



BS6207 ASSIGNMENT 1

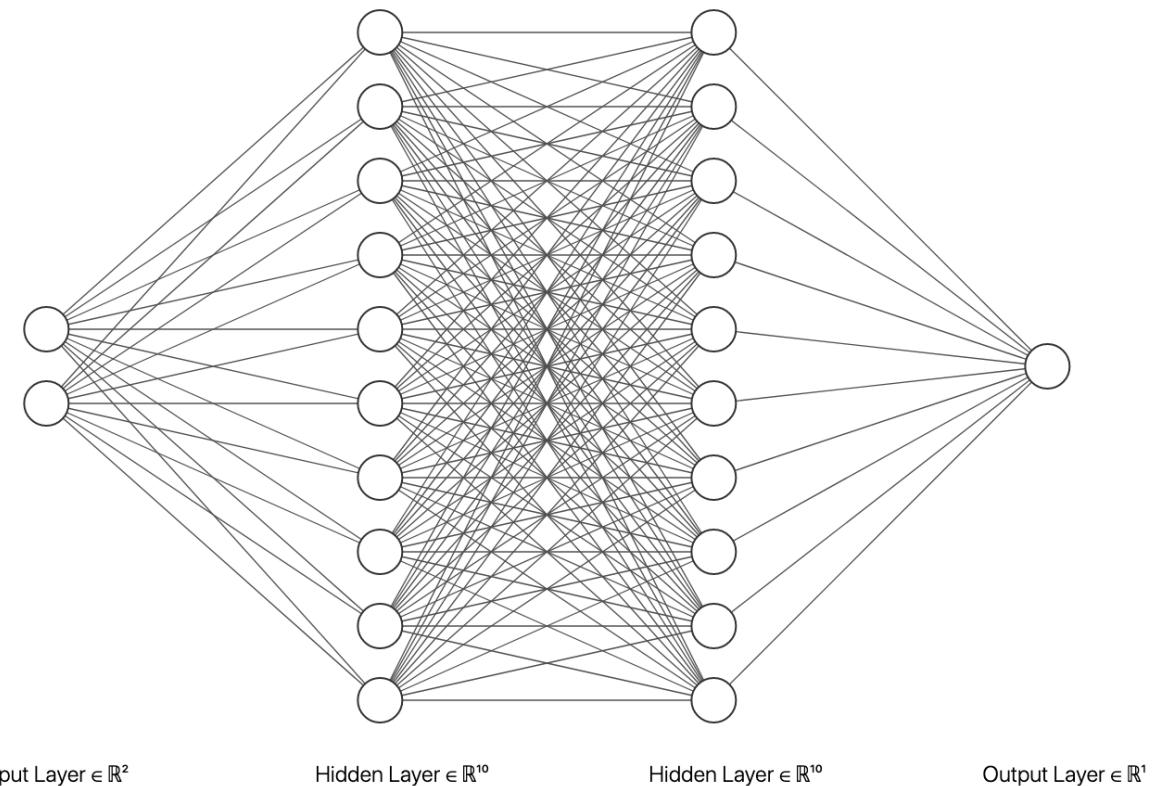
Zhang Xinxin | Li Yonghao | Ni Yuxin
03 Apr 2021

ASSIGNMENT DESCRIPTION

Given a fully connected Neural Network as follows:

1. Input (x_1, x_2): 2 nodes
2. First hidden layer: 10 nodes, with weights (w) and bias (b), sigmoid activation function
3. Second hidden layer: 10 nodes, with weights (w) and bias (b), sigmoid activation function
4. Output (predict): 1 node

