NYCU DL

Lab1 - Backpropagation

311551170

林琨堯

1. Introduction

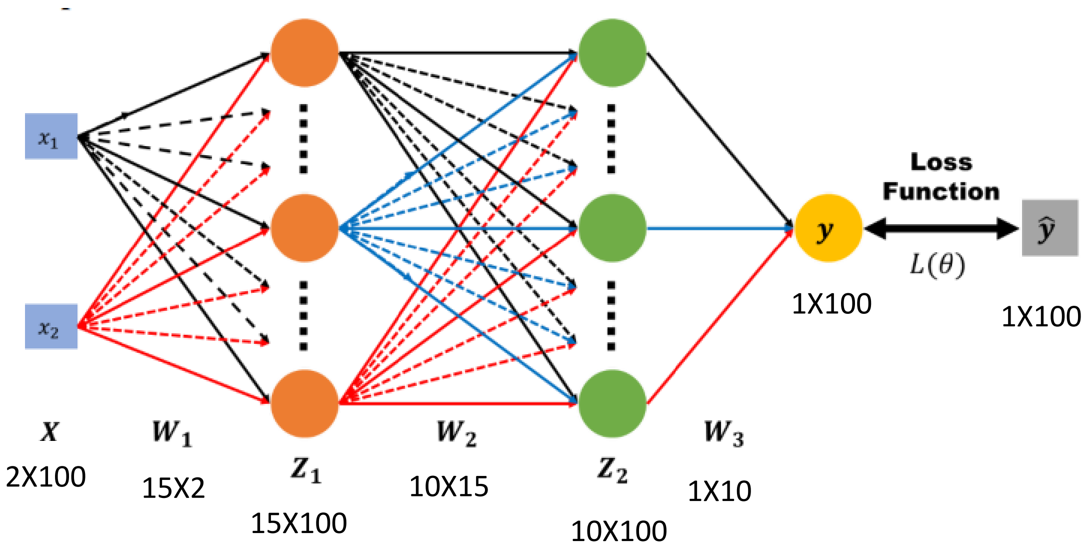
實作一個 neural network，分成兩個部分：forward propagation 和 backpropagation。Forward 把每層的 neurons 乘上 weights 再加 bias，之後再使用 sigmoid 作為 activation function。連續計算幾次後得到預測結果。Backpropagation 則是利用微分計算各個 neuron、weight、bias 對 loss 的影響，以此更新 weights 和 bias。

2. Experiment setups

A. Sigmoid function

```
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

def derivative_sigmoid(x):
    return np.multiply(x, 1.0-x)
```

B. Neural network



我的 network 有 2 層 hidden layer，分別有 15 個和 10 個 neuron。下圖是 neural network class，可傳入 learning rate、epoch、hidden layer size，hidden_size 是一 list，包含所有 hidden layer 的 size。get_loss 計算 MSE loss。Forward_pass 及 backward_pass 分別是 forward propagation 及 backpropagation。Visualize 將資料視覺化。

```
class Net:
    def __init__(self, lr, epoch, hidden_size):

    def get_loss(self, y, y_hat):

    def forward_pass(self, x):

    def backward_pass(self, y, y_hat):

    def visualize(self,x, y, pred_y):
```

__Init__紀錄 weight(w)、bias(b)、線性運算結果(l)、sigmoid 結果(s)，資料結構如下圖:

```
def __init__(self, lr, epoch, hidden_size):
    self.w=[None, np.random.randn(hidden_size[0],2),np.random.randn(hidden_size[1],hidden_siz
    self.b=[None, np.zeros((hidden_size[0],1)),np.zeros((hidden_size[1],1)),np.zeros((1,1))]
    self.l=[None, np.zeros((hidden_size[0],1)),np.zeros((hidden_size[1],1)),np.zeros((1,1))]
    self.s=[None, np.zeros((hidden_size[0],1)),np.zeros((hidden_size[1],1)),np.zeros((1,1))]
```

