Report Spec

1. Introduction

在 Lab1 實作中，主要為練習與了解當 output 出現類似 XOR 這種無法用 linear function 來準確分別結果時，我們會需要使用 nonlinear 的 activation function 例如 : sigmoid,relu,softmax…

但當使用這種 nonlinear function 時，gradient descent 會變得難尋找到其中參數的最佳解，而本次實驗將使用 backpropagation 來快速尋找各個 weight 對應的 gradient descent，進而在每次的 step 中更新 weight 值

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 (hidden layer 可自己設定)

```
self.x = np.zeros((2,1))
self.w = [np.random.rand(hidden_size[0],2),np.random.rand(hidden_size[1],hidden_size[0]),np.random.rand(1,hidden_size[1])
self.z = [np.zeros((hidden_size[0],1)),np.zeros((hidden_size[1],1)),np.zeros((1,1))]
self.a = [np.zeros((hidden_size[0],1)),np.zeros((hidden_size[1],1)),np.zeros((1,1))]
```

C. Backpropagation

```
if self.costf:
    self.cost = np.vdot((actual-pred),(actual-pred))
    pCa2 = -2*(actual-pred)
      print("mse")
else:
    self.cost = -(actual*math.log(pred+1e-9)+(1-actual)*math.log(1-pred+1e-9))
    pCa2 = -(actual/(pred+self.EPS)-(1-actual)/(1-pred+self.EPS))
      print("ce")
pa2z2 = de_sigmoid(self.a[2])
pCz2 = pCa2*pa2z2
pCw2 = self.a[1]*pCz2
pCw2 = pCw2.T
```

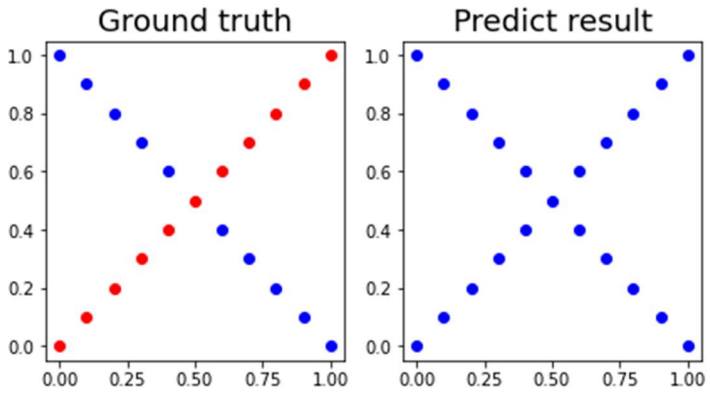
```
pz1w1 = self.a[0]
pCa1 = pCz2.T@self.w[2]
pa1z1 = de_sigmoid(self.a[1])
n = pCa1.shape[1]
repCa1 = []
for j in range(n):
    repCa1.append(pCa1[0][j])
    if j != n-1:
        for k in range(n):
            repCa1.append(0)
repCa1 = np.array(repCa1).reshape(n,n)
pCz1 = repCa1@pa1z1
pCw1 = pCz1@pz1w1.T
```

```
pz0w0 = self.x
pCa0 = pCz1.T@self.w[1]
pa0z0 = de_sigmoid(self.a[0])
n = pCa0.shape[1]
repCa0 = []
for j in range(n):
    repCa0.append(pCa0[0][j])
    if j != n-1:
        for k in range(n):
            repCa0.append(0)
repCa0 = np.array(repCa0).reshape(n,n)
pCz0 = repCa0@pa0z0
pCw0 = pCz0@pz0w0.T
```

