机器学习第五章作业

任齐轩
重庆大学-计卓 2 班-20204154

日期：October 14, 2022

1 第一题

该线性函数无法为神经网络增加非线性，对于输入 x，经过 f(x) 后，本质上还是在进行线性变换。因此，当神经网络的层数增加时，其表达能力不会增加，仍然是线性的，无法解决非线性问题，达不到激活函数的目的。

2 第二题

Sigmoid 函数本质上与对率回归解决的问题类似，都是将输入映射到 0-1 之间。区别在于对率回归一般应用于二分类问题，将输出值与一个既定的阈值如 0.5 进行比较，依据此来判断结果为 0 还是 1；而 Sigmoid 函数将输入值映射到 0-1 之间，可以当作是概率值。

3 第三题

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h}\frac{\partial b_h}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h}\frac{\partial b_h}{\partial a_h}\frac{\partial a_h}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h}\frac{\partial b_h}{\partial a_h}x_i = -e_h x_i \tag{1}$$

即：

$$\frac{\partial E_k}{\partial v_{ih}} = -e_h x_i \tag{2}$$

对于给定学习率 η，则有：

$$\triangle v_{ih} = -\eta\frac{\partial E_k}{\partial v_{ih}} = \eta e_h x_i \tag{3}$$

即为书中 5.13 式。

4 第四题

学习率对每一轮迭代时更新步长的大小有影响，若学习率取值过大，可能会导致梯度下降过快，使得最终的结果不是全局最优解，而是局部最优解。若学习率取值过小，可能会导致梯度下降过慢，使得最终的结果不是全局最优解，而是局部最优解。

5 第十题

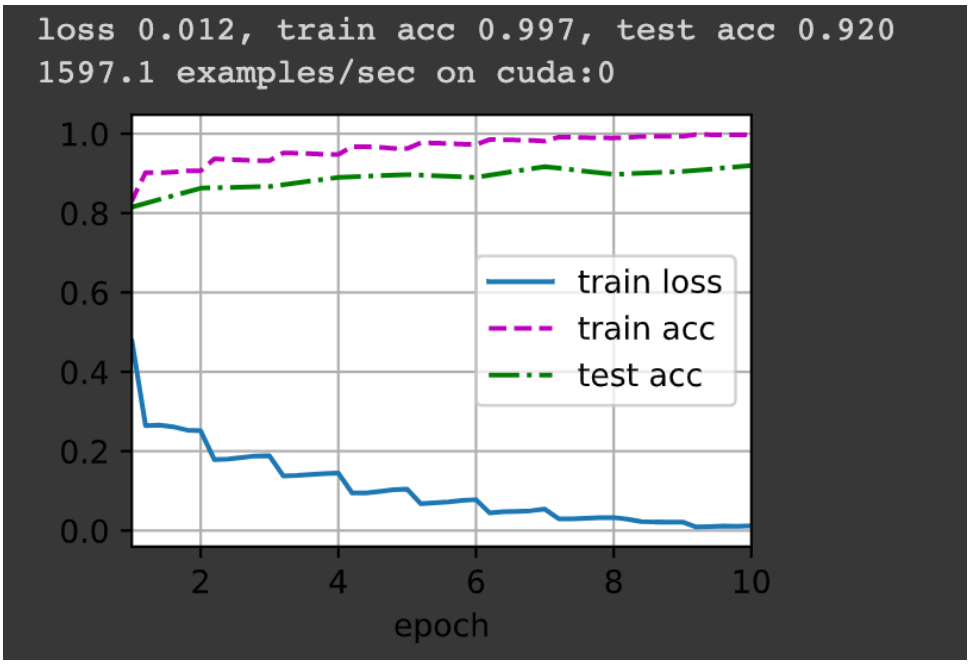

图 1: 实验测试结果

```
from d2l import torch as d2l
import torch
import torch.nn as nn
import torch.nn.functional as F

class residual_block(nn.Module):
    def __init__(self, in_channels, num_channels, use_1x1conv=False, strides=1)
        :
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, num_channels, stride=strides,
            padding=1, kernel_size=3)
        self.conv2 = nn.Conv2d(num_channels, num_channels, kernel_size=3,
            padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d(in_channels, num_channels, kernel_size=1,
                stride=strides)
        else:
            self.conv3 = None
        self.bn1 = nn.BatchNorm2d(num_channels)
        self.bn2 = nn.BatchNorm2d(num_channels)

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
```

```
        Y += X
        Y = F.relu(Y)
        return Y

def blk1(in_channels, out_channels):
    blk = nn.Sequential(nn.Conv2d(in_channels, out_channels, kernel_size=7,
        stride=2, padding=3),
                        nn.BatchNorm2d(out_channels), nn.ReLU(),
                        nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
                        )
    return blk

def resnet_blk(in_channels, out_channels, num_blks, first_blk=False):
    blk = []
    for i in range(num_blks):
        if i == 0 and not first_blk:
            blk.append(residual_block(in_channels, out_channels, use_1x1conv=
                True, strides=2))
        else:
            blk.append(residual_block(out_channels, out_channels))
    return blk

block1 = blk1(1, 64)
block2 = nn.Sequential(*resnet_blk(64, 64, 2, first_blk=True))
block3 = nn.Sequential(*resnet_blk(64, 128, 2))
block4 = nn.Sequential(*resnet_blk(128, 256, 2))
block5 = nn.Sequential(*resnet_blk(256, 512, 2))
net = nn.Sequential(block1, block2, block3, block4, block5,
                    nn.AdaptiveAvgPool2d((1,1)),
                    nn.Flatten(), nn.Linear(512, 10)
                    )

lr, num_epochs, batch_size = 0.05, 10, 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size, resize=96)
d2l.train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```