皆為長度 4 的 list，記錄每一層的結果。為了方便確認，index=0 設定為 None

C. Backpropagation

如下圖，從預測值 y_hat 開始，根據 chain rule 依序計算各個變數的 gradient，然後再更新 weight 和 bias。

$\hat{y}$ 對 loss 的偏微分

$$MSE = \frac{1}{N} \times (y - \hat{y})^2$$

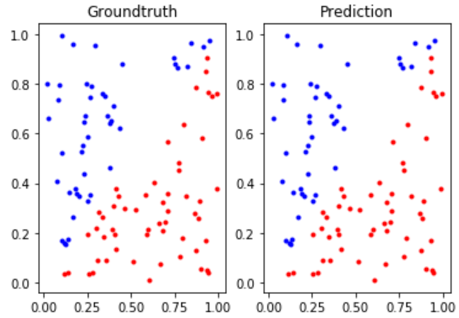
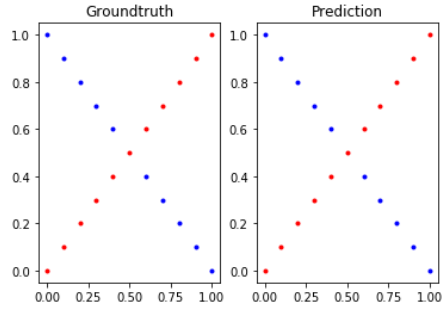
$$\Rightarrow \frac{\partial}{\partial \hat{y}} = \frac{2}{N} \times (y - \hat{y}) \times (-1$$

$$= \frac{2}{N} \times (\hat{y} - y)$$

```python
def backward_pass(self, y, y_hat):
    #backward pass
    g_s3 = (2) * (y_hat - y) / y.shape[1]
    g_l3 = g_s3*derivative_sigmoid(self.s[3])
    g_w3 = g_l3.dot(self.s[2].transpose())
    g_b3 = np.sum(g_l3,axis=1,keepdims=True) * (1/y.shape[0])

    g_s2 = ((self.w[3].transpose()).dot(g_l3))
    g_l2 = g_s2*derivative_sigmoid(self.s[2])
    g_w2 = g_l2.dot(self.s[1].transpose())
    g_b2 = np.sum(g_l2,axis=1,keepdims=True) * (1/y.shape[0])

    g_s1 = ((self.w[2].transpose()).dot(g_l2))
    g_l1 = g_s1*derivative_sigmoid(self.s[1])
    g_w1 = g_l1.dot(self.s[0].transpose())
    g_b1 = np.sum(g_l1,axis=1,keepdims=True) * (1/y.shape[0])

    #update weight
    self.w[1] -= lr*g_w1
    self.w[2] -= lr*g_w2
    self.w[3] -= lr*g_w3
    self.b[1] -= lr*g_b1
    self.b[2] -= lr*g_b2
    self.b[3] -= lr*g_b3
```

3. Results of testing

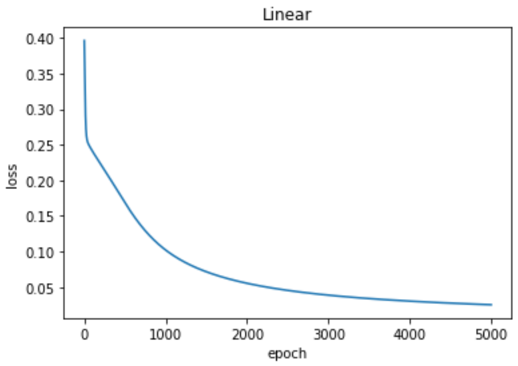
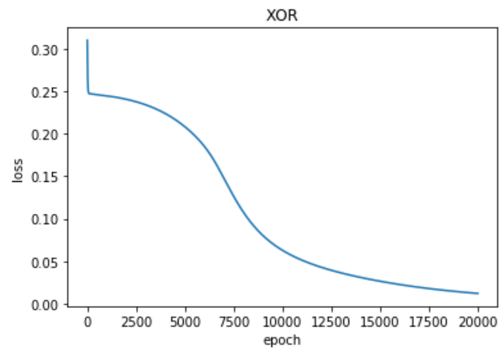
A. Screenshot and comparison figure

