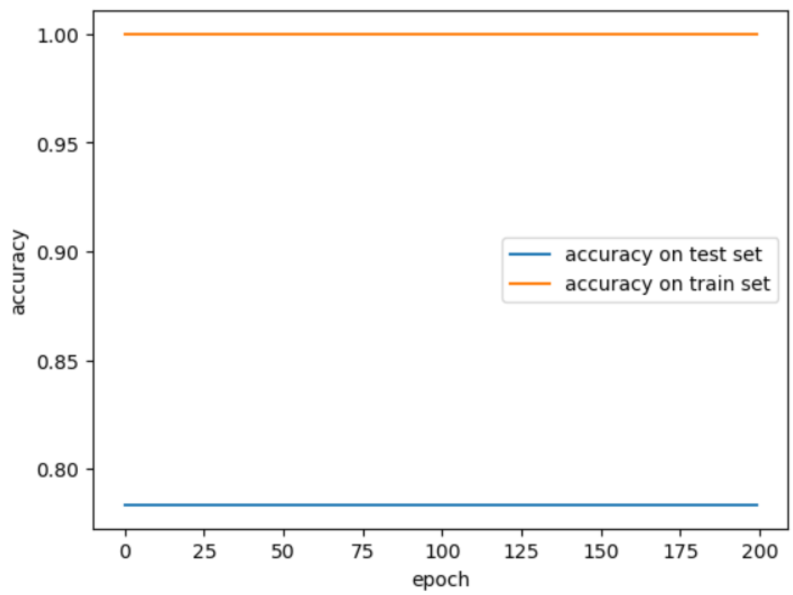

图 10: 正确率随 epoch 变化情况

改变学习率不仅没起到作用，反而陷入局部极值

2.5 改变优化器

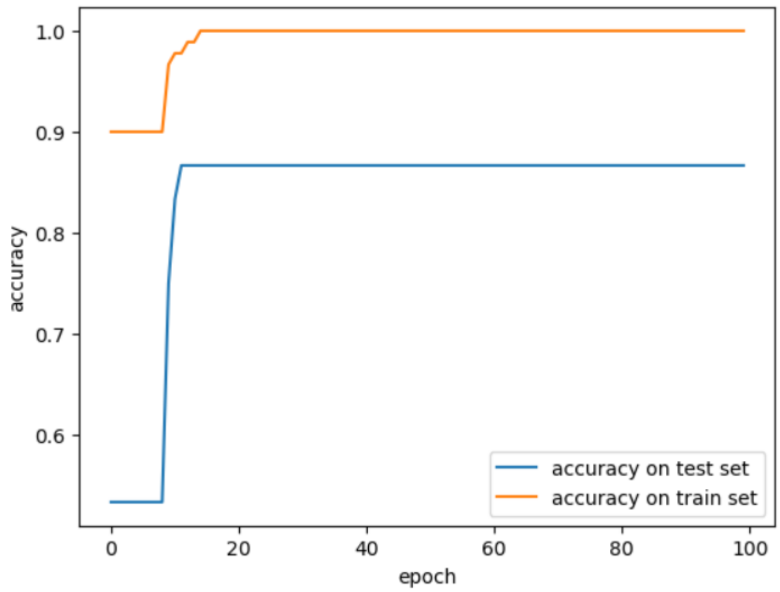

图 11: 正确率随 epoch 变化情况

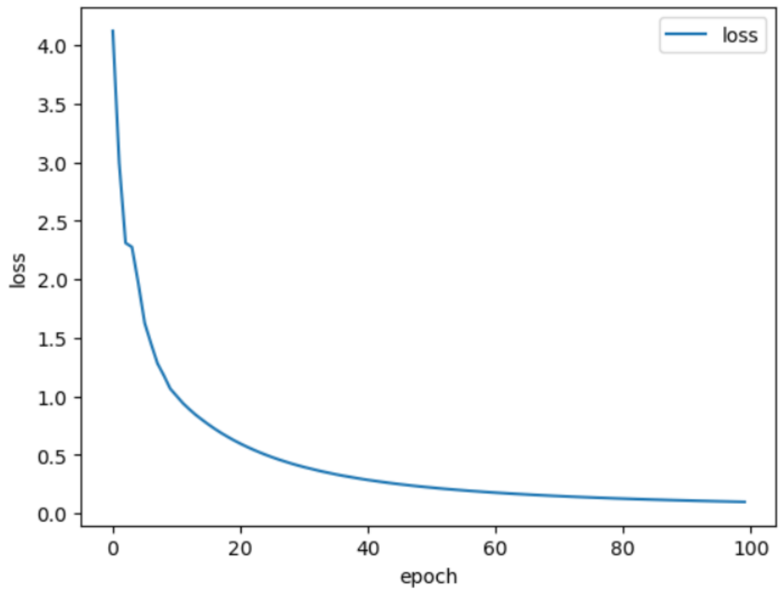

图 12: loss 随 epoch 变化情况

改变优化器的效果也相当好，这里仅仅是加了一点动量，不仅收敛加快，而且效果也提升了，这应该是帮助模型跳出局部极值点的结果

