Deep Learning/Deep Learning Labs

Experiment Report

# Lab1: back-propagation

資科工碩

郭建廷　313551072

# 目錄

# 1. Introduction

在這份報告中在 Experiment setups 中，首先帶出最基本的 Artificial Neural，說明不帶有激活函數的 Artificial Neural 無法處理非線性的問題，而代出 Sigmoid 當作例子，並且表示用 python 實作 Sigmoid function 和算其導函數的式子。

接著在 Neural network 時會先講解最基本的 Machine Learning 概念，帶出需要去修正的模型參數與其原理，接著說明 Artificial Neural 的概念與本實驗報告中 Artificial Neural 和 Neural Network model 的設計方式。Backpropagation 則是會說明計算梯度和修正模型參數的算法，會以一個簡單的例子做說明，最後以 Python 程式碼的實例做結尾。

在 Results of testing 章節中會表示基本測試資料的訓練成果，比較不同 learning rate、不同數量的 hidden unit、和使用不同的 activation function。

在 Results of testing 章節中的 D 子項，則是說明了使用不同的資料集，除了原先所規定的 Linear 與 Non-Linear(XOR)版本，也有調整 linear 的樣本數量(n)，額外設計一個帶有 Noise 的資料集來測試難度較高的 non-linear 問題。在測試這些不同的資料集時，也會針對調整超參數(hyperparameter)、網路架構、Optimizer 等情況，來記錄訓練的 loss、 training 和 testing 的效果。

在 Discussion 時，則會針對 Results of testing 所做的所有測試去討論，並且在 Discussion 的 D 子項中，會講解測試下來不同參數、架構之間的差異。

最後的 Extra 章節則是講解實作不同 activation function(ReLU)與 Optimizer(SGD、Momentum-GD)。

# 2. Experiment setups

## A. Sigmoid functions

一個基本的 Artificial Neural 屬於一種線性的方程式，表示方式如下：

$$y = w^T x + b$$

由於在處理非線性問題時，如 XOR 問題時會沒辦法用來區分，而激活函數本身帶有非線性的關係，因此在設計解決 XOR 問題時，都會在人工神經網路中加入一個激活函數，來解決非線性的問題，甚至是更加複雜的問題。而 Sigmoid 算是一個很常見的激活函數(Activation function)，除了達到非線性的狀態，Sigmoid 也可以做到正規化的輸出，下方圖 2.1 表示了他是一個 S 型曲線的函數圖形，可以將輸出限制在[0,1]之間。

在 Sigmoid 的設計上使用下方的程式碼，列印出來的 function 如圖 2.1：

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

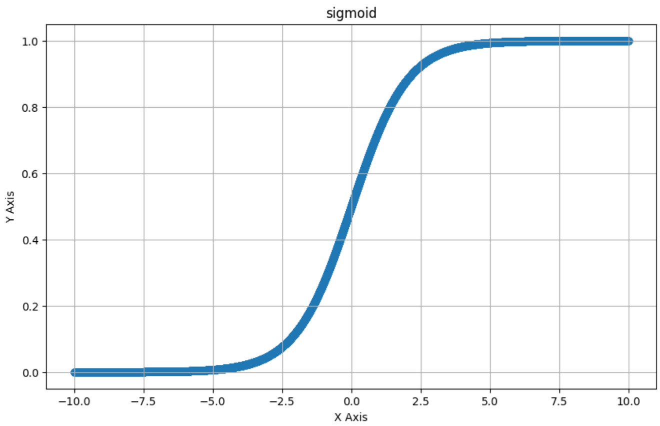



圖 2.1、Sigmoid 函數圖

Sigmoid function 的微分推倒如下：

$$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}} \quad ，Assume\ f(x) = 1 + e^{-x}，\sigma(x) = \frac{1}{f(x)}$$

$$f'(x) = \frac{d}{dx}\left(\frac{1}{f(x)}\right) = -\frac{f'(x)}{\left(f(x)\right)^2} = -\frac{-e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$(1+e^{-x})^2 = \left(1+\frac{1}{e^x}\right)^2 = \frac{(e^x+1)^2}{e^{2x}}$$

$$\sigma'(x) = \frac{\frac{1}{e^x}}{\frac{(e^x+1)^2}{e^{2x}}} = \frac{1}{e^x} \times \frac{e^{2x}}{(e^x+1)^2} = \frac{e^x}{(e^x+1)^2}$$

由原先的 function 可以推出 $\sigma(x) = \frac{e^x}{e^x+1}$ 因此

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

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

微分的 Sigmoid function 實作如下，於 Backpropagation 時會提到這樣實作的目的

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

**B. Neural network**

先帶出最基本的 Machine Learning 概念，圖 2.2 是一個 Machine Learning 的 Learning Algorithm。

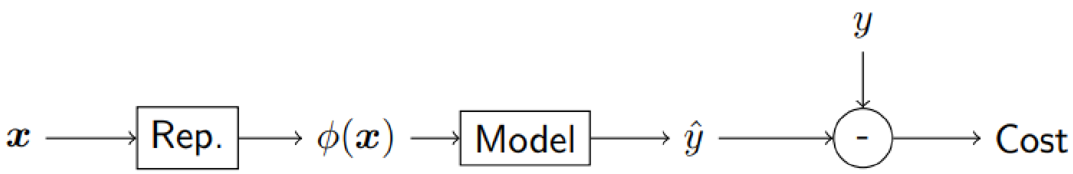



圖 2.2、Learning Algorithms

其中 $x$ 可以是任何數值、圖片甚至是音檔，透過 Representation 轉成 $\phi(x)$ 讓 Model 可以訓練的數據，而這個 Model 則是我們要訓練的模型，他會輸出一個預測的值 $\hat{y}$，要和實際地(ground-truth)值$y$做 loss function 求出 Cost，而目標則是降低 Cost 讓 Model 能預測的跟實際值相近。下方數學式子可以簡易的表示一個簡單的模型：

$$\hat{y} = f_\omega(\phi(x)) = \sigma(\omega^T \phi(x)) = \phi(x)^T \omega$$

可以看到輸入是$\phi(x)$而他會乘上一個權重矩陣$\omega$，透過修正$\omega$來達到降低$\hat{y}$與$y$的 loss。

然而一般的問題很難用一層的 Artificial Neural 解決，圖 2.3 展示了多層的複雜模型，可以看到要同時修正多個模型參數$\omega, \theta_n, \theta_{n-1}, \theta_{n-2}, \dots, \theta_1$。

$$f_{\boldsymbol{w},\boldsymbol{\theta_n},\boldsymbol{\theta_{n-1}},\cdots,\boldsymbol{\theta_1}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \underbrace{\phi_{\boldsymbol{\theta_n}}(\phi_{\boldsymbol{\theta_{n-1}}}(\phi_{\boldsymbol{\theta_{n-2}}}(\cdots\phi_{\boldsymbol{\theta_1}}(\boldsymbol{x})))))}_{\text{Hierarchy of concepts/features}}$$

圖 2.3、多層架構的 Model

常見的 Artificial Neural 設計如下圖 2.4 所示，包含多個權重$w$的和多個輸入資料$x$與一個偏量$b$和一個激活函數$\sigma$。

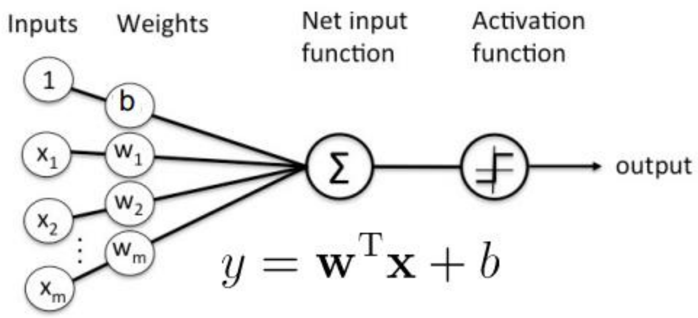



圖 2.4、Artificial Neural，此圖中$y$代表 進入激活函數前的數值

本章節要介紹 Neural Network，是由一個 Input、一個 Output 層，和多個 Hidden Layer 所組成的人工神經網路(Artificial Neural Network)，如圖 2.5 所示，每個 Artificial Neural 中都有權重和偏量要去訓練。

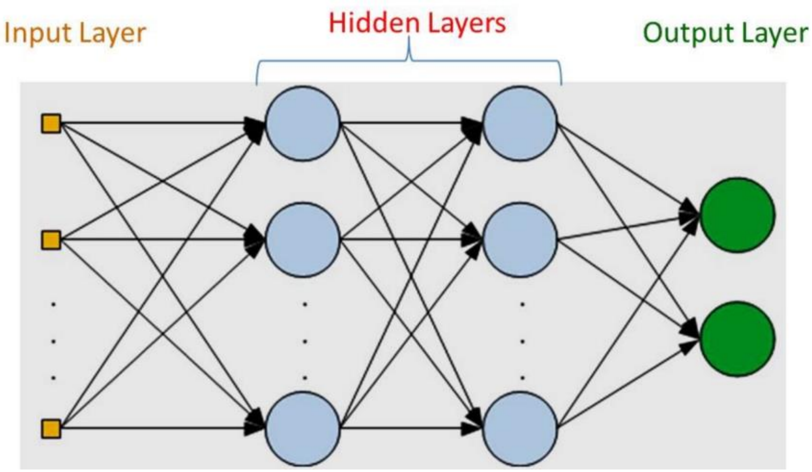



圖 2.5、Neural network，圖中每個圓形節點都是一個 Artificial Neural

在本實驗報告設計兩個 Class 來實作一個 Neural Network，分別設計 nn 與 model，nn 為一個獨立的 Artificial Neural，可以設定的參數有

- 輸入
- 輸出
- 激活函數
- 微分的激活函數
- 權重矩陣的優化器
- 偏移向量的優化器

如下方程式碼：

```
neural=nn(input,output,activate_function,derivative_function,
          weight_optimizer,bias_optimizer)
```

其中激活函數(Activation function)和優化器(Optimizer)如果不特別給值時，預設都是 None。

此 Class 包含了一層 Artificial Neural 和一個 Optimizer，如題目所要求

的 Two-layer neural network，的新增方式可以如下：(假設 2,4,4,1，sigmoid)

```
network = []
network.append(nn(2,4,sigmoid,derivative_sigmoid))
network.append(nn(4,4,sigmoid,derivative_sigmoid))
network.append(nn(4,1,sigmoid,derivative_sigmoid))
```

此外也實作的 Neural Network 所需的 forward、backward、update 功能，在 training 時步驟如下：

1. 根據現在的網絡，輸入的 data 進行前向傳播 (forward propagation)，每一層的輸出會作為下一層的輸入，依次類推。
2. nn 的實例中每一層 forward 包含輸入值$x$、權重$\omega$、偏量$b$、激活函數$\sigma$數學式子表示為：$y = \sigma(x\omega^T + b)$。
3. 最後一層通過損失函數(loss function)計算出損失值(loss)，並對整個模型進行反向傳播(back-propagation)，以計算出模型參數的梯度。
4. 計算所有要修正的梯度之後再來對整個模型參數做修正(update)。

具體的程式碼如下：

```
def forward(self,feature):
    # give feature with size input then result size output feature
    # linear function + activation function
    # y = σ(WX + b)
    self.a_input = feature
    self.a_output = self.activate_function(np.dot(feature,self.W) + self.b)
    return self.a_output
def backward(self,output_error):
    z = output_error * self.derivative_function(self.a_output) # calc this layer new error
    self.dW = np.dot(self.a_input.T , z)
    self.db = np.sum(z, axis=0, keepdims=True)
    return np.dot(z, self.W.T)
def update(self,learning_rate):
    if (self.weight_optimizer != None and self.bias_optimizer!= None ):
        self.W -= self.weight_optimizer.calc(learning_rate * self.dW)
        self.b -= self.bias_optimizer.calc(learning_rate * self.db)
    else:
        self.W -= learning_rate * self.dW
        self.b -= learning_rate * self.db
```

為了更好的控制我們的神經網路模型，設計了一個名為 model 的 Class，用來維護整個 Neural Network，並且實作 training 與 testing 的介面，減少直接對 list 裡面的 nn 實例做使用，並且也可以透過這個 model 的 Class 額外設計如設計優化器(Optimizer)、繪圖(learning curve)等功能。

## C. Backpropagation

Backpropagation 在執行完 forward 時做 backward 計算所有 neural 中的模型參數 weight 與 bias 的梯度，接著搭配 learning rate 對所有的模型參數做修正。在此會以一個簡單的網路做例子(2,2,2,1)，如圖 2.6 所示。

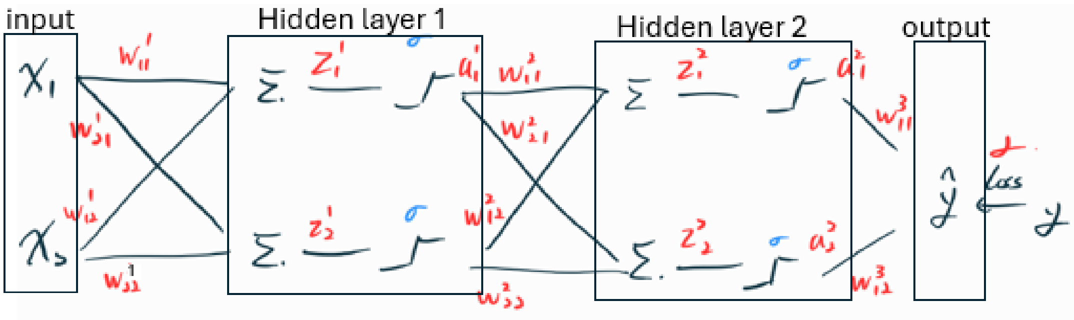



圖 2.6、擁有 2 個參數輸入與 1 個參數輸出和兩個 Hidden Layer 的 NN，其中沒有 bias 的偏量只有權重矩陣

已知微分可以求在某個曲線上的切線斜率，而透過這個求斜率的過程可以去找這個函數中的最大、最小值，但是是區域最大、區域最小值，而如果針對最終 Loss function 出來的 loss 做對要修正的權重做偏微分也能求出往區域最佳解的修正，如下方式子：

$$\theta := \theta - \eta \nabla L = \theta - \eta \frac{\partial L}{\partial \theta} \text{，} L: Loss\ , \eta: learning\ rate$$

其中$\theta$可以是模型中任何的要訓練的參數，如每個神經元的偏量$b$、矩陣權重$w$。有了這個概念之後就要一一對這些參數做偏微分求出各自的梯度，而在算梯度時會需要帶入到微積分的連鎖律，由圖 2.7 可知，計算$w_{11}^1$的偏微需要經過兩條路才能到達最終的 Loss：

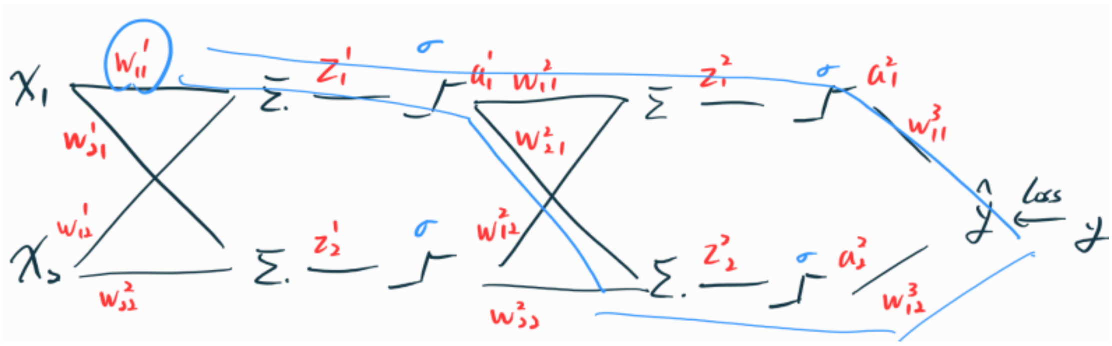


圖 2.7、計算$\frac{\partial L}{\partial w_{11}^1}$時的路徑

把他展開後得到：

$$\frac{\partial L}{\partial w_{11}^1}=\frac{\partial \mathrm{L}}{\partial z_1^3}\left[\sum_i^2\frac{\partial z_1^3}{\partial a_i^2}\frac{\partial a_i^2}{\partial z_i^2}\frac{\partial z_i^2}{\partial a_1^1}\right]\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_{11}^1}$$

