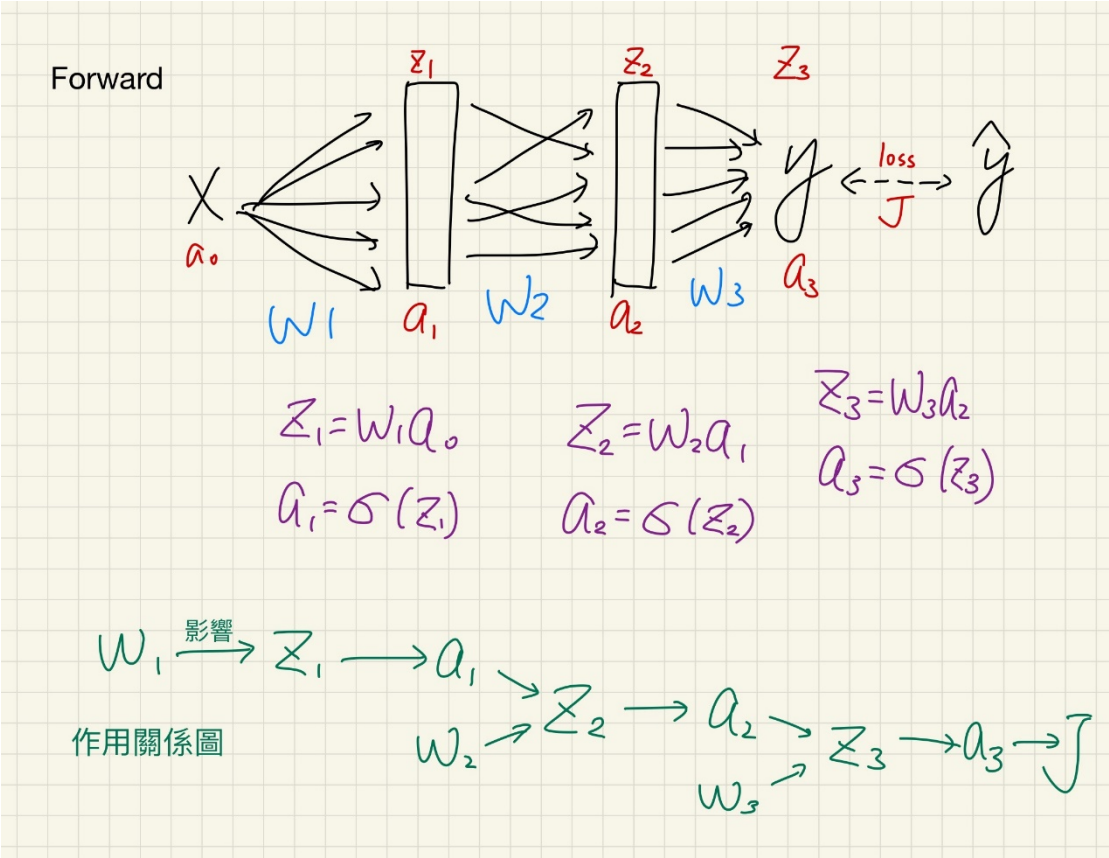
DLP Lab2 Report

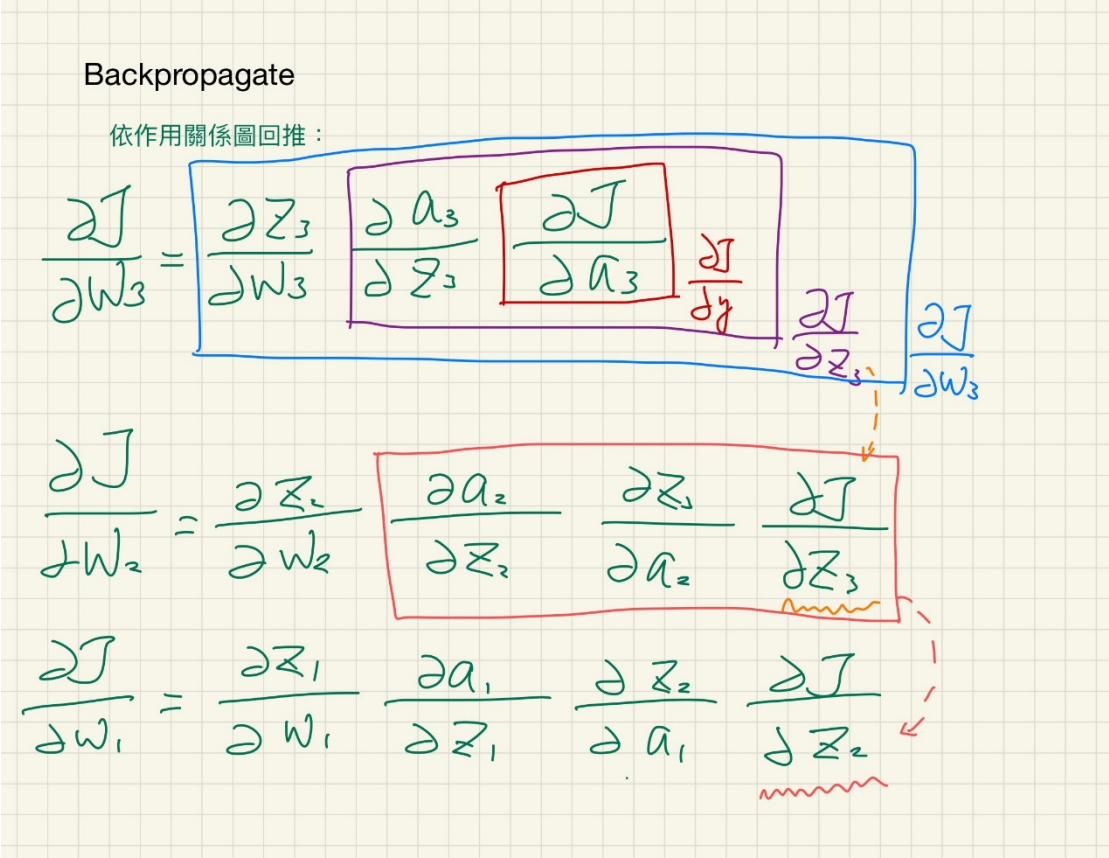
Introduction

在建立一個 fully connected NN 之前，要先進行一些必要的計算。

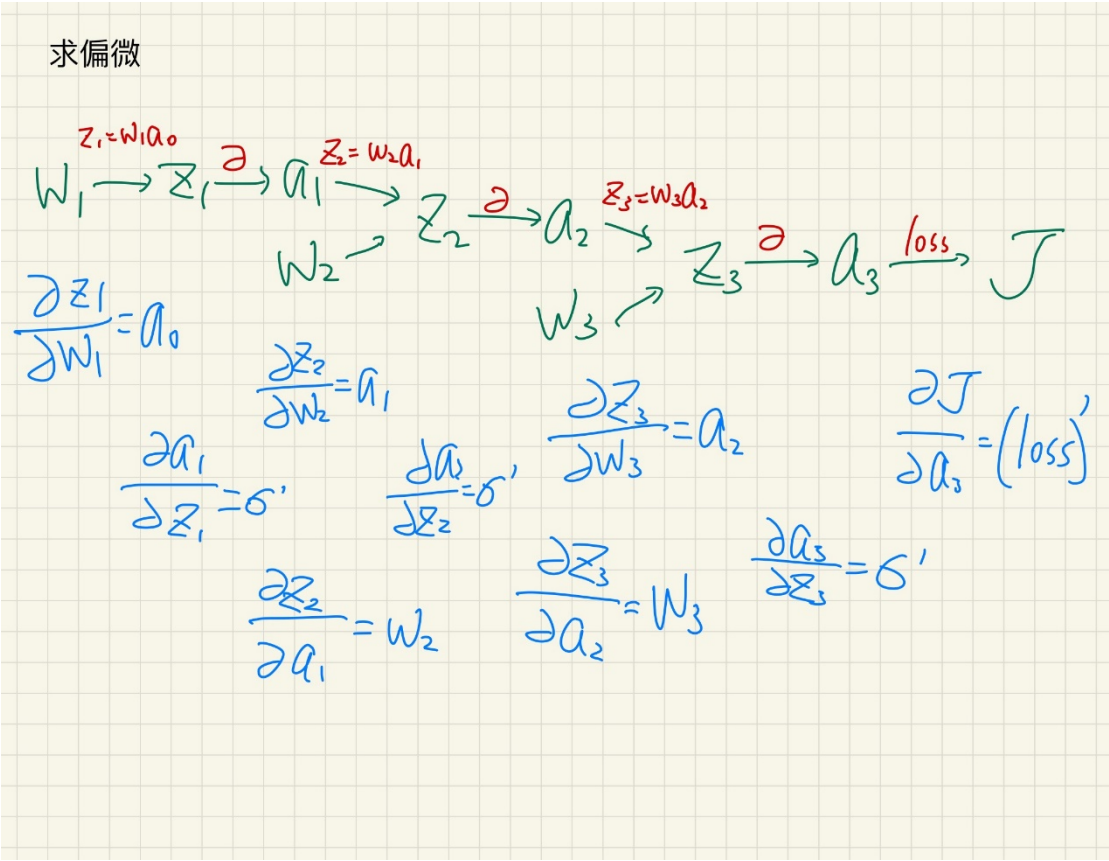
1. 寫出 forward 計算式及畫出變數之間的作用關係圖:



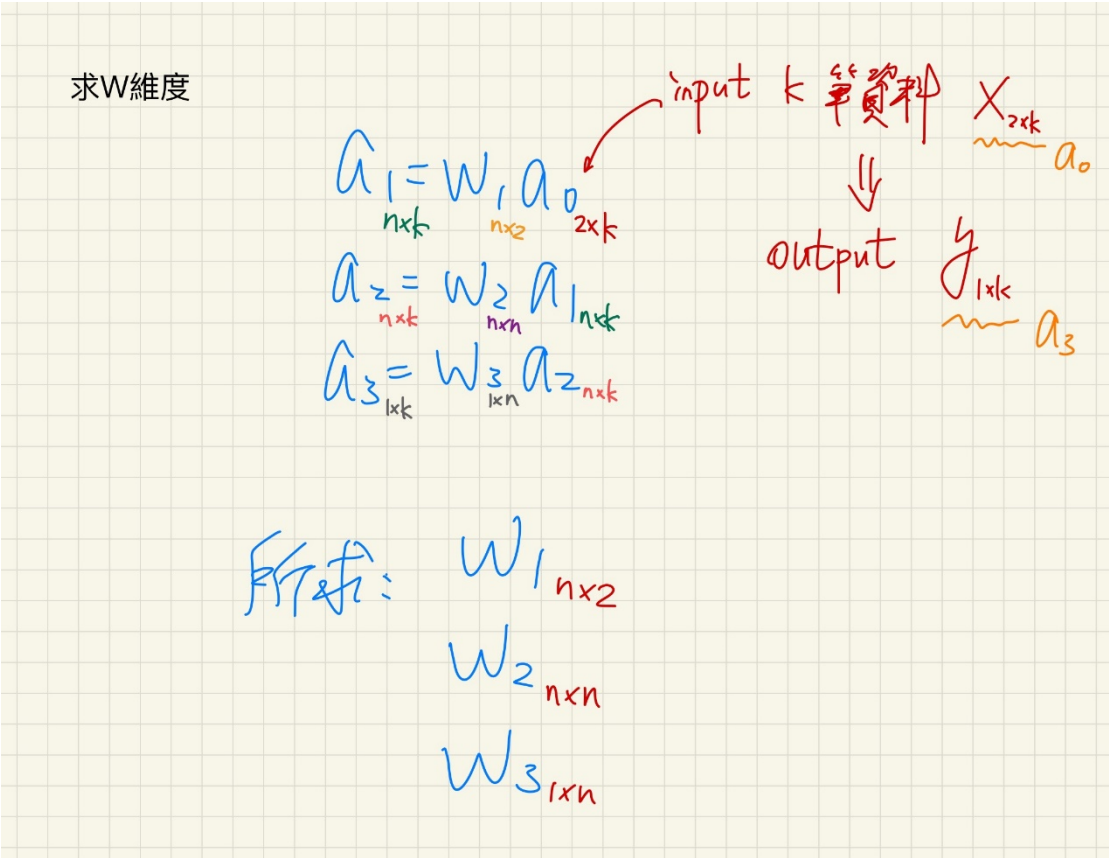
2. 依上述作用關係圖回推出 backpropagate 計算式:



3. 求出上面各偏微分的值:



4. 計算各變數維度 (主要是 W):



Experiment setups

A. Sigmoid functions

```
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

下圖中的 w1, w2, w3 即為 NN。forwarding 則依照 Introduction 第一張圖片的公式計算。

```python
def forward(a0, w1, w2, w3):
    z1 = np.dot(w1, a0)
    a1 = sigmoid(z1)
    z2 = np.dot(w2, a1)
    a2 = sigmoid(z2)
    z3 = np.dot(w3, a2)
    a3 = sigmoid(z3)

    return a3, a0, a1, a2, a3

def cost(pred_y, y):
    # cross entropy
    return float(-(1/y.shape[1])*(y @ np.log(pred_y + 0.0001).T+(1-y) @ np.log(1-p

def backpropagation(pred_y, y, learning_rate, n, a0, w1, a1, w2, a2, w3, a3):
    dJ_da3 = -(y / (pred_y + 0.0001) - (1 - y) / (1 - pred_y + 0.0001))
    dJ_dz3 = derivative_sigmoid(a3) * dJ_da3
    dJ_dw3 = np.dot(dJ_dz3, a2.T)

    dJ_da2 = np.dot(w3.T, dJ_dz3)
    dJ_dz2 = derivative_sigmoid(a2) * dJ_da2
    dJ_dw2 = np.dot(dJ_dz2, a1.T)

    dJ_da1 = np.dot(w2.T, dJ_dz2)
    dJ_dz1 = derivative_sigmoid(a1) * dJ_da1
    dJ_dw1 = np.dot(dJ_dz1, a0.T)

    w1 = w1 - learning_rate * (dJ_dw1 / n)
    w2 = w2 - learning_rate * (dJ_dw2 / n)
    w3 = w3 - learning_rate * (dJ_dw3 / n)

    return w1, w2, w3
```

```python
n = 10 # weight size
w1 = np.random.randn(n, 2)
w2 = np.random.randn(n, n)
w3 = np.random.randn(1, n)
```

初始化就用隨機值，維度在 Introduction 圖四說明。

C. backpropagation

上圖中的 backpropagation 函式。依照 Introduction 第 2、3 張圖片的式子計算。有些地方需要 transpose 來符合各參數維度。pred_y 分母部分+0.0001 避免出現 0。最後，更新 weight 的部分除以 n 是希望 weight 維度越高，每個值更新的量越少，避免矩陣乘法運算時算出來的值太大。亦可將 n 併入 learning rate 看待。

Results of your testing

A. Screenshot and comparison figure

linear training:

```
yucheng@ubuntu:~/Documents/DLP_lab$ python3 lab2.py
training...
epoch 0 loss: 0.7889839340623331
epoch 10 loss: 0.6456236348004784
epoch 20 loss: 0.6058923568308281
epoch 30 loss: 0.5549787648741095
epoch 40 loss: 0.4881121604683834
epoch 50 loss: 0.410131878899793
epoch 60 loss: 0.33422470690527534
epoch 70 loss: 0.27082956197768937
epoch 80 loss: 0.22210737043766138
epoch 90 loss: 0.18565086723949087
epoch 100 loss: 0.15830038965841894
epoch 110 loss: 0.13747599226013882
epoch 120 loss: 0.12131864559196792
epoch 130 loss: 0.10853578280209288
epoch 140 loss: 0.09823242413434574
epoch 150 loss: 0.08978359128302874
epoch 160 loss: 0.0827467391654886
epoch 170 loss: 0.07680341571764392
epoch 180 loss: 0.0717206089157903
epoch 190 loss: 0.06732500148694917
epoch 200 loss: 0.06348563889736991
epoch 210 loss: 0.06010210419811306
epoch 220 loss: 0.05709633163687766
epoch 230 loss: 0.054406854325849734
epoch 240 loss: 0.05198470245248112
epoch 250 loss: 0.04979043666006803
epoch 260 loss: 0.04779197323815874
epoch 270 loss: 0.04596296927402918
epoch 280 loss: 0.04428160908091387
epoch 290 loss: 0.04272968183936545
epoch 300 loss: 0.04129187312011358
epoch 310 loss: 0.039955215277354975
epoch 320 loss: 0.03870865711426146
epoch 330 loss: 0.0375427239950944
epoch 340 loss: 0.03644924719566135
epoch 350 loss: 0.03542114673021935
epoch 360 loss: 0.03445225582844767
epoch 370 loss: 0.03353717810863465
epoch 380 loss: 0.03267117060999178
epoch 390 loss: 0.03185004742107244
epoch 400 loss: 0.031070099821899087
```

