

# **Lab1 : back-propagation**

**IOC/資科工碩**

**學號: 310551031**

**張皓雲**

# 目錄

1. Introduction.....	4
A. 問題定義與達成目標 .....	4
A.1. Linear 分類問題分析 .....	4
A.2. XOR 分類問題分析 .....	4
B. 實驗流程大綱 .....	4
C. 實驗環境與檔案使用說明 .....	5
C.1.實驗環境.....	5
C.2.檔案使用說明.....	6
2. Experiment setups.....	6
A. Linear 與 XOR 訓練資料與測試資料的生成說明 .....	6
B. 模型整體架構與細節 .....	7
B.1. Activation Function .....	7
B.1.1. Sigmoid.....	7
B.1.2. ReLU.....	7
B.2. Loss Function.....	8
B.3. Neural network.....	8
B.3.1.初始化、定義權重(Weights)與誤差(Bias) .....	9
B.3.2. Forward propagation .....	10
B.3.3. Backward propagation .....	11
B.3.4. Update Weights.....	12
3. Results of testing .....	12
A. Testing Ground Truth 與 Predict Result 之間的比較.....	13
A.1. Linear 分類 .....	13
A.2. XOR 分類.....	13
B. Testing Accuracy 與 Testing 預測結果 .....	14
B.1. Linear 分類 .....	14
B.2. XOR 分類.....	14
C. Loss 與 Accuracy 的 Training 學習曲線.....	15
C.1. Linear 分類 .....	15
C.2. XOR 分類.....	15
D. Testing predict result 的 Confusion Matrix.....	16
D.1. Linear 分類 .....	16
D.2. XOR 分類.....	16
E. 隨著 Training Epochs 變化的 Predict Result.....	16
E.1. Linear 分類 .....	16
E.2. XOR 分類.....	17
4. Discussion.....	17
A. 不同 Learning rate 下的模型 Training 與 Testing 情況.....	17
B. 不同 Hidden Unit 下的模型 Training 與 Testing 情況 .....	18
B.1. 改變第一層 Hidden Unit.....	18

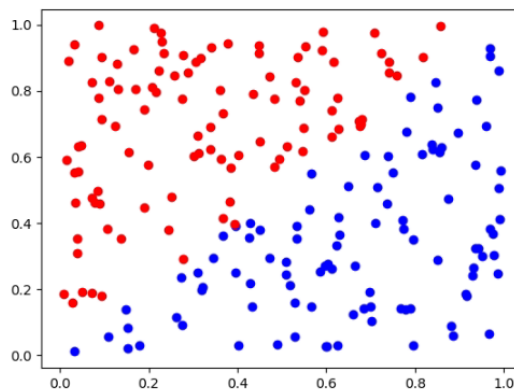
B.2. 改變第二層 Hidden Unit.....	19
C. 不同 Activation Function 下的模型 Training 與 Testing 情況 .....	19
D. 不同訓練與測試資料點數量下的模型 Training 與 Testing 情況 .....	20
5. Reference .....	21

# 1. Introduction

## A. 問題定義與達成目標

### A.1. Linear 分類問題分析

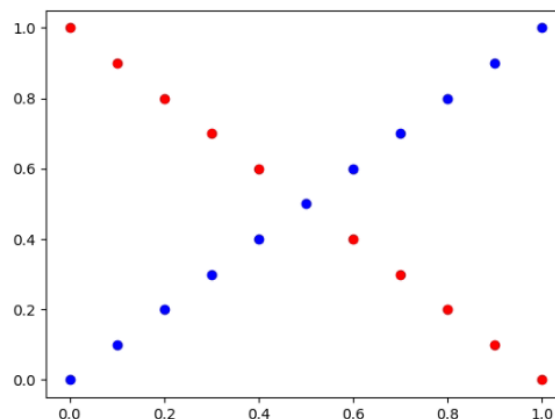
本次實驗的目標是希望將圖一及圖二中的座標點進行紅色與藍色的分類。首先看到圖一可發現，紅色座標點位在圖一的左上角，藍色座標點位在圖一的右下角，因此假定圖一中的任意座標點為 $(x,y)$ ，並經由推斷後發現。紅色座標點的  $y$  值都大於  $x$  值，藍色座標點的  $x$  值都大於  $y$  值，因此假設現在突然出現一張未被分色的座標點圖，接下來就可依據座標點 $(x,y)$ 大小的值分類該點為紅色還是藍色。



圖一、二分類線性分類問題

### A.2. XOR 分類問題分析

接著看到圖二可得知，紅色座標點位在圖二的左上至右下的正方形對角線，藍色座標點位在圖二的左下至右上的正方形對角線。因此假定圖二中的任意座標點為 $(x,y)$ ，並經由推斷後發現紅色座標點的  $x$  值與  $y$  值都是 0.1 的倍數，並且  $x$  值變大時， $y$  值也會同等變大。反之藍色座標點的  $x$  值越大  $y$  值反而會越小，因此假設現在突然出現一張未被分色的座標點圖，接下來就可依據此特性將座標點分類為紅色或藍色。

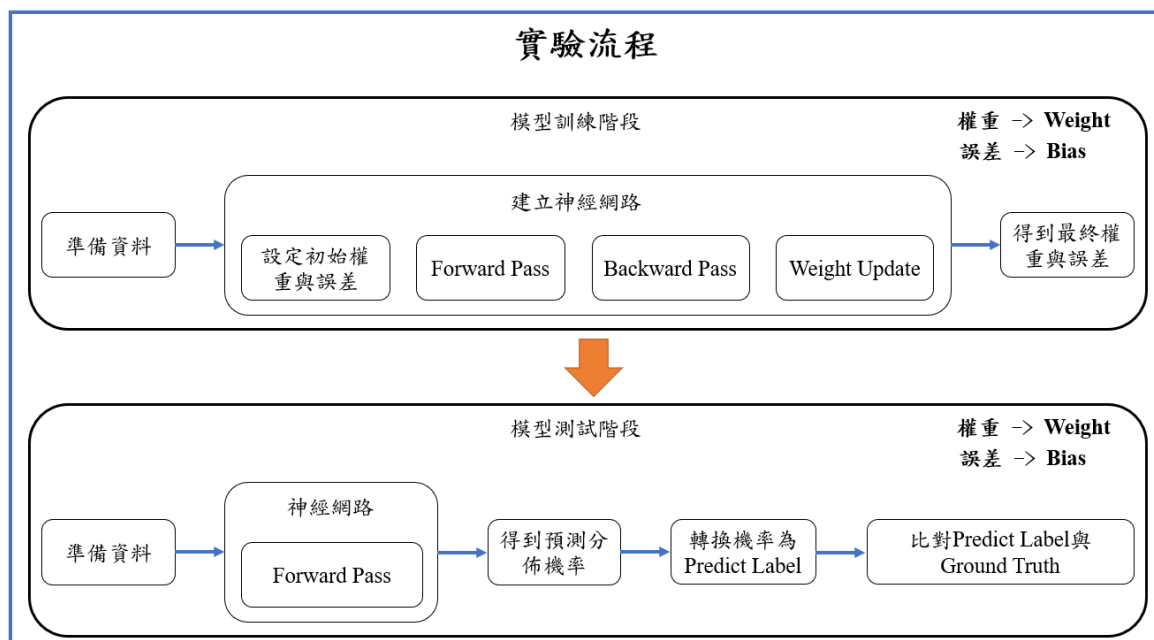


圖二、二分類 XOR 分類問題

## B. 實驗流程大綱

為了解決 A. 問題定義與達成目標敘述的問題，接下來分別建立兩階段的實驗以進行座標點的分類。實驗流程如圖三所示。實驗流程總共可以分為兩個階段。第一階段為模型訓練階段，在

