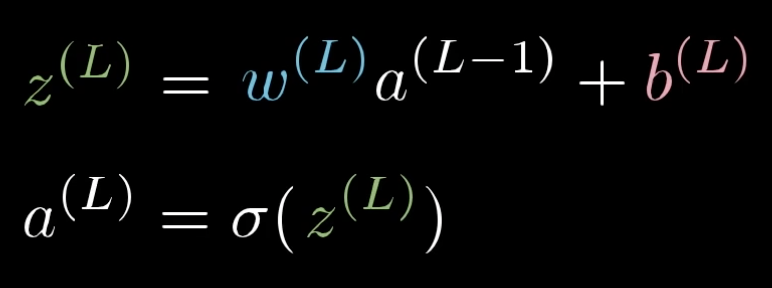
Lab1：Backpropagation

* Introduction：

本次作業用python實作NN with Backpropagation：

* 1. Neural Network：



a：the inputs of each layers

L：the layer number

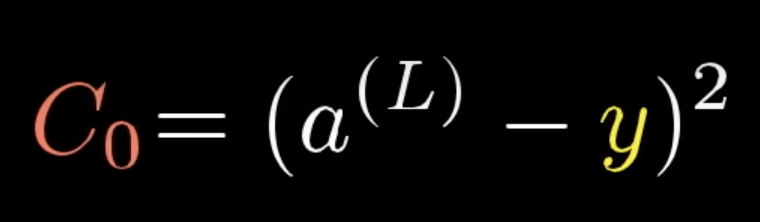
a(L)：output of the layer

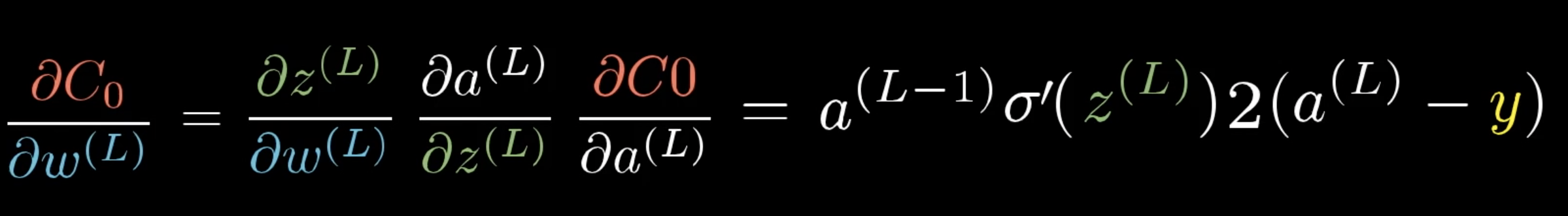
w：weights

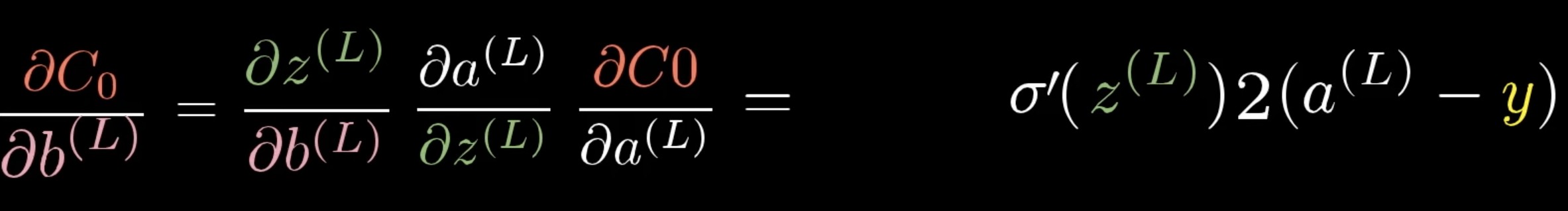
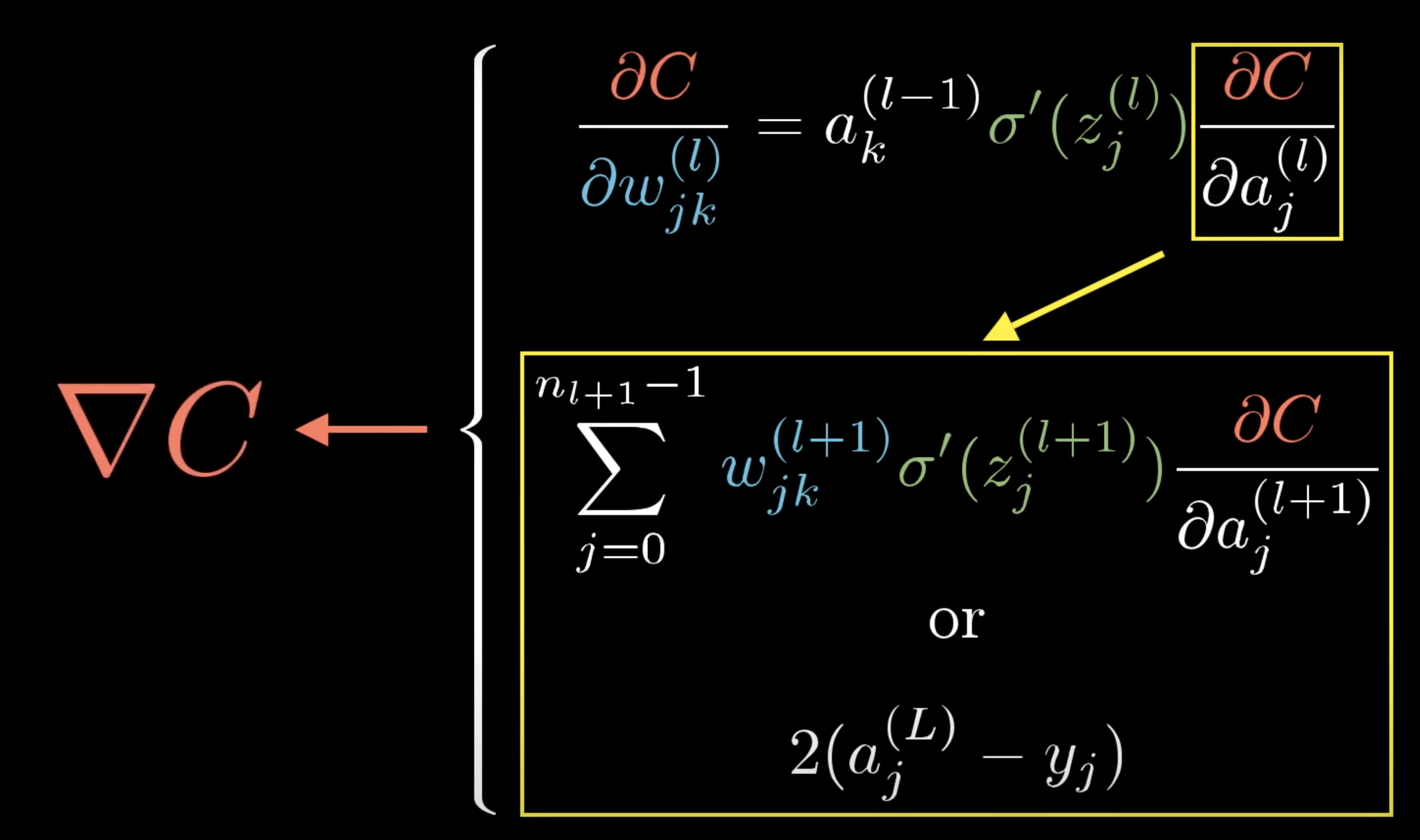
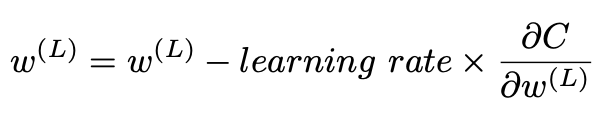
b：bias

σ：activation function

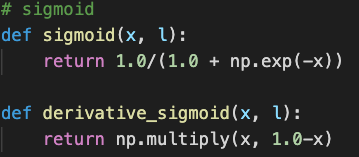
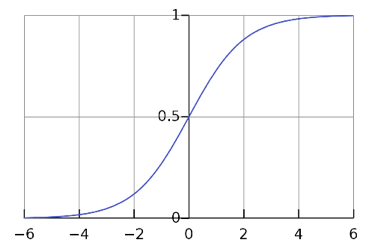
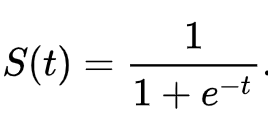
* 1. Backpropagation：

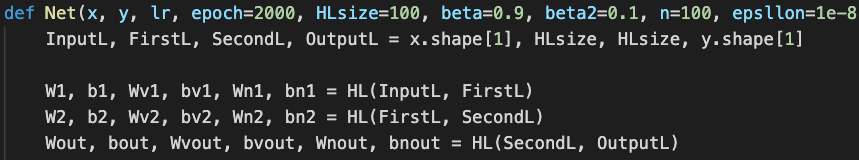
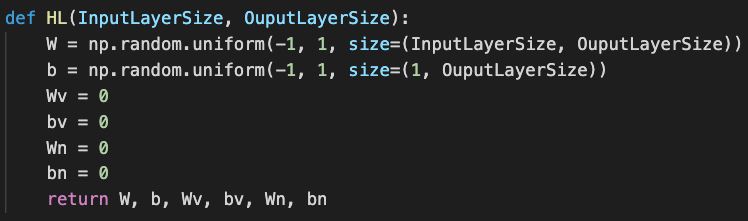
Backpropagation is used to calculate the gradients, starting from the output layer and propagating backwards, and updating weights and biases for each layer. The idea is that we nudge the weights and biases to get the desired output and minimize the cost function. The cost function is defined below: 

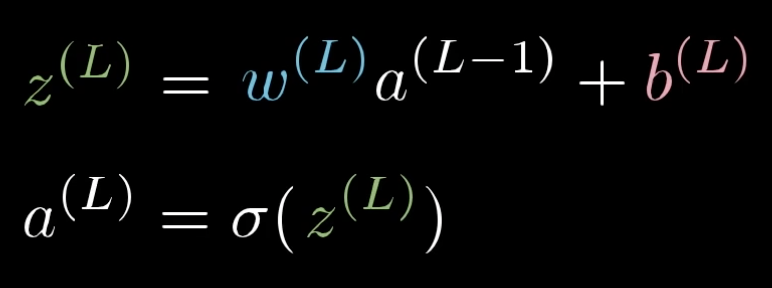
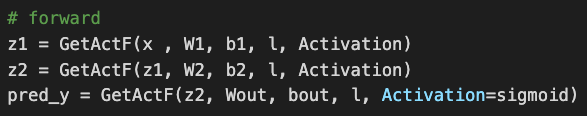
Then use partial derivatives and chain rule to calculate the relationship between the neural network components and the cost function from last layer to first layer. When we know what affects it, we can effectively change the relevant weights and biases to minimize the cost function.   
  
Weights：

Biases：  
  
Gradient：  
  
In the end, we subtract the weights from the weights multiply by the learning rate to update the weights as below:   


* Experiment setups：
  1. Sigmoid functions：

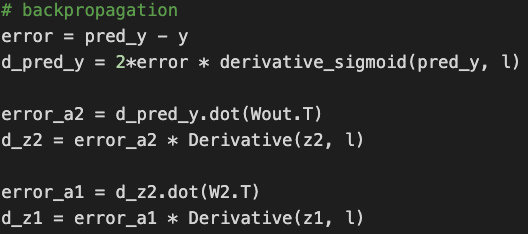
  


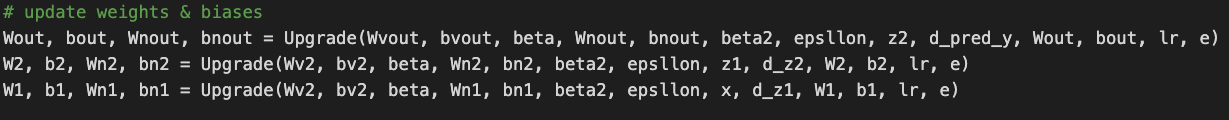
* 1. Neural networks：  
     1. Init：  
     設定好要訓練的epoch數，以及每一層Neurons的數目  
       
     隨機給定初始weight & bias的值  
     

2. Forward：  
  
重複直到最後一層  


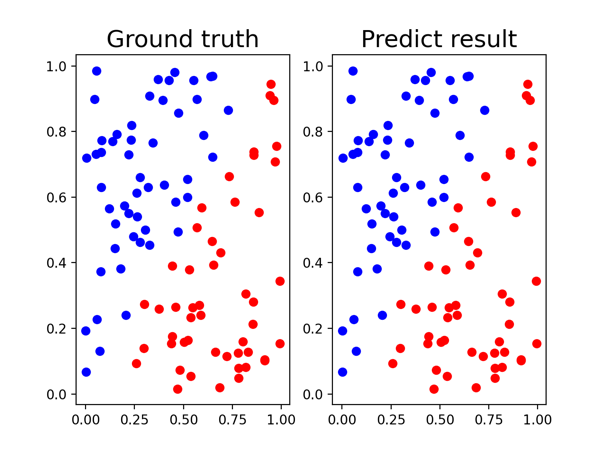
* 1. Backpropagation：

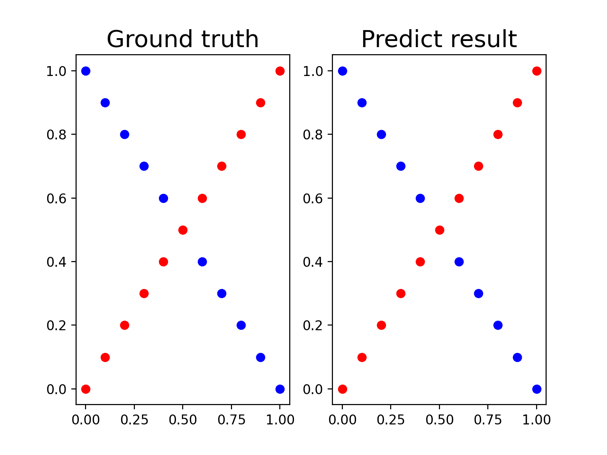
1. 計算每次結果的誤差



2. 更新weights & biases  


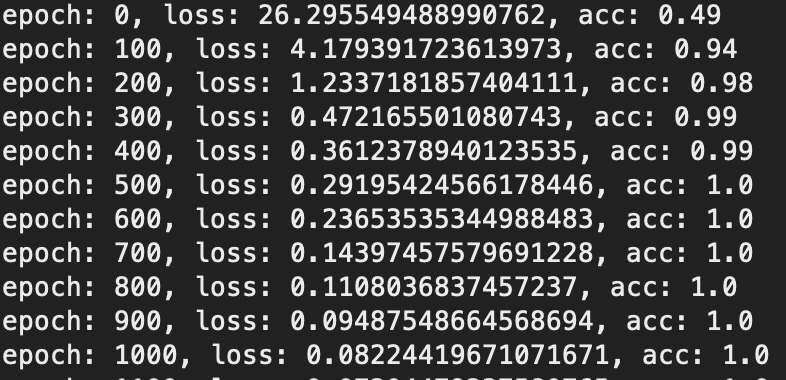
* Results of testing：
  1. Screenshot and comparison figure



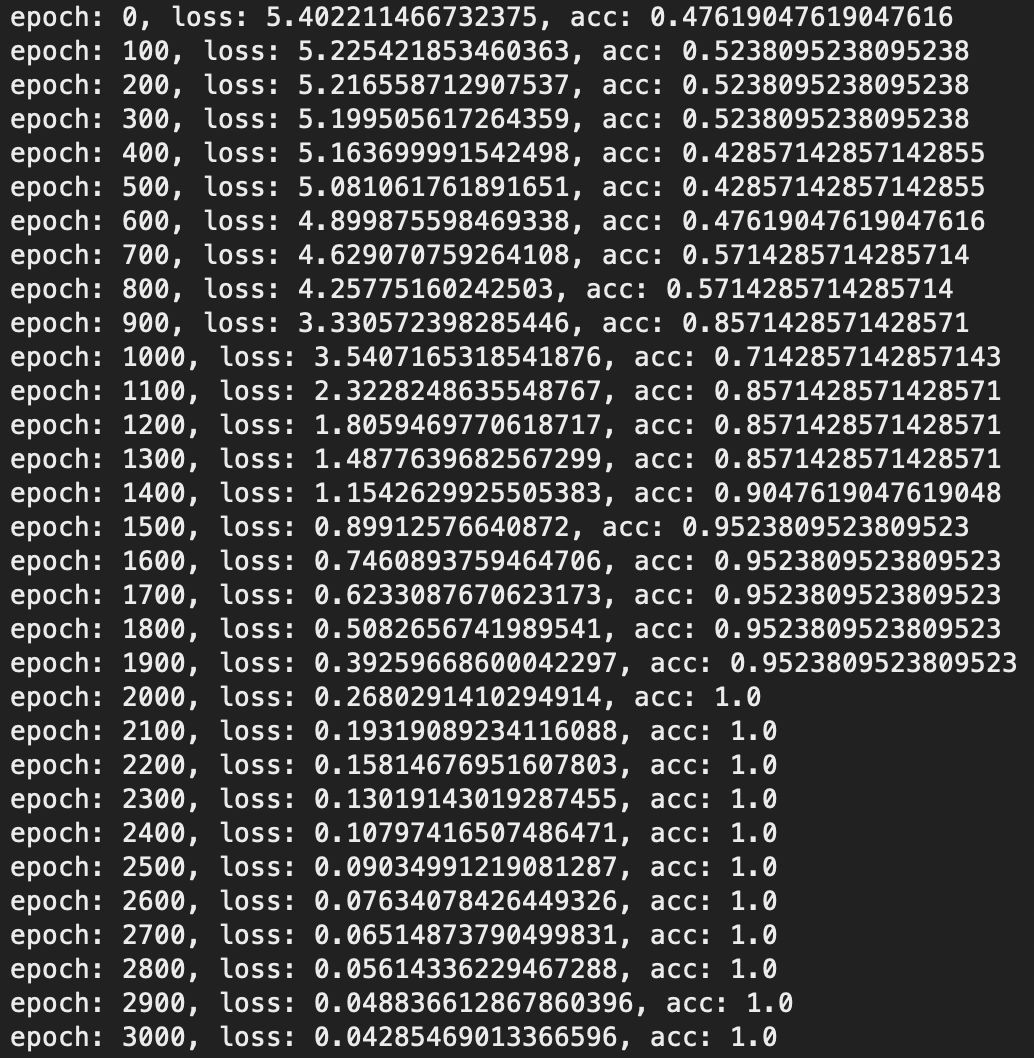
Linear：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

* 1. Show the accuracy of your prediction

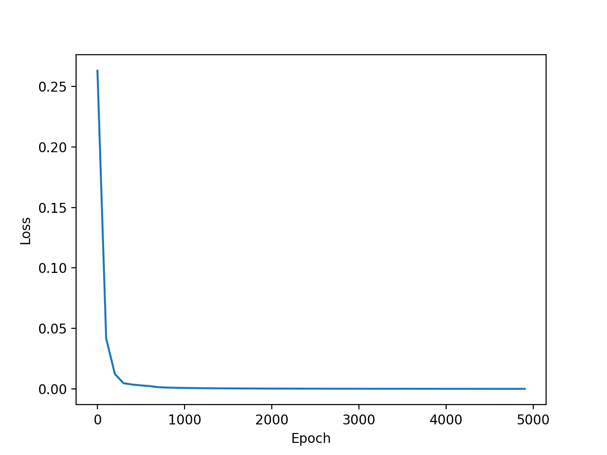


Linear：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

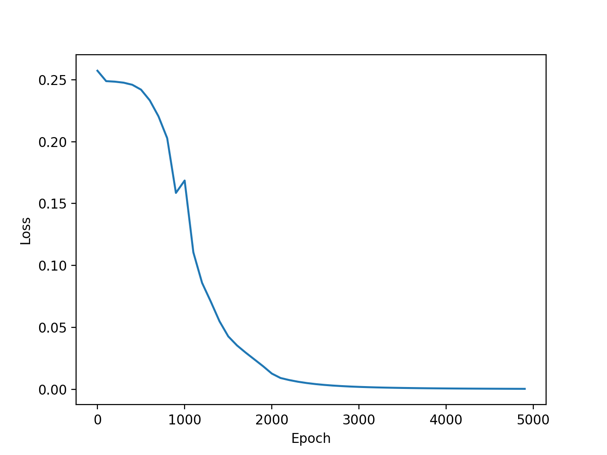


XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

* 1. Learning curve (loss, epoch curve)

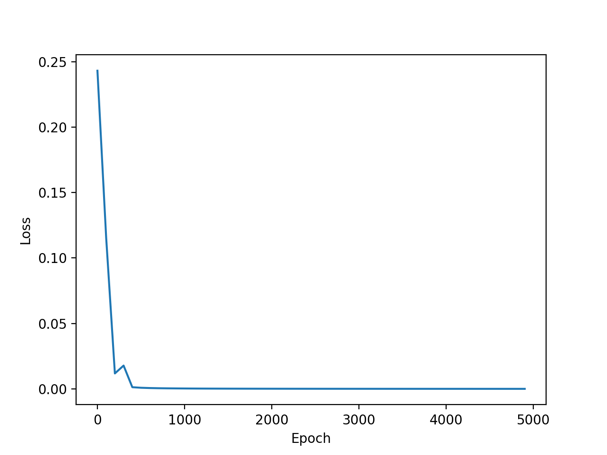
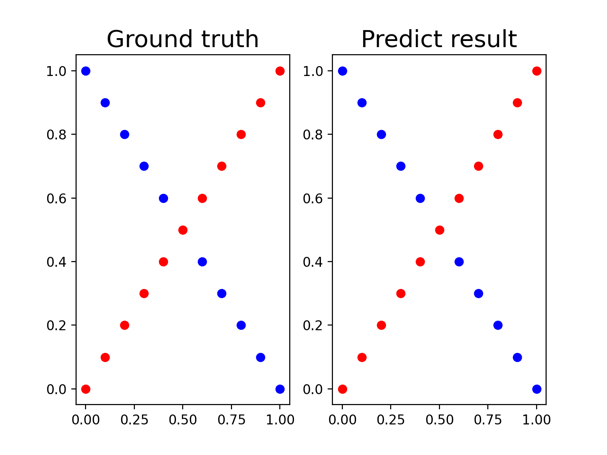


Linear：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

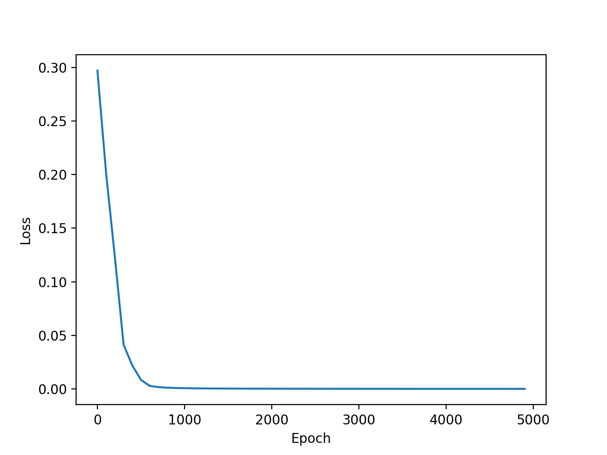
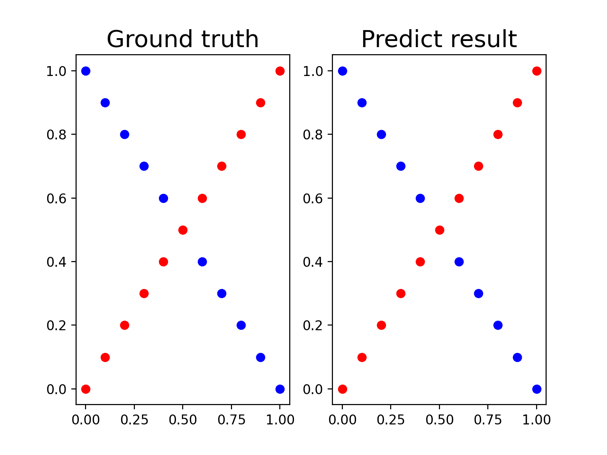


XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

* 1. Other activation：relu, tanh

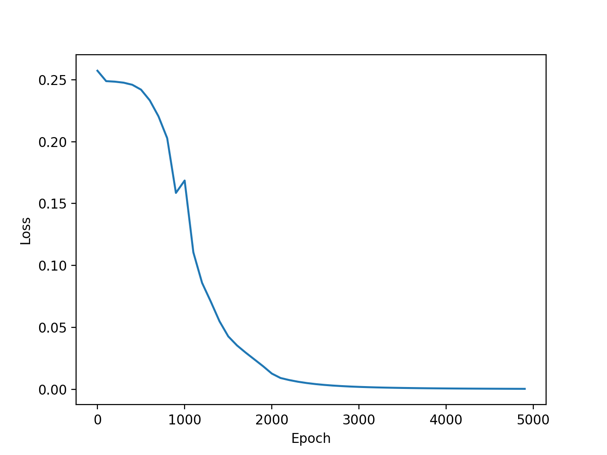
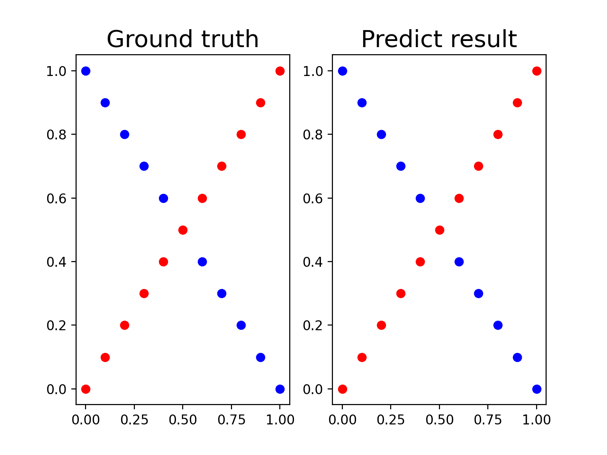


XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=relu, optimizer=GD

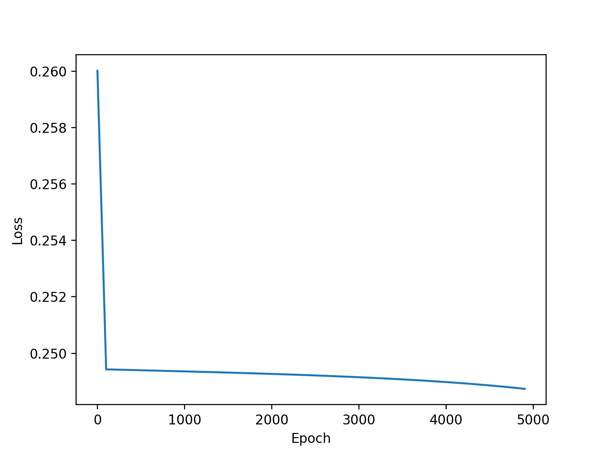
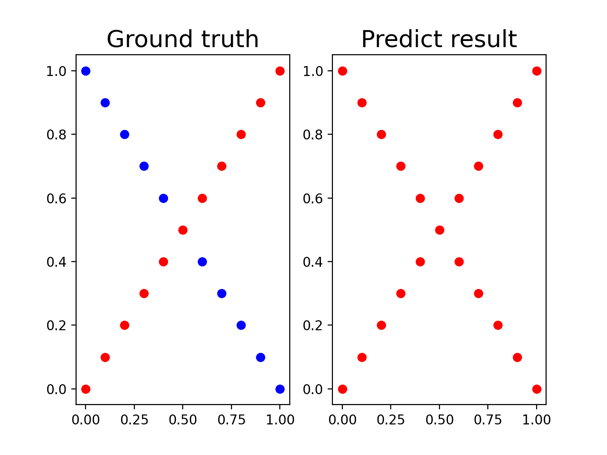


XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=tanh, optimizer=GD

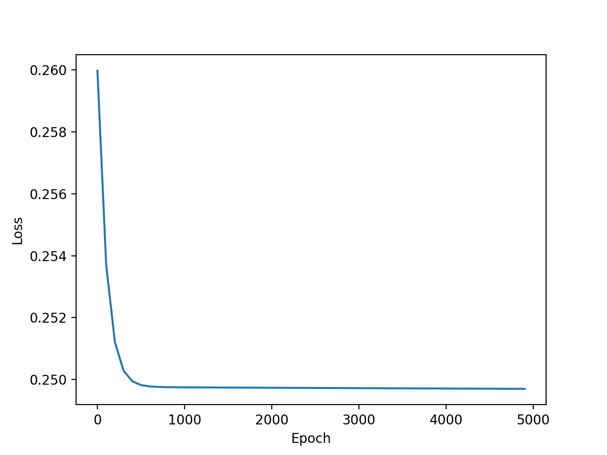
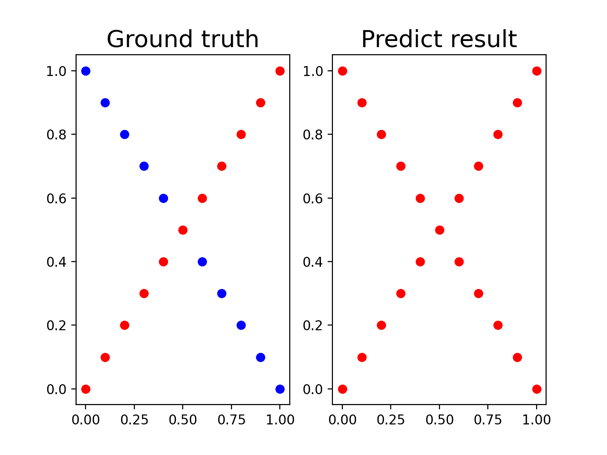
* Discussion：
  1. Try different learning rates：



XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD



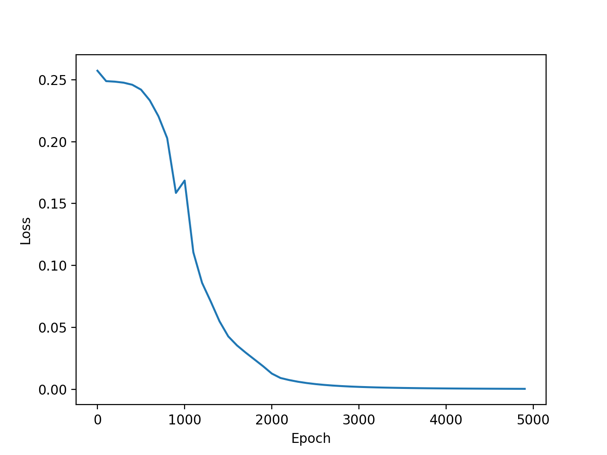
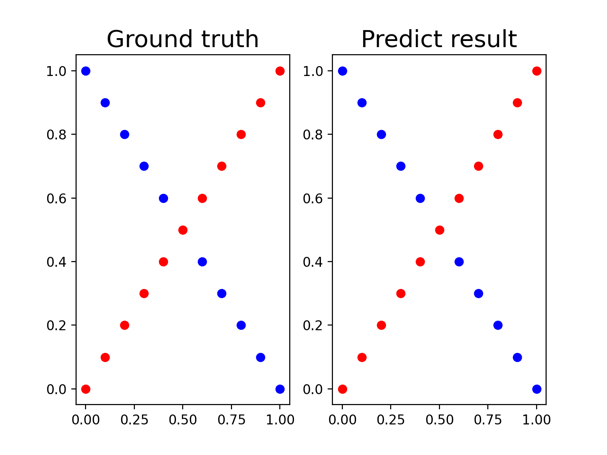
XOR：lr=0.01, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD



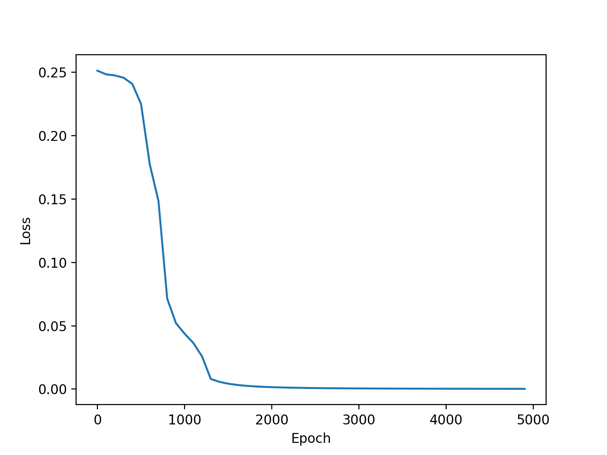
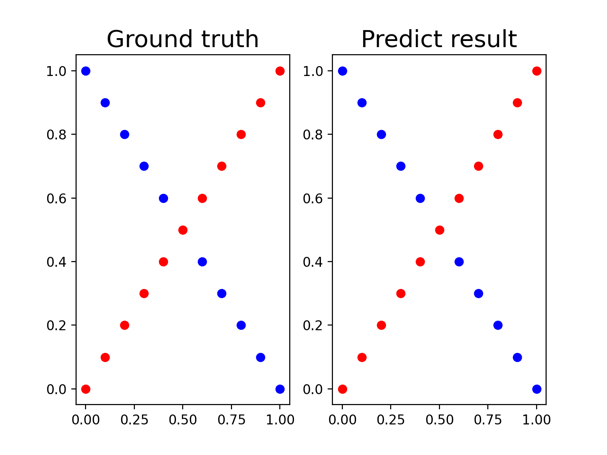
XOR：lr=0.001, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD

當使用sigmoid作為activation function時，learning rate越低，反而容易造成gradient卡住，準確率剩一半，loss卡在0.25。

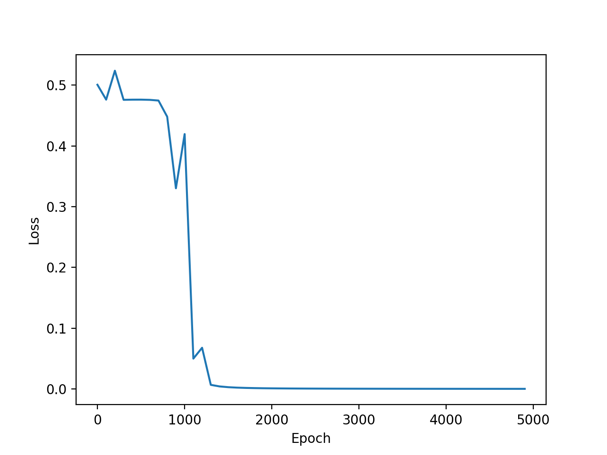
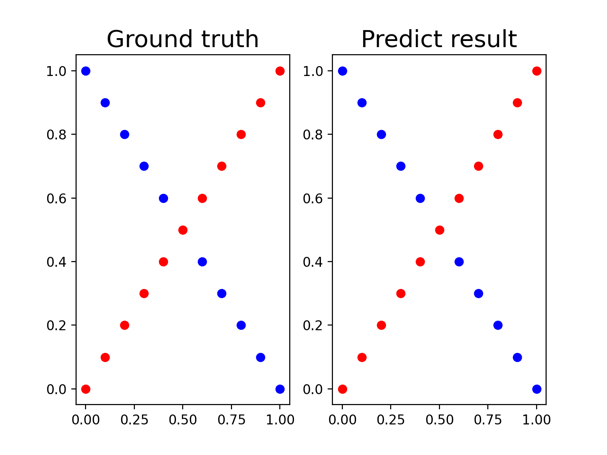
* 1. Try different numbers of hidden units：



XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=GD



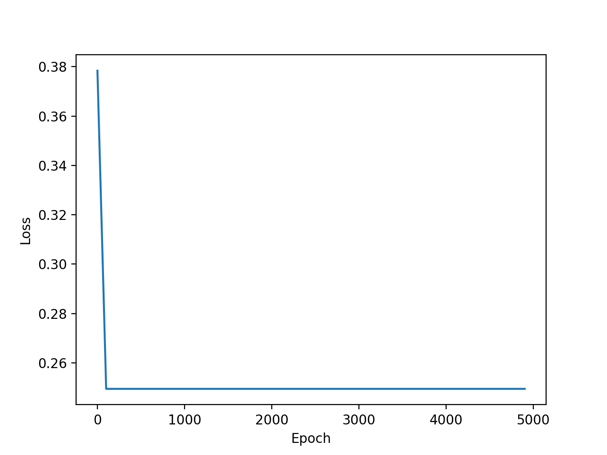
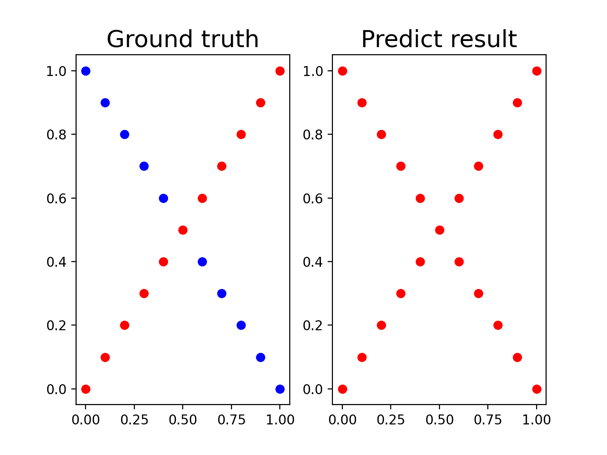
XOR：lr=0.1, epoch=5000, layers=2,8,8,1, activation=sigmoid, optimizer=GD



XOR：lr=0.1, epoch=5000, layers=2,100,100,1, activation=sigmoid, optimizer=GD

在Neuron=4, 8, 100中，模型的預測能力都很好，不過在Neuron=100，epoch=1000以下的loss跳動很大，可能learning rate在neuron=100時太大了。

* 1. Try without activation functions：

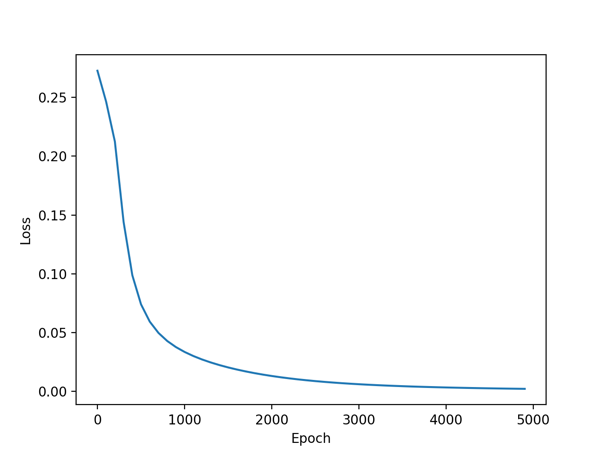
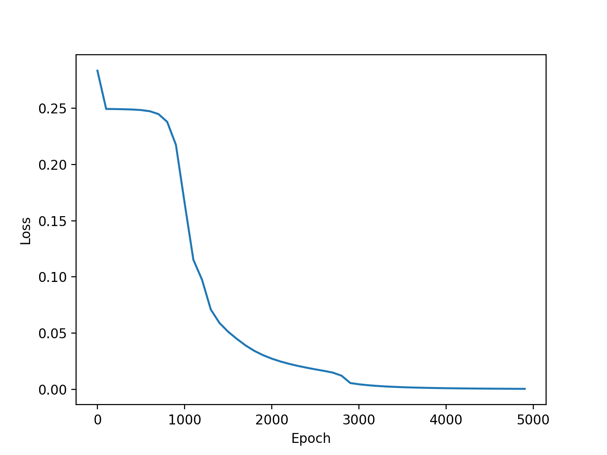


XOR：lr=0.1, epoch=5000, layers=2,4,4,1, activation=none, optimizer=GD

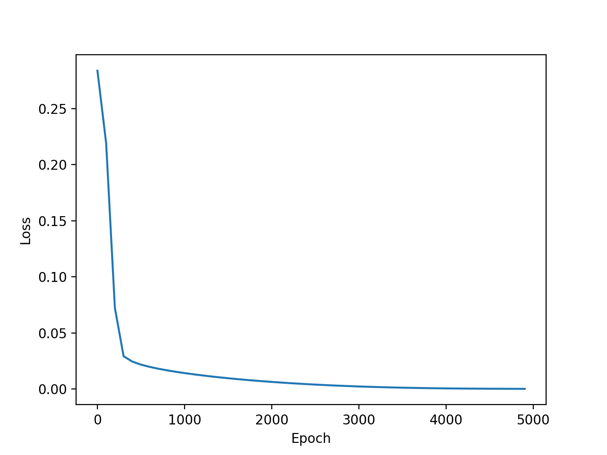
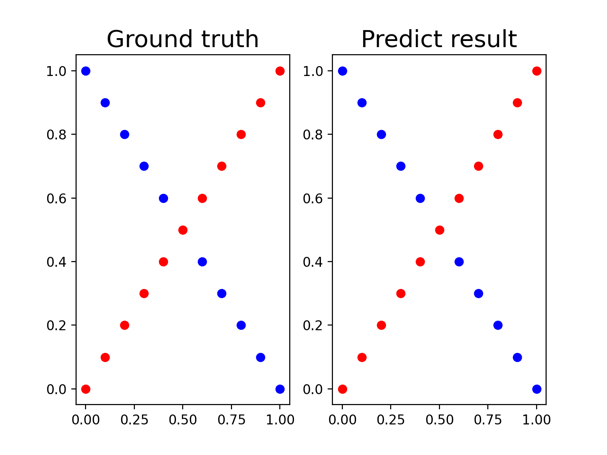
沒有activation function等同於找一個切平面切出最佳解，在XOR中人眼預期切出0.75的準確度，但觀察模型可以發現loss卡在0.25附近，也就是準確度只會有0.5，模型無法切出0.75的平面。

* 1. Other optimizer：momentum, adagram, adam

XOR：lr=0.01, epoch=5000, layers=2,4,4,1, activation=sigmoid



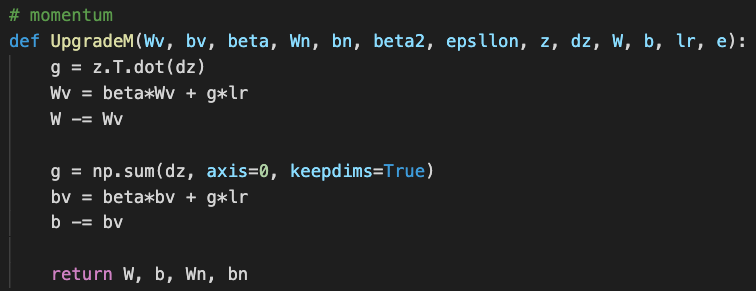
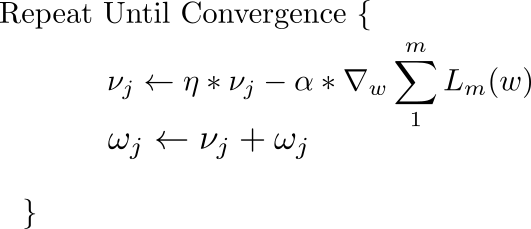
optimizer=momentum optimizer=adagrad



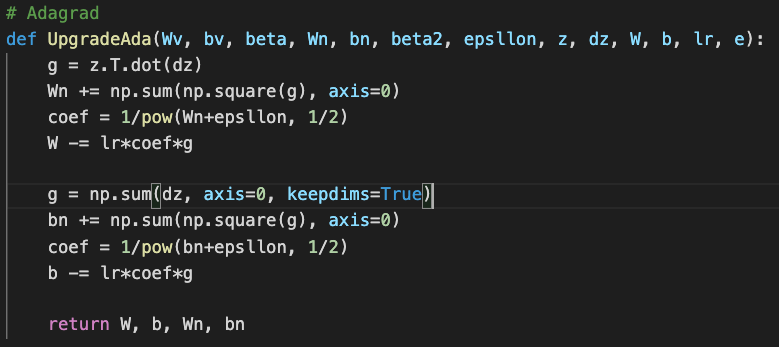
XOR：lr=0.01, epoch=5000, layers=2,4,4,1, activation=sigmoid, optimizer=adam

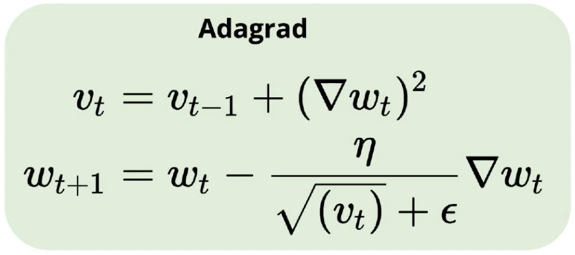
* Extra：
  1. Implement different optimizers：

1. momentum：

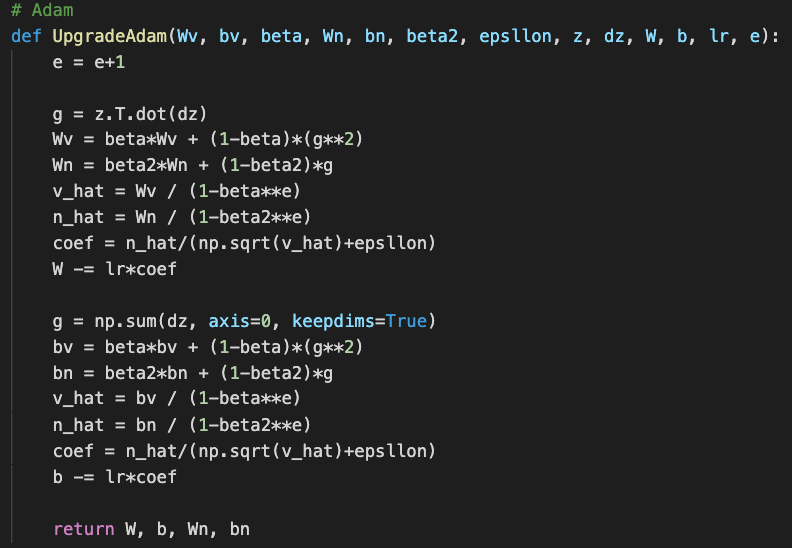
 

2. adagrad：





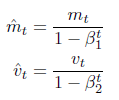
3. adam：



First moment & second moment (mean, variance)



Bias correction

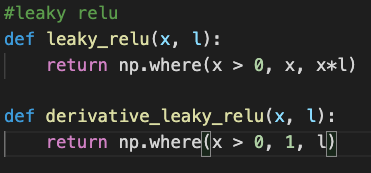


Update parameters

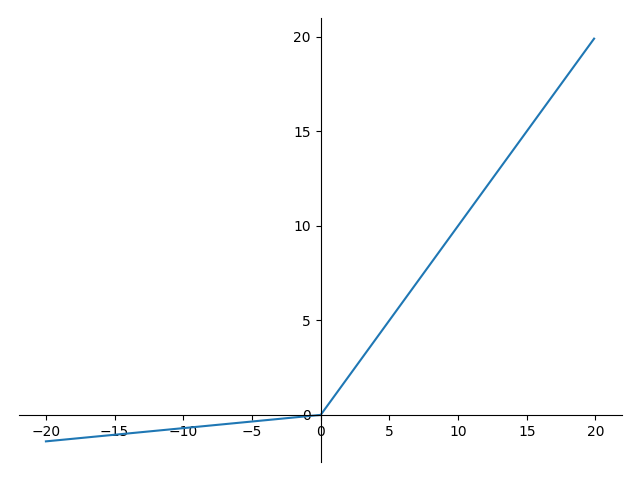
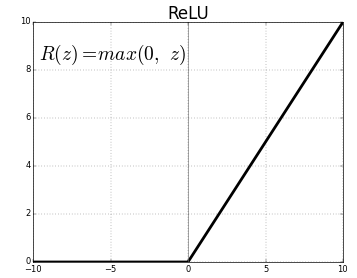


* 1. Implement different activation functions：

1. leaky relu, relu (when l=0)：



Leaky ReLu



2. tanh：

