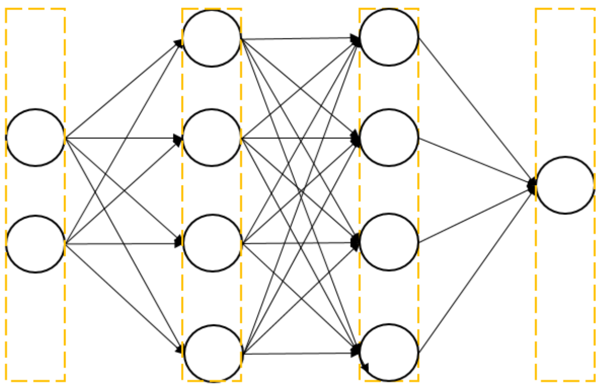
1. Introduction

此次作業用 python 實作 neural network with Backpropagation。整個 network 的架構與助教提供的 PTT 中的架構圖完全一致。



層數和每層的節點數都與上圖相同（二層 Hidden layer、每層有四個 Neurons）Model 設計用 10000 個 epoch 去跑，且 learning rate 會隨著 epoch 的增加而減少：

```
if e < epochs/4:
    lr = 0.3
else:
    lr = 0.3*(1 - e/epochs)
```

上述的寫法中，learning rate 在前 1/4 的總 epoch 是不會變動的，之後 learning rate 會成線性遞減。

2. Experiment setups

A. Sigmoid functions

此 model 的 activate function 是用 sigmoid。Sigmoid 的 python 實作與助教提供的相同，沒什麼特別的。

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

def sigmoid_derivative(x):
    return x * (1 - x)
```

B. Neural network

a. initial

設定好要訓練的總 epoch 數，以及每一層 Neurons 的數目：

```
epochs = 10000
inputLayerNeurons, hidden1LayerNeurons, hidden2LayerNeurons, outputLayerNeurons = 2,4,4,1
```

隨機給定初始的 weight 和 bias 的值：

```
#Random weights and bias init
hidden1_weights = np.random.uniform(size=(inputLayerNeurons,hidden1LayerNeurons))
hidden1_bias =np.random.uniform(size=(1,hidden1LayerNeurons))

hidden2_weights = np.random.uniform(size=(hidden1LayerNeurons,hidden2LayerNeurons))
hidden2_bias =np.random.uniform(size=(1,hidden2LayerNeurons))

output_weights = np.random.uniform(size=(hidden2LayerNeurons,outputLayerNeurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))
```

b. forward

forward 要做的事情：

(1) 將前一層的 output 矩陣與這層的 weight 矩陣相乘

(2) 再將每一 row 的結果加上 bias

(3) 將加上 bias 的結果過 sigmoid

重複上述動作直到最後一層（output layer）

```
#Forward Propagation
hidden1_layer_activation = np.dot(inputs,hidden1_weights)
hidden1_layer_activation += hidden1_bias
hidden1_layer_output = sigmoid(hidden1_layer_activation)

hidden2_layer_activation = np.dot(hidden1_layer_output,hidden2_weights)
hidden2_layer_activation += hidden2_bias
hidden2_layer_output = sigmoid(hidden2_layer_activation)

output_layer_activation = np.dot(hidden2_layer_output,output_weights)
output_layer_activation += output_bias
predicted_output = sigmoid(output_layer_activation)
```

C. Backpropagation

a. backpropagation

```
#Backpropagation
error = expected_output - predicted_output
#print(error)
for each in error:
    loss += each[0]*each[0]
d_predicted_output = error * sigmoid_derivative(predicted_output)

error_hidden2_layer = d_predicted_output.dot(output_weights.T)
d_hidden2_layer = error_hidden2_layer * sigmoid_derivative(hidden2_layer_output)

error_hidden1_layer = d_hidden2_layer.dot(hidden2_weights.T)
d_hidden1_layer = error_hidden1_layer * sigmoid_derivative(hidden1_layer_output)
```

將 Ground truth 與此次 epoch 預測出來的結果相減，得出誤差。並進一步得出每一層的每個 Neurons 的 delta 。

上圖中的變數 loss 是計算 MSE ，用於之後 learning curve 的顯示。

b. Update Weights and Biases

用之前求出的 delta 值，更新 weight 和 bias。

