

# 計算機視覺作業

干皓丞，2101212850, 信息工程學院

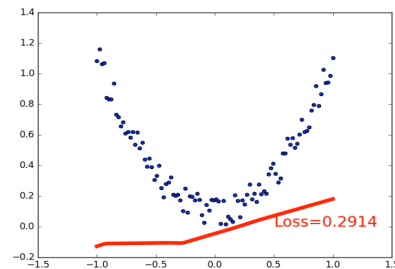
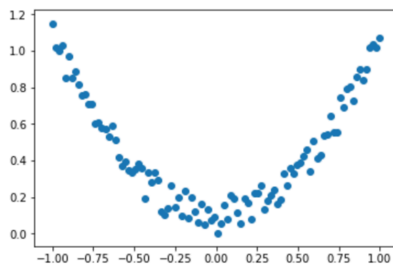
2021 年 10 月 23 日

## 1 題目

Pytorch 搭建兩層全連接網路

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

### PyTorch 搭建两层全连接网络 - 作业



```
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())
```

1. 补全两层全连接代码 W4\_Homework.ipynb
2. 给出变量 W1, b1, W2, b2 导数表达式

$$h = XW_1 + b_1$$

$$h_{\text{sigmoid}} = \text{sigmoid}(h)$$

$$Y_{\text{pred}} = h_{\text{sigmoid}}W_2 + b_2$$

$$f = \|Y - Y_{\text{pred}}\|_F^2$$

思想自由 兼容并包

W5\_Regression.ipynb

< 38 >

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:

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

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

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)。

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

$$(1) \quad \frac{\partial f}{\partial w_1} \quad (2) \quad \frac{\partial f}{\partial b_1} \quad (3) \quad \frac{\partial f}{\partial w_2} \quad (4) \quad \frac{\partial f}{\partial b_2}$$

## 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。

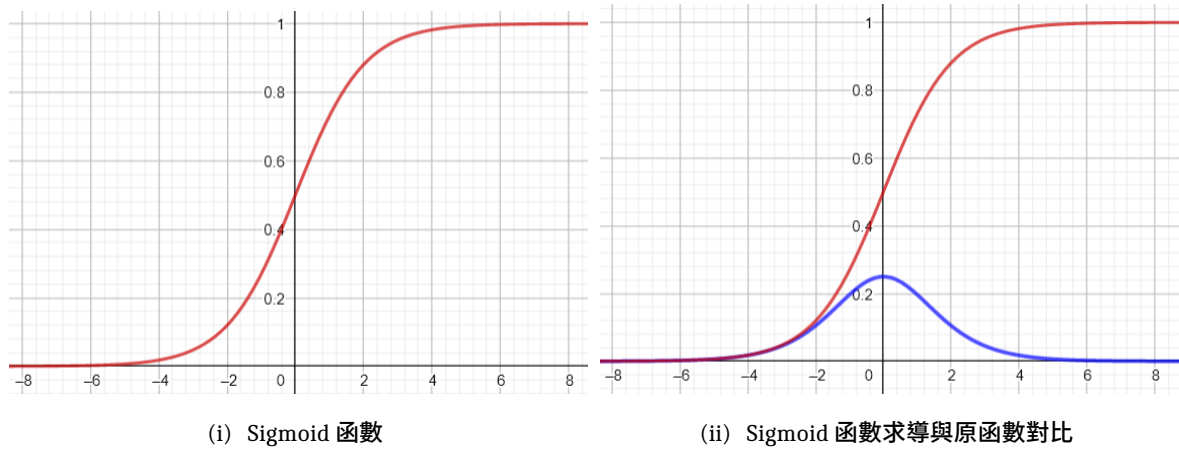


Fig. 2. Sigmoid 函數狀態

### (2) Sigmoid 定義

Sigmoid 定義如數學式 (2.1)：

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

### (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)：

$$S'(x) = S(x)(1 - S(x)) \quad (2.2)$$

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

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

## (4) 程式碼中的數學意義

```

1 import torch
2 import numpy as np
3 torch.manual_seed(0)
4
5 x = torch.randn(100, 1, requires_grad=True)
6 y = torch.randn(100, 1, requires_grad=True)
7 w1 = torch.randn(1, 20, requires_grad=True)
8 w2 = torch.randn(20, 1, requires_grad=True)
9 b1 = torch.randn(100, 20, requires_grad=True)
10 b2 = torch.randn(100, 1, requires_grad=True)
11 print( "x : ", np.shape(x))
12 print( "y : ", np.shape(y))
13 print( "w1 : ", np.shape(w1))
14 print( "w2 : ", np.shape(w2))
15 print( "b1 : ", np.shape(b1))
16 print( "b2 : ", np.shape(b2))

```

## 給定實驗資料

```

In [1]: 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))

x : torch.Size([100, 1])
y : torch.Size([100, 1])
w1 : torch.Size([1, 20])
w2 : torch.Size([20, 1])
b1 : torch.Size([100, 20])
b2 : torch.Size([100, 1])

