

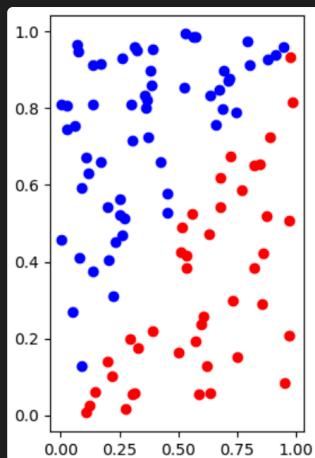
Lab 1 Back-propagation

313551055 柯柏旭

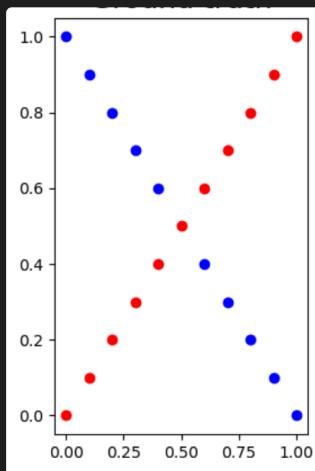
1. Introduction

在本報告中，我們將探討使用反向傳播算法訓練神經網絡的過程與結果。在 section **4.Discussion** 和 **5.Extra**，我們將利用不同的學習率、激活函數、和優化器等等來訓練，目的是評估這些因素對模型性能的影響。通過詳細的測試和討論，我們將展示這些方法如何影響網絡在處理**Linear數據**和**XOR數據**時的學習和預測能力，以下為Linear數據和XOR數據的呈現方式：

- Linear data



- XOR data



對於 section **2.Experiment setups** 和 **3.Result of testing** 皆由 base code 的 **main.py** 來陳述。

至於 section **4.Discussion** 和 **5.Extra** 則會另外寫成 **diff_** 開頭的 codes。

2. Experiment setups

Hardware overview

```
Model Name: MacBook Pro  
Model Identifier: Mac15,6  
Model Number: MRX33TA/A  
Chip: Apple M3 Pro  
Total Number of Cores: 11 (5 performance and 6 efficiency)  
Memory: 18 GB
```

Python version

```
Python 3.9.19
```

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)
```

(refer to lab1 document)

B. Neural network

```
def forward_propagation(X, weights):  
    Z1 = np.dot(X, weights['W1']) + weights['b1']  
    A1 = sigmoid(Z1)  
    Z2 = np.dot(A1, weights['W2']) + weights['b2']  
    A2 = sigmoid(Z2)  
    Z3 = np.dot(A2, weights['W3']) + weights['b3']  
    A3 = sigmoid(Z3)  
    return Z1, A1, Z2, A2, Z3, A3
```

Each hidden layers is $y = wx + b$ and *sigmoid* function

C. Backpropagation

```

def back_propagation(X, y, weights, Z1, A1, Z2, A2, Z3, A3,
learning_rate):
    m = y.shape[0] # number of samples

    # compute gradients
    dZ3 = (A3 - y) * 2 * derivative_sigmoid(A3)
    dW3 = np.dot(A2.T, dZ3) / m
    db3 = np.sum(dZ3, axis=0, keepdims=True) / m # 沿著 row 求和，並保持
    維度

    dA2 = np.dot(dZ3, weights['W3'].T)
    dZ2 = dA2 * derivative_sigmoid(A2)
    dW2 = np.dot(A1.T, dZ2) / m
    db2 = np.sum(dZ2, axis=0, keepdims=True) / m

    dA1 = np.dot(dZ2, weights['W2'].T)
    dZ1 = dA1 * derivative_sigmoid(A1)
    dW1 = np.dot(X.T, dZ1) / m
    db1 = np.sum(dZ1, axis=0, keepdims=True) / m

    # update weights
    weights['W3'] -= learning_rate * dW3
    weights['b3'] -= learning_rate * db3
    weights['W2'] -= learning_rate * dW2
    weights['b2'] -= learning_rate * db2
    weights['W1'] -= learning_rate * dW1
    weights['b1'] -= learning_rate * db1

```

拿 $Z3, A3, W3$ 為例，推導如下：

首先定義 $dZ3$ 如下

$$dZ3 = \frac{\partial L}{\partial Z3} , dZ3 = 2 * (A3 - y) * \text{derivative_sigmoid}(A3)$$

接著我們需要計算損失函數 (L) 對 權重($W3$) 的梯度 ($dW3 = \frac{\partial L}{\partial W3}$)。

由於 ($Z3 = A2 \cdot W3 + b3$)，我們可以將 (L) 對 ($W3$) 的偏微分寫成：

$$\frac{\partial L}{\partial W3} = \frac{\partial L}{\partial Z3} \cdot \frac{\partial Z3}{\partial W3}$$

然後運用 chain rule，

首先計算 ($\frac{\partial Z_3}{\partial W_3}$) :

$$Z_3 = A_2 \cdot W_3 + b_3$$

$$\frac{\partial Z_3}{\partial W_3} = A_2$$

將上面的結果代入鏈式法則：

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial Z_3} \cdot A_2$$

由於 ($dZ_3 = \frac{\partial L}{\partial Z_3}$)，我們可以得到：

$$dW_3 = dZ_3 \cdot A_2$$

D. Weight Initialization

```
def initialize_weights(input_size, hidden_size1, hidden_size2,
output_size):
    weights = {
        'W1': np.random.randn(input_size, hidden_size1), # shape:
        (input_size, hidden_size1)
        'b1': np.zeros((1, hidden_size1)), # shape: (1, hidden_size1)
        'W2': np.random.randn(hidden_size1, hidden_size2),
        'b2': np.zeros((1, hidden_size2)),
        'W3': np.random.randn(hidden_size2, output_size),
        'b3': np.zeros((1, output_size))
    }
    return weights
```

*weights*大多會初始化在[-3, 3]之間

*bias*皆初始化為0

E. Loss function

```
# cost(loss) function: MSE
def compute_loss(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
```

loss (cost) function 為傳統 L2 norm

F. Train

```

def train(X, y, input_size, hidden_size1, hidden_size2, output_size,
learning_rate, epochs):
    weights = initialize_weights(input_size, hidden_size1,
hidden_size2, output_size)

    for epoch in range(epochs):
        Z1, A1, Z2, A2, Z3, A3 = forward_propagation(X, weights)
        loss = compute_loss(y, A3)
        back_propagation(X, y, weights, Z1, A1, Z2, A2, Z3, A3,
learning_rate)

        if epoch % 5000 == 0:
            print(f'Epoch {epoch}, Loss: {loss}')

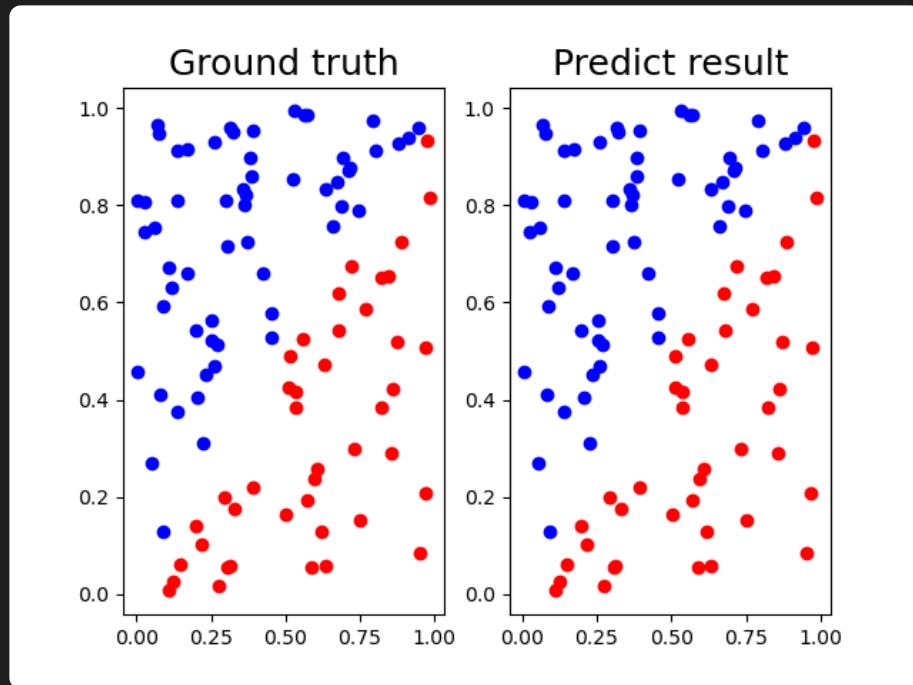
    return weights

```

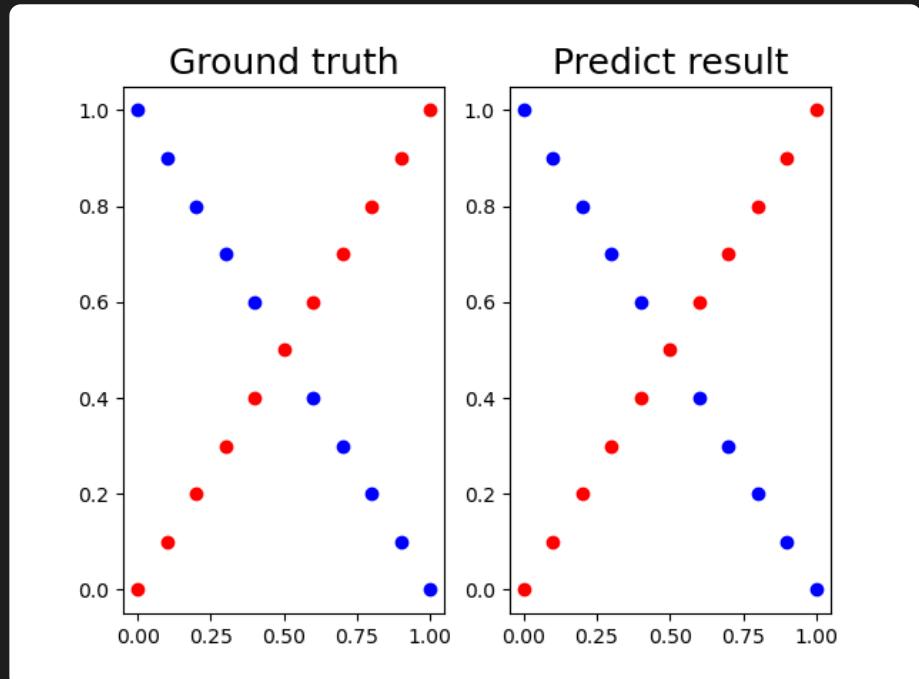
3. Result of testing

A. Screenshot and comparison figure

- Linear data



- XOR data



B. Show the accuracy of your prediction

- Linear data (acc = 100%)