模型訓練階段中，本實驗會進行兩個步驟，分別是準備 Linear 與 XOR 的訓練資料、建立神經網路。實驗流程的第二階段為模型測試階段，在此階段中，本實驗主要會進行三項步驟，分別為準備 Linear 與 XOR 的測試資料、使用第一階段得到的模型權重進行模型預測、轉換機率分佈為預測資料標籤。設計兩階段實驗流程的主要目的是希望透過正向與反向傳播的概念不斷更新模型的權重與誤差，得到一個最接近 Ground Truth 的模型權重與誤差，方能解決上述問題敘述中的座標點顏色分類問題。



圖三、實驗流程簡易表示圖

## C. 實驗環境與檔案使用說明

### C.1. 實驗環境

本次實驗使用的 python 套件有 numpy、pandas、matplotlib 及 collections，numpy 的用途為建立資料點、建立神經網路及數值的運算，pandas 與 collections 的用途則為進行數據結果的分析，例如: Confusion Matrix...，matplotlib 的用途則是繪出許多結果圖，例如: Training accuracy 隨著 epochs 變動的曲線、Training loss 隨著 epochs 變動的曲線、Ground Truth 與 Prediction 之間的結果圖比較等....。使用套件的 logo 如圖四、圖五與圖六所示。硬體設備、虛擬環境如表一所述。

表一、軟硬體實作環境

	實驗實作環境
處理器	Intel(R) Core(TM) i7-6700 <u>CPU@3.40GHz</u>
Python 版本	3.6.13
虛擬環境	Anaconda 2.0.4
實作套件	Numpy、Matplotlib、Collection、Pandas
作業系統	Microsoft Windows10



圖四、Pandas 套件 Logo

圖五、Matplotlib 套件 Logo

圖六、Numpy 套件 Logo

## C.2. 檔案使用說明

Source code 資料夾中總共有三種檔案，分別為 `generate_data.py`、`linear_training_testing.py` 及 `XOR_training_testing.py`(圖七至圖九)。`generate_data.py` 檔案的功能為生成實驗所需的訓練、測試座標點資料，`generate_data.py` 的檔案中僅提供 `numpy` 資料，並不包含生成 `csv` 檔。`linear_training_testing.py` 檔案的功能是訓練與測試 `linear` 座標點資料、產生訓練結果、產生測試結果、產生其他相關結果及示意圖等...。`XOR_training_testing.py` 檔案的功能是訓練與測試 `XOR` 座標點資料、產生訓練結果、產生測試結果、產生示意圖及產生其他相關結果圖等...。在使用 `linear_training_testing.py` 及 `XOR_training_testing.py` 檔案時，需將 `generate_data.py` 放置在與 `linear_training_testing.py` 或 `XOR_training_testing.py` 檔案的同一個資料夾下，才能使得 `linear_training_testing.py` 及 `XOR_training_testing.py` 檔案能夠正常運行。



圖七、`generate_data.py`



圖八、`linear_training_testing.py`



圖九、`XOR_training_testing.py`

## 2. Experiment setups

### A. Linear 與 XOR 訓練資料與測試資料的生成說明

訓練資料與測試資料的生成函式如圖十和圖十一所示，圖十為生成 `Linear` 座標點資料的函式，函式名稱為 `generator_linear`。`generator_linear` 函式的輸入為資料點的總個數(可為隨機整數)，函式的輸出為資料的座標點及座標點對應的標籤。以輸入  $n=100$  為例，輸出的 `inputs` 的資料維度為  $(100,2)$ ，`labels` 的資料維度為  $(100,1)$ 。圖十一為生成 `XOR` 座標點資料的函式，函式名稱為 `generator_XOR_easy`。`generator_XOR_easy` 函式的輸入為資料點的個數(需為 11 的倍數)，函式的輸出為資料的座標點及座標點對應的標籤。以輸入  $n=11$  為例，輸出的 `inputs` 的資料維度為  $(21,2)$ ，`labels` 的資料維度為  $(21,1)$ 。`generator_XOR_easy` 函式的  $n$  代表的是左上至右下那條線的點數量，因此 `return` 的點數量總共為  $2*n-1$ 。

```
def generator_linear(n=100):
    pts = np.random.uniform(0, 1, (n, 2))
    inputs, labels = [], []
    for pt in pts:
        distance = (pt[0]-pt[1])/1.414
        inputs.append([pt[0],pt[1]])
        if pt[0]>pt[1]:
            labels.append(0)
        else:
            labels.append(1)
    inputs = np.array(inputs)
    labels = np.array(labels).reshape(n, 1)
    return inputs, labels
```

圖十、Linear 座標點資料生成函式

```
def generator_XOR_easy(n = 11):
    inputs, labels = [], []
    for i in range(n):
        prod_number = 11/n
        inputs.append([0.1*prod_number*i,0.1*prod_number*i])
        labels.append(0)
        if 0.1*prod_number*i==0.5:
            continue
        inputs.append([0.1*prod_number*i,1-0.1*prod_number*i])
        labels.append(1)
    inputs = np.array(inputs)
    labels = np.array(labels).reshape([n*2-1, 1])
    return inputs, labels
```

圖十一、XOR 座標點資料生成函式

## B. 模型整體架構與細節

### B.1. Activation Function

在模型架構中，本實驗使用兩種 Activation Function，分別為 Sigmoid 及 ReLU，這兩種在模型內是具有選擇性的，以下分別為 Sigmoid 及 ReLU 介紹以及微分函數。

#### B.1.1. Sigmoid

Sigmoid 是一種 S 型曲線的可微分函數，他的作用是可以將輸入的連續實值變換為 0 和 1 之間的輸出，因此 Sigmoid 是適合用來進行二分類的 Activation Function。式(1)及式(2)為 Sigmoid 的公式以及微分結果，其可以對應到圖十二及圖十三的程式碼函式。

$$y = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad \text{式(1)}$$

$$\frac{d}{dx} = \frac{d}{dx} \text{sigmoid}(x) = y * (1 - y) \quad \text{式(2)}$$

```
def sigmoid(Z):
    return 1/(1+np.exp(-Z))
```

圖十二、用於 Forward 的 Sigmoid

```
def sigmoid_backward(dA, Z):
    sig = sigmoid(Z)
    return dA * sig * (1 - sig)
```

圖十三、用於 Backward 的 Sigmoid

#### B.1.2. ReLU

在 ReLU 公式中，任何一個小於 0 的值都會被歸類到 0，反之當任何一個值大於 0 時，其結果會等於該數字，因此這種公式的特性有助於減低梯度消失的發生，並能夠加快收斂速度，公式可以對應到式(3)。其可以對應到圖十四及圖十五的程式碼函式。但由於 ReLU 函式是不可微分的，因此會用到次梯度的概念，其結果為，當 x 小於 0 時，其導數會為 0；當 x 大於 0 時，其導數會為 1。如果 x 等於 0 時的次梯度為 0 到 1 之間的任意數。

$$ReLU(x) = \max(0, x) \quad \text{式(3)}$$

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

圖十四、用於 Forward 的 ReLU

```
def relu_backward(dA, Z):
    dZ = np.array(dA, copy = True)
    dZ[Z <= 0] = 0
    return dZ
```

