1. Introduction

深度學習在各項領域中表現優秀，我們要去了解其背後的原理以及概念，本次 LAB 透過練習不使用現成函式庫，手刻深度學習模型，並且去實作所有細節

2. Implementation Details

A. Sigmoid function

由於是 activation function，最好是可微分的函式，實作上建立了一個字典，儲存各種 activation function，字典的 value 為一個 tuple，儲存函數和導函數。

```
self.activation_functions = {
    'sigmoid': (lambda x: 1 / (1 + np.exp(-x)), lambda x: (1 / (1 + np.exp(-x))) * (1 - (1 / (1 + np.exp(-x))))),
    'tanh': (lambda x: np.tanh(x), lambda x: 1 - x ** 2),
    'relu': (lambda x: np.maximum(0, x), lambda x: (x > 0).astype(int))
}
```

B. Neural network architecture

使用了 add_layer 函式，能夠方便動態新增不同樣式的 layer，weights 裡面儲存的是每一層的參數，分別是 W、bias、activation function。

```python
def add_layer(self, nurons, activation):
    if activation not in self.activation_functions:
        raise ValueError('Activation function not supported')
    activation = self.activation_functions[activation]
    self.weights.append([np.random.randn(nurons, self.d[-1]), np.random.randn(nurons, 1), activation])
    self.d.append(nurons)
```