```

Iter1 | Ground truth: 0 | prediction: 0.00000 |
Iter2 | Ground truth: 1 | prediction: 1.00000 |
Iter3 | Ground truth: 0 | prediction: 0.00000 |
Iter4 | Ground truth: 1 | prediction: 1.00000 |
Iter5 | Ground truth: 0 | prediction: 0.00000 |
Iter6 | Ground truth: 0 | prediction: 0.00000 |
Iter7 | Ground truth: 0 | prediction: 0.00000 |
Iter8 | Ground truth: 0 | prediction: 0.00000 |
Iter9 | Ground truth: 1 | prediction: 1.00000 |
Iter10 | Ground truth: 1 | prediction: 1.00000 |
Iter11 | Ground truth: 1 | prediction: 1.00000 |
Iter12 | Ground truth: 1 | prediction: 1.00000 |
Iter13 | Ground truth: 1 | prediction: 1.00000 |
Iter14 | Ground truth: 1 | prediction: 1.00000 |
Iter15 | Ground truth: 1 | prediction: 1.00000 |
Iter16 | Ground truth: 1 | prediction: 1.00000 |
Iter17 | Ground truth: 0 | prediction: 0.00000 |
Iter18 | Ground truth: 1 | prediction: 1.00000 |
Iter19 | Ground truth: 1 | prediction: 1.00000 |

```

Iter20		Ground truth: 1		prediction: 1.00000	
Iter21		Ground truth: 0		prediction: 0.00000	
Iter22		Ground truth: 1		prediction: 0.98980	
Iter23		Ground truth: 1		prediction: 1.00000	
Iter24		Ground truth: 0		prediction: 0.02334	
Iter25		Ground truth: 0		prediction: 0.00000	
Iter26		Ground truth: 0		prediction: 0.00000	
Iter27		Ground truth: 0		prediction: 0.00000	
Iter28		Ground truth: 0		prediction: 0.00000	
Iter29		Ground truth: 1		prediction: 1.00000	
Iter30		Ground truth: 1		prediction: 1.00000	
Iter31		Ground truth: 1		prediction: 1.00000	
Iter32		Ground truth: 0		prediction: 0.00000	
Iter33		Ground truth: 0		prediction: 0.00002	
Iter34		Ground truth: 1		prediction: 1.00000	
Iter35		Ground truth: 0		prediction: 0.00000	
Iter36		Ground truth: 0		prediction: 0.00000	
Iter37		Ground truth: 0		prediction: 0.00000	
Iter38		Ground truth: 0		prediction: 0.00000	
Iter39		Ground truth: 1		prediction: 0.99978	
Iter40		Ground truth: 0		prediction: 0.00000	
Iter41		Ground truth: 1		prediction: 1.00000	
Iter42		Ground truth: 1		prediction: 1.00000	
Iter43		Ground truth: 0		prediction: 0.00000	
Iter44		Ground truth: 0		prediction: 0.00000	
Iter45		Ground truth: 0		prediction: 0.00000	
Iter46		Ground truth: 1		prediction: 1.00000	
Iter47		Ground truth: 0		prediction: 0.00003	
Iter48		Ground truth: 0		prediction: 0.00000	
Iter49		Ground truth: 0		prediction: 0.00000	
Iter50		Ground truth: 0		prediction: 0.00006	
Iter51		Ground truth: 1		prediction: 1.00000	
Iter52		Ground truth: 0		prediction: 0.00000	
Iter53		Ground truth: 0		prediction: 0.00000	
Iter54		Ground truth: 1		prediction: 1.00000	
Iter55		Ground truth: 1		prediction: 1.00000	
Iter56		Ground truth: 1		prediction: 1.00000	

Iter57		Ground truth: 1		prediction: 1.00000	
Iter58		Ground truth: 1		prediction: 1.00000	
Iter59		Ground truth: 0		prediction: 0.00000	
Iter60		Ground truth: 1		prediction: 1.00000	
Iter61		Ground truth: 0		prediction: 0.00000	
Iter62		Ground truth: 0		prediction: 0.00000	
Iter63		Ground truth: 1		prediction: 1.00000	
Iter64		Ground truth: 0		prediction: 0.00000	
Iter65		Ground truth: 0		prediction: 0.00000	
Iter66		Ground truth: 1		prediction: 1.00000	
Iter67		Ground truth: 0		prediction: 0.00000	
Iter68		Ground truth: 0		prediction: 0.00000	
Iter69		Ground truth: 1		prediction: 0.99995	
Iter70		Ground truth: 1		prediction: 1.00000	
Iter71		Ground truth: 1		prediction: 1.00000	
Iter72		Ground truth: 1		prediction: 1.00000	
Iter73		Ground truth: 0		prediction: 0.00000	
Iter74		Ground truth: 0		prediction: 0.00000	
Iter75		Ground truth: 1		prediction: 1.00000	
Iter76		Ground truth: 1		prediction: 1.00000	
Iter77		Ground truth: 1		prediction: 1.00000	
Iter78		Ground truth: 1		prediction: 1.00000	
Iter79		Ground truth: 0		prediction: 0.00000	
Iter80		Ground truth: 1		prediction: 1.00000	
Iter81		Ground truth: 0		prediction: 0.00000	
Iter82		Ground truth: 1		prediction: 1.00000	
Iter83		Ground truth: 1		prediction: 0.98803	
Iter84		Ground truth: 1		prediction: 1.00000	
Iter85		Ground truth: 0		prediction: 0.00000	
Iter86		Ground truth: 1		prediction: 1.00000	
Iter87		Ground truth: 1		prediction: 1.00000	
Iter88		Ground truth: 1		prediction: 1.00000	
Iter89		Ground truth: 0		prediction: 0.00000	
Iter90		Ground truth: 0		prediction: 0.00000	
Iter91		Ground truth: 1		prediction: 1.00000	
Iter92		Ground truth: 0		prediction: 0.00000	
Iter93		Ground truth: 0		prediction: 0.00000	

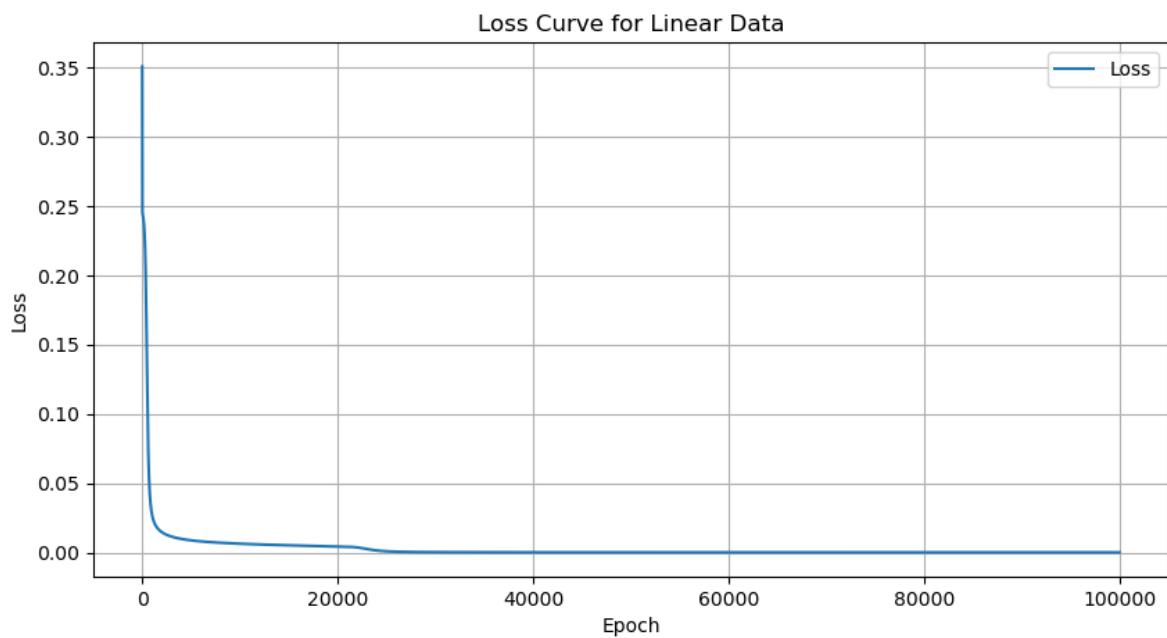
```
Iter94 | Ground truth: 1 | prediction: 1.00000 |
Iter95 | Ground truth: 0 | prediction: 0.00000 |
Iter96 | Ground truth: 1 | prediction: 1.00000 |
Iter97 | Ground truth: 1 | prediction: 1.00000 |
Iter98 | Ground truth: 0 | prediction: 0.00001 |
Iter99 | Ground truth: 1 | prediction: 1.00000 |
Iter100 | Ground truth: 1 | prediction: 1.00000 |
loss=0.0000079244 accuracy=100.00%
```

- XOR data (acc = 100%)

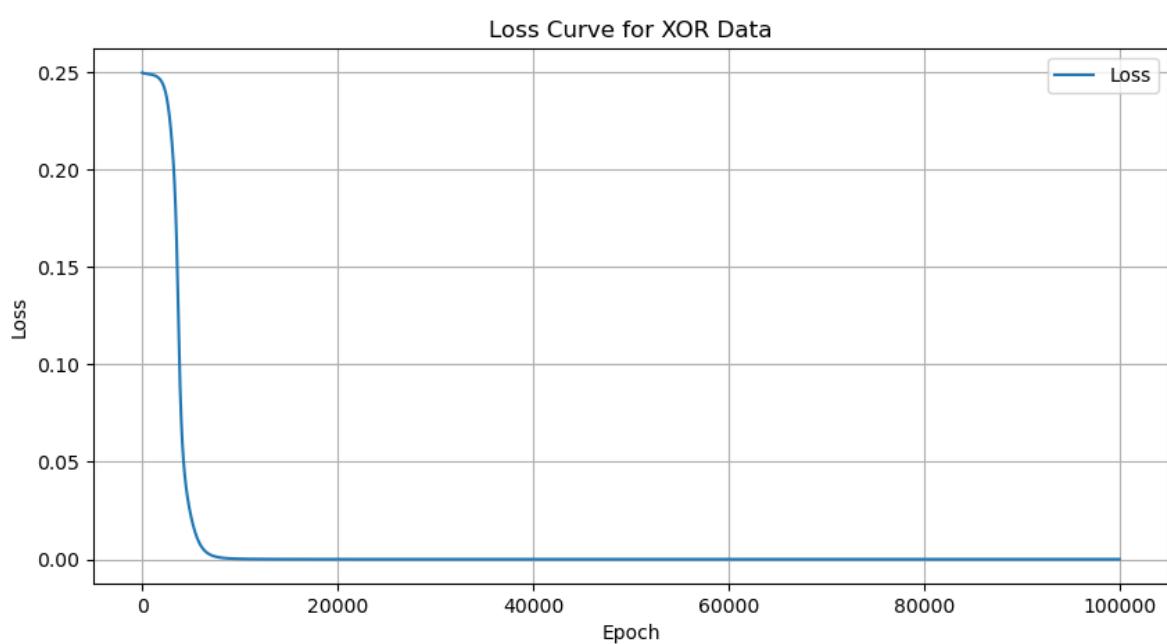
```
Iter1 | Ground truth: 0 | prediction: 0.00021 |
Iter2 | Ground truth: 1 | prediction: 0.99973 |
Iter3 | Ground truth: 0 | prediction: 0.00023 |
Iter4 | Ground truth: 1 | prediction: 0.99973 |
Iter5 | Ground truth: 0 | prediction: 0.00027 |
Iter6 | Ground truth: 1 | prediction: 0.99973 |
Iter7 | Ground truth: 0 | prediction: 0.00031 |
Iter8 | Ground truth: 1 | prediction: 0.99971 |
Iter9 | Ground truth: 0 | prediction: 0.00035 |
Iter10 | Ground truth: 1 | prediction: 0.99902 |
Iter11 | Ground truth: 0 | prediction: 0.00037 |
Iter12 | Ground truth: 0 | prediction: 0.00036 |
Iter13 | Ground truth: 1 | prediction: 0.99909 |
Iter14 | Ground truth: 0 | prediction: 0.00034 |
Iter15 | Ground truth: 1 | prediction: 0.99992 |
Iter16 | Ground truth: 0 | prediction: 0.00031 |
Iter17 | Ground truth: 1 | prediction: 0.99993 |
Iter18 | Ground truth: 0 | prediction: 0.00029 |
Iter19 | Ground truth: 1 | prediction: 0.99993 |
Iter20 | Ground truth: 0 | prediction: 0.00026 |
Iter21 | Ground truth: 1 | prediction: 0.99994 |
loss=0.0000001487 accuracy=100.00%
```

C. Learning curve (loss, epoch curve)

- Linear data



- XOR data



D. Anything you want to present

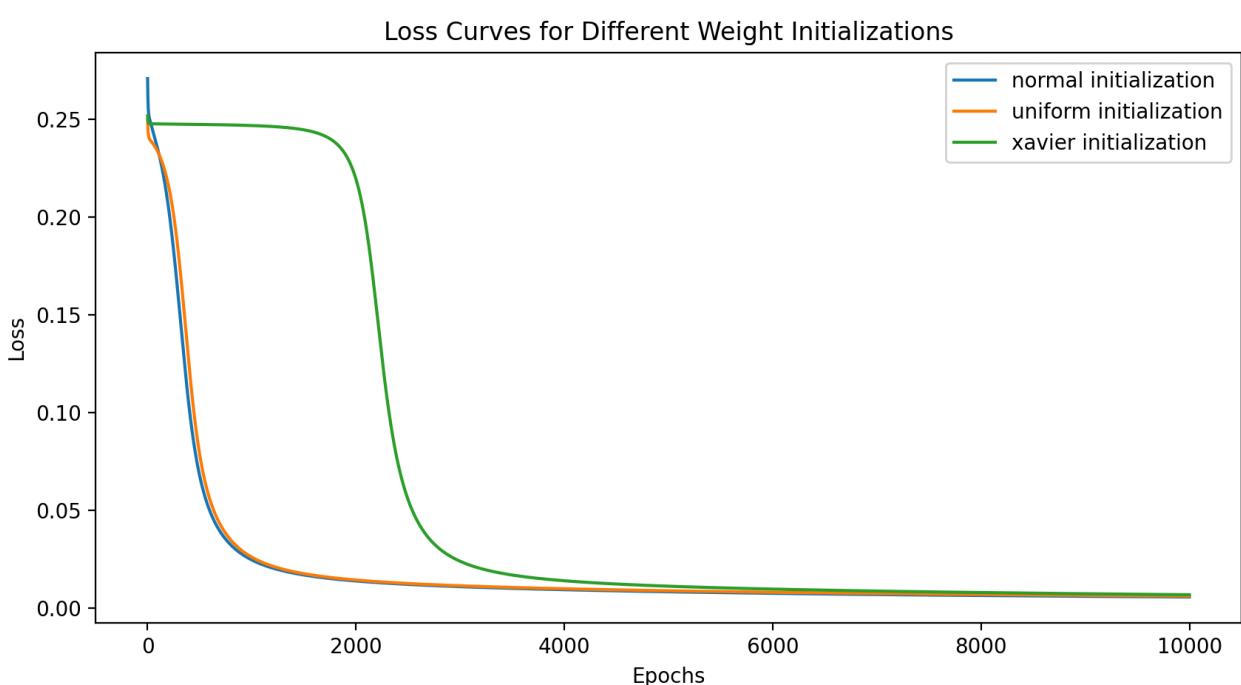
對於不同的 weight 初始化方法，我做了以下實驗比較：

- Normal (原始的方法)
- Uniform
- Xavier