Define the label of input as $\frac{x_1^2 + x_2^2}{2}$

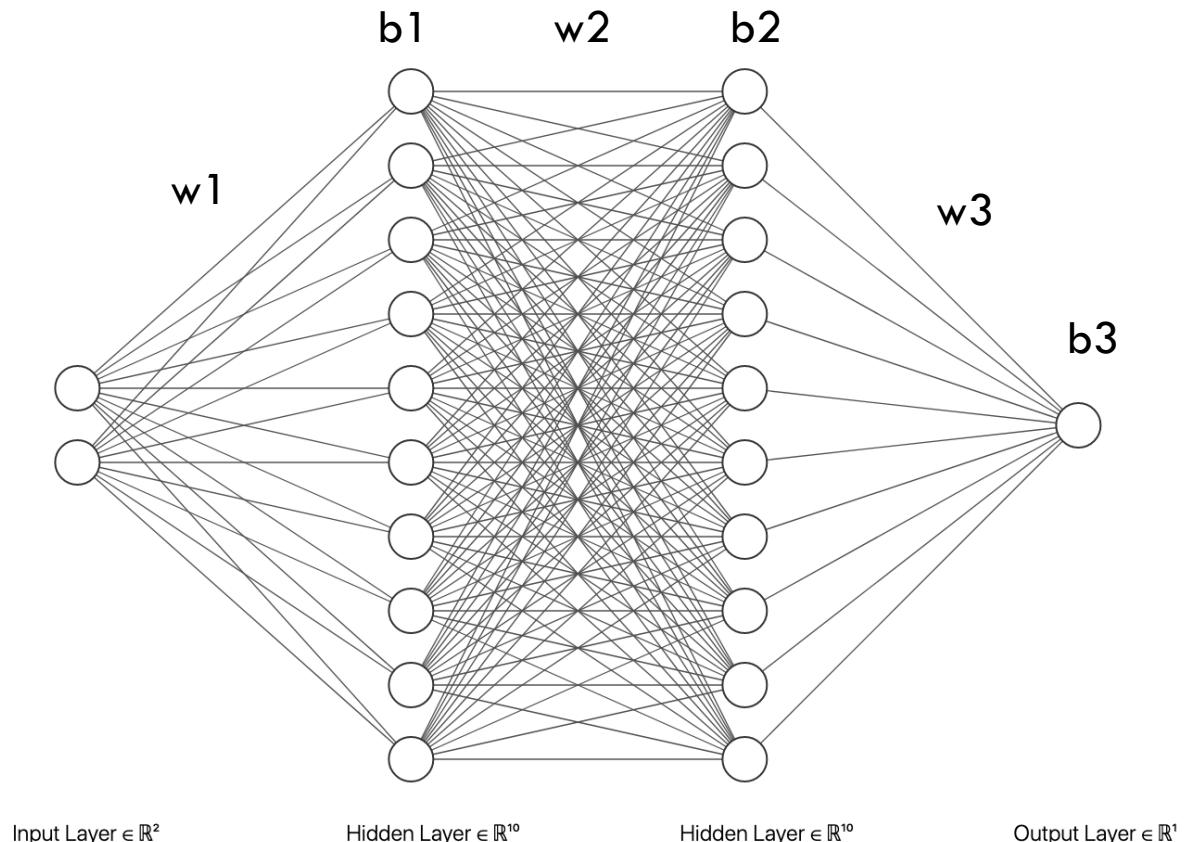


NETWORK PARAMETERS

Use 'w' to denote weights and 'b' to denote biases of each layer

$$y = x w^T + b$$

Layer/ Node	Dimension
Input	1 x 2
w1	10 x 2
b1	1 x 10
w2	10 x 10
b2	1 x 10
w3	1 x 10
b3	1 x 1
Output	1 x 1



PART I – BUILD THE NETWORK BY PYTORCH

1. Use Pytorch to construct the neural network
2. Define input, label and loss function
3. One-time forward propagation
4. One-time backward propagation
5. Get the gradient of loss function against weight/bias by

```
net.layer_name.weight.grad  
net.layer_name.bias.grad
```

```
# 1. implement the neural network in pytorch  
class Net(nn.Module):  
    def __init__(self):  
        super(Net, self).__init__()  
        self.fc1 = nn.Linear(2, 10)  
        self.fc2 = nn.Linear(10, 5)  
        self.fc3 = nn.Linear(5, 1)  
  
    def forward(self, x):  
        x = torch.sigmoid(self.fc1(x))  
        x = torch.sigmoid(self.fc2(x))  
        return self.fc3(x)  
  
net = Net()  
print(net)  
  
Net(  
    (fc1): Linear(in_features=2, out_features=10, bias=True)  
    (fc2): Linear(in_features=10, out_features=5, bias=True)  
    (fc3): Linear(in_features=5, out_features=1, bias=True)  
)  
  
random.seed(127)  
# 2. generate input data x1 x2 from uniform random distribution of [0,1]  
input = torch.rand(1,2, requires_grad = False)  
  
# 3. generate the labels of input data  
y = (input[0][0]**2 + input[0][1]**2)/2  
  
# 4. define loss function  
def my_loss(output, target):  
    loss = (output - target)**2  
    return loss  
  
# 5. One time forward propagation with one data point  
y_pred = net(input)  
  
# 6. compute dL/dw, dL/db using pytorch autograd  
loss = my_loss(y_pred, y)  
loss.backward(retain_graph = True)
```