```
#Update Weights and Biases
output_weights += hidden2_layer_output.T.dot(d_predicted_output) * lr
output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
hidden2_weights += hidden1_layer_output.T.dot(d_hidden2_layer) * lr
hidden2_bias += np.sum(d_hidden2_layer,axis=0,keepdims=True) * lr
hidden1_weights += inputs.T.dot(d_hidden1_layer) * lr
hidden1_bias += np.sum(d_hidden1_layer,axis=0,keepdims=True) * lr
```

Weight 的更新公式：

$$w_i = w_i - \alpha * \delta * x_i$$

Bias 的更新公式：

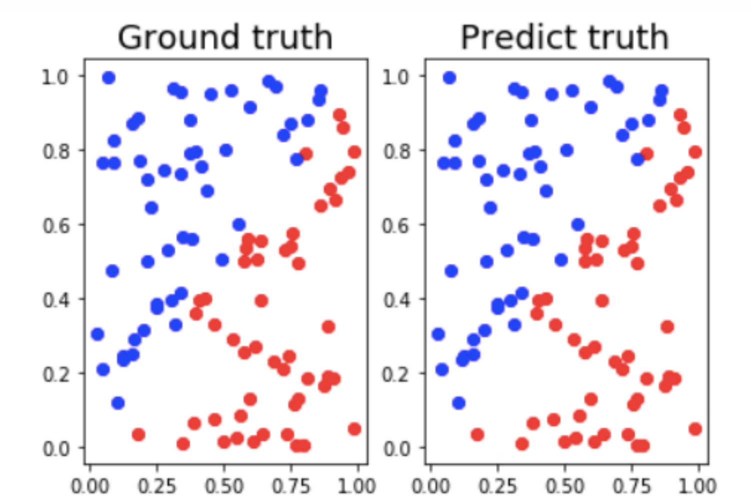
$$w_{bias} = w_{bias} - \alpha * \delta$$

3. Results of your testing

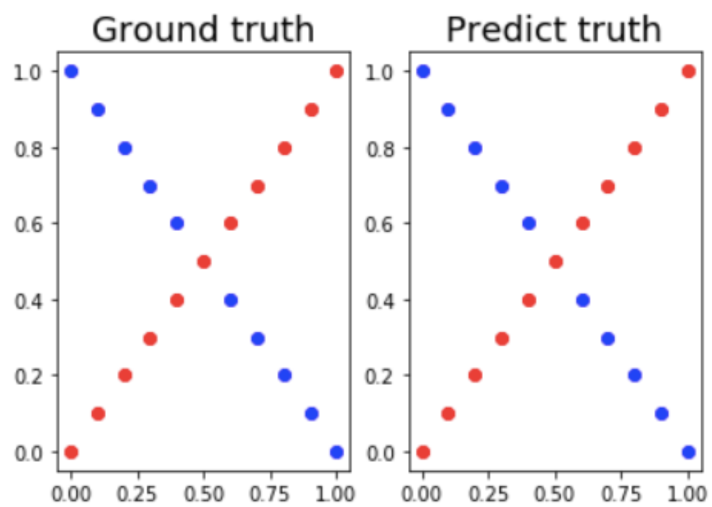
下方將分別顯示二組測資（Linear、XOR）的訓練結果

A. Screenshot and comparison figure

a. Linear



b. XOR



B. Show the accuracy of your prediction

每 500 epoch 的 loss

```
epoch=0, error=8.70293353145995
epoch=500, error=5.237242678934333
epoch=1000, error=5.234508086028232
epoch=1500, error=5.218932033234961
epoch=2000, error=4.611783509256094
epoch=2500, error=1.3551102884185766
epoch=3000, error=0.9553069549791888
epoch=3500, error=0.9266055626490098
epoch=4000, error=0.9192612815074984
epoch=4500, error=0.9161987229249557
epoch=5000, error=0.9145864935466854
epoch=5500, error=0.9136175075851073
epoch=6000, error=0.9129849748529013
epoch=6500, error=0.9125494709016938
epoch=7000, error=0.9122394973111349
epoch=7500, error=0.9120151256623902
epoch=8000, error=0.9118526801750648
epoch=8500, error=0.9117375780002027
epoch=9000, error=0.9116607186454503
epoch=9500, error=0.911616571410535
```

最終的 prediction (XOR 測資)