其中中間的 Sigma 項可以解釋成 走上下兩條路，而$\frac{\partial \mathrm{L}}{\partial z_1^3}$則是 L 針對$z_1^3$的偏微分，已知 L 是一個 Loss function 計算出來的結果，以 MSE 為例的話則是$L=\frac{1}{n}(y-\hat{y})^2$，而在最後一曾並沒有設計一個 activation function 因此 $z_1^3=\hat{y}$，也就是可以將式子化作$\frac{\partial}{\partial\hat{y}}\frac{1}{n}(y-\hat{y})^2$ 得到 $-\frac{2}{n}(y-\hat{y})$，依此類推去推算。

不過由前面開始算太複雜了，如果由後往前看會發現他是有規律的，如圖 2.8 所示：

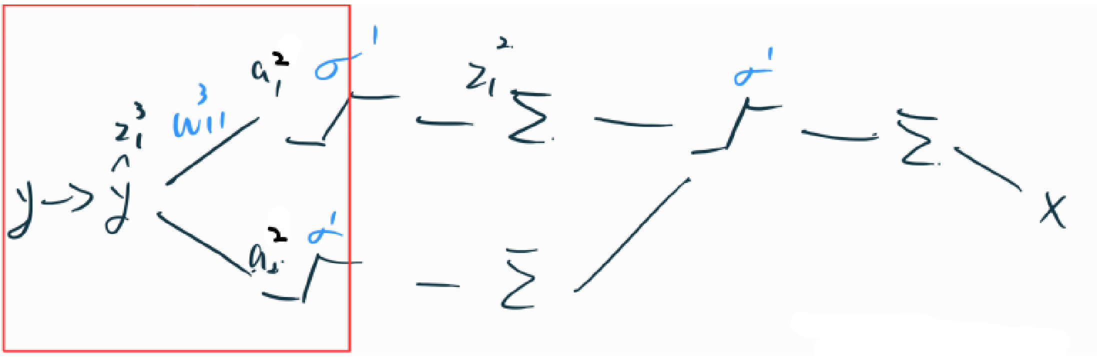



圖 2.8、由後往前推算 $w_{11}^3$

同樣帶出求修正率和梯度的算法：$\frac{\partial L}{\partial w_{11}^3}=\frac{\partial L}{\partial z_1^3}\frac{\partial z_1^3}{\partial w_{11}^3}$，上述提到 $\frac{\partial \mathrm{L}}{\partial z_1^3}$ 的算法，而$z_1^3=a_1^2\,w_{11}^3$，因此可以變成 $\frac{\partial L}{\partial w_{11}^3}=-\frac{2}{n}(y-\hat{y})\,a_1^2$，這就是對$w_{11}^3$的修改，那如果要繼續往下算的話，如圖 2.9 所示

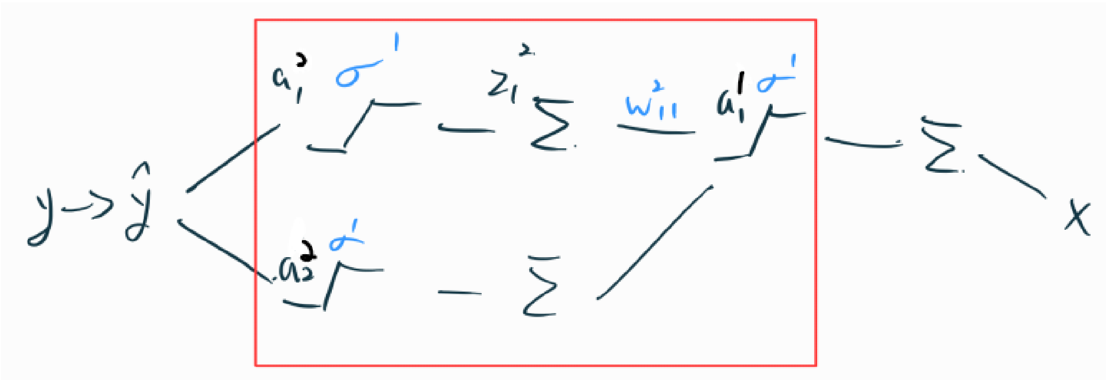



圖 2.9、由後往前推算 $w_{11}^2$

同樣的展開後得到 $\frac{\partial L}{\partial w_{11}^2}=\frac{\partial L}{\partial z_1^3}\frac{\partial z_1^3}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2}=-\frac{2}{n}(y-\hat{y})w_{11}^3\sigma'(z_1^2)a_1^1$會發現前面的值都一樣，但會有一個激活函數需要去求微分$\frac{\partial a_1^2}{\partial z_1^2}=\frac{\partial}{\partial z_1^2}\sigma(z_1^2)=\sigma'(z_1^2)$，後續推額外的參數都是利用相似的方式，接著回到第一層：

$$\frac{\partial L}{\partial w_{11}^1}=\frac{\partial \mathrm{L}}{\partial z_1^3}\left[\sum_i^2\frac{\partial z_1^3}{\partial a_i^2}\frac{\partial a_i^2}{\partial z_i^2}\frac{\partial z_i^2}{\partial a_1^1}\right]\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_{11}^1}$$

拆開來表示則是$\frac{\partial L}{\partial w_{11}^1}=-\frac{2}{n}(y-\hat{y})[\sum_i^2(w_{1i}^3)\sigma'(z_i^2)(w_{i1}^2)]\sigma'(z_1^1)x_1$，而在實作上

以矩陣乘法(np.dot)的方式可以直接由後往前算，在計算上還有一些小技巧，原先的$-\frac{2}{n}(y-\hat{y})$因為 loss function 挑選 MSE 的關係因此微分後會帶負號，所以在 update 值時會成加號，如下方式子：

$$\theta \coloneqq \theta + \eta \frac{\partial L}{\partial \theta}$$

此外在程式的實作上最外層的 2 這個參數可以省略，會被 learning rate $(\eta)$ 吸收，因此可以省略掉。

最後由上方的算式中求出 $\frac{\partial a_1^2}{\partial z_1^2} = \frac{\partial}{\partial z_1^2}\sigma(z_1^2) = \sigma'(z_1^2)$

已知$\sigma'(x) = \sigma(x)\,(1-\sigma(x))$ 並且 $\sigma(z_1^2) = a_1^2$，$\sigma'(z_1^2) = \sigma(z_1^2)(1-\sigma(z_1^2))$等同於$\sigma'(z_1^2) = a_1^2 \cdot (1-a_1^2)$ ，因此回到一開始 Sigmoid function 的微分設計，適用於此直接針對 上一層的輸入來算下一層輸出的微分。

回頭來看，微分的 Sigmoid function 與 backward 的 function，Python 程式設計如下

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

def backward(self,output_error):
    z = output_error * self.derivative_function(self.a_output) # calc this layer new error
    self.dW = np.dot(self.a_input.T , z)
    self.db = np.sum(z, axis=0, keepdims=True)
    return np.dot(z, self.W.T)
```

以$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial z_1^3}\frac{\partial z_1^3}{\partial a_1^2}\frac{\partial a_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial w_{11}^2} = -\frac{2}{n}(y-\hat{y})w_{11}^3\sigma'(z_1^2)a_1^1$為例

根據上一層計算完的結果：$-\frac{2}{n}(y-\hat{y})w_{11}^3$

乘上微分的 Sigmoid function：$\sigma'(z_1^2) = a_1^2 \cdot (1-a_1^2)$

再乘上原先的輸入：$a_1^1$

# 3. Results of testing

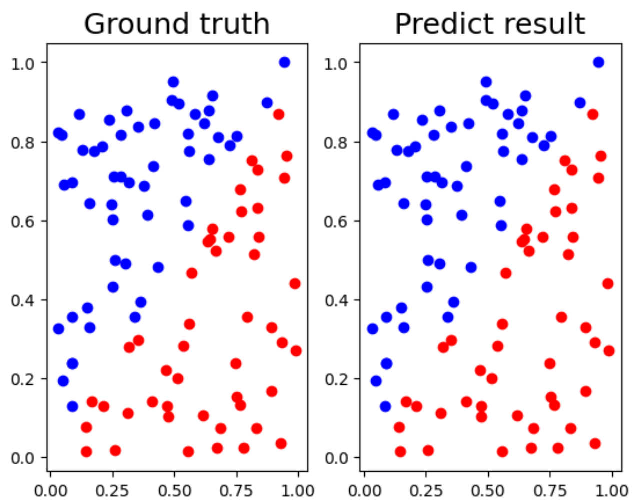
## A. Screenshot and comparison figure

*測試一*

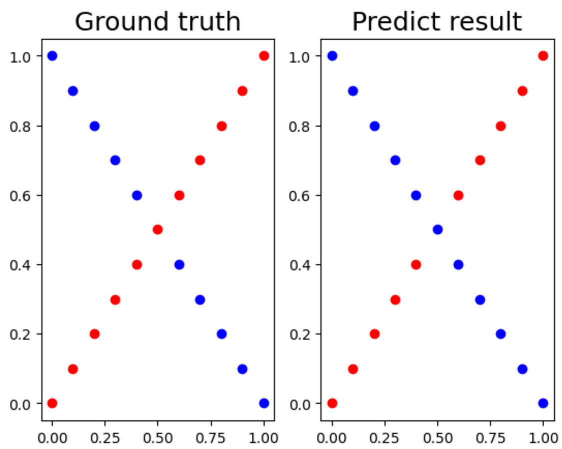
| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

Dataset: Linear
n: 100




```
epoch     1000: loss : 0.0042380260
epoch     2000: loss : 0.0238643824
epoch     3000: loss : 0.0061306842
epoch     4000: loss : 0.0033579711
epoch     5000: loss : 0.0014456249
epoch     6000: loss : 0.0003044721
epoch     7000: loss : 0.0001621299
epoch     8000: loss : 0.0001091968
epoch     9000: loss : 0.0000816992
epoch    10000: loss : 0.0000645244
```

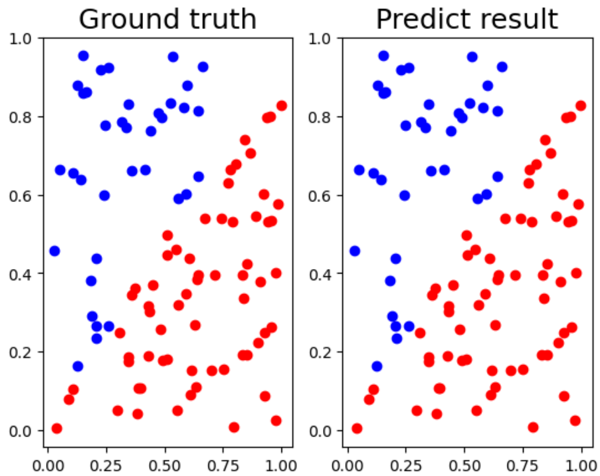
| | |
|---|---|
| Dataset:  easy_XOR<br>n:  21 | <br>epoch      1000: loss : 0.2484968851<br>epoch      2000: loss : 0.1993588533<br>epoch      3000: loss : 0.0471286351<br>epoch      4000: loss : 0.0438679142<br>epoch      5000: loss : 0.0435439540<br>epoch      6000: loss : 0.0434442117<br>epoch      7000: loss : 0.0433983630<br>epoch      8000: loss : 0.0433726142<br>epoch      9000: loss : 0.0433563199<br>epoch     10000: loss : 0.0433451606 |

*測試二*

| | |
|---|---|
| 訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=256<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | 訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=256)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

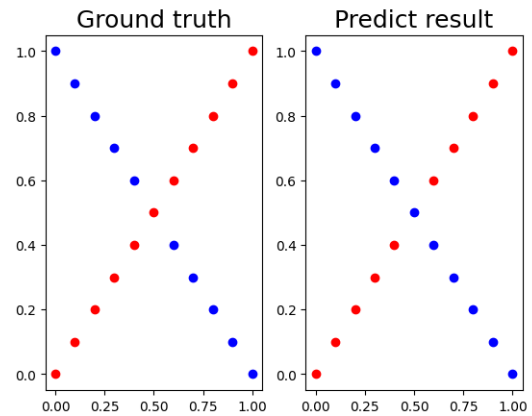
實驗結果

<table>
<tr>
<td>Dataset: Linear<br>n: 100</td>
<td>




```
epoch     1000: loss : 0.0165096744
epoch     2000: loss : 0.0133546540
epoch     3000: loss : 0.0120894287
epoch     4000: loss : 0.0098347713
epoch     5000: loss : 0.0030836749
epoch     6000: loss : 0.0041756704
epoch     7000: loss : 0.0099000349
epoch     8000: loss : 0.0003910581
epoch     9000: loss : 0.0001986959
epoch    10000: loss : 0.0001308822
```

</td>
</tr>
<tr>
<td>Dataset: easy_XOR<br>n: 21</td>
<td>




```
epoch     1000: loss : 0.2493228528
epoch     2000: loss : 0.2488068536
epoch     3000: loss : 0.2291101695
epoch     4000: loss : 0.0777789849
epoch     5000: loss : 0.0441914597
epoch     6000: loss : 0.0435886130
epoch     7000: loss : 0.0434538671
epoch     8000: loss : 0.0433994892
epoch     9000: loss : 0.0433710116
epoch    10000: loss : 0.0433537579
```

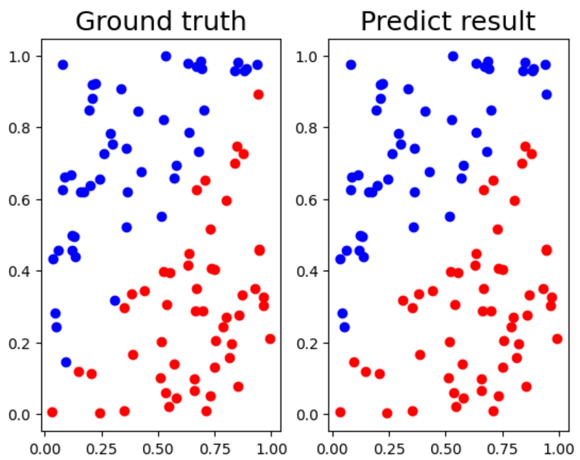
</td>
</tr>
</table>

*測試三*

<table>
<tr>
<td>
訓練的參數、架構為<br>
● 四層網路，兩層隱藏層，2-4-4-1<br>
● 激活函數皆為 Sigmoid<br>
● Optimizer: SGD ,batch=32<br>
● Learning rate: $\eta = 0.001$<br>
● Epoch = 10,000<br>
● Loss function: MSE<br>
● 每 20 場記錄一次 loss
</td>
<td>
訓練的程式<br>
m = model(memory_epoch=20)<br>
m.clear()<br>
m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>
m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>
m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>
m.training(10000,x,y,0.001,1000,batch=32)<br>
pred_y = m.testing(x,y,show_info=True)<br>
show_result(x,y,(pred_y > 0.5))<br>
m.show_learning_curve_loss()<br>
m.show_learning_curve_accuracy()
</td>
</tr>
</table>

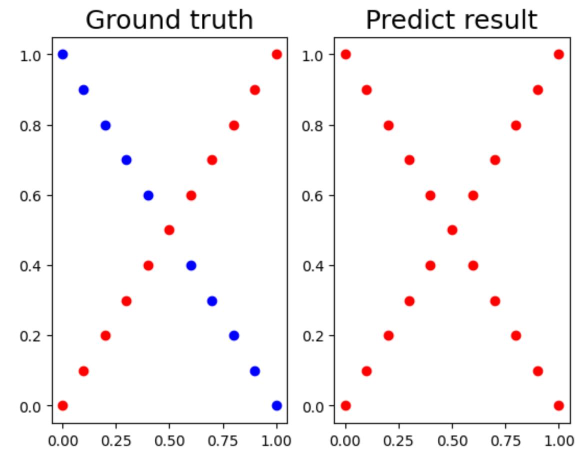
實驗結果

<table>
<tr>
<td>
Dataset: Linear<br>
n: 100
</td>
<td>