PART II – BUILD THE NETWORK FROM SCRATCH

1. Obtain the weight and bias from the Pytorch model
2. Define sigmoid and its derivatives
3. One-time forward propagation
4. One-time backward propagation

```
# 7. Implement forward propagation and backward propagation from scratch  
# get the same weight and bias from the model above  
w1 = net.fc1.weight  
w2 = net.fc2.weight  
w3 = net.fc3.weight
```

```
b1 = net.fc1.bias  
b2 = net.fc2.bias  
b3 = net.fc3.bias
```

```
# define the sigmoid activation function from scratch  
def sigmoid(input):  
    return 1/(1 + torch.exp(-input))
```

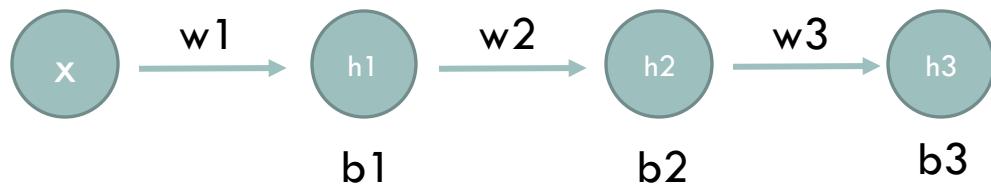
```
# define function for derivative of sigmoid  
def sigmoid_der(input):  
    s = sigmoid(input)  
    return torch.transpose(s*(1-s), 0, 1)
```

```
# forward propagation  
  
# first layer  
z1 = torch.mm(input, torch.transpose(w1, 0, 1)) + b1  
h1 = sigmoid(z1)  
# second layer  
z2 = torch.mm(h1, torch.transpose(w2, 0, 1)) + b2  
h2 = sigmoid(z2)  
# output  
h3 = torch.mm(h2, torch.transpose(w3, 0, 1)) + b3
```

```
# backward propagation  
loss_2 = my_loss(h3, y)  
loss_2 # same as the loss from the model above
```

PART II – BUILD THE NETWORK FROM SCRATCH (CONT'D)

By chain rule, calculate the derivatives of loss function against weights and biases as below:



$$\text{Loss: } L = (h_3 - y)^2$$

$$\text{Output: } h_3 = h_2 w_3 + b_3$$

$$\text{Nodes in hidden layer 2: } h_2 = \sigma(z_2)$$

$$\text{Nodes in hidden layer 1: } h_1 = \sigma(z_1)$$

$$\text{Nodes in hidden layer 2 (before activation): } z_2 = h_1 w_2 + b_2$$

$$\text{Nodes in hidden layer 1 (before activation): } z_1 = x w_1 + b_1$$

$$\frac{\partial L}{\partial w_3} = 2(h_3 - y)h_2$$

$$\frac{\partial L}{\partial w_2} = 2(h_3 - y)w_3 h_1 \sigma'(z_2)$$

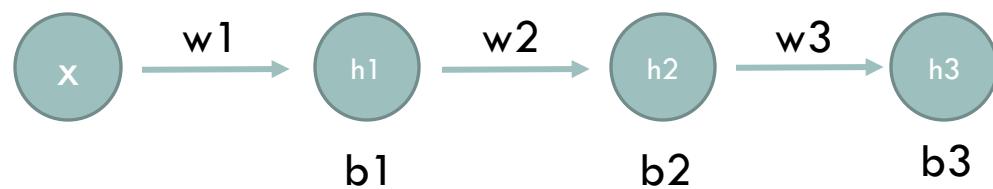
$$\frac{\partial L}{\partial w_1} = 2(h_3 - y)w_3 \sigma'(z_2)w_2 \sigma'(z_1)x$$