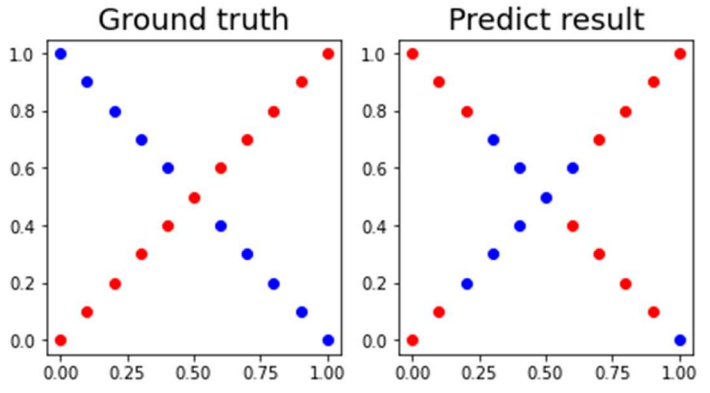
3. Results of your testing

A. Screenshot and comparison & B. figureShow the accuracy of your prediction

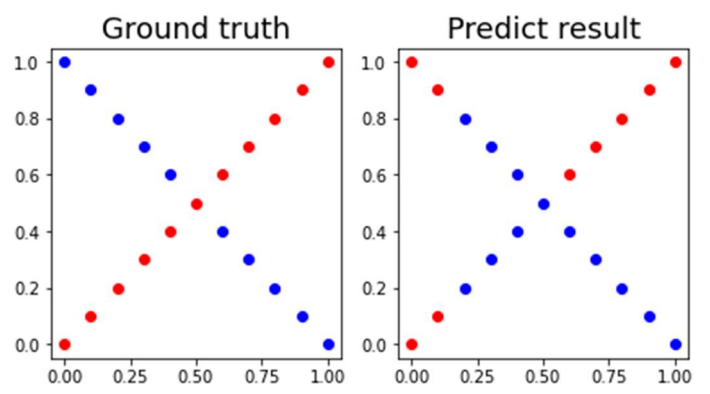
```
epoch 0 loss : 0.47918790877492445  accuracy : 47.62%
[[0.88119007]
 [0.97194278]
 [0.91874194]
 [0.97133636]
 [0.942593  ]
 [0.9708173 ]
 [0.95688569]
 [0.97038345]
 [0.96529309]
 [0.97001438]
 [0.97023914]
 [0.97313604]
 [0.96936909]
 [0.97483306]
 [0.96908007]
 [0.97578375]
 [0.96879441]
 [0.97626563]
 [0.96850585]
 [0.97644653]
 [0.96821002]]
```



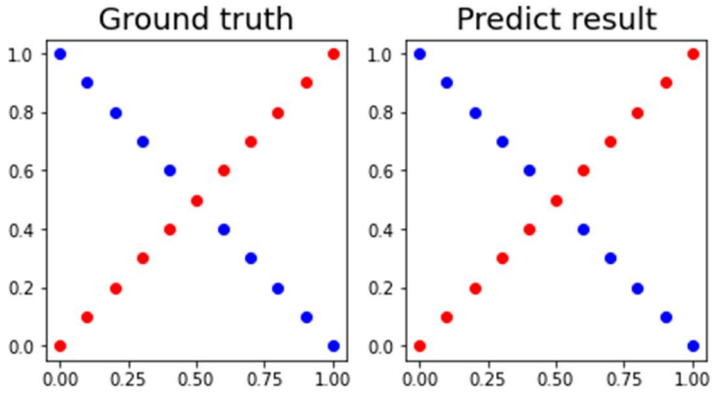

```
epoch 250 loss : 0.24970534221050272  accuracy : 42.86%
[[0.44895431]
 [0.4820663 ]
 [0.48441823]
 [0.49244918]
 [0.51253034]
 [0.49934293]
 [0.52845933]
 [0.50359663]
 [0.53295879]
 [0.5061277 ]
 [0.5293647 ]
 [0.50010677]
 [0.48934981]
 [0.49085035]
 [0.49204741]
 [0.48113536]
 [0.49484812]
 [0.47176887]
 [0.49787084]
 [0.46314309]
 [0.50117671]]
```




```
epoch 500 loss : 0.2264796673710582  accuracy : 71.43%
[[0.32442558]
 [0.48953944]
 [0.45089613]
 [0.49904812]
 [0.54017268]
 [0.5040007 ]
 [0.5748198 ]
 [0.50757963]
 [0.57283578]
 [0.51197474]
 [0.55118121]
 [0.48943654]
 [0.50196873]
 [0.45707183]
 [0.52036095]
 [0.42503241]
 [0.54310851]
 [0.3947015 ]
 [0.56896021]
 [0.36662858]
 [0.59598477]]
```




```
epoch 750 loss : 0.02842777862349817  accuracy : 100.00%
[[0.23212474]
 [0.90879809]
 [0.24579059]
 [0.90914904]
 [0.23057454]
 [0.90767373]
 [0.19194941]
 [0.89382885]
 [0.15457142]
 [0.67315256]
 [0.14133918]
 [0.11976361]
 [0.71386801]
 [0.11024015]
 [0.86609201]
 [0.10091746]
 [0.86791151]
 [0.09499006]
 [0.8686261 ]
 [0.09124609]
 [0.86933274]]
```

