DLP HW1

309553008

1. Introduction

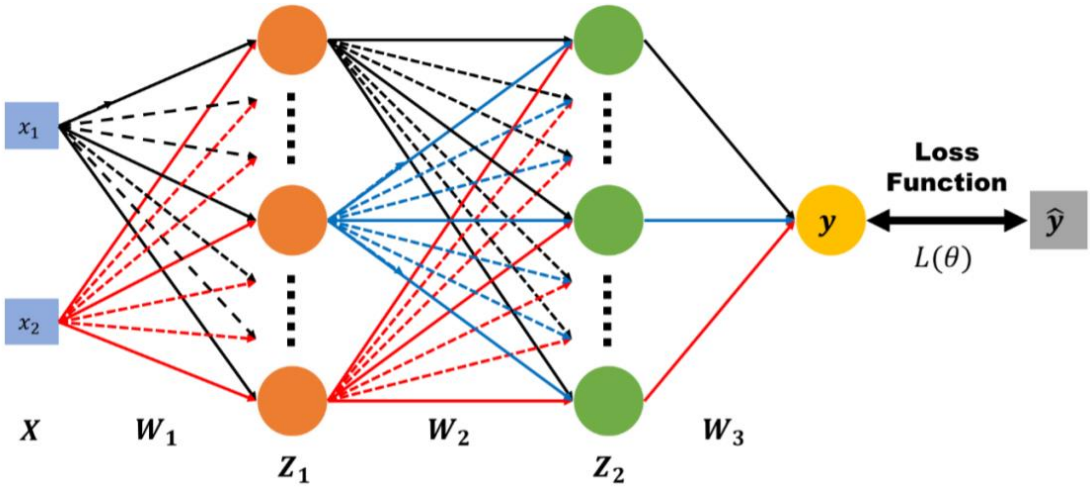
   這次作業是要實作一個有兩層 hidden layer、一層 input layer、一層 output layer 的 neural network，首先 input 經過 forward propagation 得到一個預測值 y，將此預測值與 ground truth 比對得到 Loss，再將此 Loss 分別對不同的 weight 做偏微分得到該 weight 的斜率，然後根據斜率更新 weight (backward propagation)，最後依據更新後的 weight 做下一輪迭代直到 fit。

2. Experiment setups

   A. Sigmoid functions :

      Sigmoid functions 是一種值域在[0, 1]的一個函數。他可以把值壓在[0, 1]之間，算是一種 normalize，經過 Sigmoid functions 後，就不會因為原本的值差異太大而影響 network，而且因為這是一個非線性函數，這使得 network 可以 fit 非線性的資料。

   B. Neural network :



      這是一個有兩層 hidden layer 的 fully connected network，input 為一個點的座標(x, y)，經過 $W_1 = [w_1, w_2, ..., w_{hidden_unit_num}]$後得到 Z_1，Z_1 經過 W_2 得到 Z_2，Z_2 經過 W_3 最後的到預測的 y，藉由預測的 y 與 ground truth y 計算 loss (我是採用 MSE)，其中 W_1、W_2、W_3 分別是[2, hidden_unit_num], [hidden_unit_num, hidden_unit_num], [hidden_unit_num, 1]的矩陣

   C. Backpropagation :

      藉由 Loss (L(**θ**))對不同的 W 做偏微分，我們可以得到 Loss 對該 W 的 Gradient。藉由這些 Gradient，我們可以去更新 W 使 Loss 達到最小。

3. Results of your testing

A. Screenshot and comparison figure

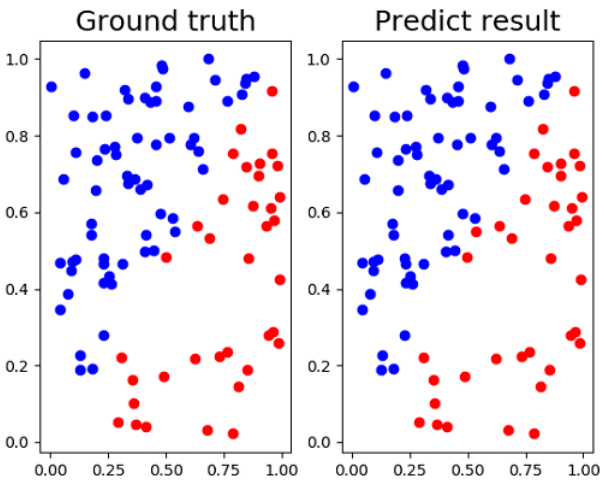
Training(linear):

```
start training
Epoch: 0, loss:  0.311527363
Epoch: 1000, loss:  0.000011483
Epoch: 2000, loss:  0.000003678
Epoch: 3000, loss:  0.000002020
Epoch: 4000, loss:  0.000001344
Epoch: 5000, loss:  0.000000986
Epoch: 6000, loss:  0.000000769
Epoch: 7000, loss:  0.000000625
Epoch: 8000, loss:  0.000000523
Epoch: 9000, loss:  0.000000447
```

