311551069 余忠旻 Lab1 : back-propagation

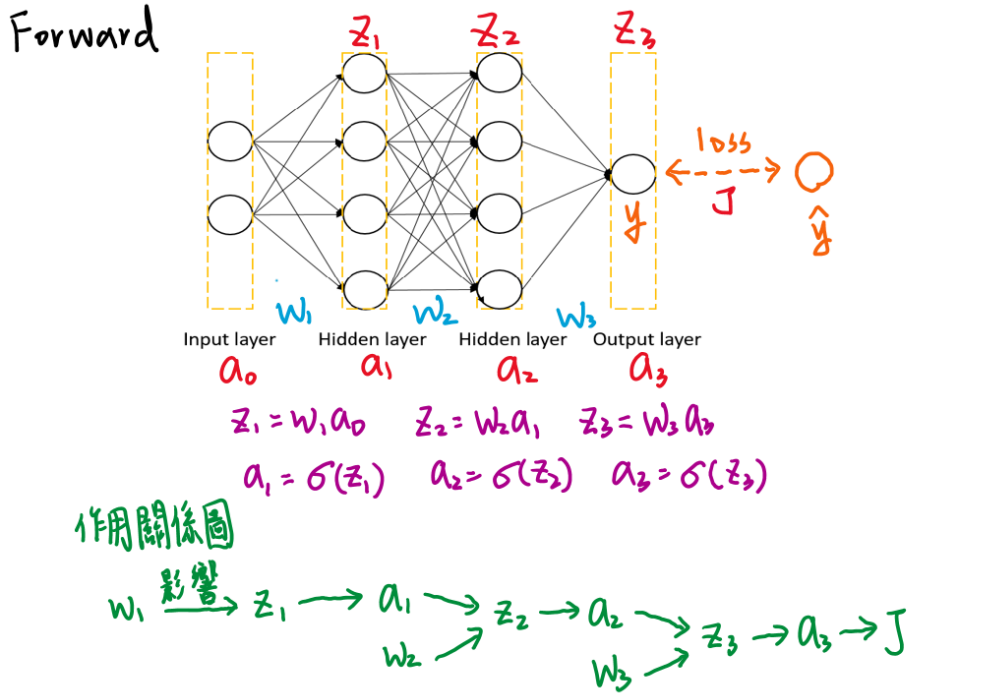
1. Introduction

A. LAB objective

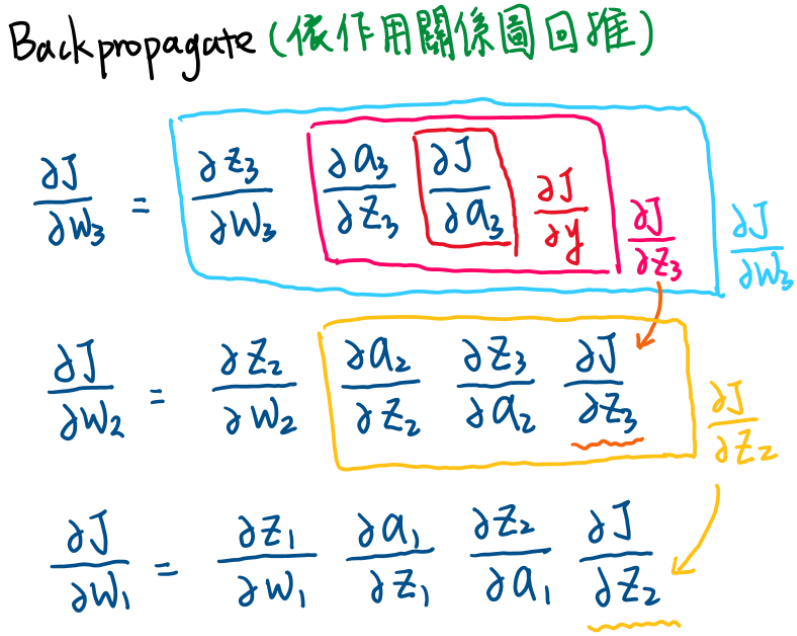
這次 LAB 是建構一個有 2 層 hidden layer 的 fully connected neural network 餵進去 Linear 和 XOR 的 data 來 classify input data

B. Fully connected NN 計算

(1) 根據關係圖可以寫出 forward 的計算式及變數的作用關係圖

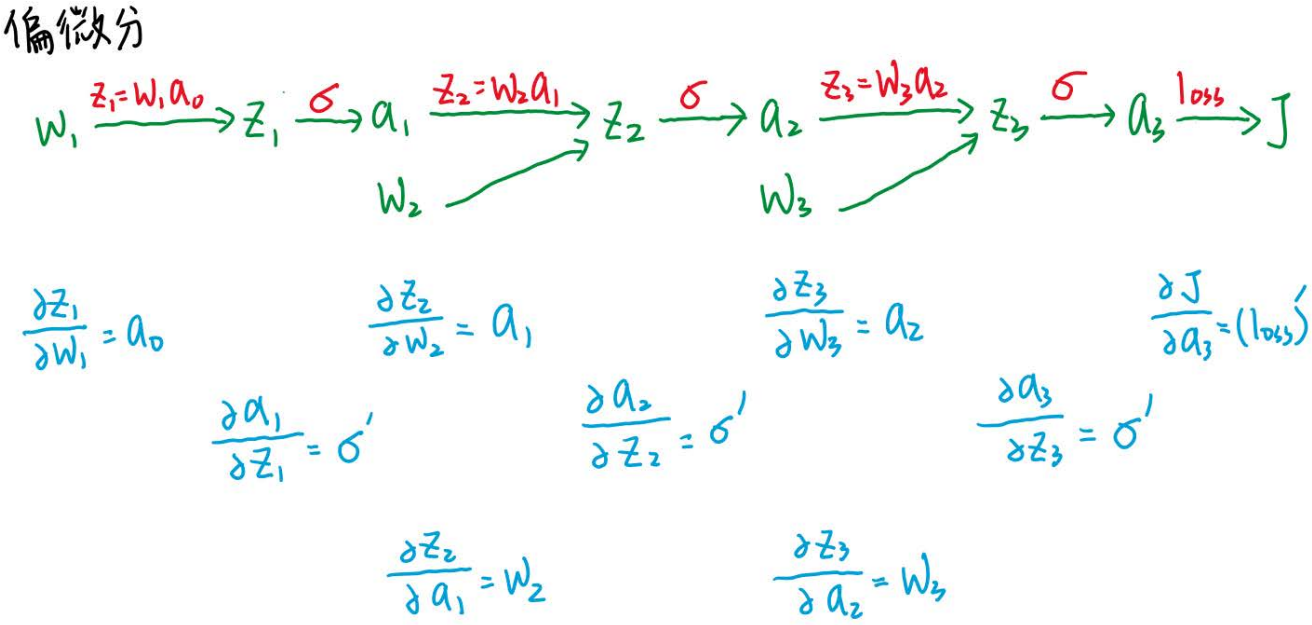


(2) 並且根據作用關係圖可以推回 backpropagation 公式 (利用 chain rule)