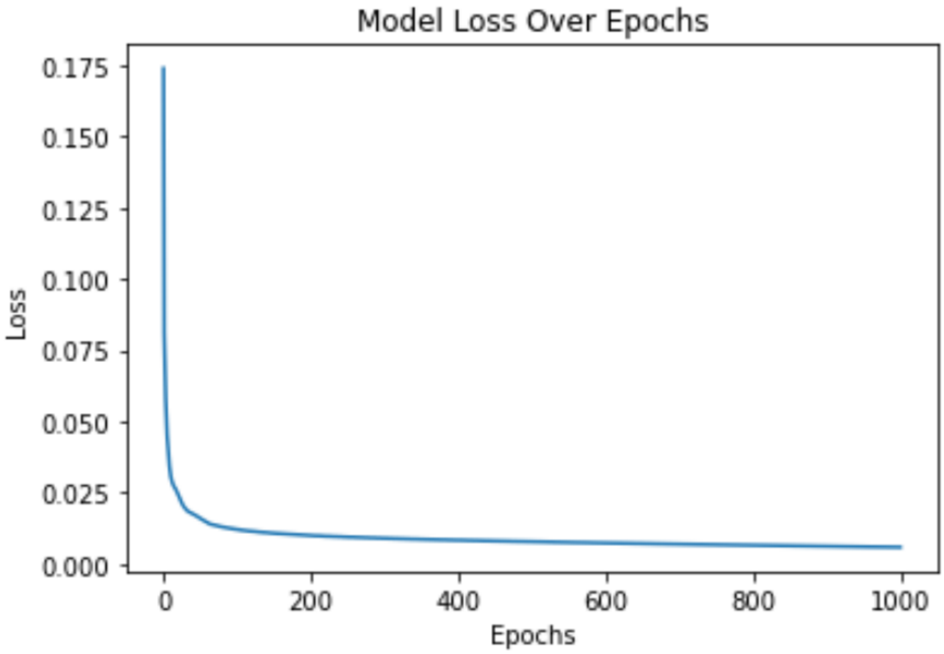
C. Back-propagation

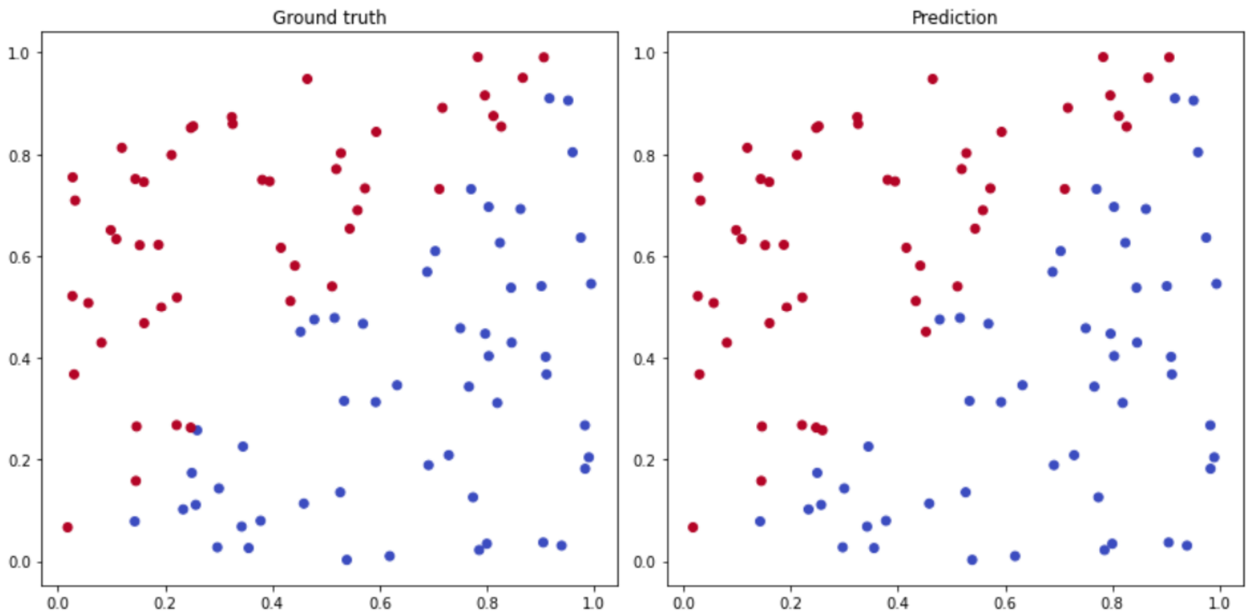
根據 loss function，去計算他對模型參數的梯度是多少，因為可以使用 chain rule，所以從 output 的地方往回更新回去比較快，cache[i]裡儲存的是第 i 層 layer 中，input x 在 forward 中的轉換變化，而其中 X=activation(Wx+b)，z=Wx+b。

```python
def backward(self, x, y, output):
    batch_size = x.shape[1]
    gradient = self.loss[1](output, y) # dL/dy
    for i in range(len(self.weights) - 1, -1, -1):
        W, B, A = self.weights[i]
        x, z, X = self.cache[i]
        gradient = gradient * A[1](X) # dL/dy * dy/dz,  z = W * f(...) + b
        # dL/db = dL/dy * dy/dz * dz/db, dz/db = 1
        B -= self.lr * np.mean(gradient, axis=1, keepdims=True)
        # dL/dw = dL/dy * dy/dz * dz/dw, dz/dw = x
        W -= self.lr * np.matmul(gradient, x.T) / batch_size
        # dL/dx = dL/dy * dy/dz * dz/d(f), dz/d(f) = W
        gradient = np.matmul(W.T, gradient)
```