Testing(linear):

(因為有 100 個點，這邊貼其中幾個示意)

```
start testing
[[9.97662779e-01]
 [3.67789997e-05]
 [9.99999953e-01]
 [4.83536043e-01]
 [2.83710352e-08]
 [9.99999907e-01]
 [9.99994257e-01]
 [9.99999912e-01]
 [2.92814521e-07]
 [9.98182452e-01]
 [2.02236475e-08]
 [9.99999891e-01]
 [9.99999322e-01]
 [9.99999760e-01]
 [4.41782323e-06]
 [1.04174140e-07]
 [9.99997475e-01]
 [9.99999938e-01]
 [3.95923986e-08]
 [9.99999836e-01]
 [1.52632856e-02]
 [9.99670851e-01]
 [9.99999945e-01]
 [9.99997329e-01]
 [7.83289021e-07]
 [9.99999915e-01]
 [9.99999933e-01]
```

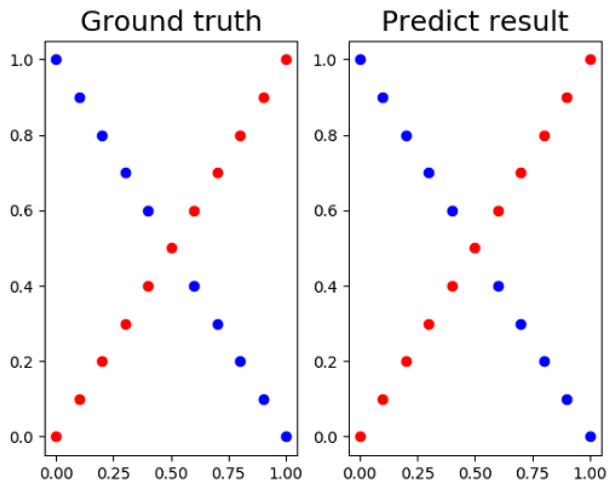


Training(XOR):

```
start training
Epoch: 0, loss:  0.381412890
Epoch: 1000, loss:  0.000066863
Epoch: 2000, loss:  0.000022755
Epoch: 3000, loss:  0.000013001
Epoch: 4000, loss:  0.000008892
Epoch: 5000, loss:  0.000006667
Epoch: 6000, loss:  0.000005287
Epoch: 7000, loss:  0.000004353
Epoch: 8000, loss:  0.000003683
Epoch: 9000, loss:  0.000003180
```

Testing(XOR):

```
start testing
[[0.00124372]
 [0.99895986]
 [0.00132595]
 [0.99896286]
 [0.00208569]
 [0.99876409]
 [0.00421017]
 [0.99803594]
 [0.00680985]
 [0.9903045 ]
 [0.00733196]
 [0.00607104]
 [0.99170087]
 [0.00450951]
 [0.9972249 ]
 [0.0032667 ]
 [0.99753621]
 [0.0023958 ]
 [0.99764474]
 [0.00180525]
 [0.99763785]]
```

