NYCU DL Lab1 – Backpropagation

313552049 鄭博涵

1. Introduction

在這次作業中，我使用 numpy 和一些 python standard 實作了一個簡單的兩層 hidden layers neural network，其中有使用不同的 activation function 例如: Sigmoid, Softmax, ReLU 以及嘗試不同的 learning rate, 並利用 Linear data 和 XOR data 去 train neural network.

2. Experiment setups

A.Sigmoid function:

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

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

定義 sigmoid 函式以及 derivative_sigmoid 函式

B. Neural network

```python
class TwoLayerNN:
    # initiate weight and bias
    def __init__(self, input_size, hidden_size1, hidden_size2, output_size):
        self.w1 = np.random.randn(input_size, hidden_size1) * np.sqrt(2. / input_size)  # Xavier 初始化
        self.b1 = np.zeros((1, hidden_size1))
        self.w2 = np.random.randn(hidden_size1, hidden_size2) * np.sqrt(2. / input_size)
        self.b2 = np.zeros((1, hidden_size2))
        self.w3 = np.random.randn(hidden_size2, output_size) * np.sqrt(2. / input_size)
        self.b3 = np.zeros((1, output_size))

```

```python
    def train(self, x, y, epochs):
        import matplotlib.pyplot as plt

        self.loss = []
        for epoch in range(epochs):
            self.y_pred = self.forward(x)
            self.backward(x, y)
            loss = MSE_Loss(y, self.y_pred)
            self.loss.append(loss)

            if epoch % 1000 == 0:
                print(f"Epoch: {epoch}, Loss: {loss}")

        # plot the learning curve
        plt.subplot(2, 1, 1)     # 在2列1行的圖形中的第一個位置繪製 (即上半部)
        plt.title("Learning Curve", fontsize = 18)
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.plot(self.loss)

    def show_result(self, x, y):
        accuracy = sum((self.y_pred > 0.5) == (y == 1)) / y.size    # 預測值 > 0.5 且 y為1 的情況數 再除以y.size
        print(f"Accruracy: {accuracy}")
        print("Prediction: ")

        for i in range(y.size):
            print(f"Iter{i+1} |  Ground truth: {y[i]}  |   Prediction: {self.y_pred[i]}")
        show_result(x, y, self.y_pred > 0.5)
```

建立 model 時可決定你要傳入的參數，例如: hidden layer 的 size，那透過

TwoLayerNN.train 可直接對 model 進行 train

並印出 Learning Curve (loss 的動態變化)

C. Back propagation

```python
def forward(self, x):
    self.z1 = np.dot(x, self.w1) + self.b1
    self.a1 = ReLU(self.z1)

    self.z2 = np.dot(self.a1, self.w2) + self.b2
    self.a2 = ReLU(self.z2)

    self.z3 = np.dot(self.a2, self.w3) + self.b3
    self.a3 = sigmoid(self.z3)
    return self.a3

def backward(self, x, y):
    loss_gradient = derivative_MSE_Loss(y, self.a3)
    d_z3 = loss_gradient * derivative_sigmoid(self.z3)

    d_w3 = np.dot(self.a2.T, d_z3)
    d_b3 = np.sum(d_z3, axis = 0, keepdims = True)    # keepdim = true: 確保d_b3的維度跟b3的一致

    d_a2 = np.dot(d_z3, self.w3.T)  # z3 = a2 * w3 + b3
    d_z2 = d_a2 * derivative_ReLU(self.z2)
    d_w2 = np.dot(self.a1.T, d_z2)
    d_b2 = np.sum(d_z2, axis = 0, keepdims = True)    # axis = 0: 將每列的 d_z2 進行求和

    d_a1 = np.matmul(d_z2, self.w2.T)
    d_z1 = d_a1 * derivative_ReLU(self.z1)
    d_w1 = np.matmul(x.T, d_z1)
    d_b1 = np.sum(d_z1, axis = 0, keepdims = True)

    # update weight and bias
    learning_rate = 1e-2
    self.w1 -= learning_rate * d_w1
    self.w2 -= learning_rate * d_w2
    self.w3 -= learning_rate * d_w3
    self.b1 -= learning_rate * d_b1
    self.b2 -= learning_rate * d_b2
    self.b3 -= learning_rate * d_b3
```

透過 forward 前向傳播後取得輸出 self.a3，並由 loss function 算出

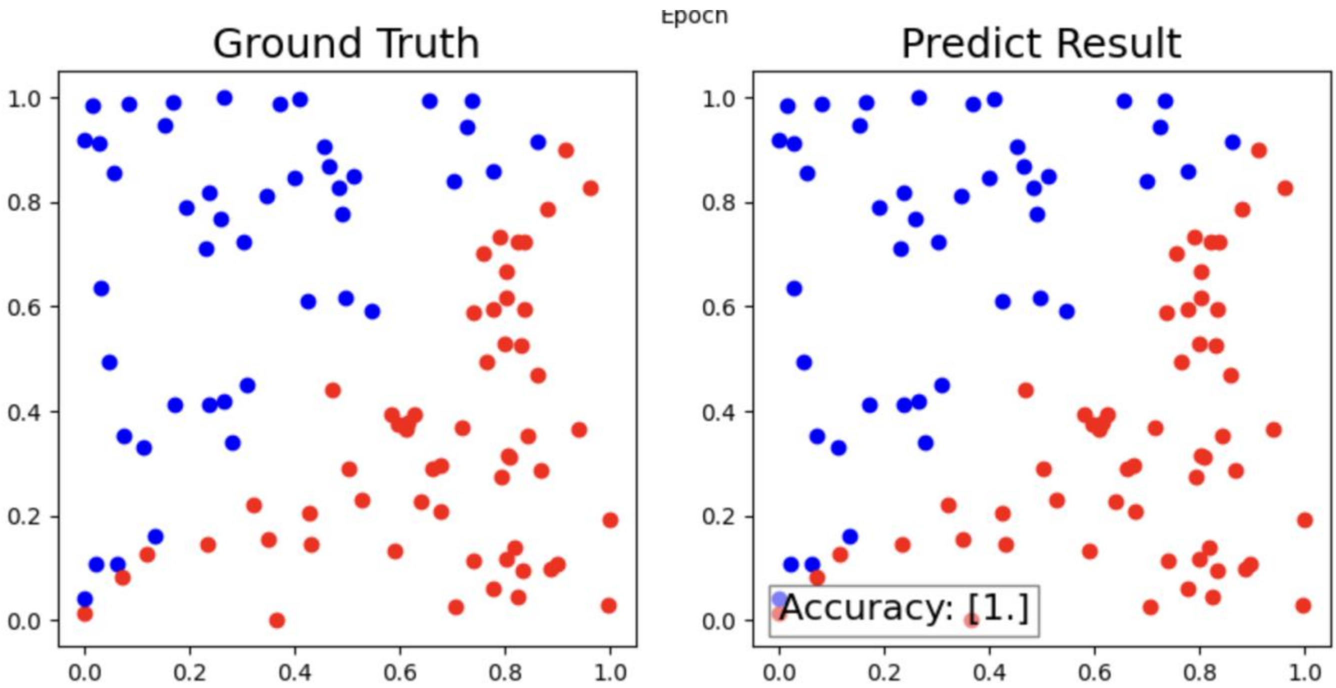
loss_gradient 後 用 backpropagation 來算出 a,z,w,b 的導數