3. Experimental result

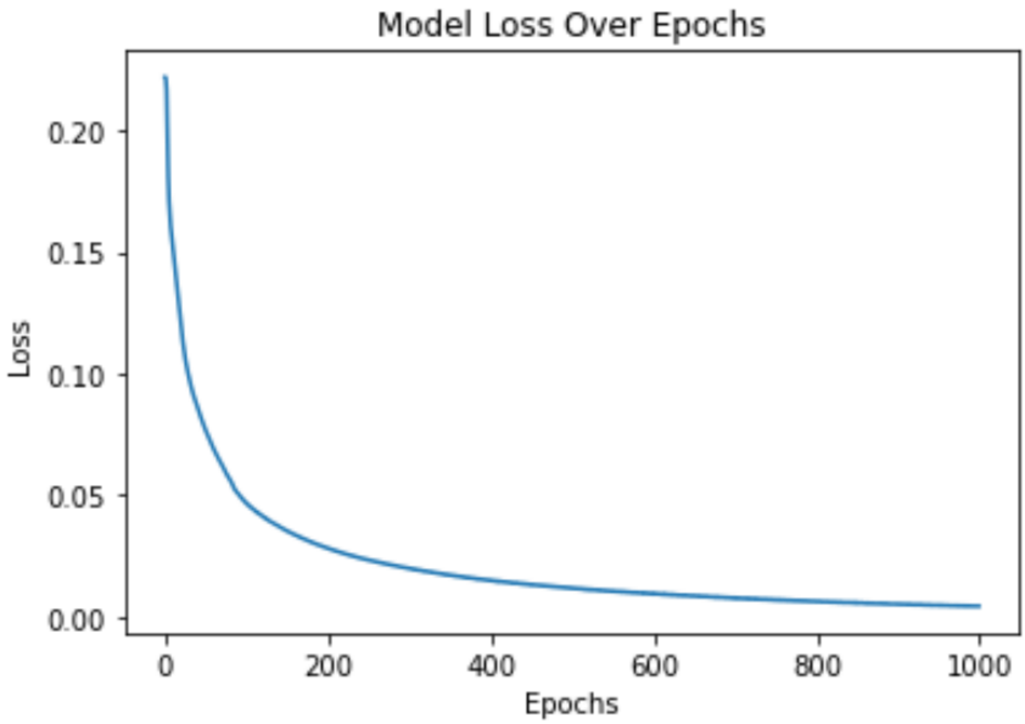
A. Linear

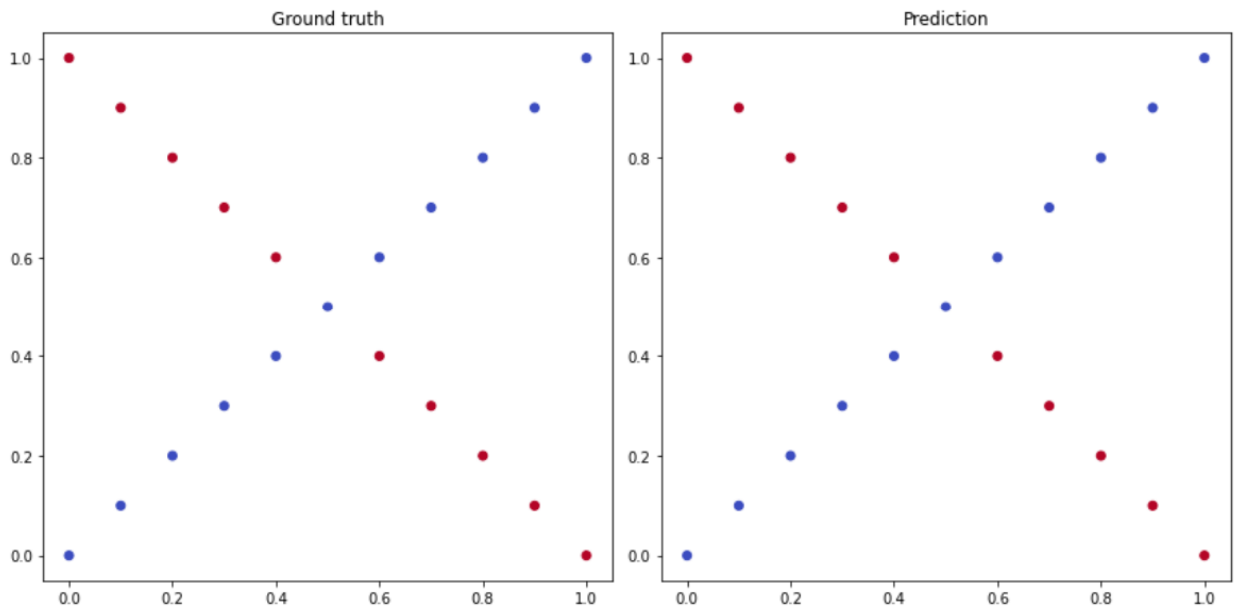





```
Epoch 998/1000, Loss: 0.005758986066730396
Epoch 999/1000, Loss: 0.0057535291828335325
Epoch 1000/1000, Loss: 0.0057522395957180774
testing data...
Loss: 0.004675638856878492, Accuracy: 0.9800
```