```

if method == 'normal':
    weight_initializer = lambda x, y: np.random.randn(x, y)
elif method == 'uniform':
    weight_initializer = lambda x, y: np.random.uniform(-1.0, 1.0,
(x, y))
elif method == 'xavier':
    weight_initializer = lambda x, y: np.random.randn(x, y) *
np.sqrt(1 / x)

```



對於 weight initialization 的收斂速度比較為

Normal >= Uniform >> Xavier

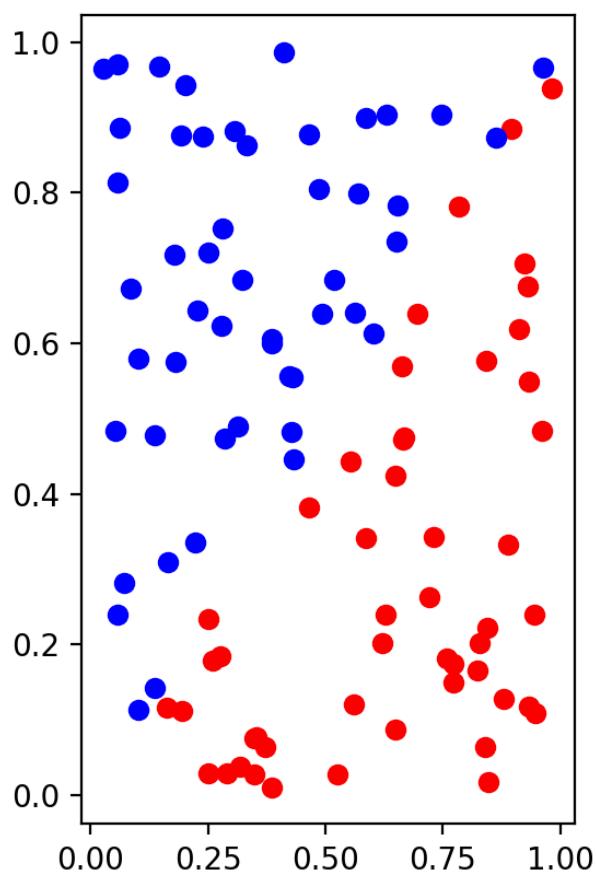
4. Discussion

A. Try different learning rates

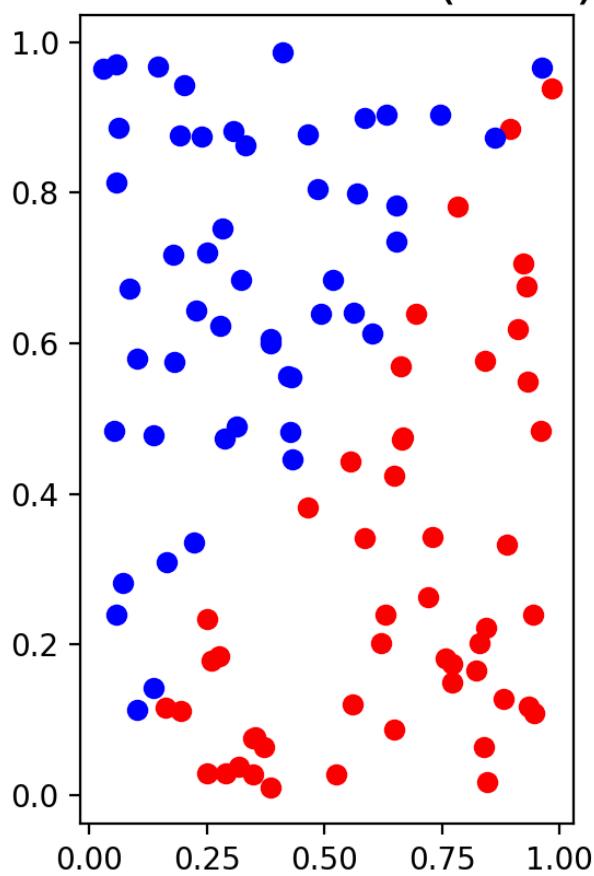
For linear data

- lr = 1, acc = 100%

Ground truth

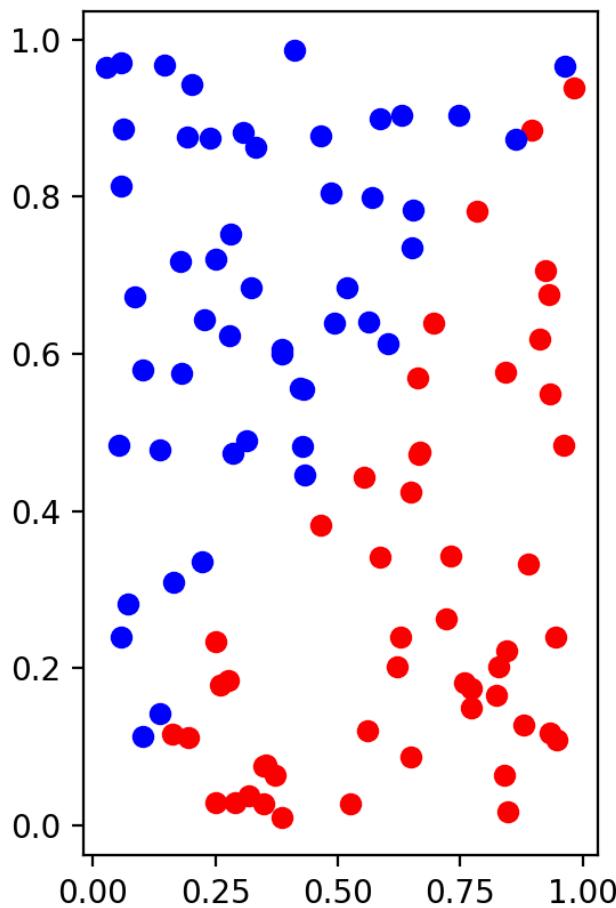


Predict result ($\text{lr}=1$)

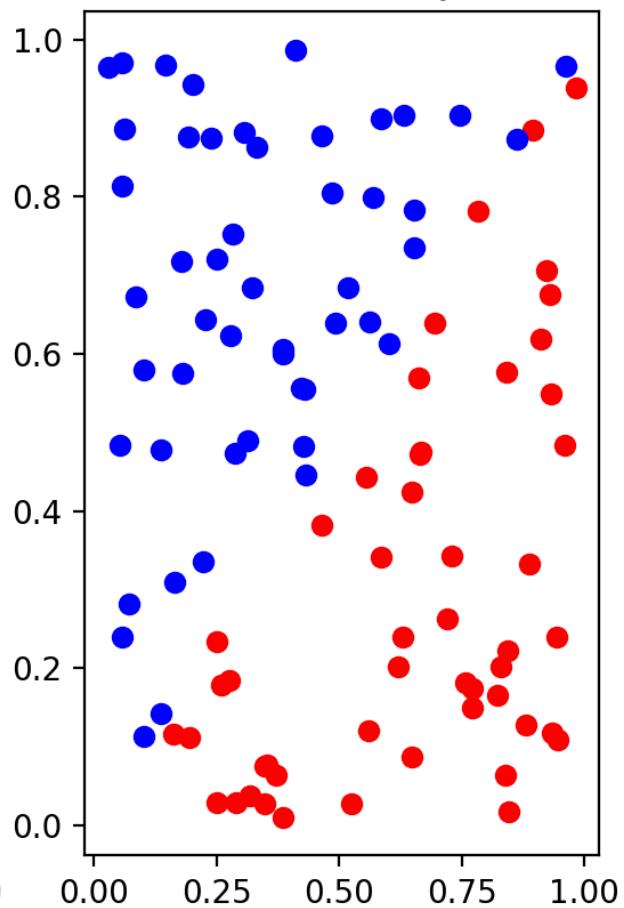


- $\text{lr} = 0.1, \text{acc} = 100\%$

Ground truth

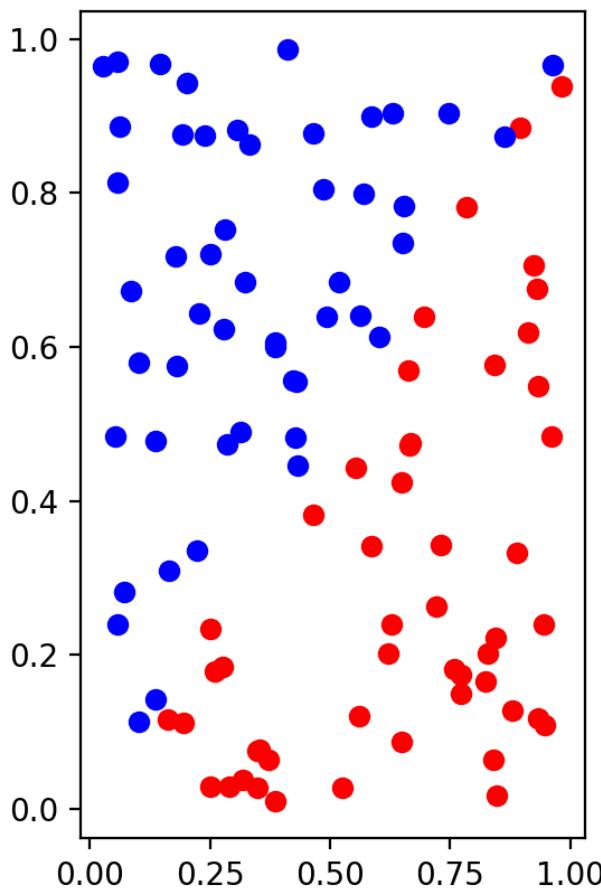


Predict result ($\text{lr}=0.1$)

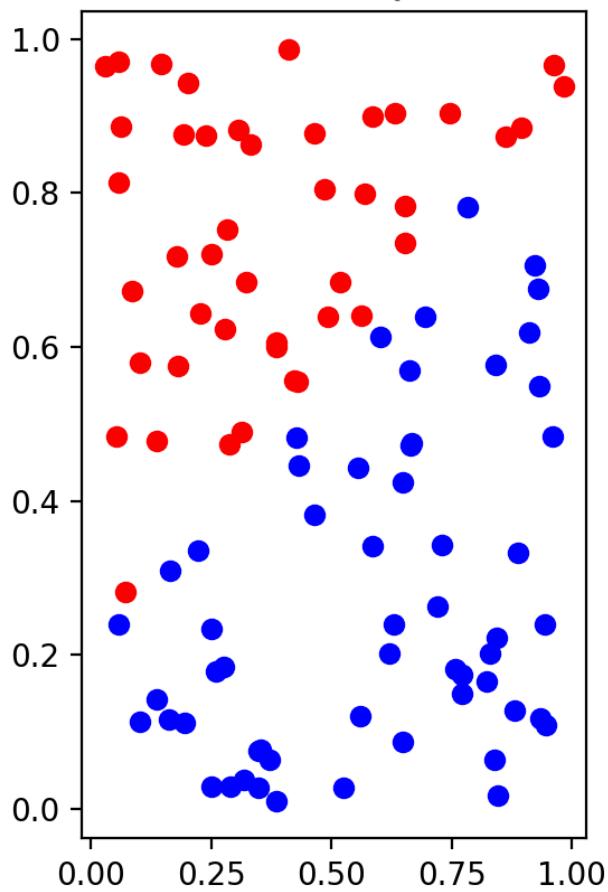


- $\text{lr} = 0.0001, \text{acc} = 52\%$

Ground truth