圖十五、用於 Backward 的 ReLU

## B.2. Loss Function

在模型架構中，本實驗使用到的 Loss Function 為 BinaryCrossentropy，計算 Loss 值的公式可以對應到式(4)，式(4)對應到的函式為圖十六。在式(4)中， $y_i$  為真實標記的分佈，例如紅色為 0，藍色為 1， $p(y_i)$  為訓練後的模型預測標記分佈。BinaryCrossentropy 可以衡量  $y_i$  與  $p(y_i)$  的相似性。而用於反向傳播中計算需要多少變化量的 Loss Function 公式則可以對應到式(5)，式(5)對應到的公式為圖十七。式(5)中的  $y_i$  等於式(4)中的  $y_i$ 。式(5)中的  $\hat{y}_i$  等於式(4)中的  $p(y_i)$ 。

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \quad \text{式(4)}$$

$$\frac{\partial \text{loss}}{\partial y} = -\sum_{i=1}^n \frac{\hat{y}_i}{y_i} - \frac{1 - \hat{y}_i}{1 - y_i} \quad \text{式(5)}$$

```
def get_loss(Y_hat, Y):
    m = Y_hat.shape[1]
    cost = -1 / m * (np.dot(Y, np.log(Y_hat).T) + np.dot(1 - Y, np.log(1 - Y_hat).T))
    return np.squeeze(cost)
```

圖十六、計算 Loss 值的函式

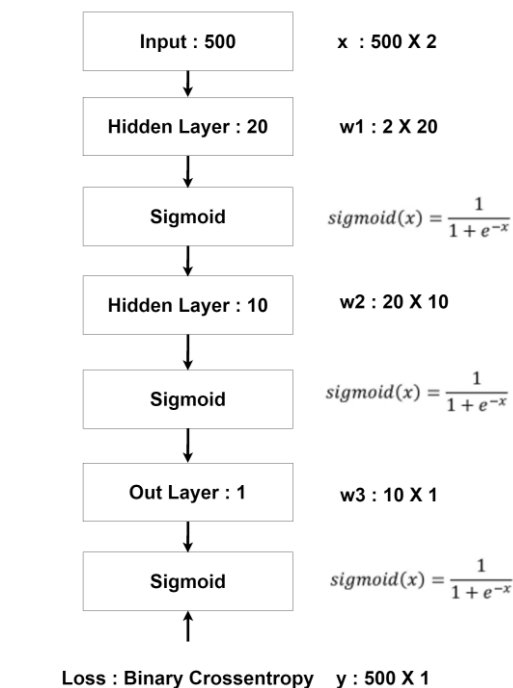
```
dA_prev = - (np.divide(Y, Y_hat) - np.divide(1 - Y, 1 - Y_hat))
```

圖十七、得到反向傳播中需要多少變化量

## B.3. Neural network

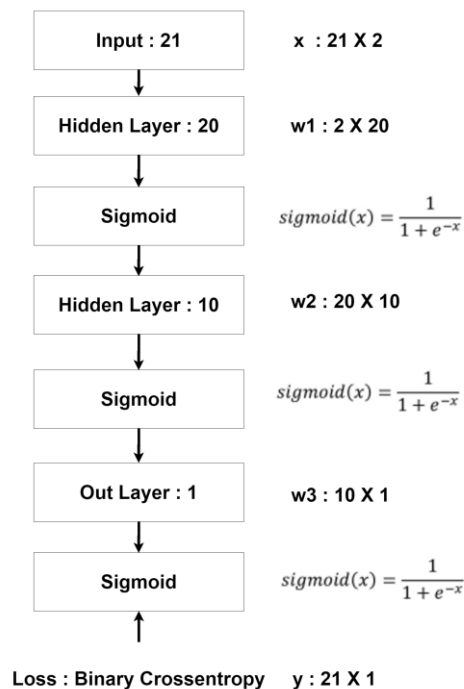
本實驗在進行 Linear 及 XOR 分類時，使用的 neuron 數量、layer 數量、Activation Function、Loss Function 都是一樣的。圖十八為本實驗的 Linear 模型架構，圖十九為本實驗的 XOR 的模型架構。可以分別看到圖十八與圖十九中不同的部份在於隨機的權重值、誤差值及輸入模型的資料數量。剩餘的部份包含 Hidden Layer、Activation Function、Loss Function 等都相同。





$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

圖十八、Linear 模型架構圖



$$-\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

圖十九、XOR 模型架構圖

### B.3.1. 初始化、定義權重(Weights)與誤差(Bias)

本實驗在此階段中透過圖二十的 TwoLayerNet 變數定義模型各層的 input 維度、output 維度及使用的 activation function。當這些資訊都被定義之後，就可將這些資訊輸入至圖二十一中的函式中。圖二十一函式總共需要處理兩件事情。第一件事情是根據剛剛定義的模型各層維度資訊，隨機配送 Weight 與 Bias 給模型各層(圖二十一的紅色框框)。第二件事情是將這些在每一層定義好的 Weight 與 Bias 儲存至 params\_values 字典變數(圖二十一的紅色框框)。因此圖二十一 init\_layer 函式內的 nn\_architecture 變數可為圖二十的 TwoLayerNet 變數，圖二十一 params\_values 字典變數則使用 W 及 b 名稱分別儲存各層的 Weight 及 bias，因此模型第一層的權重與誤差分別為 W1 及 b1，模型第二層的權重與誤差為 W2 及 b2，剩餘各層的權重與誤差則以此類推。最後 init\_layers 函式會將 params\_values 字典變數回傳至主程式中。

```
TwoLayerNet = [
{"input_dim": 2, "output_dim": 20, "activation": "sigmoid"},
{"input_dim": 20, "output_dim": 10, "activation": "sigmoid"},
{"input_dim": 10, "output_dim": 1, "activation": "sigmoid"},
]
```

圖二十、定義模型各層輸出入

```
def init_layers(nn_architecture, seed = 99):
    # 設定隨機種子
    np.random.seed(seed)
    # neural network的數量
    number_of_layers = len(nn_architecture)
    # 當前權重w0 w1 及誤差b1 b2的儲存地方
    params_values = {}

    # 開始隨機設置每一層的權重跟誤差，初始設定的值需要小一點會比較好
    for idx, layer in enumerate(nn_architecture):
        # w1 w2 b1 b2的數字，從1開始
        layer_idx = idx + 1

        layer_input_size = layer["input_dim"]
        layer_output_size = layer["output_dim"]

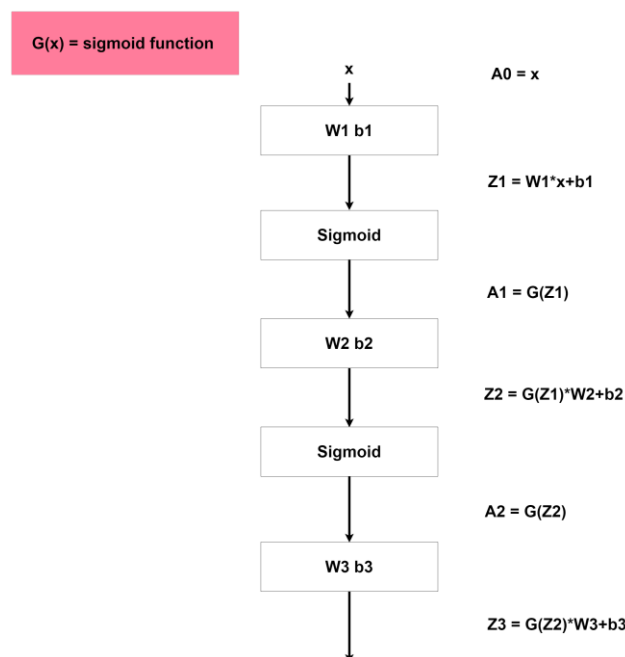
        params_values['W' + str(layer_idx)] = np.random.randn(
            layer_output_size, layer_input_size) * 0.1
        params_values['b' + str(layer_idx)] = np.random.randn(
            layer_output_size, 1) * 0.1

    return params_values
```