$$\frac{\partial L}{\partial b_3} = 2(h_3 - y)$$

$$\frac{\partial L}{\partial b_2} = 2(h_3 - y)w_3 \sigma'(z_2)$$

$$\frac{\partial L}{\partial b_1} = 2(h_3 - y)w_3 \sigma'(z_2)w_2 \sigma'(z_1)$$

PART II – DERIVATION OF LOSS FUNCTION (W)



$$\text{Loss: } L = (h_3 - y)^2$$

$$\text{Output: } h_3 = h_2 w_3 + b_3$$

$$\text{Nodes in hidden layer 2: } h_2 = \sigma(z_2)$$

$$\text{Nodes in hidden layer 1: } h_1 = \sigma(z_1)$$

$$\text{Nodes in hidden layer 2 (before activation): } z_2 = h_1 w_2 + b_2$$

$$\text{Nodes in hidden layer 1 (before activation): } z_1 = x w_1 + b_1$$

Derivation process for dL/dw_1 :

$$\begin{aligned} \frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial w_3} \\ &= 2(h_3 - y)h_2 \end{aligned}$$

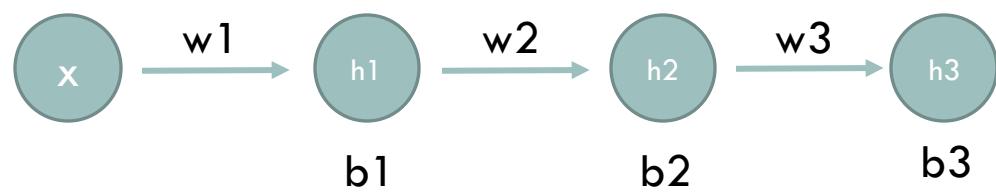
Derivation process for dL/dw_2 :

$$\begin{aligned} \frac{\partial L}{\partial w_2} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial w_2} \\ &= 2(h_3 - y)w_3 \sigma'(z_2)h_1 \\ &= 2(h_3 - y)w_3 \sigma(z_2)(1 - \sigma(z_2))h_1 \end{aligned}$$

Derivation process for dL/dw_1 :

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} \\ &= 2(h_3 - y)w_3 \sigma'(z_2)h_1 w_2 \sigma'(z_1)x \end{aligned}$$

PART II – DERIVATION OF LOSS FUNCTION (B)



$$\text{Loss: } L = (h_3 - y)^2$$

$$\text{Output: } h_3 = h_2 w_3 + b_3$$

$$\text{Nodes in hidden layer 2: } h_2 = \sigma(z_2)$$

$$\text{Nodes in hidden layer 1: } h_1 = \sigma(z_1)$$

$$\text{Nodes in hidden layer 2 (before activation): } z_2 = h_1 w_2 + b_2$$

$$\text{Nodes in hidden layer 1 (before activation): } z_1 = x w_1 + b_1$$

Derivation process for dL/db_3 :

$$\begin{aligned} \frac{\partial L}{\partial b_3} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial b_3} \\ &= 2(h_3 - y) \end{aligned}$$

Derivation process for dL/db_2 :

$$\begin{aligned} \frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial b_2} \\ &= 2(h_3 - y) w_3 \sigma'(z_2) \end{aligned}$$

Derivation process for dL/db_1 :

$$\begin{aligned} \frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial h_1} \frac{\partial h_1}{\partial b_1} \\ &= 2(h_3 - y) w_3 \sigma'(z_2) w_2 \sigma'(z_1) \\ &= 2(h_3 - y) w_3 \sigma(z_2) (1 - \sigma(z_2)) w_2 \sigma(z_1) (1 - \sigma(z_1)) \end{aligned}$$