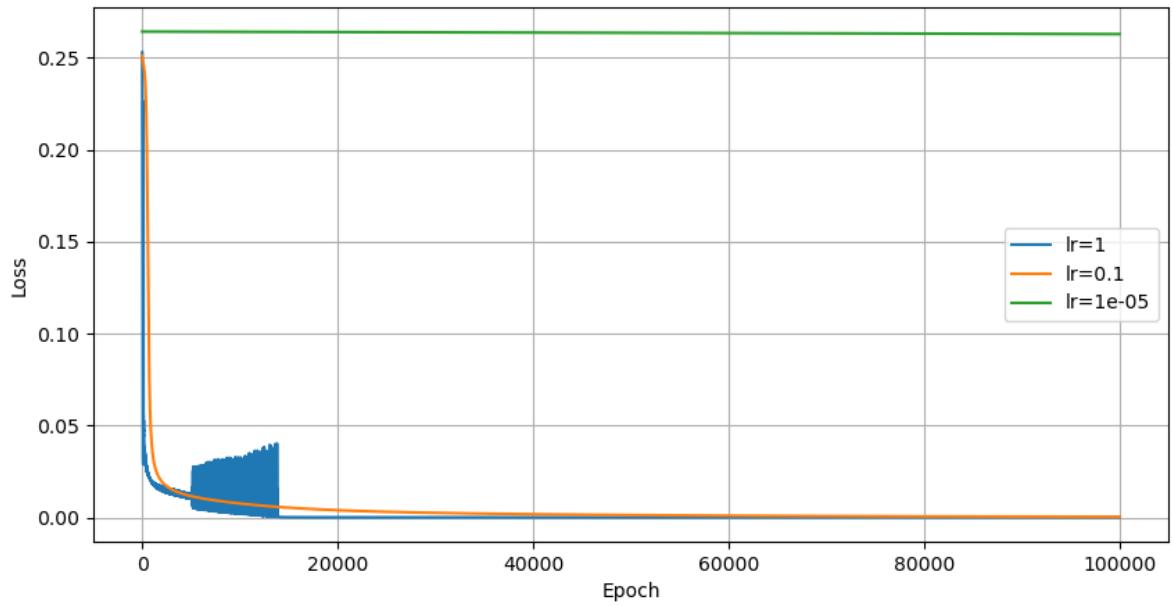


Predict result ($\text{lr}=1\text{e}-05$)



- Comparison figure

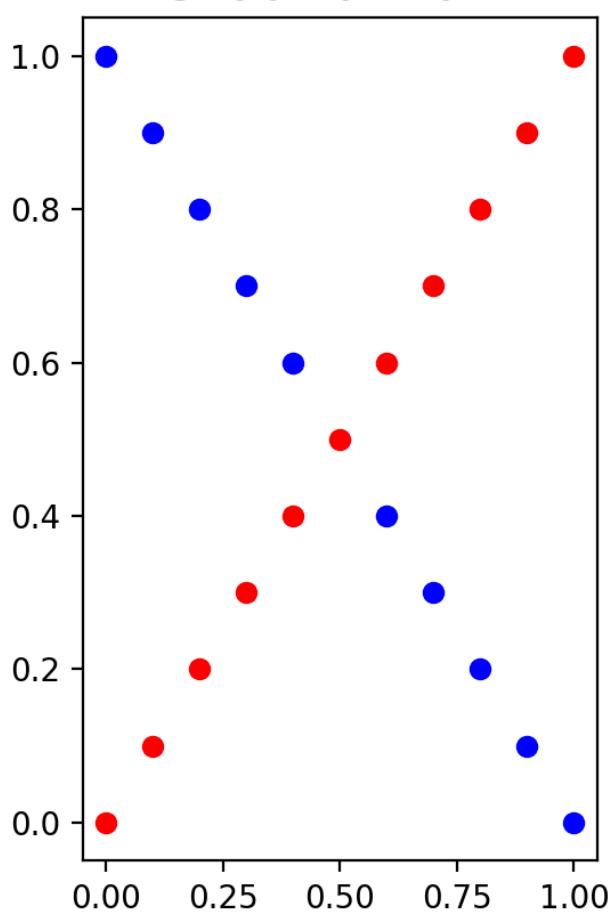
Loss Curves for Linear Data



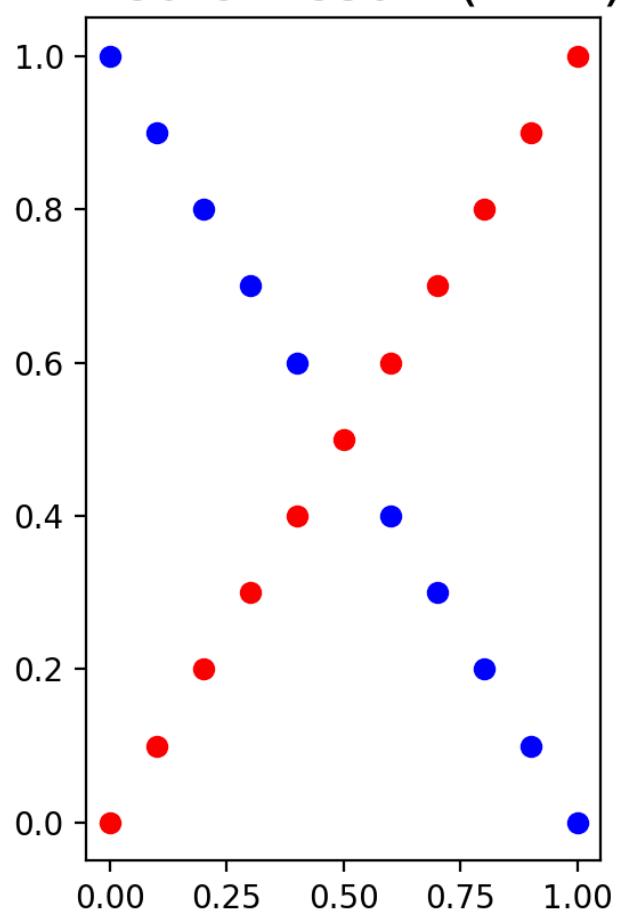
For XOR data

- $\text{lr} = 1$, acc = 100%

Ground truth

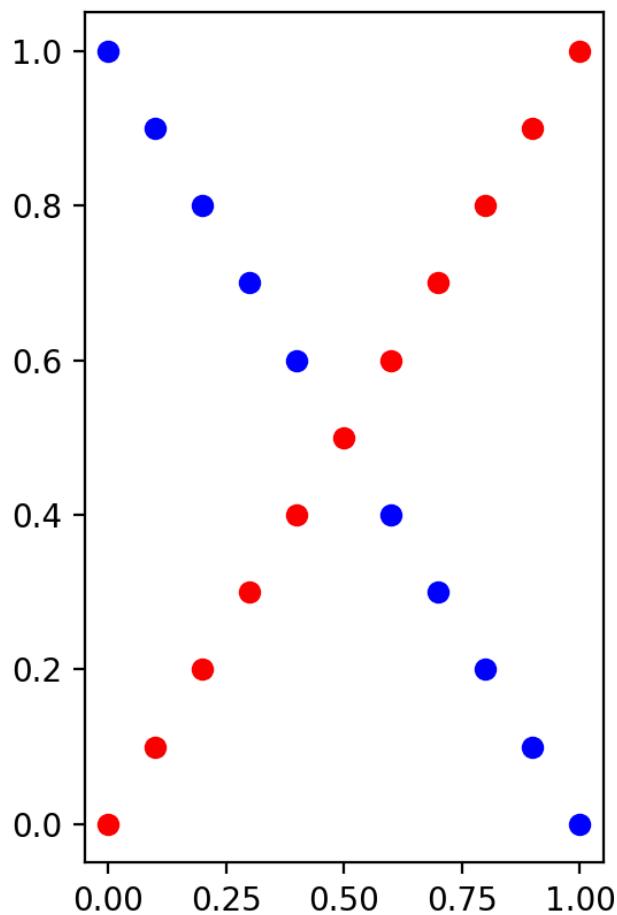


Predict result ($\text{lr}=1$)

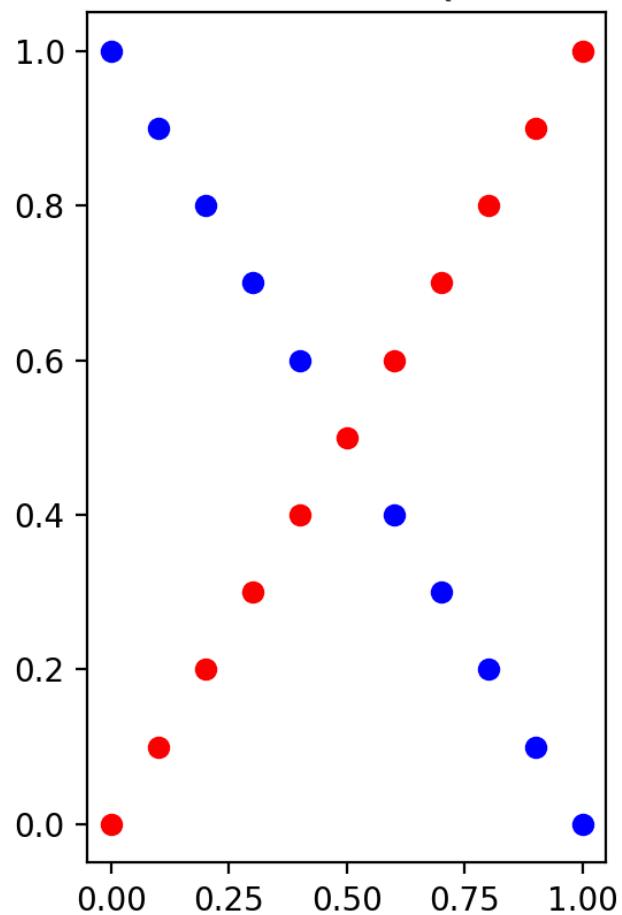


- $\text{lr} = 0.1, \text{acc} = 100\%$

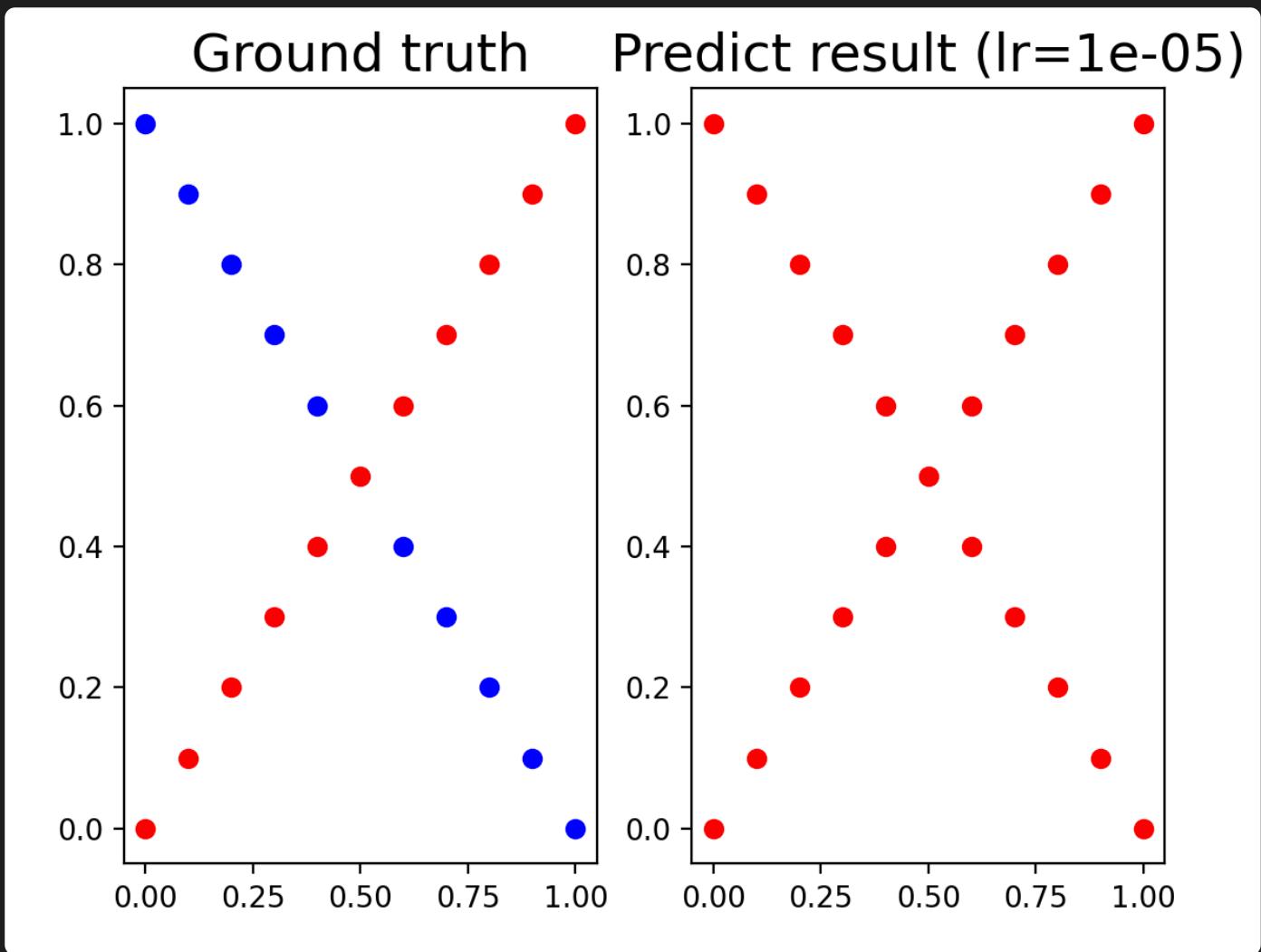
Ground truth



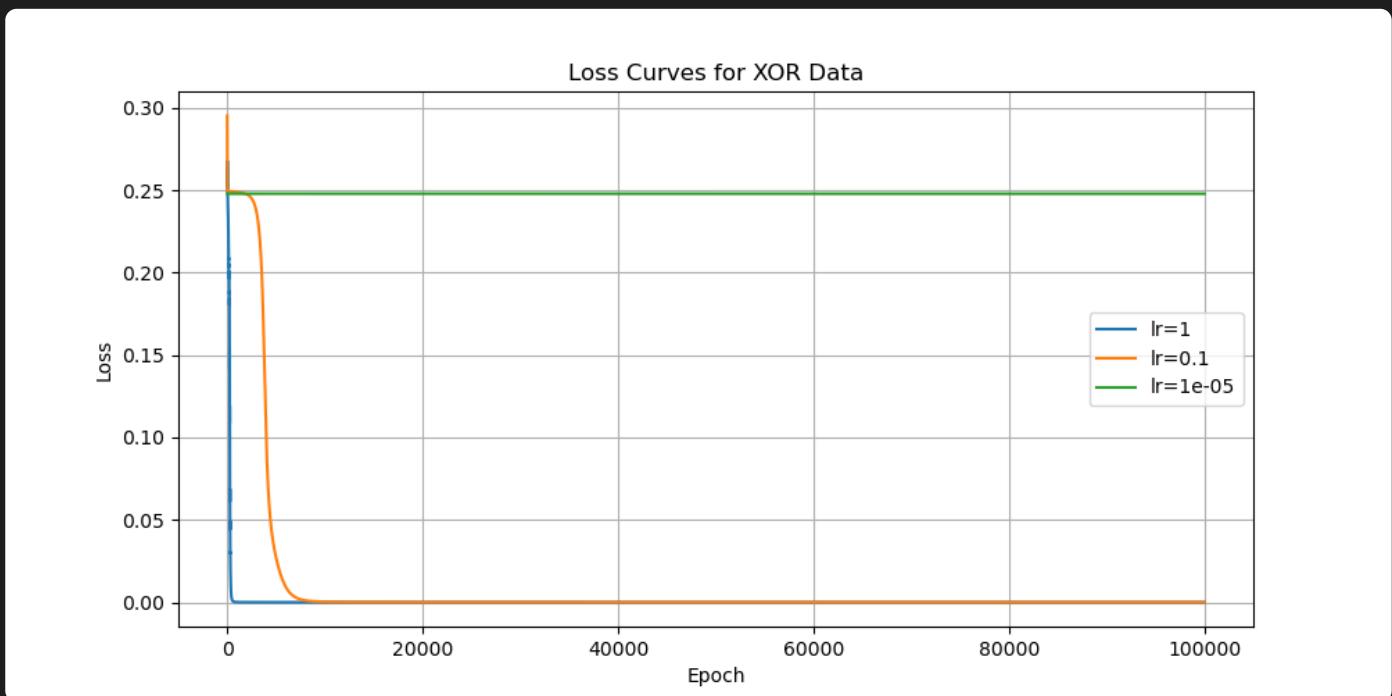
Predict result ($lr=0.1$)



- $lr = 0.0001$, acc = 52.38%



■ Comparison figure

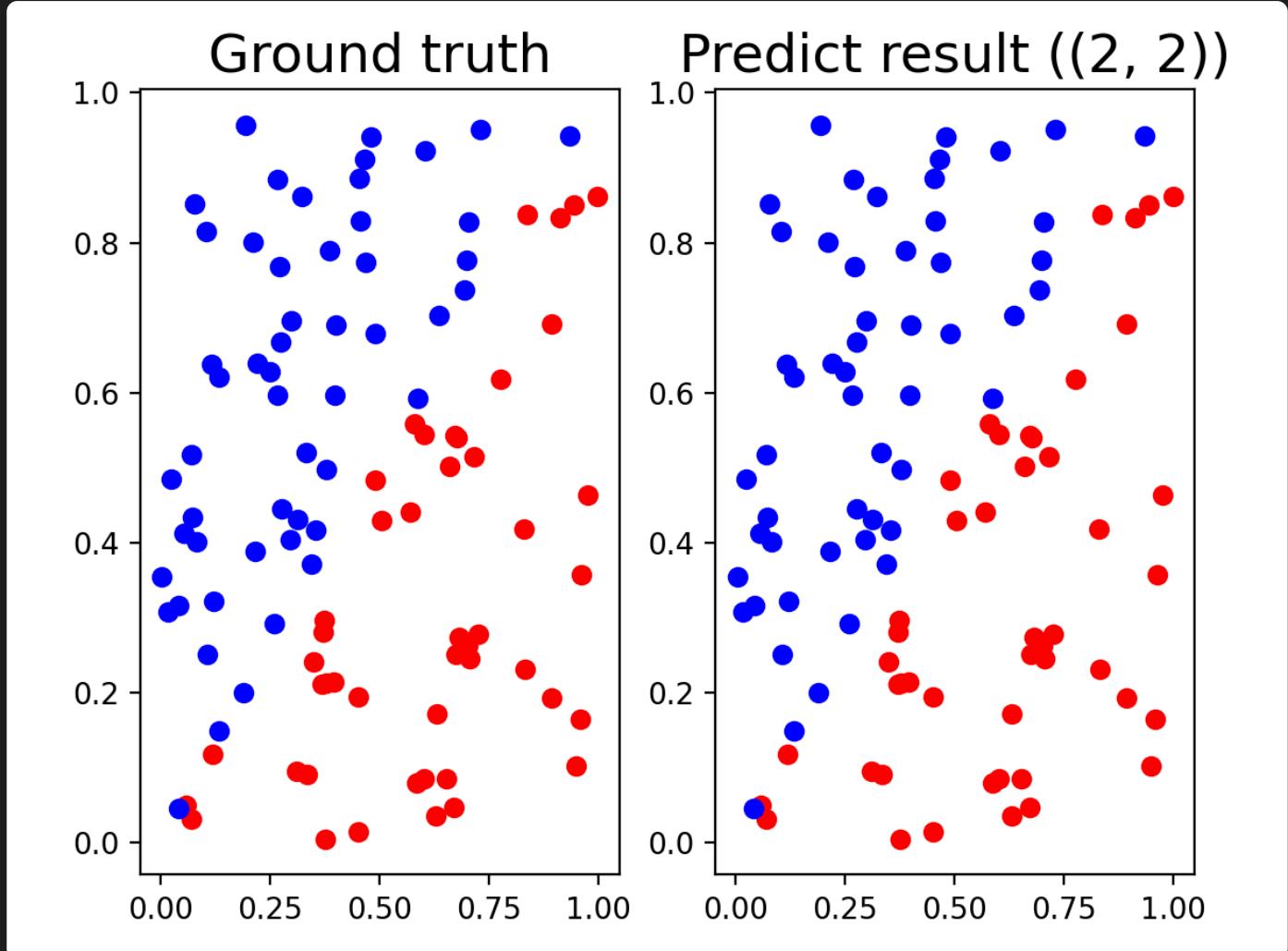


Learning rate 設在 0.00001 太小了，原本以為是 epochs 不夠多無法讓他收斂，但觀察上面的比較圖後，看起來模型早在 10000 epoch 就沒有辦法降低 loss 了。

B. Try different numbers of hidden units

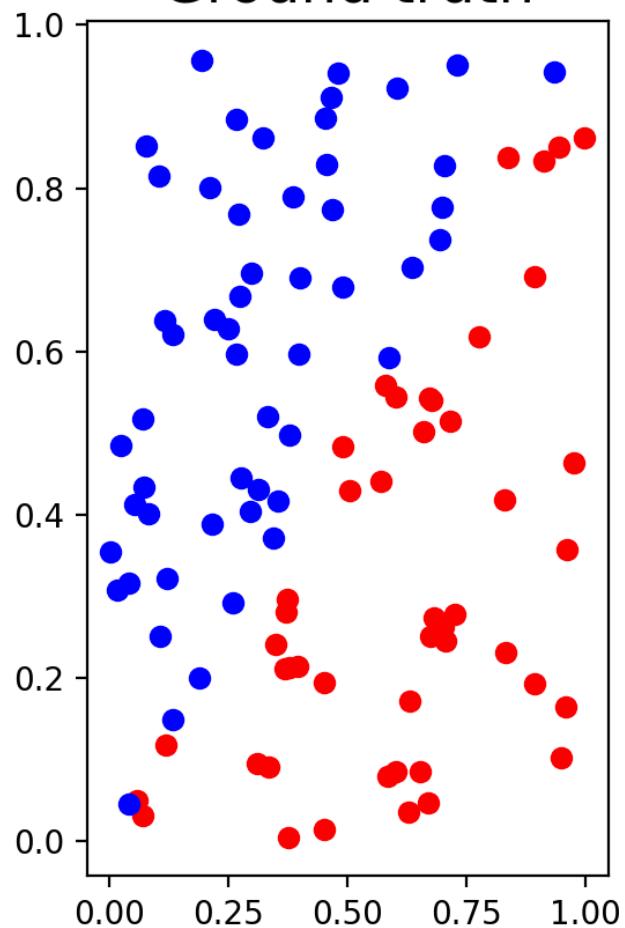
For linear data

- hidden units = (2, 2) , acc = 100%