```

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  $100 \times 1$  大小的矩陣， $w1$  矩陣為  $1 \times 20$  大小的矩陣， $w2$  矩陣為  $20 \times 1$  大小的矩陣， $b1$  矩陣為  $100 \times 20$  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

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

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

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

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

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

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

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

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

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

$$\therefore dh = db_1$$

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

$$\therefore dh = Xdw_1$$

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

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

## 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。

```

1 # 給定實驗資料
2 import torch
3 import numpy as np
4 torch.manual_seed(0)
5
6 x = torch.randn(100, 1, requires_grad=True)
7 y = torch.randn(100, 1, requires_grad=True)
8 w1 = torch.randn(1, 20, requires_grad=True)
9 w2 = torch.randn(20, 1, requires_grad=True)
10 b1 = torch.randn(100, 20, requires_grad=True)
11 b2 = torch.randn(100, 1, requires_grad=True)
12 print("x :", np.shape(x))
13 print("y :", np.shape(y))
14 print("w1 :", np.shape(w1))
15 print("w2 :", np.shape(w2))
16 print("b1 :", np.shape(b1))
17 print("b2 :", np.shape(b2))
18
19 # 數學式
20 import torch.nn as nn
21 tm = nn.Sigmoid()
22 hs = tm(x.mm(w1)+b1)
23 # print(s)
24 yp = (hs).mm(w2)+b2
25 f1 = (y - yp).pow(2).sum()
26 ft = (y - yp).pow(2)
27 print(f1)

```

```
28 print( "x : ", x.grad)
29 print( "y : ", y.grad)
30 print( "w1 : ", w1.grad)
31 print( "w2 : ", w2.grad)
32 print( "b1 : ", b1.grad)
33 print( "b2 : ", b2.grad)
34 f1.backward()
35
36 # 直接求導
37 print( "x : ", x.grad)
38 print( "y : ", y.grad)
39 print( "w1 : ", w1.grad)
40 print( "w2 : ", w2.grad)
41 print( "b1 : ", b1.grad)
42 print( "b2 : ", b2.grad)
43
44 # 手動求導
45 w1_grad = -2 * x.t().mm((y-yp).mm(w2.t()).mul(hs).mul(1-hs))
46 print( "w1 : ", w1_grad)
47 w2_grad = -2 * hs.t().mm(y-yp)
48 print( "w2 : ", w2_grad)
49 b1_grad = -2 * ((y-yp).mm(w2.t()).mul(hs).mul((1-hs)))
50 print( "b1 : ", b1_grad)
51 b2_grad = -2 * (y - yp)
52 print( "b2 : ", b2_grad)
53 # 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。

```

1 %matplotlib inline
2 import torch
3 import torch.nn.functional as F
4 import matplotlib.pyplot as plt
5 import torch.nn as nn
6 import numpy as np
7 torch.manual_seed(1)      # reproducible
8 # Data
9 x = torch.unsqueeze(torch.linspace(-1, 1, 100), dim=1) # x data (tensor)
    , shape=(100, 1)
10 y = x.pow(2) + 0.2*torch.rand(x.size())
11 # 繪圖
12 plt.scatter(x.numpy(), y.numpy())
13 # 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()
14 class Net(torch.nn.Module):
15     def __init__(self, n_feature, n_hidden, n_output):
16         super(Net, self).__init__()
17         self.hidden = torch.nn.Linear(n_feature, n_hidden)
18         self.predict = torch.nn.Linear(n_hidden, n_output) # output layer
19
20     def forward(self, x):
21         # x = F.relu(self.hidden(x))      # activation function for
            hidden layer
22         # x = self.predict(x)
23         tm = nn.Sigmoid()
24         g = tm(self.hidden(x))
25         x = self.predict(g)
26         return x
27 net = Net(n_feature=1, n_hidden=20, n_output=1)      # define the network
28 print(net) # net architecture
29 optimizer = torch.optim.SGD(net.parameters(), lr=0.2)
30 loss_func = torch.nn.MSELoss() # this is for regression mean squared
    loss
31
32
33

```

```
34 plt.ion()    # something about plotting
35
36 for t in range(201):
37     prediction = net(x)    # input x and predict based on x
38     loss = loss_func(prediction, y)    # must be (1. nn output, 2.
        target)
39
40     optimizer.zero_grad()    # clear gradients for next train
41     loss.backward()    # backpropagation, compute gradients
42     optimizer.step()    # apply gradients
43
44     if t % 20 == 0:
45         # plot and show learning process
46         plt.cla()
47         plt.scatter(x.numpy(), y.numpy())
48         plt.plot(x.numpy(), prediction.data.numpy(), 'r-', lw=5)
49         plt.text(0.5, 0, 't = %d, Loss=%.4f' % (t, loss.data.numpy()),
            fontdict={'size': 20, 'color': 'red'})
50         plt.pause(0.1)
51         plt.show()
52
53 plt.ioff()
54 # plt.show()
```