B. xor

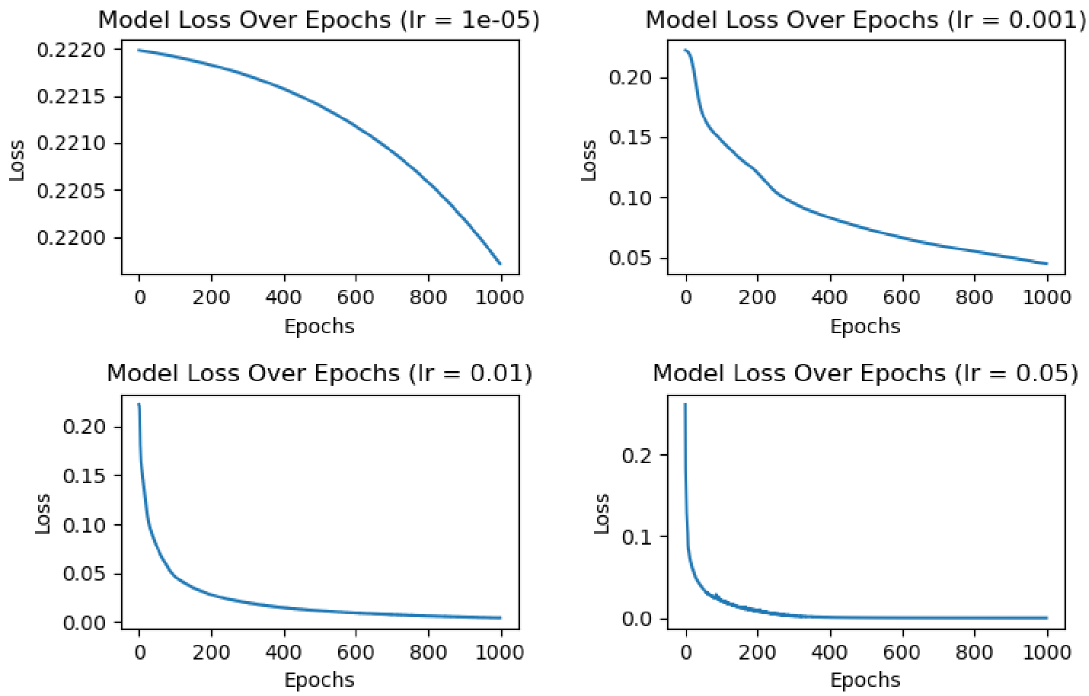





```
Epoch 998/1000, Loss: 0.004272018462680681
Epoch 999/1000, Loss: 0.004246126444996156
Epoch 1000/1000, Loss: 0.004321399773729657
testing data...
Loss: 0.003286591301389271, Accuracy: 1.0000
```