```
epoch 4580 loss: 0.0014327525296149635
epoch 4590 loss: 0.0014279873258807264
epoch 4600 loss: 0.0014232474873740193
epoch 4610 loss: 0.0014185328258986701
epoch 4620 loss: 0.0014138431550347263
epoch 4630 loss: 0.001409178290118418
epoch 4640 loss: 0.001404538048221991
epoch 4650 loss: 0.0013999222481342177
epoch 4660 loss: 0.0013953307103407393
epoch 4670 loss: 0.0013907632570050272
epoch 4680 loss: 0.0013862197119493742
epoch 4690 loss: 0.0013816999906362662
epoch 4700 loss: 0.0013772036501498131
epoch 4710 loss: 0.0013727307891777314
epoch 4720 loss: 0.001368281147993264
epoch 4730 loss: 0.0013638545584374446
epoch 4740 loss: 0.0013594508539017114
epoch 4750 loss: 0.0013550698693105257
epoch 4760 loss: 0.0013507114411043684
epoch 4770 loss: 0.0013463754072229723
epoch 4780 loss: 0.0013420616070886394
epoch 4790 loss: 0.0013377698815899076
epoch 4800 loss: 0.0013335000730653399
epoch 4810 loss: 0.0013292520252875847
epoch 4820 loss: 0.0013250255834475297
epoch 4830 loss: 0.0013208205941389994
epoch 4840 loss: 0.0013166369053429566
epoch 4850 loss: 0.0013124743664128136
epoch 4860 loss: 0.0013083328280591017
epoch 4870 loss: 0.0013042121423348446
epoch 4880 loss: 0.001300112162620965
epoch 4890 loss: 0.0012960327436118886
epoch 4900 loss: 0.0012919737413012672
epoch 4910 loss: 0.001287935012967923
epoch 4920 loss: 0.0012839164171620644
epoch 4930 loss: 0.0012799178136915163
epoch 4940 loss: 0.0012759390636082823
epoch 4950 loss: 0.0012719800291950787
epoch 4960 loss: 0.0012680405739522367
epoch 4970 loss: 0.0012641205625846977
epoch 4980 loss: 0.0012602198609890955
epoch 4990 loss: 0.0012563383362410952
```

linear testing:

```
testing...
[[9.99999883e-01 9.99596115e-01 9.99999916e-01 4.62178005e-09
  9.99714437e-01 9.99999893e-01 2.15566177e-07 2.60759049e-06
  8.64250621e-06 8.91112728e-07 3.80527523e-09 9.99984815e-01
  9.94590090e-01 9.99987893e-01 5.46860280e-07 9.99945759e-01
  9.99999894e-01 9.99999909e-01 9.99996817e-01 8.57458293e-05
  9.99999928e-01 9.99996869e-01 9.99999542e-01 9.99851463e-01
  7.35438161e-09 9.99986704e-01 9.99888945e-01 9.99087923e-01
  2.18546613e-08 4.51492559e-09 9.99999900e-01 9.99999802e-01
  9.99999916e-01 8.52725587e-09 9.87471816e-01 9.99999930e-01
  9.99998496e-01 5.63064672e-09 9.99999797e-01 5.51520727e-07
  9.99999881e-01 9.99999400e-01 1.35141435e-08 9.99957546e-01
  9.99999562e-01 9.99999920e-01 9.99999333e-01 1.19648245e-08
  9.94317694e-01 2.20822982e-08 7.65984424e-09 9.99999821e-01
  9.99696185e-01 1.69256539e-01 4.19982645e-09 2.09646681e-08
  2.02786412e-03 3.43503081e-07 9.99999921e-01 9.99999937e-01
  1.37629842e-07 1.22206347e-07 7.19693845e-03 4.12132975e-09
  1.48415703e-04 3.99490361e-09 9.99999933e-01 9.99999872e-01
  9.99999195e-01 4.41897635e-01 9.99999876e-01 9.99999935e-01
  9.99999822e-01 9.99999824e-01 4.17591732e-09 1.34221121e-08
  5.28819892e-05 7.44154776e-04 9.99994774e-01 9.95905936e-01
  9.99999839e-01 9.99999523e-01 9.99999553e-01 4.36691460e-08
  4.60761215e-09 9.99999924e-01 9.99999922e-01 9.99999932e-01
  1.29644037e-08 1.12551123e-08 9.99832923e-01 4.78510896e-09
  9.99822778e-01 2.18250129e-07 9.99998322e-01 1.26793847e-07
  9.07981653e-09 9.99999868e-01 2.37003976e-06 4.04051941e-09]]
accuracy: 99.0%
```

XOR training:

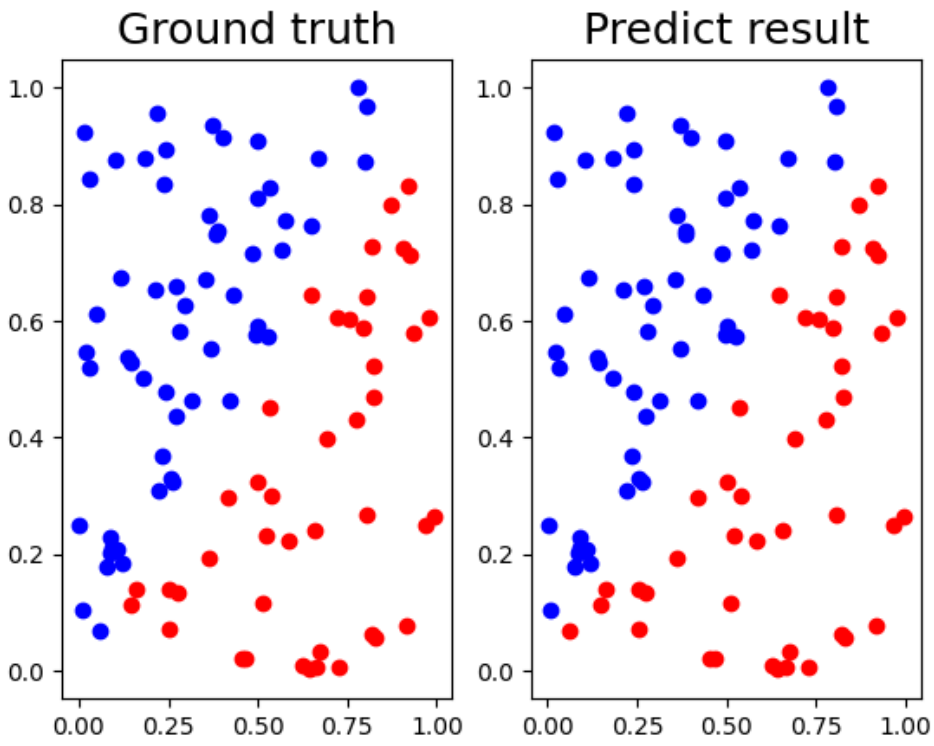
```
yucheng@ubuntu:~/Documents/DLP_lab$ python3 lab2.py
training...
epoch 0 loss: 1.2869219634860003
epoch 10 loss: 0.7071313069728244
epoch 20 loss: 0.6946744734706624
epoch 30 loss: 0.6944169617486617
epoch 40 loss: 0.6942649476089721
epoch 50 loss: 0.6941157982925045
epoch 60 loss: 0.6939685618382043
epoch 70 loss: 0.6938231136294728
epoch 80 loss: 0.6936793389333343
epoch 90 loss: 0.693537126257948
epoch 100 loss: 0.6933963671184183
epoch 110 loss: 0.6932569558494355
epoch 120 loss: 0.6931187894266824
epoch 130 loss: 0.6929817672972842
epoch 140 loss: 0.6928457912187856
epoch 150 loss: 0.6927107651060813
epoch 160 loss: 0.6925765948857613
epoch 170 loss: 0.692443188357373
epoch 180 loss: 0.6923104550611353
epoch 190 loss: 0.6921783061516816
epoch 200 loss: 0.6920466542774328
epoch 210 loss: 0.6919154134652384
epoch 220 loss: 0.6917844990099449
epoch 230 loss: 0.6916538273685813
epoch 240 loss: 0.6915233160588709
epoch 250 loss: 0.6913928835618027
epoch 260 loss: 0.6912624492280174
epoch 270 loss: 0.6911319331877782
epoch 280 loss: 0.6910012562643196
epoch 290 loss: 0.6908703398903769
epoch 300 loss: 0.6907391060277269
epoch 310 loss: 0.6906074770895686
epoch 320 loss: 0.6904753758656026
epoch 330 loss: 0.6903427254496712
epoch 340 loss: 0.6902094491698353
epoch 350 loss: 0.6900754705207818
epoch 360 loss: 0.6899407130984568
epoch 370 loss: 0.6898051005368394
epoch 380 loss: 0.689668556446776
epoch 390 loss: 0.689531004356807
epoch 400 loss: 0.6893923676559262
```

