计算机视觉作业 5

张瀚文 2201212865

1. 作业要求

· 补全两层全连接代码 W5_Homework.ipynb

· 给出变量 W1,b1,W2,b2 导数表达式

2. 理论推导

已知：

$$h = XW_1 + b_1$$
$$h_{sigmoid} = sigmoid(h)$$
$$Y_{pred} = h_{sigmoid}W_2 + b_2$$
$$f = \left\|Y - Y_{pred}\right\|_F^2$$

需要推导：

$$\frac{\partial f}{\partial w1}, \frac{\partial f}{\partial w2}, \frac{\partial f}{\partial b1}, \frac{\partial f}{\partial b2}$$

2.1 Sigmoid 函数的求导

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

$$S'(x) = \frac{d}{dx}(1+e^{-x})^{-1} = (-1)\times(1+e^{-x})^{-2}(-e^{-x})$$
$$= (1+e^{-x})^{-1}(1-(1+e^{-x})^{-1}) = S(x)(1-S(x))$$

2.2 反向传播公式的推导

$$f = \left\|Y - Y_{pred}\right\|_F^2 \rightarrow f = \|Y - (S(XW_1 + b_1)\cdot W_2 + b_2)\|_F^2$$

$$df = d(tr((Y-Y_p)^T(Y-Y_p)))$$

$$= tr(d(Y-Y_p)^T(Y-Y_p) + (Y-Y_p)^T d(Y-Y_p)) = (-2)tr((Y-Y_p)dY_p)$$

$$\frac{\partial f}{\partial b_2} = -2(Y-Y_p)^T$$

$$\frac{\partial f}{\partial w_2} = -2h_s^T(Y-Y_p)$$

$$df = -2tr(\left((Y-Y_p)w_2{}^T\right)^T S(h)\odot(1-S(h))\odot dh)$$

$$\frac{\partial f}{\partial b_1} = -2(Y-Y_p)w_2^T \odot h_s \odot (1-h_s)$$

$$\frac{\partial f}{\partial w_1} = -2X^T\left((Y-Y_p)w_2^T \odot h_s \odot (1-h_s)\right)$$

3. 实验过程

3.1 构建数据集

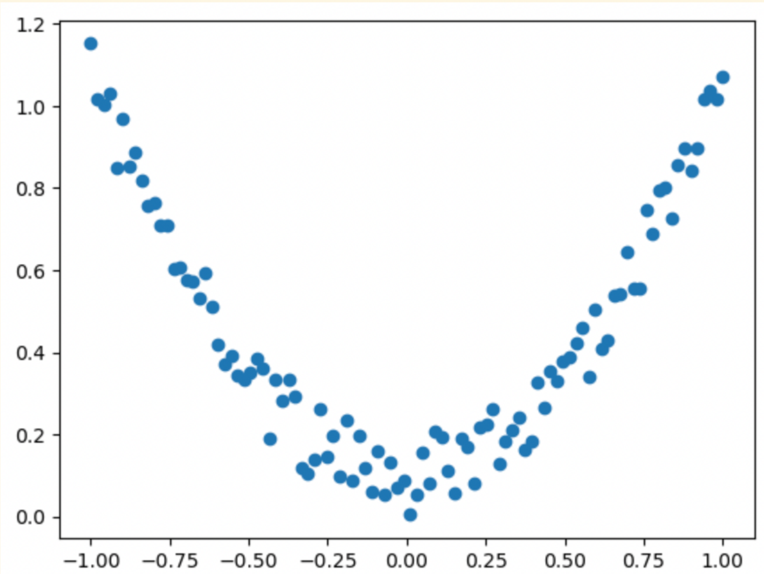
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt


torch.manual_seed(1)    # reproducible

x = torch.unsqueeze(torch.linspace(-1, 1, 100), dim=1)  # x data (tensor), shape=(100, 1)
y = x.pow(2) + 0.2*torch.rand(x.size())
```

[1]

```python
plt.scatter(x.numpy(), y.numpy())
```

[2]

```
<matplotlib.collections.PathCollection at 0x7fb8b13ee4f0>
```



3.2 搭建两层全连接网络，隐藏层输出个数为 20，激活函数为 Sigmoid 函数

```python
class Net(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.W1 = nn.Parameter(nn.init.xavier_normal_(torch.Tensor(n_feature, n_hidden)))
        self.W2 = nn.Parameter(nn.init.xavier_normal_(torch.Tensor(n_hidden, n_output)))
        self.b1 = nn.Parameter(nn.init.xavier_normal_(torch.Tensor(100, n_hidden)))
        self.b2 = nn.Parameter(nn.init.xavier_normal_(torch.Tensor(100, n_output)))

    def forward(self, x):
        outlayer = nn.Sigmoid() # 需要先对类实例化，才能调用
        y_pred = outlayer(x.mm(self.W1)+self.b1).mm(self.W2)+self.b2
        return y_pred
```

[23]

3.3 实例化神经网络模型