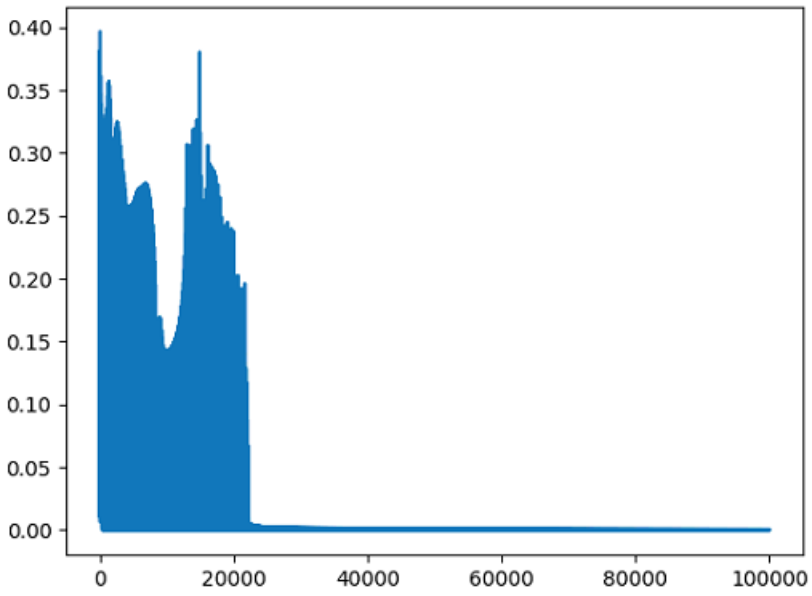


B. Show the accuracy of your prediction

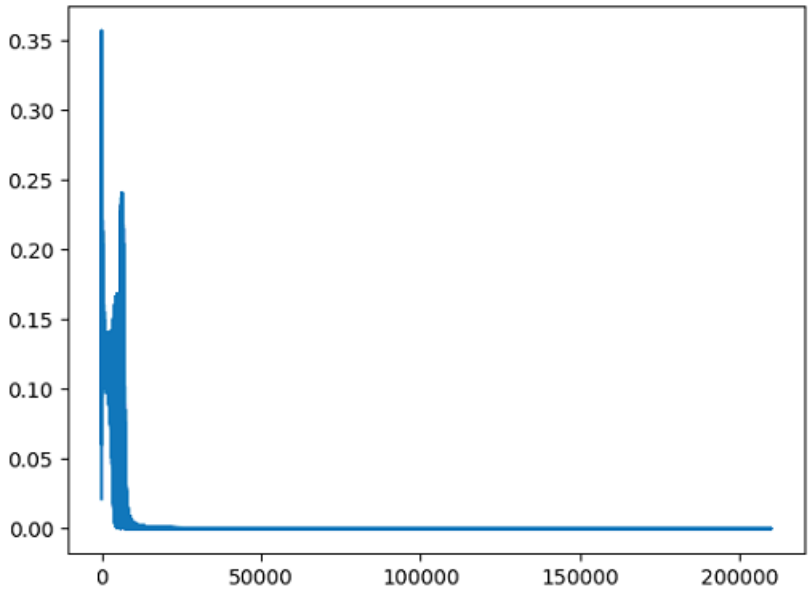
不管是 linear 還是 XOR 都是 100%

C. Learning curve (loss, epoch curve)

Linear:



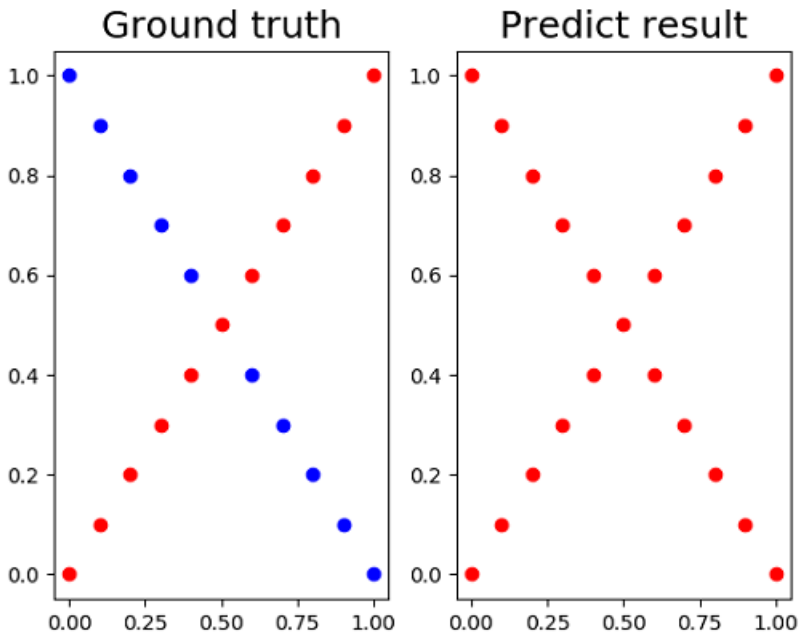
XOR:

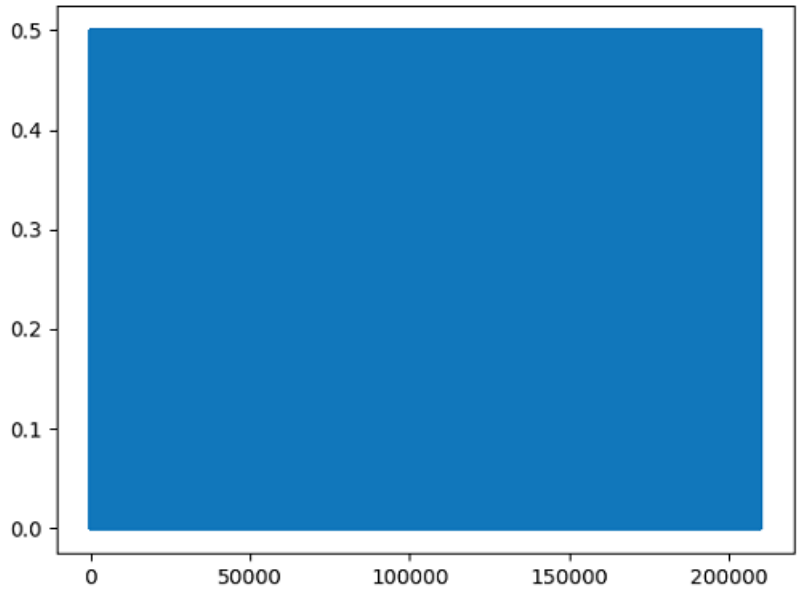


4. Discussion

A. Try different learning rates

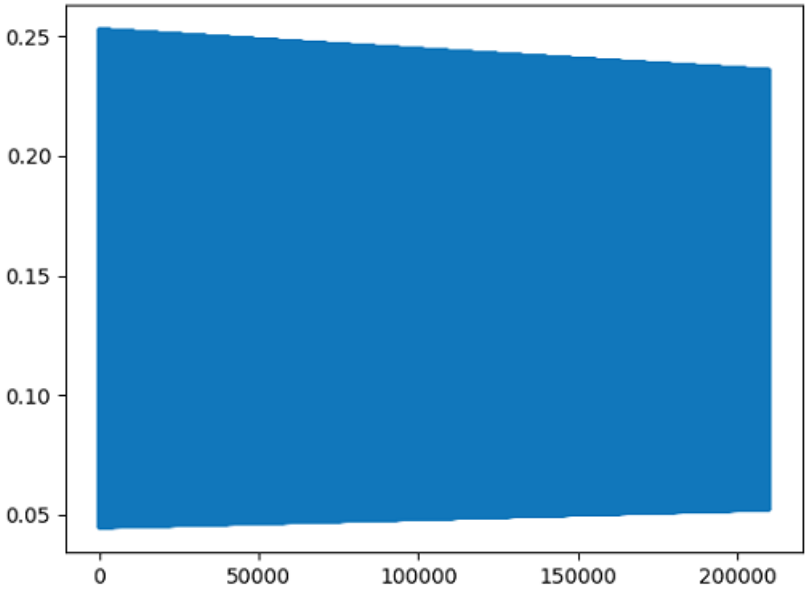
如同助教課所說的一樣，如果 learning rate 太大，會讓點一直震盪，無法收斂到谷底(Loss 最小的點)以下面的圖來說，我把 learning rate 開超大，Loss 只能在 0.5 就下不去了






```
Accuracy: 52.38%
```

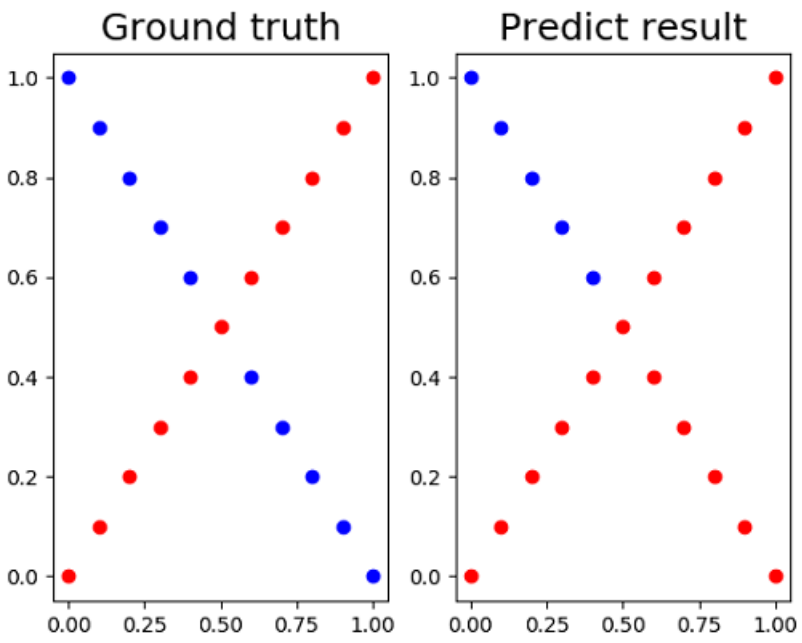
而 learning rate 太小，則收斂速度太慢




```
Accuracy: 47.62%
```