```
epoch    1000: loss : 0.2493466112
epoch    2000: loss : 0.2492566579
epoch    3000: loss : 0.2491205025
epoch    4000: loss : 0.2488982489
epoch    5000: loss : 0.2484926869
epoch    6000: loss : 0.2476104307
epoch    7000: loss : 0.2450217673
epoch    8000: loss : 0.2326173269
epoch    9000: loss : 0.1529540925
epoch   10000: loss : 0.0637364654
```

</td>
</tr>
</table>

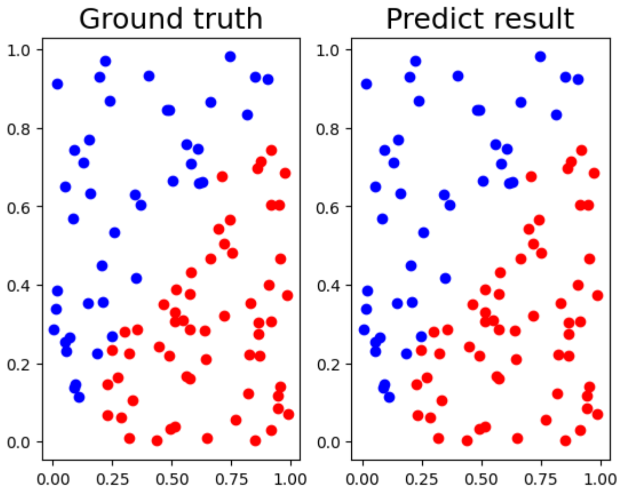
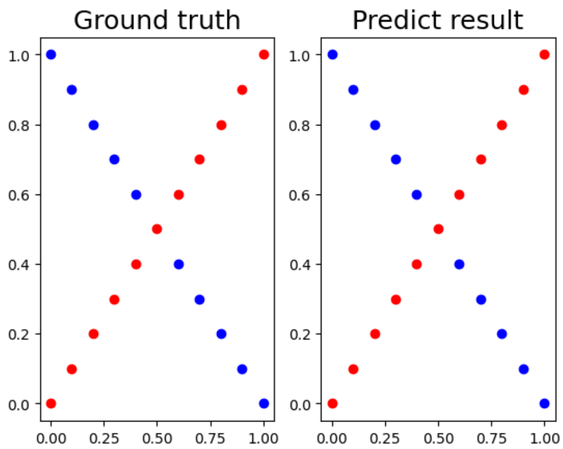
<table>
<tr>
<td>Dataset: easy_XOR<br>n: 21</td>
<td>



```
epoch     1000: loss : 0.2495185016
epoch     2000: loss : 0.2495149425
epoch     3000: loss : 0.2495114096
epoch     4000: loss : 0.2495079003
epoch     5000: loss : 0.2495044128
epoch     6000: loss : 0.2495009455
epoch     7000: loss : 0.2494974968
epoch     8000: loss : 0.2494940649
epoch     9000: loss : 0.2494906483
epoch    10000: loss : 0.2494872454
```
</td>
</tr>
</table>

*測試四*

<table>
<tr>
<td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-8-8-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td>
<td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,8,sigmoid,derivative_sigmoid)<br>m.add_layer(8,8,sigmoid,derivative_sigmoid)<br>m.add_layer(8,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td>
</tr>
</table>

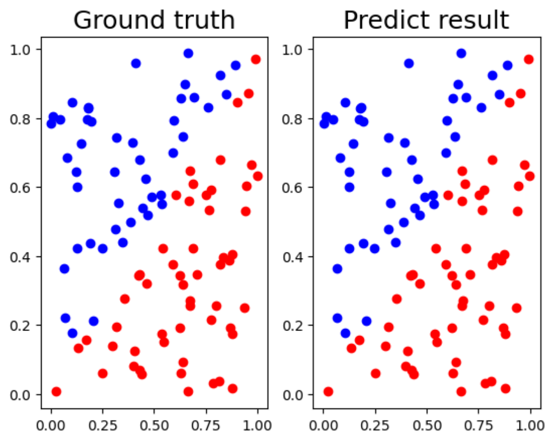
實驗結果

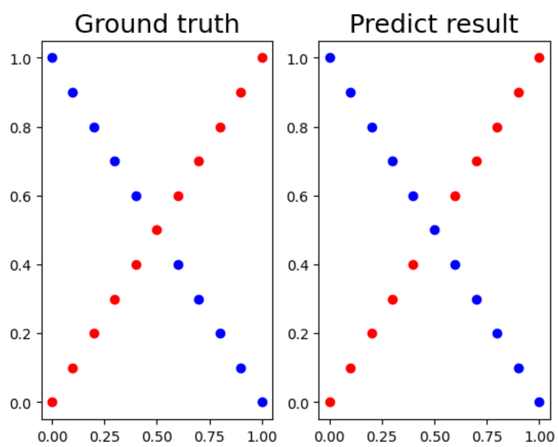
| | |
|---|---|
| Dataset: Linear<br>n: 100 | <br><br>epoch 1000: loss : 0.0075868540<br>epoch 2000: loss : 0.0009450248<br>epoch 3000: loss : 0.0001706738<br>epoch 4000: loss : 0.0000878456<br>epoch 5000: loss : 0.0000589636<br>epoch 6000: loss : 0.0000445133<br>epoch 7000: loss : 0.0000357681<br>epoch 8000: loss : 0.0000298923<br>epoch 9000: loss : 0.0000256569<br>epoch 10000: loss : 0.0000224584 |
| Dataset: easy_XOR<br>n: 21 | <br><br>epoch 1000: loss : 0.2500396203<br>epoch 2000: loss : 0.1810731834<br>epoch 3000: loss : 0.0176151838<br>epoch 4000: loss : 0.0020716506<br>epoch 5000: loss : 0.0007890739<br>epoch 6000: loss : 0.0004377849<br>epoch 7000: loss : 0.0002896388<br>epoch 8000: loss : 0.0002114016<br>epoch 9000: loss : 0.0001641316<br>epoch 10000: loss : 0.0001329178 |

*測試五*

<table>
<tr>
<td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數改成兩個隱藏層 ReLU、輸出層 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td>
<td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,relu,derivative_relu)<br>m.add_layer(4,4,relu,derivative_relu)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td>
</tr>
</table>

實驗結果

<table>
<tr>
<td>Dataset: Linear<br>n: 100</td>
<td>

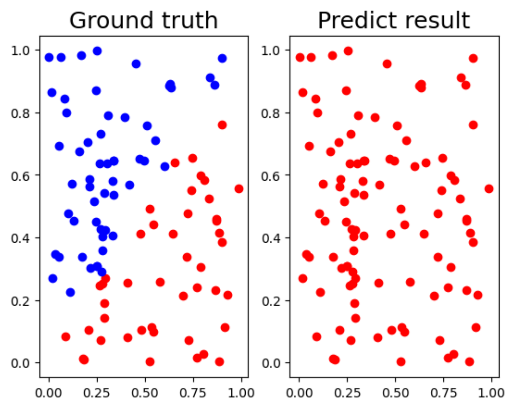
<br>epoch 1000: loss : 0.0000504649<br>epoch 2000: loss : 0.0000125865<br>epoch 3000: loss : 0.0000068719<br>epoch 4000: loss : 0.0000046453<br>epoch 5000: loss : 0.0000034771<br>epoch 6000: loss : 0.0000027638<br>epoch 7000: loss : 0.0000022851<br>epoch 8000: loss : 0.0000019424<br>epoch 9000: loss : 0.0000016866<br>epoch 10000: loss : 0.0000014866</td>
</tr>
<tr>
<td>Dataset: easy_XOR<br>n: 21</td>
<td>
</td>
</tr>
</table>

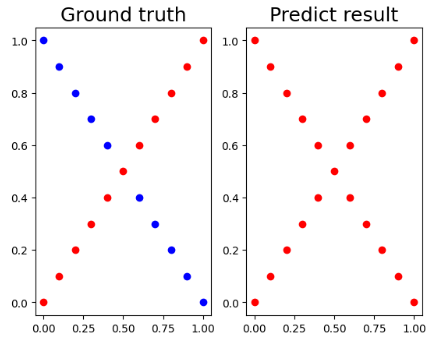
| | |
|---|---|
| | epoch 1000: loss : 0.0433059546<br>epoch 2000: loss : 0.0432979897<br>epoch 3000: loss : 0.0432945570<br>epoch 4000: loss : 0.0432944942<br>epoch 5000: loss : 0.0432930391<br>epoch 6000: loss : 0.0432927944<br>epoch 7000: loss : 0.0432951611<br>epoch 8000: loss : 0.0432922964<br>epoch 9000: loss : 0.0432921912<br>epoch 10000: loss : 0.0432917886 |

*測試六*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 無激活函數<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4)<br>m.add_layer(4,4)<br>m.add_layer(4,1)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

| | |
|---|---|
| Dataset: Linear<br>n: 100 | <br>epoch 1000: loss : nan<br>epoch 2000: loss : nan<br>epoch 3000: loss : nan<br>epoch 4000: loss : nan<br>epoch 5000: loss : nan<br>epoch 6000: loss : nan<br>epoch 7000: loss : nan<br>epoch 8000: loss : nan<br>epoch 9000: loss : nan<br>epoch 10000: loss : nan |

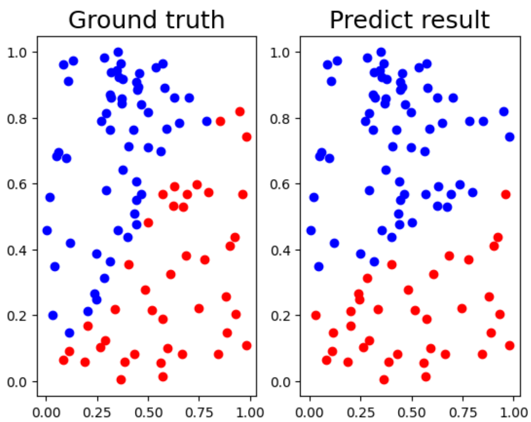
<table>
<tr>
<td>Dataset:  easy_XOR<br>n:  21</td>
<td>

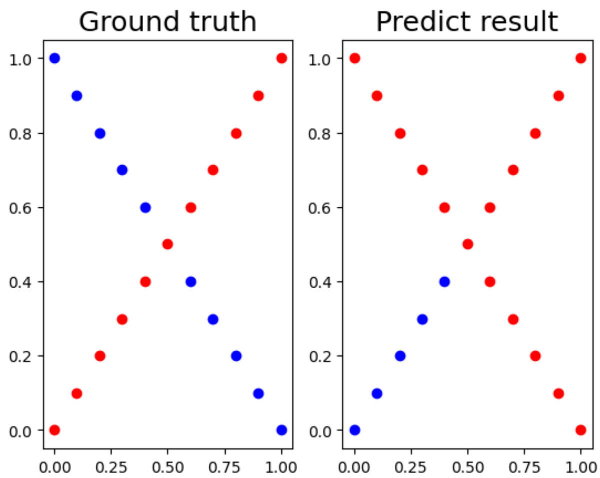
<pre>
epoch     1000: loss : nan
epoch     2000: loss : nan
epoch     3000: loss : nan
epoch     4000: loss : nan
epoch     5000: loss : nan
epoch     6000: loss : nan
epoch     7000: loss : nan
epoch     8000: loss : nan
epoch     9000: loss : nan
epoch    10000: loss : nan
</pre></td>
</tr>
</table>

*測試七*

<table>
<tr>
<td>訓練的參數、架構為<br>
● 四層網路，兩層隱藏層，2-4-4-1<br>
● 無激活函數<br>
● Optimizer: SGD ,batch=32<br>
● Learning rate: $\eta = 0.00001$<br>
● Epoch = 10,000<br>
● Loss function: MSE<br>
● 每 20 場記錄一次 loss</td>
<td>訓練的程式<br>
m = model(memory_epoch=20)<br>
m.clear()<br>
m.add_layer(2,4)<br>
m.add_layer(4,4)<br>
m.add_layer(4,1)<br>
m.training(10000,x,y,0.00001,1000,batch=32)<br>
pred_y = m.testing(x,y,show_info=True)<br>
show_result(x,y,(pred_y > 0.5))<br>
m.show_learning_curve_loss()<br>
m.show_learning_curve_accuracy()</td>
</tr>
</table>

實驗結果

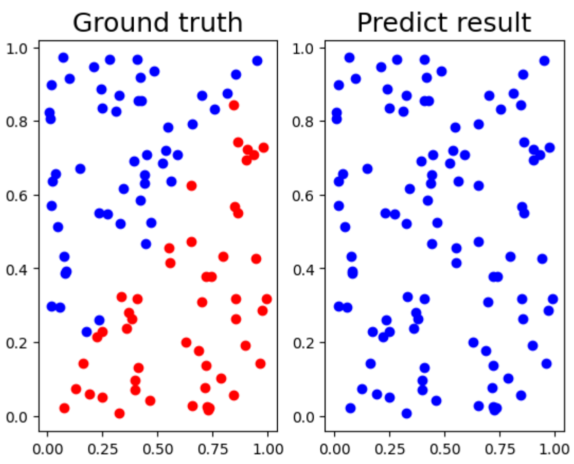
<table>
<tr>
<td>Dataset:  Linear<br>n:  100</td>
<td>
</td>
</tr>
</table>

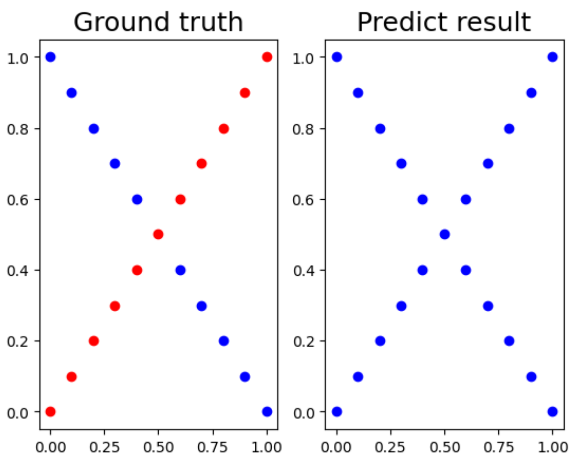
<table>
<tr>
<td></td>
<td>epoch 1000: loss : 0.2155863554<br>epoch 2000: loss : 0.2089800308<br>epoch 3000: loss : 0.2020457350<br>epoch 4000: loss : 0.1942884402<br>epoch 5000: loss : 0.1851128308<br>epoch 6000: loss : 0.1738070957<br>epoch 7000: loss : 0.1596788711<br>epoch 8000: loss : 0.1428130919<br>epoch 9000: loss : 0.1249810990<br>epoch 10000: loss : 0.1095959436</td>
</tr>
<tr>
<td>Dataset: easy_XOR<br>n: 21</td>
<td>
<br>epoch 1000: loss : 0.2545789661<br>epoch 2000: loss : 0.2543662980<br>epoch 3000: loss : 0.2541879990<br>epoch 4000: loss : 0.2540170222<br>epoch 5000: loss : 0.2538529250<br>epoch 6000: loss : 0.2536954212<br>epoch 7000: loss : 0.2535442403<br>epoch 8000: loss : 0.2533991249<br>epoch 9000: loss : 0.2532598299<br>epoch 10000: loss : 0.2531261220</td>
</tr>
</table>

*測試八*

<table>
<tr>
<td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-40-40-1<br>● All Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td>
<td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td>
</tr>
</table>

實驗結果

<table>
<tr>
<td>Dataset: Linear<br>n: 100</td>
<td>


<pre>
epoch     1000: loss : 0.4999999631
epoch     2000: loss : 0.4999999625
epoch     3000: loss : 0.4999999619
epoch     4000: loss : 0.4999999613
epoch     5000: loss : 0.4999999607
epoch     6000: loss : 0.4999999600
epoch     7000: loss : 0.4999999594
epoch     8000: loss : 0.4999999587
epoch     9000: loss : 0.4999999580
epoch    10000: loss : 0.4999999572
</pre>
</td>
</tr>
<tr>
<td>Dataset: easy_XOR<br>n: 21</td>
<td>


<pre>
epoch     1000: loss : 0.5238095219
epoch     2000: loss : 0.5238095219
epoch     3000: loss : 0.5238095219
epoch     4000: loss : 0.5238095219
epoch     5000: loss : 0.5238095219
epoch     6000: loss : 0.5238095219
epoch     7000: loss : 0.5238095219
epoch     8000: loss : 0.5238095219
epoch     9000: loss : 0.5238095219
epoch    10000: loss : 0.5238095219
</pre>
</td>
</tr>
</table>

## B. Show the accuracy of your prediction

*測試一*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

Dataset: Linear

n: 100

```
Iter  71 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  72 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  73 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  74 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  75 |    Ground truth: 1.000000 |    prediction: 0.999736 |
Iter  76 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  77 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  78 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  79 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  80 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  81 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter  82 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  83 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  84 |    Ground truth: 0.000000 |    prediction: 0.000006 |
Iter  85 |    Ground truth: 0.000000 |    prediction: 0.051820 |
Iter  86 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  87 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  88 |    Ground truth: 1.000000 |    prediction: 0.999996 |
Iter  89 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  90 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  91 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  92 |    Ground truth: 1.000000 |    prediction: 0.999997 |
Iter  93 |    Ground truth: 0.000000 |    prediction: 0.000022 |
Iter  94 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  95 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  96 |    Ground truth: 0.000000 |    prediction: 0.000005 |
Iter  97 |    Ground truth: 0.000000 |    prediction: 0.000012 |
Iter  98 |    Ground truth: 1.000000 |    prediction: 0.999998 |
Iter  99 |    Ground truth: 0.000000 |    prediction: 0.000005 |
loss=0.0001 accuracy=100.00%
```

Dataset: easy_XOR
n: 21