linear	XOR
	

B. Show the accuracy of your prediction

<table>
<tr><th>Linear(epoch=5000)</th><th>XOR(epoch=20000)</th></tr>
<tr><td><pre>
epoch 0: loss=[0.39638473]
epoch 500: loss=[0.16835384]
epoch 1000: loss=[0.10196538]
epoch 1500: loss=[0.0714353]
epoch 2000: loss=[0.05523141]
epoch 2500: loss=[0.04526954]
epoch 3000: loss=[0.03857217]
epoch 3500: loss=[0.03377901]
epoch 4000: loss=[0.0301792]
epoch 4500: loss=[0.02736925]
epoch 4999: loss=[0.02511435]
</pre><pre>
[0.00965284, 0.26541905, 0.00947198, 0.99331129, 0.99794716,
 0.76879729, 0.00730543, 0.75594816, 0.00455743, 0.99111841,
 0.00371766, 0.99699118, 0.01645108, 0.00199151, 0.02347688,
 0.04856875, 0.00981155, 0.99748593, 0.09858846, 0.99542424,
 0.10856115, 0.98201922, 0.991485 , 0.00408458, 0.88958407,
 0.00418795, 0.04825392, 0.65427629, 0.6789397 , 0.01278036,
 0.876289 , 0.7932977 , 0.28227698, 0.93930033, 0.98986952,
 0.00183549, 0.00247756, 0.30713533, 0.97833317, 0.24120272,
 0.01165979, 0.01546342, 0.38896107, 0.00358218, 0.03671822,
 0.84182504, 0.97546795, 0.0615675 , 0.99446605, 0.24824142,
 0.9755006 , 0.74371326, 0.11035563, 0.98247935, 0.27389008,
 0.99036906, 0.64253908, 0.00236275, 0.99707722, 0.62074689,
 0.99783367, 0.00179821, 0.92010236, 0.06274777, 0.02933985,
 0.29775376, 0.6279911 , 0.96425181, 0.9886784 , 0.98095015,
 0.10573111, 0.96480692, 0.00411452, 0.00801138, 0.3625924 ,
 0.06066485, 0.07544771, 0.6583304 , 0.00377543, 0.73080271,
 0.04835956, 0.04983208, 0.45288233, 0.0620136 , 0.99820755,
 0.84982007, 0.98996604, 0.90879276, 0.00721775, 0.08403276,
 0.79643956, 0.22548704, 0.99672396, 0.00306367, 0.21136068,
 0.03120272, 0.93335908, 0.12534966, 0.94871401, 0.02158489]
</pre></td><td><pre>
epoch 0: loss=[0.30985773]
epoch 2000: loss=[0.24049675]
epoch 4000: loss=[0.22367769]
epoch 6000: loss=[0.18505993]
epoch 8000: loss=[0.10760084]
epoch 10000: loss=[0.06284737]
epoch 12000: loss=[0.04283786]
epoch 14000: loss=[0.0309986]
epoch 16000: loss=[0.02265669]
epoch 18000: loss=[0.01655852]
epoch 19999: loss=[0.01221297]
</pre><pre>
[0.01559064]
[0.99922558]
[0.02000857]
[0.99895485]
[0.03458137]
[0.9980541]
[0.06788238]
[0.99195767]
[0.10660276]
[0.8194594]
[0.11885697]
[0.10335934]
[0.84384749]
[0.07826415]
[0.97335384]
[0.05581602]
[0.98010456]
[0.03941056]
[0.98085065]
[0.02833696]
[0.97999176]
</pre></td></tr>
</table>

