國立陽明交通大學 深度學習實驗 LAB1

313553036 多工所 1 葉軒宇

1. Introduction

這次作業為在不利用 Tensorflow、Pytorch 等函式庫的情況下實作具有 forward、back propagation 與 activation function 之 Neural Network。並使用 linear 與 XOR 之 data 對 Neural Network 進行訓練，觀測不同 learning rate、neural width 對整體的影響。

2. Experiment setups

a. Sigmoid functions

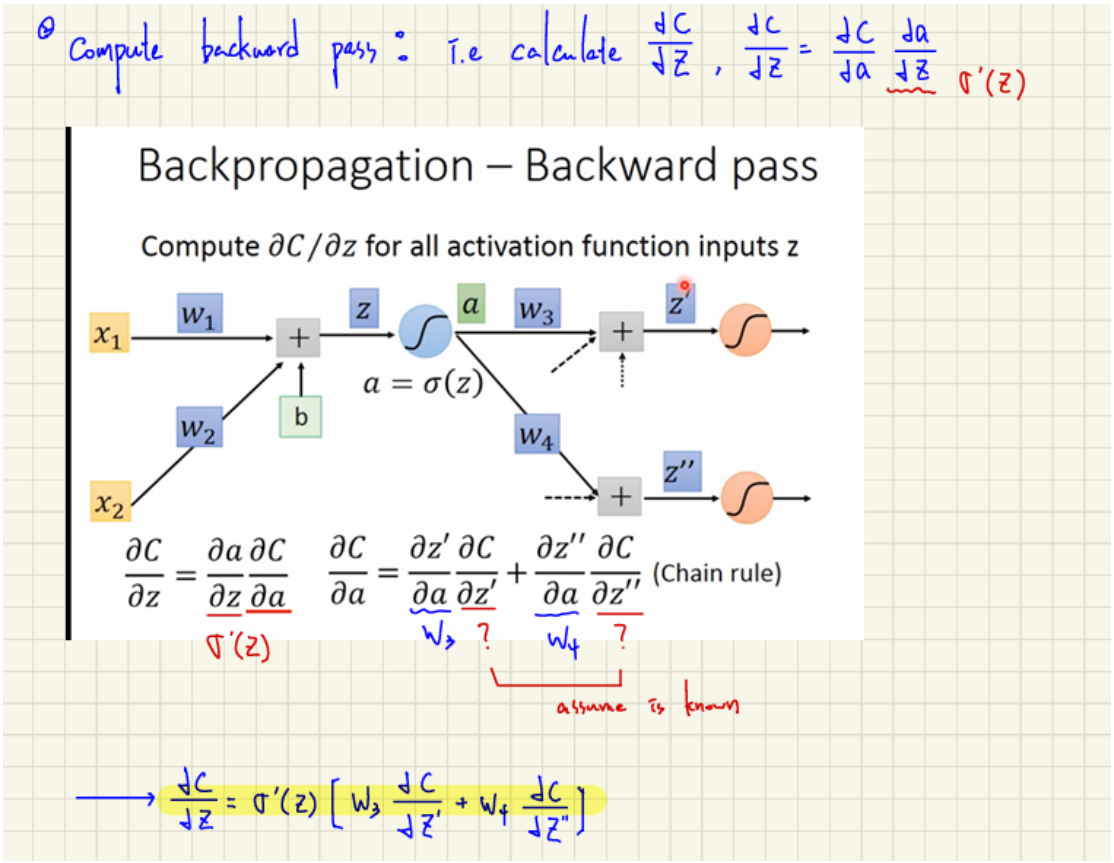
```python
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def derivative_sigmoid(x):
    return x * (1.0 - x)
```

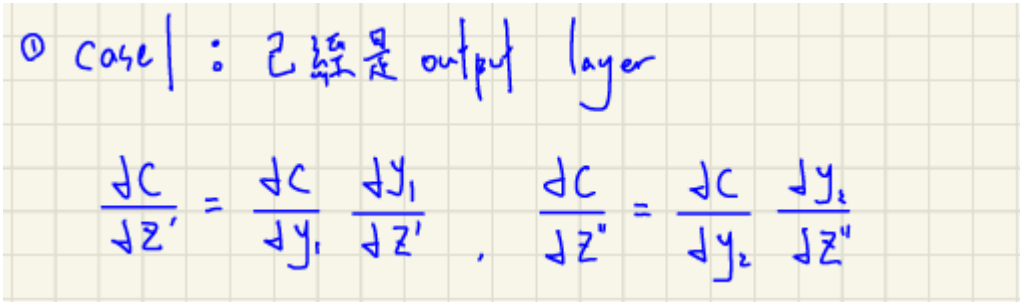
b. Neural networks

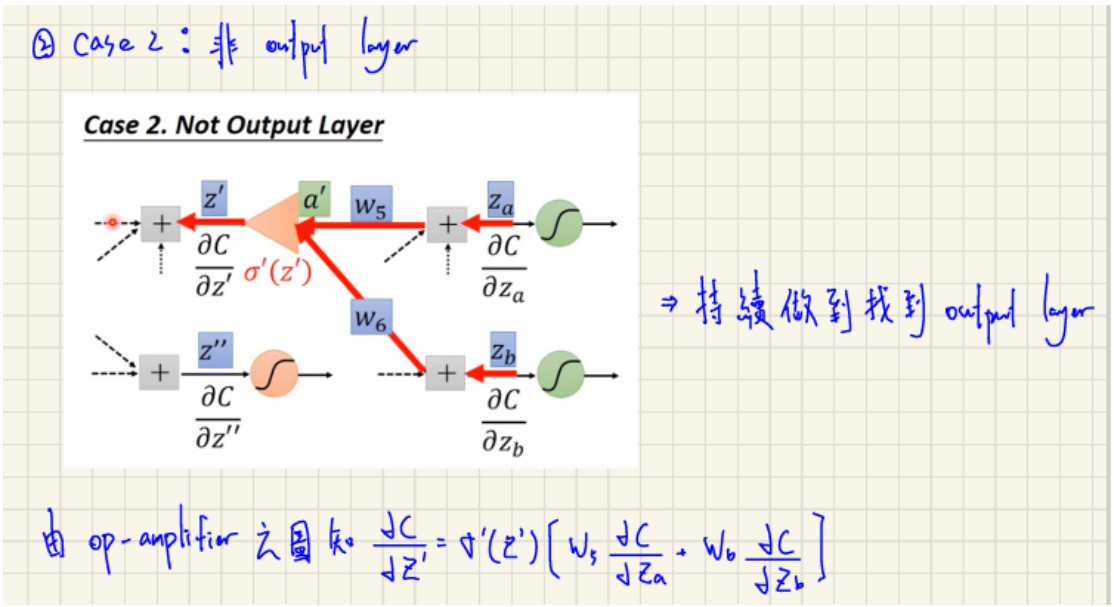
```python
class Model:
    def __init__(self, input_size = 2, hidden_size = 10, lr = 0.1):
        self.layer1 = Layer(input_size, hidden_size)
        self.layer2 = Layer(hidden_size, hidden_size)
        self.output = Layer(hidden_size, output_size: 1)
        self.lr = lr
        self.loss = []
        self.epoch = 0

    def train(self, x, y, epoch):
        self.epoch = epoch
        for i in range(epoch):
            output = self.output.forward(self.layer2.forward(self.layer1.forward(x)))
            loss, grad = MSELoss(output, y)
            self.layer1.backward(self.layer2.backward(self.output.backward(grad, self.lr), self.lr), self.lr)
            self.loss.append(loss)
            if i % 5000 == 0: #control how many cycle print loss
                print(f"epoch {i}  loss : {loss}")
        self.prediction = output
        plt.subplot( *args: 2, 1, 1)
        plt.title( label: "Learning Curve", fontsize=18)
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.plot(self.loss)
        return output

    def show_result(self, x, y):
        print(f"Accuracy : {sum((self.prediction > 0.5) == (y == 1)) / y.size}")
        print("Prediction : ")
        for i in range(y.size):
            print(f"Iter{i + 1} |    Ground truth: {y[i]} |     prediction: {self.prediction[i]} |")
        show_result(x, y, self.prediction > 0.5)

plt.figure(figsize=(12, 12))
```

建立一個具有兩層 hidden layer 的神經網路，具有線性以及 activation function 的部分，Forward 以及 backward 的部分則是包含在 layer class 裡面。在 model 的最後面則是會輸出學習曲線。

c. Back propagation







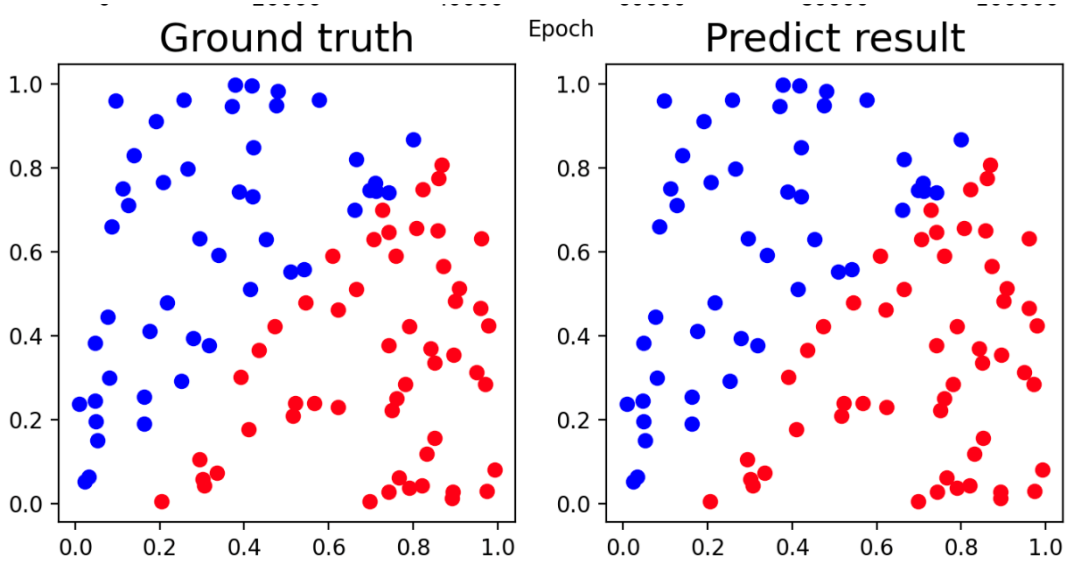

```
class Layer:
    def __init__(self, input_size, output_size):
        self.input_size = input_size
        self.output_size = output_size
        self.w = np.random.randn(input_size, output_size)

    def forward(self, x):
        self.x = x
        self.z = sigmoid(np.dot(x, self.w))
        return self.z

    def backward(self, pre_g, lr):
        self.w -= np.dot(self.x.T, (pre_g * derivative_sigmoid(self.z))) * lr
        return np.dot((pre_g * derivative_sigmoid(self.z)), self.w.T)
```

這邊定義了每一層 hidden layer 中 forward 與 backward 的計算過程。

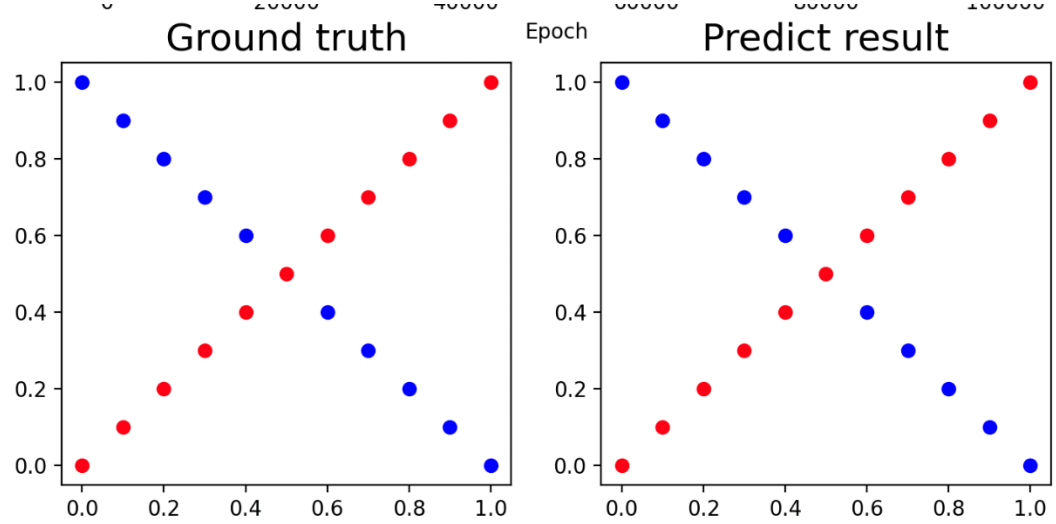
3. Result of your testing

a. Screenshot and comparison figure

Linear:



XOR:



b. Show the accuracy of your prediction

Learning rate = 0.1, hidden units = 10, using SGD to optimize

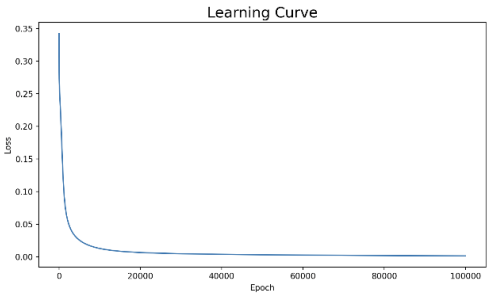
Linear:

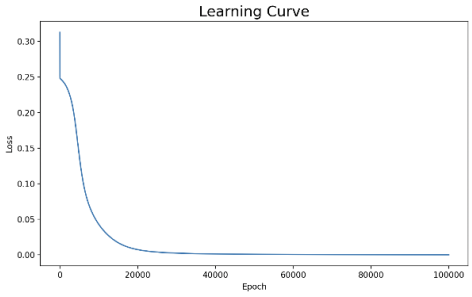
```
Iter1 |    Ground truth: [0] |     prediction: [1.08295064e-05] |
Iter2 |    Ground truth: [0] |     prediction: [1.12183416e-05] |
Iter3 |    Ground truth: [1] |     prediction: [0.99994474] |
Iter4 |    Ground truth: [0] |     prediction: [7.21906516e-05] |
Iter5 |    Ground truth: [1] |     prediction: [0.99624616] |
Iter6 |    Ground truth: [0] |     prediction: [1.25404731e-05] |
Iter7 |    Ground truth: [1] |     prediction: [0.99993656] |
Iter8 |    Ground truth: [0] |     prediction: [3.79155452e-05] |
Iter9 |    Ground truth: [1] |     prediction: [0.99990859] |
Iter10 |    Ground truth: [1] |     prediction: [0.99994601] |
Iter11 |    Ground truth: [0] |     prediction: [1.1098067e-05] |
Iter12 |    Ground truth: [1] |     prediction: [0.99990169] |
Iter13 |    Ground truth: [1] |     prediction: [0.99959192] |
Iter14 |    Ground truth: [0] |     prediction: [4.34922614e-05] |
Iter15 |    Ground truth: [0] |     prediction: [0.00010552] |
Iter16 |    Ground truth: [0] |     prediction: [1.19640821e-05] |
Iter17 |    Ground truth: [0] |     prediction: [1.03780129e-05] |
Iter18 |    Ground truth: [1] |     prediction: [0.99984451] |
Iter19 |    Ground truth: [1] |     prediction: [0.99947852] |
```

XOR:

```
Prediction :
Iter1 |    Ground truth: [0] |     prediction: [0.00042637] |
Iter2 |    Ground truth: [1] |     prediction: [0.99253175] |
Iter3 |    Ground truth: [0] |     prediction: [0.00142423] |
Iter4 |    Ground truth: [1] |     prediction: [0.99693925] |
Iter5 |    Ground truth: [0] |     prediction: [0.00516162] |
Iter6 |    Ground truth: [1] |     prediction: [0.99862674] |
Iter7 |    Ground truth: [0] |     prediction: [0.01433159] |
Iter8 |    Ground truth: [1] |     prediction: [0.99875469] |
Iter9 |    Ground truth: [0] |     prediction: [0.02460738] |
Iter10 |    Ground truth: [1] |     prediction: [0.96697704] |
Iter11 |    Ground truth: [0] |     prediction: [0.02643264] |
Iter12 |    Ground truth: [0] |     prediction: [0.02015218] |
Iter13 |    Ground truth: [1] |     prediction: [0.96413894] |
Iter14 |    Ground truth: [0] |     prediction: [0.01253923] |
Iter15 |    Ground truth: [1] |     prediction: [0.99965733] |
Iter16 |    Ground truth: [0] |     prediction: [0.00714381] |
Iter17 |    Ground truth: [1] |     prediction: [0.99993677] |
Iter18 |    Ground truth: [0] |     prediction: [0.00403535] |
Iter19 |    Ground truth: [1] |     prediction: [0.99996938] |
Iter20 |    Ground truth: [0] |     prediction: [0.00236733] |
Iter21 |    Ground truth: [1] |     prediction: [0.99997883] |
```

c. Learning curve(loss, epoch curva)





linear

```
epoch 0 loss : 0.34248978819407855
epoch 10000 loss : 0.012865494885691078
epoch 20000 loss : 0.00663867735646655
epoch 30000 loss : 0.004795368880659037
epoch 40000 loss : 0.003833102415110956
epoch 50000 loss : 0.00318373049250351
epoch 60000 loss : 0.0026900620496413314
epoch 70000 loss : 0.0022936451502687805
epoch 80000 loss : 0.0019675277335485177
epoch 90000 loss : 0.0016964984165490611
Accuracy : [1.]
```

XOR

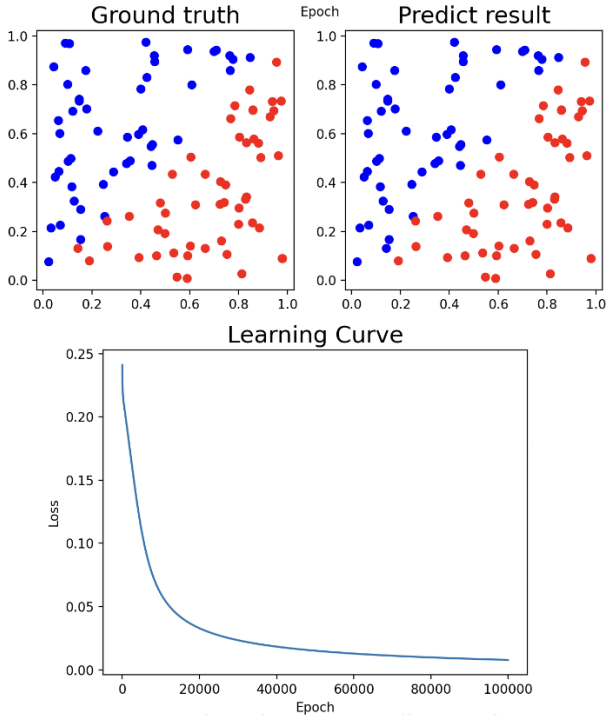
```
epoch 0 loss : 0.31266926275298734
epoch 10000 loss : 0.04294103048980748
epoch 20000 loss : 0.007426536065405919
epoch 30000 loss : 0.002499919806726341
epoch 40000 loss : 0.0013116816174656865
epoch 50000 loss : 0.0008434184691277308
epoch 60000 loss : 0.0006053356064198953
epoch 70000 loss : 0.0004648586742633025 4
epoch 80000 loss : 0.0003735749303861188
epoch 90000 loss : 0.00031012656770461794
Accuracy : [1.]
```

4. Discussion

a. Try different learning rates

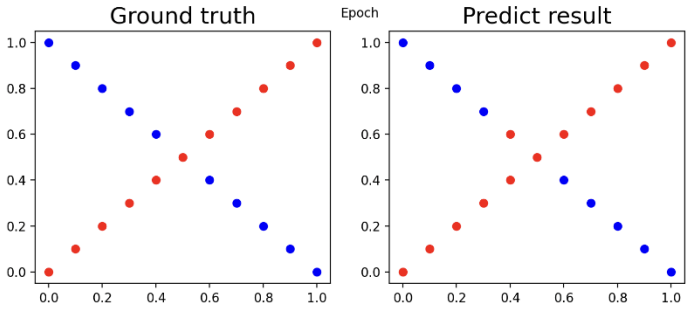
Learning rate = 0.01, hidden units = 10, using SGD to optimize

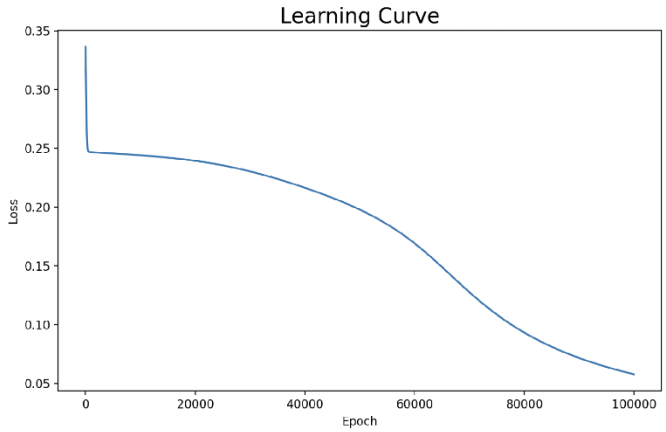
Linear:



當 learning rate 下降時，loss 下降速率也變慢，但在整體上來說 accuracy 都可以保持在相當高的數值。

XOR:

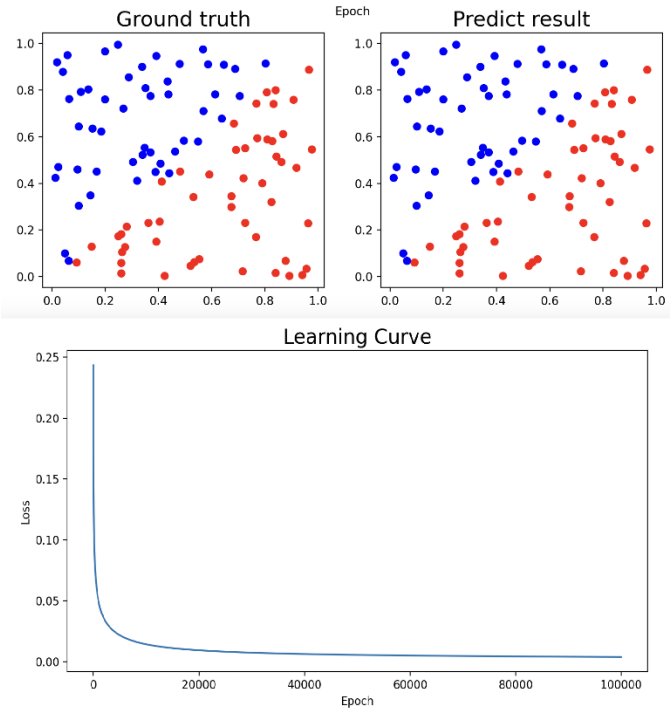




b. Try different numbers of hidden units

Linear with hidden units = 50, learning rate = 0.1:

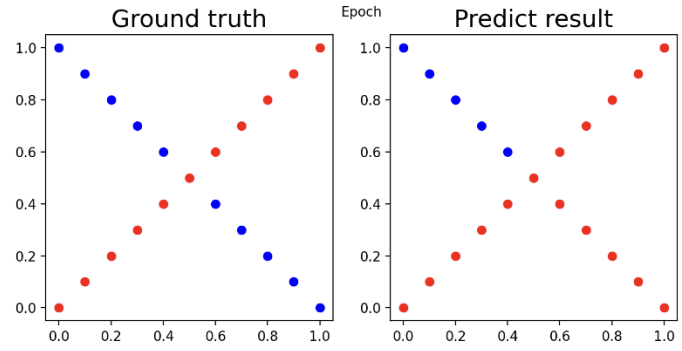
在調整 linear 的 hidden units 時，並不會有太大的變化。可是在調大時，與相同的 hidden units 的 XOR 相比，訓練時間會長不少。

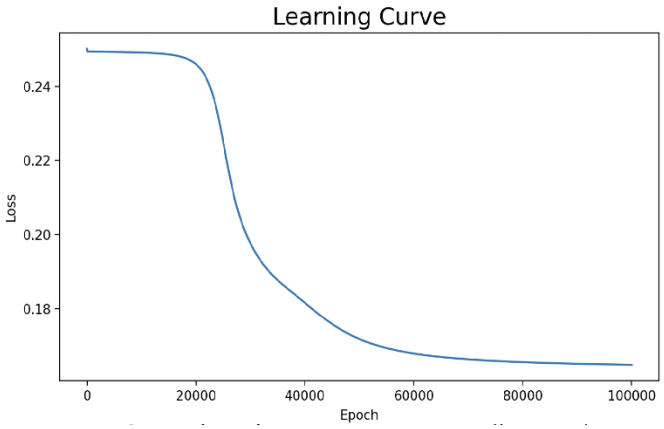


XOR with hidden units = 2, learning rate = 0.1:

XOR 在調低 hidden units 時會導致其 accuracy 變得相當差甚至低於 0.9

```
epoch 0 loss : 0.2501862258875396
epoch 10000 loss : 0.24918486601504122
epoch 20000 loss : 0.2460436695161326
epoch 30000 loss : 0.19762502819911365
epoch 40000 loss : 0.18156912558033228
epoch 50000 loss : 0.17168559480099835
epoch 60000 loss : 0.1677714133011635
epoch 70000 loss : 0.1662090972711316
epoch 80000 loss : 0.16544432160211175
epoch 90000 loss : 0.16500884401828325
Accuracy : [0.76190476]
```


Ground truth
Predict result
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Learning Curve
Loss
0.24
0.22
0.20
0.18
0
20000
40000
60000
80000
100000
Epoch