```
Iter   0 |    Ground truth: 0.000000 |    prediction: 0.002705 |
Iter   1 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter   2 |    Ground truth: 0.000000 |    prediction: 0.002160 |
Iter   3 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter   4 |    Ground truth: 0.000000 |    prediction: 0.001547 |
Iter   5 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter   6 |    Ground truth: 0.000000 |    prediction: 0.001382 |
Iter   7 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter   8 |    Ground truth: 0.000000 |    prediction: 0.022396 |
Iter   9 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter  10 |    Ground truth: 0.000000 |    prediction: 0.908012 |
Iter  11 |    Ground truth: 0.000000 |    prediction: 0.024985 |
Iter  12 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter  13 |    Ground truth: 0.000000 |    prediction: 0.001104 |
Iter  14 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter  15 |    Ground truth: 0.000000 |    prediction: 0.000712 |
Iter  16 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter  17 |    Ground truth: 0.000000 |    prediction: 0.000648 |
Iter  18 |    Ground truth: 1.000000 |    prediction: 0.908012 |
Iter  19 |    Ground truth: 0.000000 |    prediction: 0.000629 |
Iter  20 |    Ground truth: 1.000000 |    prediction: 0.908012 |
loss=0.0433 accuracy=95.24%
```

*測試二*

訓練的參數、架構為

- 四層網路，兩層隱藏層，2-4-4-1
- 激活函數皆為 Sigmoid
- Optimizer: SGD ,batch=256
- Learning rate: $\eta = 0.1$
- Epoch = 10,000
- Loss function: MSE
- 每 20 場記錄一次 loss

訓練的程式

```
m = model(memory_epoch=20)
m.clear()
m.add_layer(2,4,sigmoid,derivative_sigmoid)
m.add_layer(4,4,sigmoid,derivative_sigmoid)
m.add_layer(4,1,sigmoid,derivative_sigmoid)
m.training(10000,x,y,0.1,1000,batch=256)
pred_y = m.testing(x,y,show_info=True)
show_result(x,y,(pred_y > 0.5))
m.show_learning_curve_loss()
m.show_learning_curve_accuracy()
```

實驗結果

Dataset: Linear
n: 100

```
Iter   0 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter   1 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter   2 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter   3 |    Ground truth: 1.000000 |    prediction: 0.999923 |
Iter   4 |    Ground truth: 0.000000 |    prediction: 0.000053 |
Iter   5 |    Ground truth: 1.000000 |    prediction: 0.992149 |
Iter   6 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter   7 |    Ground truth: 1.000000 |    prediction: 0.999987 |
Iter   8 |    Ground truth: 1.000000 |    prediction: 0.999987 |
Iter   9 |    Ground truth: 1.000000 |    prediction: 0.999987 |
Iter  10 |    Ground truth: 1.000000 |    prediction: 0.999987 |
Iter  11 |    Ground truth: 0.000000 |    prediction: 0.000007 |
...
Iter  97 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter  98 |    Ground truth: 0.000000 |    prediction: 0.000007 |
Iter  99 |    Ground truth: 1.000000 |    prediction: 0.999987 |
loss=0.0001 accuracy=100.00%
```

<table>
<tr><td>Dataset: easy_XOR<br>n: 21</td><td>

```
Iter   0 |    Ground truth: 0.000000 |   prediction: 0.000681 |
Iter   1 |    Ground truth: 1.000000 |   prediction: 0.907954 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 0.000641 |
Iter   3 |    Ground truth: 1.000000 |   prediction: 0.907955 |
Iter   4 |    Ground truth: 0.000000 |   prediction: 0.000611 |
Iter   5 |    Ground truth: 1.000000 |   prediction: 0.907955 |
Iter   6 |    Ground truth: 0.000000 |   prediction: 0.000766 |
Iter   7 |    Ground truth: 1.000000 |   prediction: 0.907955 |
Iter   8 |    Ground truth: 0.000000 |   prediction: 0.024865 |
Iter   9 |    Ground truth: 1.000000 |   prediction: 0.907956 |
Iter  10 |    Ground truth: 0.000000 |   prediction: 0.907956 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 0.026515 |
Iter  12 |    Ground truth: 1.000000 |   prediction: 0.907956 |
Iter  13 |    Ground truth: 0.000000 |   prediction: 0.000629 |
Iter  14 |    Ground truth: 1.000000 |   prediction: 0.907957 |
Iter  15 |    Ground truth: 0.000000 |   prediction: 0.000360 |
Iter  16 |    Ground truth: 1.000000 |   prediction: 0.907957 |
Iter  17 |    Ground truth: 0.000000 |   prediction: 0.000317 |
Iter  18 |    Ground truth: 1.000000 |   prediction: 0.907957 |
Iter  19 |    Ground truth: 0.000000 |   prediction: 0.000303 |
Iter  20 |    Ground truth: 1.000000 |   prediction: 0.907958 |
loss=0.0434 accuracy=95.24%
```

</td></tr>
</table>

*測試三*

<table>
<tr><td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.001$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td><td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.001,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td></tr>
</table>

實驗結果

<table>
<tr><td>Dataset: Linear<br>n: 100</td><td>

```
Iter   0 |    Ground truth: 1.000000 |   prediction: 0.448613 |
Iter   1 |    Ground truth: 0.000000 |   prediction: 0.129603 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 0.152898 |
Iter   3 |    Ground truth: 1.000000 |   prediction: 0.804449 |
Iter   4 |    Ground truth: 1.000000 |   prediction: 0.707990 |
Iter   5 |    Ground truth: 0.000000 |   prediction: 0.360320 |
Iter   6 |    Ground truth: 0.000000 |   prediction: 0.235400 |
Iter   7 |    Ground truth: 1.000000 |   prediction: 0.799039 |
Iter   8 |    Ground truth: 0.000000 |   prediction: 0.145608 |
Iter   9 |    Ground truth: 0.000000 |   prediction: 0.200941 |
Iter  10 |    Ground truth: 0.000000 |   prediction: 0.135244 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 0.337128 |
...
Iter  97 |    Ground truth: 1.000000 |   prediction: 0.863003 |
Iter  98 |    Ground truth: 0.000000 |   prediction: 0.346247 |
Iter  99 |    Ground truth: 1.000000 |   prediction: 0.840305 |
loss=0.0637 accuracy=97.00%
```

</td></tr>
</table>

Dataset: easy_XOR

n: 21

```
Iter   0 |     Ground truth: 0.000000 |   prediction: 0.478059 |
Iter   1 |     Ground truth: 1.000000 |   prediction: 0.476940 |
Iter   2 |     Ground truth: 0.000000 |   prediction: 0.477729 |
Iter   3 |     Ground truth: 1.000000 |   prediction: 0.476881 |
Iter   4 |     Ground truth: 0.000000 |   prediction: 0.477424 |
Iter   5 |     Ground truth: 1.000000 |   prediction: 0.476824 |
Iter   6 |     Ground truth: 0.000000 |   prediction: 0.477145 |
Iter   7 |     Ground truth: 1.000000 |   prediction: 0.476768 |
Iter   8 |     Ground truth: 0.000000 |   prediction: 0.476891 |
Iter   9 |     Ground truth: 1.000000 |   prediction: 0.476715 |
Iter  10 |     Ground truth: 0.000000 |   prediction: 0.476663 |
Iter  11 |     Ground truth: 0.000000 |   prediction: 0.476458 |
...
Iter  18 |     Ground truth: 1.000000 |   prediction: 0.476474 |
Iter  19 |     Ground truth: 0.000000 |   prediction: 0.475848 |
Iter  20 |     Ground truth: 1.000000 |   prediction: 0.476431 |
loss=0.2495 accuracy=52.38%
```

*測試四*

訓練的參數、架構為

- 四層網路，兩層隱藏層，2-8-8-1
- 激活函數皆為 Sigmoid
- Optimizer: SGD ,batch=32
- Learning rate: $\eta = 0.1$
- Epoch = 10,000
- Loss function: MSE
- 每 20 場記錄一次 loss

訓練的程式

```
m = model(memory_epoch=20)
m.clear()
m.add_layer(2,8,sigmoid,derivative_sigmoid)
m.add_layer(8,8,sigmoid,derivative_sigmoid)
m.add_layer(8,1,sigmoid,derivative_sigmoid)
m.training(10000,x,y,0.1,1000,batch=32)
pred_y = m.testing(x,y,show_info=True)
show_result(x,y,(pred_y > 0.5))
m.show_learning_curve_loss()
m.show_learning_curve_accuracy()
```

實驗結果

Dataset: Linear

n: 100

```
Iter   0 |     Ground truth: 1.000000 |   prediction: 0.999991 |
Iter   1 |     Ground truth: 1.000000 |   prediction: 0.999942 |
Iter   2 |     Ground truth: 0.000000 |   prediction: 0.000007 |
Iter   3 |     Ground truth: 1.000000 |   prediction: 0.999996 |
Iter   4 |     Ground truth: 1.000000 |   prediction: 0.999997 |
Iter   5 |     Ground truth: 0.000000 |   prediction: 0.000006 |
Iter   6 |     Ground truth: 1.000000 |   prediction: 0.999996 |
Iter   7 |     Ground truth: 0.000000 |   prediction: 0.000004 |
Iter   8 |     Ground truth: 1.000000 |   prediction: 0.999995 |
Iter   9 |     Ground truth: 0.000000 |   prediction: 0.000005 |
Iter  10 |     Ground truth: 1.000000 |   prediction: 0.999988 |
Iter  11 |     Ground truth: 0.000000 |   prediction: 0.000004 |
...
Iter  97 |     Ground truth: 0.000000 |   prediction: 0.000005 |
Iter  98 |     Ground truth: 0.000000 |   prediction: 0.000005 |
Iter  99 |     Ground truth: 0.000000 |   prediction: 0.000049 |
loss=0.0000 accuracy=100.00%
```

Dataset: easy_XOR
n: 21

```
Iter  0 |   Ground truth: 0.000000 |   prediction: 0.000200 |
Iter  1 |   Ground truth: 1.000000 |   prediction: 0.999950 |
Iter  2 |   Ground truth: 0.000000 |   prediction: 0.000169 |
Iter  3 |   Ground truth: 1.000000 |   prediction: 0.999942 |
Iter  4 |   Ground truth: 0.000000 |   prediction: 0.000181 |
Iter  5 |   Ground truth: 1.000000 |   prediction: 0.999925 |
Iter  6 |   Ground truth: 0.000000 |   prediction: 0.000380 |
Iter  7 |   Ground truth: 1.000000 |   prediction: 0.999813 |
Iter  8 |   Ground truth: 0.000000 |   prediction: 0.006881 |
Iter  9 |   Ground truth: 1.000000 |   prediction: 0.972305 |
Iter 10 |   Ground truth: 0.000000 |   prediction: 0.036735 |
Iter 11 |   Ground truth: 0.000000 |   prediction: 0.007042 |
...
Iter 18 |   Ground truth: 1.000000 |   prediction: 0.999906 |
Iter 19 |   Ground truth: 0.000000 |   prediction: 0.000089 |
Iter 20 |   Ground truth: 1.000000 |   prediction: 0.999832 |
loss=0.0001 accuracy=100.00%
```

*測試五*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數改成兩個隱藏層 ReLU、輸出層 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,relu,derivative_relu)<br>m.add_layer(4,4,relu,derivative_relu)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

Dataset: Linear
n: 100

```
Iter  0 |   Ground truth: 1.000000 |   prediction: 0.999400 |
Iter  1 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter  2 |   Ground truth: 1.000000 |   prediction: 0.999400 |
Iter  3 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter  4 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter  5 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter  6 |   Ground truth: 1.000000 |   prediction: 0.999400 |
Iter  7 |   Ground truth: 1.000000 |   prediction: 0.999400 |
Iter  8 |   Ground truth: 1.000000 |   prediction: 0.999400 |
Iter  9 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter 10 |   Ground truth: 0.000000 |   prediction: 0.004252 |
Iter 11 |   Ground truth: 0.000000 |   prediction: 0.000000 |
...
Iter 97 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter 98 |   Ground truth: 0.000000 |   prediction: 0.000000 |
Iter 99 |   Ground truth: 0.000000 |   prediction: 0.005139 |
loss=0.0000 accuracy=100.00%
```

Dataset: easy_XOR

n: 21

```
Iter   0 |    Ground truth: 0.000000 |   prediction: 0.000919 |
Iter   1 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 0.000919 |
Iter   3 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter   4 |    Ground truth: 0.000000 |   prediction: 0.000919 |
Iter   5 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter   6 |    Ground truth: 0.000000 |   prediction: 0.000919 |
Iter   7 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter   8 |    Ground truth: 0.000000 |   prediction: 0.004363 |
Iter   9 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter  10 |    Ground truth: 0.000000 |   prediction: 0.909045 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 0.003771 |
...
Iter  18 |    Ground truth: 1.000000 |   prediction: 0.909045 |
Iter  19 |    Ground truth: 0.000000 |   prediction: 0.000000 |
Iter  20 |    Ground truth: 1.000000 |   prediction: 0.909046 |
loss=0.0433 accuracy=95.24%
```

*測試六*

訓練的參數、架構為

- 四層網路，兩層隱藏層，2-4-4-1
- 無激活函數
- Optimizer: SGD ,batch=32
- Learning rate: $\eta = 0.1$
- Epoch = 10,000
- Loss function: MSE
- 每 20 場記錄一次 loss

訓練的程式

```
m = model(memory_epoch=20)
m.clear()
m.add_layer(2,4)
m.add_layer(4,4)
m.add_layer(4,1)
m.training(10000,x,y,0.1,1000,batch=32)
pred_y = m.testing(x,y,show_info=True)
show_result(x,y,(pred_y > 0.5))
m.show_learning_curve_loss()
m.show_learning_curve_accuracy()
```

實驗結果

Dataset: Linear

n: 100

```
Iter   0 |    Ground truth: 0.000000 |   prediction: nan |
Iter   1 |    Ground truth: 1.000000 |   prediction: nan |
Iter   2 |    Ground truth: 0.000000 |   prediction: nan |
Iter   3 |    Ground truth: 0.000000 |   prediction: nan |
Iter   4 |    Ground truth: 0.000000 |   prediction: nan |
Iter   5 |    Ground truth: 1.000000 |   prediction: nan |
Iter   6 |    Ground truth: 1.000000 |   prediction: nan |
Iter   7 |    Ground truth: 1.000000 |   prediction: nan |
Iter   8 |    Ground truth: 1.000000 |   prediction: nan |
Iter   9 |    Ground truth: 0.000000 |   prediction: nan |
Iter  10 |    Ground truth: 1.000000 |   prediction: nan |
Iter  11 |    Ground truth: 0.000000 |   prediction: nan |
Iter  12 |    Ground truth: 1.000000 |   prediction: nan |
...
Iter  97 |    Ground truth: 1.000000 |   prediction: nan |
Iter  98 |    Ground truth: 1.000000 |   prediction: nan |
Iter  99 |    Ground truth: 0.000000 |   prediction: nan |
loss=nan accuracy=45.00%
```

Dataset: easy_XOR

n: 21

```
Iter   0 |    Ground truth: 0.000000 |   prediction: nan |
Iter   1 |    Ground truth: 1.000000 |   prediction: nan |
Iter   2 |    Ground truth: 0.000000 |   prediction: nan |
Iter   3 |    Ground truth: 1.000000 |   prediction: nan |
Iter   4 |    Ground truth: 0.000000 |   prediction: nan |
Iter   5 |    Ground truth: 1.000000 |   prediction: nan |
Iter   6 |    Ground truth: 0.000000 |   prediction: nan |
Iter   7 |    Ground truth: 1.000000 |   prediction: nan |
Iter   8 |    Ground truth: 0.000000 |   prediction: nan |
Iter   9 |    Ground truth: 1.000000 |   prediction: nan |
Iter  10 |    Ground truth: 0.000000 |   prediction: nan |
Iter  11 |    Ground truth: 0.000000 |   prediction: nan |
...
Iter  18 |    Ground truth: 1.000000 |   prediction: nan |
Iter  19 |    Ground truth: 0.000000 |   prediction: nan |
Iter  20 |    Ground truth: 1.000000 |   prediction: nan |
loss=nan accuracy=52.38%
```

*測試七*

訓練的參數、架構為

- 四層網路，兩層隱藏層，2-4-4-1
- 無激活函數
- Optimizer: SGD ,batch=32
- Learning rate: $\eta = 0.00001$
- Epoch = 10,000
- Loss function: MSE
- 每 20 場記錄一次 loss

訓練的程式