(3) 算出上面各個偏微分的值

偏微分



(4) 由於 input data 為 k 筆資料 X_{2*k} (X_{2*k} 為上述計算的 a0)

而 output data 為 y_{1*k} (y_{1*k} 為上述計算的 a3)

$$\therefore\ a1_{n*k} = w1_{n*2} * a0_{2*k}$$

$$a2_{n*k} = w2_{n*n} * a1_{n*k}$$

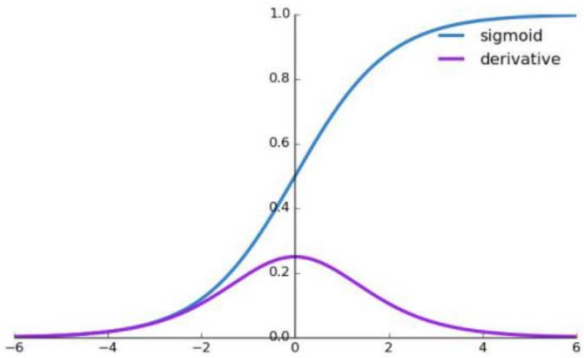
$$a3_{1*k} = w3_{1*n} * a2_{n*k}$$

$\rightarrow$ $w1_{n*2}$, $w2_{n*n}$, $w3_{1*n}$ (weight 維度可得)

2. Experiment setups:

A. Sigmoid functions

Sigmoid function 是其中一種 activation functions，目的是利用非線性方程式，來解決非線性問題，像是可以應用在這次 LAB 的 XOR 這種資料集的分類問題。底下是 sigmoid 以及 sigmoid derivative 的圖：



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

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

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

推導過程：

$$\frac{\partial_\sigma}{\partial_x} = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}(1+e^{-x})^{-1}$$

$$= -(1+e^{-x})^{-2} \times -e^{-x}$$

$$= \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}}$$

$$= \sigma(x)(1-\sigma(x))$$

B. Neural network

根據上面 fully connected NN 關係圖顯示，我們實作了兩層 hidden layer，並且兩層都使用 10 個 neuron。

NN 運作如下：一開始 weight w1, w2, w3 初始化為隨機值就好，維度則可以在 Introduction B(4) 可以算出來，透過 forwarding pass 可以得出 output，之後做 backpropagation，透過計算 gradient 更新 weight，而 learning rate 則是會影響更新幅度，反覆迭代後就能訓練出 model

而 NN 的 forwarding 依照 Introduction B(1) 的圖實作就能實現：

```
def forwarding_pass(func, x, w1, w2, w3):
    a0 = x
    z1 = w1 @ a0
    a1 = activation(func, z1)
    z2 = w2 @ a1
    a2 = activation(func, z2)
    z3 = w3 @ a2
    a3 = sigmoid(z3)
    pred_y = a3
    return pred_y, a0, a1, a2, a3
```

C. Backpropagation

Backpropagation 部分則是依照 Introduction B(2)、B(3)的圖實作就能實現，其中我使用 cross entropy 當作 loss function，而在計算 cross entropy 或 derivative of cross entropy 時會有一項 epsilon 來避免分母或 log 為 0 :
(下面 optimizer 我用的是 Momentum Optimizer，Extra part 會詳細討論)

```
def backpropagation(func, pred_y, y, a3, w3, a2, w2, a1, w1, a0):
    eps = 0.0001
    dJ_da3 = -( y / (pred_y + eps) - (1 - y) / (1 - pred_y + eps) ) # derivative of cross-entropy
    dJ_dz3 = derivative_sigmoid(a3) * dJ_da3
    dJ_dw3 = dJ_dz3 @ a2.T

    dJ_da2 = w3.T @ dJ_dz3
    dJ_dz2 = derivative_activation(func, a2) * dJ_da2
    dJ_dw2 = dJ_dz2 @ a1.T

    dJ_da1 = w2.T @ dJ_dz2
    dJ_dz1 = derivative_activation(func, a1) * dJ_da1
    dJ_dw1 = dJ_dz1 @ a0.T

    return dJ_dw1, dJ_dw2, dJ_dw3
```

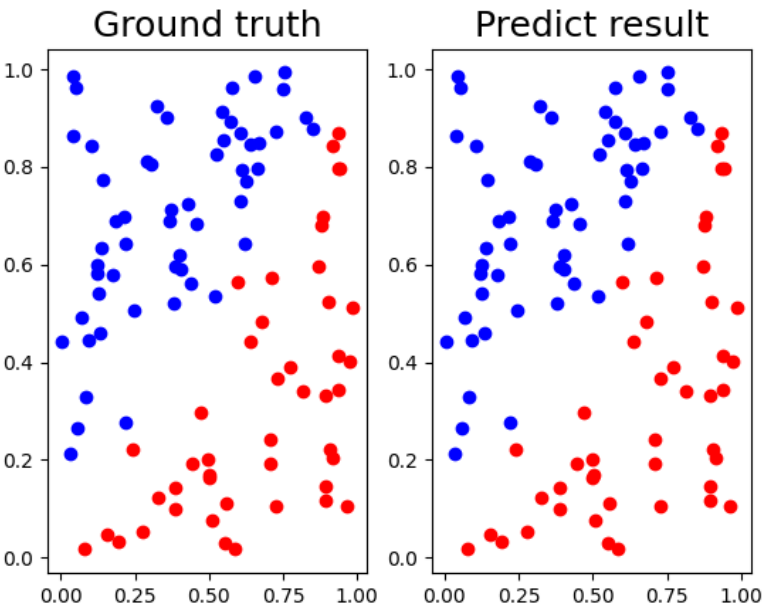
```
def weight_update(optimizer, learning_rate, n, dJ_dw1, dJ_dw2, dJ_dw3, w1, w2, w3, m1, m2, m3):
    if optimizer == True:
        m1 = 0.9 * m1 - learning_rate * (dJ_dw1 / n)
        m2 = 0.9 * m2 - learning_rate * (dJ_dw2 / n)
        m3 = 0.9 * m3 - learning_rate * (dJ_dw3 / n)
        w1 = w1 + m1
        w2 = w2 + m2
        w3 = w3 + m3
    else:
        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, m1, m2, m3
```