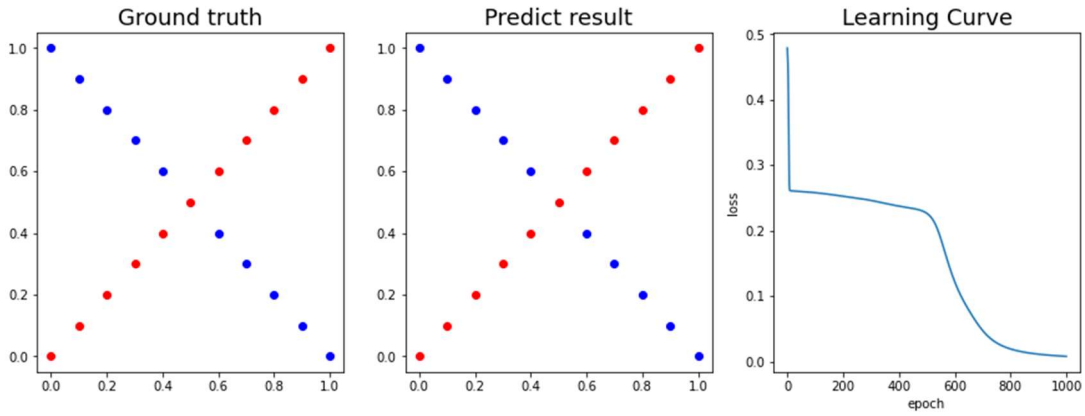


C. Learning curve(loss, epoch curve)

hyperparmeter set : hidden_size = (8,8) / learning_rate = 0.05 / epoch = 1000 /

activation function(hide1 / hide2 / output_layer) = (no / sigmoid / sigmoid)

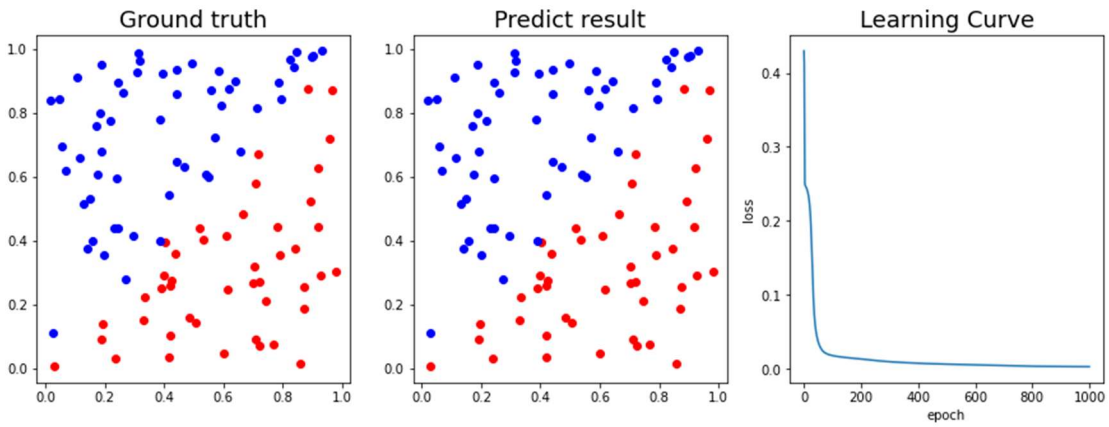
```
epoch 1000 loss : 0.007880216448060343  accuracy : 100.00%
[[0.18917526]
 [0.94154705]
 [0.13103763]
 [0.94345171]
 [0.09516259]
 [0.94501028]
 [0.0759052 ]
 [0.94504624]
 [0.0656459 ]
 [0.86007404]
 [0.06126302]
 [0.05736602]
 [0.86539607]
 [0.05539473]
 [0.92014172]
 [0.05396434]
 [0.92138334]
 [0.05308321]
 [0.92244249]
 [0.05254986]
 [0.92340615]]
```