圖二十一、初始化模型各層權重及誤差

### B.3.2. Forward propagation

以圖二十四為例，本實驗在這 forward propagation 階段中有兩件需要處理的事情。第一件事情是將 x 值(input)經過各層進行運算(圖二十四的紅色框框，single\_forward\_propagation 的韓式則可以對應至圖二十三)。第二件事情則是將 x 與各層 W、b 及 Sigmoid 運算完的結果，儲存至 memory 字典變數中(圖二十四的藍色框框)。實作第二件事情的目的則是為了方便 backward propagation 階段進行運算。運算結果的儲存方法則可以對應至圖二十二，如果該層運算結果是關於 Activation Function 的運算結果，其運算結果在字典裡的名稱就會是 A+該層數字。如果該層運算結果是關於權重跟誤差的運算結果，其運算結果在字典裡的名稱就會是 Z+該層數字。圖二十三則定義了各層的運算方式(權重與值的矩陣乘法、Activation Function 的數值轉換等...)。最後當事情處理完成之後，圖二十四的函式就會將預測結果(A\_curr)與各層運算結果(memory 字典變數)回傳至主程式中。



圖二十二、Forward propagation 示意圖

```
def single_forward_propagation(A_prev, W_curr, b_curr, activation="relu"):
    Z_curr = np.dot(W_curr, A_prev) + b_curr

    if activation is "relu":
        activation_func = relu
    elif activation is "sigmoid":
        activation_func = sigmoid
    else:
        activation_func = normal

    return activation_func(Z_curr), Z_curr
```

圖二十三、單層 forward 運算

```
def forward_propagation(X, params_values, nn_architecture):
    memory = {}
    A_curr = X

    for idx, layer in enumerate(nn_architecture):
        layer_idx = idx + 1
        A_prev = A_curr

        activ_function_curr = layer["activation"]
        W_curr = params_values["W" + str(layer_idx)]
        b_curr = params_values["b" + str(layer_idx)]
        A_curr, Z_curr = single_forward_propagation(A_prev, W_curr, b_curr, activ_function_curr)

        memory["A" + str(idx)] = A_prev
        memory["Z" + str(layer_idx)] = Z_curr

    return A_curr, memory
```

圖二十四、forward propagation 運算

### B.3.3. Backward propagation

在 backward propagation 階段中，總共有三個部份的事情需要處理，第一件事情是將剛剛從 forward propagation 得到的預測結果，與實際結果進行計算，以得到預測結果與實際結果相差多少(圖二十六的紅色框框)。當第一件事情處理完之後，第二件事就可以透過相差的結果逐一去對之前運算得到的 A0 至 A2 及 Z1 至 Z3 進行運算，以得到 Weight 與 bias 在各層間需要有多少的變化量(圖二十六的藍色框框，single\_backward\_propagation 函式可以對應至圖二十五)，第三件事就是將每一層需要變化多少 Weight 與 bias 值儲存至 grad\_values 字典變數中，以供後續更新每一層的 Weight 跟 bias(圖二十六的綠色框框)。Weight 跟 bias 變化量在 grad\_values 字典變數的儲存方式為，第一層要變化的 Weight 跟 bias 就會分別儲存在 dW1 及 db1 的名稱中、第二層要變化的 Weight 跟 bias 就會分別儲存在 dW2 及 db2 的名稱中，剩餘的層以此類推。最後再將 grad\_values 字典變數回傳至主程式。

```
def single_backward_propagation(dA_curr, W_curr, b_curr, Z_curr, A_prev, activation="relu"):
    m = A_prev.shape[1]

    if activation is "relu":
        backward_activation_func = relu_backward
    elif activation is "sigmoid":
        backward_activation_func = sigmoid_backward
    else:
        backward_activation_func = normal_backward

    dZ_curr = backward_activation_func(dA_curr, Z_curr)

    dW_curr = np.dot(dZ_curr, A_prev.T) / m
    db_curr = np.sum(dZ_curr, axis=1, keepdims=True) / m
    dA_prev = np.dot(W_curr.T, dZ_curr)

    return dA_prev, dW_curr, db_curr
```

圖二十五、單層 backward propagation 運算

```

def backward_propagation(Y_hat, Y, memory, params_values, nn_architecture):
    grads_values = {}

    m = Y.shape[1]
    Y = Y.reshape(Y_hat.shape)

    dA_prev = - (np.divide(Y, Y_hat) - np.divide(1 - Y, 1 - Y_hat))

    for layer_idx_prev, layer in reversed(list(enumerate(nn_architecture))):
        layer_idx_curr = layer_idx_prev + 1
        activ_function_curr = layer["activation"]

        dA_curr = dA_prev

        A_prev = memory["A" + str(layer_idx_prev)]
        Z_curr = memory["Z" + str(layer_idx_curr)]

        W_curr = params_values["W" + str(layer_idx_curr)]
        b_curr = params_values["b" + str(layer_idx_curr)]

        dA_prev, dW_curr, db_curr = single_backward_propagation(
            dA_curr, W_curr, b_curr, Z_curr, A_prev, activ_function_curr)

        grads_values["dW" + str(layer_idx_curr)] = dW_curr
        grads_values["db" + str(layer_idx_curr)] = db_curr

    return grads_values

```

圖二十六、backward propagation 運算

### B.3.4. Update Weights

在此階段中，要處理的事情為更新每一層的權重(Weight)與誤差(bias)。因此在此函式中，總共會需要輸入四個變數，分別為 `params_values`、`grads_values`、`nn_architecture` 及 `learning_rate`。`params_values` 為 forward propagation 階段處理回傳的結果，其定義每一層當前的權重與誤差。`grads_values` 為 backward propagation 處理回傳的字典變數，其定義每一層需要變化多少的權重及誤差。`nn_architecture` 為需要更新幾層的權重與誤差。`learning_rate` 則為需要下降多少比率的概念。最後當各層的權重與誤差更新完成之後，函式就會將 `params_values` 字典變數回傳至主程式，以進入至下一個 epochs 進行處理。如圖二十七所示

```

def update_model_weight_bias(params_values, grads_values, nn_architecture, learning_rate):
    for layer_idx, layer in enumerate(nn_architecture, 1):
        params_values["W" + str(layer_idx)] -= learning_rate * grads_values["dW" + str(layer_idx)]
        params_values["b" + str(layer_idx)] -= learning_rate * grads_values["db" + str(layer_idx)]

    return params_values

```

圖二十七、backward propagation 運算

## 3. Results of testing

Results of testing 的各實驗參數與結果等可以對應至表二，表二的實驗參數都是實作結果比較好的參數。表二包含實驗中用到的 learning rate、epochs、activation function、模型各層的數量、Training data 的點數量、Testing data 的點數量、Training Accuracy 及 Testing Accuracy 等。