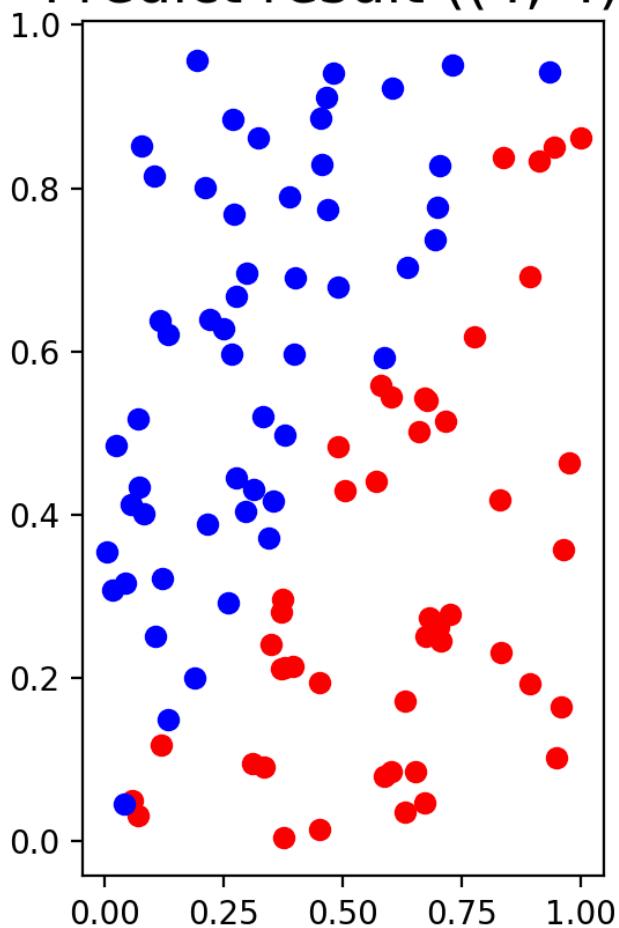


- hidden units = (4, 4) , acc = 100%

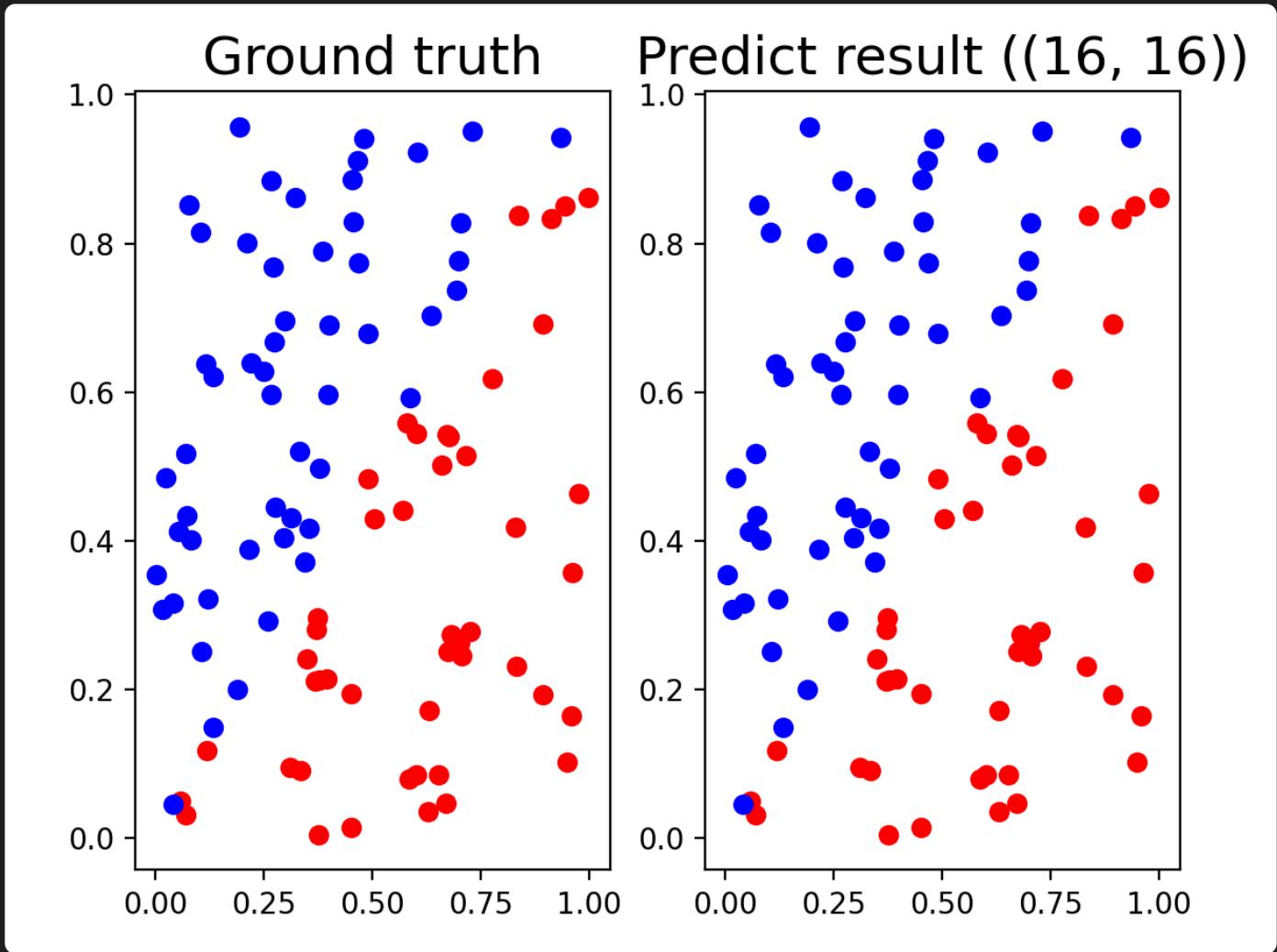
Ground truth



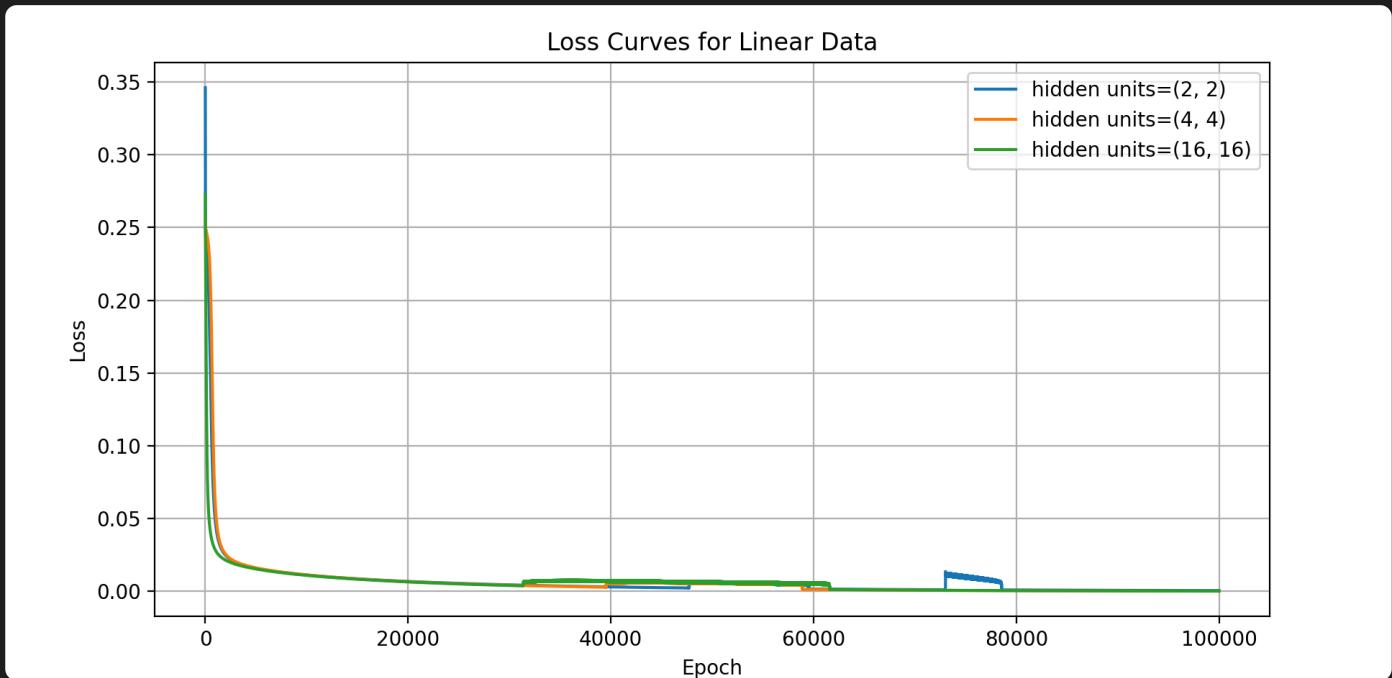
Predict result ((4, 4))



- hidden units = (16, 16) , acc = 100%



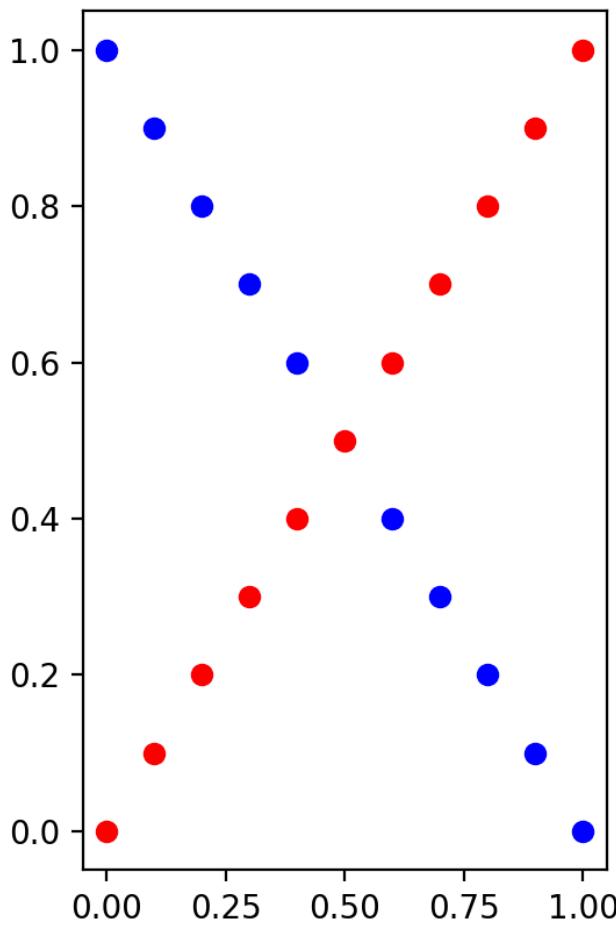
- Comparison figure



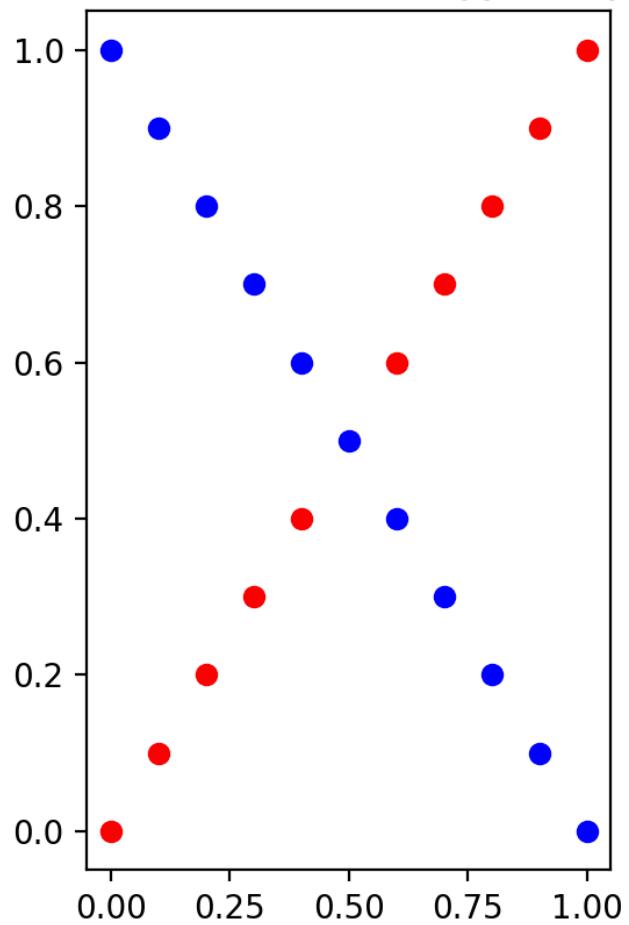
For XOR data

- hidden units = (2, 2), acc = 100%

Ground truth

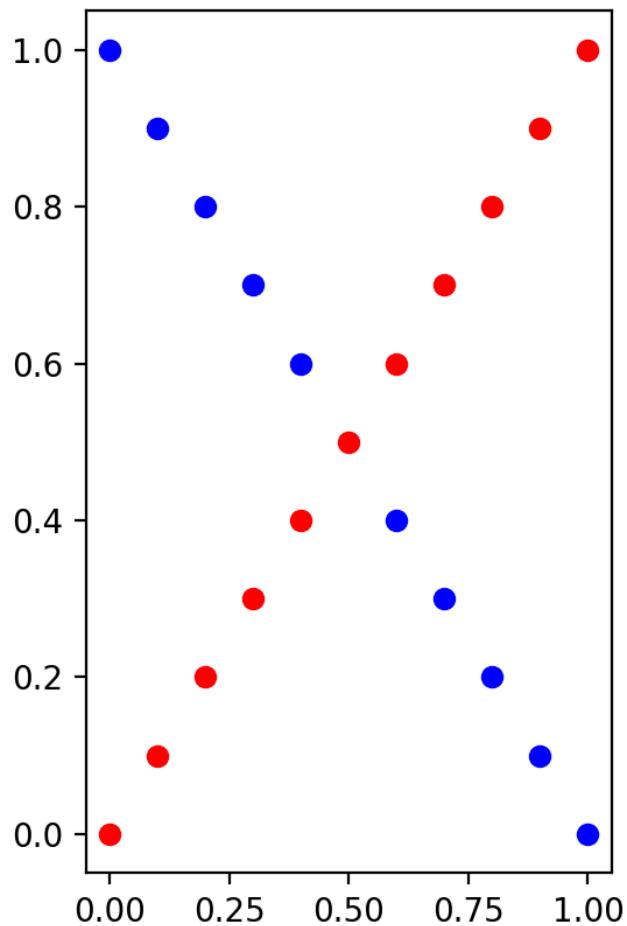


Predict result ((2, 2))

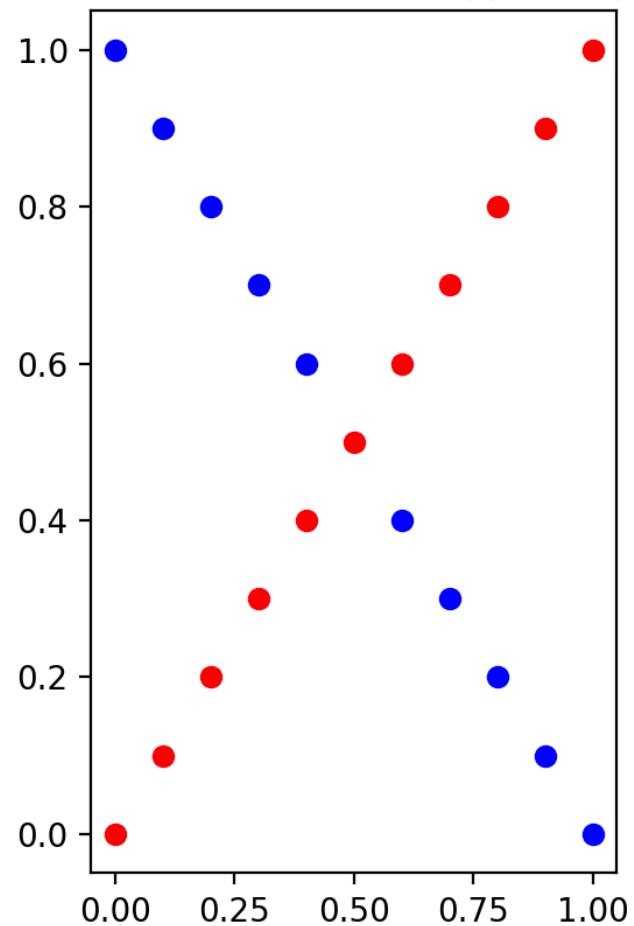


- hidden units = (4, 4), acc = 100%

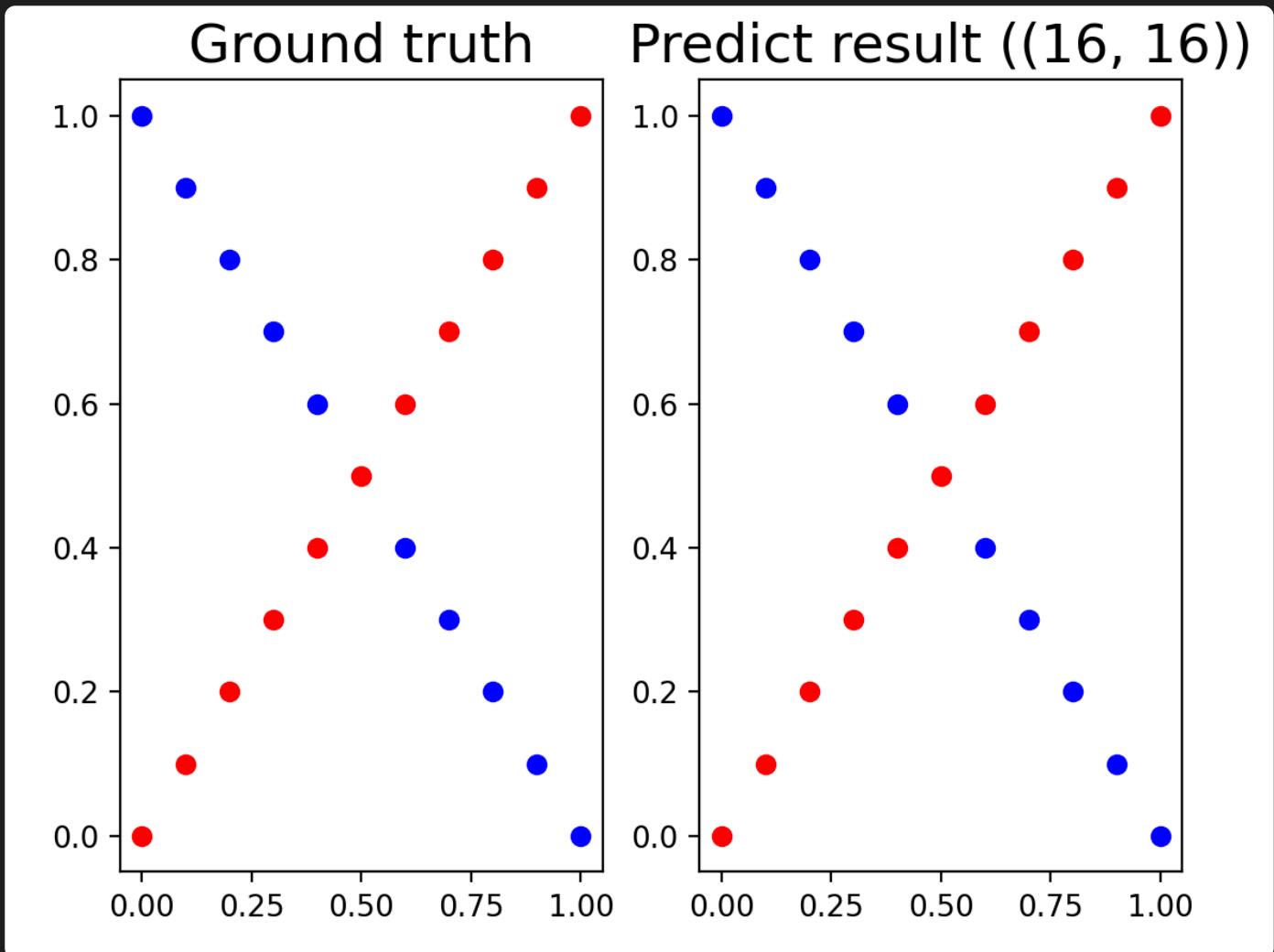
Ground truth



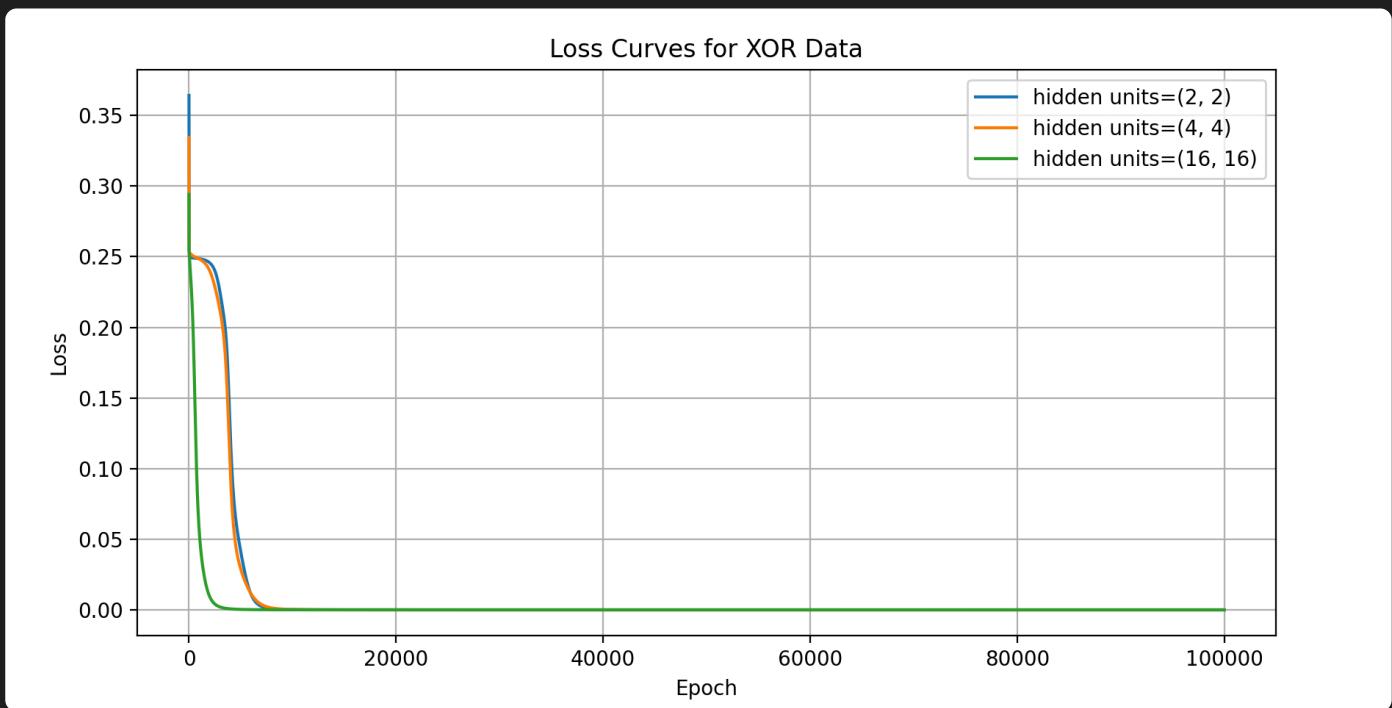
Predict result ((4, 4))



- hidden units = (16, 16), acc = 100%



■ Comparison figure



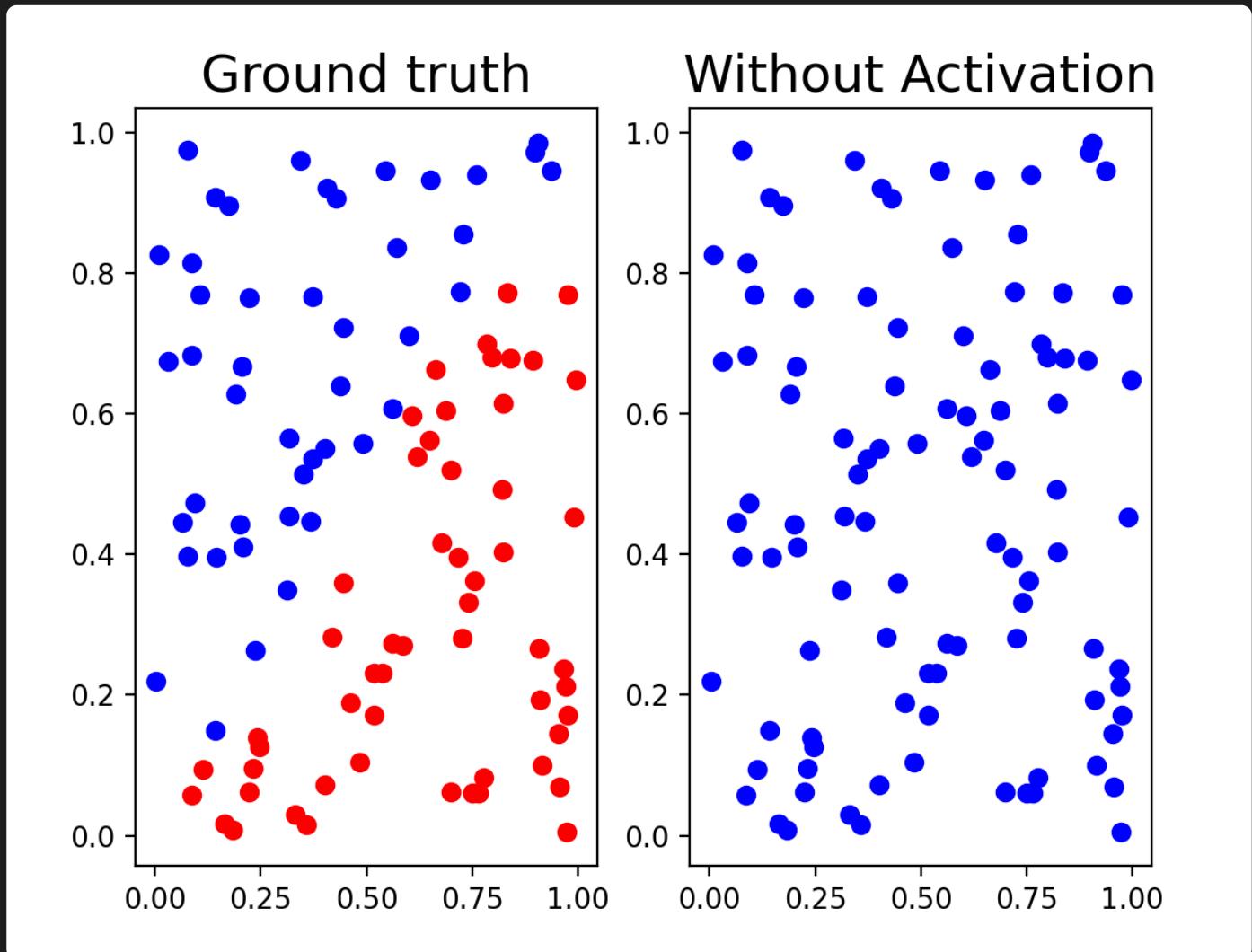
觀察下來，好像 hidden units 設在 (16, 16)，收斂速度會更快一些。

C. Try without activation functions

Without activation function (acc 還是能反映大致訓練狀況，但僅參考...)