3. Results of your testing

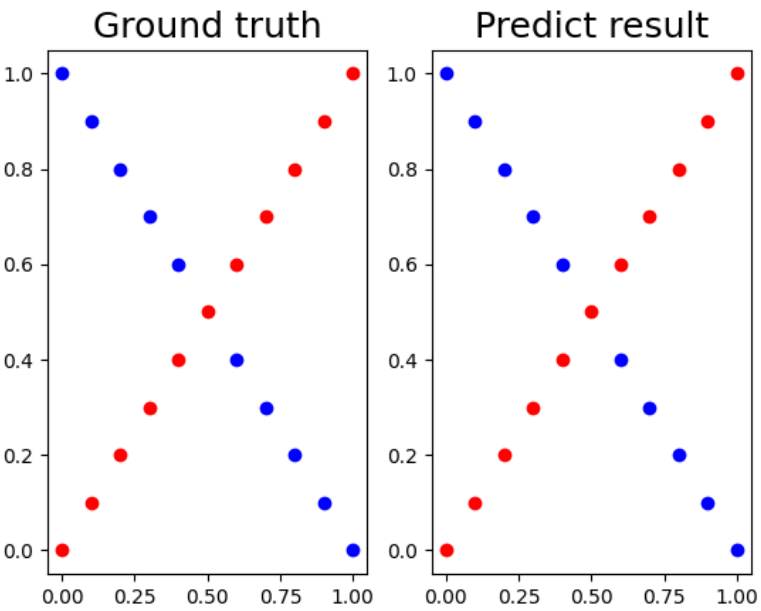
A. Screenshot and comparison figure

兩層都使用 10 個 neuron，learning rate = 0.1

Linear data: accuracy = 100%



XOR data: accuracy = 100%



B. Show the accuracy of your prediction

Linear training

```
Training ...
epoch 100 loss : 0.4137254512412997
epoch 200 loss : 0.10116533042046656
epoch 300 loss : 0.05668578606138413
epoch 400 loss : 0.04180481958449406
epoch 500 loss : 0.03434249936403518
epoch 600 loss : 0.029757891362925847
epoch 700 loss : 0.026583322044295716
epoch 800 loss : 0.02420773354145286
epoch 900 loss : 0.02233254513705826
epoch 1000 loss : 0.020794662929888387
epoch 1100 loss : 0.01949728334173696
epoch 1200 loss : 0.01837909643083457
epoch 1300 loss : 0.017399186861810162
epoch 1400 loss : 0.01652904905106762
epoch 1500 loss : 0.01574809722661112
epoch 1600 loss : 0.015041008263291172
epoch 1700 loss : 0.014396079319518196
epoch 1800 loss : 0.013804173914359832
epoch 1900 loss : 0.013258023091472398
epoch 2000 loss : 0.012751748457094094
epoch 2100 loss : 0.012280528193250505
epoch 2200 loss : 0.011840357770606916
epoch 2300 loss : 0.011427874953068922
epoch 2400 loss : 0.011040229434523296
epoch 2500 loss : 0.010674984093428456
epoch 2600 loss : 0.010330039062688604
epoch 2700 loss : 0.010003572542542059
epoch 2800 loss : 0.009693994091118126
epoch 2900 loss : 0.009399907346147008
epoch 3000 loss : 0.009120079968057282
epoch 3100 loss : 0.008853419178613898
epoch 3200 loss : 0.008598951682981972
epoch 3300 loss : 0.008355807060436789
epoch 3400 loss : 0.008123203925476876
epoch 3500 loss : 0.007900438320747136
epoch 3600 loss : 0.00768687392225885
epoch 3700 loss : 0.007481933727166847
epoch 3800 loss : 0.007285092962733586
epoch 3900 loss : 0.007095873007664075
epoch 4000 loss : 0.006913836157748019
epoch 4100 loss : 0.006738581099610001
epoch 4200 loss : 0.006569738981468081
epoch 4300 loss : 0.006406969989722337
epoch 4400 loss : 0.006249960356104998 5
epoch 4500 loss : 0.006098419732920165
epoch 4600 loss : 0.005952078884247736
epoch 4700 loss : 0.005810687649406242 5
epoch 4800 loss : 0.005674013141855108
epoch 4900 loss : 0.005541838152380272
epoch 5000 loss : 0.005413959730086311
```

Linear testing

```
Testing ...
[[2.06731907e-07 9.99996052e-01 3.69096878e-07 9.99999605e-01
  9.99999573e-01 7.45664570e-04 9.99999403e-01 1.04259632e-07
  9.99999402e-01 9.99999543e-01 9.99996468e-01 9.99999359e-01
  9.99999292e-01 1.80406256e-06 9.99999584e-01 9.99999596e-01
  9.99996806e-01 9.78849586e-01 1.35494978e-07 2.15284394e-07
  9.68142772e-08 1.93109022e-07 1.30775805e-07 6.05360005e-03
  1.60466881e-07 1.04294108e-07 9.99999523e-01 3.75337074e-06
  9.99999175e-01 9.99999603e-01 9.99999562e-01 1.02253202e-07
  4.28676333e-07 1.58632867e-07 9.99999427e-01 1.02924403e-07
  9.99993563e-01 9.99998781e-01 1.13077125e-07 9.99999550e-01
  9.99999446e-01 9.99999061e-01 3.86597537e-03 9.99999507e-01
  9.99999550e-01 9.99999390e-01 9.99999556e-01 1.06538877e-07
  9.31982839e-08 9.99998922e-01 1.37818544e-03 1.13254299e-07
  9.12490069e-08 9.99999533e-01 9.93221464e-08 9.99999418e-01
  9.99998538e-01 9.99999559e-01 5.15216538e-07 9.04828252e-06
  3.83456039e-07 9.99968571e-01 9.99999247e-01 9.99999448e-01
  2.58863126e-07 9.99997487e-01 9.98399894e-01 9.99998964e-01
  1.15016501e-07 9.42468656e-08 1.27517792e-06 9.57630252e-08
  1.17064852e-07 9.99999606e-01 4.41185022e-06 9.99989888e-01
  9.99999029e-01 9.99999558e-01 7.10464319e-07 9.99999498e-01
  8.51606797e-01 9.99999444e-01 1.15298151e-07 1.35617493e-07
  9.99993842e-01 4.02983794e-07 9.99998980e-01 9.99998294e-01
  9.99999521e-01 1.96776779e-07 9.99999517e-01 9.99999553e-01
  1.07298111e-07 9.99999485e-01 1.08377494e-07 1.39734400e-07
  9.99998140e-01 1.41893041e-07 9.99998592e-01 9.89086070e-01]]
accuracy : 100.0 %
```