```
m = model(memory_epoch=20)
m.clear()
m.add_layer(2,4)
m.add_layer(4,4)
m.add_layer(4,1)
m.training(10000,x,y,0.00001,1000,batch=32)
pred_y = m.testing(x,y,show_info=True)
show_result(x,y,(pred_y > 0.5))
m.show_learning_curve_loss()
m.show_learning_curve_accuracy()
```

實驗結果

Dataset: Linear

n: 100

```
Iter   0 |    Ground truth: 0.000000 |   prediction: 0.543818 |
Iter   1 |    Ground truth: 0.000000 |   prediction: 0.132900 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 0.041377 |
Iter   3 |    Ground truth: 0.000000 |   prediction: 0.141744 |
Iter   4 |    Ground truth: 1.000000 |   prediction: 0.747050 |
Iter   5 |    Ground truth: 1.000000 |   prediction: 0.888372 |
Iter   6 |    Ground truth: 1.000000 |   prediction: 0.708583 |
Iter   7 |    Ground truth: 0.000000 |   prediction: 0.343214 |
Iter   8 |    Ground truth: 1.000000 |   prediction: 0.477823 |
Iter   9 |    Ground truth: 1.000000 |   prediction: 0.677741 |
Iter  10 |    Ground truth: 0.000000 |   prediction: 0.371764 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 0.261242 |
...
Iter  97 |    Ground truth: 1.000000 |   prediction: 0.927617 |
Iter  98 |    Ground truth: 1.000000 |   prediction: 0.626567 |
Iter  99 |    Ground truth: 0.000000 |   prediction: 0.173710 |
loss=0.1096 accuracy=83.00%
```

<table>
<tr><td>Dataset: easy_XOR<br>n: 21</td><td><pre>
Iter   0 |    Ground truth: 0.000000 |   prediction: 0.619733 |
Iter   1 |    Ground truth: 1.000000 |   prediction: 0.483830 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 0.594077 |
Iter   3 |    Ground truth: 1.000000 |   prediction: 0.485354 |
Iter   4 |    Ground truth: 0.000000 |   prediction: 0.568420 |
Iter   5 |    Ground truth: 1.000000 |   prediction: 0.486879 |
Iter   6 |    Ground truth: 0.000000 |   prediction: 0.542764 |
Iter   7 |    Ground truth: 1.000000 |   prediction: 0.488403 |
Iter   8 |    Ground truth: 0.000000 |   prediction: 0.517108 |
Iter   9 |    Ground truth: 1.000000 |   prediction: 0.489927 |
Iter  10 |    Ground truth: 0.000000 |   prediction: 0.491452 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 0.465796 |
...
Iter  18 |    Ground truth: 1.000000 |   prediction: 0.497549 |
Iter  19 |    Ground truth: 0.000000 |   prediction: 0.363171 |
Iter  20 |    Ground truth: 1.000000 |   prediction: 0.499074 |
loss=0.2531 accuracy=28.57%
</pre></td></tr>
</table>

*測試八*

<table>
<tr><td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-40-40-1<br>● All Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td><td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td></tr>
</table>

實驗結果

<table>
<tr><td>Dataset: Linear<br>n: 100</td><td><pre>
Iter   0 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   1 |    Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   2 |    Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   3 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   4 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   5 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   6 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   7 |    Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   8 |    Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   9 |    Ground truth: 0.000000 |   prediction: 1.000000 |
Iter  10 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter  11 |    Ground truth: 0.000000 |   prediction: 1.000000 |
...
Iter  97 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter  98 |    Ground truth: 1.000000 |   prediction: 1.000000 |
Iter  99 |    Ground truth: 1.000000 |   prediction: 1.000000 |
loss=0.5000 accuracy=50.00%
</pre></td></tr>
</table>

Dataset: easy_XOR

n: 21

```
Iter   0 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   1 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   2 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   3 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   4 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   5 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   6 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   7 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter   8 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter   9 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter  10 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter  11 |     Ground truth: 0.000000 |   prediction: 1.000000 |
...
Iter  18 |     Ground truth: 1.000000 |   prediction: 1.000000 |
Iter  19 |     Ground truth: 0.000000 |   prediction: 1.000000 |
Iter  20 |     Ground truth: 1.000000 |   prediction: 1.000000 |
loss=0.5238 accuracy=47.62%
```

## C. Learning curve (loss, epoch curve)

*測試一*

訓練的參數、架構為

- 四層網路，兩層隱藏層，2-4-4-1
- 激活函數皆為 Sigmoid
- Optimizer: SGD ,batch=32
- Learning rate: $\eta = 0.1$
- Epoch = 10,000
- Loss function: MSE
- 每 20 場記錄一次 loss

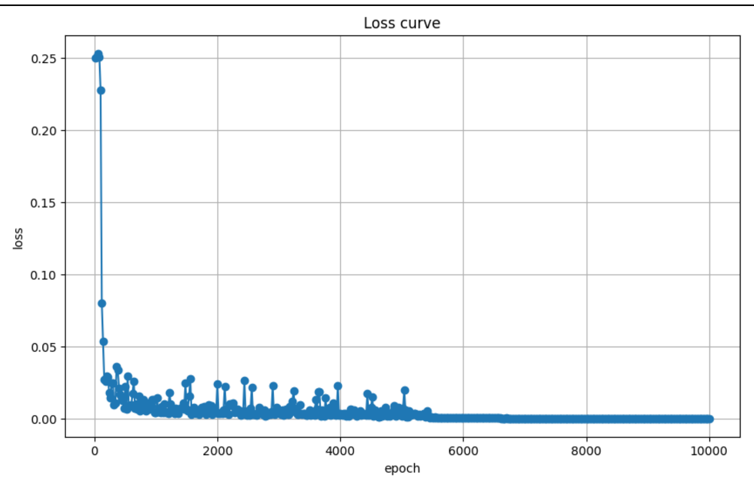
訓練的程式

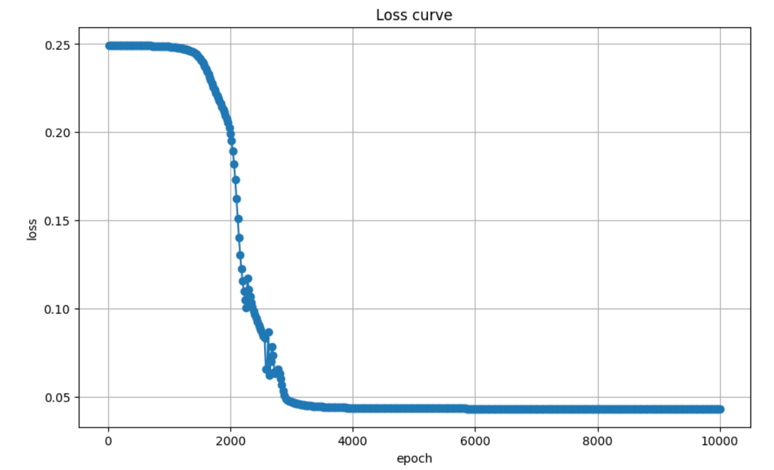
```
m = model(memory_epoch=20)
m.clear()
m.add_layer(2,4,sigmoid,derivative_sigmoid)
m.add_layer(4,4,sigmoid,derivative_sigmoid)
m.add_layer(4,1,sigmoid,derivative_sigmoid)
m.training(10000,x,y,0.1,1000,batch=32)
pred_y = m.testing(x,y,show_info=True)
show_result(x,y,(pred_y > 0.5))
m.show_learning_curve_loss()
m.show_learning_curve_accuracy()
```

實驗結果

Dataset: Linear

n: 100

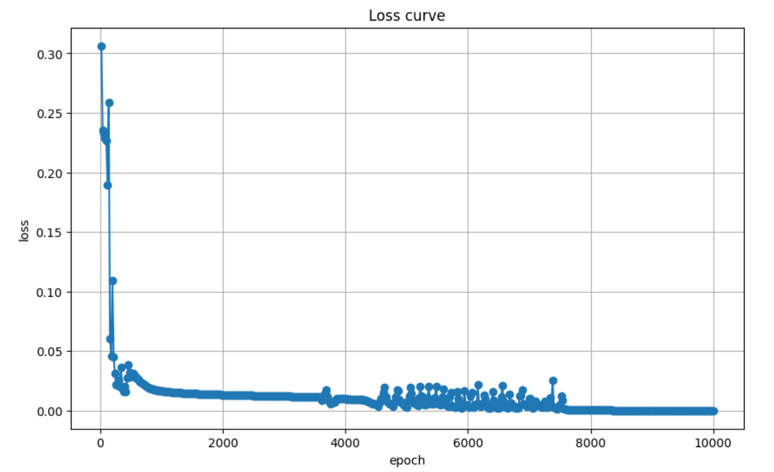


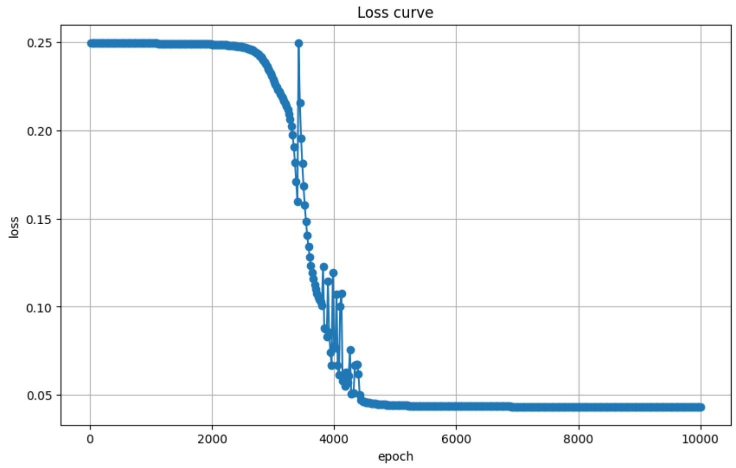
| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

*測試二*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=256<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=256)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

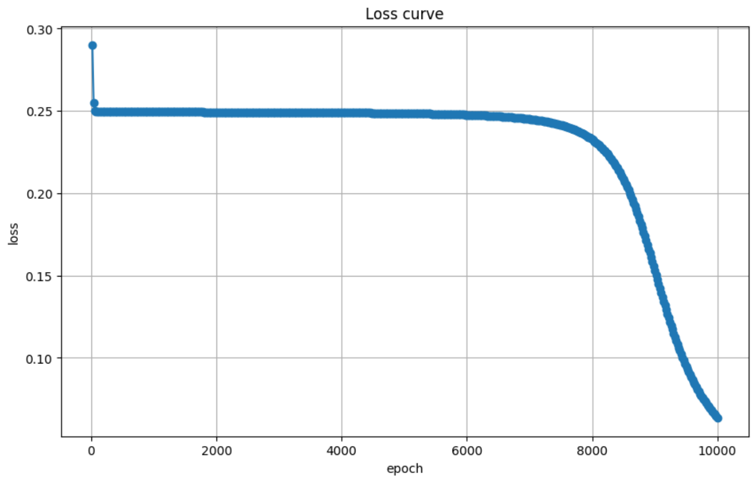
| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

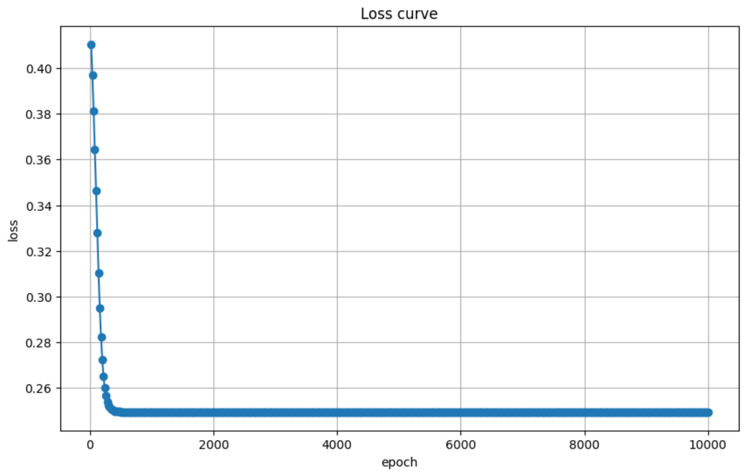


*測試三*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta$ = 0.001<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,4,sigmoid,derivative_sigmoid)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.001,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

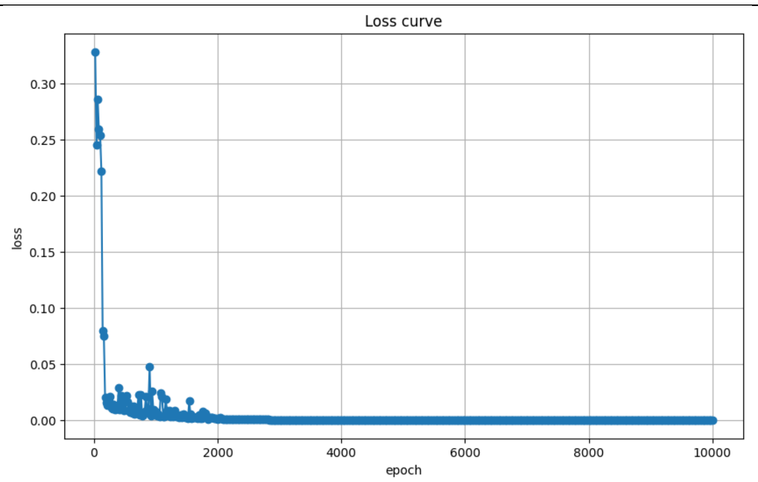
| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

*測試四*

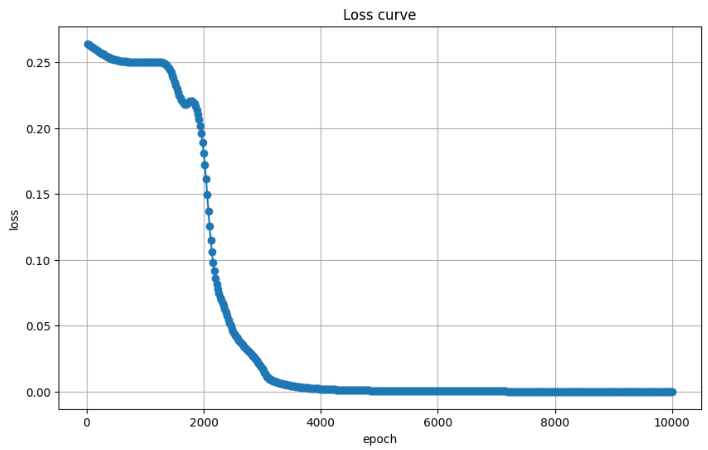
| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-8-8-1<br>● 激活函數皆為 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,8,sigmoid,derivative_sigmoid)<br>m.add_layer(8,8,sigmoid,derivative_sigmoid)<br>m.add_layer(8,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

Dataset: easy_XOR

n: 21



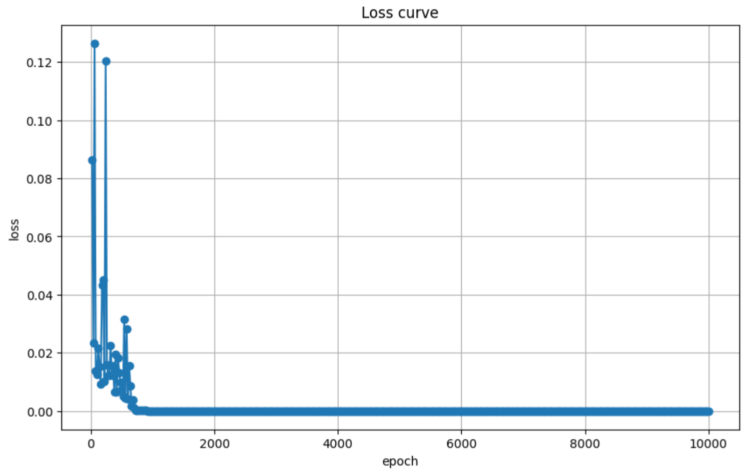

*測試五*

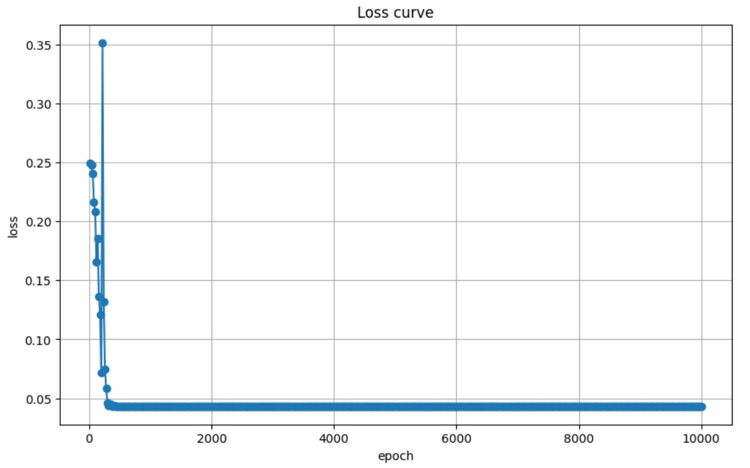
| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 激活函數改成兩個隱藏層 ReLU、輸出層 Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4,relu,derivative_relu)<br>m.add_layer(4,4,relu,derivative_relu)<br>m.add_layer(4,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

Dataset: Linear

n: 100



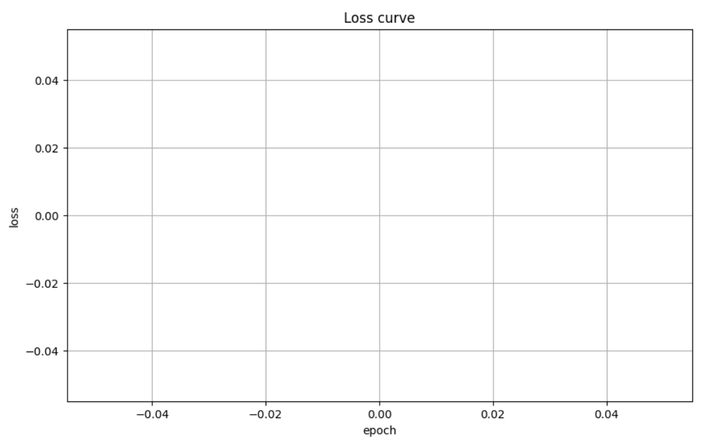
| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

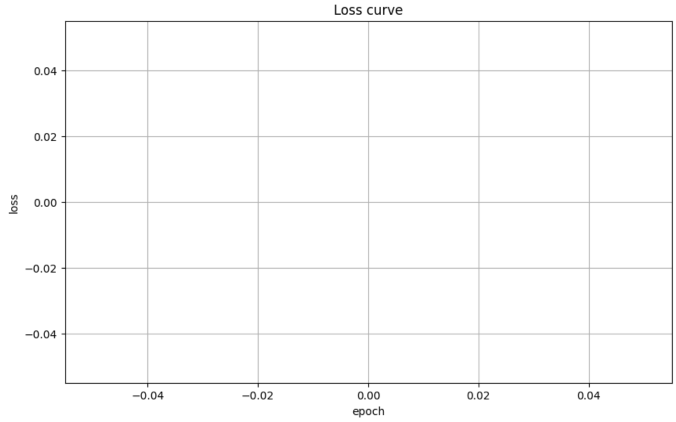


*測試六*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 無激活函數<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4)<br>m.add_layer(4,4)<br>m.add_layer(4,1)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

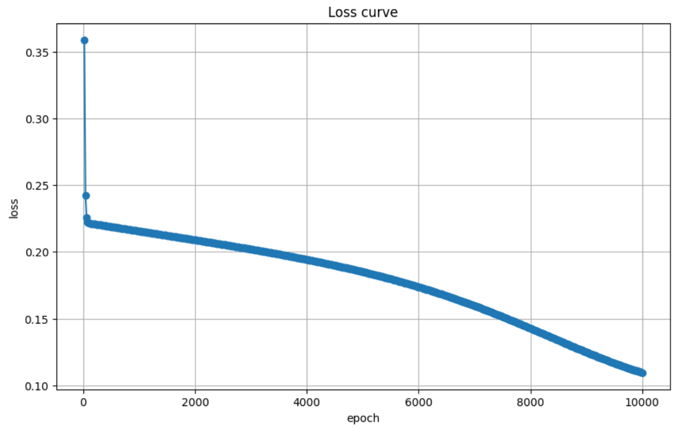
| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

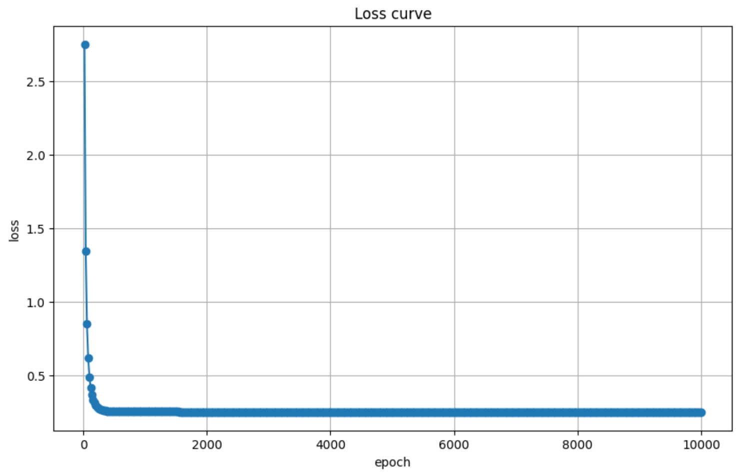
| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

*測試七*

| 訓練的參數、架構為 | 訓練的程式 |
|---|---|
| ● 四層網路，兩層隱藏層，2-4-4-1<br>● 無激活函數<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.00001$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss | m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,4)<br>m.add_layer(4,4)<br>m.add_layer(4,1)<br>m.training(10000,x,y,0.00001,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy() |

實驗結果

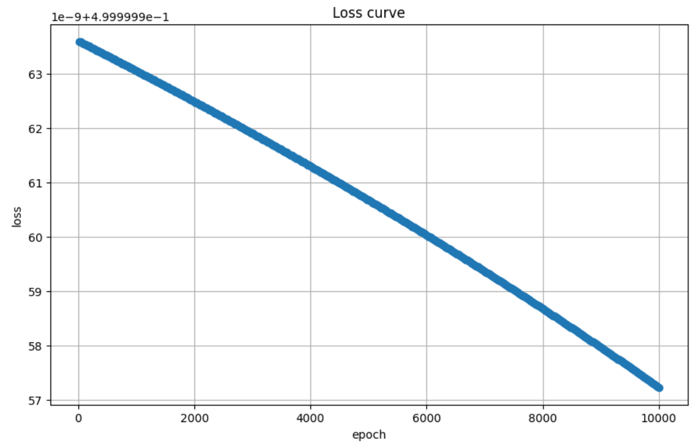
| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

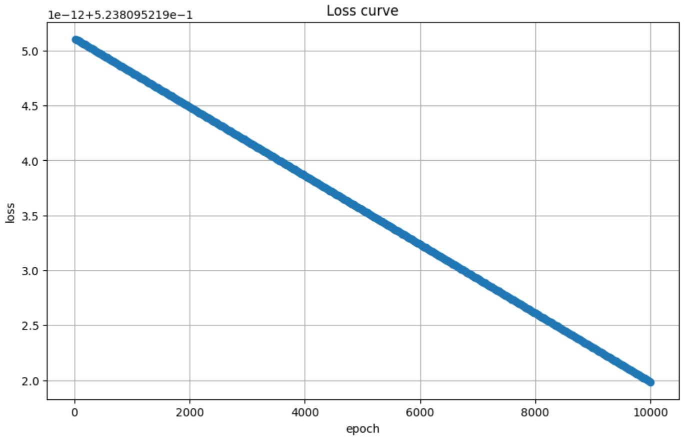
| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

*測試八*

<table>
<tr>
<td>訓練的參數、架構為<br>● 四層網路，兩層隱藏層，2-40-40-1<br>● All Sigmoid<br>● Optimizer: SGD ,batch=32<br>● Learning rate: $\eta = 0.1$<br>● Epoch = 10,000<br>● Loss function: MSE<br>● 每 20 場記錄一次 loss</td>
<td>訓練的程式<br>m = model(memory_epoch=20)<br>m.clear()<br>m.add_layer(2,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,40,sigmoid,derivative_sigmoid)<br>m.add_layer(40,1,sigmoid,derivative_sigmoid)<br>m.training(10000,x,y,0.1,1000,batch=32)<br>pred_y = m.testing(x,y,show_info=True)<br>show_result(x,y,(pred_y > 0.5))<br>m.show_learning_curve_loss()<br>m.show_learning_curve_accuracy()</td>
</tr>
</table>

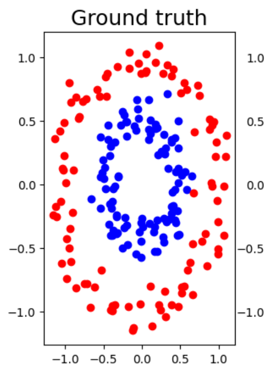
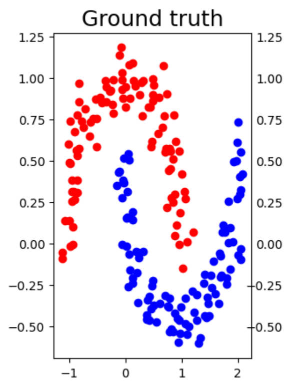
實驗結果

| | |
|---|---|
| Dataset: Linear<br>n: 100 |  |

| | |
|---|---|
| Dataset: easy_XOR<br>n: 21 |  |

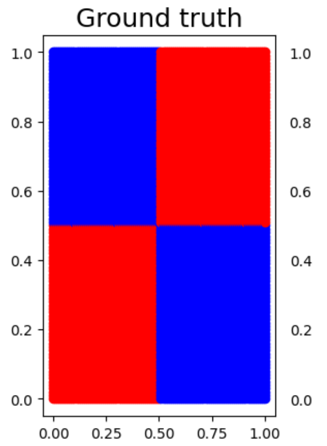
## D. Anything you want to present

此部分會提出四個不同圖形訓練的結果，和線性時 n=10k 時的訓練結果，並且測試 ReLU 針對 XOR 甚至其他資料集有 noise 時的訓練情況。但是此部分就不會特別顯示 dataset 中每一個 data 的跌代結果，而是以準確度/Loss 作比較。

| | |
|---|---|
| From sklearn.datasets<br>Dataset: circles<br>n: 200<br>noise: 0.1<br>factor: 0.5 |  |
| From sklearn.datasets<br>Dataset: moon<br>n: 200<br>noise: 0.1 |  |

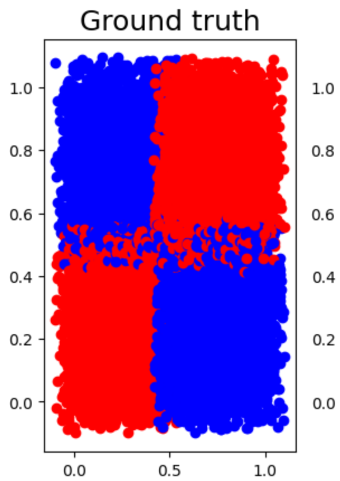
Dataset: FULL_XOR