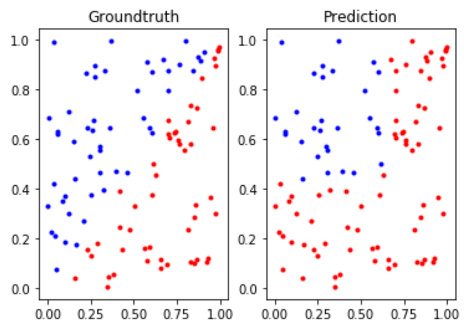
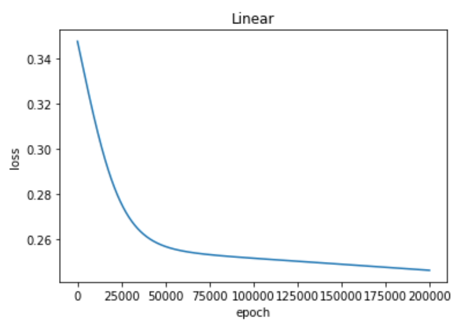
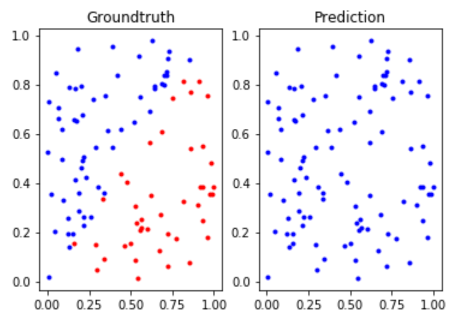
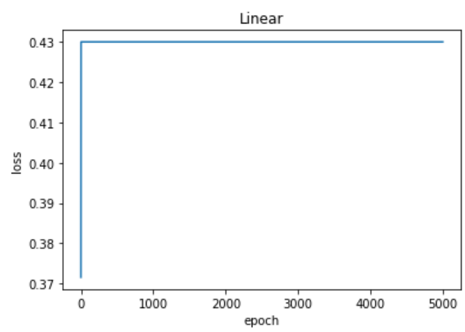
C. Learning curve (loss, epoch curve)

Linear(lr=0.06)	XOR(lr=0.08)
	

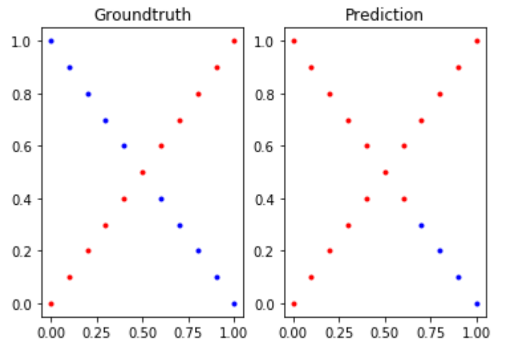
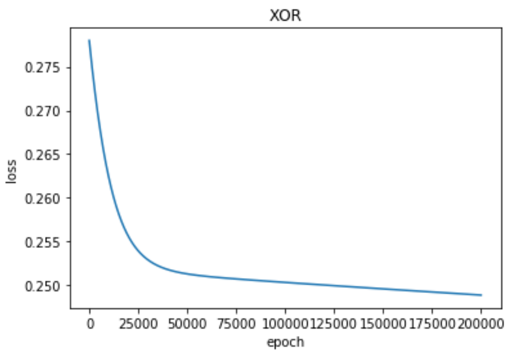
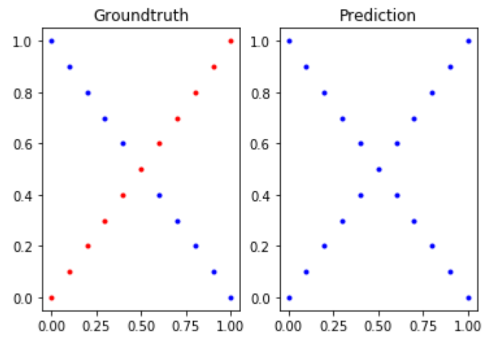
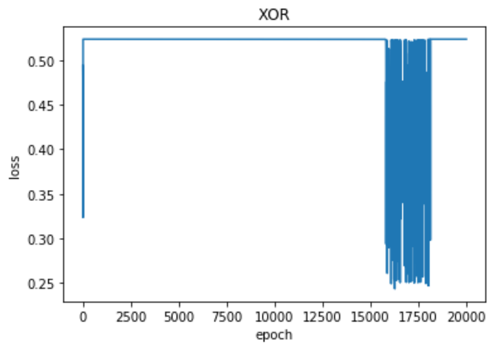
4. Discussion

A. Try different learning rates

Linear:

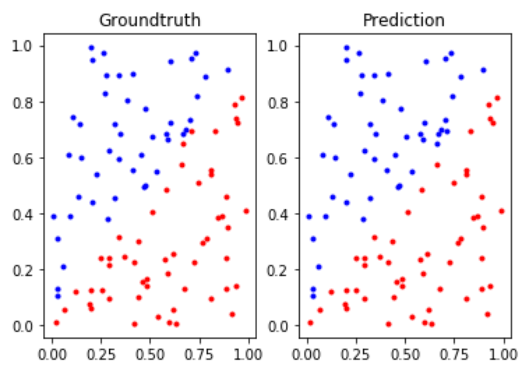
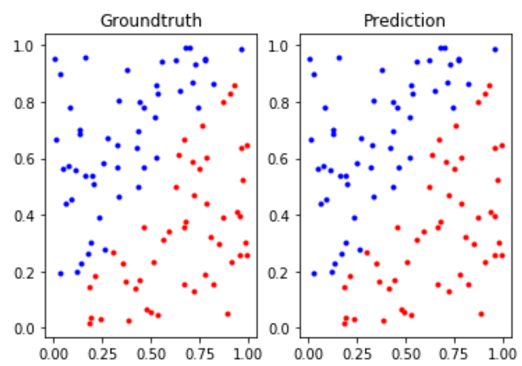
lr=0.0001	lr=100
	
Learning rate 太小，loss 很難達到 minimum，影響預測結果	Learning rate 太大，loss 跳過最小值

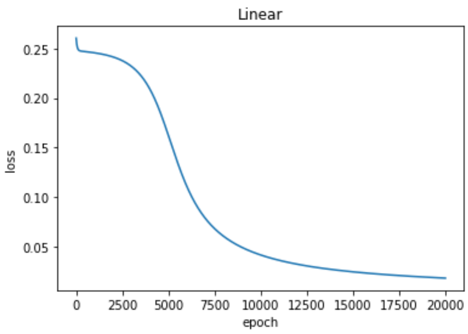
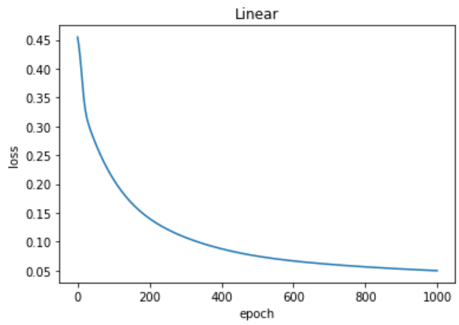
XOR

lr=0.0001	lr=100
	
Learning rate 太小，loss 很難達到 minimum，影響預測結果	Learning rate 太大，loss 來回震盪

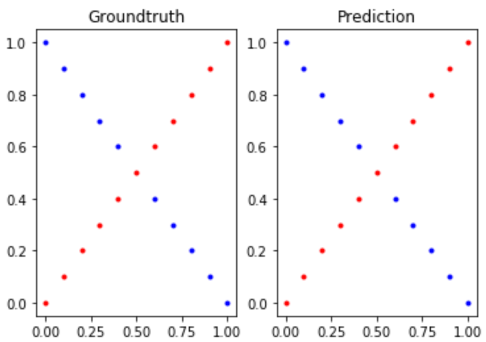
B. Try different numbers of hidden units

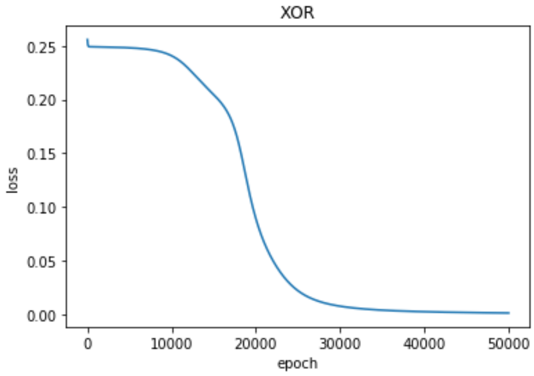
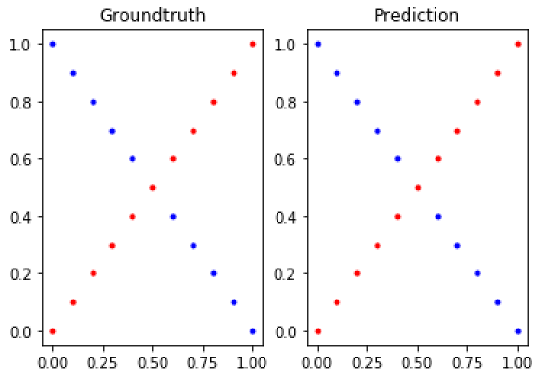
Linear