XOR training

```
Training ...
epoch 100 loss : 0.6653445425154287
epoch 200 loss : 0.6474825628395305
epoch 300 loss : 0.6281300112123017
epoch 400 loss : 0.6029398011611152
epoch 500 loss : 0.5681956815108885
epoch 600 loss : 0.5209395546048553
epoch 700 loss : 0.46104717288420966
epoch 800 loss : 0.39493575060459146
epoch 900 loss : 0.3324379195583182
epoch 1000 loss : 0.27886286738038296
epoch 1100 loss : 0.23465158720650267
epoch 1200 loss : 0.19834049902455506
epoch 1300 loss : 0.1682558743717377
epoch 1400 loss : 0.14308163870213633
epoch 1500 loss : 0.12191652328043755
epoch 1600 loss : 0.1041440574947012
epoch 1700 loss : 0.08929008599880518
epoch 1800 loss : 0.07693802291550701
epoch 1900 loss : 0.06670093536606383
epoch 2000 loss : 0.05822359308809774
epoch 2100 loss : 0.051191530523873215
epoch 2200 loss : 0.045336393038808945
epoch 2300 loss : 0.04043565627618318
epoch 2400 loss : 0.036308402171802236
epoch 2500 loss : 0.032809304552965934
epoch 2600 loss : 0.02982235021125366
epoch 2700 loss : 0.027255107748218602
epoch 2800 loss : 0.025033858942033017
epoch 2900 loss : 0.023099630548126936
epoch 3000 loss : 0.02140503751349847
epoch 3100 loss : 0.019911806113092335
epoch 3200 loss : 0.018588844358191903
epoch 3300 loss : 0.01741074286230665
epoch 3400 loss : 0.016356609915877687
epoch 3500 loss : 0.01540916431315633
epoch 3600 loss : 0.014554026450904966
epoch 3700 loss : 0.013779161964225408
epoch 3800 loss : 0.013074442944398293
epoch 3900 loss : 0.012431300083673007
epoch 4000 loss : 0.011842445417442051
epoch 4100 loss : 0.011301650129760718
epoch 4200 loss : 0.010803565516795802
epoch 4300 loss : 0.010343577949278576
epoch 4400 loss : 0.009917690757634878
epoch 4500 loss : 0.009522427547179102
epoch 4600 loss : 0.009154752659427813
epoch 4700 loss : 0.008812005421782853
epoch 4800 loss : 0.008491845540700945
epoch 4900 loss : 0.008192207544661045
epoch 5000 loss : 0.007911262611439804
```

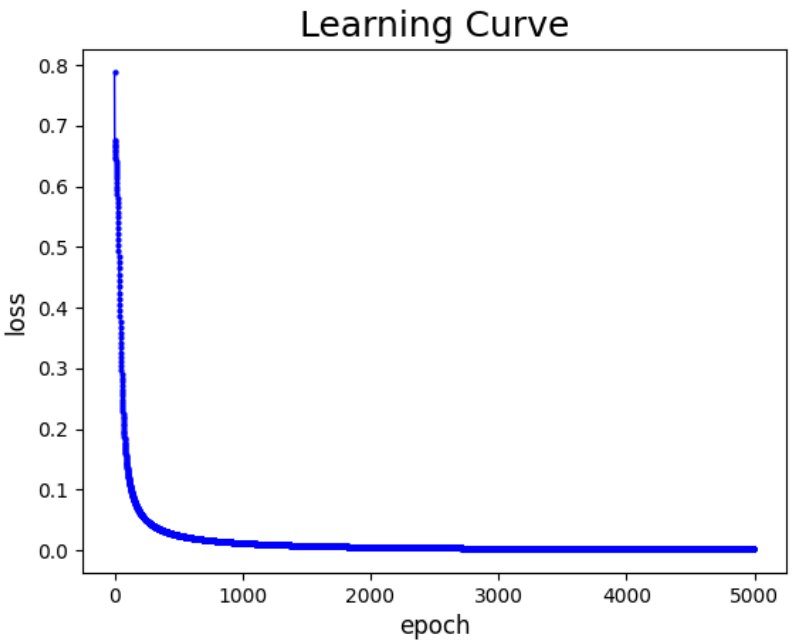
XOR testing

```
Testing ...
[[0.00705865 0.99905373 0.00726961 0.99897215 0.00746848 0.99877455
  0.00763297 0.99795665 0.00774196 0.96028033 0.00777871 0.00773351
  0.96916213 0.00760493 0.99745518 0.00739952 0.9978112  0.00712996
  0.9978055  0.00681246 0.99777819]]
accuracy : 100.0 %
```

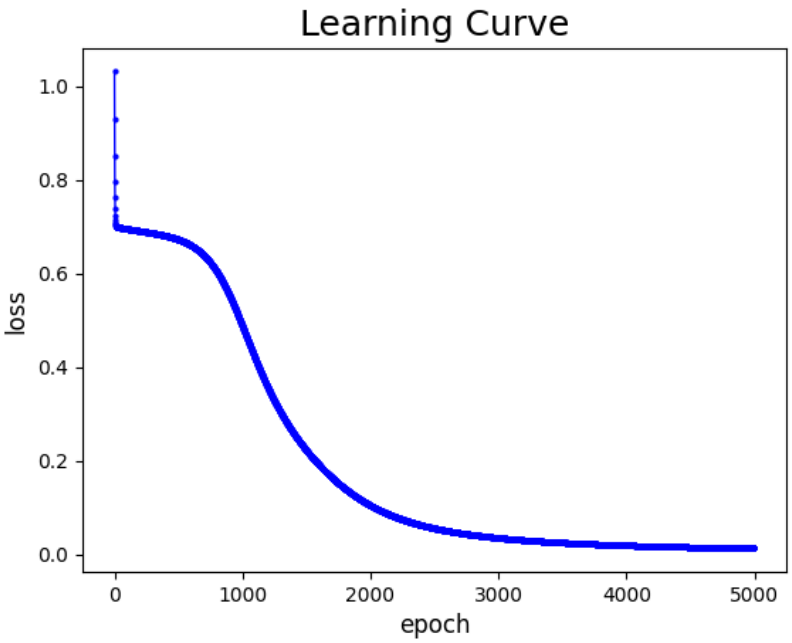
D. Learning curve (loss, epoch curve)

兩層都使用 10 個 neuron，learning rate = 0.1

Linear data: accuracy = 100%



XOR data: accuracy = 100%



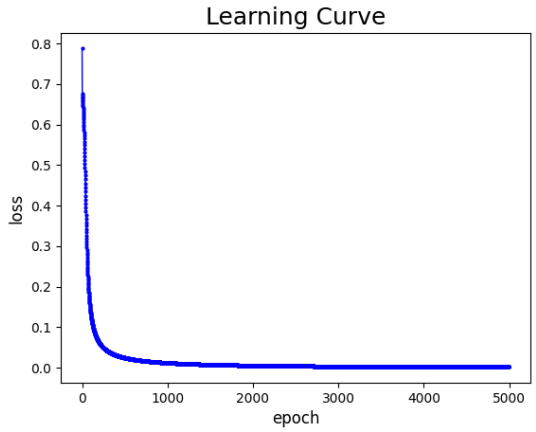
4. Discussion

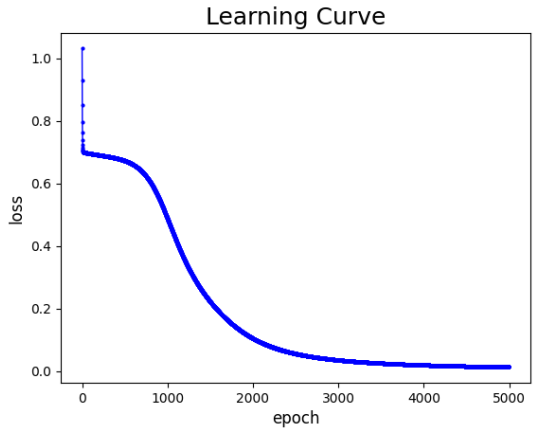
A. Try different learning rates

當 learning rate 越大時，更新 weight 的幅度越大，因此 loss 能夠越快達到收斂。下面是跑 learning rate 為 0.1, 1, 0.01 的 learning curve