表二、各項實驗參數

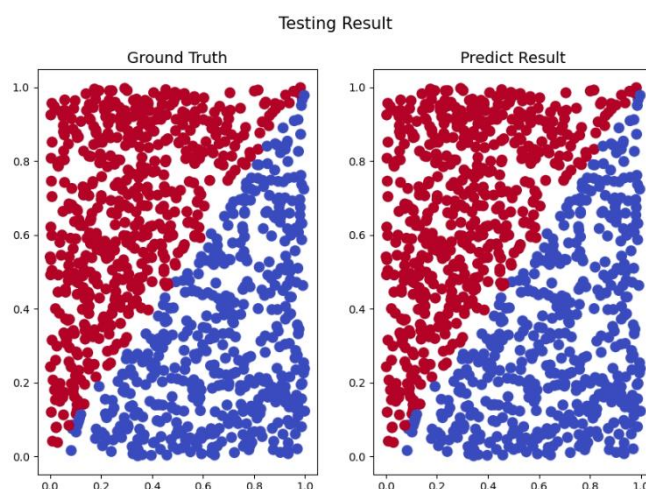
	Learn ing	Epochs	Activation Function	第一 層	第二 層	第三 層	Training data 的	Testing data 的	Training Accuracy	Testing Accuracy
--	--------------	--------	------------------------	---------	---------	---------	--------------------	-------------------	----------------------	---------------------

	rate			ouput 數量	ouput 數量	ouput 數量	點數量	點數量		
Linear	0.5	6000	Sigmoid	20	10	1	1000	1000	1.00	100%
XOR	0.5	6000	Sigmoid	20	10	1	43	43	1.00	100%

## A. Testing Ground Truth 與 Predict Result 之間的比較

### A.1. Linear 分類

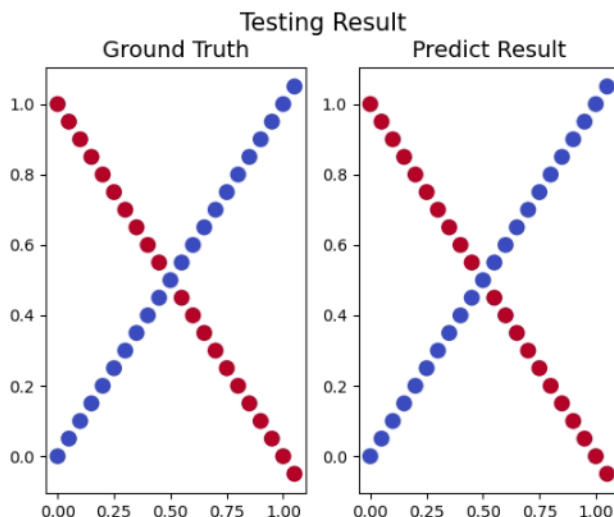
圖二十八為 Linear 分類中 Ground Truth 與 Predict Result 之間的比較圖。本次實驗使用 1000 個點座標進行分類，其中紅色的點數量為 471 個點，藍色的點數量為 529 個點。如圖二十八所示，本次實驗可完全分類紅藍點座標。除此之外，本實驗可另外設定訓練與測試的點數量。



圖二十八、Linear 分類的 Ground Truth 與 Predict Result 比較圖

### A.2. XOR 分類

圖二十九為 XOR 分類中 Ground Truth 與 Predict Result 之間的比較圖。本次實驗使用 43 個點座標進行分類，其中紅色的點數量為 21 個點，藍色的點數量為 22 個點。如圖二十九所示，本次實驗可完全分類紅藍點座標。除此之外，本實驗可另外設定訓練與測試的點數量。



圖二十九、XOR 分類的 Ground Truth 與 Predict Result 比較圖



## B. Testing Accuracy 與 Testing 預測結果

## B.1. Linear 分類

圖三十表示的為 Testing accuracy，預測的點座標分佈則可對應到圖二十八，而圖三十一及圖三十二則分別為圖二十八中 Ground Truth 的值與 Predict Result 的預測結果，其座標點的個數為 1000 個。

Testing accuracy: 1.000000

圖三十、Testing accuracy 的值

[illegible]

### 圖三十一、Ground Truth 的值

[illegible]

### 圖三十二、Predict Result 的值

## B.2. XOR 分類

圖三十三表示的為 Testing accuracy，預測的點座標分佈則可對應到圖二十九，而圖三十四及圖三十五則分別為圖二十九中 Ground Truth 的值與 Predict Result 的預測結果，其座標點的個數為 43 個。

Testing accuracy: 1.000000

圖三十三、Testing accuracy 的值

Ground Truth: [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0.  
1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]

### 圖三十四、Ground Truth 的值

```
Predict Result: [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0.
 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
```

圖三十五、Predict Result 的值

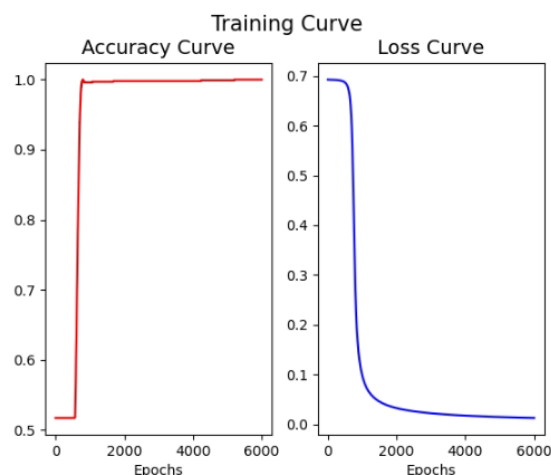
## C. Loss 與 Accuracy 的 Training 學習曲線

### C.1. Linear 分類

如圖三十七所示，模型在 1800 個 epochs 時就達到完全擬合狀態。Loss 值最低可到達 0.0125 甚至是更低的情況，如需要更低的 Loss 值，可再將 epochs 數量拉高。本實驗將 epochs 數量拉到 6000 的原因是因為需要觀察在後續的 epochs 中是否會有震盪的情形發生，但依據圖三十七來看，本次實驗的模型的訓練曲線是很穩定的。圖三十六則為訓練過程中的 epochs、loss 跟 accuracy，因其總共有 6000 個 epochs，因此本實驗僅取後面的 epochs 訓練圖做顯示。

|               |                |                    |
|---------------|----------------|--------------------|
| Epoch: 005900 | Loss: 0.012665 | Accuracy: 1.000000 |
| Epoch: 005910 | Loss: 0.012649 | Accuracy: 1.000000 |
| Epoch: 005920 | Loss: 0.012632 | Accuracy: 1.000000 |
| Epoch: 005930 | Loss: 0.012616 | Accuracy: 1.000000 |
| Epoch: 005940 | Loss: 0.012600 | Accuracy: 1.000000 |
| Epoch: 005950 | Loss: 0.012584 | Accuracy: 1.000000 |
| Epoch: 005960 | Loss: 0.012568 | Accuracy: 1.000000 |
| Epoch: 005970 | Loss: 0.012552 | Accuracy: 1.000000 |
| Epoch: 005980 | Loss: 0.012536 | Accuracy: 1.000000 |
| Epoch: 005990 | Loss: 0.012520 | Accuracy: 1.000000 |