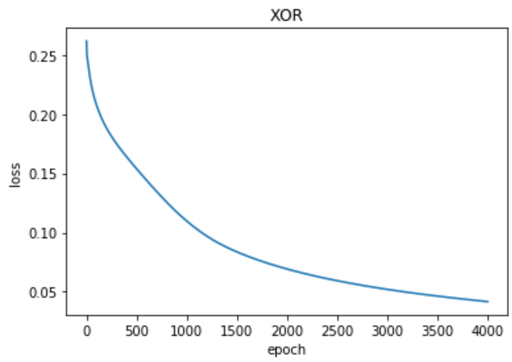
Hidden_size=[3,2]	Hidden_size=[75,50]
	

<table>
<tr>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>需要訓練更多次才能獲得足夠低的 loss</td>
<td>不用訓練很多次 loss 就會開始收斂</td>
</tr>
</table>

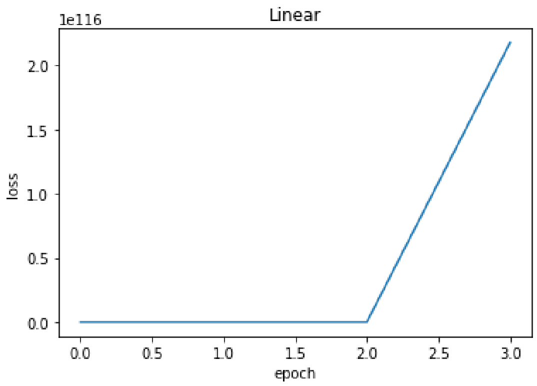
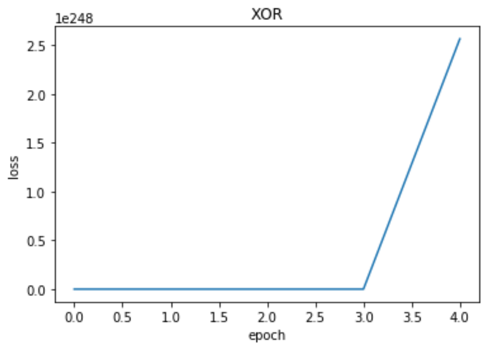
XOR

<table>
<tr>
<th>Hidden_size=[3,2]</th>
<th>Hidden_size=[75,50]</th>
</tr>
<tr>
<td>

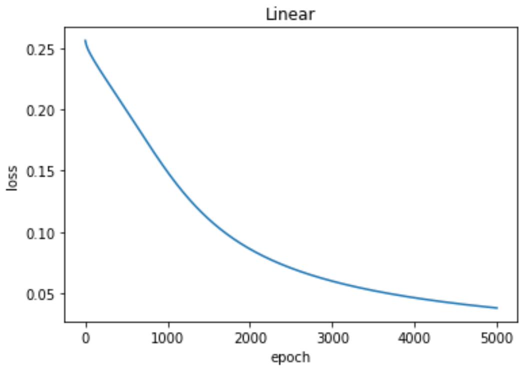
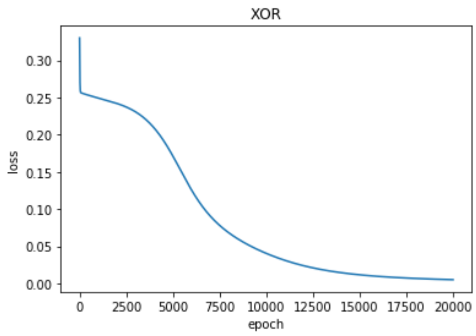

</td>
<td>


</td>
</tr>
<tr>
<td>需要訓練更多次才能獲得足夠低的 loss</td>
<td>不用訓練很多次 loss 就會開始收斂</td>
</tr>
</table>

C. Try without activation functions

<table>
<tr><th>Linear</th><th>XOR</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Loss 一直上升到 NAN</td><td>Loss 一直上升到 NAN</td></tr>
</table>

D. Try without bias

<table>
<tr><th>Linear</th><th>XOR</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>似乎沒有影響</td><td>似乎沒有影響</td></tr>
</table>