D. anything you want to present

the result of linear case

在 hyperparmeter 不變為前提之下，loss 降的速度快 nonlinear 很多



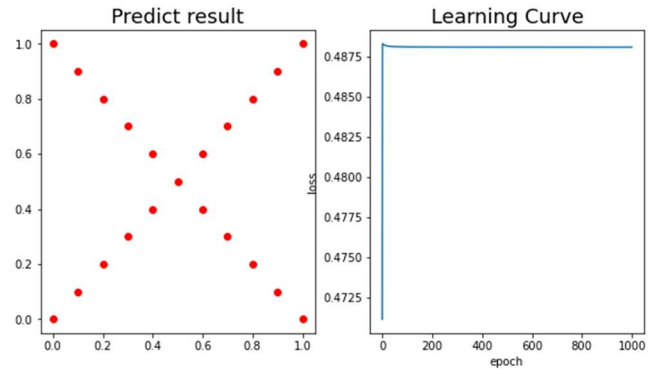
4. Discussion

A. Try different learning rates

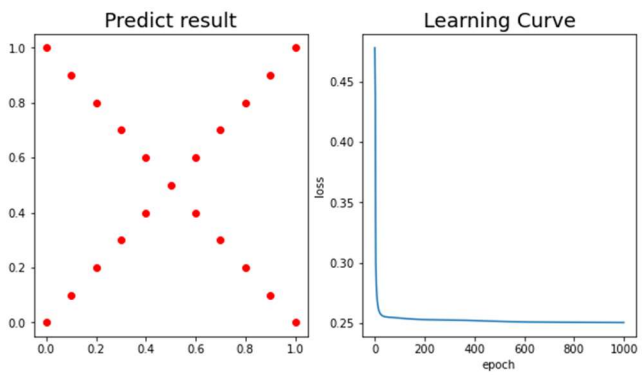
hyperparameter :

epoch 1000 , hide_layer(8,8), cross entropy , activation funt : (hide1 : no ,hide2 : relu, hide3 : sigmoid)