圖三十六、Epoch、Loss 與 Accuracy 的值



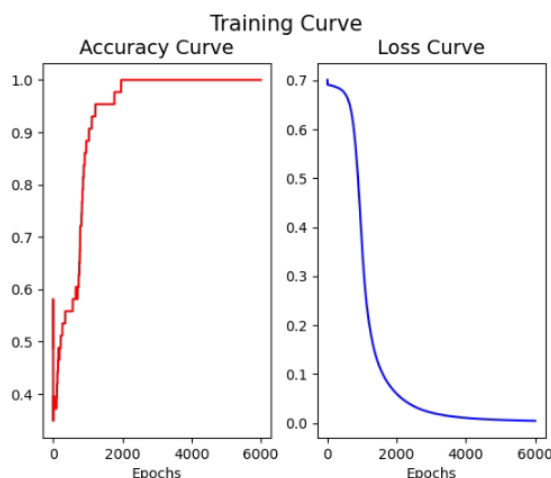
圖三十七、訓練學習曲線

### C.2. XOR 分類

如圖三十九所示，模型在 2000 個 epochs 時就達到擬合狀態。Loss 值最低可到達 0.004841 甚至是更低的情況，如需要更低的 Loss 值，可再將 epochs 數量拉高。依據圖三十九來看，本次實驗的模型的訓練曲線是很穩定的。圖三十八則為訓練過程中的 epochs、loss 跟 accuracy，因其總共有 6000 個 epochs，因此本實驗僅取後面的 epochs 訓練圖做顯示。

|               |                |                    |
|---------------|----------------|--------------------|
| Epoch: 005780 | Loss: 0.005165 | Accuracy: 1.000000 |
| Epoch: 005790 | Loss: 0.005148 | Accuracy: 1.000000 |
| Epoch: 005800 | Loss: 0.005132 | Accuracy: 1.000000 |
| Epoch: 005810 | Loss: 0.005116 | Accuracy: 1.000000 |
| Epoch: 005820 | Loss: 0.005100 | Accuracy: 1.000000 |
| Epoch: 005830 | Loss: 0.005084 | Accuracy: 1.000000 |
| Epoch: 005840 | Loss: 0.005068 | Accuracy: 1.000000 |
| Epoch: 005850 | Loss: 0.005052 | Accuracy: 1.000000 |
| Epoch: 005860 | Loss: 0.005037 | Accuracy: 1.000000 |
| Epoch: 005870 | Loss: 0.005021 | Accuracy: 1.000000 |
| Epoch: 005880 | Loss: 0.005006 | Accuracy: 1.000000 |
| Epoch: 005890 | Loss: 0.004990 | Accuracy: 1.000000 |
| Epoch: 005900 | Loss: 0.004975 | Accuracy: 1.000000 |
| Epoch: 005910 | Loss: 0.004960 | Accuracy: 1.000000 |
| Epoch: 005920 | Loss: 0.004945 | Accuracy: 1.000000 |
| Epoch: 005930 | Loss: 0.004929 | Accuracy: 1.000000 |
| Epoch: 005940 | Loss: 0.004915 | Accuracy: 1.000000 |
| Epoch: 005950 | Loss: 0.004900 | Accuracy: 1.000000 |
| Epoch: 005960 | Loss: 0.004885 | Accuracy: 1.000000 |
| Epoch: 005970 | Loss: 0.004870 | Accuracy: 1.000000 |
| Epoch: 005980 | Loss: 0.004855 | Accuracy: 1.000000 |
| Epoch: 005990 | Loss: 0.004841 | Accuracy: 1.000000 |

圖三十八、Epoch、Loss 與 Accuracy 的值

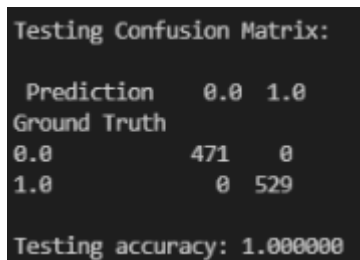


圖三十九、訓練學習曲線

## D. Testing predict result 的 Confusion Matrix

### D.1. Linear 分類

圖四十顯示的資訊為圖二十八 Predict Result 與 Ground Truth 的 Confusion Matrix。如圖所示，紅色點座標(標籤為 0)的數量為 471 個，藍色點座標(標籤為 1)的數量為 529 個。訓練資料的座標點數量也為 43 個，因此點座標的分佈算是較為平均的狀況。而且分類結果如 Confusion Matrix 所示，分類較為完全。



```
Testing Confusion Matrix:

Prediction    0.0  1.0
Ground Truth
0.0           471   0
1.0            0  529

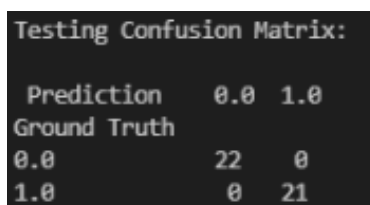
Testing accuracy: 1.000000
```

A screenshot of a terminal window showing the testing confusion matrix for a linear classification model. The matrix is a 2x2 grid with 'Prediction' on the columns and 'Ground Truth' on the rows. The values are 471 for (0,0), 0 for (0,1), 0 for (1,0), and 529 for (1,1). The testing accuracy is 1.000000.

圖四十、Linear Predict Result 與 Ground Truth 的 Confusion Matrix

### D.2. XOR 分類

圖四十一顯示的資訊為圖二十九 Predict Result 與 Ground Truth 的 Confusion Matrix。如圖所示，紅色點座標(標籤為 0)的數量為 22 個，藍色點座標(標籤為 1)的數量為 21 個。訓練資料的座標點數量也為 43 個，因此點座標的分佈算是較為平均的狀況。而且分類結果如 Confusion Matrix 所示，分類較為完全。



```
Testing Confusion Matrix:

Prediction    0.0  1.0
Ground Truth
0.0           22   0
1.0            0  21
```

A screenshot of a terminal window showing the testing confusion matrix for an XOR classification model. The matrix is a 2x2 grid with 'Prediction' on the columns and 'Ground Truth' on the rows. The values are 22 for (0,0), 0 for (0,1), 0 for (1,0), and 21 for (1,1).

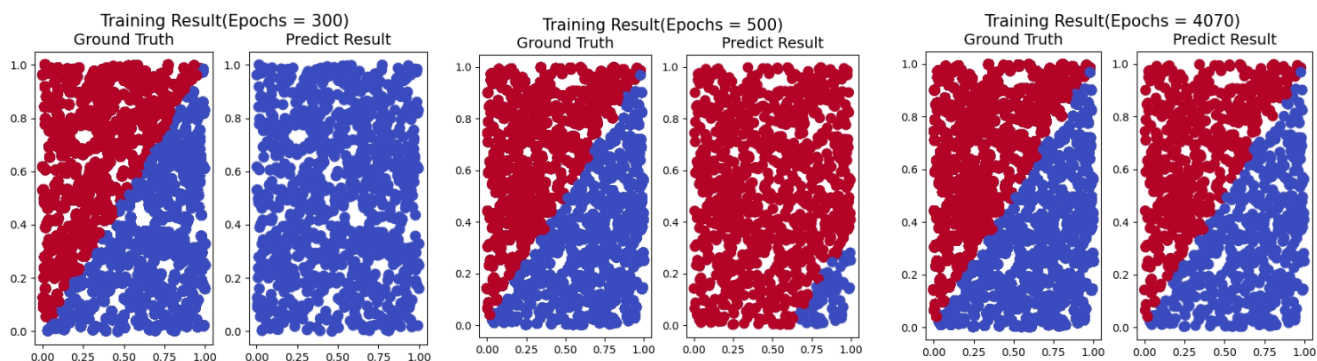
圖四十一、XOR Predict Result 與 Ground Truth 的 Confusion Matrix