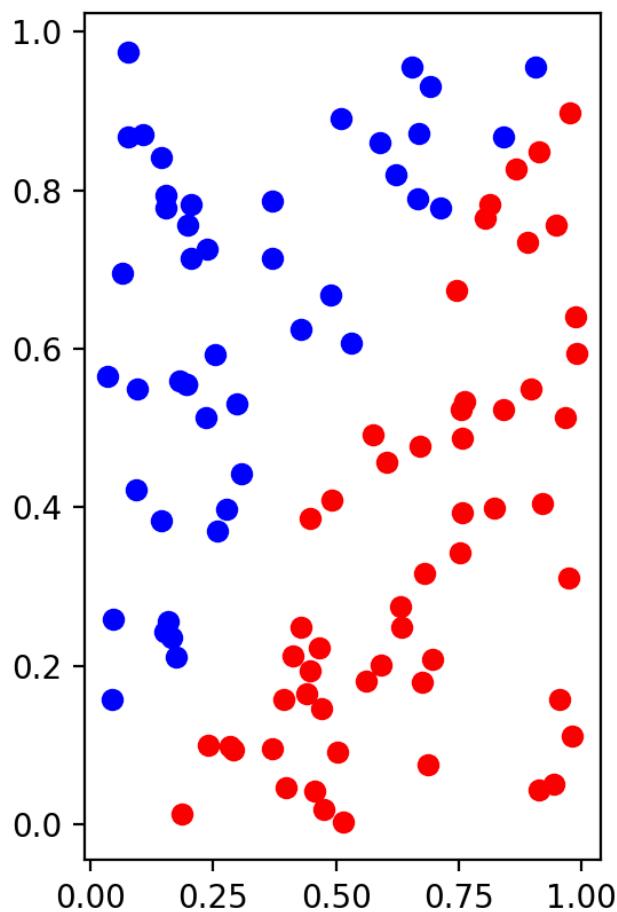
- Linear data

少數情況會完全 train 不起來， $acc = 55\%$

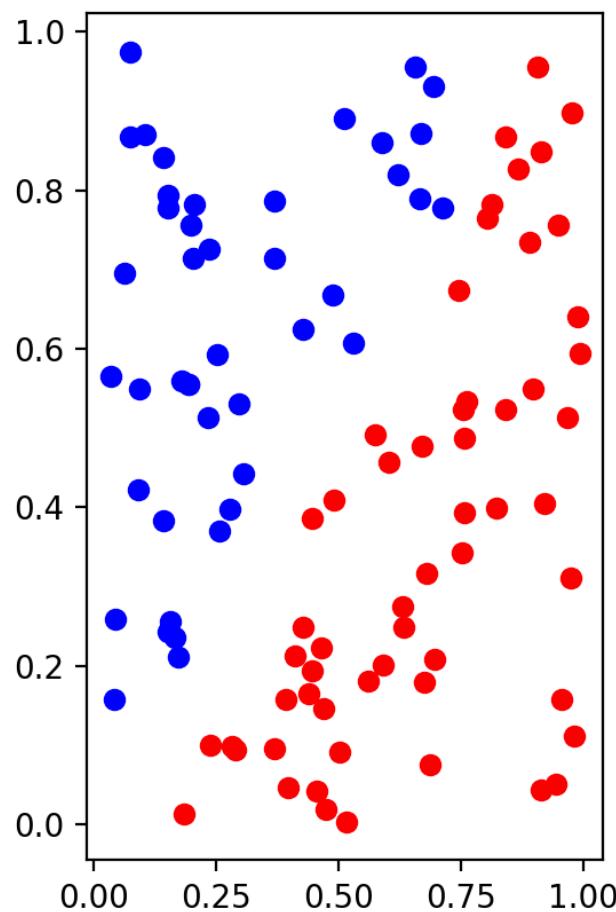


通常是都 train 得起來， $acc = 98\%$ ，但還是找不到最佳解

Ground truth

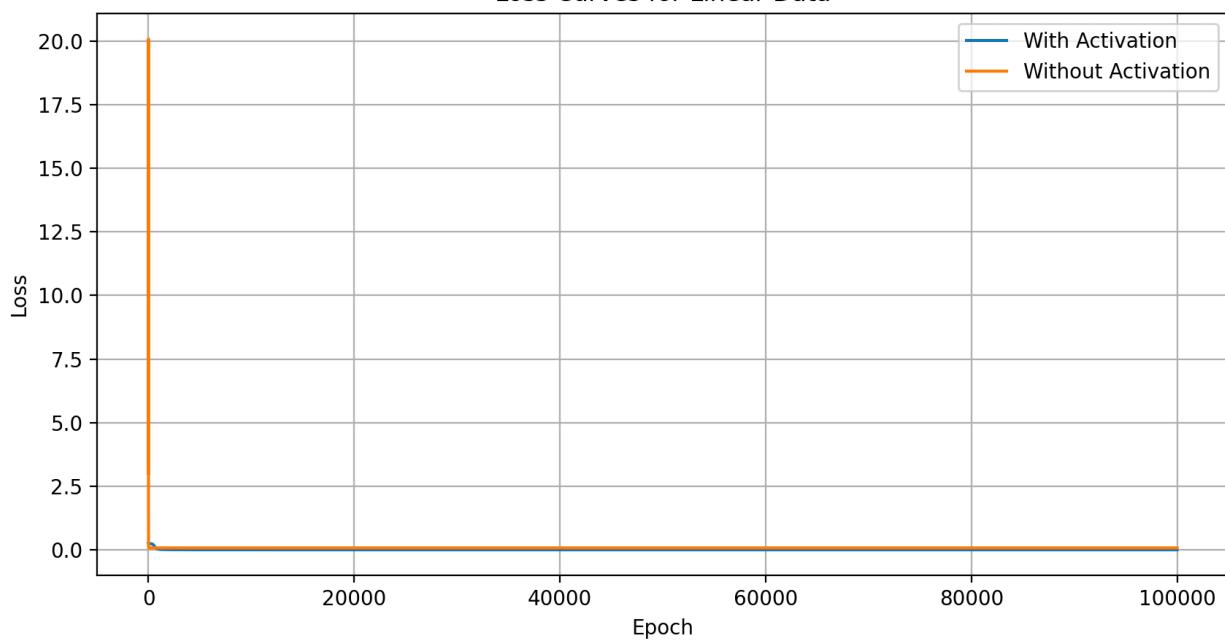


Without Activation



Comparison figure

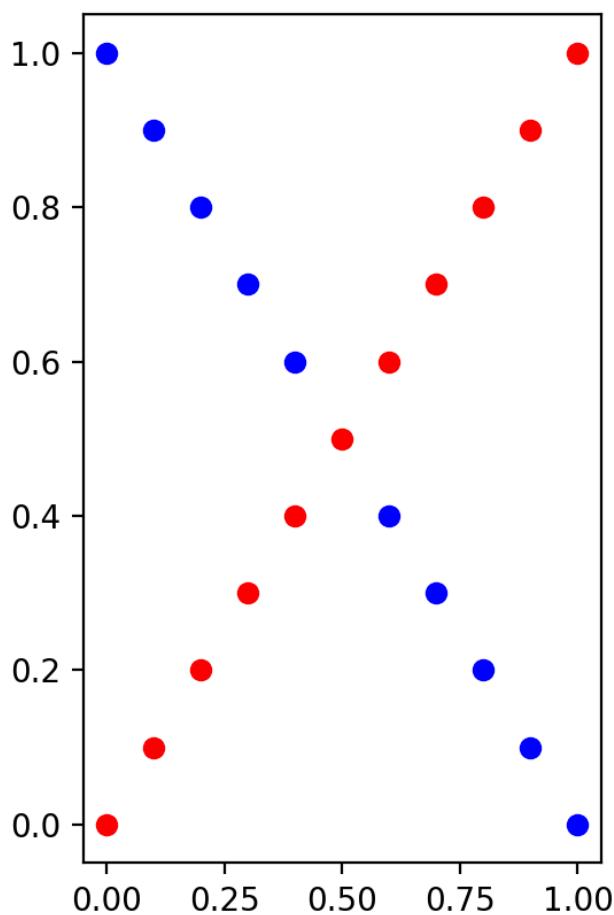
Loss Curves for Linear Data



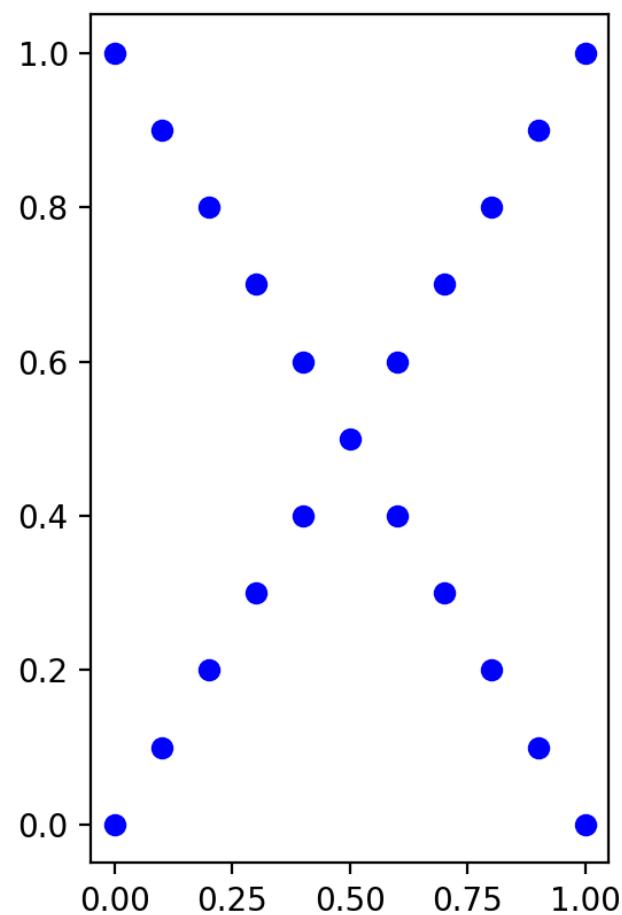
▪ XOR data

完全無法 train , acc = 52 %

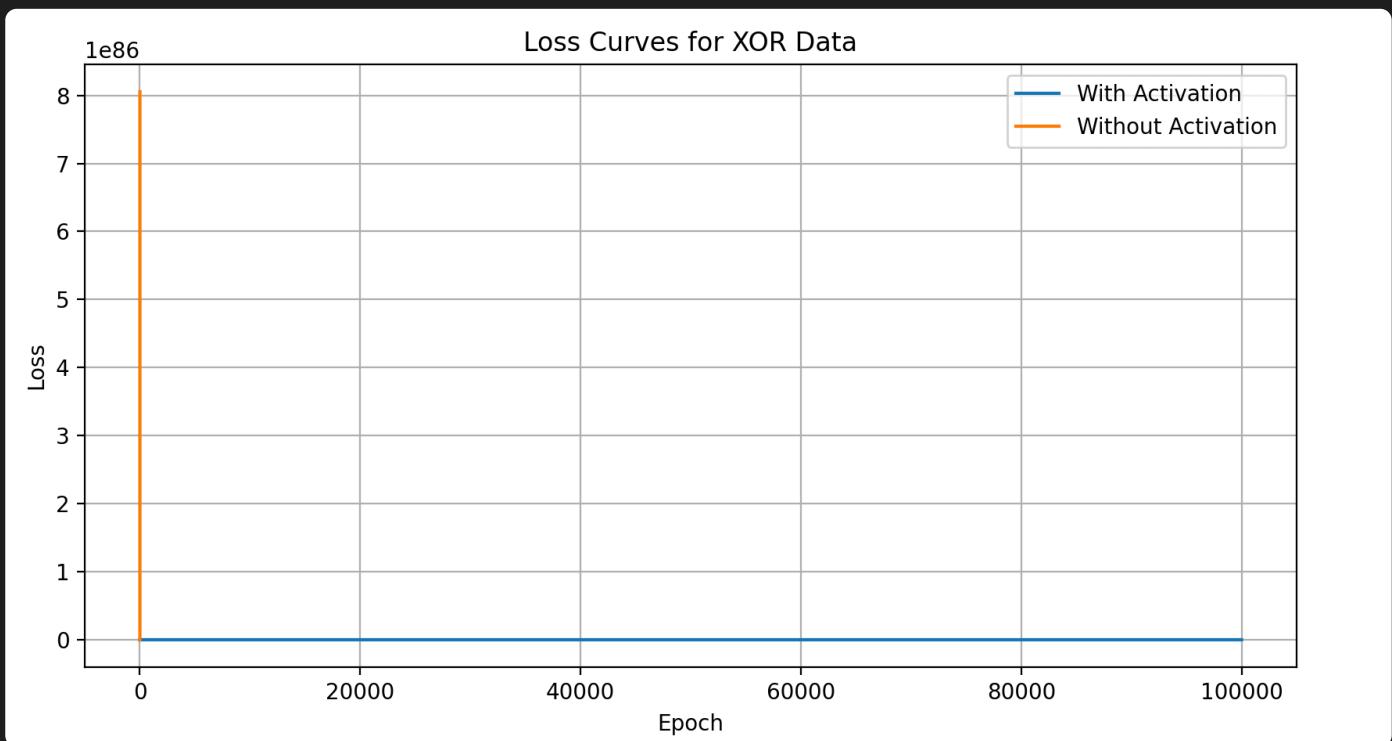
Ground truth



Without Activation



Comparison figure



loss 會直接突破天際...

D. Anything you want to share

嘗試了一下不同的 loss function (MSE -> Cross entropy)

$$L(y, \hat{y}) = - (y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

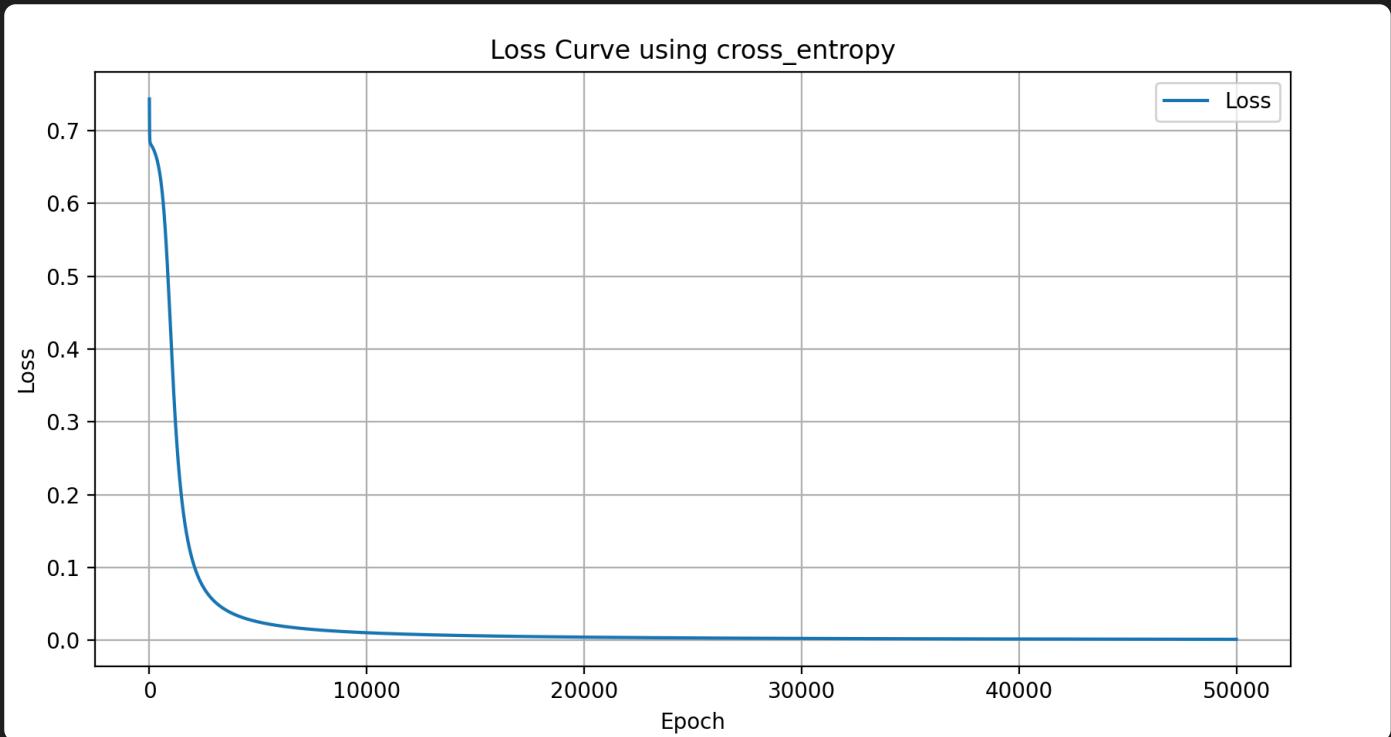
```
def cross_entropy_loss(y_true, y_pred):
    epsilon = 1e-12
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
```

- Back propagation 的開頭變這樣