```
epoch 4600 loss: 0.11700773631416561
epoch 4610 loss: 0.11632512558908827
epoch 4620 loss: 0.1156471507595819
epoch 4630 loss: 0.11497374418597794
epoch 4640 loss: 0.11430483982699066
epoch 4650 loss: 0.11364037320859048
epoch 4660 loss: 0.11298028139326916
epoch 4670 loss: 0.1123245029497009
epoch 4680 loss: 0.11167297792280569
epoch 4690 loss: 0.11102564780421835
epoch 4700 loss: 0.11038245550316769
epoch 4710 loss: 0.10974334531776833
epoch 4720 loss: 0.1091082629067245
epoch 4730 loss: 0.10847715526145119
epoch 4740 loss: 0.10784997067860913
epoch 4750 loss: 0.10722665873305486
epoch 4760 loss: 0.10660717025120427
epoch 4770 loss: 0.1059914572848085
epoch 4780 loss: 0.10537947308513987
epoch 4790 loss: 0.10477117207758314
epoch 4800 loss: 0.10416650983663309
epoch 4810 loss: 0.10356544306128905
epoch 4820 loss: 0.10296792955084826
epoch 4830 loss: 0.10237392818108776
epoch 4840 loss: 0.10178339888083598
epoch 4850 loss: 0.10119630260892323
epoch 4860 loss: 0.1006126013315107
epoch 4870 loss: 0.1000322579997887
epoch 4880 loss: 0.09945523652804233
epoch 4890 loss: 0.09888150177207448
epoch 4900 loss: 0.09831101950798488
epoch 4910 loss: 0.09774375641129494
epoch 4920 loss: 0.09717968003641705
epoch 4930 loss: 0.09661875879645732
epoch 4940 loss: 0.09606096194335023
epoch 4950 loss: 0.09550625954831723
epoch 4960 loss: 0.09495462248264369
epoch 4970 loss: 0.0944060223987675
epoch 4980 loss: 0.09386043171167628
epoch 4990 loss: 0.09331782358060509
```

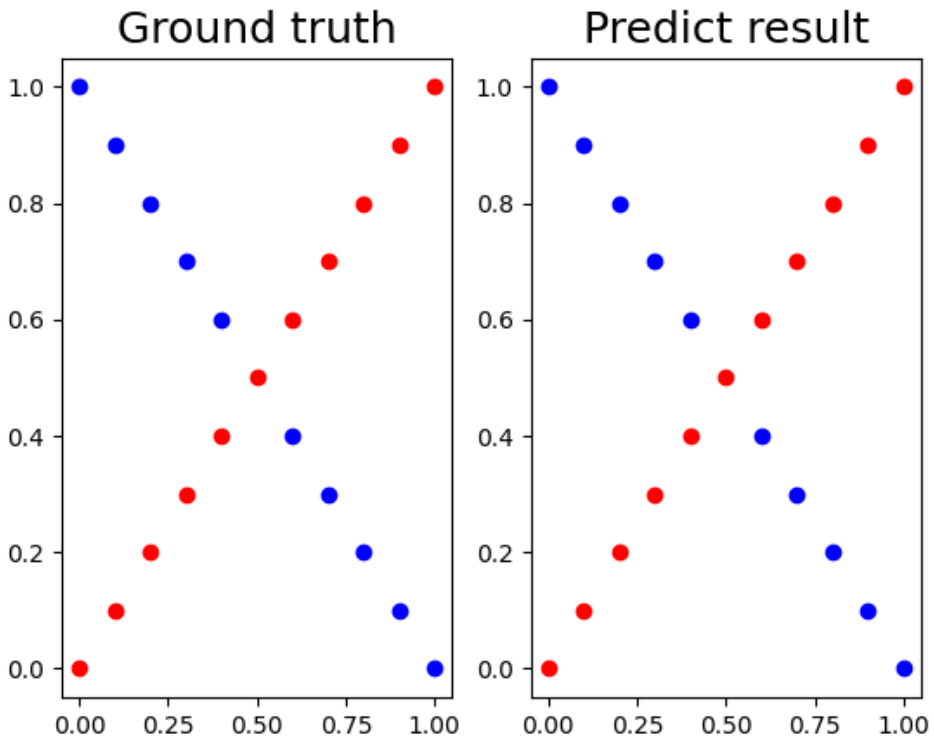
XOR testing:

```
testing...
[[0.00874118 0.99361994 0.0083291  0.99274139 0.0140075  0.98797288
  0.06787061 0.94943452 0.23765186 0.59213496 0.25048922 0.13900079
  0.67399327 0.06002461 0.98362245 0.02614197 0.99593379 0.01285487
  0.9953626  0.00727795 0.99035646]]
accuracy: 100.0%
```

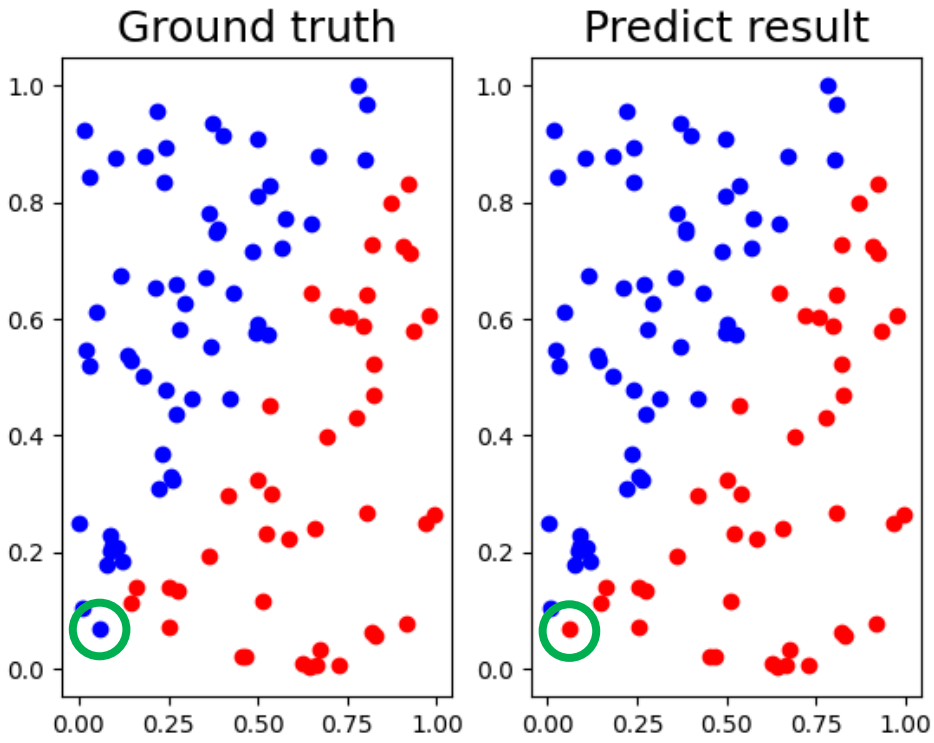
linear figure:


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

XOR figure:

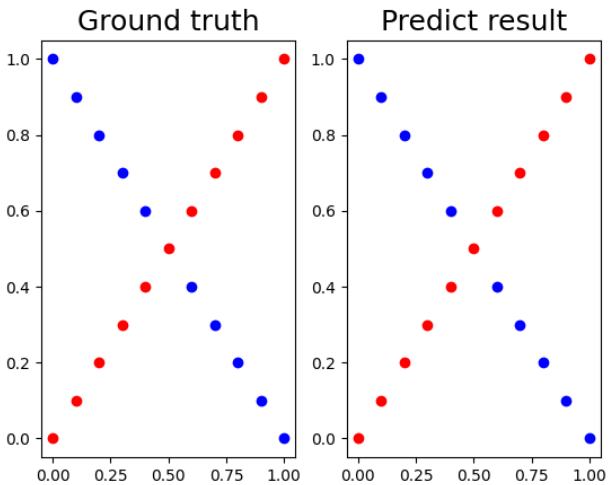

Ground truth
Predict result
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00

B. Show the accuracy of your prediction
linear: 99%

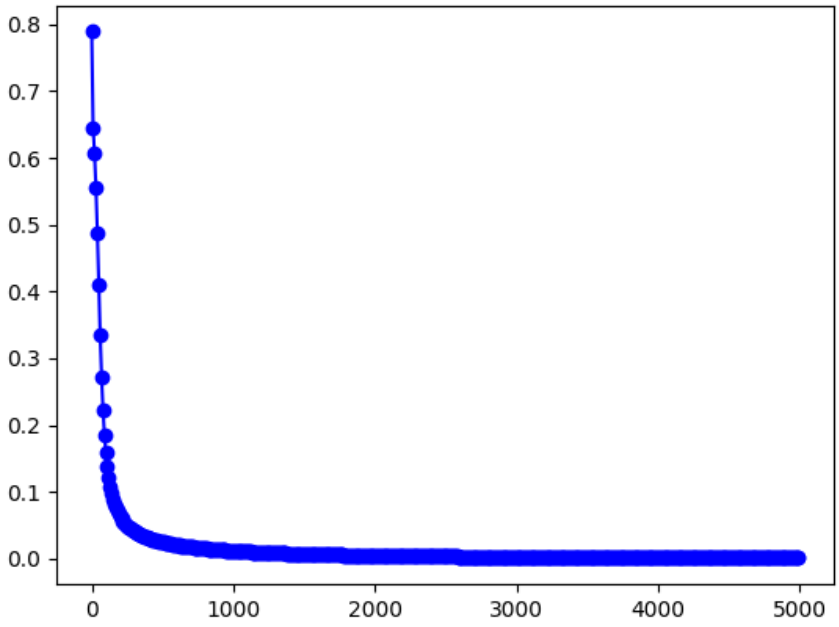


綠色圈圈為 100 個點中唯一錯的一個點。

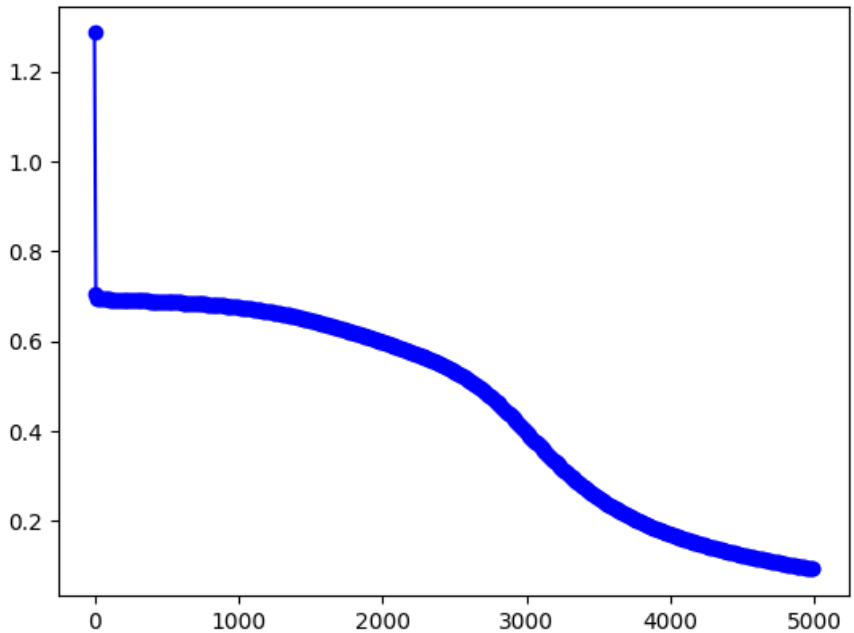
XOR: 100%



C. Learning curve (loss, epoch curve) (x: epoch, y: loss)
linear:



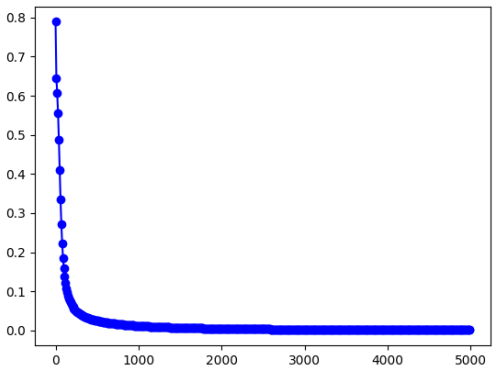
XOR:

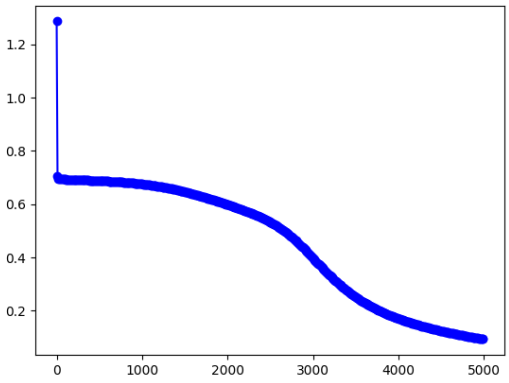


Discussion

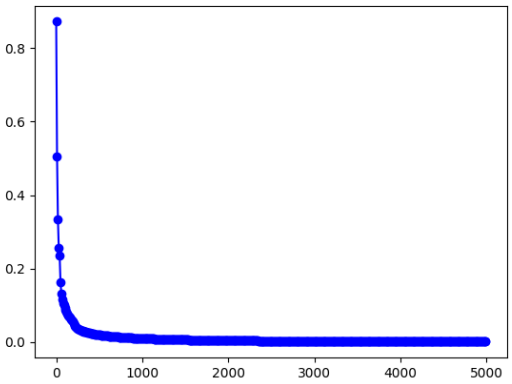
A. Try different learning rates (左: linear learning curve；右: XOR)

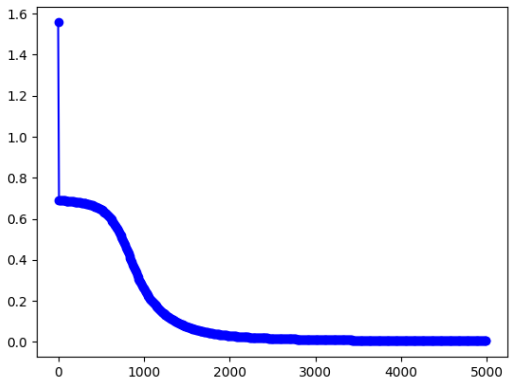
origin: learning rate = 0.1　accuracy = 99% , 100%



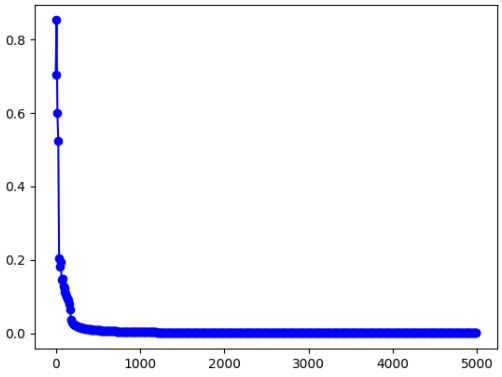


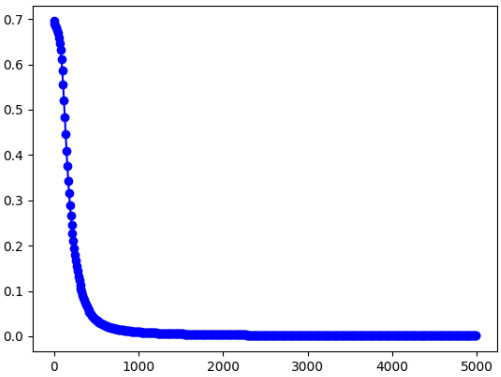
learning rate = 0.25　accuracy = 99% , 100%



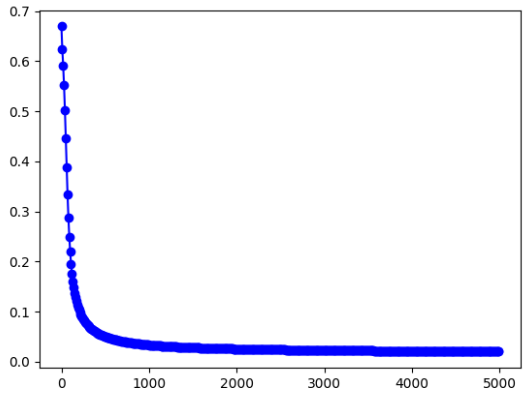


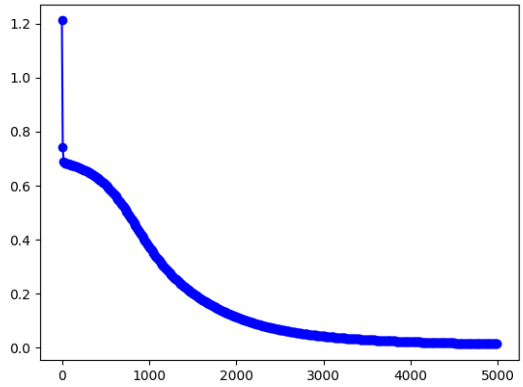
learning rate = 0.5　accuracy = 99% , 100%



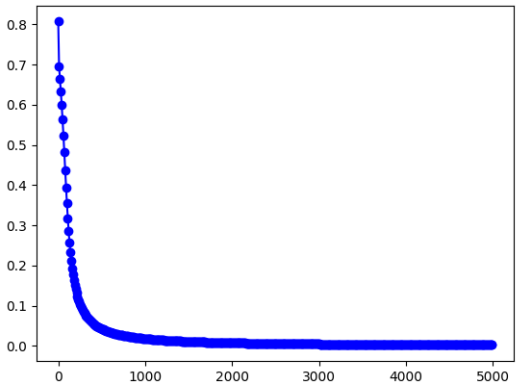