```
def generate_XOR_hard():
    import numpy as np
    inputs = []
    labels = []

    for i in range(101):
        for j in range(101):
            x1 = 0.01 * i
            x2 = 0.01 * j
            inputs.append([x1, x2])
            labels.append(int((x1 > 0.5) != (x2 > 0.5)))

    return np.array(inputs), np.array(labels).reshape(-1, 1)
```




Dataset: NOISE_XOR

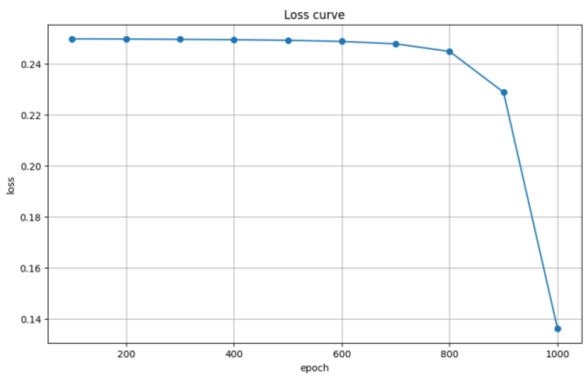
```
def generate_XOR_hard_2(noise_level=0.1):
    inputs = []
    labels = []

    for i in range(101):
        for j in range(101):
            x1 = 0.01 * i
            x2 = 0.01 * j
            # 引入隨機噪音
            x1_noisy =x1+np.random.uniform(-noise_level, noise_level)
            x2_noisy=x2+np.random.uniform(-noise_level, noise_level)
            inputs.append([x1_noisy, x2_noisy])
            labels.append(int((x1 > 0.5) != (x2 > 0.5)))

    return np.array(inputs), np.array(labels).reshape(-1, 1)
```




實驗結果 Dataset: Linear (n=10000)

訓練的參數、架構為

- 2-4-4-1
- All Sigmoid
- Optimizer: SGD ,batch=32
- Learning rate: $\eta = 0.0001$
- Epoch = 1000
- Loss function: MSE
- 每 100 場記錄一次 loss



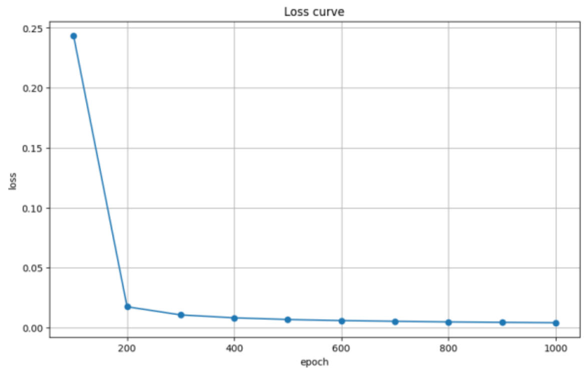
```
epoch      100: loss : 0.2498853331
epoch      200: loss : 0.2498003714
epoch      300: loss : 0.2496937947
epoch      400: loss : 0.2495432784
epoch      500: loss : 0.2493050988
epoch      600: loss : 0.2488727215
epoch      700: loss : 0.2479128666
epoch      800: loss : 0.2449369282
epoch      900: loss : 0.2289297439
epoch     1000: loss : 0.1361682368
```

loss=0.1362 accuracy=99.80%

訓練的參數、架構為

- 2-4-4-1
- All Sigmoid
- Optimizer: MGD ,batch=32
- MGD: m=0.9
- Learning rate: $\eta$ = 0.0001
- Epoch = 1000
- Loss function: MSE

每 100 場記錄一次 loss



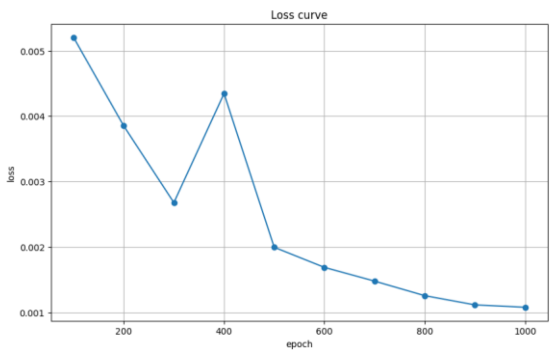

