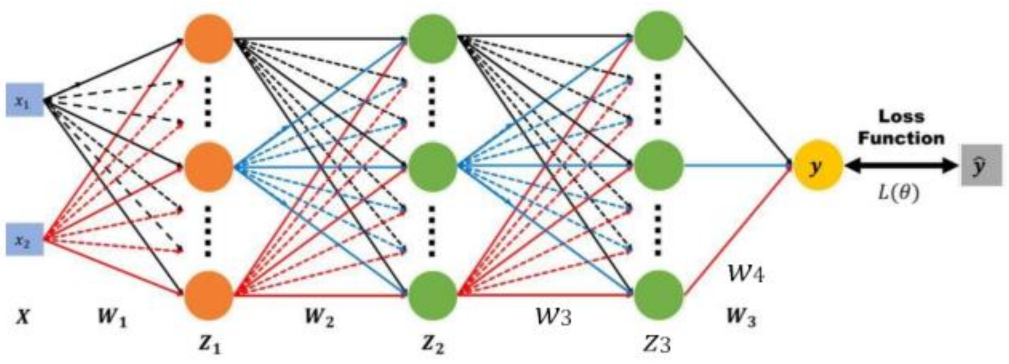


NN 的過程為:

input > weight/bias initialization > forward propagation > loss calculation > back propagation > update weight/bias

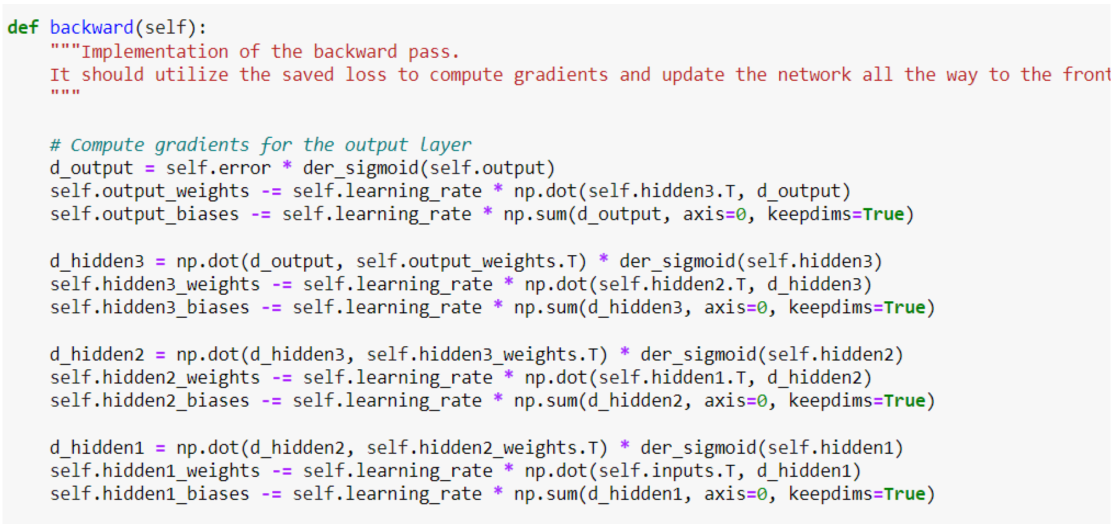
重複執行直到訓練完成。

```python
class SimpleNet:
    def __init__(self, num_step=6000, print_interval=100, learning_rate=0.05):  # don't change print_interval
        """A hand-crafted implementation of simple network.

        Args:
            num_step (optional):    the total number of training steps.
            print_interval (optional):  the number of steps between each reported number.
        """
        self.num_step = num_step
        self.print_interval = print_interval
        self.learning_rate = learning_rate

        # Model parameters initialization
        # hidden layer 1: 100 nodes
        # hidden layer 2: 50 nodes
        # hidden layer 3: 10 nodes
        # Please initiate your network parameters here.

        # Model parameters initialization
        self.input_size = 2
        self.hidden1_size = 100
        self.hidden2_size = 50
        self.hidden3_size = 10
        self.output_size = 1

        self.hidden1_weights = np.random.randn(self.input_size, 100)
        self.hidden1_biases = np.zeros((1, 100))

        self.hidden2_weights = np.random.randn(100, 50)
        self.hidden2_biases = np.zeros((1, 50))

        self.hidden3_weights = np.random.randn(50, 10)
        self.hidden3_biases = np.zeros((1, 10))

        self.output_weights = np.random.randn(10, 1)
        self.output_biases = np.zeros((1, 1))
```

