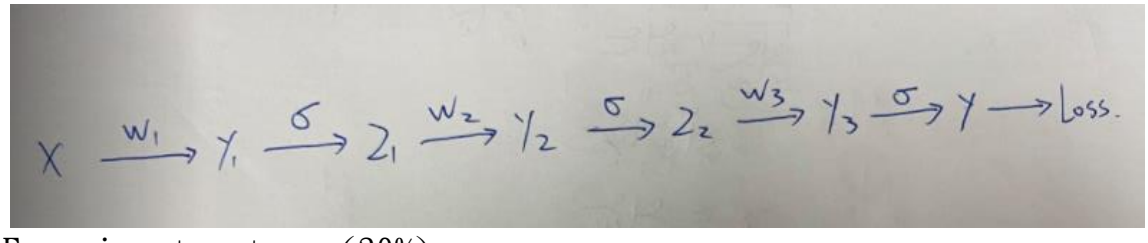
HW1-Report

1. Introduction (20%)

透過助教提供的 generate_linear、generate_XOR_easy，得到對應的input data 以及對應的gruond truth，實作神經網路中含有兩層hidden layer的forward propagation、back propagation功能，不斷迭代後使我們對training data的預測能夠越來越精準，activation function預設是sigmoid。

- forward propagation:讓input data 經由每層神經網路透過不同的weight、activation function計算出 prediction。
- backward propagation:藉由計算prediction與ground truth的誤差得到loss function，再透過chain rule實作gradient decent，算出各個weight對應到loss的partial derivative，進而更新各個weight。

不斷重複上述兩步驟使得loss function遞減後透過助教提供的show_result看comparison graph，下圖為程式碼對應的flowchar。



2. Experiment setups (30%):

A. Sigmoid functions

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

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

助教提供的sigmoid functions，要注意的是derivative_sigmoid傳進的input要是sigmoid後的值。// s’(x) = s(x)*(1-s(x))

B. Neural network

```
class Model():
    def __init__(self, dims):
        self.weight = [
            np.random.uniform(size=(dims[i], dims[i+1]))
            for i in range(len(dims) - 1)]


# In[4]:


m = Model([2,4,4,1])
```

定義了一個有兩層hidden layer的neural network，其中層跟層之間的大小依序為w1:(2*4)、w2:(4*4)、w3:(4*1)，且透過uniform distribution從[0,1]間去取值並附值給weight。

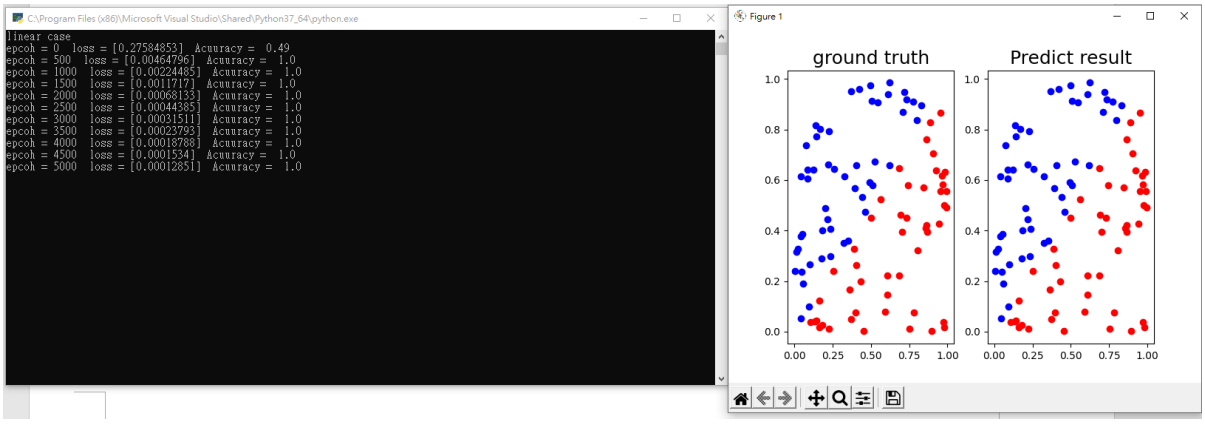
C. Backpropagation

```
    # backward propagation
    dldy3  = 2*(y[i]-y_[i])  *derivative_sigmoid(y[i])# scalar
    dldw3  = z2[i].reshape(4,1) * dldy3  # 符合 dldw3 = 4*1

    dldy2  = (m.weight[2].T * dldy3) * derivative_sigmoid(z2[i]) # 1*4
    dldw2  = z1[i].reshape(4,1) @ dldy2  # 符合 dldw2 = 4*4

    dldy1  = (dldy2 @ m.weight[1].T) * derivative_sigmoid(z1[i]) # 1*4
    dldw1  = x[i].T.reshape(2,1) @ dldy1 # 符合 dldw1 = 2*4
```