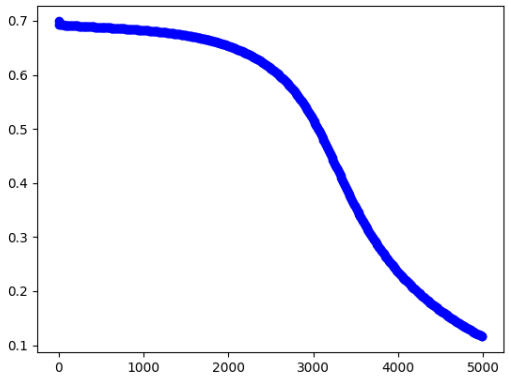
learning rate = 0.075　accuracy = 98% , 100%





learning rate = 0.05　accuracy = 98% , 100%





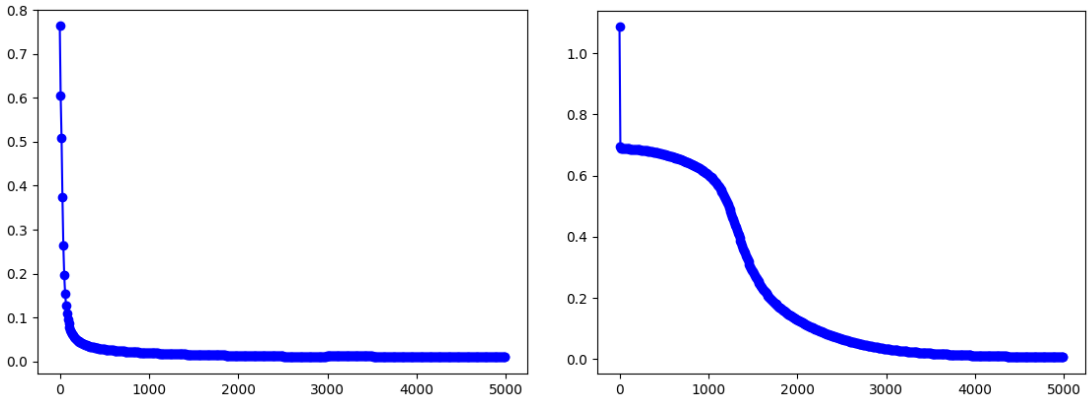
結論

當 learning rate 越大時，向著 gradient 方向一步走得越大步，因此能夠越快達到收斂。這可以從 linear 前期點的密集程度以及 linear 和 XOR 的圖形走勢得知。

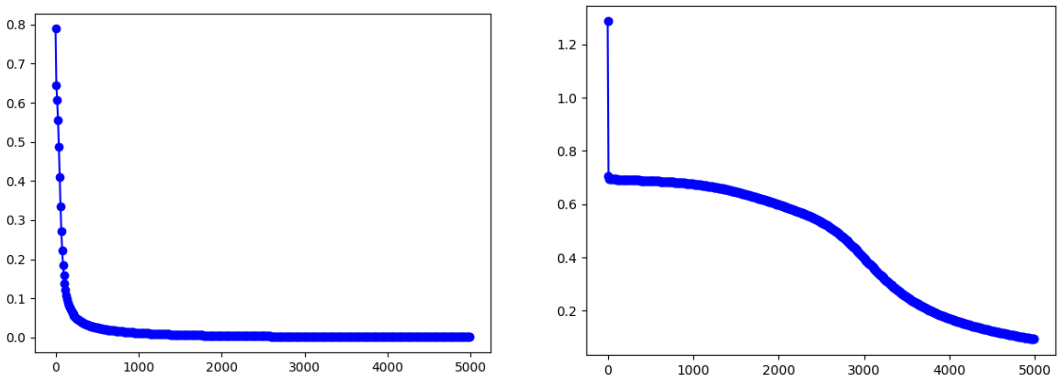
B. Try different numbers of hidden units

改變 W(nxn)大小，就能改變乘出來的 hidden layer 大小。

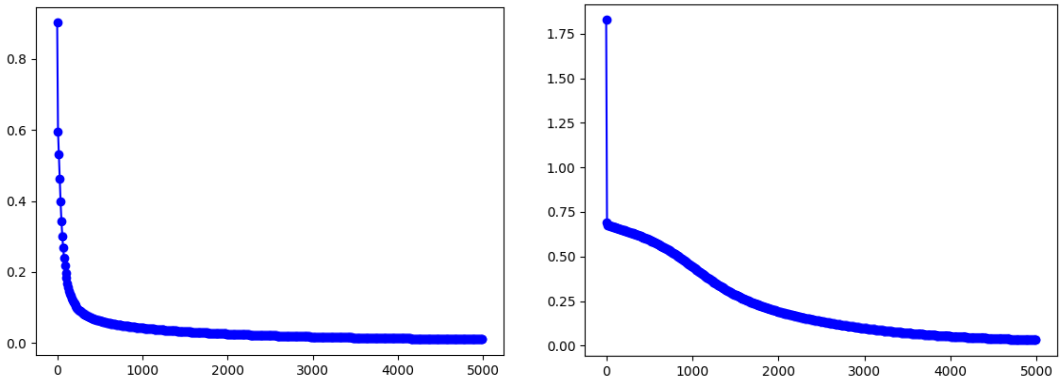
n = 5



origin: n = 10



n = 15



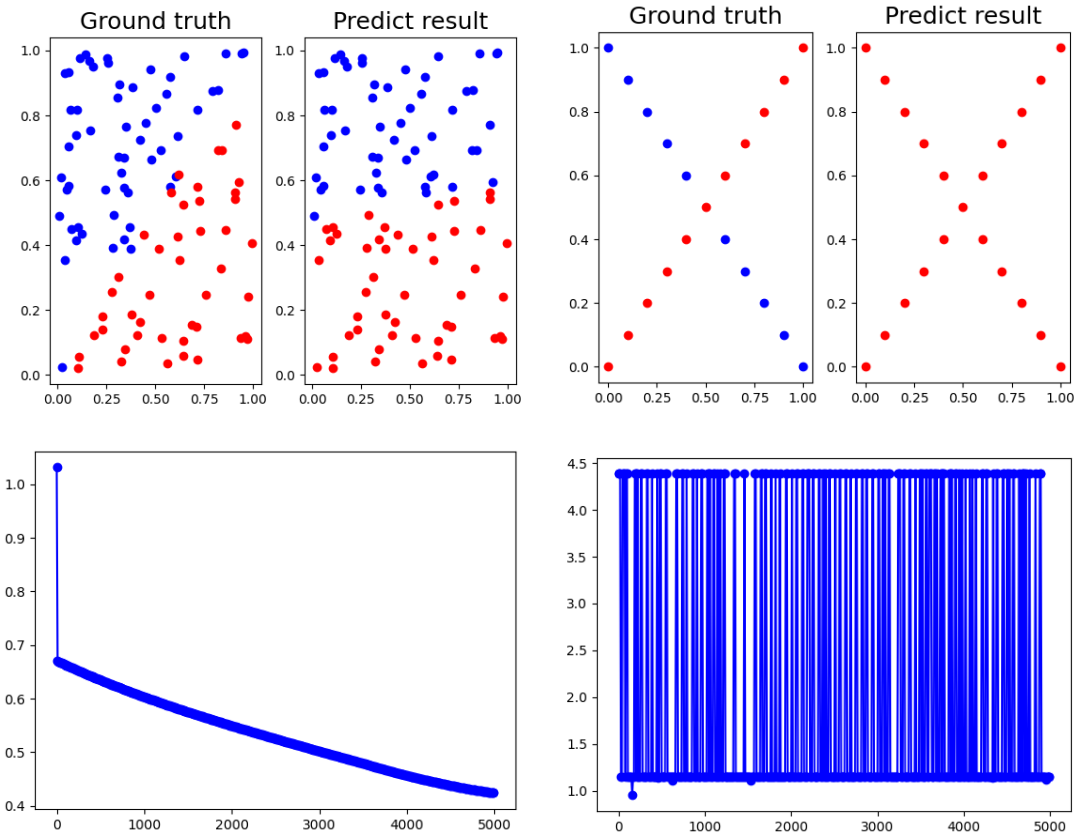
結論

n 越大、hidden units 越多時，曲線較為平緩、穩定。XOR 為明顯。

C. Try without activation functions

因 ground truth 是 0 或 1，就算不用 activation function 也必須將輸出值轉成 0 到 1 之間。

without sigmoid　accuracy: 82% , 52%

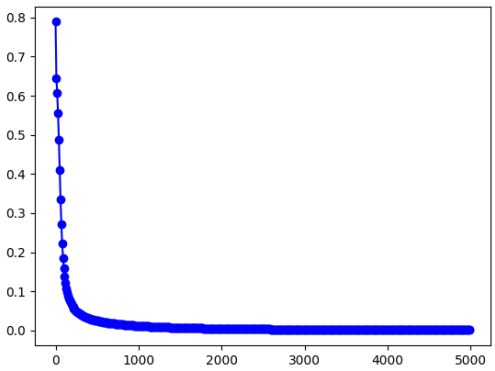


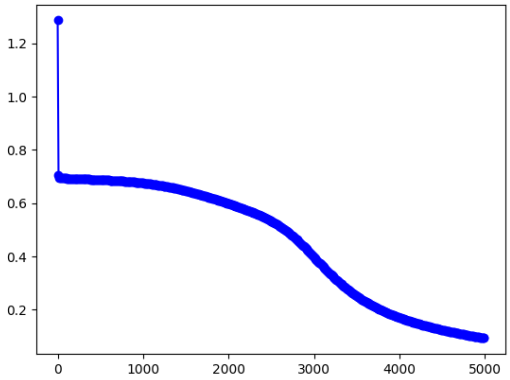
結論

不用 sigmoid 的情況下 linear 還勉強能達到 82%的準確率，且 loss 有確實在下降；而 XOR 就沒辦法預測，loss 也呈現忽高忽低的震盪現象。

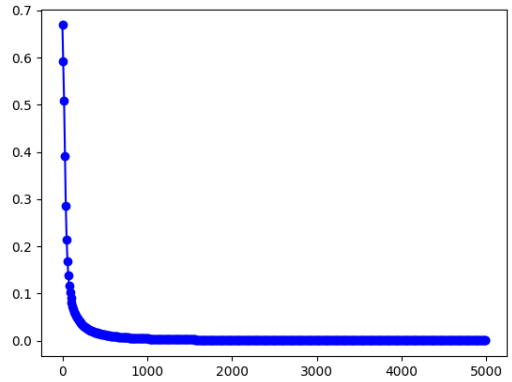
D. Try different numbers of hidden layers

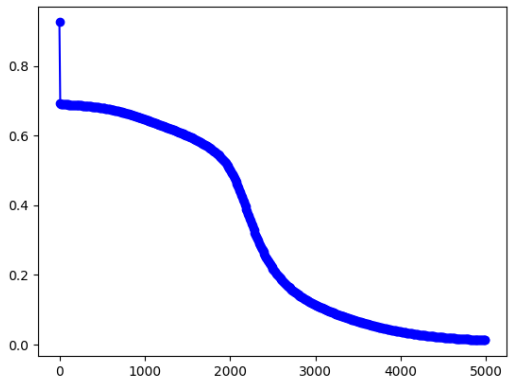
origin: 2 hidden layers



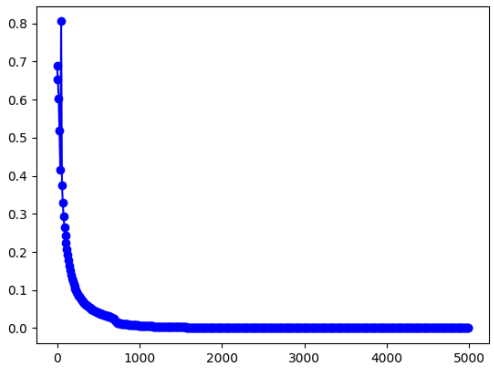


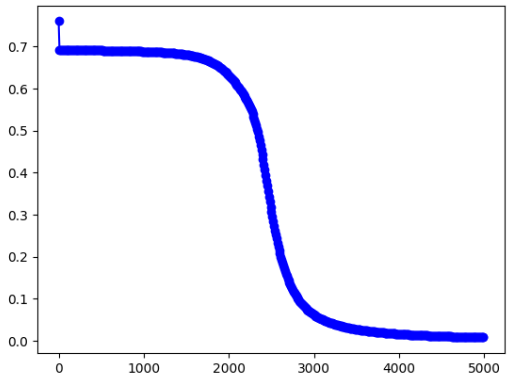
3 hidden layers





4 hidden layers





結論

hidden layer 數越多，曲線越陡峭、收斂得越快。XOR 尤為明顯。

Extra

Implement different activation functions

Implement ReLU:

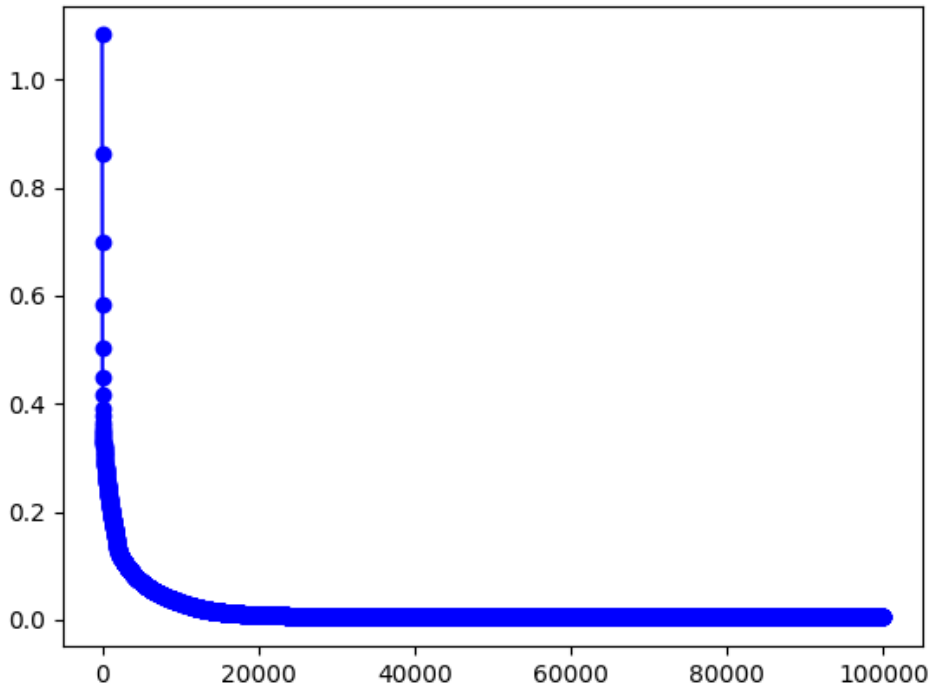
```
def ReLU(x):
    x = np.maximum(0, x)
    return x

