計算機視覺作業

干皓丞，2101212850, 信息工程學院

2021 年 10 月 23 日

1 題目

Pytorch 搭建兩層全連接網路

```python
torch.manual_seed(1) # reproducible
x = torch.unsqueeze(torch.linspace(-1, 1, 100), dim=1)
y = x.pow(2) + 0.2*torch.rand(x.size())
```

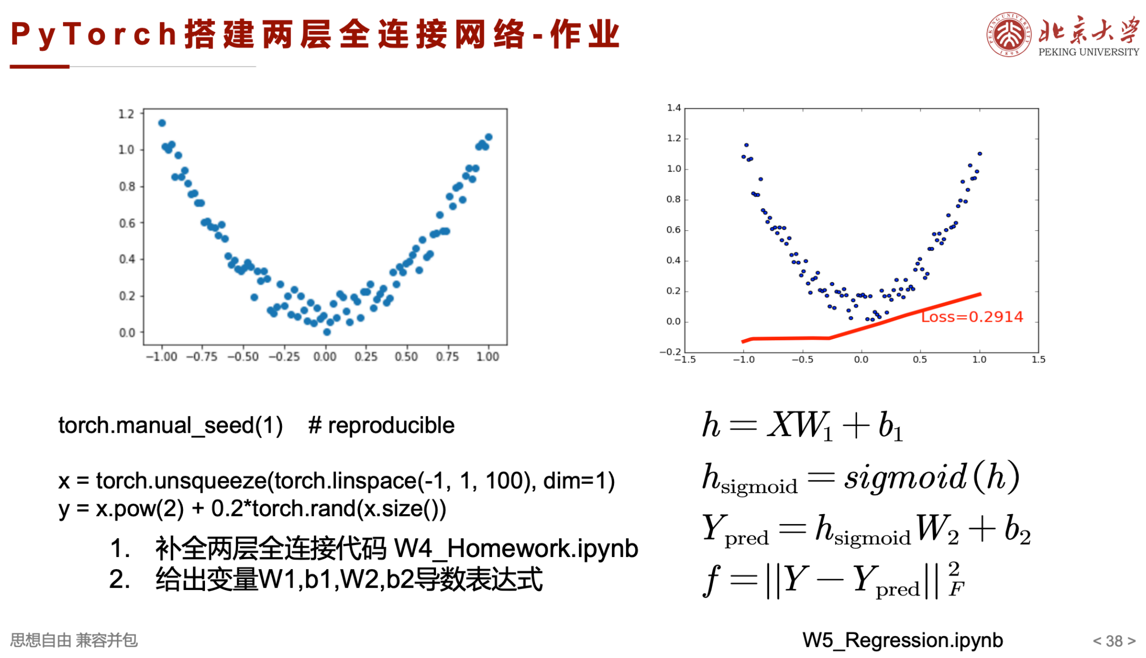


Fig. 1. 作業目標

1. 補全兩層全連接代碼 W4_Homework.ipynb
2. 給出變量 W1, b1, W2, b2 導數表達式

目標函數:

$$f = ||\,Y - Y_{pred}||_F^2 \quad (1.1)$$

$$h = XW_1 + b_1 \quad (1.2); \quad h_{sigmoid} = sigmoid(h) \quad (1.3); \quad Y_{pred} = h_{sigmoid}W_2 + b_2 \; (1.4)$$

手推寫出已下表達式，並用 Pytorch 進行實現，在此為了方便表示則省略表達 Y_{pred} 為 Y_p ，$h_{sigmoid}$ 則為 h_s ，而 $S()$ 即為 Sigmoid 函數，最後數學式 (1.1) 又可以表達如 (1.5)。

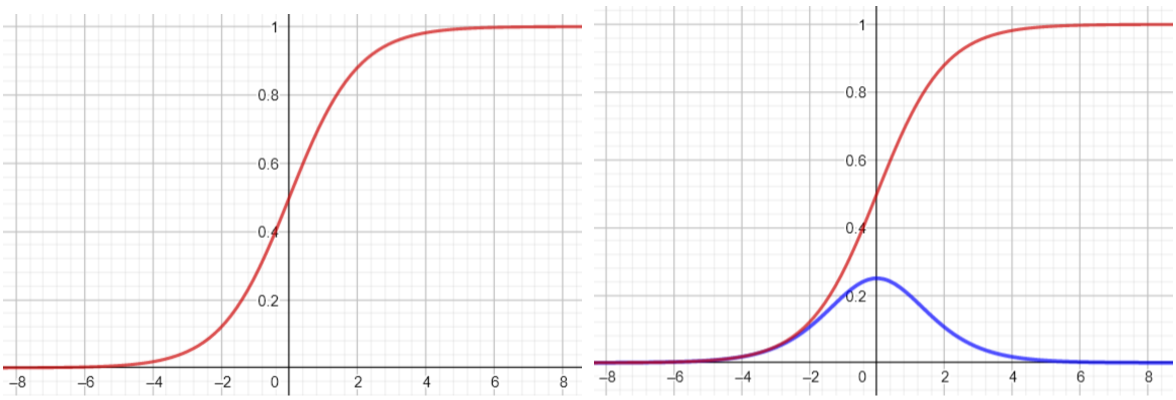
$$f = ||\, Y - \; (\; S(\; XW_1 + b_1).W_2 + b_2)||_F^2 \qquad (1.5)$$