## E. 隨著 Training Epochs 變化的 Predict Result

### E.1. Linear 分類

圖四十二顯示的資訊為不同 epochs 下的辨識率以及猜測答案。因此本實驗就可依據此數據進行模型參數的調整，圖四十二最左邊的表示的是 Epochs 等於 300 的預測結果與答案，圖四十二中間的圖表示的是 Epochs 等於 500 的預測結果與答案，圖四十二最右邊的表示的是 Epochs 等於 4070 的預測結果與答案，顯示出在經過 4070 個 epochs 之後已經可達完全分類。

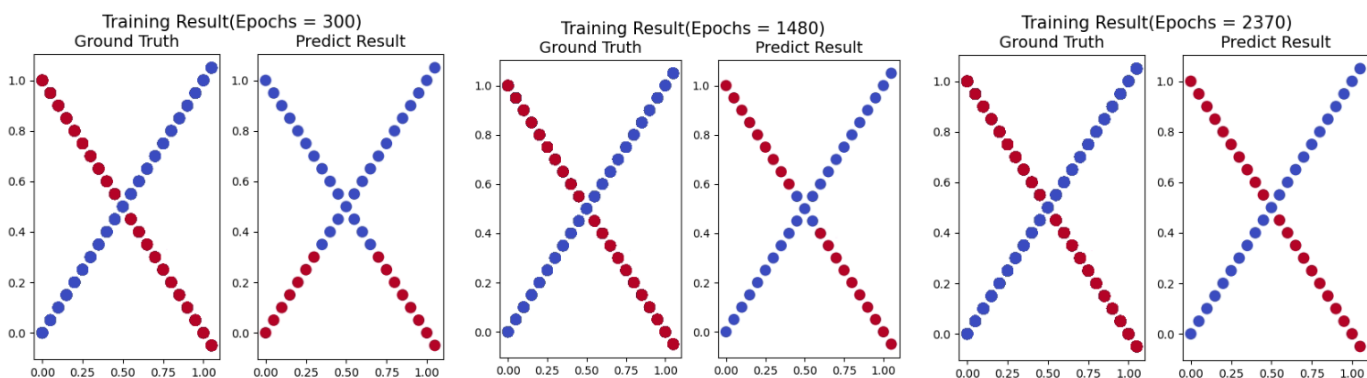




圖四十二、不同 Epoch 下的比較圖

## E.2. XOR 分類

圖四十三顯示的資訊為不同 epochs 下的辨識率以及猜測答案。因此本實驗就可依據此數據進行模型參數的調整，圖四十三最左邊的表示的是 Epochs 等於 300 的預測結果與答案，圖四十三中間的圖表示的是 Epochs 等於 1480 的預測結果與答案，圖四十三最右邊的表示的是 Epochs 等於 2370 的預測結果與答案，顯示出在經過 2370 個 epochs 之後已可達完全分類。



圖四十三、不同 Epoch 下的比較圖

## 4. Discussion

在此階段中，不同參數下的結果都會在這部份進行比較。各參數變化的比較基準可以參考至表二。

### A. 不同 Learning rate 下的模型 Training 與 Testing 情況

在此部份中，本實驗僅對 Learning rate 進行改變，其餘的參數皆與表二無異。接下來以表三及表四分別進行 Linear 與 XOR 分類的討論。在表三中，第一列顯示的為 Learning rate 的數值，第一欄顯示的為 Training Accuracy 與 Testing Accuracy，欄與列對應的數值則為 Training accuracy 與 Testing accuracy。可以看到表三中的各項 Learning rate 都能將結果分類至 95% 以上甚至是 100% 的 accuracy。

表三、Linear 分類改變 Learning rate 後 Training accuracy 與 Testing accuracy 的值

|  | 1e-1 | 2e-1 | 3e-1 | 4e-1 | 5e-1 | 6e-1 | 7e-1 | 8e-1 | 9e-1 |
|--|------|------|------|------|------|------|------|------|------|
|--|------|------|------|------|------|------|------|------|------|

|                   |       |       |       |       |      |       |       |       |       |
|-------------------|-------|-------|-------|-------|------|-------|-------|-------|-------|
| Training Accuracy | 99.5% | 99.4% | 99.9% | 100%  | 100% | 99.9% | 99.7% | 100%  | 99.9% |
| Testing Accuracy  | 99.1% | 99.6% | 99.9% | 99.8% | 100% | 99.9% | 99.8% | 99.9% | 99.8% |

在表四中，第一列顯示的同樣為 Learning rate 的數值，第一欄顯示的為 Training Accuracy 與 Testing Accuracy，欄與列對應的數值則為 Testing 的 accuracy。但在此表中可發現，當 Learning rate 調整至 0.1 之後，反而會使得 accuracy 降至 93%。但在表四中除 0.1 外的 Learning rate 都能使得分類 accuracy 達到 95% 以上甚至是 100%。

**表四、XOR 分類改變 Learning rate 後 Training accuracy 與 Testing accuracy 的值**

|                   | 1e-1   | 2e-1 | 3e-1 | 4e-1 | 5e-1 | 6e-1 | 7e-1 | 8e-1 | 9e-1 |
|-------------------|--------|------|------|------|------|------|------|------|------|
| Training Accuracy | 93.02% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| Testing Accuracy  | 93.02% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |

## B. 不同 Hidden Unit 下的模型 Training 與 Testing 情況

### B.1. 改變第一層 Hidden Unit

在此部份中，本實驗僅對第一層 Hidden Unit 進行改變，其餘的參數皆與表二無異。接下來以表五及表六分別進行 Linear 與 XOR 分類的討論。在表五中，第一列顯示的為第一層 Hidden Unit 的數量，第一欄顯示的為 Training Accuracy 與 Testing Accuracy，欄與列對應的數值則為 Training accuracy 與 Testing accuracy。可以看到表五中的各項第一層 Hidden Unit 都能將結果分類至 95% 以上甚至是 100% 的 accuracy。

**表五、Linear 分類改變第一層 Hidden Unit 後 Training accuracy 與 Testing accuracy 的值**

|                   | 100   | 90    | 80    | 70    | 60    | 50    | 40    | 30    | 20   |
|-------------------|-------|-------|-------|-------|-------|-------|-------|-------|------|
| Training Accuracy | 99.2% | 99.8% | 99.6% | 99.9% | 99.3% | 99.9% | 99.8% | 100%  | 100% |
| Testing Accuracy  | 99%   | 99.8% | 99.8% | 99.9% | 99.8% | 99.5% | 99.9% | 99.8% | 100% |

在表六中，第一列顯示的同樣為第一層 Hidden Unit 的數值。第一欄顯示的為 Training Accuracy 與 Testing Accuracy。欄與列對應的數值則為 Training accuracy 與 Testing accuracy。