```
[[0.01084396]
 [0.90718881]
 [0.01043695]
 [0.90713268]
 [0.00983959]
 [0.90707516]
 [0.00938408]
 [0.90701629]
 [0.02906366]
 [0.90695611]
 [0.90689466]
 [0.03403991]
 [0.90683197]
 [0.00388654]
 [0.90676808]
 [0.00251581]
 [0.90670304]
 [0.00217562]
 [0.90663688]
 [0.00203501]
 [0.90656965]]
```

計算 accuracy 的程式：

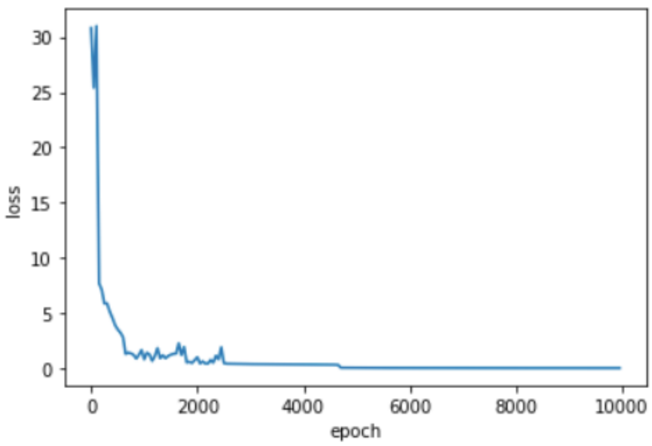
```python
def test_accuracy(train_y, pred_y):
    wrong = 0.0
    for i in range(len(train_y)):
        if train_y[i] != pred_y[i]:
            wrong += 1.0
    return (len(train_y)-wrong)/len(train_y)
```

最後得出測資 Linear 的 accuracy 為 1 (全部正確)，XOR 的 accuracy 也為 1。

C. Learning curve

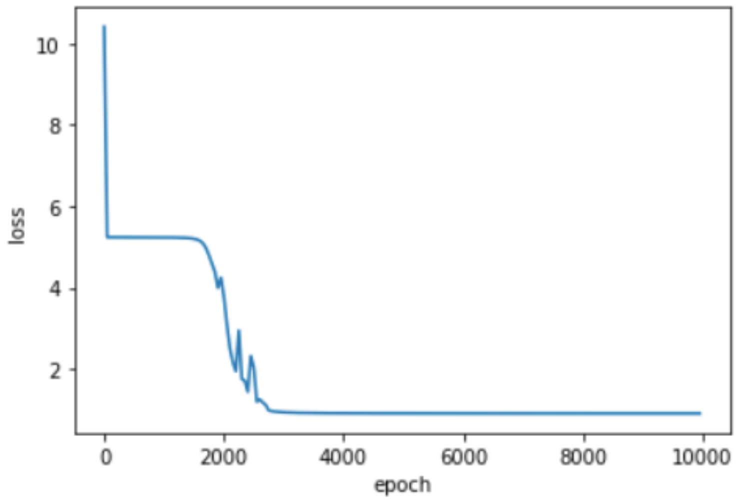
下方圖中的橫軸為 epoch，縱軸為 MSE。

a. Linear



Loss 大約在 epoch 為 100 左右時大幅下降，
之後慢慢趨近於 0。

b. XOR



Loss 大約在 epoch 為 50 左右時大幅下降，50 ~ 1600 時 loss 保持不變。1600 後 loss 才再次下降。

4. Discussion

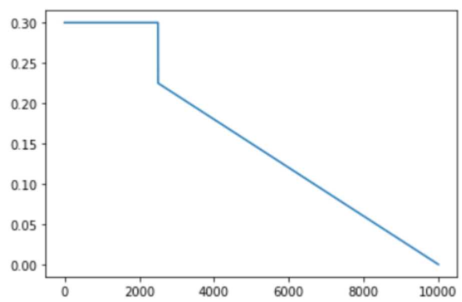
A. Try different learning rates

比較下列三種不同的 learning rate：

a. Origin Learning rate：