$$(1) \quad \frac{\partial f}{\partial w1} \quad (2) \quad \frac{\partial f}{\partial b1} \quad (3) \quad \frac{\partial f}{\partial w2} \quad (4) \quad \frac{\partial f}{\partial b2}$$

2 數學式定義與程式碼的數學意義說明

(1) Sigmoid 函數

Sigmoid 函數與函數自身求導的圖形如下所示，紅線為 Sigmoid 函數，藍線為 Sigmoid 函數求導後的函數圖形。



(i) Sigmoid 函數 (ii) Sigmoid 函數求導與原函數對比

Fig. 2. Sigmoid 函數狀態

(2) Sigmoid 定義

Sigmoid 定義如數學式 (2.1) :

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x) \qquad (2.1)$$

(3) Sigmoid 求導與推導過程:

Sigmoid 求導為數學式 (2.2) ，而該函數求導過程則詳見數學式 (2.3) :

$$S'(x) = \; S(x)(1 - S(x)) \qquad (2.2)$$

$$\frac{d}{dx}\rho(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}\;\left[1+e^{-x}\right]^{-1} = -1 \times \left(1+e^{-x}\right)^{-2}\left(-e^{-x}\right)$$

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

$$= \frac{1}{1+e^{-x}} \times \frac{(1+e^{-x})-1}{1+e^{-x}} = \frac{1}{1+e^{-x}}\left[\frac{(1+e^{-x})}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right]$$

$$= \frac{1}{1+e^{-x}}\left[1-\frac{1}{1+e^{-x}}\right] = \rho(x)(1-\rho(x)) \tag{2.3}$$

(4) 程式碼中的數學意義

```
import torch
import numpy as np
torch.manual_seed(0)

x = torch.randn(100, 1, requires_grad=True)
y = torch.randn(100, 1, requires_grad=True)
w1 = torch.randn(1, 20, requires_grad=True)
w2 = torch.randn(20, 1, requires_grad=True)
b1 = torch.randn(100, 20, requires_grad=True)
b2 = torch.randn(100, 1, requires_grad=True)
print( "x : ", np.shape(x))
print( "y : ", np.shape(y))
print( "w1 : ", np.shape(w1))
print( "w2 : ", np.shape(w2))
print( "b1 : ", np.shape(b1))
print( "b2 : ", np.shape(b2))
```

Fig. 3. Pytorch 矩陣

程式碼當中的 x、y、w1、w2、b1、b2 在數學上分別代表了六個矩陣，x、y、b1 皆為 100 X 1 大小的矩陣，w1 矩陣為 1 X 20 大小的矩陣，w2 矩陣為 20 X 1 大小的矩陣，b1 矩陣為 100 X 20 大小的

矩陣，而 Pytorch 則會隨機產生矩陣中的值，其數學表達如下。

$$X = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{100\ 1} \end{bmatrix},\ Y = \begin{bmatrix} y_{11} \\ y_{21} \\ \vdots \\ y_{100\ 1} \end{bmatrix},\ b1 = \begin{bmatrix} b1_{11} \\ b1_{21} \\ \vdots \\ b1_{100\ 1} \end{bmatrix}$$

$$w1 = \begin{bmatrix} w1_{11} & \dots & w1_{1\ 20} \end{bmatrix},\ w2 = \begin{bmatrix} w2_{11} \\ \vdots \\ w2_{20\ 1} \end{bmatrix},\ b1 = \begin{bmatrix} b1_{11} & \dots & b1_{1\ 20} \\ \vdots & \ddots & \vdots \\ b1_{100\ 1} & \dots & b1_{100\ 20} \end{bmatrix} \quad (2.4)$$

3 數學推導證明

$$f = ||\, Y - \ Y_{pred}||_F^2 \rightarrow f = ||\, Y - \ (\ S(\ XW_1 + b_1).W_2 + b_2)||_F^2 \quad (3.1)$$