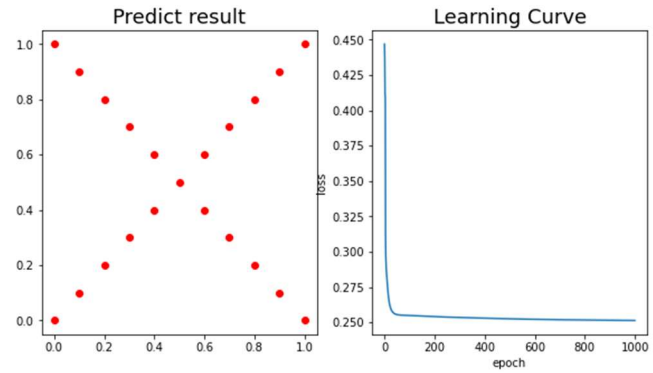
1. lr = 5



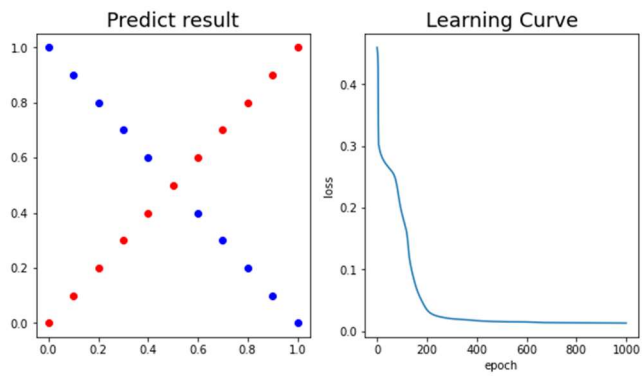
2. lr = 1



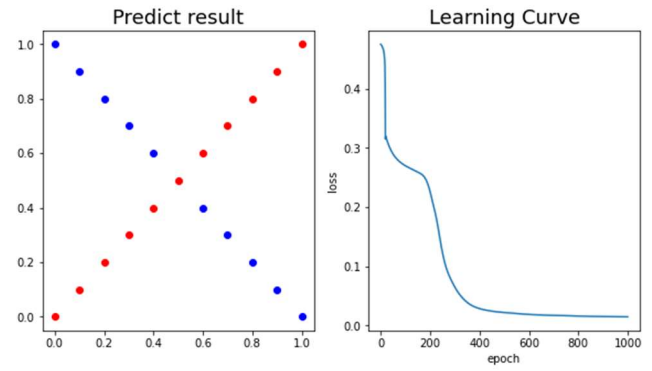
3. 0.5



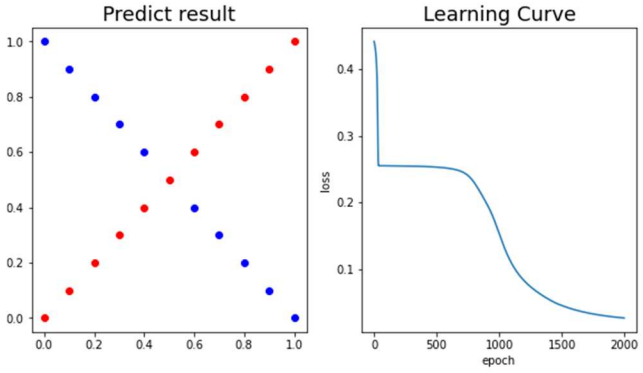
4. 0.1



5. 0.05



6. 0.01



總結 ：

learning_rate 過小 > 要 train 較久 ， 右下 lr=0.01 要等到 800+才開始降 loss

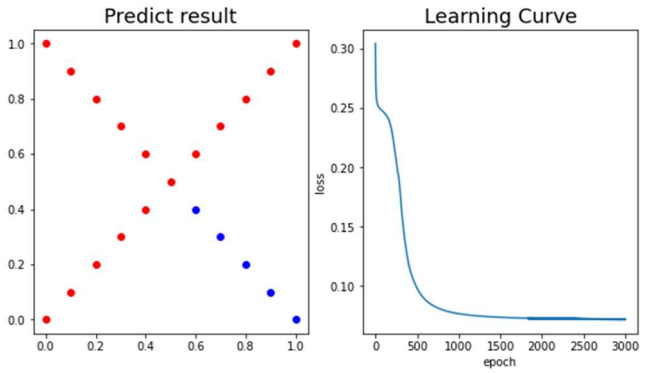
learning_rate 過大 > 找不到 optimal 的地方

B. Try different numbers of hidden units

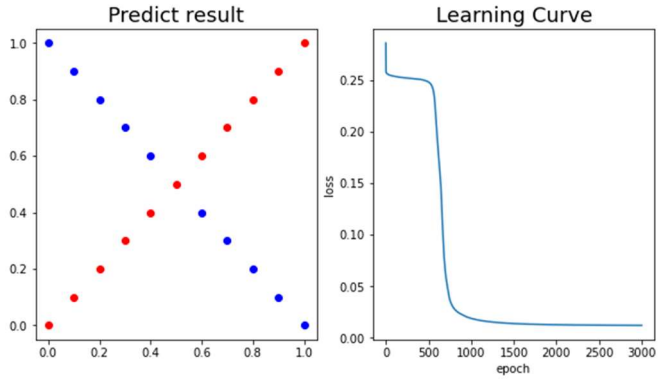
hyperparameter : epoch 3000, learning_rate 0.05, cross entropy

activation funt : (hide1 : no ,hide2 : relu, hide3 : sigmoid)