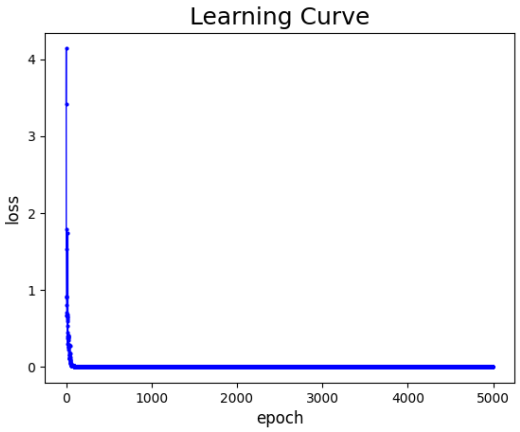
Linear data **XOR data**

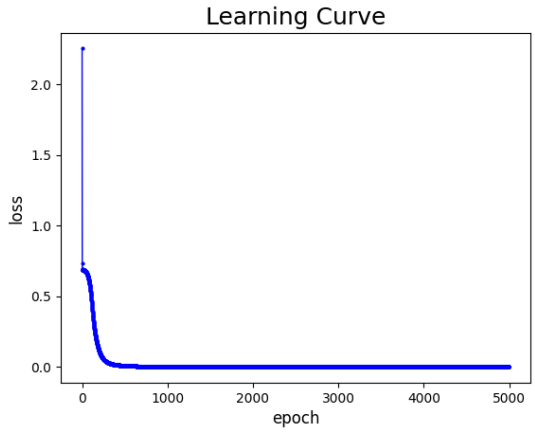
Origin: learning rate = 0.1



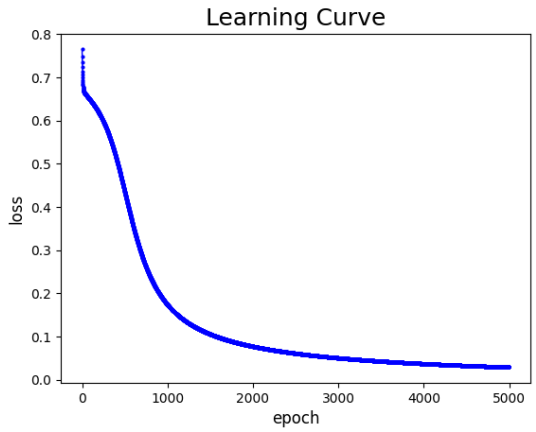


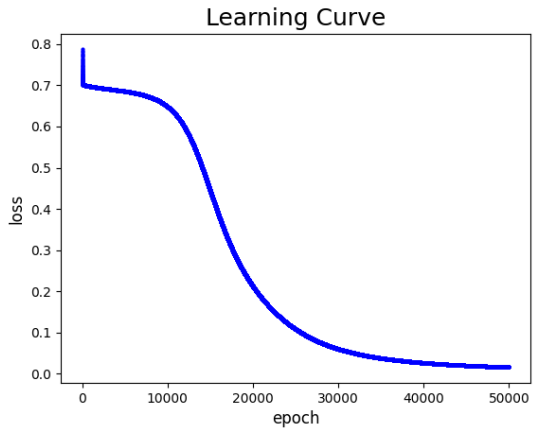
learning rate = 1





learning rate = 0.01



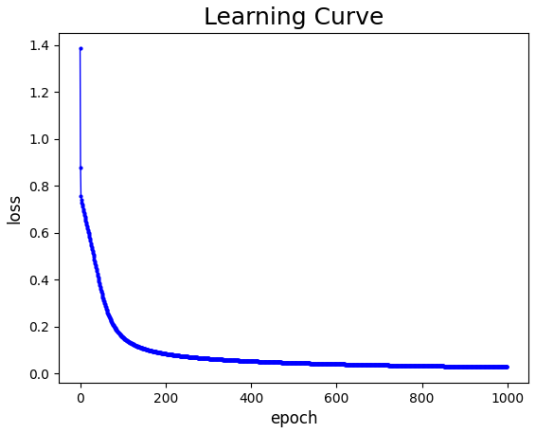


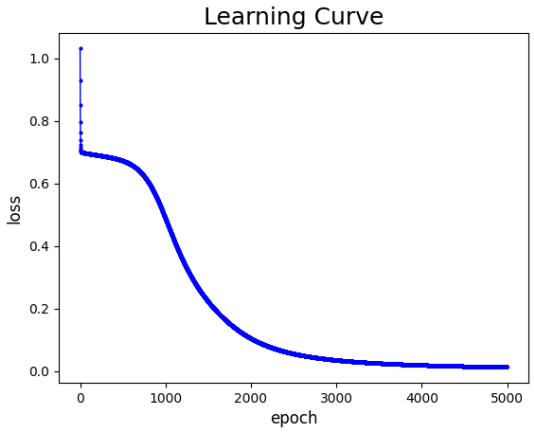
B. Try different numbers of hidden units

改變 W(nxn)大小，hidden units 總數也會跟著改變，隨著 n 變大，hidden units 變多，loss 下降速度也越快。下面是跑 n 為 10, 15, 5 的 learning curve