最後更新權重以及偏差

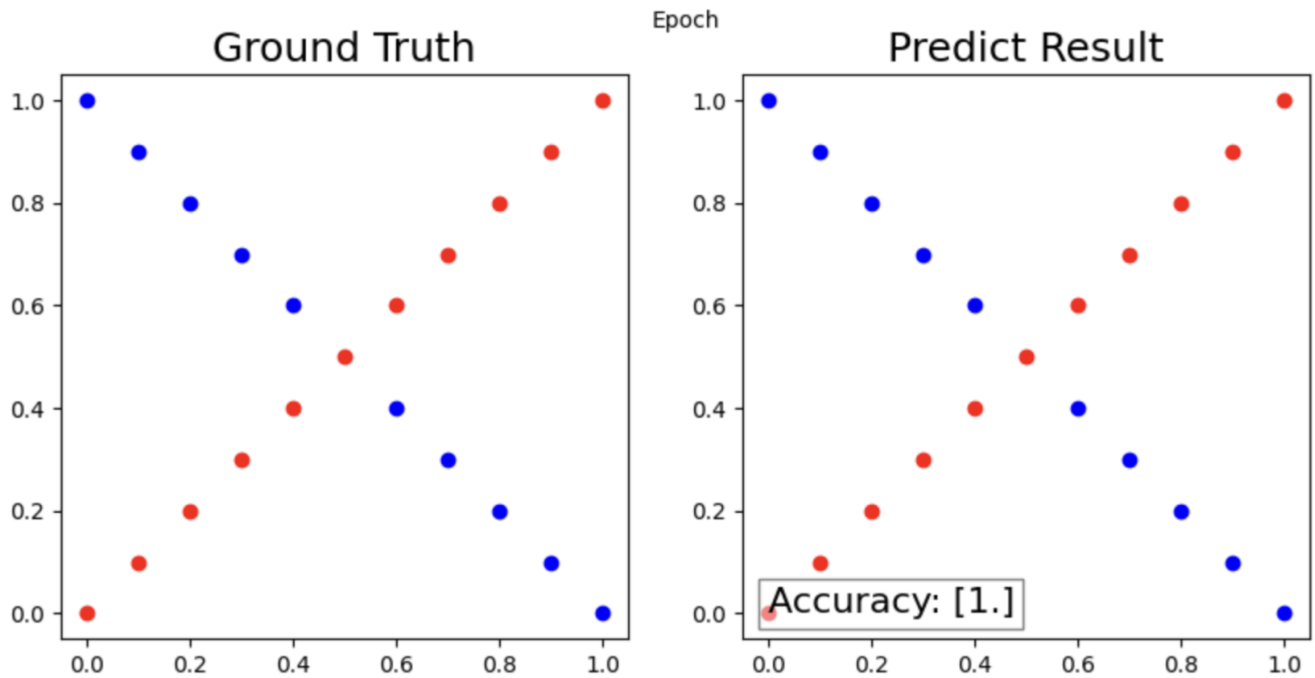
3. Result of your testing

A. Screenshot and comparison figure

Linear:



XOR:



B. Show the accuracy of your prediction

Lr = 0.01, Hidden size = 4, Optimizer = SGD, 激活函數: ReLU, sigmoid

由於我全部用單一的 Activate function，執行結果會怪怪的,

所以後來是在 hidden layer 使用 ReLU，在 output layer 使用 sigmoid

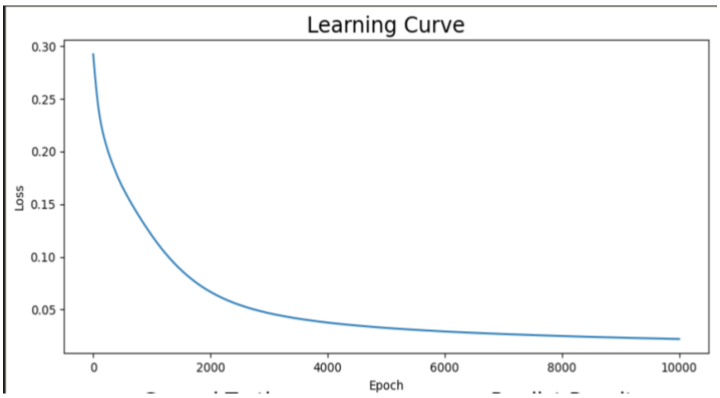
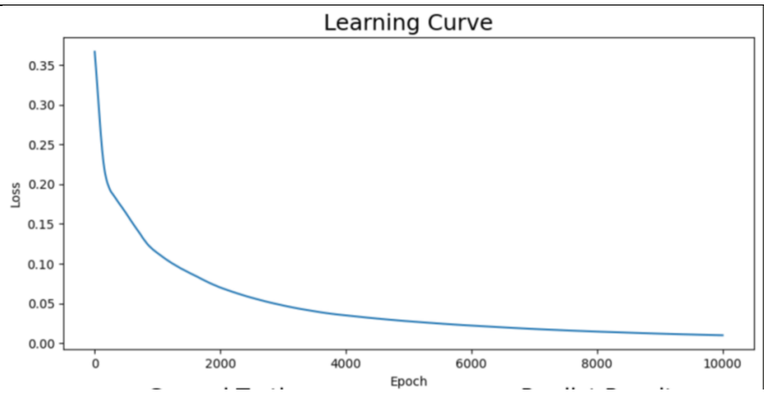
Linear:

```
Accruracy: [1.]
Prediction:
Iter1 |  Ground truth: [0]  |   Prediction: [0.47135232]
Iter2 |  Ground truth: [1]  |   Prediction: [0.99547637]
Iter3 |  Ground truth: [0]  |   Prediction: [0.02303764]
Iter4 |  Ground truth: [0]  |   Prediction: [0.03630578]
Iter5 |  Ground truth: [1]  |   Prediction: [0.77143755]
Iter6 |  Ground truth: [1]  |   Prediction: [0.99088147]
Iter7 |  Ground truth: [0]  |   Prediction: [0.02620623]
Iter8 |  Ground truth: [0]  |   Prediction: [0.03015457]
Iter9 |  Ground truth: [1]  |   Prediction: [0.99614696]
Iter10 |  Ground truth: [0]  |   Prediction: [0.01321711]
Iter11 |  Ground truth: [0]  |   Prediction: [0.14512579]
Iter12 |  Ground truth: [1]  |   Prediction: [0.99615758]
Iter13 |  Ground truth: [1]  |   Prediction: [0.99856409]
...
Iter97 |  Ground truth: [1]  |   Prediction: [0.97234584]
Iter98 |  Ground truth: [1]  |   Prediction: [0.99463235]
Iter99 |  Ground truth: [1]  |   Prediction: [0.9958064]
Iter100 |  Ground truth: [1]  |   Prediction: [0.9969974]
```

XOR:

```
Accruracy: [1.]
Prediction:
Iter1 |  Ground truth: [0]  |   Prediction: [0.0947721]
Iter2 |  Ground truth: [1]  |   Prediction: [0.99992109]
Iter3 |  Ground truth: [0]  |   Prediction: [0.0947765]
Iter4 |  Ground truth: [1]  |   Prediction: [0.99947312]
Iter5 |  Ground truth: [0]  |   Prediction: [0.0947809]
Iter6 |  Ground truth: [1]  |   Prediction: [0.9964911]
Iter7 |  Ground truth: [0]  |   Prediction: [0.0947853]
Iter8 |  Ground truth: [1]  |   Prediction: [0.97658815]
Iter9 |  Ground truth: [0]  |   Prediction: [0.09478969]
Iter10 |  Ground truth: [1]  |   Prediction: [0.73397418]
Iter11 |  Ground truth: [0]  |   Prediction: [0.09479409]
Iter12 |  Ground truth: [0]  |   Prediction: [0.09479849]
Iter13 |  Ground truth: [1]  |   Prediction: [0.80534657]
...
Iter18 |  Ground truth: [0]  |   Prediction: [0.09481169]
Iter19 |  Ground truth: [1]  |   Prediction: [0.99997741]
Iter20 |  Ground truth: [0]  |   Prediction: [0.09481609]
Iter21 |  Ground truth: [1]  |   Prediction: [0.99999863]
```

C. Learning curve

Linear	XOR
	

Epoch: 0, Loss: 0.292372327365041 Epoch: 1000, Loss: 0.12023942988886194 Epoch: 2000, Loss: 0.06675965377114214 Epoch: 3000, Loss: 0.046509592400175014 Epoch: 4000, Loss: 0.03757912637935705 Epoch: 5000, Loss: 0.03251227283050282 Epoch: 6000, Loss: 0.029151249184145977 Epoch: 7000, Loss: 0.02670769307814082 Epoch: 8000, Loss: 0.02481012258250511 Epoch: 9000, Loss: 0.02326970783601087	Epoch: 0, Loss: 0.3666773863104507 Epoch: 1000, Loss: 0.11330392707028356 Epoch: 2000, Loss: 0.07001066614001625 Epoch: 3000, Loss: 0.047421633807543086 Epoch: 4000, Loss: 0.0348658803343669 Epoch: 5000, Loss: 0.027530500294848444 Epoch: 6000, Loss: 0.02204676961348613 Epoch: 7000, Loss: 0.017832388979784213 Epoch: 8000, Loss: 0.014521119697131622 Epoch: 9000, Loss: 0.011943841602291753

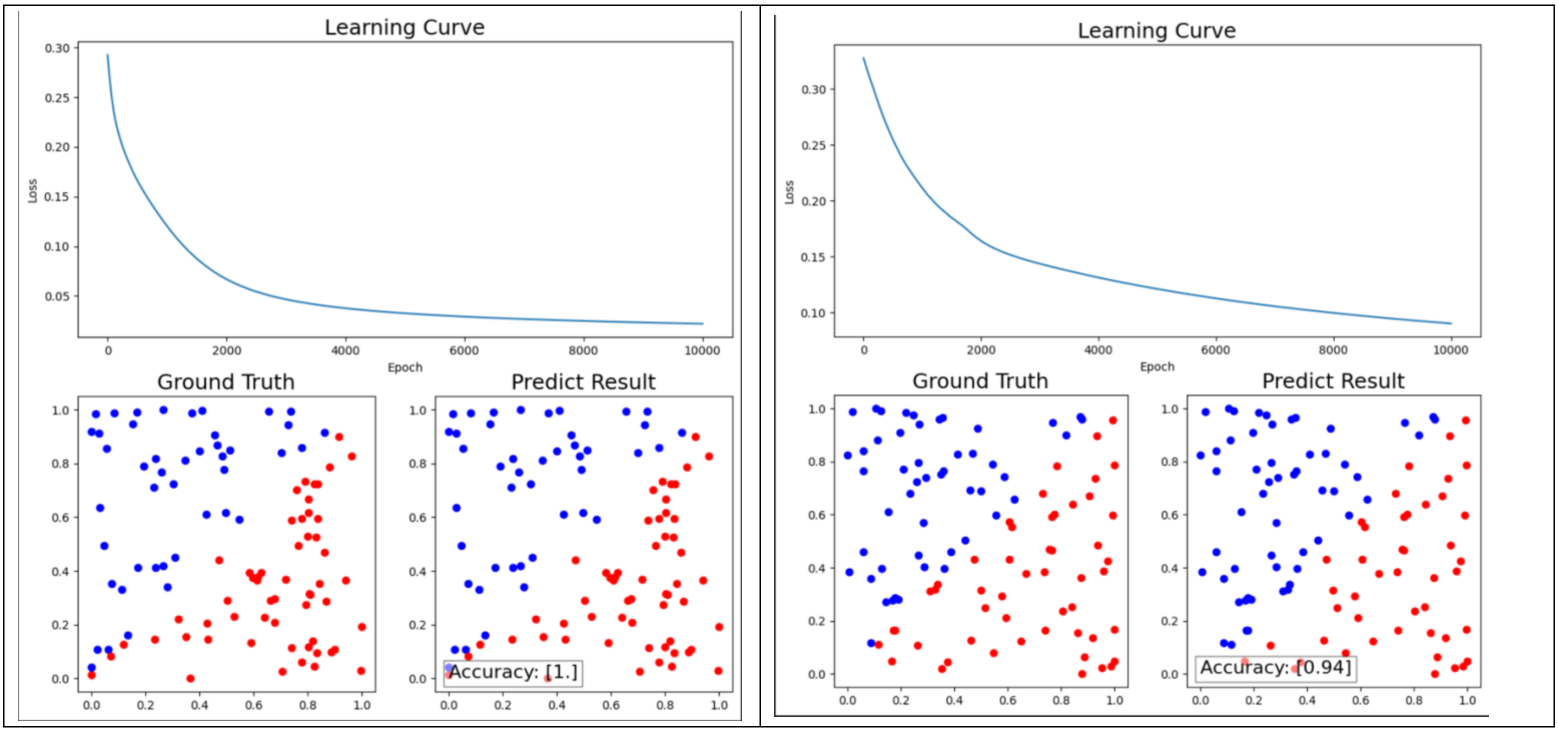
4. Discussion

A. Try different learning rates

Hidden unit = 4, Optimizer = SGD, Activate function: ReLU, Sigmoid

Linear:

Lr = 0.01	Lr = 0.001



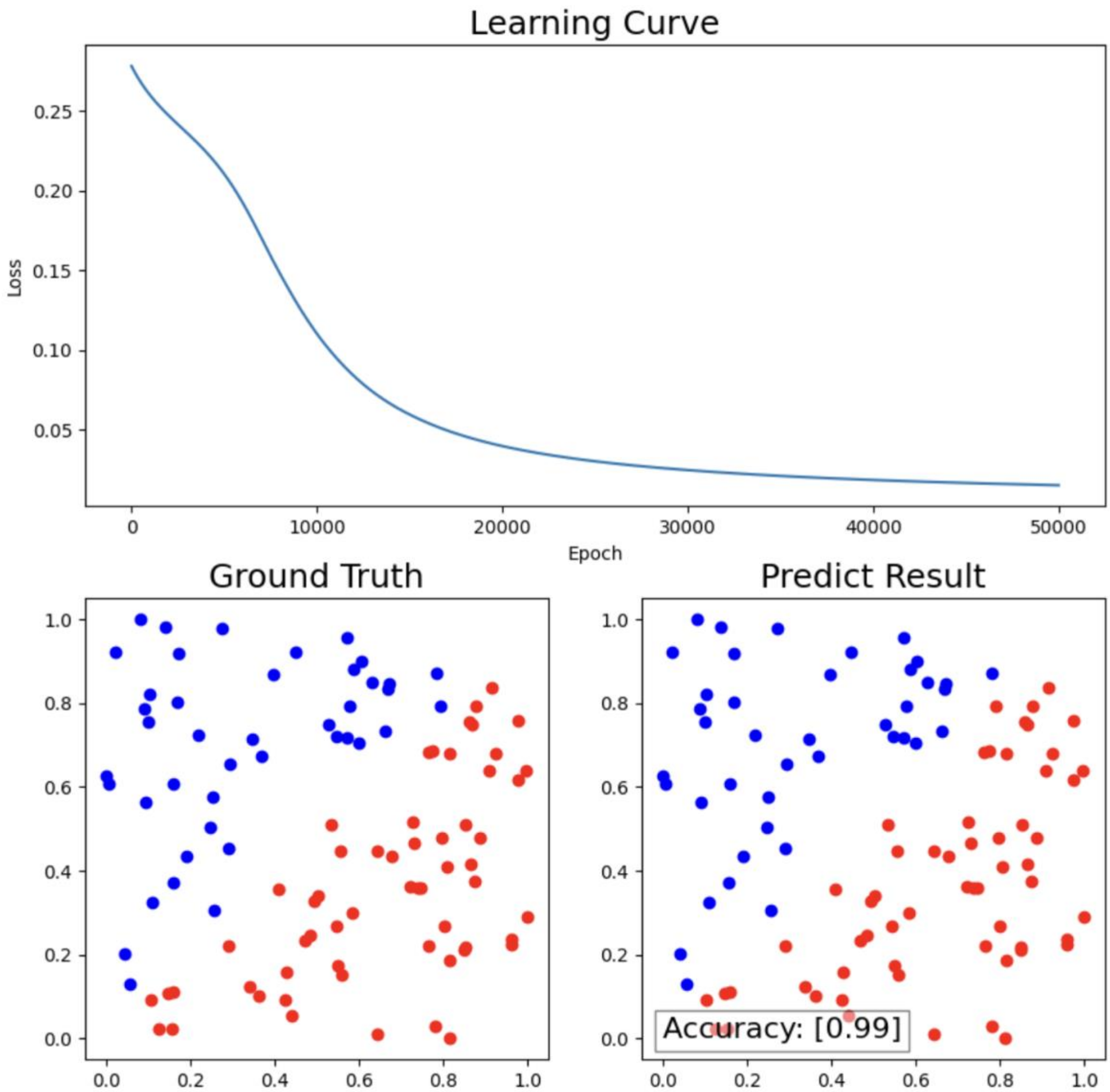
透過降低 learning rate 我們可以看出 loss function 下降的速度變慢了

所以 Accuracy 也隨之降低，而在 learning rate 過低的情況下，我們也可以

透過提升 epoch 數的方式來增加我們的 Accuracy

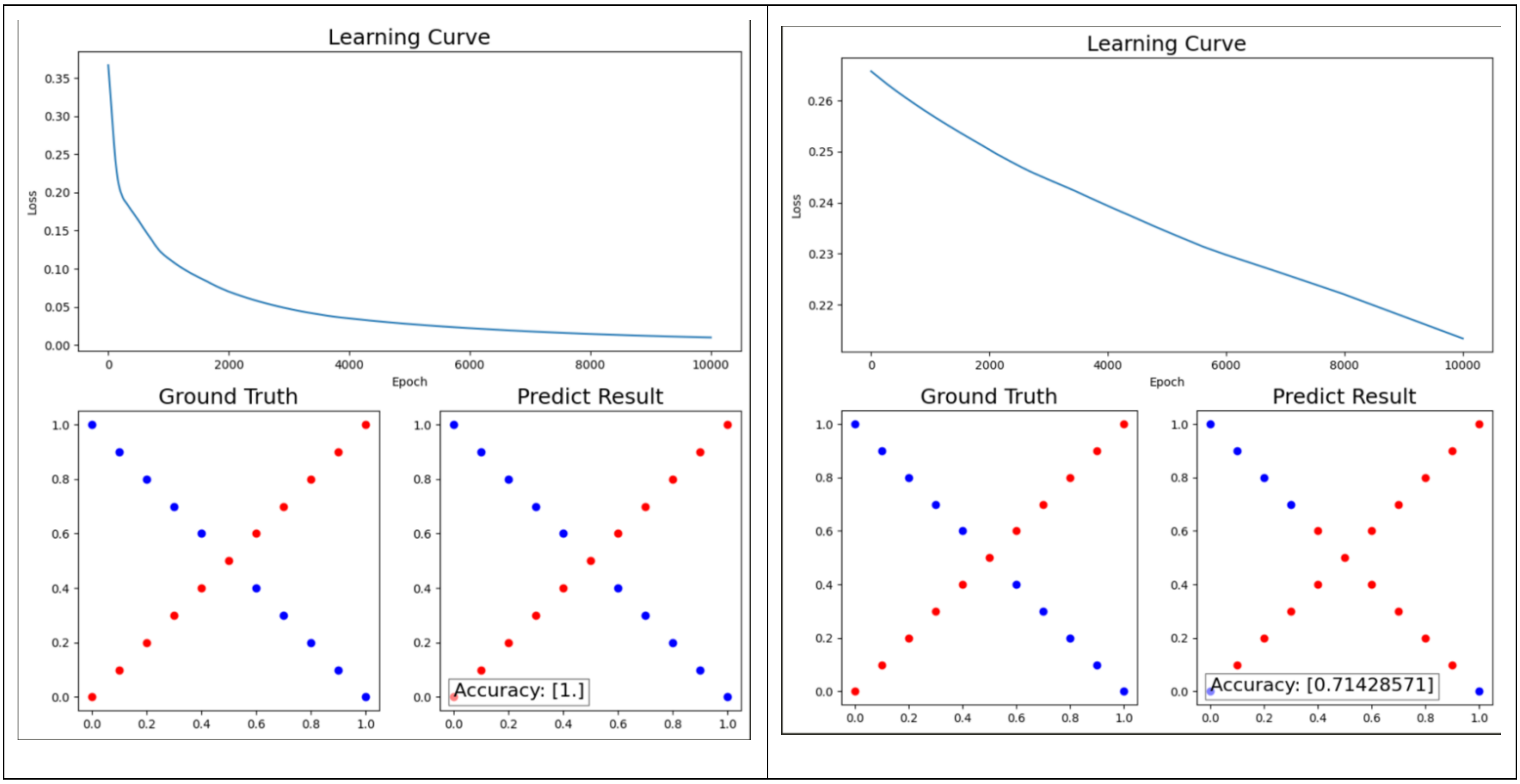
例如以下: (try different number of epoch)

lr = 0.001, epoch = 50000



XOR:

Lr = 0.01	Lr = 0.001

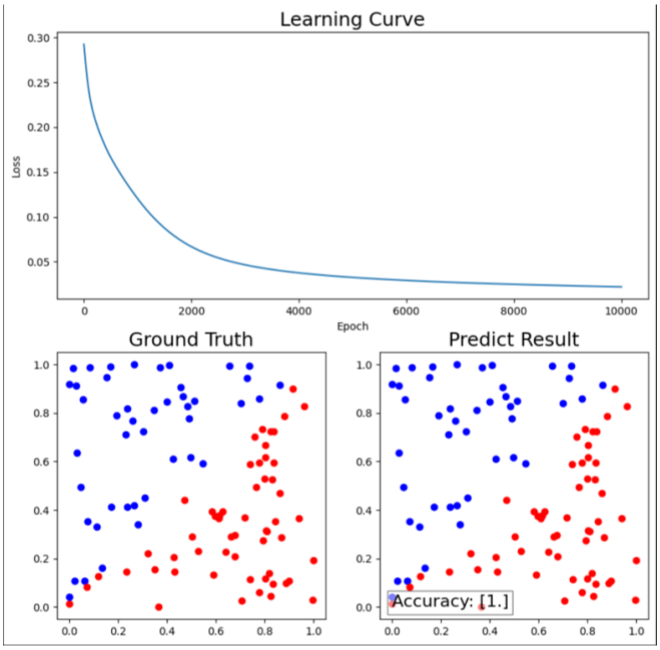
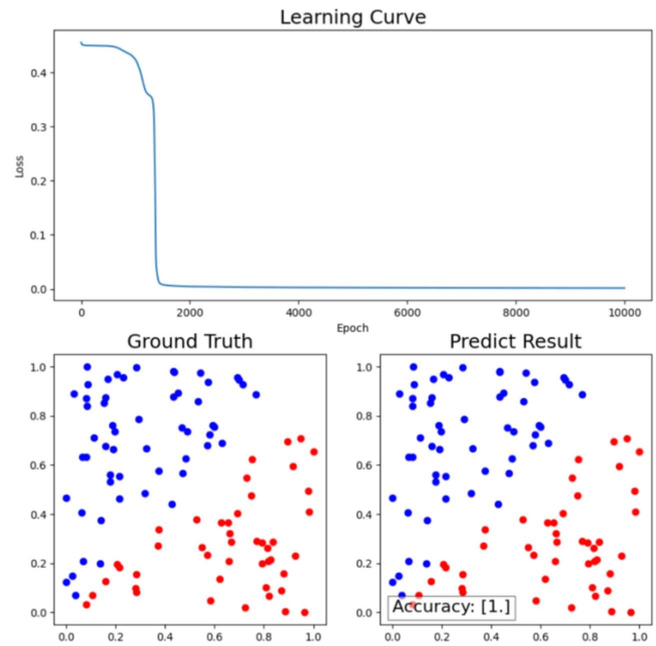


如同 linear, loss function 會緩慢降低，所以 lr = 0.01 是個比較適合這個 model 的 learning rate

B. Try different numbers of hidden units

Lr = 0.01, Optimizer = SGD, Activate function: sigmoid, ReLU

Linear:

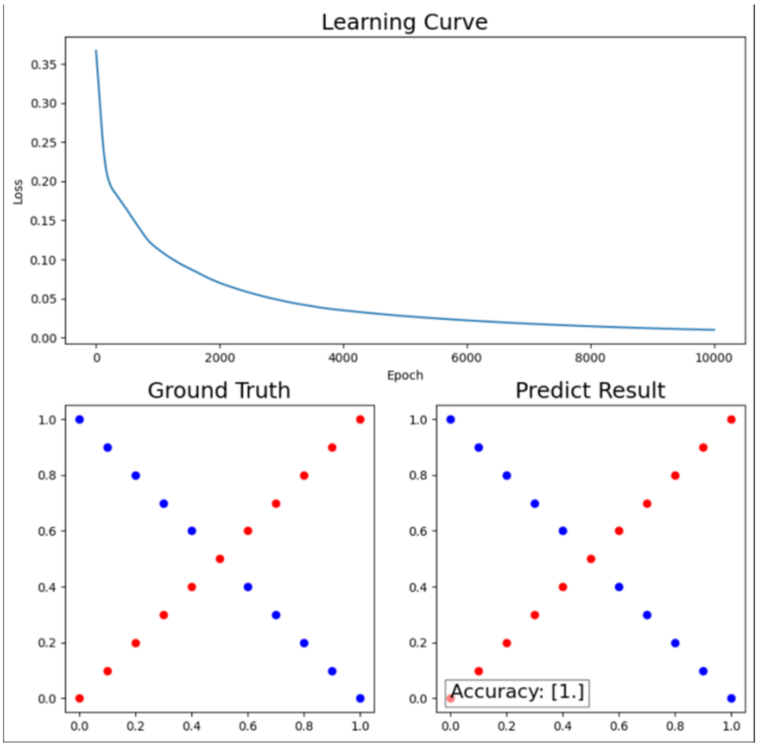
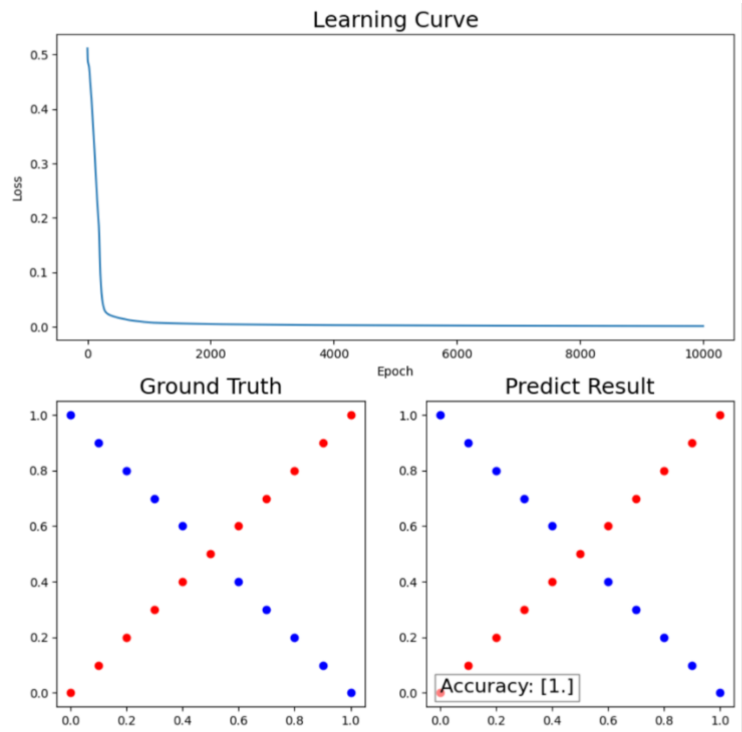
<table>
<tr><td>Hidden_unit = 4</td><td>Hidden_unit = 50</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