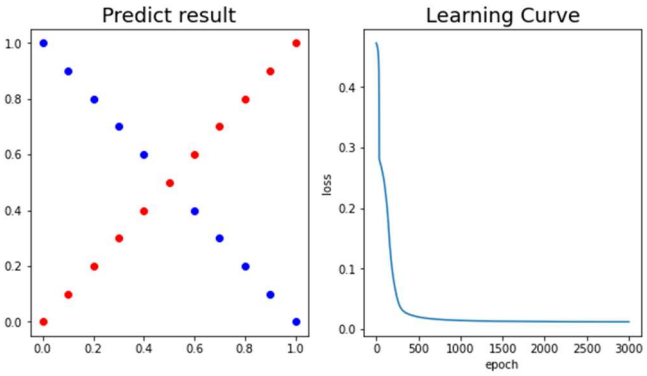
1. hidden_unit = (2,2)



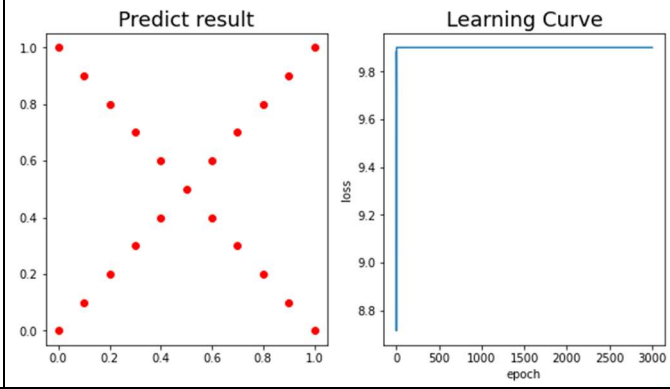
2. hidden_unit = (4,4)



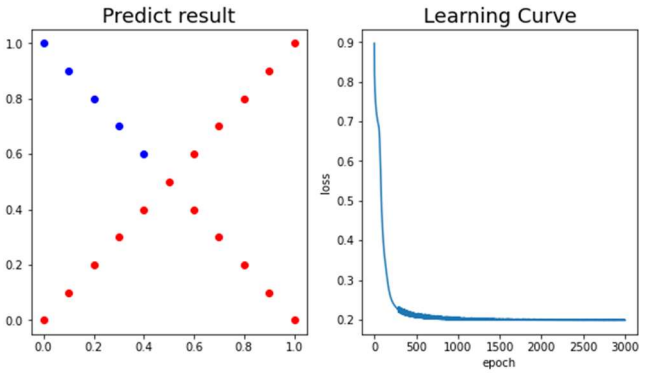
3. (8,8)



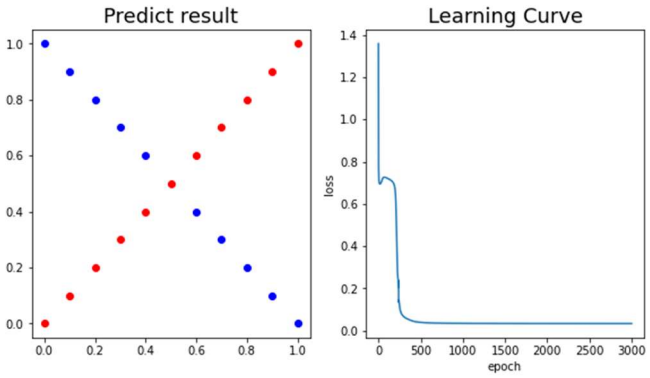
4. (32,32)