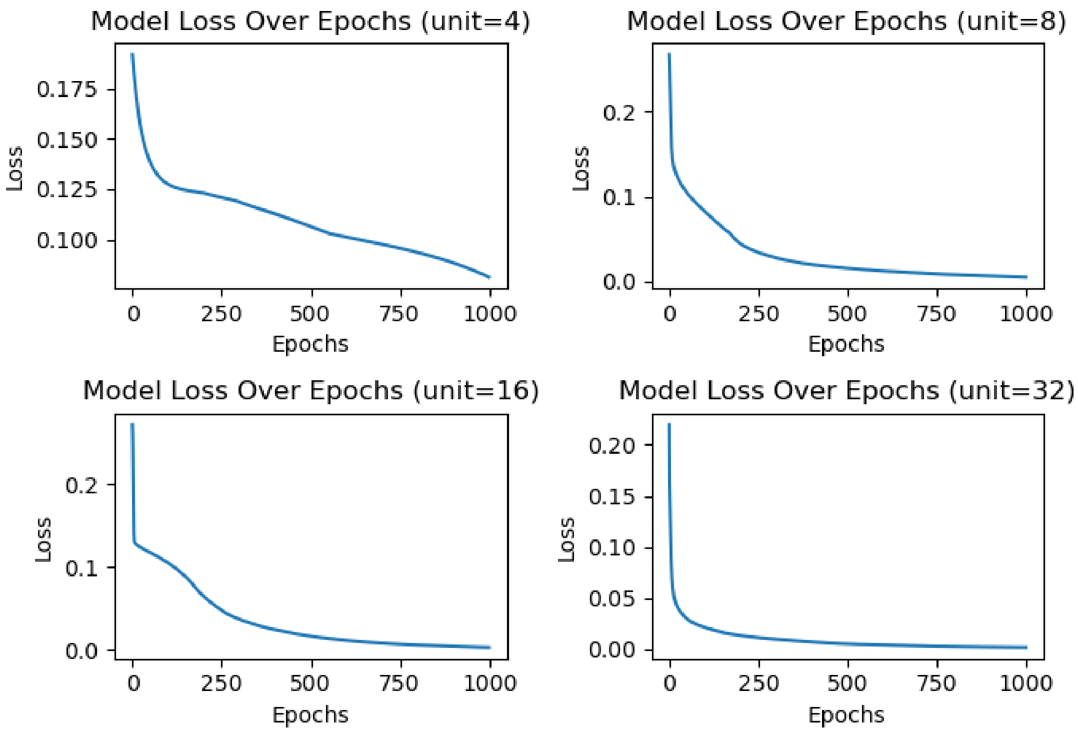
4. Discussion

A. different learning rate



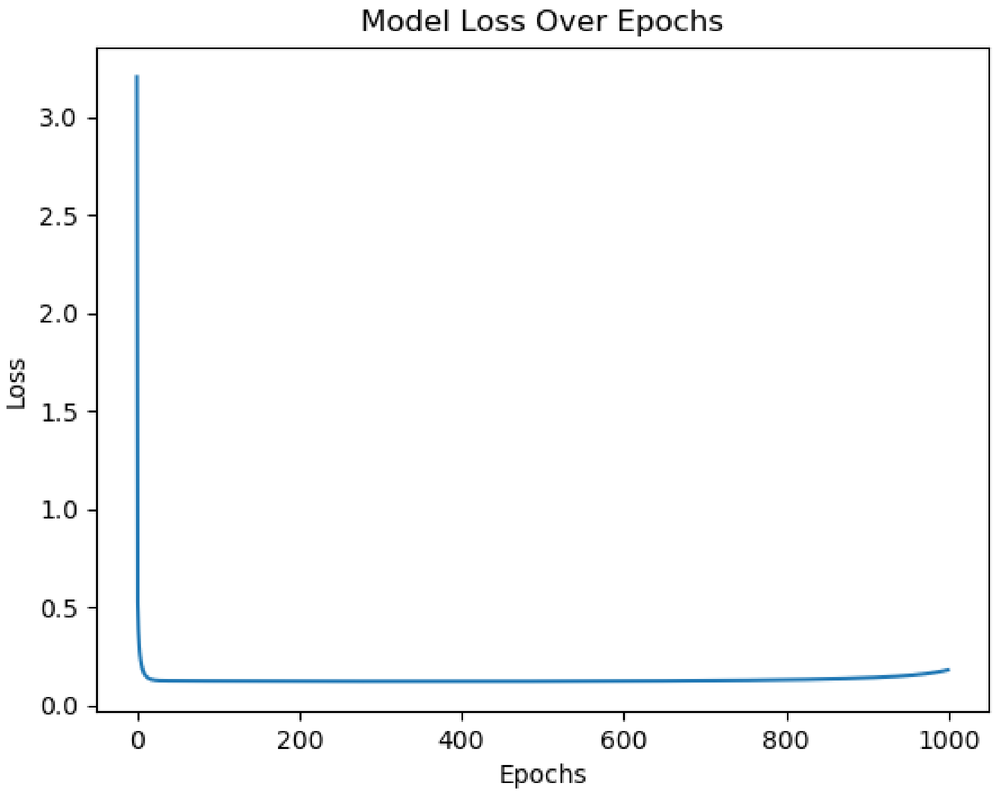

```
Learning rate:  1e-05
Epoch 1000/1000, Loss: 0.21971688017129826
testing data... (lr=1e-05)
Loss: 0.33458696101671564, Accuracy: 0.3333
--------------------------------------------------
Learning rate:  0.001
Epoch 1000/1000, Loss: 0.044953936129313005
testing data... (lr=0.001)
Loss: 0.1769290410302633, Accuracy: 0.3333
--------------------------------------------------
Learning rate:  0.01
Epoch 1000/1000, Loss: 0.004321399773729657
testing data... (lr=0.01)
Loss: 1.2701164657811895e-06, Accuracy: 1.0000
--------------------------------------------------
Learning rate:  0.05
Epoch 1000/1000, Loss: 3.123252092251372e-05
testing data... (lr=0.05)
Loss: 9.060502115341496e-16, Accuracy: 1.0000
```