```python
n_feature, n_hidden, n_output = 1, 20, 1
net = Net(n_feature, n_hidden, n_output)     # define the network
print(net)  # net architecture
optimizer = torch.optim.SGD(net.parameters(), lr=0.2)
loss_func = torch.nn.MSELoss()  # this is for regression mean squared loss

plt.ion()   # something about plotting
```

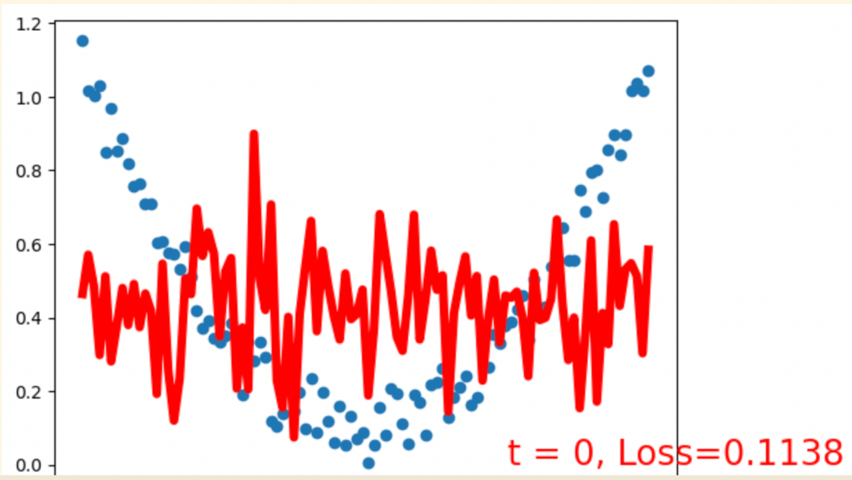
[24]

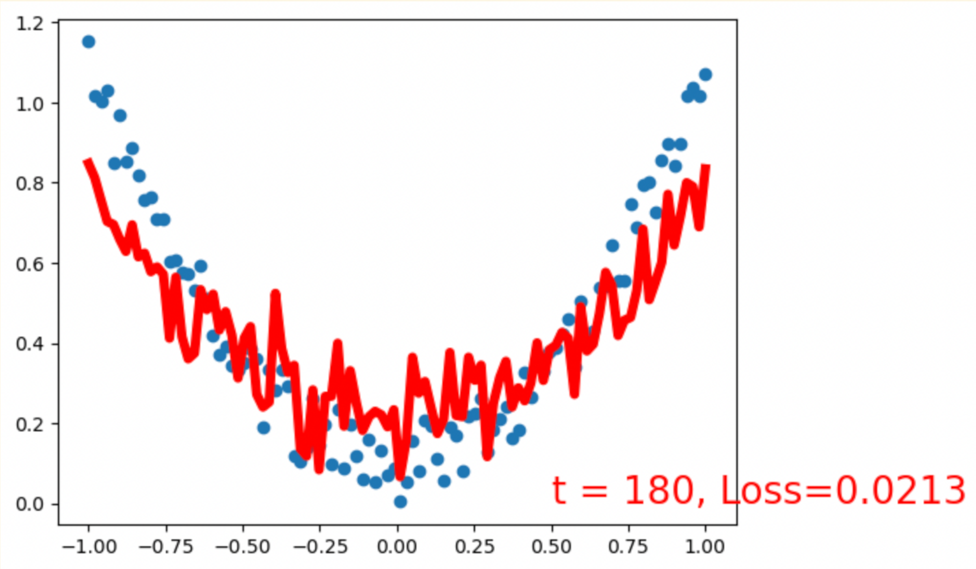
```
Net()
```

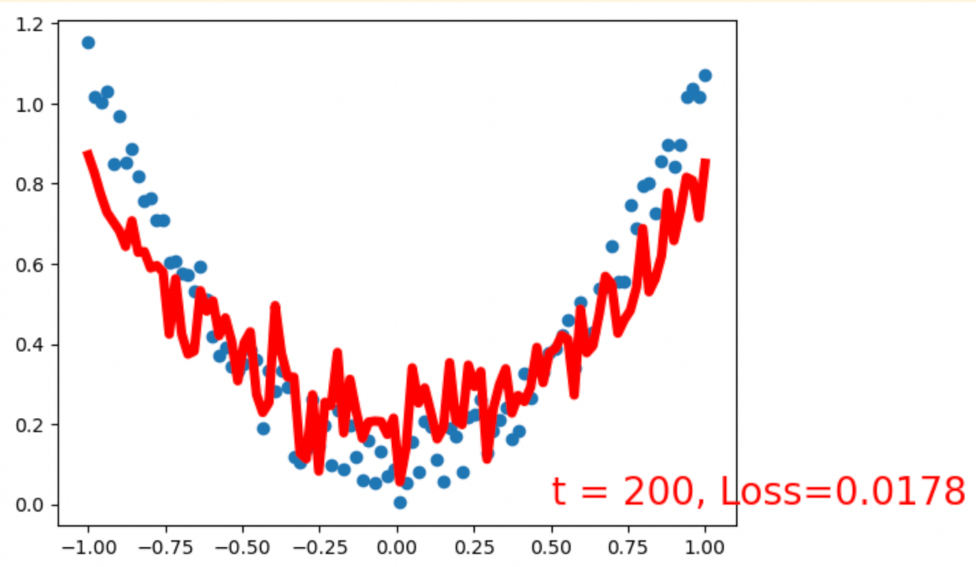
3.4 训练神经网络，输出结果并作图

```python
for t in range(201):
    prediction = net(x)     # input x and predict based on x
    loss = loss_func(prediction, y)     # must be (1. nn output, 2. target)

    optimizer.zero_grad()   # clear gradients for next train
    loss.backward()         # backpropagation, compute gradients
    optimizer.step()        # apply gradients

    if t % 20 == 0:
        # plot and show learning process
        plt.cla()
        plt.scatter(x.numpy(), y.numpy())
        plt.plot(x.numpy(), prediction.data.numpy(), 'r-', lw=5)
        plt.text(0.5, 0, 't = %d, Loss=%.4f' % (t, loss.data.numpy()), fontdict={'size': 20, 'color':  'red'})
        plt.pause(0.1)
        plt.show()

plt.ioff()
# plt.show()
```

[25]




1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
t = 180, Loss=0.0213


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
t = 200, Loss=0.0178