```
if e < epochs/4:
    lr = 0.3
else:
    lr = 0.3*(1 - e/epochs)
```

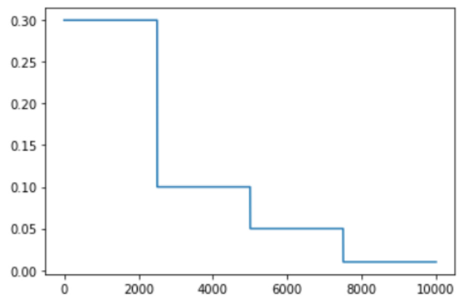
Learning rate 走勢：



b. Piecewise constant Learning rate：

```
if e < epochs/4:
    lr = 0.3
elif e < epochs/2:
    lr = 0.1
elif e < epochs/4*3:
    lr = 0.05
else:
    lr = 0.01
```

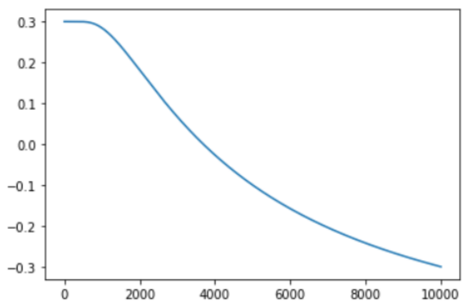
Learning rate 走勢：



c. Exponential Learning rate：

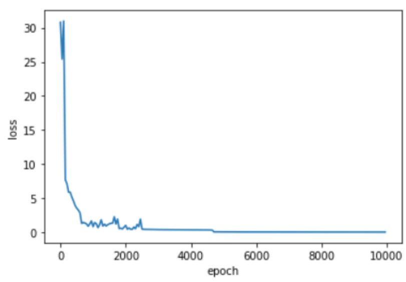
```
lr = 0.3 - 0.3 * pow(5, (1 - epochs / (e + 0.00001)))
```

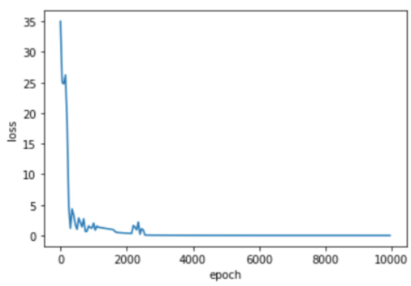
Learning rate 走勢：

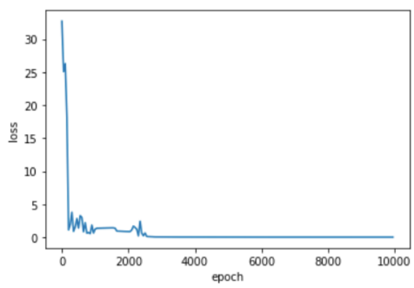


結果比較：

Origin Learning rate　　Piecewise constant Learning rate　　Exponential Learning rate







由上述三圖可以得知，Piecewise constant 和 Exponential Learning rate 的表現都比 Origin Learning rate 的表現稍佳，但彼此差異不大。推測差異不大的原因可能是因為我們代入的測資（Linear）對模型而言都太過簡單，導致無論 Learning rate 怎麼設計皆可成功訓練。

B. Try different numbers of hidden units

測試並比較下列四種 network：

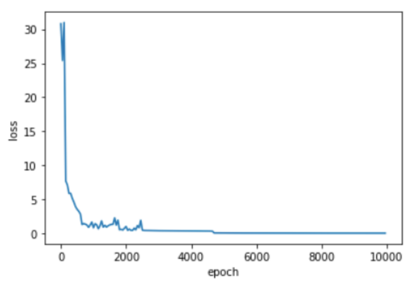
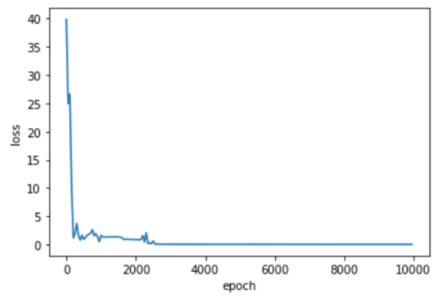
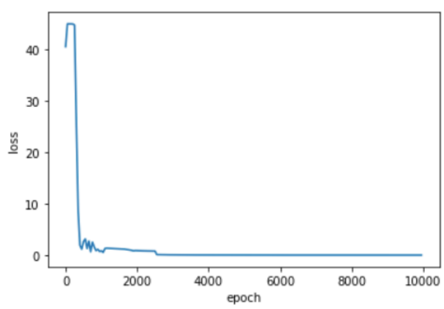
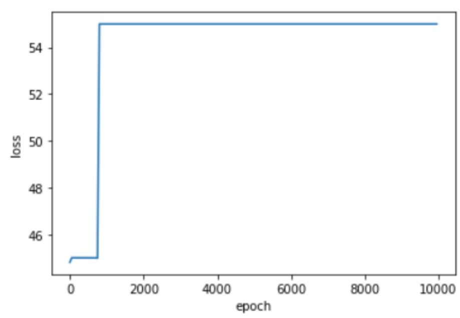
a. Hidden layer1 4 Neurons，Hidden layer2 4 Neurons (就原本的)

b. Hidden layer1 8 Neurons，Hidden layer2 4 Neurons

c. Hidden layer1 4 Neurons，Hidden layer2 8 Neurons

d. Hidden layer1 16 Neurons，Hidden layer2 16 Neurons

測試結果如下表：

<table>
<tr>
<td>a

</td>
<td>b

</td>
</tr>
<tr>
<td>c

</td>
<td>d

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

由上表可知，訓練成果最好的是 Hidden layer1 8 Neurons，Hidden layer2 4 Neurons 的 Models，但與 a（Hidden layer1 4 Neurons，Hidden layer2 4 Neurons） 和 c （Hidden layer1 4 Neurons，Hidden layer2 8 Neurons）差異不

大．至於 d （Hidden layer1 16 Neurons，Hidden layer2 16 Neurons），則是完全爆掉，loss 趨近於 55 左右．

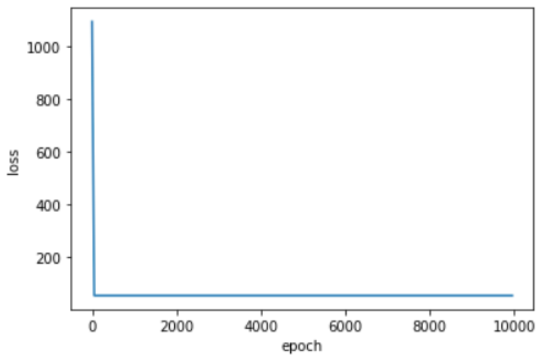
C. Try without sigmoid function

a. 使用 ReLU 作為 activate function：

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

def ReLU_derivative(x):
    tmp = x
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i][j] > 0:
                tmp[i][j] = 1
            else:
                tmp[i][j] = 0
    return tmp
```

結果：



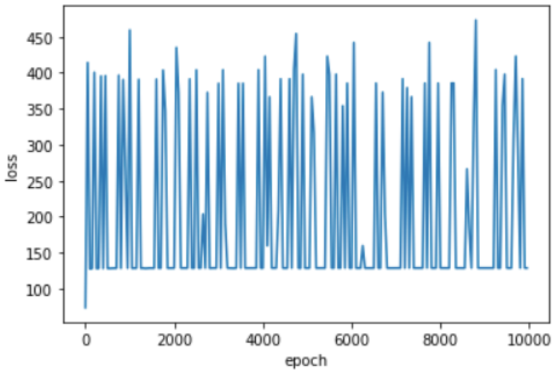
訓練結果極差， loss 大約卡在 55．

b. 使用 arctan 作為 activate function：

```
def arctan (x):
    return np.arctan(x)

def arctan_derivative(x):
    tmp = x
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            tmp[i][j] = 1 / (x[i][j]**2 + 1)
    return tmp
```

結果：



訓練結果極差，loss 大約在 450 和 130 之間來回跳動．