2.6 更换激活函数

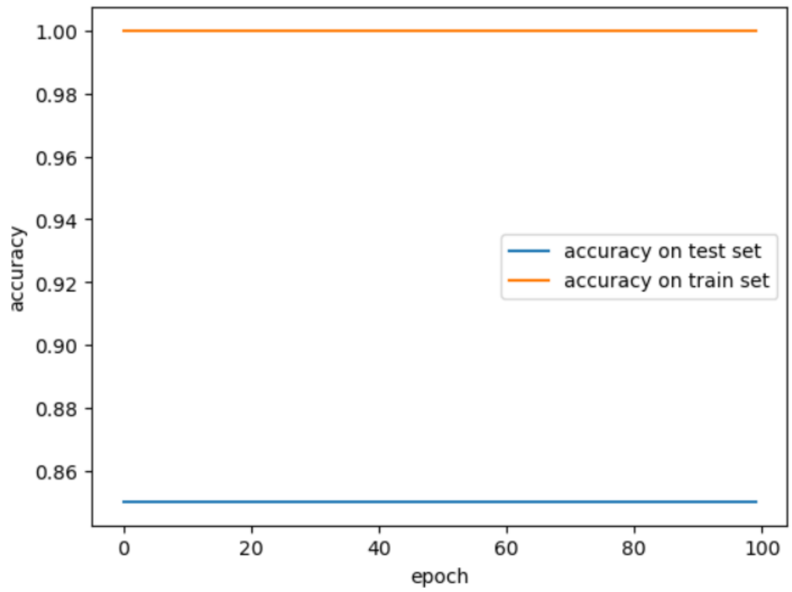

图 13: 正确率随 epoch 变化情况

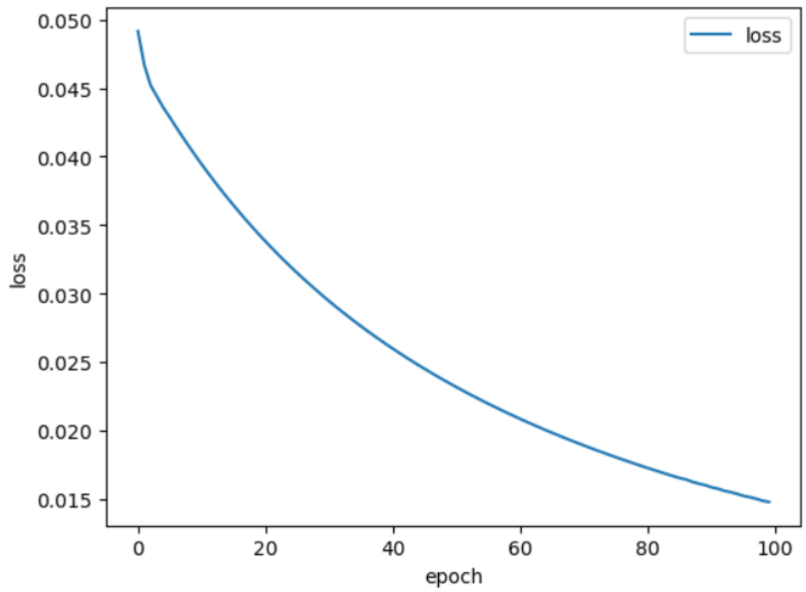

图 14: loss 随 epoch 变化情况

改变激活函数不仅没起到作用，反而陷入局部极值

2.7 LeNet 实验结果

按老师给的超参，把这个调收敛不是一件太简单的事，经过多次尝试，我采用了随机初始化和改用 Adam 优化器，得到一个相对较好的结果，如下：