5. (16,2)



6. (2,16)



總結 : hidden_unit

過多 > code runtime 時間變長 ,適當的數量可以增加訓練速度(loss 降的快)

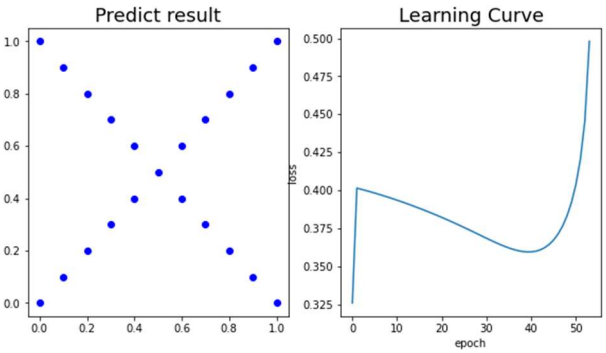
過少 > loss 卡住 train 不起來，適當的數量減少 code runtime

另外有發現後面的 hidden_unit 較多效果比較好（左下、右下）

C. Try without activation functions lazy to use

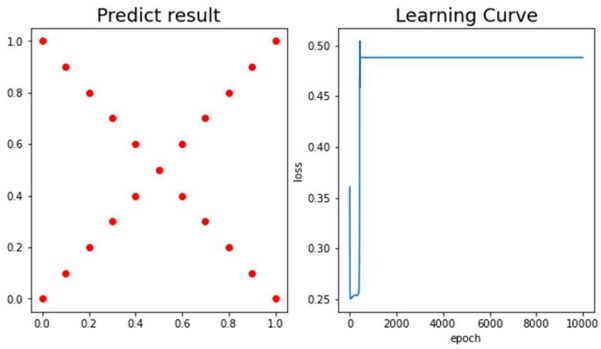
hyperparameter : epoch 10000, learning_rate 0.05, , hidden_unit(4,4), mean square error

1. (no , no , no)



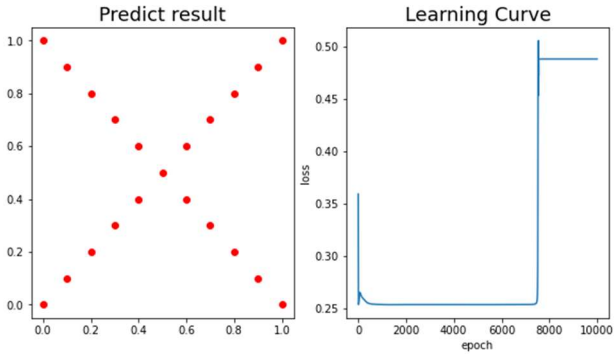
loss 會有 overflow 問題 擴散

2. (no , no ,sigmoid)



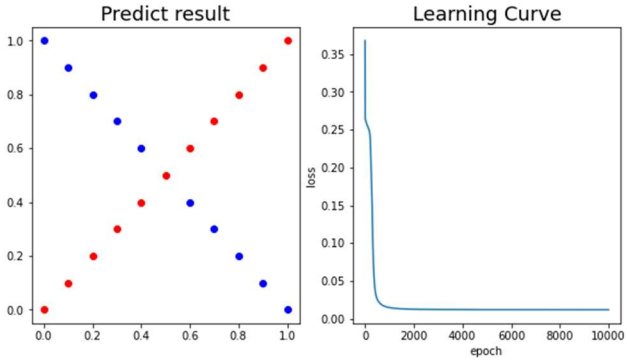
雖然沒有擴散的問題，但沒有辦法有效的收斂

3. (relu, no , sigmoid) (穿插)