透過dldw3 = dldy * dydw(since chain rule) 計算各個gradient
需要注意的是，矩陣乘法不能直接follow偏微分的順序由左乘到右，需要讓size符合 dldw1 = (2*4)、dldw2 = (4*4)、dldw3=(4*1)，所以需要適時地做transpose、reshpae或是交換矩陣相乘的順序。

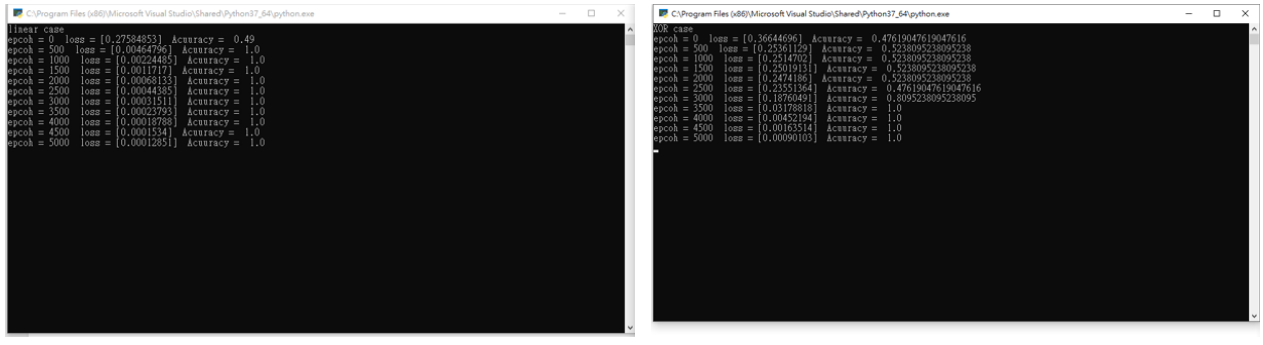
3. Results of your testing (20%)

A. Screenshot and comparison figure



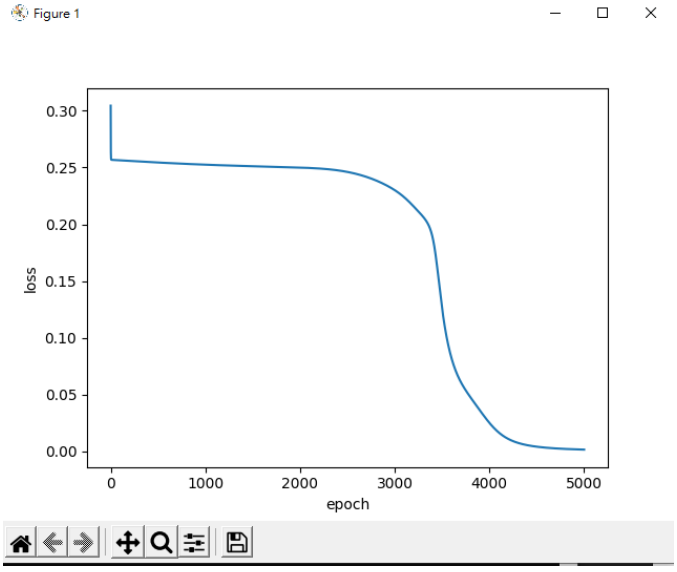
經過5000個epoch後達到預測命中率100%，且可看出訓練XOR比起linear較耗時。

B. Show the accuracy of your prediction



經過5000個epoch後達到預測命中率100%

C. Learning curve (loss, epoch curve)

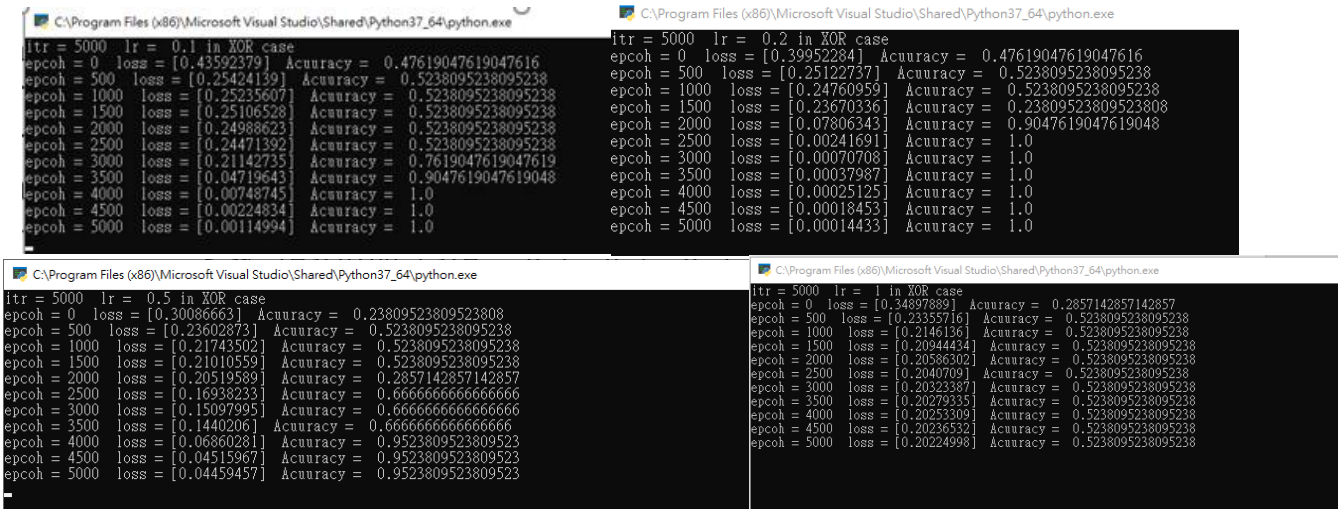


D. anything you want to present

4. Discussion (30%)

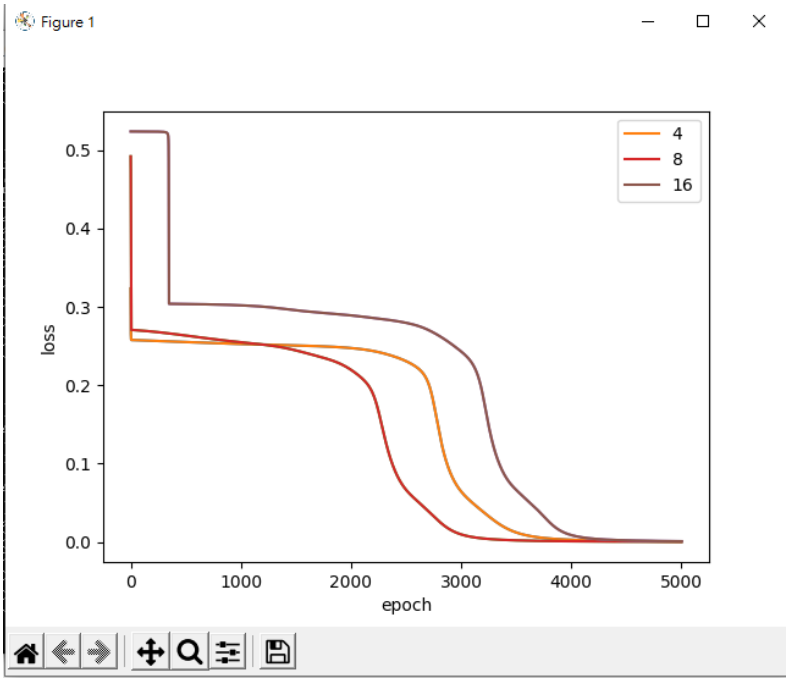
A. Try different learning rates

嘗試 learning rate = 0.1 ,0.2, 0.5 , 1 三種



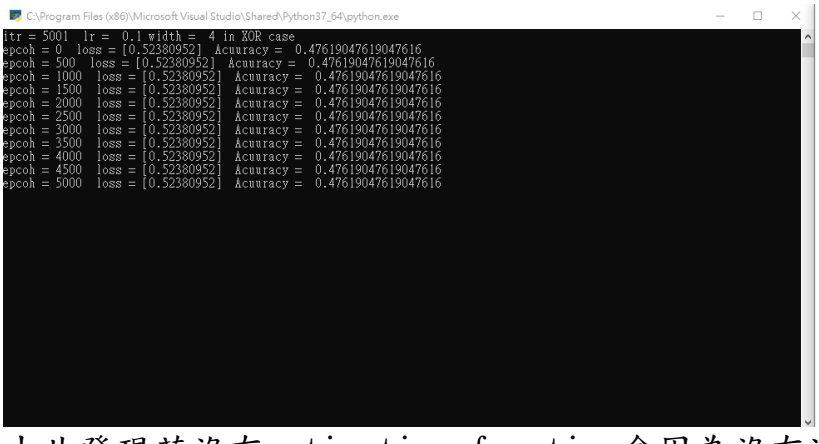
可以看到在epoch次數夠多時，lr足夠小的能達到命中率百分之百，但learning rate太大時(0.5、1)，可能會導致一次跨得步伐太大而無法逼近到minimum。

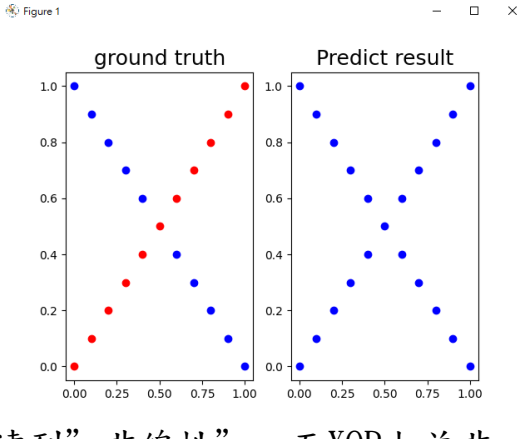
B. Try different numbers of hidden units



嘗試了寬度分別是4、8、16的收斂速率。

C. Try without activation functions





由此發現若沒有activation function會因為沒有達到”非線性”，而XOR也並非線性可劃分開來的圖形，因此train不起來，不work。

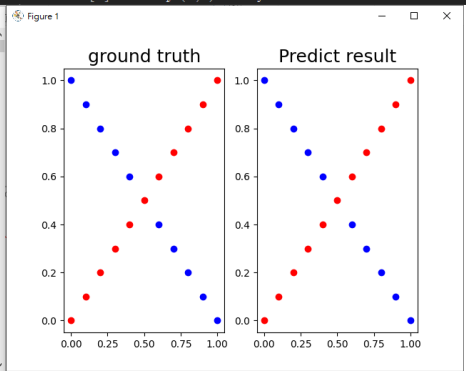
5. Extra (10%)

B. Implement different activation functions. (3)

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

def derivative_relu(x):
    return x>0
```

```
# forward propagation
y1[i] = x[i]  @ m.weight[0]
z1[i] = relu(y1[i])
y2[i] = z1[i] @ m.weight[1]
z2[i] = relu(y2[i])
y3[i] = z2[i] @ m.weight[2]
y[i]  = relu(y3[i])
loss[epoch] = np.square(np.subtract(y, y_)).mean()
# backward propagation
dldy3  = 2*(y[i]-y_[i])  *derivative_relu(y3[i])
dldw3  = z2[i] * dldy3

dldy2  = (m.weight[2].T * dldy3) * derivative_relu(y2[i])
dldw2  = z1[i].reshape(width,1) @ dldy2

dldy1  = (dldy2 @ m.weight[1].T) * derivative_relu(y1[i])
dldw1  = x[i].T.reshape(2,1) @ dldy1
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
itr = 10001  lr =  0.01 width =  8 in XOR case
epoch = 0   loss = [3.71989067]  Acuuracy =  0.5238095238095238
epoch = 500   loss = [0.0683052]  Acuuracy =  0.9523809523809523
epoch = 1000   loss = [0.06294834]  Acuuracy =  0.9523809523809523
epoch = 1500   loss = [0.06039755]  Acuuracy =  0.9523809523809523
epoch = 2000   loss = [0.05921015]  Acuuracy =  0.9523809523809523
epoch = 2500   loss = [0.05786431]  Acuuracy =  0.9523809523809523
epoch = 3000   loss = [0.05702325]  Acuuracy =  0.9523809523809523
epoch = 3500   loss = [0.05576655]  Acuuracy =  0.9523809523809523
epoch = 4000   loss = [0.05480147]  Acuuracy =  0.9523809523809523
epoch = 4500   loss = [0.05387922]  Acuuracy =  0.9523809523809523
epoch = 5000   loss = [0.05277488]  Acuuracy =  0.9523809523809523
epoch = 5500   loss = [0.05180072]  Acuuracy =  0.9523809523809523
epoch = 6000   loss = [0.0067683]  Acuuracy =  1.0
epoch = 6500   loss = [0.004313]  Acuuracy =  1.0
epoch = 7000   loss = [0.0034752]  Acuuracy =  1.0
epoch = 7500   loss = [0.00270158]  Acuuracy =  1.0
epoch = 8000   loss = [0.00201883]  Acuuracy =  1.0
epoch = 8500   loss = [0.00141864]  Acuuracy =  1.0
epoch = 9000   loss = [0.00095802]  Acuuracy =  1.0
epoch = 9500   loss = [0.00066008]  Acuuracy =  1.0
epoch = 10000   loss = [0.00044571]  Acuuracy =  1.0
```



透過另一種 activation function :relu，得到 accuray =1.0 的結果。