```
if loss_type == 'cross_entropy':
    dA3 = -(y / A3) + (1 - y) / (1 - A3)
    dZ3 = dA3 * derivative_sigmoid(A3)
else: # mse
    dZ3 = (A3 - y) * 2 * derivative_sigmoid(A3)
```

依然可以正常收斂

```
Epoch 0, Loss (cross_entropy): 0.7435800152511739
Epoch 5000, Loss (cross_entropy): 0.02526995709076904
Epoch 10000, Loss (cross_entropy): 0.01036484196405397
Epoch 15000, Loss (cross_entropy): 0.006214804735750703
Epoch 20000, Loss (cross_entropy): 0.004303550005177803
Epoch 25000, Loss (cross_entropy): 0.003226119856241756
Epoch 30000, Loss (cross_entropy): 0.0025451117385857764
Epoch 35000, Loss (cross_entropy): 0.0020810679549495576
Epoch 40000, Loss (cross_entropy): 0.0017474324652610776
Epoch 45000, Loss (cross_entropy): 0.0014976622444989268
Final accuracy using cross_entropy: 100.00%
```



5. Extra

A. Implement different optimizers

Momentum

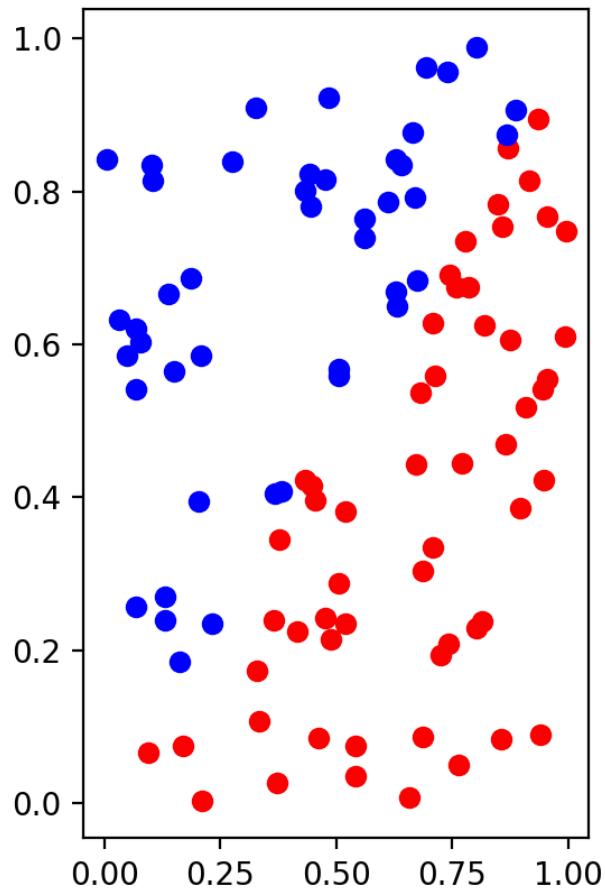
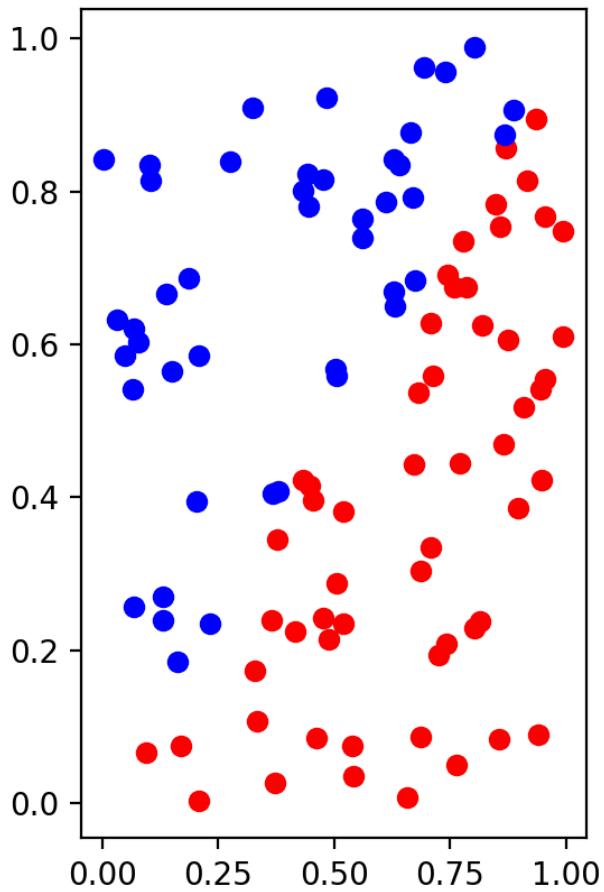
```
def momentum_update(weights, grads, velocity, learning_rate,
momentum=0.9):
    for key in weights.keys():
        velocity[key] = momentum * velocity[key] - learning_rate *
grads[key]
        weights[key] += velocity[key]
    return weights, velocity
```

`velocity[key]` 表示上一次更新的速度。

(如果更新方向和上次相反，這次更新速度會變慢，反之則變快。)

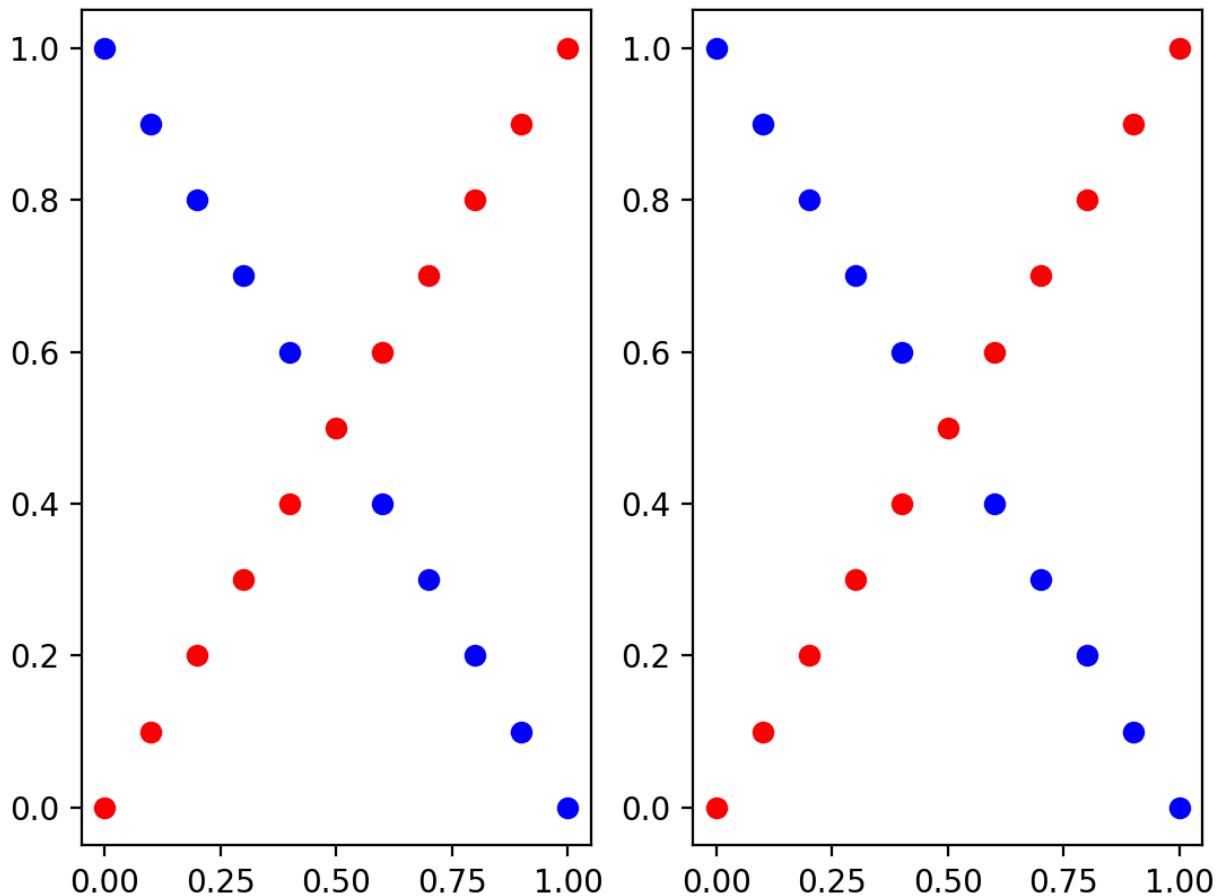
- Linear data (acc = 100%)

Ground truth Predict result (MOMENTUM)



- XOR data (acc = 100%)

Ground truth Predict result (MOMENTUM)



Adam

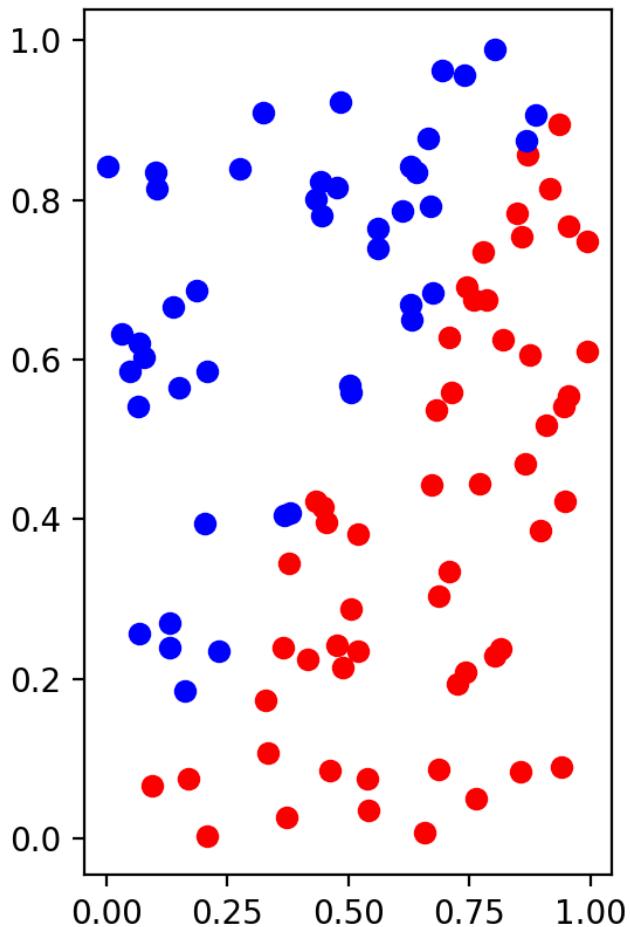
```
def adam_update(weights, grads, m, v, t, learning_rate, beta1=0.9,
beta2=0.999, epsilon=1e-8):
    m_hat = {}
    v_hat = {}
    for key in weights.keys():
        m[key] = beta1 * m[key] + (1 - beta1) * grads[key]
        v[key] = beta2 * v[key] + (1 - beta2) * np.square(grads[key])
        m_hat[key] = m[key] / (1 - beta1 ** t)
        v_hat[key] = v[key] / (1 - beta2 ** t)
        weights[key] -= learning_rate * m_hat[key] /
        (np.sqrt(v_hat[key]) + epsilon)
    return weights, m, v
```

其中：

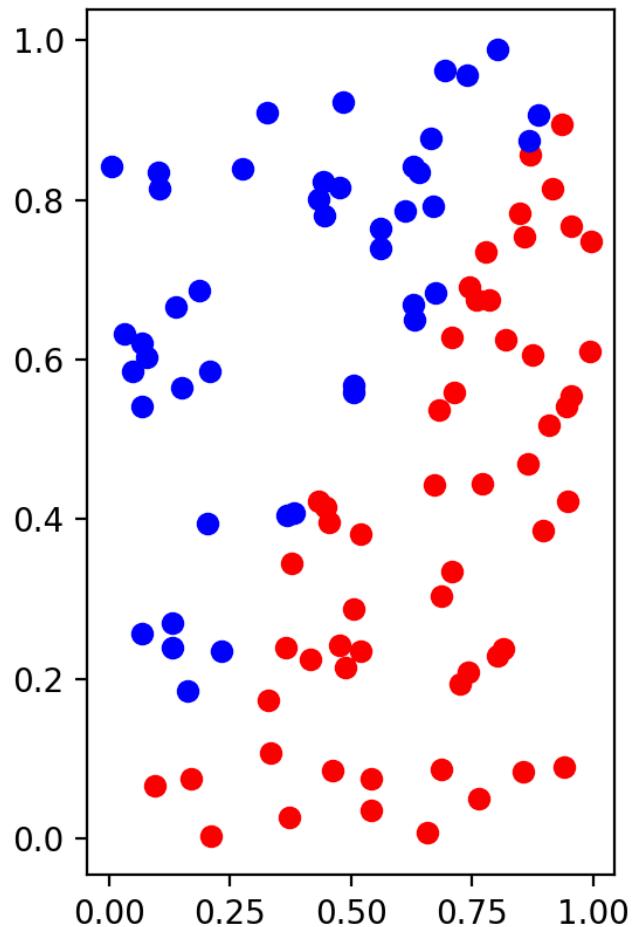
- weights: 當前的權重參數。
- grads: 當前計算出的梯度。
- m: 一階矩估計（動量）。
- v: 二階矩估計 (RMSProp)。
- t: 當前的時間步（迭代次數）。
- learning_rate: 學習率。
- beta1: 一階矩估計的衰減率，通常設為 0.9。
- beta2: 二階矩估計的衰減率，通常設為 0.999。
- epsilon: 防止除零的小數值，通常設為 $1e-8$ 。

- Linear data (acc = 100%)

Ground truth

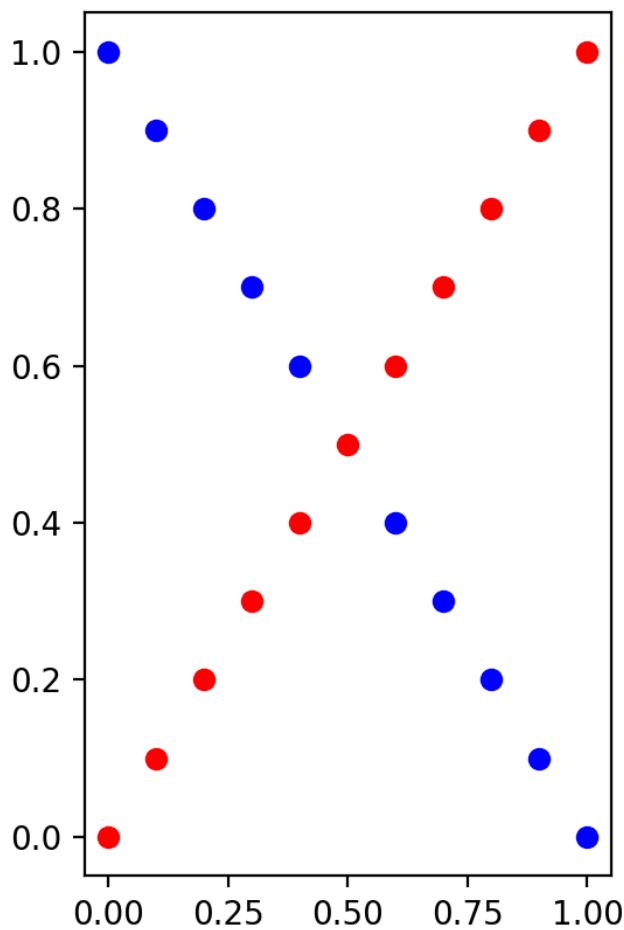


Predict result (ADAM)

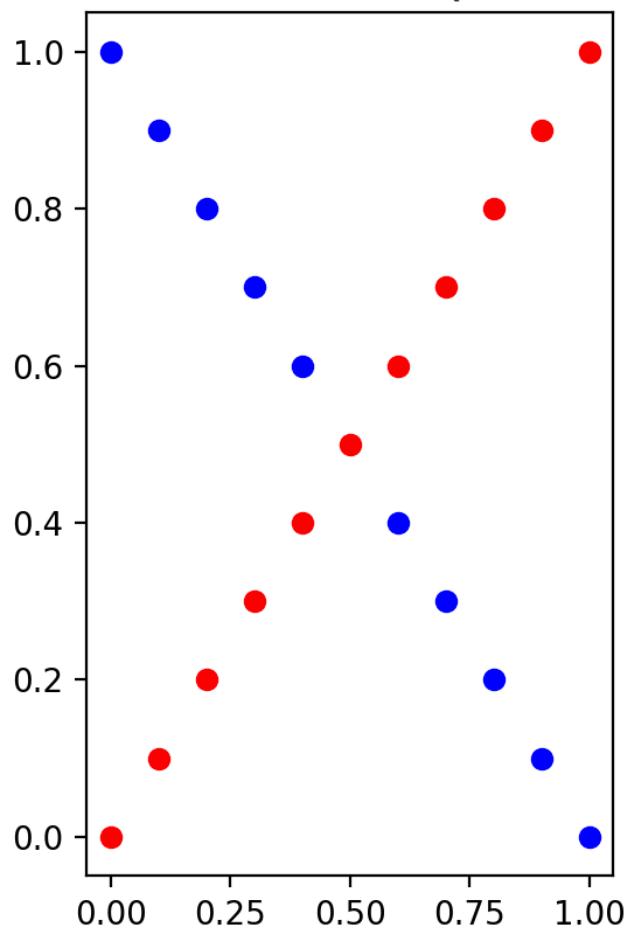


- XOR data (acc = 100%)

Ground truth

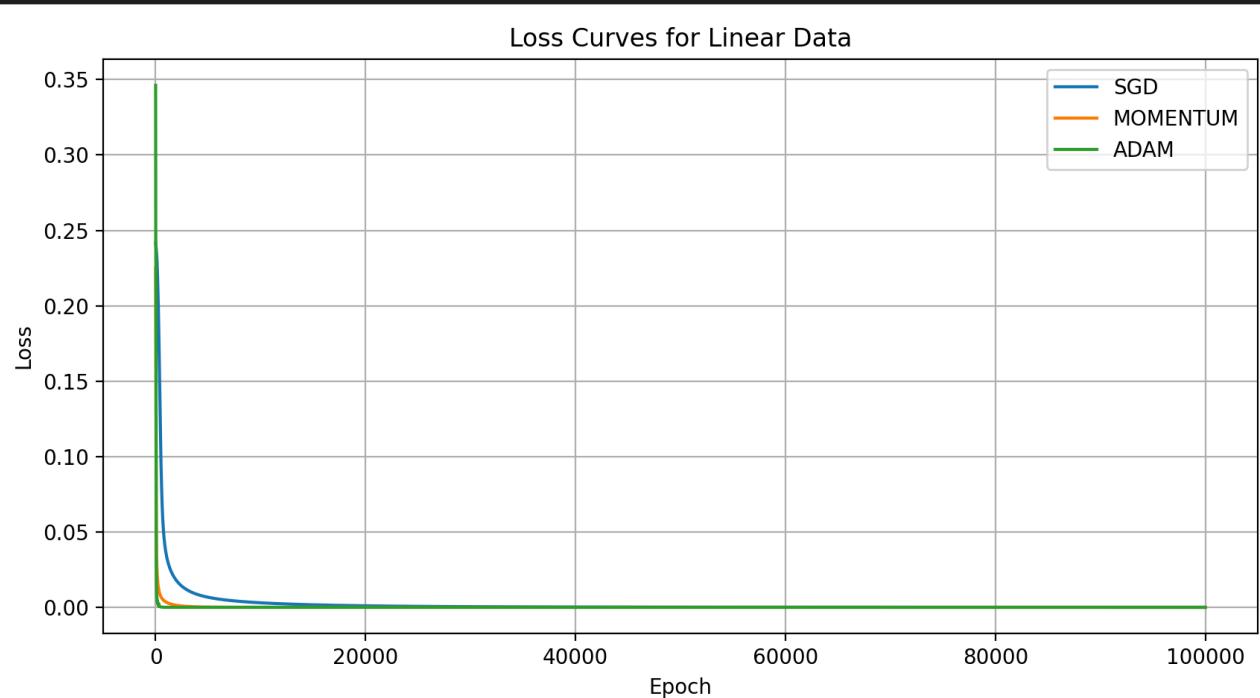


Predict result (ADAM)

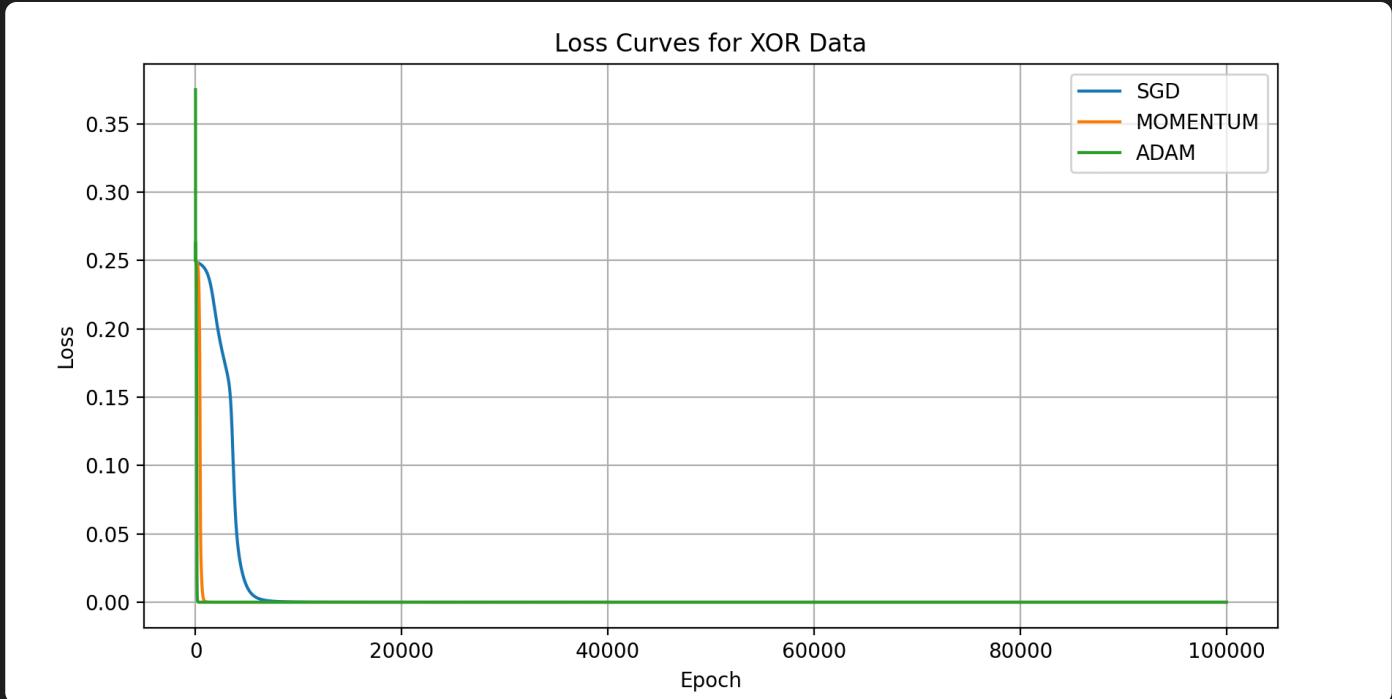


Compare figure

- Linear data



- XOR data



經過觀察可以發現，收斂速度為 **ADAM > MOMENTUM > SGD**

B. Implement different activation functions.

將 `sigmoid` 換成 `relu`

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

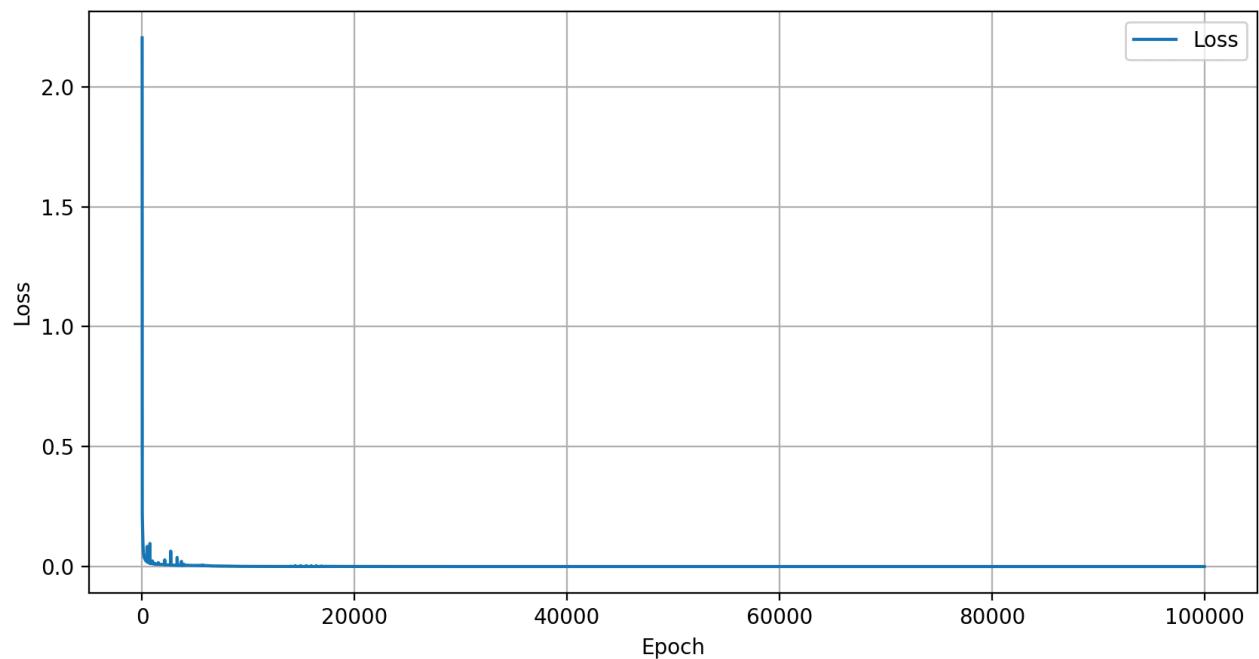
def derivative_relu(x):
    return np.where(x > 0, 1, 0)
```