**表六、XOR 分類改變第一層 Hidden Unit 後 Training accuracy 與 Testing accuracy 的值**

|                   | 100  | 90    | 80    | 70     | 60    | 50    | 40    | 30    | 20   |
|-------------------|------|-------|-------|--------|-------|-------|-------|-------|------|
| Training Accuracy | 99%  | 99.8% | 99.8% | 99.9%  | 99.8% | 99.5% | 99.9% | 99.8% | 100% |
| Testing Accuracy  | 100% | 100%  | 100%  | 97.67% | 100%  | 100%  | 100%  | 100%  | 100% |

## B.2. 改變第二層 Hidden Unit

在此部份中，本實驗僅對第二層 Hidden Unit 進行改變，其餘的參數皆與表二無異。接下來將針對 Linear 分類與 XOR 分類進行各別的討論。在表七中，第一列顯示的為第二層 Hidden Unit 的數量，第一欄顯示的為分類項目，欄與列對應的數值則為 Training 的 accuracy。表七顯示的為 Training accuracy，表八顯示的為 Testing accuracy。

表七、改變第二層 Hidden Unit 後 Training accuracy 的值

|        | 11    | 12    | 13    | 14    | 15    | 16   | 17    | 18    | 19    |
|--------|-------|-------|-------|-------|-------|------|-------|-------|-------|
| Linear | 99.9% | 99.9% | 99.7% | 99.8% | 99.9% | 100% | 99.8% | 99.9% | 99.9% |
| XOR    | 100%  | 100%  | 100%  | 100%  | 100%  | 100% | 100%  | 100%  | 100%  |

在表八中，第一列顯示的同樣為第二層 Hidden Unit 的數值，第一欄顯示的為分類項目，欄與列對應的數值則為 Testing 的 accuracy。

表八、改變第二層 Hidden Unit 後 Testing accuracy 的值

|        | 11    | 12    | 13    | 14    | 15    | 16    | 17    | 18    | 19   |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|------|
| Linear | 99.9% | 99.9% | 99.6% | 99.9% | 99.9% | 99.9% | 99.8% | 99.9% | 100% |
| XOR    | 100%  | 100%  | 100%  | 100%  | 100%  | 100%  | 100%  | 100%  | 100% |

## C. 不同 Activation Function 下的模型 Training 與 Testing 情況

在此部份中，本實驗僅對 Activation Function 進行改變，其餘的參數皆與表二無異。接下來將針對 Linear 分類與 XOR 分類進行各別的討論。在表十一中，第一列顯示的為 Activation Function 的使用狀況，第一欄顯示的為分類項目。以第一列第二欄那一格為例，第一個數字表示的是第一層使用的 Activation Function，第二個數字表示的是第二層使用的 Activation Function，第三個數字表示的是第三層使用的 Activation Function。欄與列對應的數值則為 Training 的 accuracy。表十一顯示的為 Training accuracy，表十二顯示的為 Testing accuracy。可以看到表十一與表十二中，當模型不使用 activation function 時，模型的 accuracy 會下降許多。

表十一、改變 Activation Function 後 Training accuracy 的值

|        | Sigmoid | ReLU    | Sigmoid | Sigmoid | ReLU    | ReLU    | Sigmoid | ReLU  | 不使用    |
|--------|---------|---------|---------|---------|---------|---------|---------|-------|--------|
|        | Sigmoid | Sigmoid | ReLU    | Sigmoid | ReLU    | Sigmoid | ReLU    | ReLU  | 不使用    |
|        | Sigmoid | Sigmoid | Sigmoid | ReLU    | Sigmoid | ReLU    | ReLU    | ReLU  | 不使用    |
| Linear | 100%    | 99.9%   | 99.8%   | 99.7%   | 98.2%   | 99.9%   | 99.4%   | 99.7% | 50.2%  |
| XOR    | 100%    | 100%    | 100%    | 100%    | 100%    | 100%    | 100%    | 100%  | 51.62% |

在表十二中，第一列顯示的同樣為 Activation Function 的使用狀況，第一欄顯示的為分類項目，欄與列對應的數值則為 Testing 的 accuracy。

表十二、改變 Activation Function 後 Testing accuracy 的值

|  | Sigmoid | ReLU | Sigmoid | Sigmoid | ReLU | ReLU | Sigmoid | ReLU | 不使用 |
|--|---------|------|---------|---------|------|------|---------|------|-----|
|--|---------|------|---------|---------|------|------|---------|------|-----|

|        |                    |                    |                 |                 |                 |                 |              |              |            |
|--------|--------------------|--------------------|-----------------|-----------------|-----------------|-----------------|--------------|--------------|------------|
|        | Sigmoid<br>Sigmoid | Sigmoid<br>Sigmoid | ReLU<br>Sigmoid | Sigmoid<br>ReLU | ReLU<br>Sigmoid | Sigmoid<br>ReLU | ReLU<br>ReLU | ReLU<br>ReLU | 不使用<br>不使用 |
| Linear | 100%               | 99.9%              | 99.8%           | 99.7%           | 98.2%           | 99.9%           | 99.4%        | 99.7%        | 50.2%      |
| XOR    | 100%               | 100%               | 100%            | 100%            | 100%            | 100%            | 100%         | 100%         | 51.62%     |

#### D. 不同訓練與測試資料點數量下的模型 Training 與 Testing 情況

在此部份中，本實驗僅對訓練與測試資料點數量進行改變，其餘的參數皆與表二無異。接下來將針對 Linear 進行討論。在表十三中，第一列顯示的為訓練與測試資料點的改變狀況。以第一列第二欄那一格為例，第一個數字表示的是 training data 的點數量，第二個數字表示的是 testing data 的點數量。第一欄顯示的為 Training 的 accuracy 與 Testing 的 Accuracy。

表十三、改變訓練與測試資料點數量後 Training accuracy 的值

|                   |            |            |            |            |            |            |            |            |            |             |
|-------------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|-------------|
|                   | 500<br>500 | 500<br>600 | 600<br>600 | 600<br>700 | 700<br>700 | 700<br>800 | 800<br>800 | 800<br>900 | 900<br>900 | 900<br>1000 |
| Training Accuracy | 100%       | 99.6%      | 100%       | 99.8%      | 99.7%      | 100%       | 100%       | 99.75%     | 99.77%     | 99.88%      |
| Testing Accuracy  | 100%       | 99.8%      | 100%       | 99.8%      | 99.7%      | 99.7%      | 99.7%      | 99.77%     | 99.88%     | 99.80%      |

在表十四中，第一列顯示的同樣為改變訓練與測試資料點數量的狀況，第一欄顯示的為分類項目，欄與列對應的數值則為 Testing 的 accuracy。

表十四、改變訓練與測試資料點數量後 Testing accuracy 的值

|                   |          |          |          |          |          |          |
|-------------------|----------|----------|----------|----------|----------|----------|
|                   | 21<br>21 | 21<br>43 | 43<br>43 | 43<br>63 | 63<br>63 | 63<br>87 |
| Training Accuracy | 100%     | 100%     | 100%     | 100%     | 100%     | 100%     |
| Testing Accuracy  | 100%     | 95.34%   | 100%     | 100%     | 100%     | 100%     |

## 5. Reference

1. Binary Crossentropy  
<https://zhuanlan.zhihu.com/p/89391305>
2. Sigmoid 、ReLU  
<https://www.itread01.com/content/1546354994.html>
3. <https://www.itread01.com/feffx.html>
4. <https://zhuanlan.zhihu.com/p/55497753>
5. <https://www.gushiciku.cn/pl/2Jeg/zh-tw>
6. <https://kknews.cc/zh-tw/news/ze2voea.html>