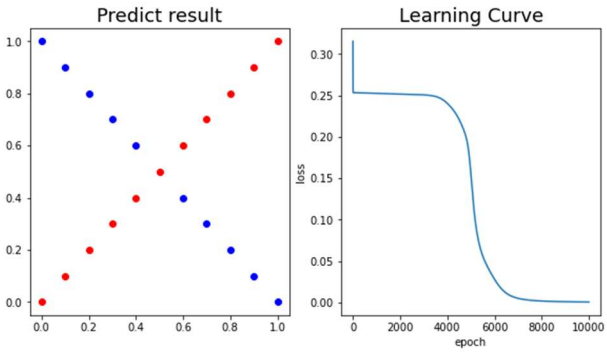
穿插使用 無法收斂

4. (no , relu, sigmoid)



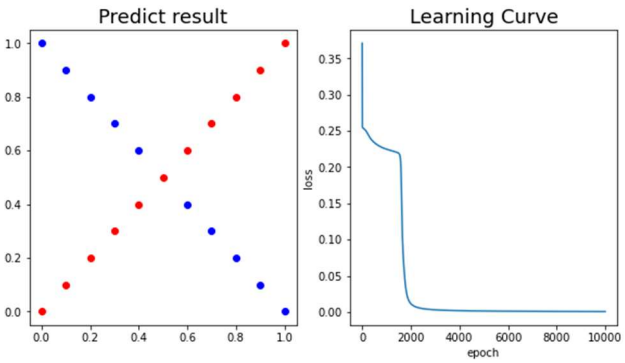
減少第一層的使用 效果最好

5. (sigmoid , sigmoid , sigmoid)



可以收斂但速度慢

6. (relu , sigmoid , sigmoid)



可以收斂但速度較慢

D. Anything you want to share

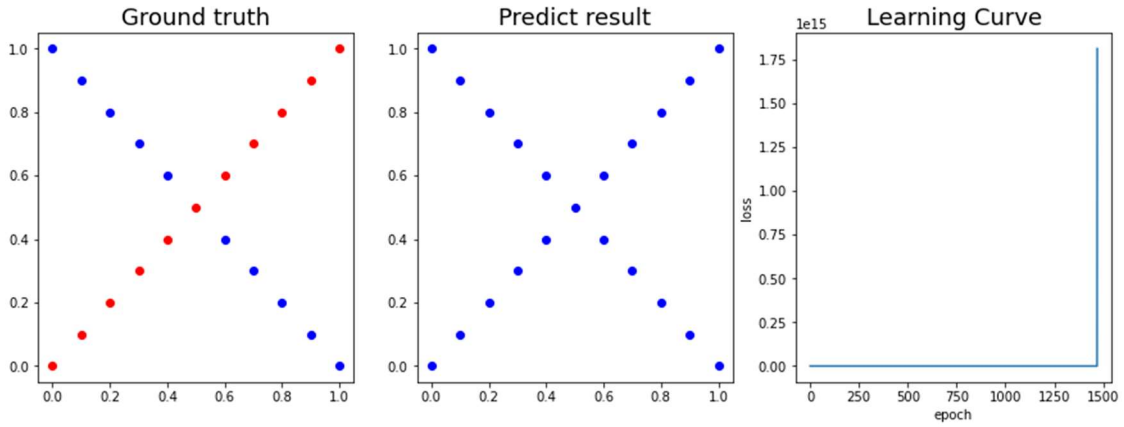
output without activation function 部分討論

首先 without activation function 無法使用太大的 hidden unit ，若使用過多的 unit，
容易因為初始 weight 設定的關係，造成 loss 上的 overflow

1. 另外會因為答案屬於 linear or nonlinear 分布，產生不同的結果，實驗如下

hyperparameter : epoch 10000, learning_rate 0.05, , hidden_unit(2,2), mean square error
activation funt : (hide1 : no ,hide2 : no, hide3 : no)

・若答案屬於 nonlinear 分布 則 Loss 無法收斂



・但答案屬於 linear 分布時，雖不到 100%正確但至少有個大概的方向

```
epoch 10000 loss : 0.129771943231645  accuracy : 82.00%
```