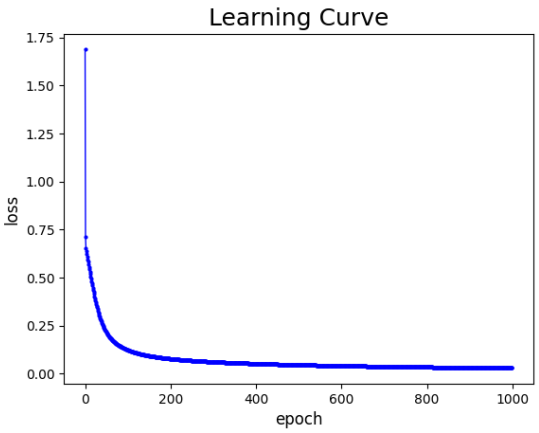
Linear data **XOR data**

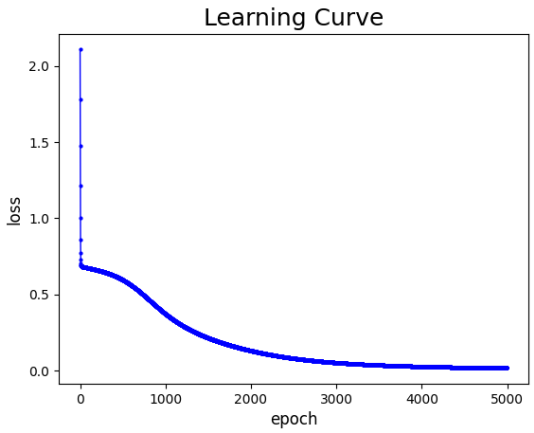
Origin: n = 10



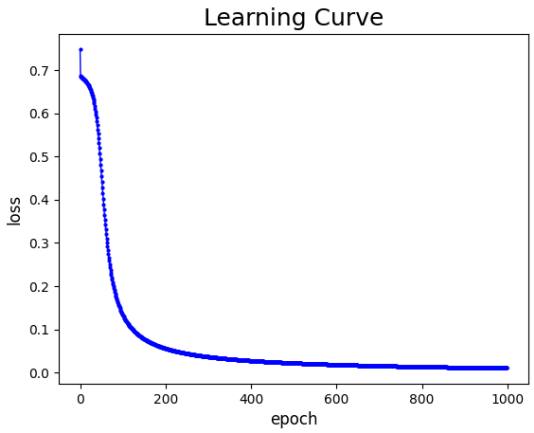


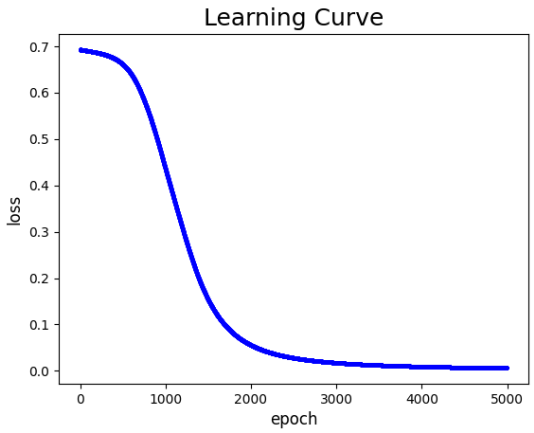
n = 15





n = 5

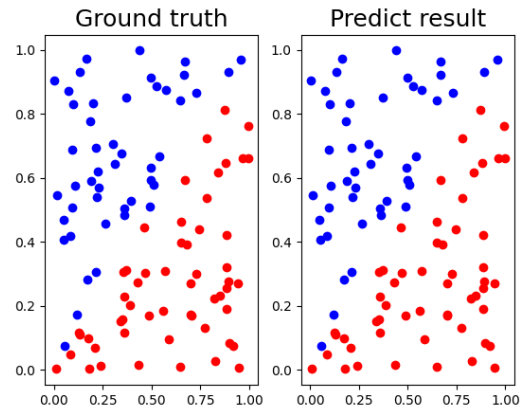


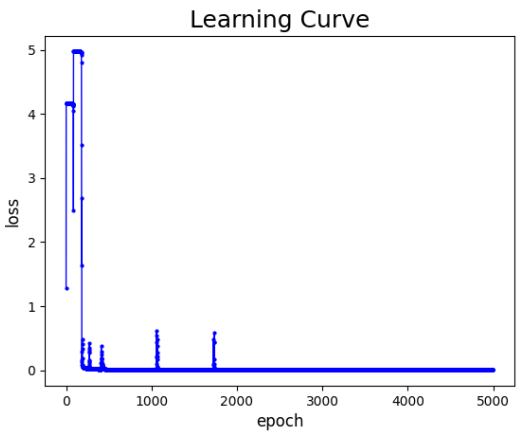


C. Try without activation functions

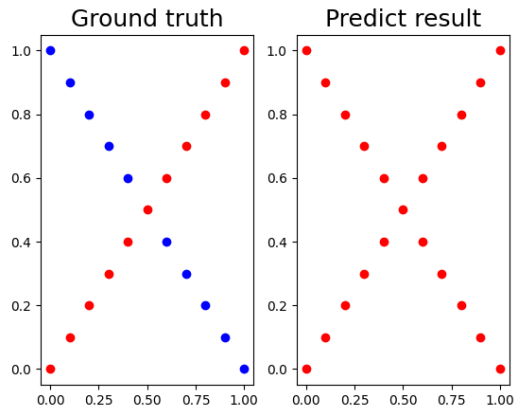
Without activation function，XOR 的準確度變得相當低，因為缺少 activation function 的參與，無法有效處理 nonlinear 的 data
在 learning curve 部分可以看到 linear data，相較使用 sigmoid 來說 loss 震盪蠻明顯的，而 XOR data 部分則是 loss 則是停在 0.6920093402625231 之後就無法往下掉了

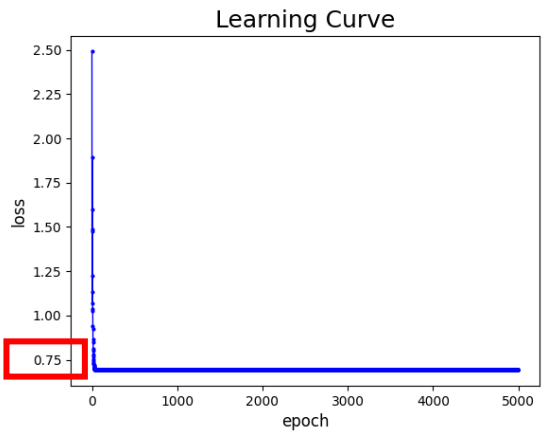
Linear data





XOR data





5. Extra

A. Implement different optimizers.

Origin: Stochastic Gradient Decent (SGD)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

Different optimizer: Momentum Optimizer

此優化器為模擬物理動量的概念，在同方向的維度上學習速度會變快，方向改變的時候學習速度會變慢。下圖可以看到 momentum optimizer 更新公式，其中 ***Vt*** 代表更新速度，會跟上一次的更新有關，如果上一次的梯度跟這次同方向的話，更新速度變大(梯度增強)，如果方向不同，更新速度變小(梯度減弱)，β 這參數可以想像成物理動量中的空氣阻力或是地面摩擦力，通常設定成 0.9，來控制 ***Vt-1*** 對 ***Vt*** 的影響大小