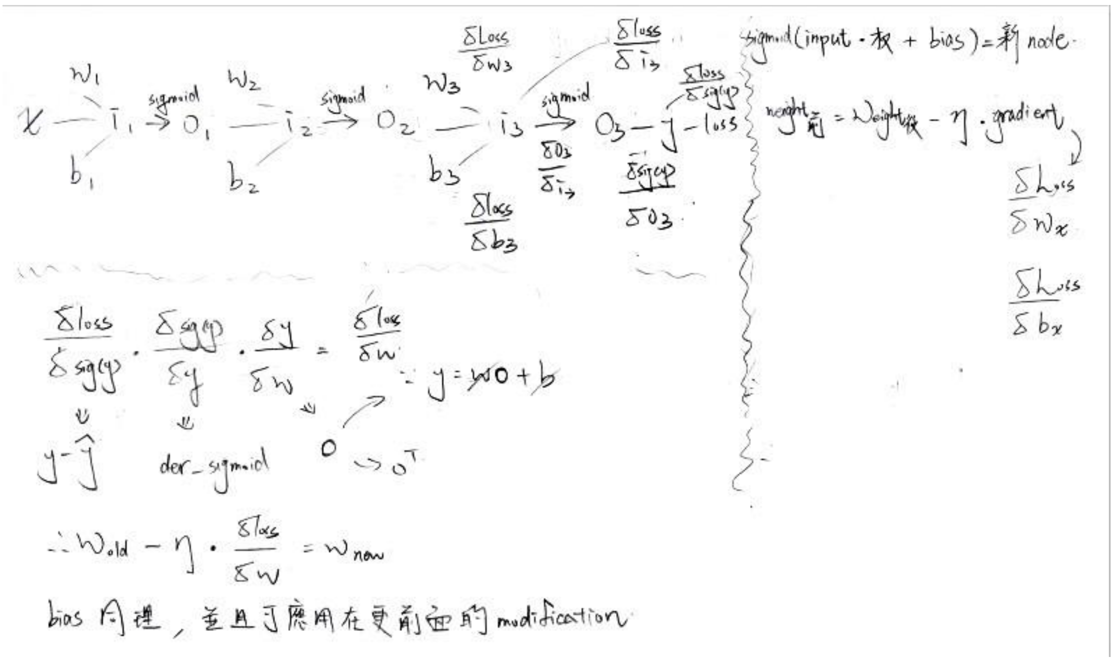
初始化，使兩個輸入經過 hidden layers(先 100 個 nodes>50>10)最後到 output layer，其中 np.random.randn(x,y)代表 hidden layer 和下一層的權重，一共 x*y 個。然後 np.zeros 的部份是要初始化 bias 為 0，以便在訓練中調整。

```python
def forward(self, inputs):
    """Implementation of the forward pass.
    It should accepts the inputs and passing them through the network and return results.
    """
    self.inputs = inputs
    self.hidden1 = sigmoid(np.dot(self.inputs, self.hidden1_weights) + self.hidden1_biases)
    self.hidden2 = sigmoid(np.dot(self.hidden1, self.hidden2_weights) + self.hidden2_biases)
    self.hidden3 = sigmoid(np.dot(self.hidden2, self.hidden3_weights) + self.hidden3_biases)
    self.output = sigmoid(np.dot(self.hidden3, self.output_weights) + self.output_biases)
    return self.output
```

