Lab 1 Report - Back-Propagation

- Student Info
  1. Student ID: 310555024
  2. Student Name: 林廷翰

- Introduction (20%)

  此次實驗內容為基於Numpy實做一個具有兩層hidden layer的neural networks，透過forward pass來進行結果預測，再透過back propagation來修正weight，目的是生成一組model weight，使其對應的預測符合產生data point時的labels。

  在實做方面，由於希望能夠更彈性的調整hidden layer及每一層layer的hidden neural數目，因此我建立了兩個class，`class Layer` 及 `class NNetwork` ，且參數可透過argument的方式傳入，方便執行。

- Experiment Setups (30%)
  1. Sigmoid functions
     - `sigmoid()`

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

       此function實做於 `class Layer` 中，並會在forward pass時被呼叫。

     - `derivative_sigmoid()`

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

       此function實做於 `class Layer` 中，並會在backward pass時被呼叫。

2. Neural Network

- `class Layer`

1. `init_weight()`

```python
def init_weight(self):
    return np.random.uniform(0, 1, (self.num_of_neurals, self.num_of_next_layer_neurals))
```

初始化layer同時會根據`num_of_neurals`及`num_of_next_layer_neurals`來隨機產生initial weight。

2. `forward()`

```python
def forward(self, inputs):
    self.forward_gradient = inputs

    if self.activation_function == 'sigmoid':
        self.forward_output = self.sigmoid(np.dot(inputs, self.weight))
    elif self.activation_function == 'relu':
        self.forward_output = self.relu(np.dot(inputs, self.weight))
    elif self.activation_function == 'none':
        self.forward_output = np.dot(inputs, self.weight)

    return self.forward_output
```

根據不同的activation function，基於input及weight計算`forward_gradient`。

3. `backward()`

```python
def backward(self, derivative_loss):
    # Compute ∂C/∂Z'
    if self.activation_function == 'sigmoid':
        self.backward_gradient = derivative_loss * self.derivative_sigmoid(self.forward_output)
    elif self.activation_function == 'relu':
        self.backward_gradient = derivative_loss * self.derivative_relu(self.forward_output)
    elif self.activation_function == 'none':
        self.backward_gradient = derivative_loss

    # return w5*(∂C/∂Za) + w6*(∂X/∂Zb) + ...
    return np.dot(self.backward_gradient, self.weight.T)
```

根據不同的activation function，基於`derivative_loss`，`forward_output`及其對應的`derivative function`，計算`backward_gradient`，並回傳`backward_gradient`及`weight`的乘積，目的是給上一層執行backward時作為input。

4. `update()`

```python
def update(self):
    gradient = np.dot(self.forward_gradient.T, self.backward_gradient)

    # Update weight.
    self.weight -= self.learning_rate * gradient
```

基於`forward_gradient` 及`backward_gradient` 計算gradient，並根據learning rate來更新`weight` 。

- `class NNetwork`

1. `init_layers()`

```python
def init_layers(self):
    # Init input layers.
    layers = [Layer(self.num_of_input_neurals, self.num_of_hidden_neurals, self.activation_function,
                    self.learning_rate)]

    # Init hidden layers.
    for _ in range(self.num_of_hidden_layers - 1):
        layers.append(
            Layer(self.num_of_hidden_neurals, self.num_of_hidden_neurals, self.activation_function,
                  self.learning_rate))

    # Init output layers.
    layers.append(Layer(self.num_of_hidden_neurals, self.num_of_output_neurals, 'sigmoid',
                        self.learning_rate))

    return layers
```

基於輸入參數分別建立input layer, hidden layers, 及output layers，並將所有layers存於`layers` 變數。

2. `train()`

```python
def train(self):
    losses = []

    for i in range(self.epoch):
        self.predict = self.forward()
        loss = self.MSE(self.compute_error(self.predict, self.labels))
        derivative_MSE = self.derivative_MSE(self.compute_error(self.predict, self.labels))
        self.backward(derivative_MSE)
        self.update_weight()

        print(f'epoch: {i + 1} loss: {loss}')
        self.losses.append(loss)
        if self.verify_prediction():
            break
```

此function為train model weights的主要function，每一個epoch都會對所有layers執行`forward()` , `backward()` 及`update_weight()` ，當超過maximum epochs或是accuracy為1.0時，會跳出迴圈，停止training。

3. `show_result()`

```python
def show_result(self):
    self.print_prediction()
    self.print_statistics()
    self.draw_result(self.inputs, self.labels, self.predict, self.losses)
```

輸出result，包含每一筆data的prediction (`print_prediction()`), 統計資料(`print_statistics()`)及對ground truth, prediction, learning curve畫圖(`draw_result()`)。

3. Backpropagation

- `class Layer`

1. `backward()`

```python
def backward(self, derivative_loss):
    # Compute ∂C/∂Z'
    if self.activation_function == 'sigmoid':
        self.backward_gradient = derivative_loss * self.derivative_sigmoid(self.forward_output)
    elif self.activation_function == 'relu':
        self.backward_gradient = derivative_loss * self.derivative_relu(self.forward_output)
    elif self.activation_function == 'none':
        self.backward_gradient = derivative_loss

    # return w5*(∂C/∂Za) + w6*(∂X/∂Zb) + ...
    return np.dot(self.backward_gradient, self.weight.T)
```

根據不同的activation function，基於`derivative_loss`，`forward_output`及其對應的`derivative function`，計算`backward_gradient`，並回傳`backward_gradient`及`weight`的乘積，目的是給上一層執行backward時作為input。

2. `update()`

```python
def update(self):
    gradient = np.dot(self.forward_gradient.T, self.backward_gradient)

    # Update weight.
    self.weight -= self.learning_rate * gradient
```