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial W}$$

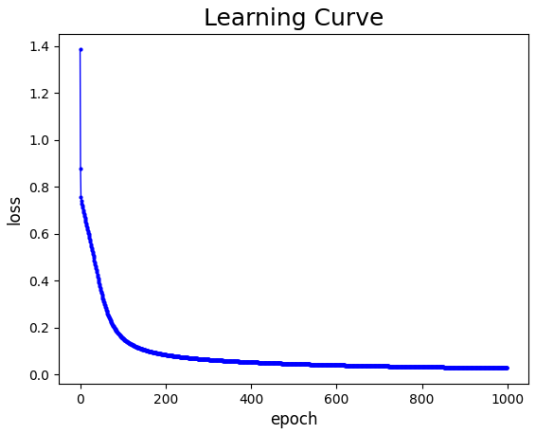
$$W \leftarrow W + V_t$$

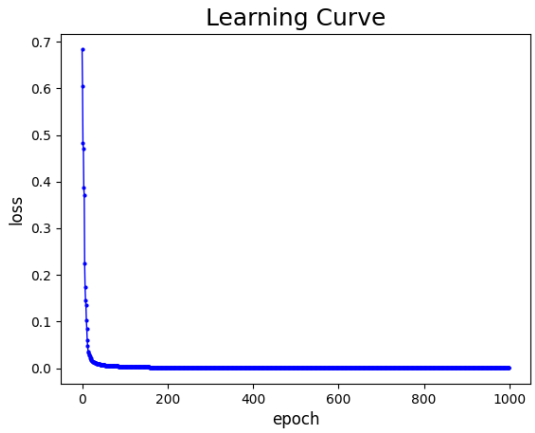
下面是跑 NN，兩層都使用 10 個 neuron，learning rate = 0.1

可以看見使用 Momentum Optimizer 會比 SGD 還要快讓 loss 收斂

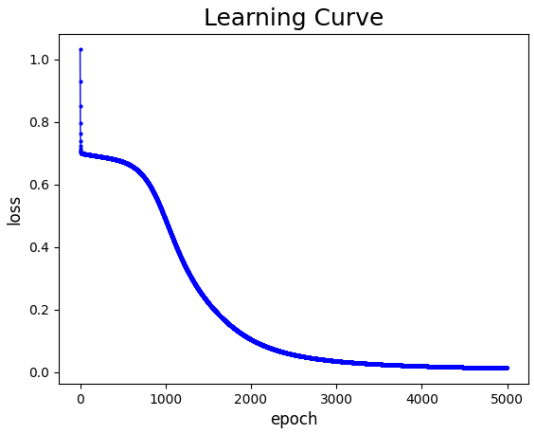
Stochastic Gradient Decent (SGD) **Momentum Optimizer**

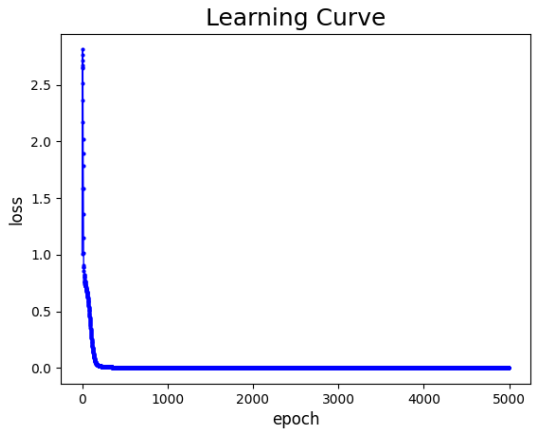
Linear data





XOR data





B. Implement different activation functions.

在 Introduction B(1) 的 NN 關係圖中把 a1 , a2 換成其他 activation function，最後一層的 output layer a3 = sigmoid(z3) 要維持不變，因為值域要在 0 和 1 之間，Backpropagation 同理。

下面是改用 ReLU 和 tanh，兩層都使用 10 個 neuron，分別跑 learning rate = 0.1 和 learning rate = 1

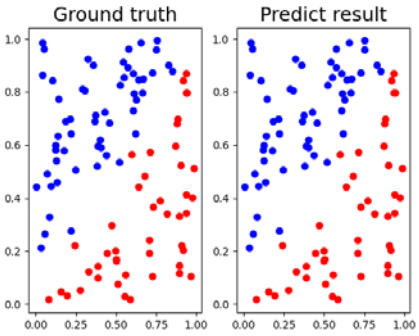
```
def ReLU(x):
    x = np.maximum(0.0, x)
    return x

def derivative_ReLU(x):
    x[x <= 0] = 0
    x[x > 0] = 1
    return x

def tanh(x):
    return np.tanh(x)

def derivative_tanh(x):
    return 1 - np.square(x)
```

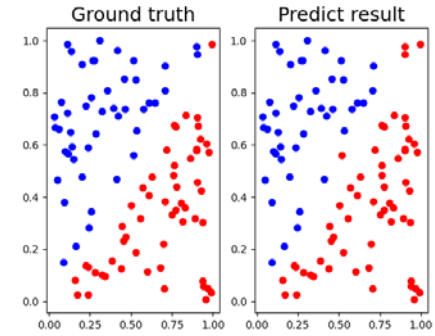
在 learning rate = 0.1，三種 activation functions 表現都還不錯，雖然 ReLU training 前期 loss 會有所震盪，但最後都有收斂。

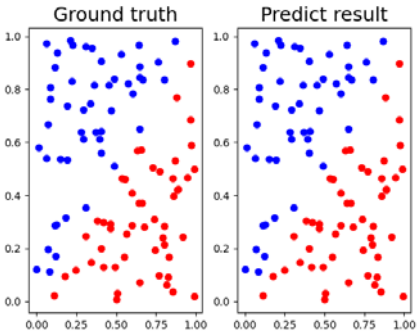
在 learning rate = 1，三種 activation functions 中 sigmoid 表現最好，ReLU 在 XOR data 震盪特別明顯，linear data loss 也相當高，tanh 則是在 linear data 時不一定每次都能收斂到 optimum point，雖然有時準確度能達到 90%以上，但很常準確度只有 40、50%。因此，在訓練時不應該把 learning rate 太高，這樣會導致 model 無法找到 optimum point。

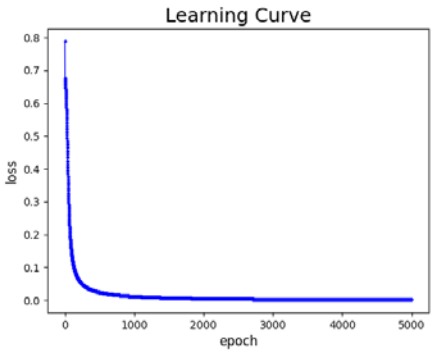
sigmoid **ReLU** **tanh**

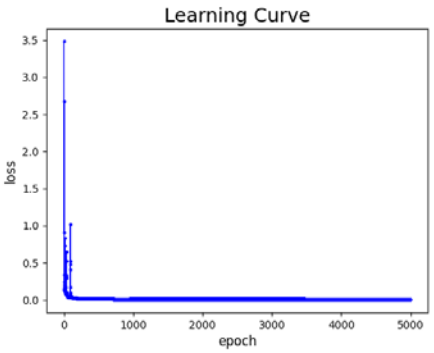
learning rate = 0.1 on linear data:

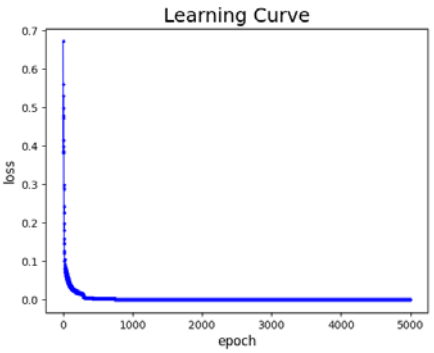




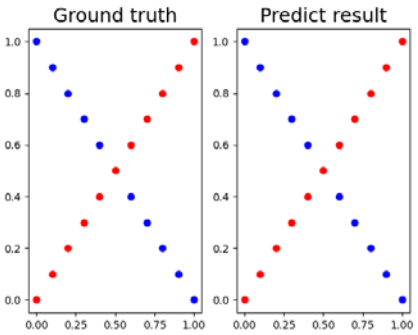


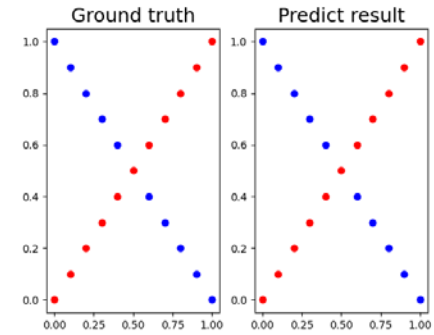


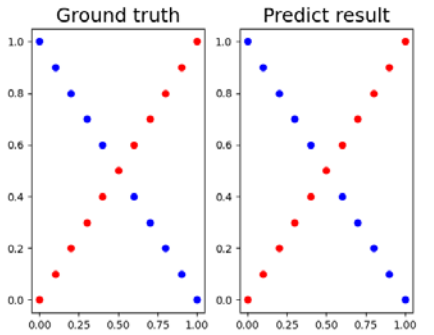


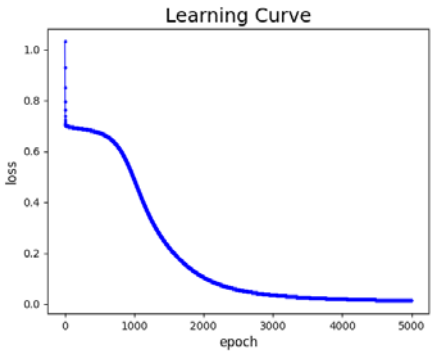


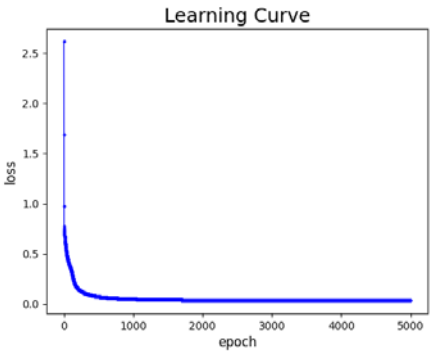
learning rate = 0.1 on XOR data:

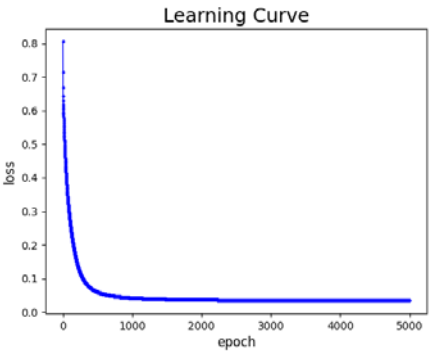






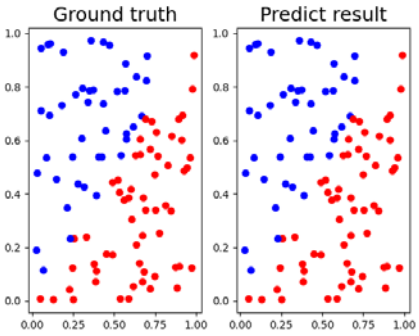


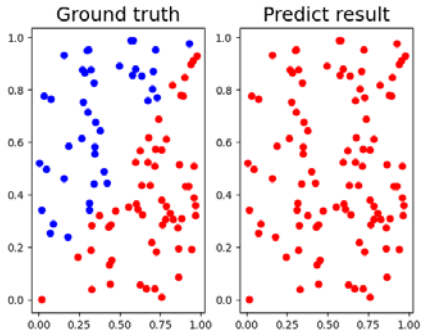


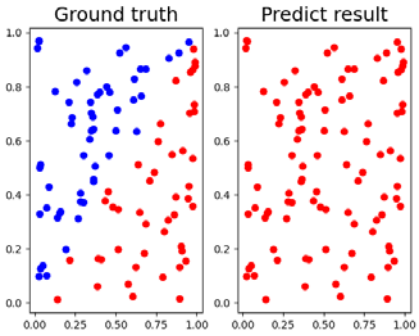


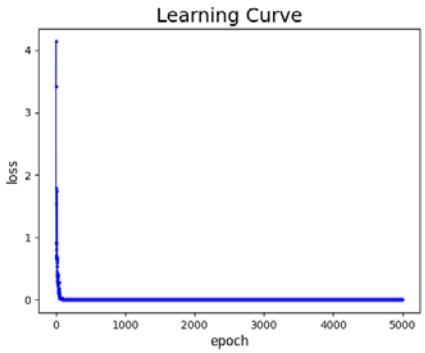
sigmoid **ReLU** **tanh**

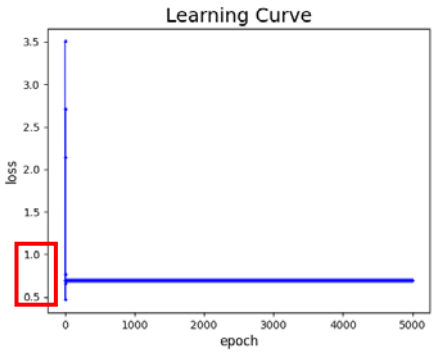
learning rate = 1 on linear data:

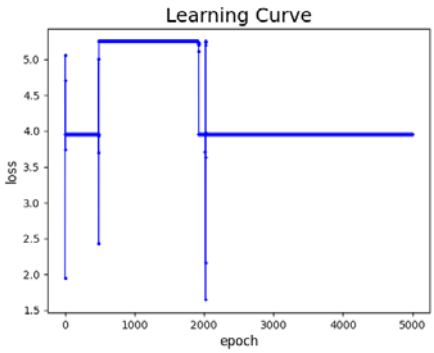












learning rate = 1 on XOR data: