Lab1 : back-propagation report

110550161 張維程

1. Introduction (5%)

lab1只需要實作出一個簡單的Neural Network，並利用forward 來預測出結果，back propagation 來更新參數。目標是讓Neural Network 預測出的答案和label越像越好。主要有兩種input來驗證該模型的好壞，一種是linear線性的，另一種是非線性的XOR。

2. Implementation Details (15%):

A. Sigmoid function

Sigmoid function能夠將input value 對應到(-1,1)之間，而且可以求導。$\sigma(x) = \frac{1}{1+e^{-x}}$ 即為sigmoid function，求導的部分會在實作back propagation 時用到，求導可得 $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

```python
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))
def derivative_sigmoid(x):
    return np.multiply(x, 1.0-x)
```

上圖為實作程式碼的部分，參考Spec內提供的內容，function 在forward和backward時直接呼叫即可。

B. Neural network architecture

a. Layers

```python
class SimpleNeuralNetwork:
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size, lr=0.1, epoch=2000, printloss_interval=100):

        #hyper par
        self.lr = lr
        self.epoch = epoch
        self.printloss_interval = printloss_interval

        # 每一層的Weight 跟bias
        self.w1 = np.random.randn(input_size, hidden_size1)
        self.b1 = np.random.randn(1, hidden_size1)
        self.w2 = np.random.randn(hidden_size1, hidden_size2)
        self.b2 = np.random.randn(1, hidden_size2)
        self.w3 = np.random.randn(hidden_size2, output_size)
        self.b3 = np.random.randn(1, output_size)
```

如同要求，建一個簡單的neural network，包含了兩層的hidden layers，其中w1 w2 w3 為weight，b1 b2 b3 為其bias (spec上的公式只有weight 但是通常也會有bias所以加上去了)。

b. Forward process

```
def forward(self, x):
    # 第一層
    self.z1 = np.dot(x, self.w1) + self.b1
    self.a1 = sigmoid(self.z1)
    # 第二層
    self.z2 = np.dot(self.a1, self.w2) + self.b2
    self.a2 = sigmoid(self.z2)
    # 輸出層
    self.z3 = np.dot(self.a2, self.w3) + self.b3
    self.output = sigmoid(self.z3)
    return self.output
```

Linear layer 會將輸入乘上weights 並再加上bias，並存在z1,z2,z3中。再將z1,z2,z3過sigmoid function 存入a1,a2,output。

c. Loss function

```
loss = -np.mean(y * np.log(self.output) + (1 - y) * np.log(1 - self.output))
```

因為這次的task 是要處理二元分類問題，而loss使用binary cross entropy 會比較適合。

公式為: $loss = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$ 即上圖程式碼之算式, 程式碼中的output變數就是$\hat{y}$

C. Backpropagation

a. 數學算式推導

已知算式：

$z_i = w_i a_{i-1} + b_i,\ i = 1,2,3\ ,a_0 = x$

$\hat{y} = sigmoid(z3)$

$\delta_3 = \hat{y} - y$

對w_3的偏微分推導：

使用chain rule 後可得個別三項，乘起來並化簡即可得結果$\delta_3 a_2$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial w_3} = -(\frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})})(\hat{y}(1-y))(a_2) = \delta_3 a_2$$

對b_3的偏微分:

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_3}\frac{\partial z_3}{\partial b_3} = -\left(\frac{y}{\hat{y}} - \frac{(1-y)}{(1-\hat{y})}\right)(\hat{y}(1-y))(1) = \delta_3$$

剩餘對w_2, b_2, w_1, b_1 的偏微分亦為同樣推導方式

整理上述算式可得結果為

$$\frac{\partial L}{\partial w_3} = \delta_3 a_2, \ \frac{\partial L}{\partial b_3} = \delta_3$$

$$\frac{\partial L}{\partial w_2} = \delta_2 a_1, \ \frac{\partial L}{\partial b_2} = \delta_2$$

$$\frac{\partial L}{\partial w_1} = \delta_1 x, \ \frac{\partial L}{\partial b_1} = \delta_1$$

$$\delta_3 = \hat{y} - y, \ \delta_2 = (w_3)^T \delta_3 \odot \sigma'(z_2), \ \delta_1 = (w_2)^T \delta_2 \odot \sigma'(z_1)$$

而back propagation利用了$\delta_3, \delta_2, \delta_1$之間彼此的關係,

省下了重複的計算, 算式可以直接call之前算過的結果就好。

b. 程式碼實作

```
def backward(self, x, y):
    # 輸出層的 error
    output_error = self.output - y
    output_delta = output_error * derivative_sigmoid(self.output)

    # 第二層得'r error
    a2_error = output_delta.dot(self.w3.T)
    a2_delta = a2_error * derivative_sigmoid(self.a2)

    # 第一層的 error
    a1_error = a2_delta.dot(self.w2.T)
    a1_delta = a1_error * derivative_sigmoid(self.a1)

    # 根據上面算出來的delta 去更新W跟bias
    self.w3 -= self.lr * self.a2.T.dot(output_delta)
    self.b3 -= self.lr * np.sum(output_delta, axis=0, keepdims=True)

    self.w2 -= self.lr * self.a1.T.dot(a2_delta)
    self.b2 -= self.lr * np.sum(a2_delta, axis=0, keepdims=True)

    self.w1 -= self.lr * x.T.dot(a1_delta)
    self.b1 -= self.lr * np.sum(a1_delta, axis=0, keepdims=True)
```

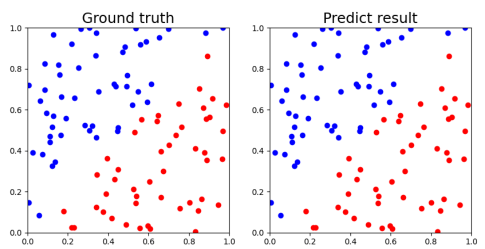
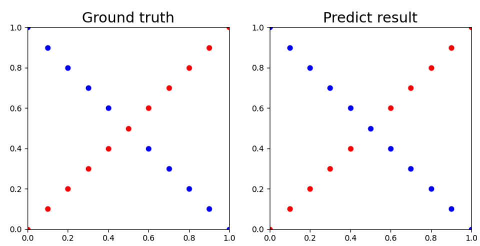
數學推導中的$\delta_3,\delta_2,\delta_1$ 分別對應程式碼中的output_delta, a2_delta , a1_delta。並進一步依據learning rate去更新參數w_i,b_i

3. Experimental Results (45%)

A. Screenshot and comparison figure

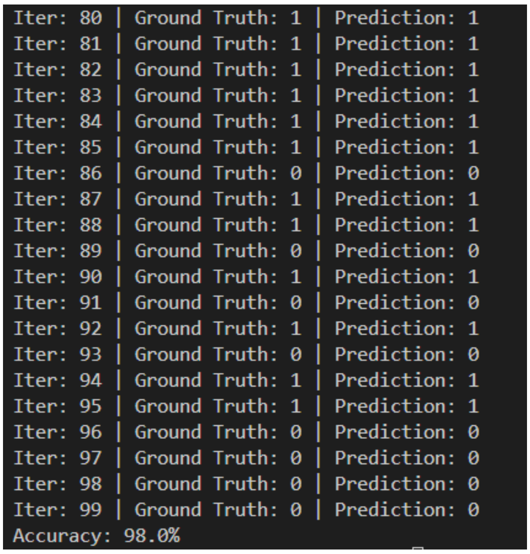
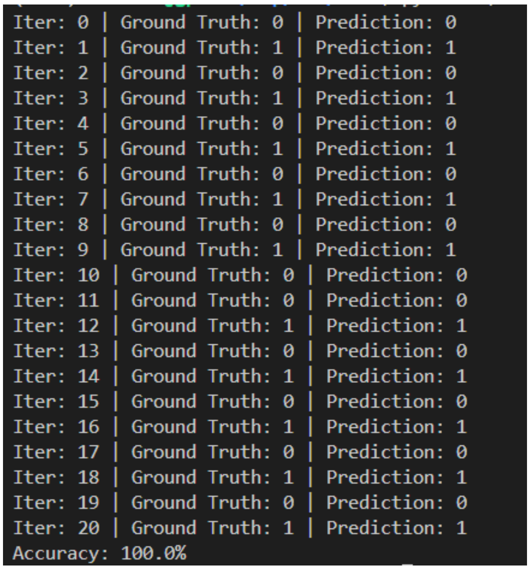
下表為各自用各自的training data 訓練過後，再將n = 100的testing data 丟進neural network 後的結果。

loss function 使用的是binary cross entropy

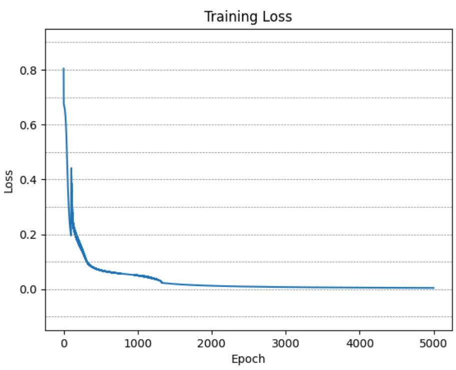
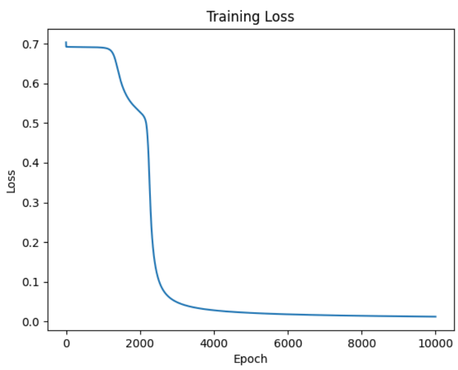
	Linear	XOR
		
lr	0.1	0.2
epoch	5000	10000
loss	Epoch 2600/5000, Loss: 0.009118746782741981 Epoch 2700/5000, Loss: 0.008735136499316384 Epoch 2800/5000, Loss: 0.008386519283655686 Epoch 2900/5000, Loss: 0.008068270981207175 Epoch 3000/5000, Loss: 0.00777654507110545 Epoch 3100/5000, Loss: 0.007508116738172378 Epoch 3200/5000, Loss: 0.007260262793881379 Epoch 3300/5000, Loss: 0.007030668238935129 Epoch 3400/5000, Loss: 0.006817352853636082 Epoch 3500/5000, Loss: 0.006618613005421283 Epoch 3600/5000, Loss: 0.0064329751336322126 Epoch 3700/5000, Loss: 0.006259158278317566 Epoch 3800/5000, Loss: 0.006096043674461145 Epoch 3900/5000, Loss: 0.005942649910774787 Epoch 4000/5000, Loss: 0.0057981125044507965 Epoch 4100/5000, Loss: 0.005661667005495411 Epoch 4200/5000, Loss: 0.005532634941236844 Epoch 4300/5000, Loss: 0.005410412060824132 Epoch 4400/5000, Loss: 0.005294458453485472 Epoch 4500/5000, Loss: 0.005184290202001566 Epoch 4600/5000, Loss: 0.005079472300805864 Epoch 4700/5000, Loss: 0.004979612621152105 Epoch 4800/5000, Loss: 0.00488435674743503 Epoch 4900/5000, Loss: 0.004793383541659571	Epoch 0/10000, Loss: 0.7389334482216277 Epoch 1000/10000, Loss: 0.5762370775258105 Epoch 2000/10000, Loss: 0.24216539516763758 Epoch 3000/10000, Loss: 0.17391168152050016 Epoch 4000/10000, Loss: 0.16844889684358946 Epoch 5000/10000, Loss: 0.1663816283112492 Epoch 6000/10000, Loss: 0.16525456766284638 Epoch 7000/10000, Loss: 0.16452922705823336 Epoch 8000/10000, Loss: 0.1640157997519811 Epoch 9000/10000, Loss: 0.16362923468337304

B. Show the accuracy of your prediction (40%)

(achieve 90% accuracy)

Linear	XOR
 Iter: 80 | Ground Truth: 1 | Prediction: 1 Iter: 81 | Ground Truth: 1 | Prediction: 1 Iter: 82 | Ground Truth: 1 | Prediction: 1 Iter: 83 | Ground Truth: 1 | Prediction: 1 Iter: 84 | Ground Truth: 1 | Prediction: 1 Iter: 85 | Ground Truth: 1 | Prediction: 1 Iter: 86 | Ground Truth: 0 | Prediction: 0 Iter: 87 | Ground Truth: 1 | Prediction: 1 Iter: 88 | Ground Truth: 1 | Prediction: 1 Iter: 89 | Ground Truth: 0 | Prediction: 0 Iter: 90 | Ground Truth: 1 | Prediction: 1 Iter: 91 | Ground Truth: 0 | Prediction: 0 Iter: 92 | Ground Truth: 1 | Prediction: 1 Iter: 93 | Ground Truth: 0 | Prediction: 0 Iter: 94 | Ground Truth: 1 | Prediction: 1 Iter: 95 | Ground Truth: 1 | Prediction: 1 Iter: 96 | Ground Truth: 0 | Prediction: 0 Iter: 97 | Ground Truth: 0 | Prediction: 0 Iter: 98 | Ground Truth: 0 | Prediction: 0 Iter: 99 | Ground Truth: 0 | Prediction: 0 Accuracy: 98.0%	 Iter: 0 | Ground Truth: 0 | Prediction: 0 Iter: 1 | Ground Truth: 1 | Prediction: 1 Iter: 2 | Ground Truth: 0 | Prediction: 0 Iter: 3 | Ground Truth: 1 | Prediction: 1 Iter: 4 | Ground Truth: 0 | Prediction: 0 Iter: 5 | Ground Truth: 1 | Prediction: 1 Iter: 6 | Ground Truth: 0 | Prediction: 0 Iter: 7 | Ground Truth: 1 | Prediction: 1 Iter: 8 | Ground Truth: 0 | Prediction: 0 Iter: 9 | Ground Truth: 1 | Prediction: 1 Iter: 10 | Ground Truth: 0 | Prediction: 0 Iter: 11 | Ground Truth: 0 | Prediction: 0 Iter: 12 | Ground Truth: 1 | Prediction: 1 Iter: 13 | Ground Truth: 0 | Prediction: 0 Iter: 14 | Ground Truth: 1 | Prediction: 1 Iter: 15 | Ground Truth: 0 | Prediction: 0 Iter: 16 | Ground Truth: 1 | Prediction: 1 Iter: 17 | Ground Truth: 0 | Prediction: 0 Iter: 18 | Ground Truth: 1 | Prediction: 1 Iter: 19 | Ground Truth: 0 | Prediction: 0 Iter: 20 | Ground Truth: 1 | Prediction: 1 Accuracy: 100.0%

C. Learning curve (loss-epoch curve)
x 軸方向為Epoch , y 軸為loss, 可看出Linear 是相對簡單的task , 而XOR 較難。

Linear	XOR
	

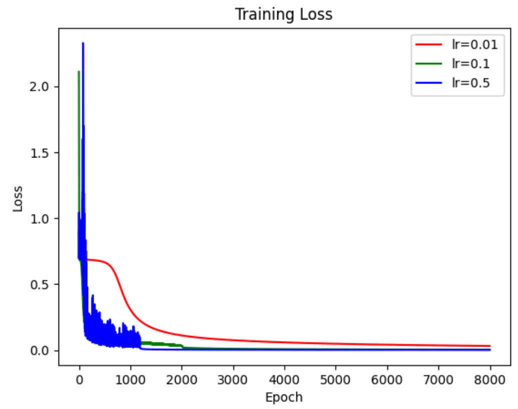
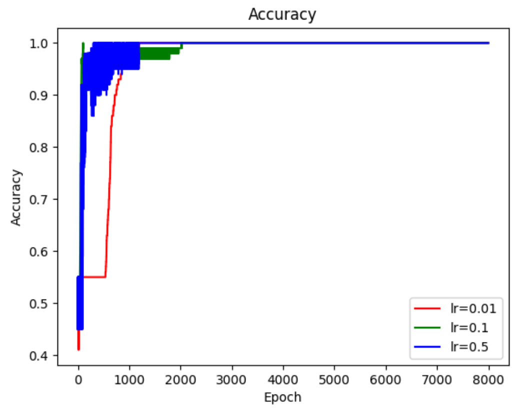
D. Anything you want to share

4. Discussion (15%)

A. Try different learning rate

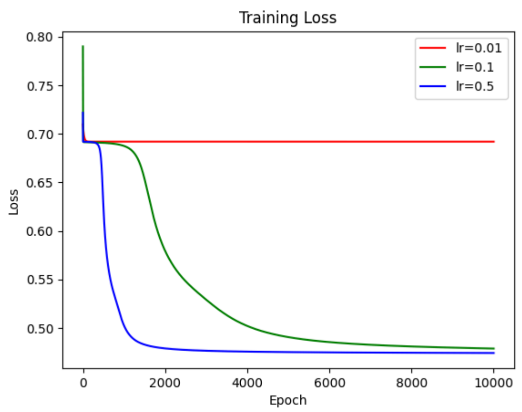
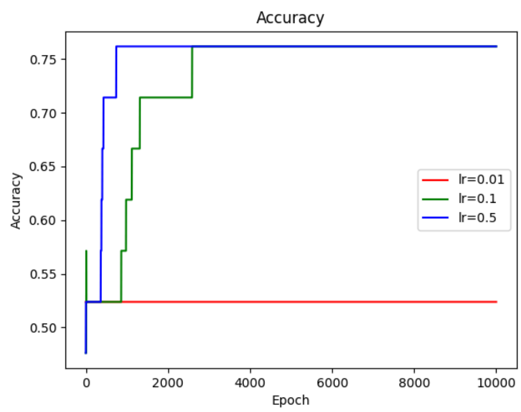
Linear

下表為將lr 設為0.01(紅色) ,0.1(綠色),0.5(藍色) 後的結果，可以看到三者之loss都下降，且accuracy上升。但是loss 的部分可以推論0.1(綠色), 0.5(藍色)相較於0.01(紅色)比較適合，因為loss穩定地比0.01(紅色)還要低。

<table>
<tr><th>Loss</th><th>Accuracy</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

XOR

下表為將lr 設為0.01(紅色) ,0.1(綠色),0.5(藍色) 後，的結果，可以看到後兩者之loss下降，且accuracy上升。但是0.01(紅色)較不適合處理該task，在訓練大約500個epoch後，loss便不再下降且accuracy也不再上升。

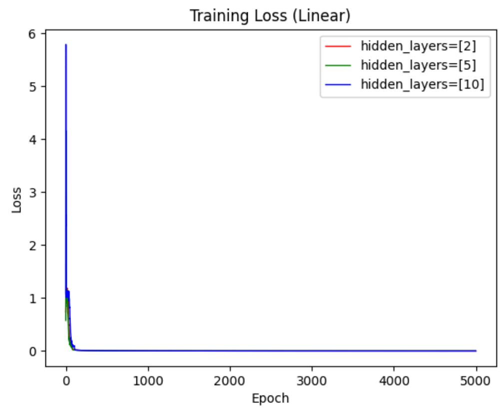
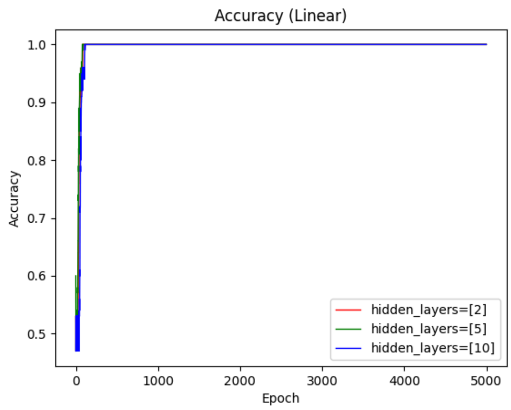
<table>
<tr><th>Loss</th><th>Accuracy</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

由上列四圖可說明lr 之於不同task的重要性，每個Task都有其較適合的learning rate。

B. Try different numbers of hidden units

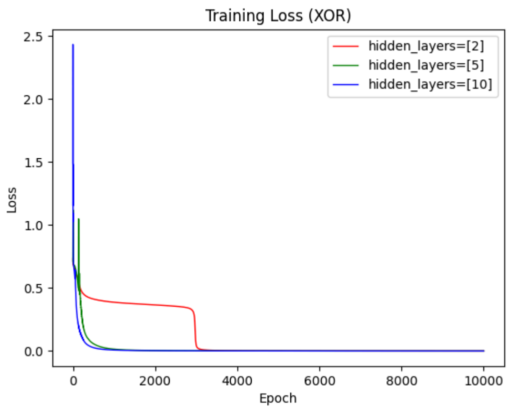
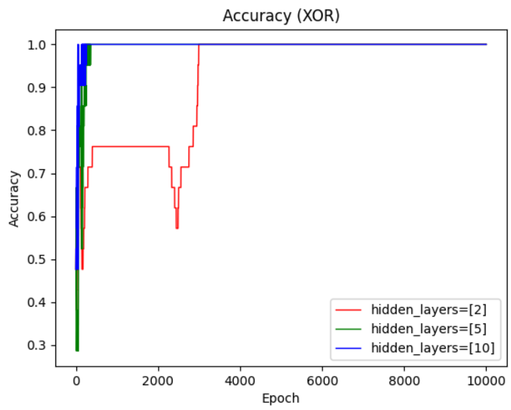
Linear

下表為將hidden unit 數量設為2(紅色) ,5(綠色),10(藍色) 後的結果，可以看到三者之loss都下降，且accuracy上升，而且曲線都非常接近。可以推測linear的這個task很簡單，2個以上的hidden unit 就足以處理，再多也只是浪費，因為accuracy 已經差不多是100%了。

<table>
<tr><th>Loss</th><th>Accuracy</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

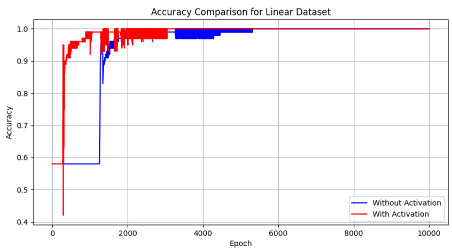
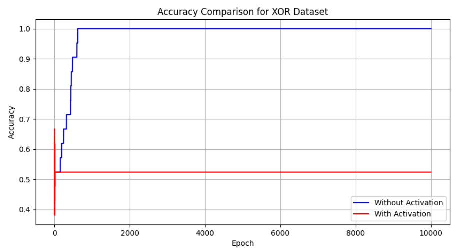
XOR

下表為將hidden unit 數量設為2(紅色) ,5(綠色),10(藍色) 後的結果，可以看到三者之loss都下降，且accuracy上升。觀察後可以推測相較於linear, XOR的這個task比較困難，因為2層hidden layers (紅色)學得比較慢，所以到了epoch 比較中段時才提升accuracy 至接近100%。

<table>
<tr><th>Loss</th><th>Accuracy</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

由上列四圖可說明對於比較難的task，適度的增加hidden layer 的數量能夠有效地提升performance。但較簡單的task，由於少少的hidden layer 也能夠做得很好，所以再增加hidden layer 也無法提升甚麼performance 。

C. Try without activation functions

Linear	XOR
	

上圖為linear/XOR case 有無activation function 之比較，如果沒有了activation function (原本為sigmoid)，那麼該神經網路就只會是原input的線性轉換而已，即wx+b ，而該情況仍舊能夠處理linearcase ，所以without activation (藍色)的accuracy仍然有提升，但是XOR為非線性的資料，所以accuracy是絕對無法提升上去的。

D. Anything you want to share

5. Questions (20%)

A. What is the purpose of activation functions? (6%)

目的是要讓神經網路能夠學習非線性的關係，如同前面提過的，如果沒有activation function，則各個layer 都做線性轉換其實等價於做一次線性轉換，這樣的性質無法讓神經網路去學習複雜以及非線性的task 。

B. What might happen if the learning rate is too large or too small? (7%)

lr太大的話可能會導致神經網路學習的梯度太大，所以沒辦法收斂，不斷地在找optimal solution 時來回震盪。
lr太小的話則可能會導致神經網路學習地太慢，讓learning time 變得很長，而且有可能會導致梯度不夠，讓其卡在local minimum 出不來。

C. What is the purpose of weights and biases in a neural network? (7%)

weights 的目的是讓神經網路能夠調整weight來判斷哪一個特徵比較重要，各個特徵會依照各自的重要程度來影響output結果。
bias 的目的是要讓神經網路在沒有輸入的時候還是可以有非零的output，而且如果沒有bias，每一層都是weight 乘上input，那所有的output 也都會被固定在原點附近，會降低performance。

6. Extra (10%)

A. Implement different optimizers. (2%)

B. Implement different activation functions. (3%)

C. Implement convolutional layers. (5%)