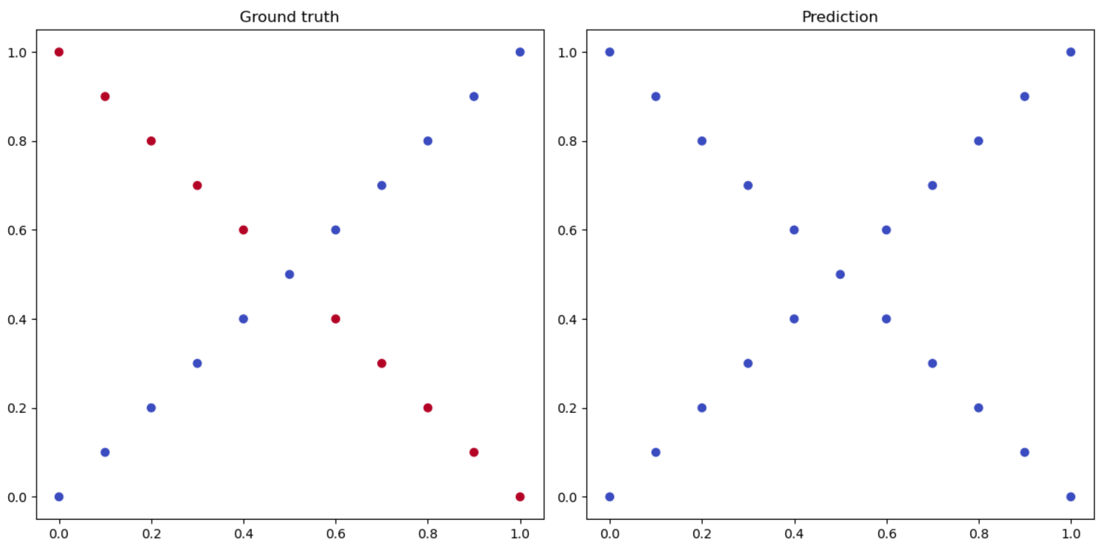
B. different hidden units




```
Unit:  4
Epoch 1000/1000, Loss: 0.0813046798405111
testing data... (unit=4)
Loss: 0.014049047706160727, Accuracy: 1.0000
--------------------------------------------------
Unit:  8
Epoch 1000/1000, Loss: 0.004754616121383414
testing data... (unit=8)
Loss: 0.0006070243742160442, Accuracy: 1.0000
--------------------------------------------------
Unit:  16
Epoch 1000/1000, Loss: 0.0030865945009540484
testing data... (unit=16)
Loss: 3.6718764058674555e-05, Accuracy: 1.0000
--------------------------------------------------
Unit:  32
Epoch 1000/1000, Loss: 0.001735157758983876
testing data... (unit=32)
Loss: 1.5726300297819886e-08, Accuracy: 1.0000
```

C. without activation function






```
Epoch 1000/1000, Loss: 0.17920522982439363
testing data... (with activation function)
Loss: 0.40556109698045767, Accuracy: 0.3333
```

D.

5. Questions

A. 為了要讓模型能夠做非線性對應。如果沒有 activation function，不管模型架了幾層 layer，都和只架一層 layer 的模型相同。也因此我們常用的 activation function 都是非線性函數。

B. Learning rate 太大的話會讓模型參數無法更新收斂到 loss function 的低點(對資料及而言)，有可能會在低點附近震盪。而 learning rate 太小的話雖然可以讓模型更新收斂到 loss function 的低點，但模型參數更新速度緩慢，訓練時間需要很久。

C. Weight 和 bias 是可以讓 input 經過線性空間對應後(或者再經過 activation function 做非線性對應後)，使得資料能夠以簡單的線性分類器(lab 中以 y>0.5 當作分類器)做分類。

6.