```
epoch      100: loss : 0.2435768228
epoch      200: loss : 0.0173853200
epoch      300: loss : 0.0106079517
epoch      400: loss : 0.0081680268
epoch      500: loss : 0.0067708078
epoch      600: loss : 0.0058787222
epoch      700: loss : 0.0052704563
epoch      800: loss : 0.0047593374
epoch      900: loss : 0.0043890697
epoch     1000: loss : 0.0040914816
```

loss=0.0041 accuracy=99.88%

訓練的參數、架構為

- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: MGD ,batch=32
- MGD: m=0.9
- Learning rate: $\eta$ = 0.0001
- Epoch = 1000
- Loss function: MSE

每 100 場記錄一次 loss



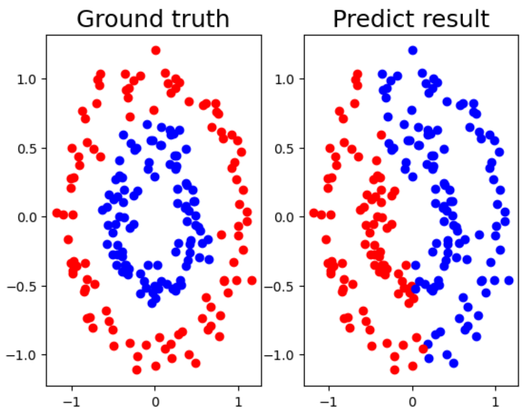
```
epoch      100: loss : 0.0052056205
epoch      200: loss : 0.0038520729
epoch      300: loss : 0.0026780970
epoch      400: loss : 0.0043465837
epoch      500: loss : 0.0019952600
epoch      600: loss : 0.0016877267
epoch      700: loss : 0.0014776867
epoch      800: loss : 0.0012551061
epoch      900: loss : 0.0011142598
epoch     1000: loss : 0.0010774817
```

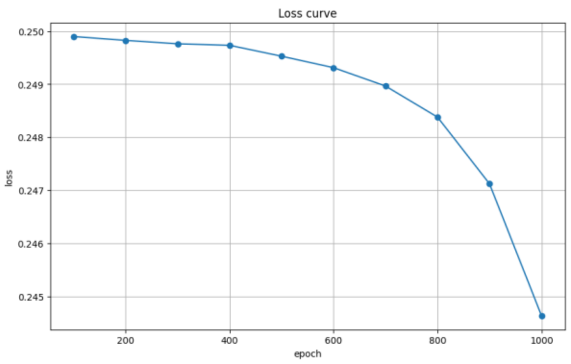
loss=0.0011 accuracy=99.96%

實驗結果 Dataset: circles

訓練的參數、架構為

- 2-4-4-1
- All Sigmoid
- Optimizer: MGD ,batch=32
- MGD: m=0.9
- Learning rate: $\eta = 0.001$
- Epoch = 1000
- Loss function: MSE
- 每 100 場記錄一次 loss




```
epoch      100: loss : 0.2498977955
epoch      200: loss : 0.2498251876
epoch      300: loss : 0.2497611847
epoch      400: loss : 0.2497329881
epoch      500: loss : 0.2495264324
epoch      600: loss : 0.2493112635
epoch      700: loss : 0.2489644927
epoch      800: loss : 0.2483789609
epoch      900: loss : 0.2471224901
epoch     1000: loss : 0.2446382968
```

loss=0.2446 accuracy=50.00%

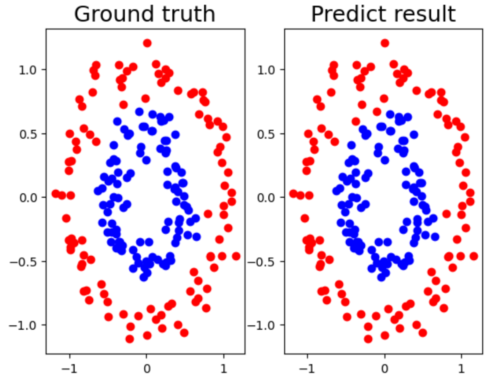
<table>
<tr>
<td>

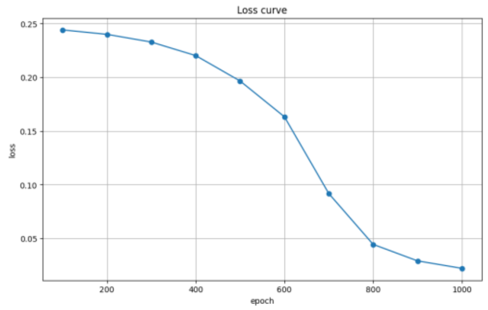
訓練的參數、架構為

- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: SGD,batch=32
- Learning rate: $\eta = 0.001$
- Epoch = 1000
- Loss function: MSE

每 100 場記錄一次 loss

</td>
<td>




```
epoch      100: loss : 0.2440455485
epoch      200: loss : 0.2399360203
epoch      300: loss : 0.2327509124
epoch      400: loss : 0.2201754903
epoch      500: loss : 0.1965452695
epoch      600: loss : 0.1632025345
epoch      700: loss : 0.0917733477
epoch      800: loss : 0.0444674842
epoch      900: loss : 0.0291325038
epoch     1000: loss : 0.0221893771
```

loss=0.0222 accuracy=99.00%

</td>
</tr>
</table>

實驗結果 Dataset: moon

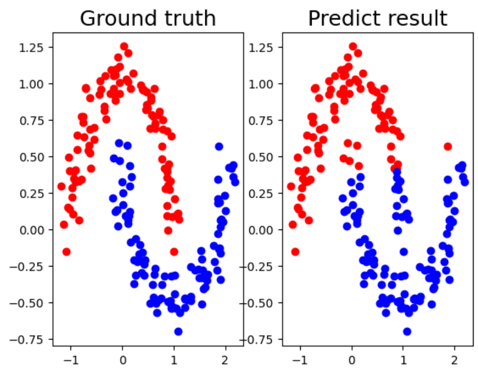
<table>
<tr>
<td>

訓練的參數、架構為

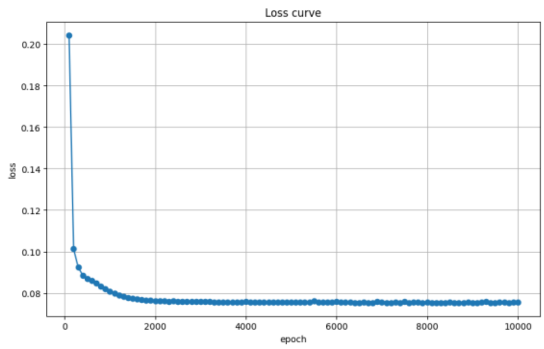
- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: SGD,batch=32
- Learning rate: $\eta = 0.001$
- Epoch = 10000
- Loss function: MSE

每 100 場記錄一次 loss

</td>
<td>



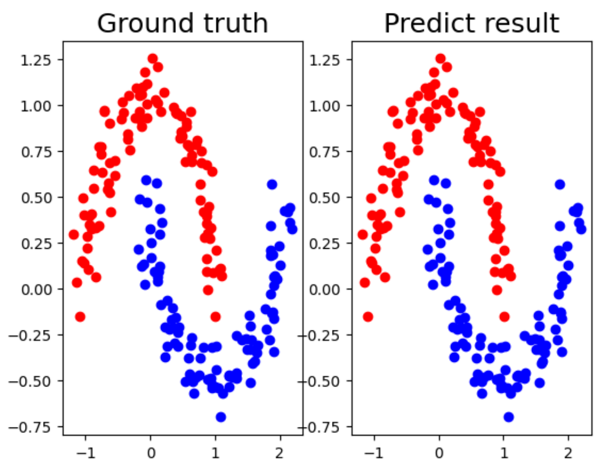

</td>
</tr>
</table>

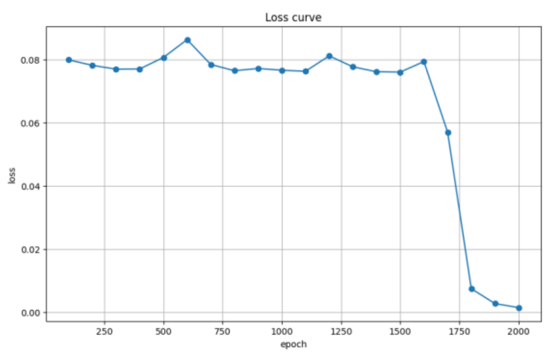
<table>
<tr>
<td></td>
<td>



```
epoch     1000: loss : 0.0808368588
epoch     2000: loss : 0.0762809994
epoch     3000: loss : 0.0757393539
epoch     4000: loss : 0.0757722426
epoch     5000: loss : 0.0755170343
epoch     6000: loss : 0.0757930216
epoch     7000: loss : 0.0753995741
epoch     8000: loss : 0.0755591715
epoch     9000: loss : 0.0753732463
epoch    10000: loss : 0.0754870536
```

loss=0.0755 accuracy=89.50%
</td>
</tr>
<tr>
<td>
訓練的參數、架構為

- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: SGD,batch=32
- Learning rate: $\eta = 0.01$
- Epoch = 2000
- Loss function: MSE

每 100 場記錄一次 loss
</td>
<td>


</td>
</tr>
</table>

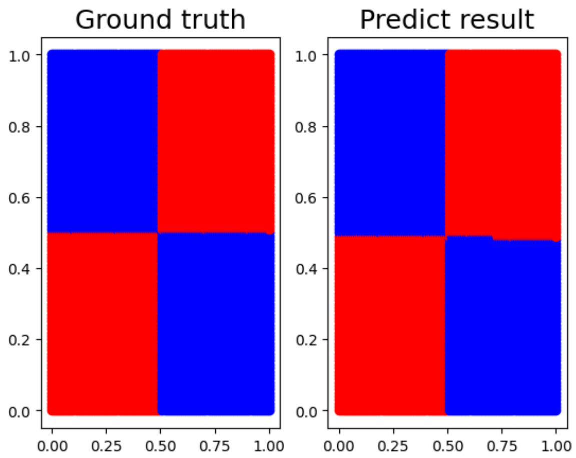
```
epoch      100: loss : 0.0799210299
epoch      200: loss : 0.0781728730
epoch      300: loss : 0.0769881142
epoch      400: loss : 0.0770503882
epoch      500: loss : 0.0806877594
epoch      600: loss : 0.0863284876
epoch      700: loss : 0.0784576838
epoch      800: loss : 0.0764851774
epoch      900: loss : 0.0772038903
epoch     1000: loss : 0.0766427659
epoch     1100: loss : 0.0763187471
epoch     1200: loss : 0.0811866722
epoch     1300: loss : 0.0777097912
epoch     1400: loss : 0.0761644141
epoch     1500: loss : 0.0760481900
epoch     1600: loss : 0.0793864424
epoch     1700: loss : 0.0570503632
epoch     1800: loss : 0.0074747011
epoch     1900: loss : 0.0027443661
epoch     2000: loss : 0.0014730818
```

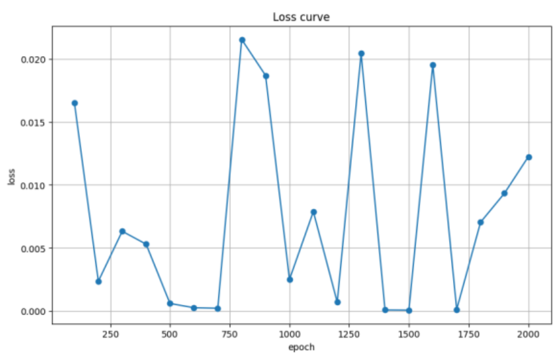
loss=0.0015 accuracy=100.00%

實驗結果 Dataset: FULL_XOR

訓練的參數、架構為

- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: SGD,batch=32
- Learning rate: $\eta = 0.01$
- Epoch = 2000
- Loss function: MSE
- 每 100 場記錄一次 loss



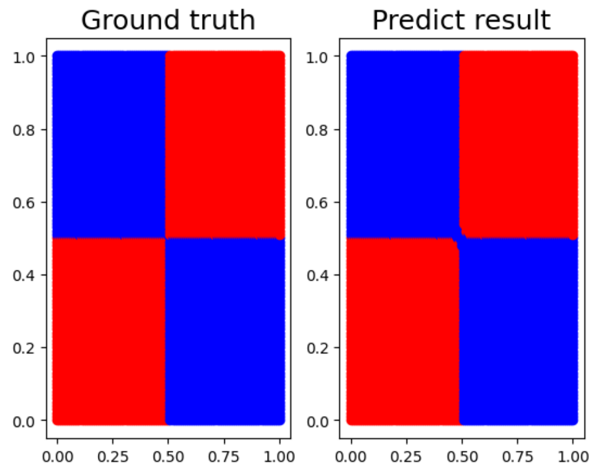
```
epoch      100: loss : 0.0165038715
epoch      200: loss : 0.0023672171
epoch      300: loss : 0.0063305697
epoch      400: loss : 0.0053082428
epoch      500: loss : 0.0006097254
epoch      600: loss : 0.0002730639
epoch      700: loss : 0.0002323780
epoch      800: loss : 0.0215271515
epoch      900: loss : 0.0186774625
epoch     1000: loss : 0.0025144624
epoch     1100: loss : 0.0078891194
epoch     1200: loss : 0.0007132203
epoch     1300: loss : 0.0204334465
epoch     1400: loss : 0.0000946403
epoch     1500: loss : 0.0000813938
epoch     1600: loss : 0.0195263886
epoch     1700: loss : 0.0001350658
epoch     1800: loss : 0.0070482221
epoch     1900: loss : 0.0093425724
epoch     2000: loss : 0.0122406437
```

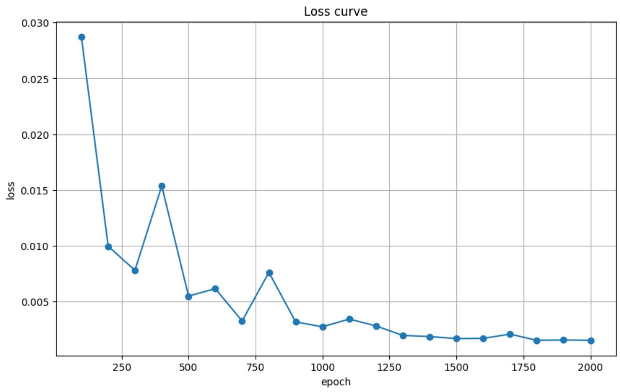
loss=0.0122 accuracy=98.73%

訓練的參數、架構為

- 2-4-4-1
- All Sigmoid
- Optimizer: SGD,batch=32
- Learning rate: $\eta = 0.01$
- Epoch = 2000
- Loss function: MSE

每 100 場記錄一次 loss