def derivative_ReLU(x):
    x[x <= 0] = 0
    return x
```

將 a1 = sigmoid(z1) 和 a2 = sigmoid(z2) 改成 a1 = ReLU(z1) 和 a2 = ReLU(z2)。最後一層要 output 的 a3 = sigmoid(z3) 則維持不變，因為輸出的值域必須在 0 到 1 之間。backpropagation 的地方同理反推。

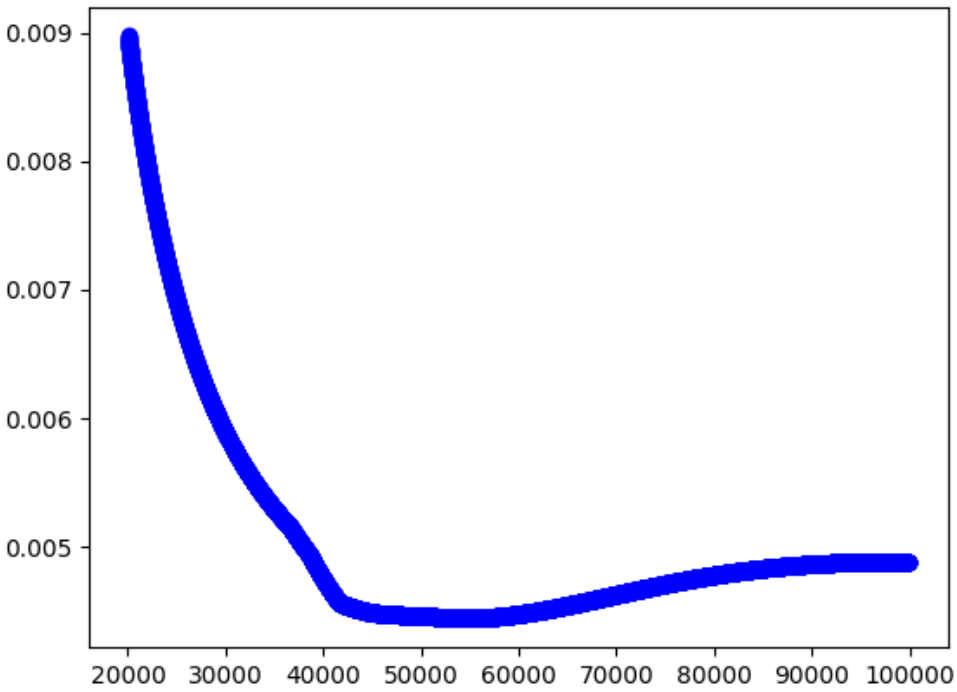
```
def forward(a0, w1, w2, w3):
    z1 = np.dot(w1, a0)
    a1 = ReLU(z1)
    #a1 = sigmoid(z1)
    z2 = np.dot(w2, a1)
    a2 = ReLU(z2)
    #a2 = sigmoid(z2)
    z3 = np.dot(w3, a2)
    #a3 = ReLU(z3)
    a3 = sigmoid(z3)