將 Hidden_unit 提高到 50 時，可以看到 loss 的變化是一開始較平緩

等到 epoch 訓練大概 1600 次左右時 loss 會驟降，之後趨近於 0

而 accuracy 則是沒有差別，皆為 1

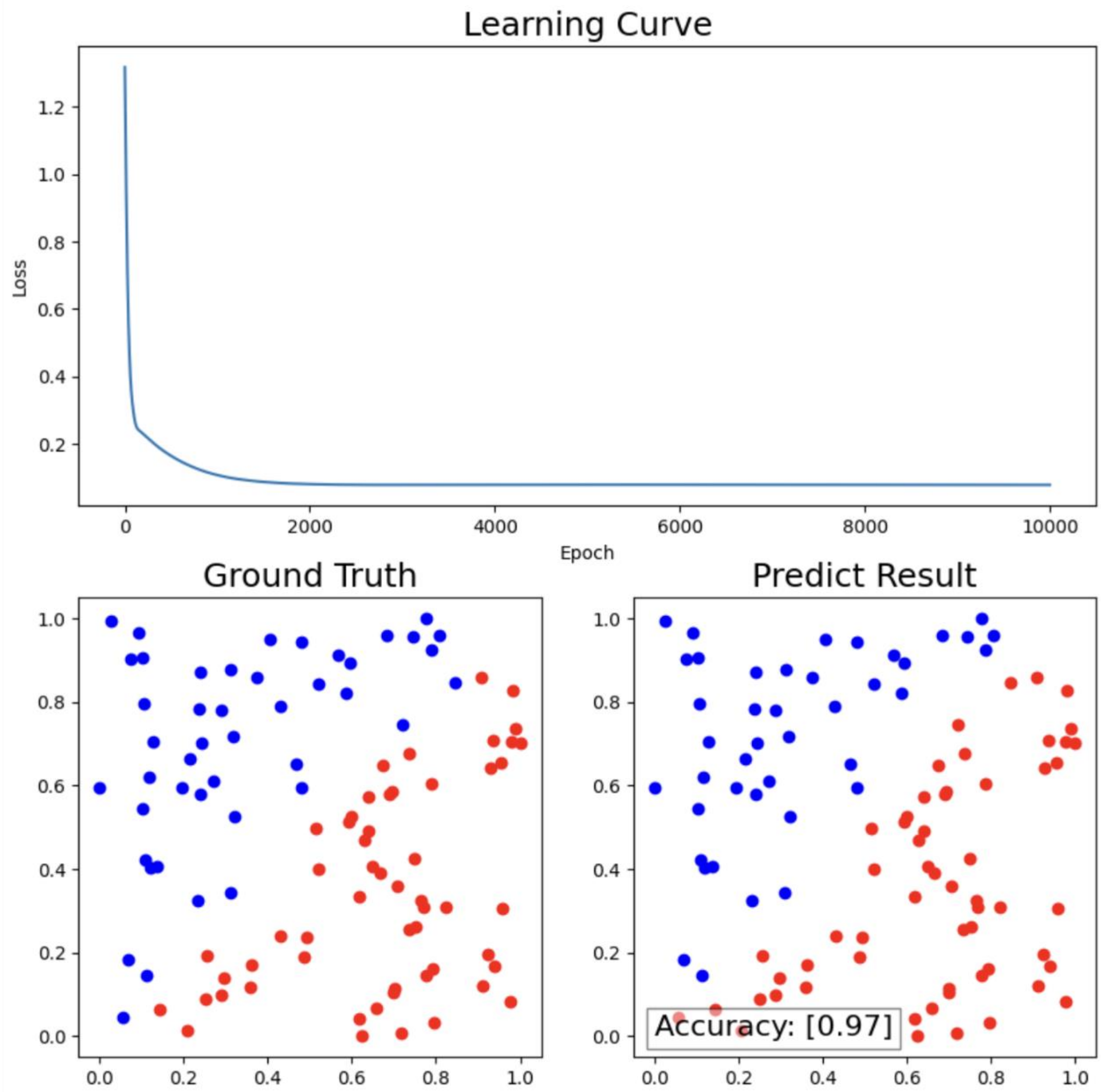
XOR:

Hidden_unit = 4	Hidden_unit = 50
	

在 XOR 上增加 hidden_unit 的話 除了維持表現之外，loss 也下降得更為快速

B. Try without activation functions

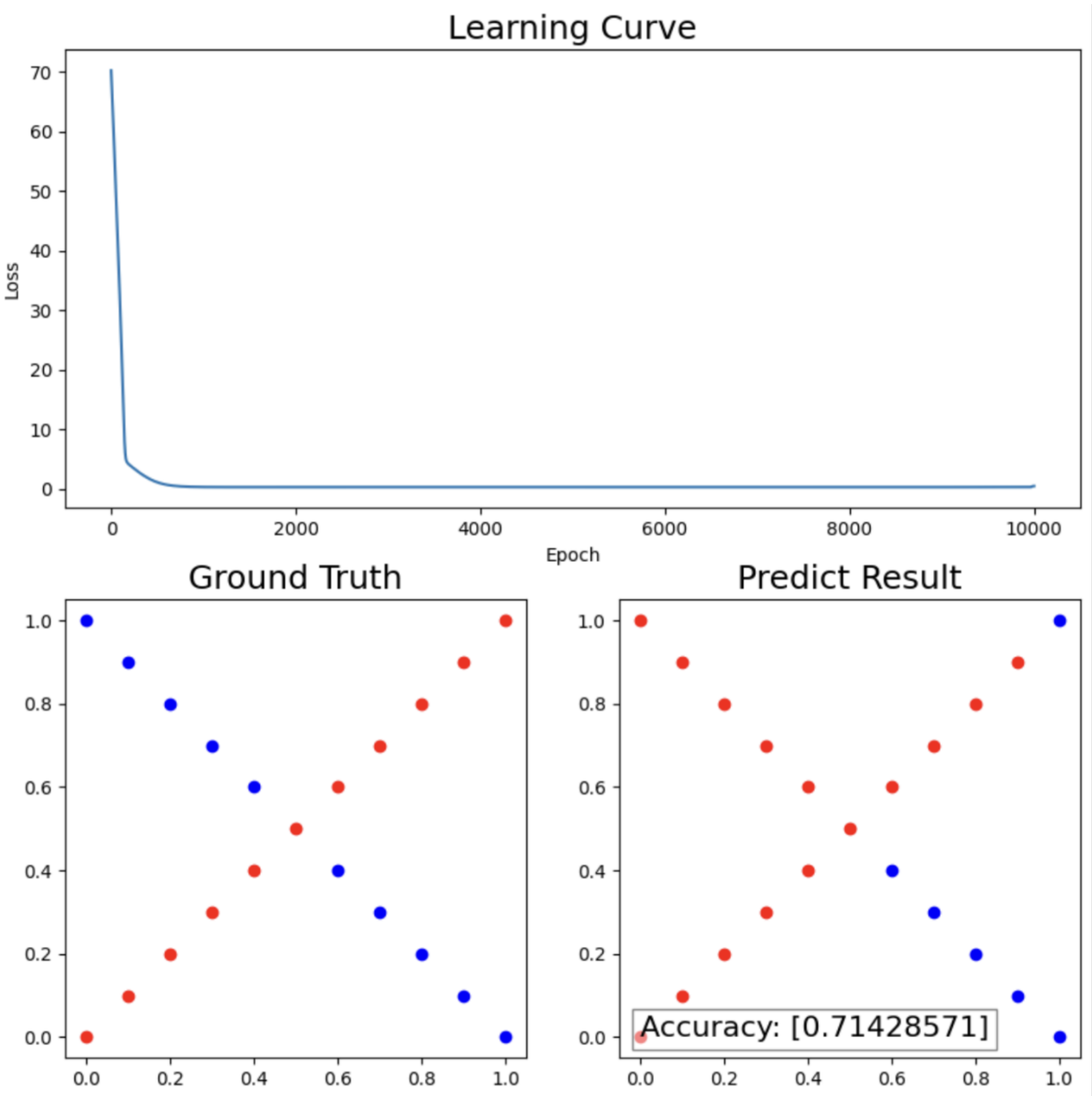
Linear:



Linear data 有無 activate function 似乎表現上差不多

但 Accuracy 會略微下降

XOR:



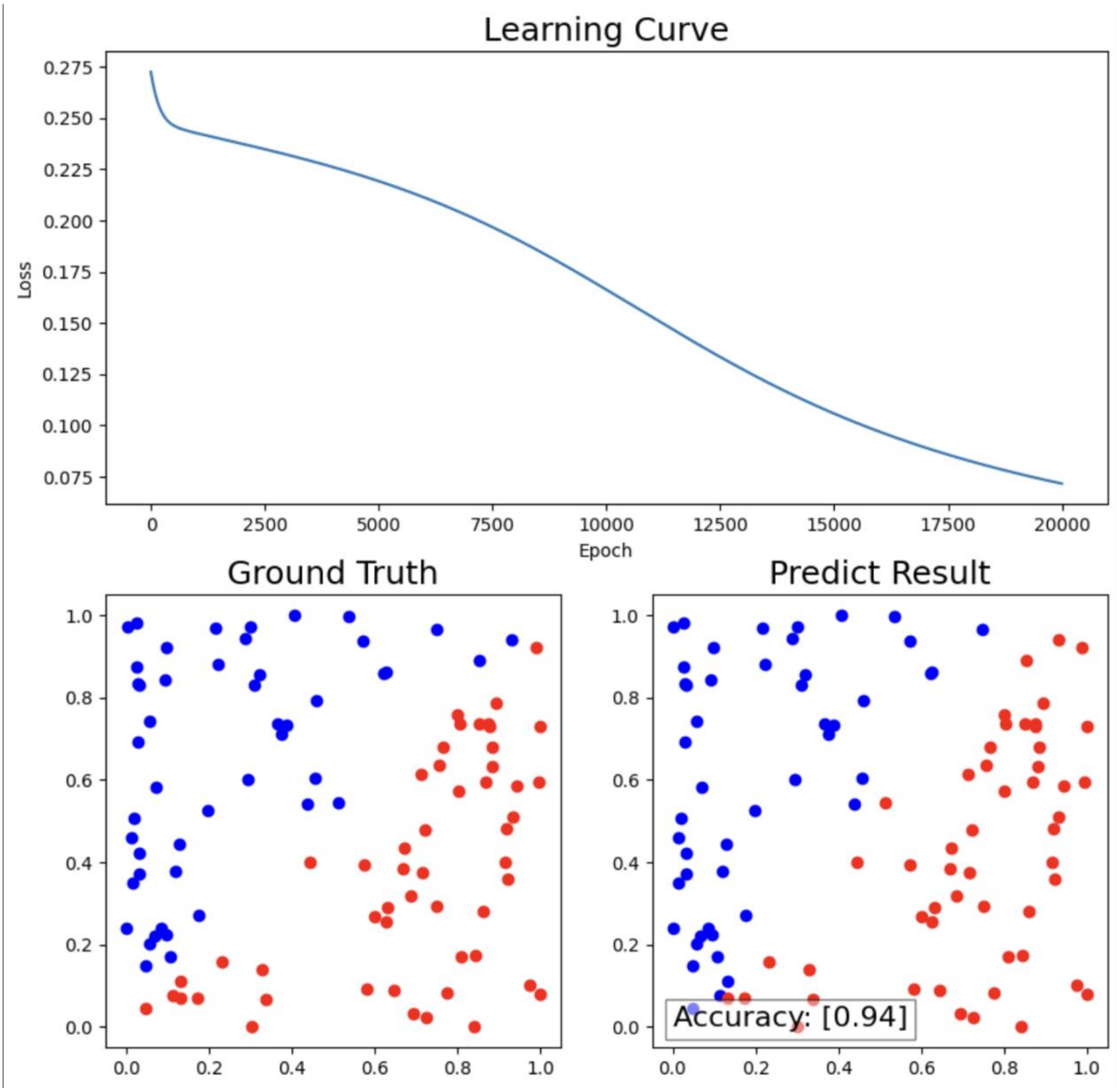
而在 XOR data 中，沒有了 activation function，表現則是會大幅下降, Accuracy 在 0.7 多

5.Extra

B. Implement different activation functions

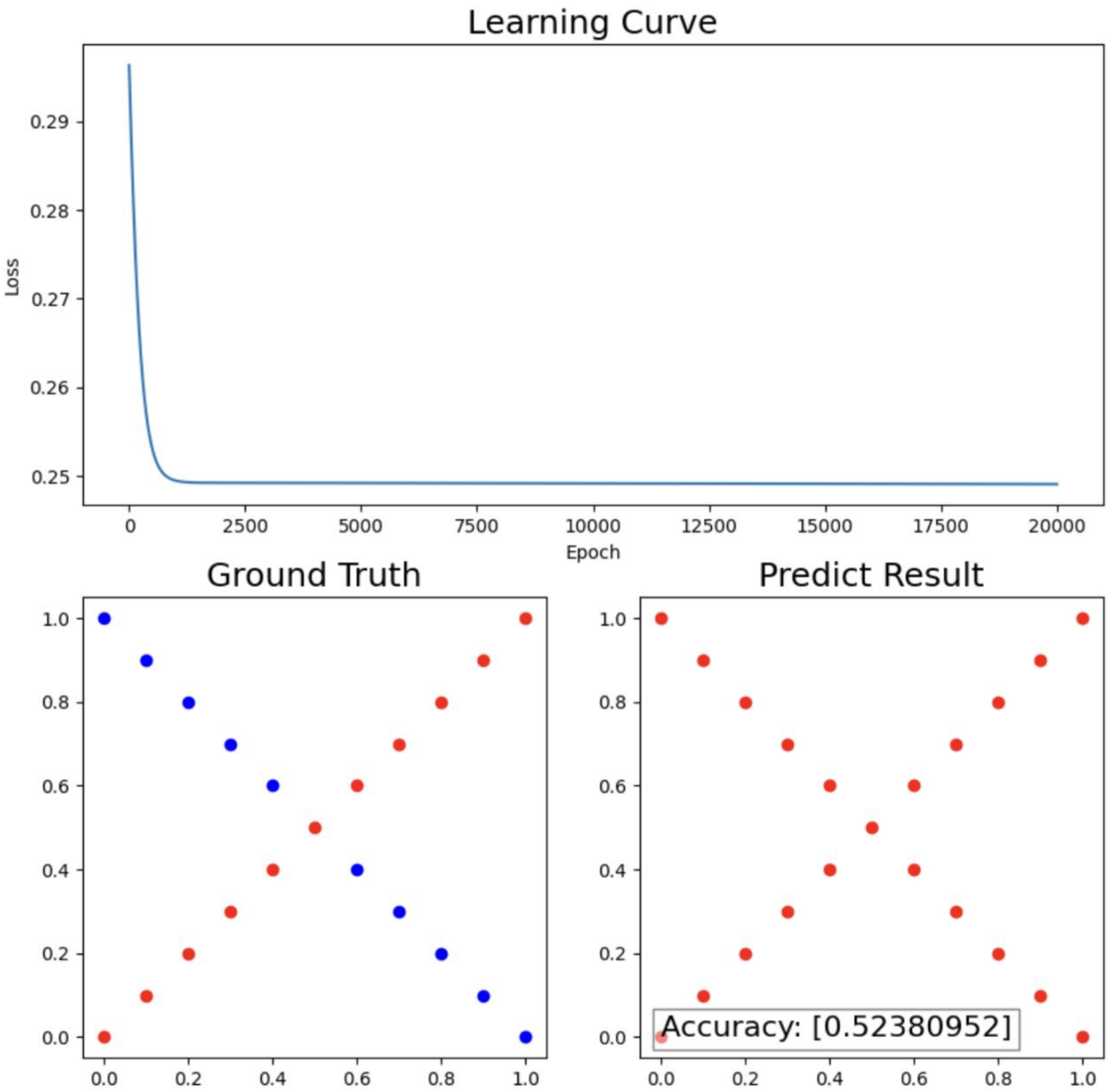
全部都用 sigmoid:

Linear:



全部都用 sigmoid 的話可看出表現會略微下降，而且有時會發生異常，例如 loss 完全沒有降低的情況

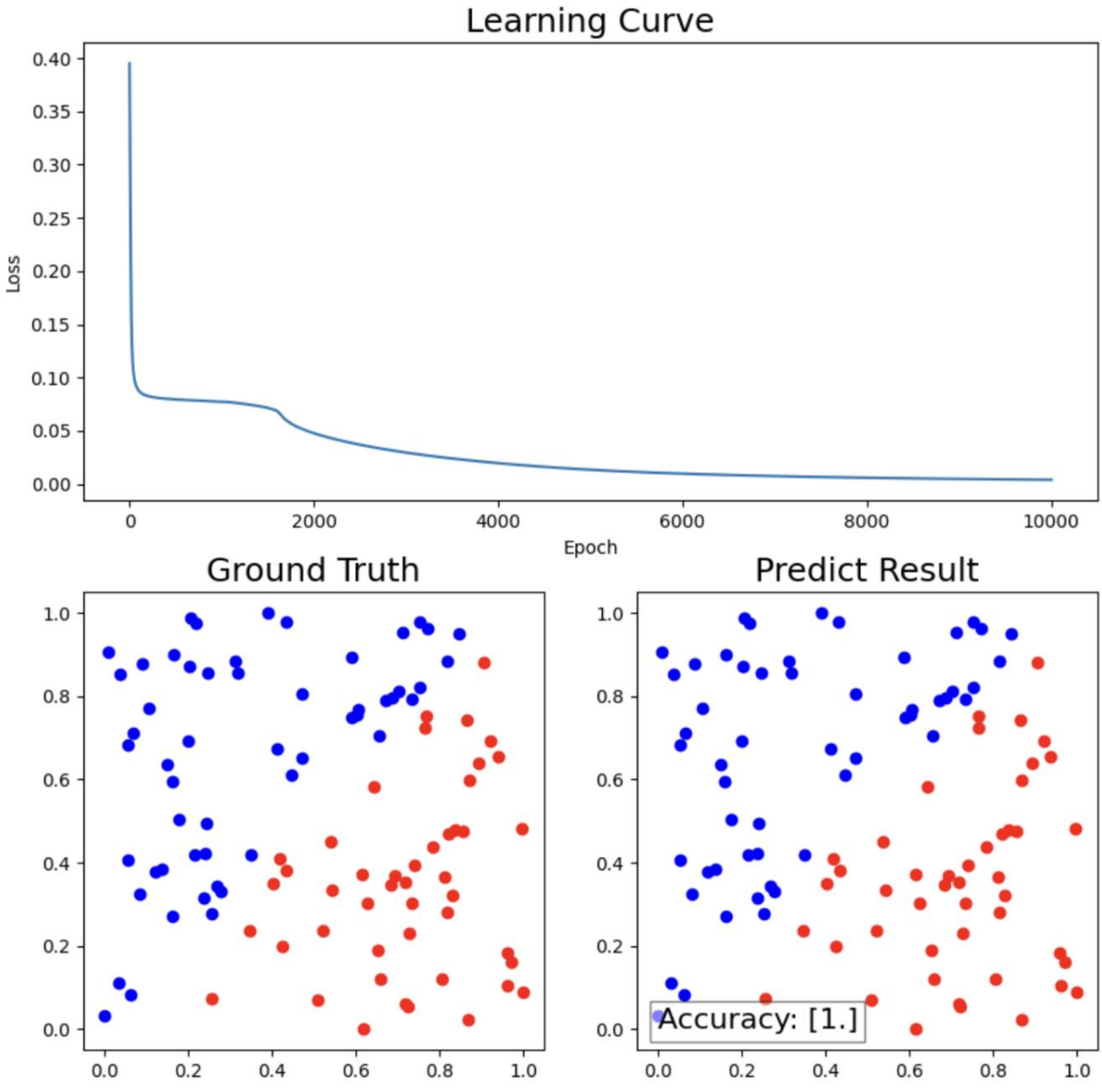
XOR:



XOR 也是時好時壞的情況

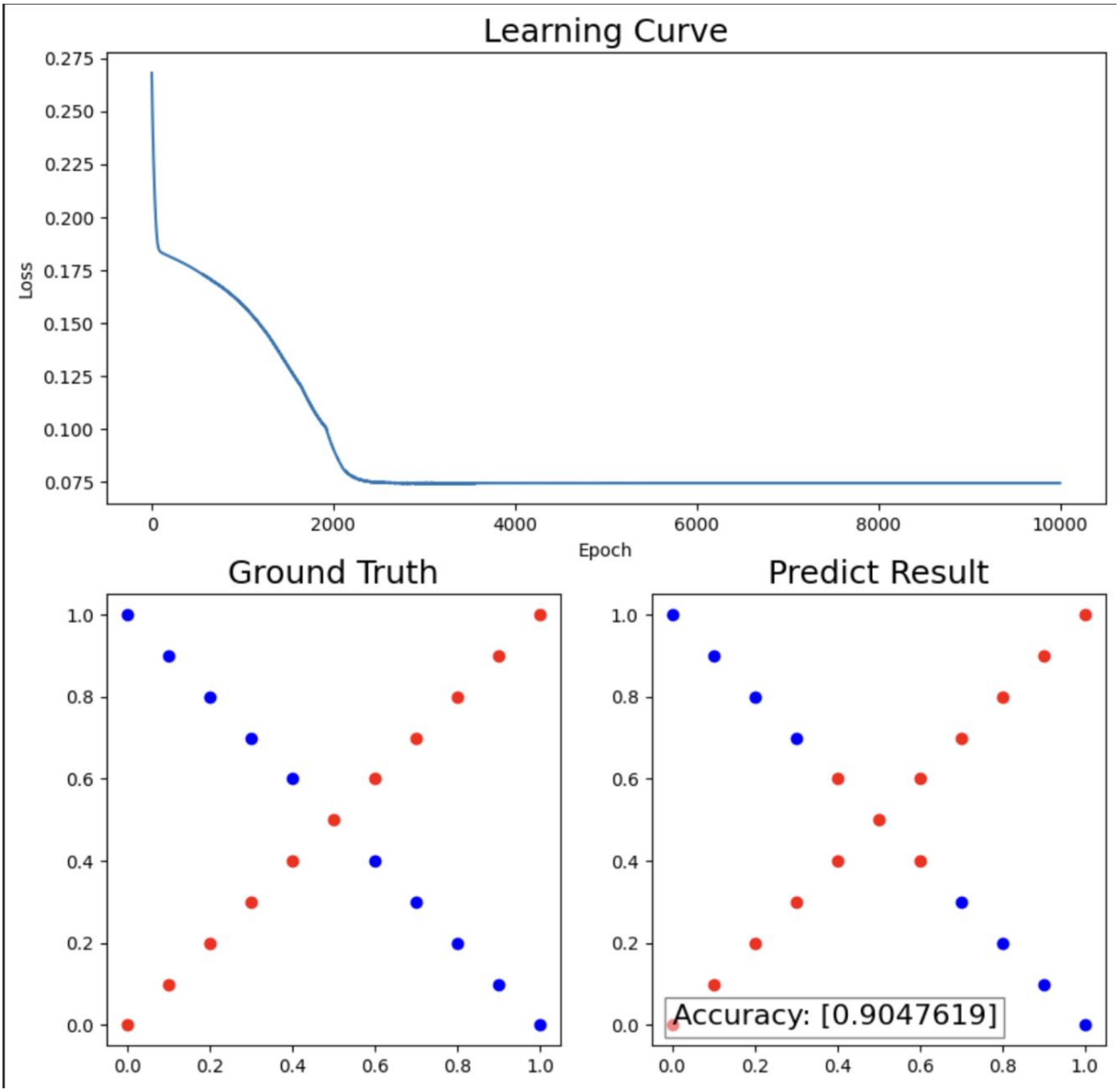
全部都用 ReLU:

Linear:



跟全部用 sigmoid 的情況差不多, loss 的減少會時好時壞

XOR:


Learning Curve
Loss
Epoch
Ground Truth
Predict Result
Accuracy: [0.9047619]