因為對於 ReLU，通常輸出值的範圍為 $[0, \infty)$ ，這邊的實驗將試著根據訓練數據的輸出值來動態調整閾值，即使用輸出值的中位數作為 threshold，但 acc 還是不能參考...

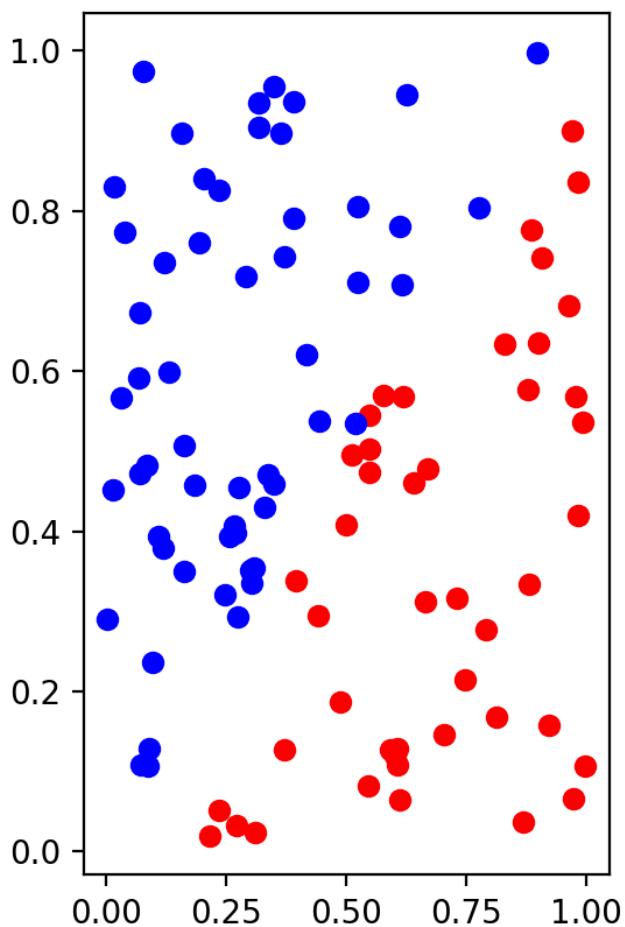
跟 sigmoid 比較起來，好像 relu 的收斂穩定度沒有它來得高。

- Linear data (acc = 98%)

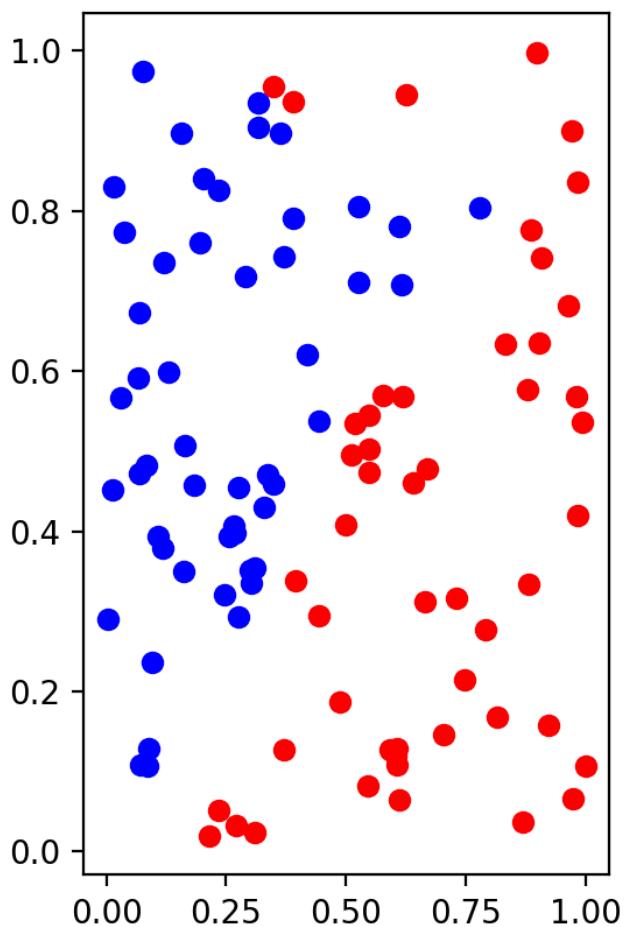
Loss Curve for Linear Data



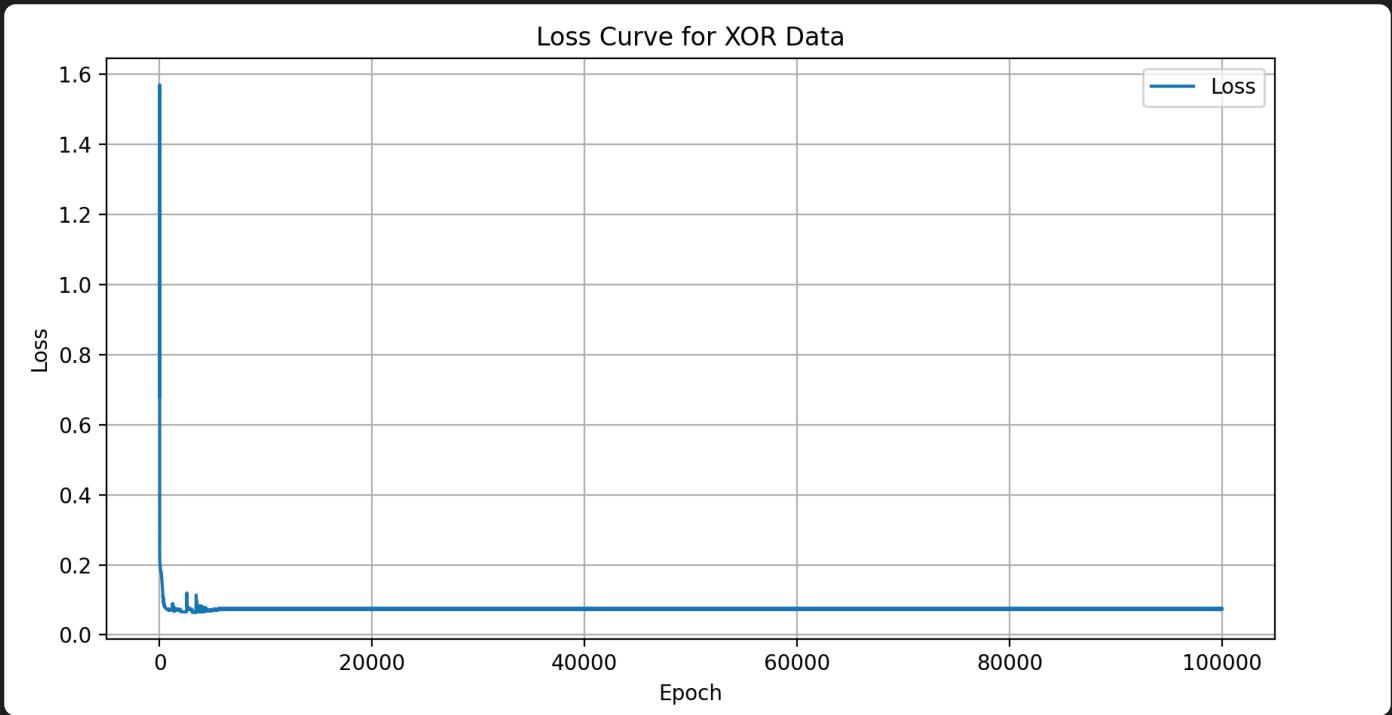
Ground truth



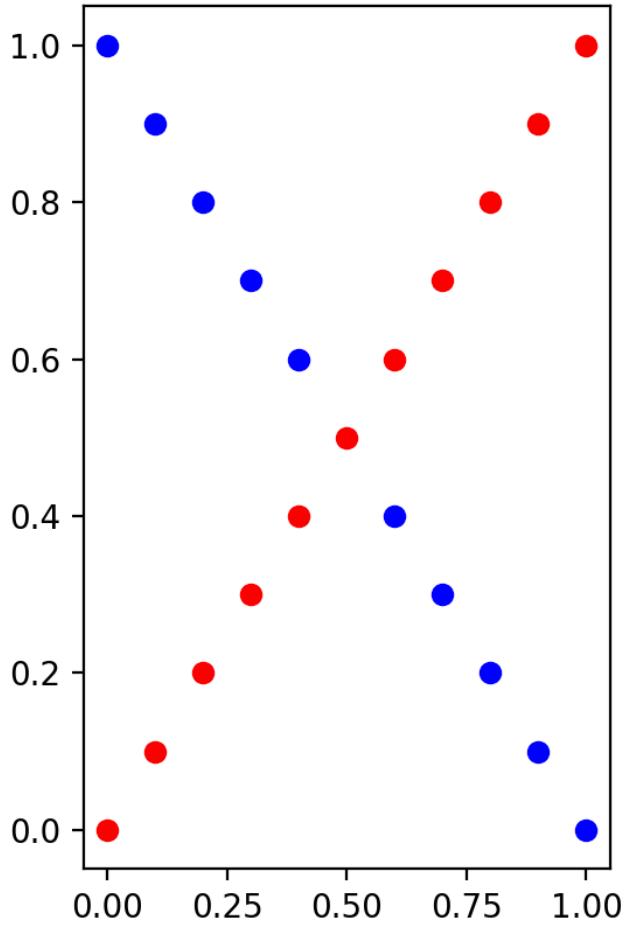
Predict result



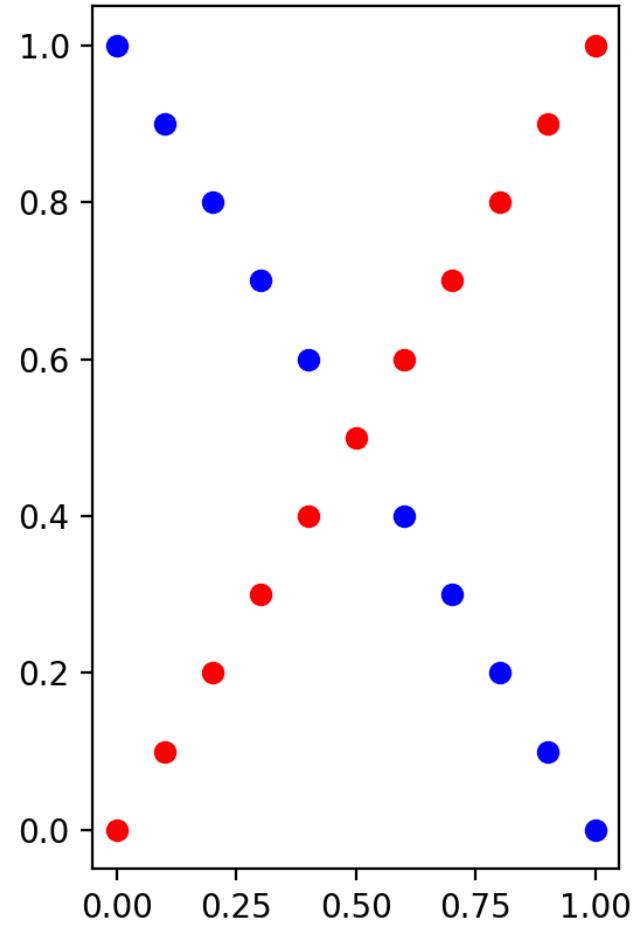
- XOR data (acc = 100%)



Ground truth



Predict result



Reference

- <https://hackmd.io/@allen108108/H1l4zqtp4> (Adagrad、RMSprop、Momentum and Adam – 特殊的學習率調整方式)

- <https://www.brilliantcode.net/1670/convolutional-neural-networks-4-backpropagation-in-kernels-of-cnns/> (卷積核的Back propagation)
- ChatGPT