np.dot 讓當前 layer node 和權重進行矩陣乘法，再把 bias 加上去來正確作用 activation function 或是增加模型表現，後面以此類推得到 output。



```python
def backward(self):
    """Implementation of the backward pass.
    It should utilize the saved loss to compute gradients and update the network all the way to the front
    """

    # Compute gradients for the output layer
    d_output = self.error * der_sigmoid(self.output)
    self.output_weights -= self.learning_rate * np.dot(self.hidden3.T, d_output)
    self.output_biases -= self.learning_rate * np.sum(d_output, axis=0, keepdims=True)

    d_hidden3 = np.dot(d_output, self.output_weights.T) * der_sigmoid(self.hidden3)
    self.hidden3_weights -= self.learning_rate * np.dot(self.hidden2.T, d_hidden3)
    self.hidden3_biases -= self.learning_rate * np.sum(d_hidden3, axis=0, keepdims=True)

    d_hidden2 = np.dot(d_hidden3, self.hidden3_weights.T) * der_sigmoid(self.hidden2)
    self.hidden2_weights -= self.learning_rate * np.dot(self.hidden1.T, d_hidden2)
    self.hidden2_biases -= self.learning_rate * np.sum(d_hidden2, axis=0, keepdims=True)

    d_hidden1 = np.dot(d_hidden2, self.hidden2_weights.T) * der_sigmoid(self.hidden1)
    self.hidden1_weights -= self.learning_rate * np.dot(self.inputs.T, d_hidden1)
    self.hidden1_biases -= self.learning_rate * np.sum(d_hidden1, axis=0, keepdims=True)
```

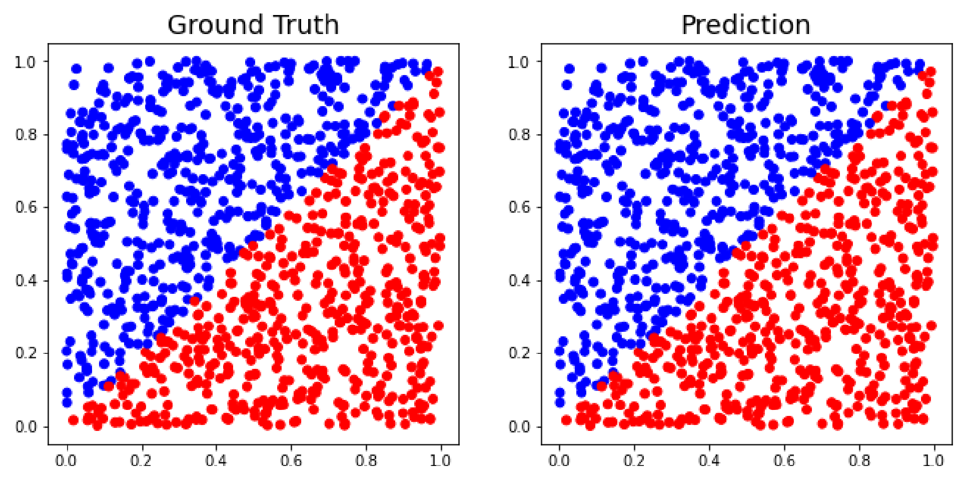


x — i_1 $\xrightarrow{\text{sigmoid}}$ o_1 — i_2 $\xrightarrow{\text{sigmoid}}$ o_2 — i_3 $\xrightarrow{\text{sigmoid}}$ o_3 — y — loss
w_1 b_1 w_2 b_2 w_3 b_3
$\frac{\delta Loss}{\delta w_3}$ $\frac{\delta loss}{\delta i_3}$ $\frac{\delta loss}{\delta sig(y)}$ $\frac{\delta o_3}{\delta i_3}$ $\frac{\delta sig(y)}{\delta o_3}$ $\frac{\delta loss}{\delta b_3}$
sigmoid(input · 權 + bias) = 新 node
weight新 = Weight舊 − η · gradient
$\frac{\delta Loss}{\delta w_x}$
$\frac{\delta Loss}{\delta b_x}$
$\frac{\delta loss}{\delta sig(y)} \cdot \frac{\delta sig(y)}{\delta y} \cdot \frac{\delta y}{\delta w} = \frac{\delta loss}{\delta w}$
$\because y = wo + b$
$y - \hat{y}$ der_sigmoid $o \to o^T$
$\therefore W_{old} - \eta \cdot \frac{\delta loss}{\delta w} = w_{new}$
bias 同理，並且可應用在更前面的 modification

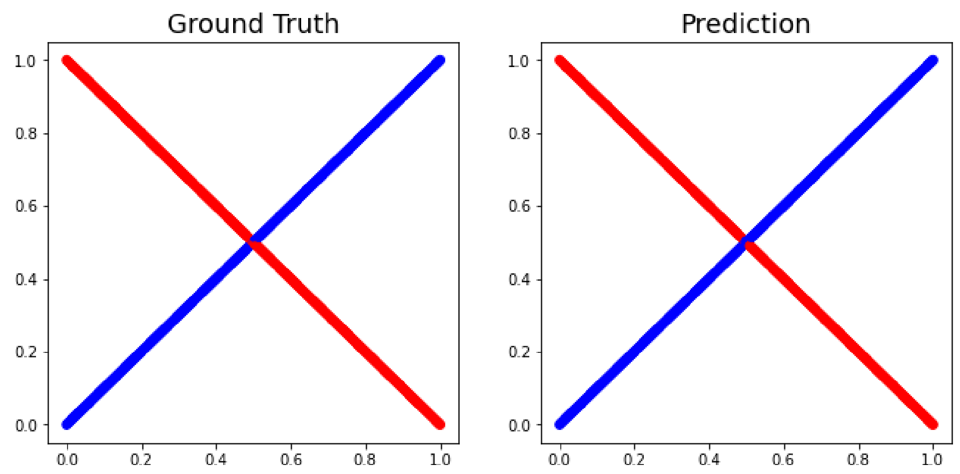
在每一步記錄當前參數，在下一個 back propagation 可以減少同樣內容的計算

Epoch 數增加至 10000，learning rate 改成 0.01

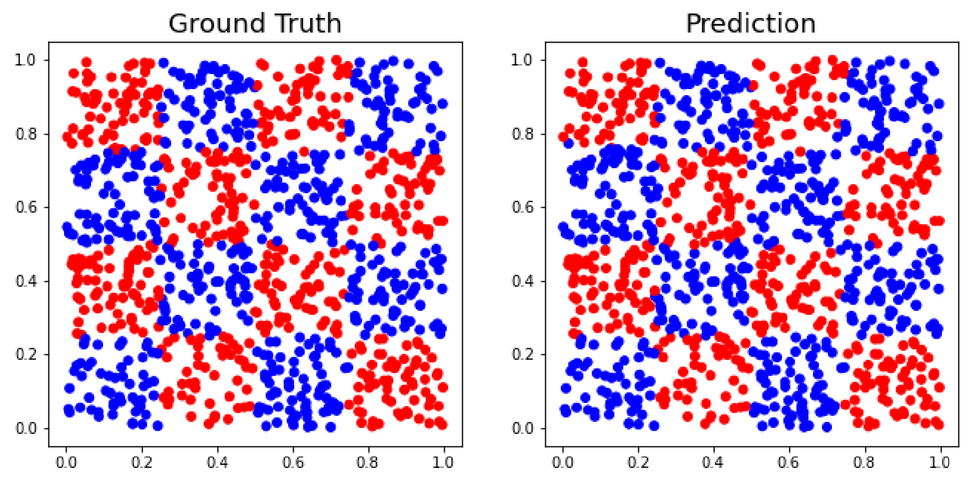
```
Training finished
accuracy: 99.52%
```



```
Training finished
accuracy: 98.62%
```



```
Training finished
accuracy: 97.45%
```


Ground Truth
Prediction