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

$$tr(2(Y - Y_p)^T d(Y - Y_p)) \ = \ -2tr((Y - Y_p)^T dY_p) \quad (3.2)$$

$$\therefore \quad df = -2tr((Y - Y_p)^T db_2)$$

$$\therefore \quad \frac{\partial f}{\partial b_2} = -2(Y - Y_p)^T \quad (3.3)$$

$$df = -2tr((Y - Y_p)^T.h_s.dw_2)$$

$$\therefore \quad \frac{\partial f}{\partial w_2} = -2h_s^T.(Y \ - \ Y_p) \quad (3.4)$$

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

$$= -2tr\left(\left((Y - Y_p).w_2^T \odot h_s \odot (1 - h_s)\right)^T .dh\right)$$

$$\because \ dh = db_1$$

$$\therefore \quad \frac{\partial f}{\partial b_1} \ = \ -2(Y - Y_p)w_2^T \odot h_s \odot (1 - h_s) \quad (3.5)$$

$$\because \ dh = Xdw_1$$

$$\therefore \quad df = -2tr(((Y - Y_p)w_2^T \odot h_s \odot (1 - h_s))^T .x.dw_1)$$

$$\therefore \quad \frac{\partial f}{\partial w_1} = -2X^T((Y - Y_p)w_2^T \odot h_s \odot (1 - h_s)) \quad (3.6)$$

4 Pytorch 程式碼實現

程式碼可以在 GitHub 專案 (kancheng/kan-cs-report-in-2021/CV/pytorch-sigmoid) 找到，詳見 sigmoid-math.ipynb 檔案，而範例結果可以從相對應的 PDF 檔案 sigmoid-math.pdf 找到，最後可以發現公式解與程式解兩者結果一致。

(1) Pytorch 實驗資料

下列為 Pytorch 所產生矩陣實驗資料，包含直接求導與手動公式求導，最後會發現兩者結果一致。

```
# 給定實驗資料
import torch
import numpy as np
torch.manual_seed(0)

x = torch.randn(100, 1, requires_grad=True)
y = torch.randn(100, 1, requires_grad=True)
w1 = torch.randn(1, 20, requires_grad=True)
w2 = torch.randn(20, 1, requires_grad=True)
b1 = torch.randn(100, 20, requires_grad=True)
b2 = torch.randn(100, 1, requires_grad=True)
print("x : ", np.shape(x))
print("y : ", np.shape(y))
print("w1 : ", np.shape(w1))
print("w2 : ", np.shape(w2))
print("b1 : ", np.shape(b1))
print("b2 : ", np.shape(b2))

# 數學式
import torch.nn as nn
tm = nn.Sigmoid()
hs = tm(x.mm(w1)+b1)
# print(s)
yp = (hs).mm(w2)+b2
f1 = (y - yp).pow(2).sum()
ft = (y - yp).pow(2)
print(f1)
```

```
print( "x : ", x.grad)
print( "y : ", y.grad)
print( "w1 : ", w1.grad)
print( "w2 : ", w2.grad)
print( "b1 : ", b1.grad)
print( "b2 : ", b2.grad)
f1.backward()

# 直接求導
print( "x : ", x.grad)
print( "y : ", y.grad)
print( "w1 : ", w1.grad)
print( "w2 : ", w2.grad)
print( "b1 : ", b1.grad)
print( "b2 : ", b2.grad)

# 手動求導
w1_grad = -2 * x.t().mm((y-yp).mm(w2.t()).mul(hs).mul(1-hs))
print( "w1 : ", w1_grad)
w2_grad = -2 * hs.t().mm(y-yp)
print( "w2 : ", w2_grad)
b1_grad = -2 * ((y-yp).mm(w2.t()).mul(hs).mul((1-hs)))
print( "b1 : ", b1_grad)
b2_grad = -2 * (y - yp)
print( "b2 : ", b2_grad)
# 兩者一致
```

5 Pytorch 搭建兩層全連接網路

程式碼可以在 GitHub 專案 (kancheng/kan-cs-report-in-2021/CV/pytorch-sigmoid) 找到，詳見 sigmoid-regression.ipynb 檔案，而範例結果可以從相對應的 PDF 檔案 sigmoid-regression.pdf 找到。

```
%matplotlib inline
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt
import torch.nn as nn
import numpy as np
torch.manual_seed(1)    # reproducible
# Data
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())
# 繪圖
plt.scatter(x.numpy(), y.numpy())
# 搭建兩層含有 bias 的全連接網路，隱藏層輸出個數為 20 ，激活函數都用 sigmoid()
class Net(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden)
        self.predict = torch.nn.Linear(n_hidden, n_output) # output layer

    def forward(self, x):
        # x = F.relu(self.hidden(x))      # activation function for hidden layer
        # x = self.predict(x)
        tm = nn.Sigmoid()
        g = tm(self.hidden(x))
        x = self.predict(g)
        return x
net = Net(n_feature=1, n_hidden=20, n_output=1)     # 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

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()
```