    return a3, a0, z1, a1, z2, a2, z3, a3

def cost(pred_y, y):
    # cross entropy
    return float(-(1/y.shape[1])*(y @ np.log(pred_y + 0.0001).T+(1-y) @ np.log(1-pred_y + 0.0001).T))

def backpropagation(pred_y, y, learning_rate, n, a0, w1, z1, a1, w2, z2, a2, w3, z3, a3):
    dJ_da3 = -(y / (pred_y + 0.0001) - (1 - y) / (1 - pred_y + 0.0001))
    dJ_dz3 = derivative_sigmoid(a3) * dJ_da3
    #dJ_dz3 = derivative_ReLU(z3) * dJ_da3
    dJ_dw3 = np.dot(dJ_dz3, a2.T)

    dJ_da2 = np.dot(w3.T, dJ_dz3)
    #dJ_dz2 = derivative_sigmoid(a2) * dJ_da2
    dJ_dz2 = derivative_ReLU(z2) * dJ_da2
    dJ_dw2 = np.dot(dJ_dz2, a1.T)

    dJ_da1 = np.dot(w2.T, dJ_dz2)
    #dJ_dz1 = derivative_sigmoid(a1) * dJ_da1
    dJ_dz1 = derivative_ReLU(z1) * dJ_da1
    dJ_dw1 = np.dot(dJ_dz1, a0.T)

    w1 = w1 - learning_rate * (dJ_dw1 / n)
    w2 = w2 - learning_rate * (dJ_dw2 / n)
    w3 = w3 - learning_rate * (dJ_dw3 / n)

    return w1, w2, w3
```

test result

linear:
accuracy: 100%
learning curve:

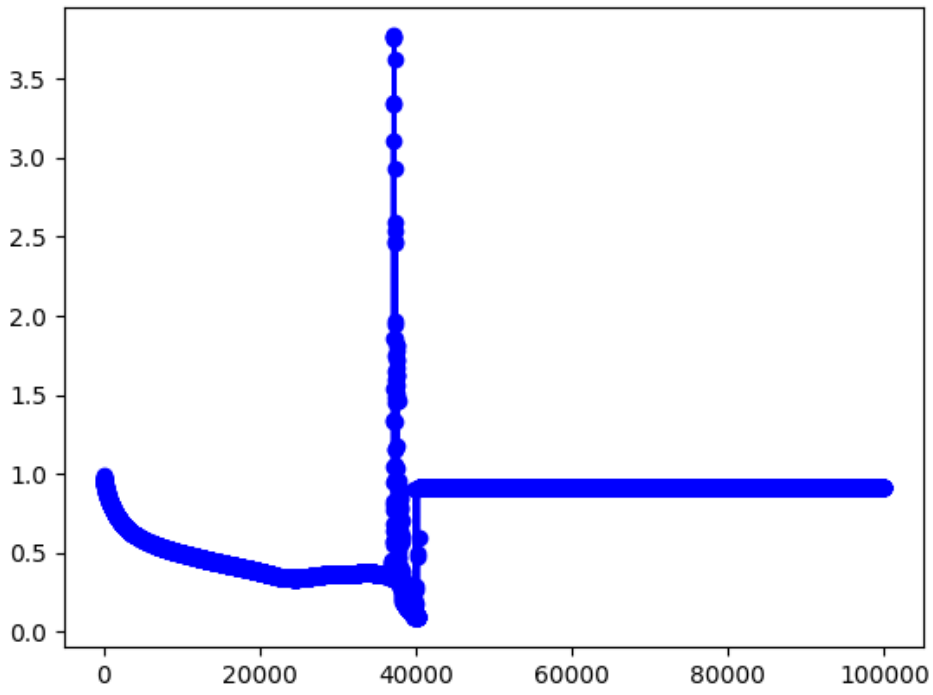


乍看之下沒問題，但當我們將 20000 到 100000 放大來看:

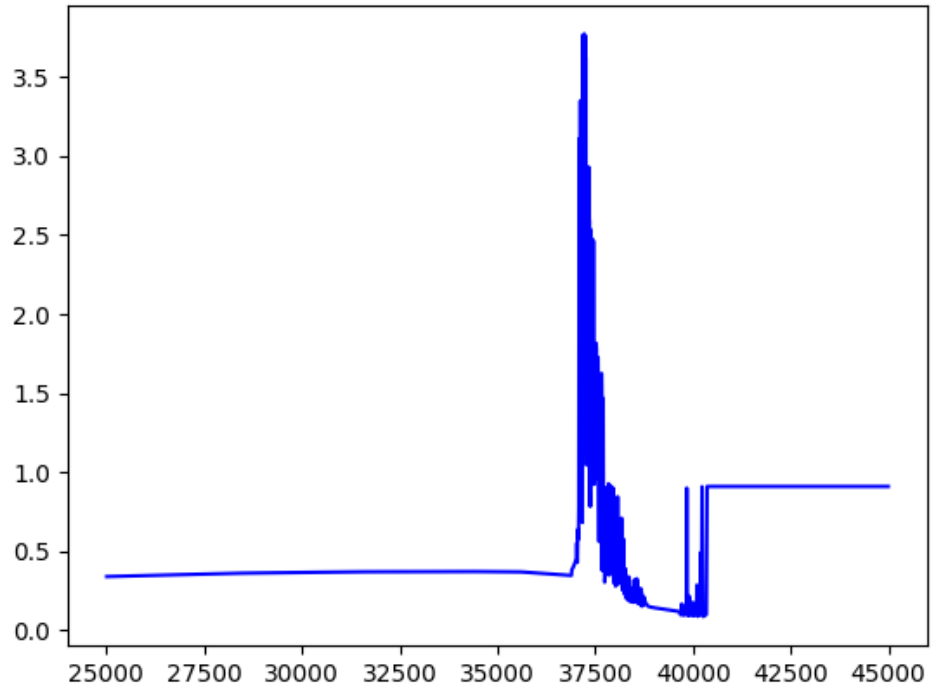


雖然準確度依然是 100%，但可以看到 loss 值其實已經過最低點了。

這個問題在 XOR 更為明顯:

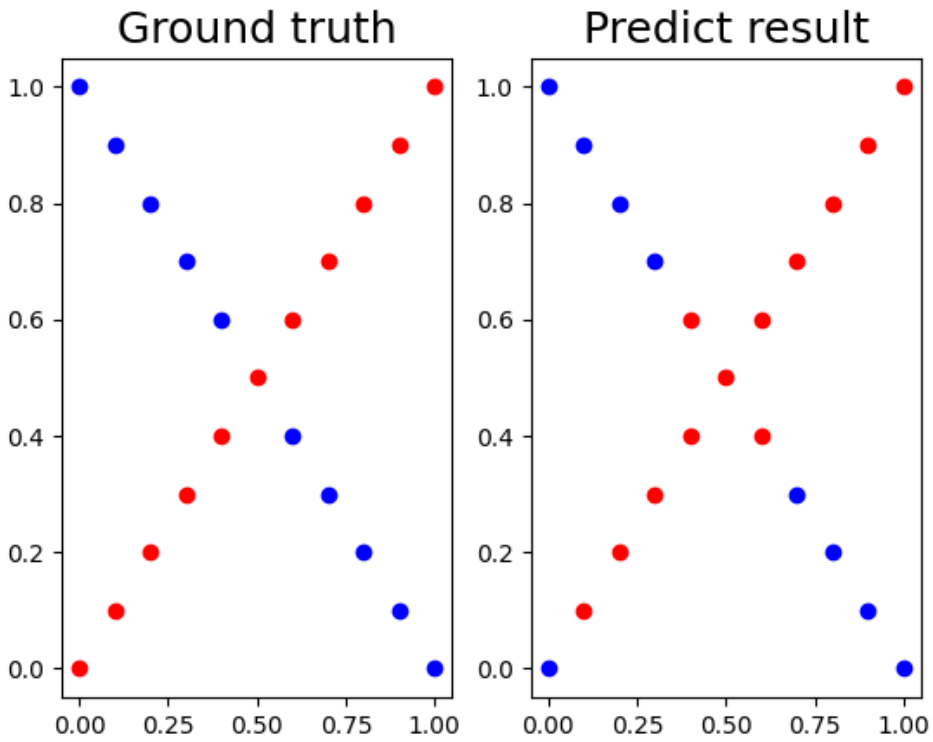


將 25000 到 45000 放大來看:



綜合上面兩張圖來看，loss 大概在 20000 多時達到最低點，接著開

始微幅上升，並在 36000 到 40000 初來了一波不穩定的震盪，之後進入穩定期。但穩定期給出的 loss 也不是 global 的 minimum。最後預測出的結果也不佳，21 個點只中了 18 個。



因此，在這兩個 case 中，我認為 sigmoid 都能表現得比 ReLU 更好。