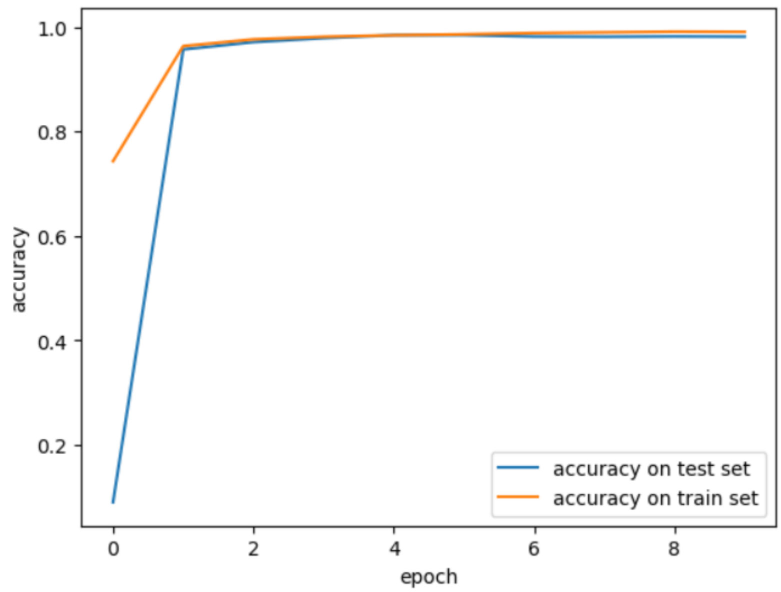

图 15: 正确率随 epoch 变化情况

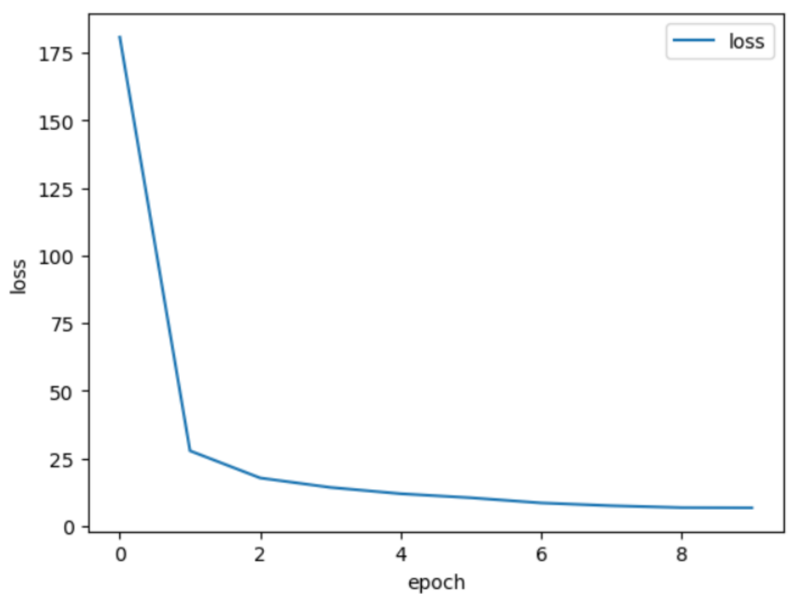

图 16: loss 随 epoch 变化情况

3 展示

这里我用 numpy 随机抽取了 10 个不重复下标，进行测试，结果如下：

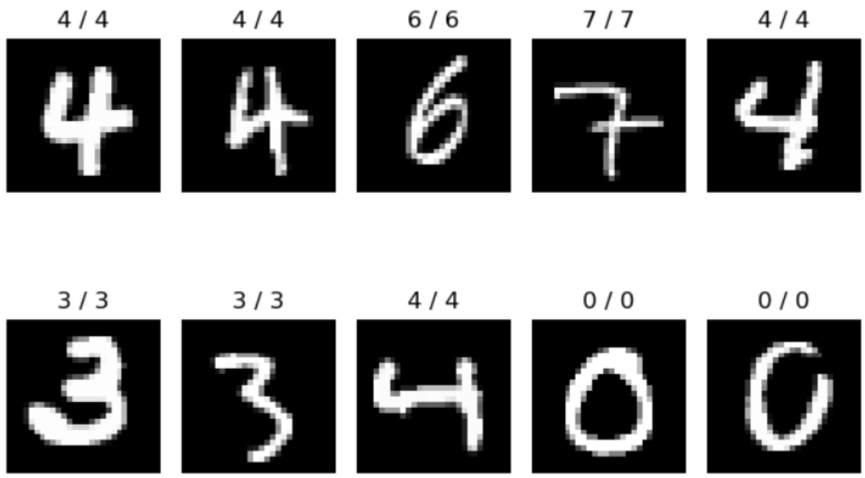

图 17: 结果展示

可以看见，抽到的 10 个全部分类正确，可见 LeNet 还是很强大的