B. Try different numbers of hidden units

以 XOR 來說，如果 hidden unit 太少會造成結果不好，因為這樣簡單的架構無法 fit 這個問題(下圖為 hidden units = 3)

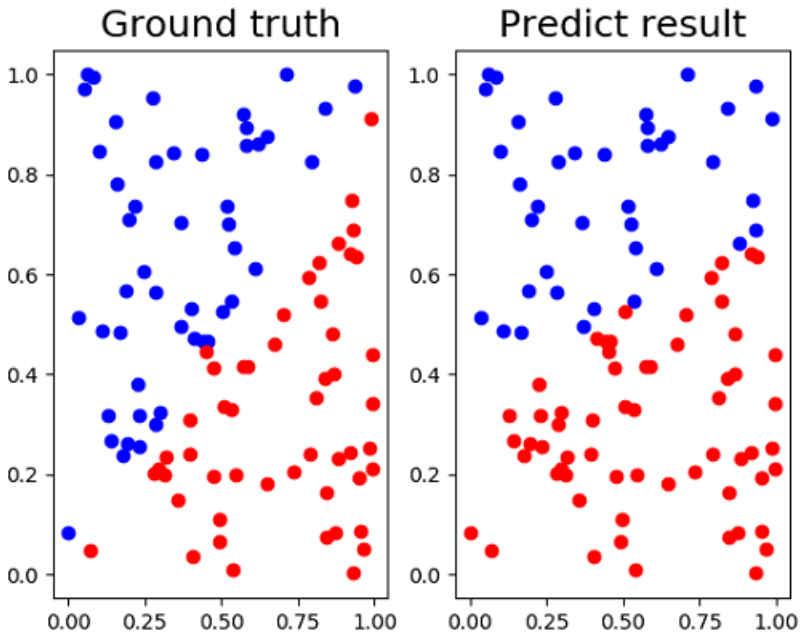



```
Accuracy: 76.19%
```

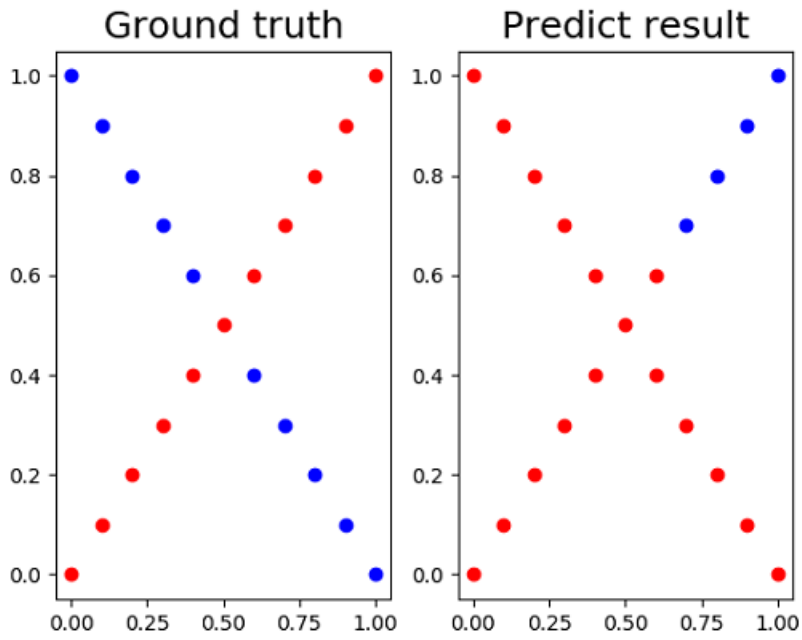
C. Try without activation functions

剛開始的時候一直爆開，後來想一想發現是因為沒有用 activation function，所以值會變得比有用大很多，如果這個情況下用原本的 learning rate 的話就會爆開，後來把 learning rate 縮小後就可以跑了。

這是 linear 的結果，雖然沒有很好，但是正確率還是有 82%，應該是因為這 data 本來就是線性的，所以效果還可以



至於 XOR 的話就因為 data 不是線性，所以這個 network 無法 fit，正確率就下去了(33.33%)


Ground truth
Predict result
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00