RESULT – GRADIENT OF LOSS COMPARISON

Gradient of loss function from Pytorch model

```
dLdw3  
[ [-1.1294516, -0.5752896, -1.1747783, -1.096516 , -1.0257739]]  
dLdw2  
[ [-0.0352469, -0.0341221, -0.0466714, -0.0492586, -0.0220244, -0.0230503,  
    -0.0425868, -0.0493314, -0.0361005, -0.0348435],  
  [ 0.0477482,  0.0462245,  0.0632248,  0.0667296,  0.029836 ,  0.0312258,  
   0.0576914,  0.0668282,  0.0489047,  0.0472018],  
  [ 0.0536075,  0.0518967,  0.0709831,  0.074918 ,  0.0334973,  0.0350576,  
   0.0647708,  0.0750287,  0.0549058,  0.052994 ],  
  [ 0.0170999,  0.0165542,  0.0226425,  0.0238976,  0.0106851,  0.0111828,  
   0.0206608,  0.023933 ,  0.0175141,  0.0169042],  
  [ 0.0212497,  0.0205716,  0.0281373,  0.0296971,  0.0132781,  0.0138966,  
   0.0256748,  0.029741 ,  0.0217644,  0.0210065]]  
dLdw1  
[ [ 0.0025872,  0.0045841],  
  [ 0.0014506,  0.0025703],  
  [-0.0044858, -0.0079482],  
  [-0.0021293, -0.0037728],  
  [-0.0004203, -0.0007447],  
  [-0.0041614, -0.0073735],  
  [ 0.0022857,  0.00405 ],  
  [ 0.003307 ,  0.0058596],  
  [ 0.0017463,  0.0030942],  
  [-0.0048133, -0.0085284]]  
dLdb3  
[-1.8975985]  
dLdb2  
[-0.0764258,  0.1035325,  0.116237 ,  0.0370777,  0.0460758]  
dLdb1  
[ 0.0065181,  0.0036547, -0.0113015, -0.0053645, -0.0010589, -0.0104844,  
  0.0057587,  0.0083318,  0.0043996, -0.0121266]
```

Gradient of loss function from self-constructed model

```
dLdw3  
[ [-1.1294516, -0.5752896, -1.1747783, -1.096516 , -1.0257739]]  
dLdw2  
[ [-0.0352469, -0.0341221, -0.0466714, -0.0492585, -0.0220244, -0.0230503,  
    -0.0425868, -0.0493314, -0.0361005, -0.0348435],  
  [ 0.0477482,  0.0462245,  0.0632248,  0.0667296,  0.029836 ,  0.0312258,  
   0.0576914,  0.0668282,  0.0489047,  0.0472018],  
  [ 0.0536075,  0.0518967,  0.0709831,  0.074918 ,  0.0334973,  0.0350576,  
   0.0647708,  0.0750287,  0.0549058,  0.052994 ],  
  [ 0.0170999,  0.0165542,  0.0226425,  0.0238976,  0.0106851,  0.0111828,  
   0.0206608,  0.023933 ,  0.0175141,  0.0169042],  
  [ 0.0212497,  0.0205716,  0.0281373,  0.0296971,  0.0132781,  0.0138966,  
   0.0256748,  0.029741 ,  0.0217644,  0.0210065]]  
dLdw1  
[ [ 0.0025872,  0.0045841],  
  [ 0.0014506,  0.0025703],  
  [-0.0044858, -0.0079482],  
  [-0.0021293, -0.0037728],  
  [-0.0004203, -0.0007447],  
  [-0.0041614, -0.0073735],  
  [ 0.0022857,  0.00405 ],  
  [ 0.003307 ,  0.0058596],  
  [ 0.0017463,  0.0030942],  
  [-0.0048133, -0.0085284]]  
dLdb3  
[-1.8975985]  
dLdb2  
[-0.0764258,  0.1035325,  0.1162371,  0.0370777,  0.0460758]  
dLdb1  
[ 0.0065181,  0.0036547, -0.0113015, -0.0053645, -0.0010589, -0.0104844,  
  0.0057587,  0.0083318,  0.0043996, -0.0121266]
```

SUMMARY

Gradients of loss function against weight and biases
are the same from Pytorch model and the model built
from scratch