```
epoch      100: loss : 0.0287416821
epoch      200: loss : 0.0099525967
epoch      300: loss : 0.0077941574
epoch      400: loss : 0.0153516585
epoch      500: loss : 0.0054871759
epoch      600: loss : 0.0061459944
epoch      700: loss : 0.0032526873
epoch      800: loss : 0.0076043243
epoch      900: loss : 0.0031686371
epoch     1000: loss : 0.0027288682
epoch     1100: loss : 0.0034240399
epoch     1200: loss : 0.0028072697
epoch     1300: loss : 0.0019539846
epoch     1400: loss : 0.0018530344
epoch     1500: loss : 0.0016738577
epoch     1600: loss : 0.0016925907
epoch     1700: loss : 0.0020718128
epoch     1800: loss : 0.0015220673
epoch     1900: loss : 0.0015485871
epoch     2000: loss : 0.0015199282
```

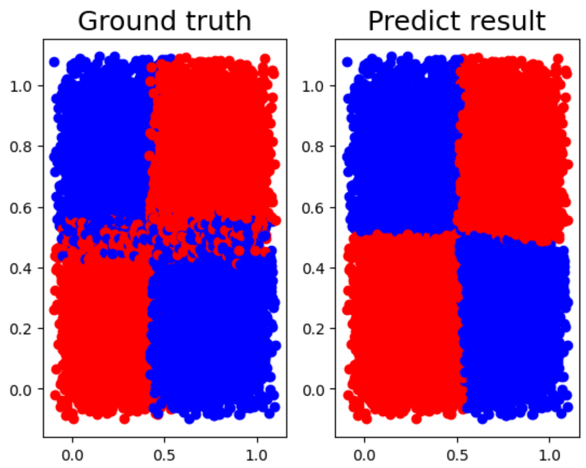
loss=0.0015 accuracy=99.90%

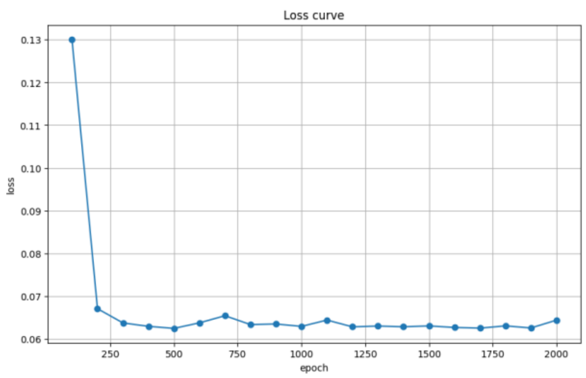
實驗結果 Dataset: NOISE_XOR

訓練的參數、架構為

- 2-4-4-1
- ReLU、ReLU、Sigmoid
- Optimizer: MGD ,batch=32
- MGD: m=0.9
- Learning rate: $\eta = 0.001$
- Epoch = 2000
- Loss function: MSE

每 100 場記錄一次 loss



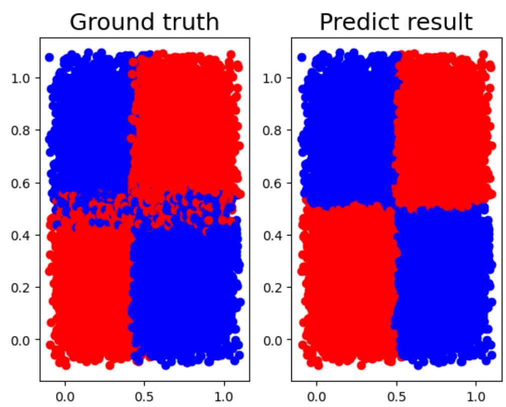
```
epoch      100: loss : 0.1301020689
epoch      200: loss : 0.0670343594
epoch      300: loss : 0.0636968503
epoch      400: loss : 0.0629052538
epoch      500: loss : 0.0624233089
epoch      600: loss : 0.0637210663
epoch      700: loss : 0.0653831682
epoch      800: loss : 0.0633119257
epoch      900: loss : 0.0634533628
epoch     1000: loss : 0.0628892998
epoch     1100: loss : 0.0643690147
epoch     1200: loss : 0.0627819950
epoch     1300: loss : 0.0629644784
epoch     1400: loss : 0.0628001270
epoch     1500: loss : 0.0629924668
epoch     1600: loss : 0.0626440852
epoch     1700: loss : 0.0624868483
epoch     1800: loss : 0.0630065514
epoch     1900: loss : 0.0625254817
epoch     2000: loss : 0.0643548811
```

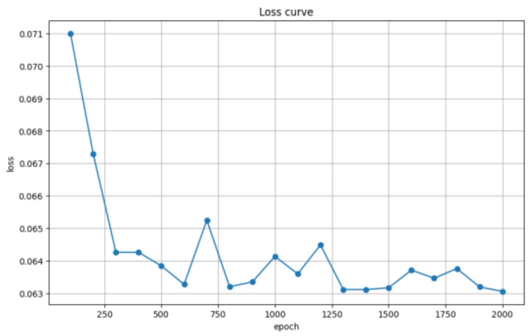
loss=0.0644 accuracy=90.39%

訓練的參數、架構為

- 2-4-4-1
- All Sigmoid
- Optimizer: MGD ,batch=32
- MGD: m=0.9
- Learning rate: $\eta = 0.001$
- Epoch = 2000
- Loss function: MSE

每 100 場記錄一次 loss



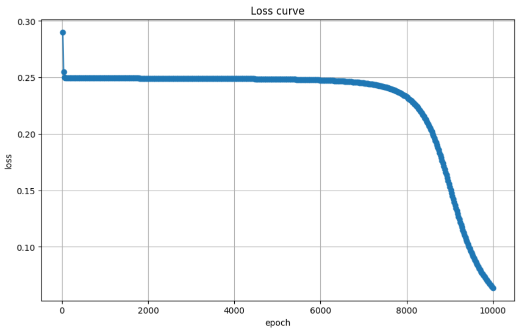
```
epoch      100: loss : 0.0710009712
epoch      200: loss : 0.0672963216
epoch      300: loss : 0.0642635233
epoch      400: loss : 0.0642603116
epoch      500: loss : 0.0638431680
epoch      600: loss : 0.0632746467
epoch      700: loss : 0.0652482769
epoch      800: loss : 0.0632033806
epoch      900: loss : 0.0633493623
epoch     1000: loss : 0.0641362119
epoch     1100: loss : 0.0635956917
epoch     1200: loss : 0.0644910811
epoch     1300: loss : 0.0631159548
epoch     1400: loss : 0.0631140738
epoch     1500: loss : 0.0631711395
epoch     1600: loss : 0.0637140844
epoch     1700: loss : 0.0634630776
epoch     1800: loss : 0.0637612659
epoch     1900: loss : 0.0631954616
epoch     2000: loss : 0.0630587606
```

loss=0.0631 accuracy=90.48%

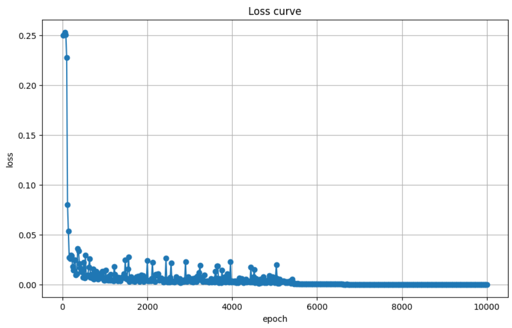
# 4. Discussion

### A. Try different learning rates

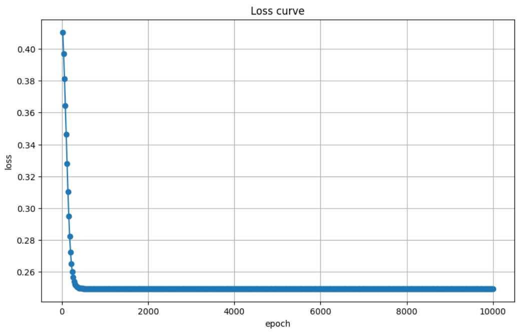
在上方的實驗中，learning rate 的影響，最基本的是影響訓練的收斂速度圖，和收斂於區域最佳解，如圖 4.1 下方四張(a),(b),(c),(d)圖，這四張圖來自於第三章提到的*測試一*與*測試三*的 Loss 曲線圖，由曲線圖中可以看到 xor 的曲線(b)、(d)都已經收斂了，但是(b)收斂在區域最佳解，沒有到非常好的結果，而由(a)、(b)兩張 linear 的 Loss 曲線可以看出 (a) 因為學習率為 0.001 的關係，因此尚未收斂，而(b) 學習率為 0.1 的嘗試大概小於一千次就收斂了，這是其中一個可以看到調整 Learning rate 會得到的結果。



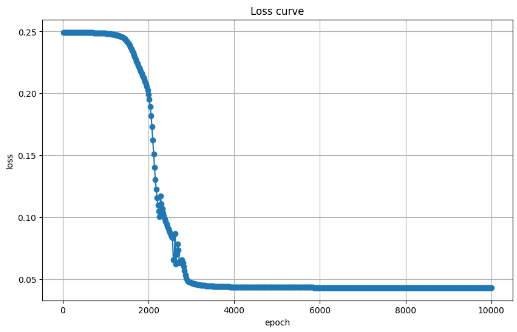
(a) linear



(c) linear



(b) non-linear



(d) non-linear

圖 4.1、(a) 與 (b) 皆為學習率 0.001 於(測試三)的 loss 曲線，分別是 (a) linear (b) xor；(c) 與 (d) 皆為學習率 0.1 於(測試一)的 loss 曲線

接著要提到另一個學習率會影響的結果，於*測試六*與*測試七*時，由於一次的移動步伐太大，或者矩陣運算時溢位了(overflow)，導致測試六無法去做訓練，但是調整學習率，調整到非常低時就可以看到學習的效果，具體的學習效果會在 C. Try without activation functions 時說明。

### B. Try different numbers of hidden units

藉由測試一與測試四，在同樣的參數下，同樣的訓練下，不同的網路

模型，多個 unit 會給予更好的訓練結果，但是過多的 unit 反倒會導致訓練收斂變慢，一次訓練時間變長，如*測試八*或圖 4.2。

```
epoch     1000: loss : 0.0042380260
epoch     2000: loss : 0.0238643824
epoch     3000: loss : 0.0061306842
epoch     4000: loss : 0.0033579711
epoch     5000: loss : 0.0014456249
epoch     6000: loss : 0.0003044721
epoch     7000: loss : 0.0001621299
epoch     8000: loss : 0.0001091968
epoch     9000: loss : 0.0000816992
epoch    10000: loss : 0.0000645244
```

(a) 測試一 線性 訓練時 已收斂

```
epoch     1000: loss : 0.4999999631
epoch     2000: loss : 0.4999999625
epoch     3000: loss : 0.4999999619
epoch     4000: loss : 0.4999999613
epoch     5000: loss : 0.4999999607
epoch     6000: loss : 0.4999999600
epoch     7000: loss : 0.4999999594
epoch     8000: loss : 0.4999999587
epoch     9000: loss : 0.4999999580
epoch    10000: loss : 0.4999999572
```

(b) 測試八 線性 訓練時收斂慢

圖 4.2、測試一與測試八 Neural network 中不同 Unit 數量收斂的差異

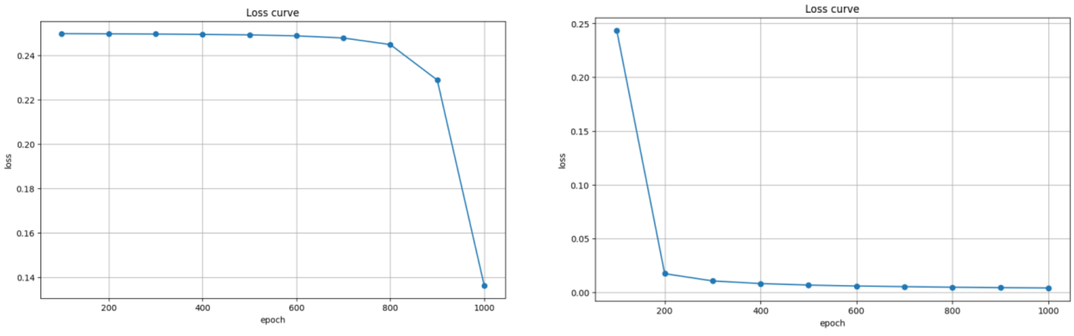
## C. Try without activation functions

在 A 小節時有提到，不使用激活函數的情況下可能會導致做矩陣乘法時溢位的問題，而調整學習率可以改善矩陣運算溢位的問題，但是學習率越小又會導致區域最佳解的行程，此外從*測試七*的圖中可以看到沒有帶入激活函數是很難達到解非線性問題的。

## D. Anything you want to share

由多個測試中，使用 ReLU 來當作激活函數通常來講要比 Sigmoid 還要來的好(於 3. D. Anything you want to present 中的 Linear (n=10000) 和 circles 的 dataset)中都可以看到，將中間隱藏層的激活函數改成 ReLU，效果都會比直接使用 Sigmoid 還要來的好，而且 Sigmoid 可能會遇到梯度消失的情況(於 Extra 中有提到)，而 ReLU 並不會有梯度消失的問題。

此外本研究報告還有實作 SGD 與 MGD，由下方圖 4.3 中可以看到，同樣是使用 Sigmoid 在相同的學習率$\eta$下，MGD 的訓練收斂表現比 SGD 還要來的快。




(a)學習率低時利用 SGD 的收斂狀況 (b)學習率低時利用 MGD 的收斂狀況

圖 4.3、相同網路架構，學習率 $\eta = 0.0001$ MGD 與 SGD 的訓練收斂比較圖

# 5. Extra

## A. Implement different optimizers

在本實驗中以 SGD(Stochastic Gradient Descent)作為基礎的 Optimizer，他會將輸出的 training data 分成多個指定的大小(batch)，然後分批去做訓練，但是在分的過程中會去打亂分布，確保資料不會一值相同的訓練。在實作上則是以下方的程式碼為例

```
def training(self,n,training_data,ground_truth,learning_rate=0.1,show_info=5000,batch=32):
    training_set = np.hstack((training_data,ground_truth))
    tot_dataset = len(training_data)

    for epoch in range(n):
        np.random.shuffle(training_set)
        batch_training_data = training_set[:,:-1]
        batch_prediction = training_set[:,-1:]
        for i in range(int(math.ceil(tot_dataset / batch))):
            st = batch * i
            ed = min(batch * (i+1),tot_dataset)

            prediction = self.forward(batch_training_data[st:ed])
            # backward pass compute \delta weights
            self.backward(batch_prediction[st:ed],prediction)

            # update
            self.update(learning_rate)
```

先把 training_data 與 ground_truth 堆疊起來，確保在 shuffle 時不會失去對應的值，然後接著根據 batch 的大小去做分配，每一次 batch 的訓練完畢才會去做 backward 的動作和 update 參數。

這樣的優點在於當資料集過大時，分批可以更好的去訓練模型，比起一般的 GD 一次性全部的資料去訓練，準確度大幅提升。

本實驗報告也嘗試了另一種的優化器，Momentum-based Gradient Descent，MGD 帶出了以下式子：

$$v^t = \begin{cases} \eta\frac{\partial L}{\theta} & \text{if } t = 0 \\ mv^{(t-1)} + \eta\frac{\partial L}{\theta} & \text{if } t \geq 1 \end{cases}$$

$$\theta = \theta - v^t$$

MGD 的原理是控制在更新時的移動步數，如果跟上一次更新同向則

會加倍反之削減，在每一個 neural 中的參數都需要維護一個 MGD 的 Optimizer，MGD 有一個參數$m$用來調整根據上一次移動的大小偏差，如果越高，那往同一方向的修正時移動量會增加。下方程式碼為 MGD 的實例：

```
class Momentum(Optimizer):
    m = 0.9
    # v^t = [ learning_rate * gradient                  t=0
    #         [ mv(t-1) + learning_rate * gradient t>=1
    #
    # theta = theta - v(t)

    last_v = None
    def __init__(self,m=0.9):
        self.m = m

    # calc    v^(t)
    def calc(self,gradient):
        if (self.last_v is None):
            ret = gradient
        else:
            ret = self.last_v * self.m + gradient
        self.last_v = ret
        return ret
```

相對的在 neural network update 時也會判斷是否有優化器(optimizer)，有的話則使用該優化器。

```
def update(self,learning_rate):
    if (self.weight_optimizer != None and self.bias_optimizer!= None ):
        self.W -= self.weight_optimizer.calc(learning_rate * self.dW)
        self.b -= self.bias_optimizer.calc(learning_rate * self.db)
    else:
        self.W -= learning_rate * self.dW
        self.b -= learning_rate * self.db
```

## B. Implement different activation functions

Sigmoid 存在當輸入值$x$很大時$\sigma(x)$ ≈1，或$x$很小時$\sigma(x) \approx 0$，這兩者的微分$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$ 都會偏向於 0，導至梯度消失(Vanishing Gradient)的現象。會導致權重的更新變得緩慢，甚至直接收斂。

而為了避免梯度消失的情況，本實驗報告有額外設計一個激活函數 ReLU(Rectified Linear Unit)，函數圖形如圖 5.1 所示

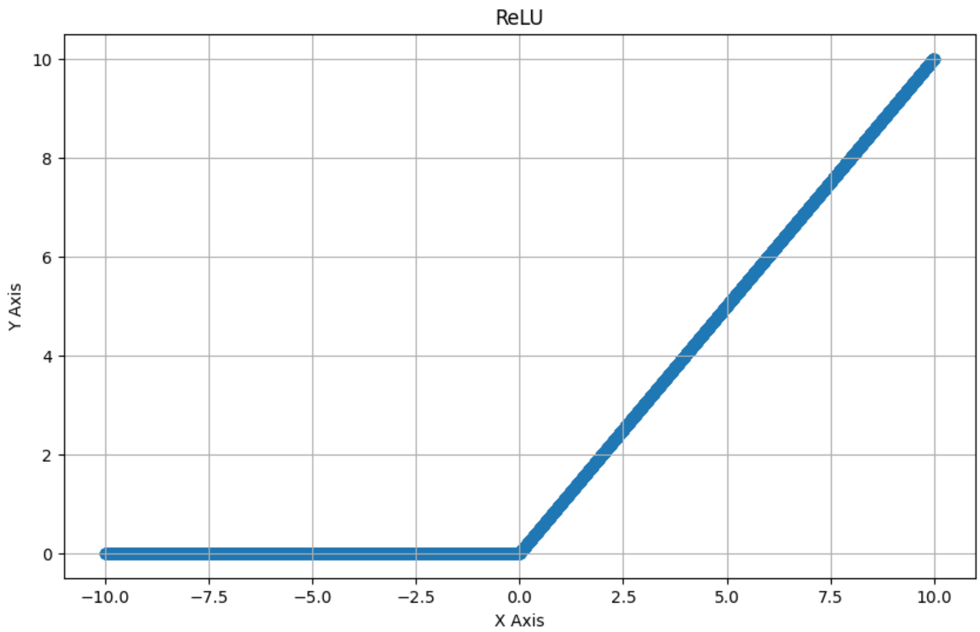



圖 5.1、ReLU 函數圖

ReLU 的數學式子如下：

$$ReLU(x) = \max(0, x)$$

與 Sigmoid 的導數相比，ReLU 本身的導數更容易計算，計算速度上也提升，並且也能防止上述提到的梯度消失的問題，並且 ReLU 比 Sigmoid 更利於模型學習稀疏特徵，ReLU 的導數式子如下：

$$ReLU'(x) = 1, x > 0$$

$$ReLU'(x) = 0, x \leq 0$$

```
def relu(x):
    return x * (x > 0)


def derivative_relu(x):
    return 1. * (x > 0)
```