基於`forward_gradient`及`backward_gradient`計算gradient，並根據learning rate來更新`weight`。

- `class NNetwork`

1. `train()`

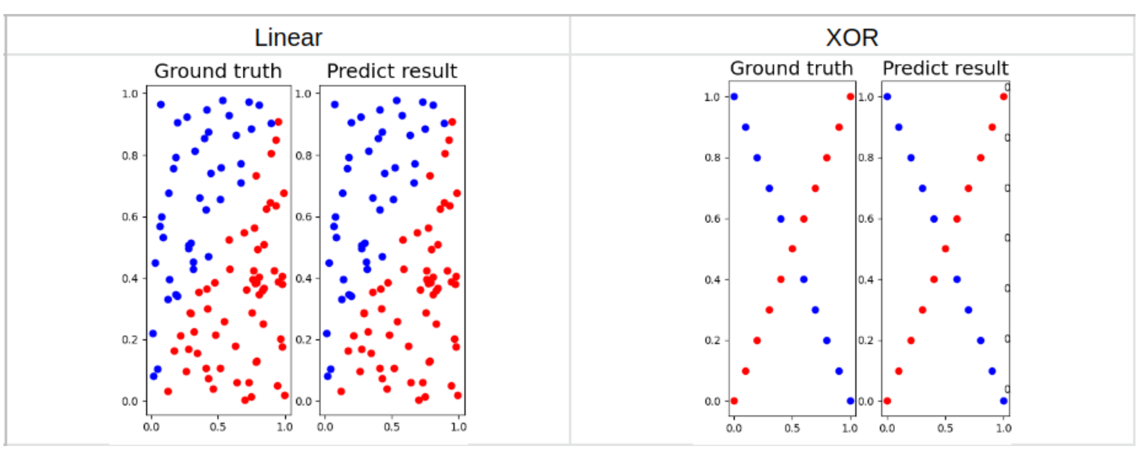
```python
    def train(self):
        losses = []

        for i in range(self.epoch):
            self.predict = self.forward()
            loss = self.MSE(self.compute_error(self.predict, self.labels))
            derivative_MSE = self.derivative_MSE(self.compute_error(self.predict, self.labels))
            self.backward(derivative_MSE)
            self.update_weight()

            print(f'epoch: {i + 1} loss: {loss}')
            self.losses.append(loss)
            if self.verify_prediction():
                break
```

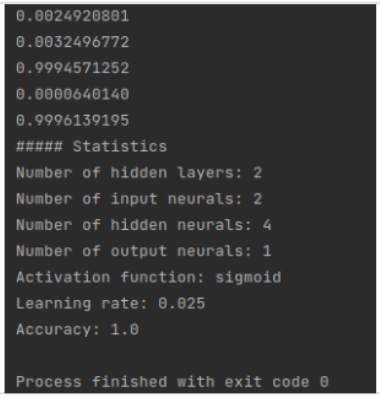
此function為train model weights的主要function，每一個epoch都會對所有layers執行`forward()`，`backward()`及`update_weight()`，當超過maximum epochs或是accuracy為1.0時，會跳出迴圈，停止training。

- Results of your Testing (20%)
  1. Input Parameters
     - Number of hidden layers: 2
     - Number of input neurals: 2
     - Number of hidden neurals: 4
     - Number of output neurals: 1
     - Activation function: sigmoid
     - Learning rate: 0.025
  2. Screenshot and comparison figure



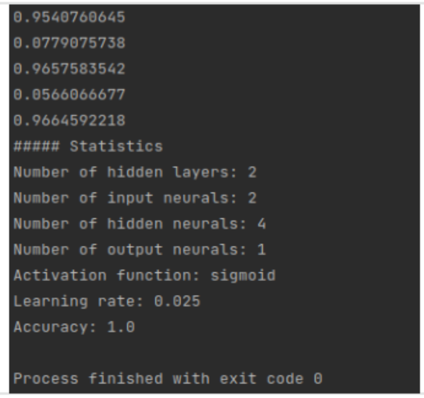
由上二圖得知，neural network對兩種input data皆可正確地預測答案。

3. Show the accuracy of your prediction



```
0.0024920801
0.0032496772
0.9994571252
0.0000640140
0.9996139195
##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0

Process finished with exit code 0
```

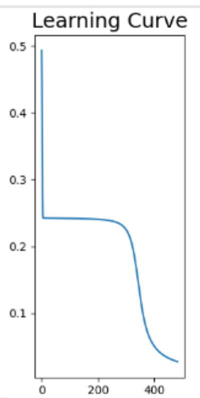
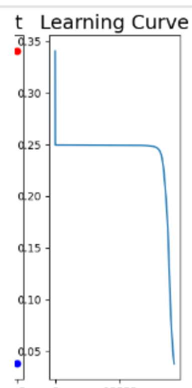


```
0.9540760645
0.0779075738
0.9657583542
0.0566066677
0.9664592218
##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0

Process finished with exit code 0
```

由上二圖得知，neural network對兩種input data皆能有100%的accuracy。

4. Learning curve (loss, epoch curve)

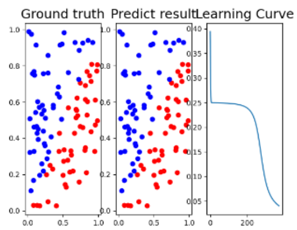
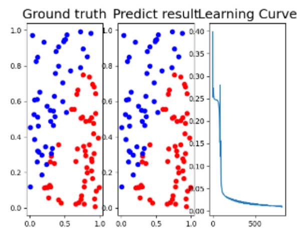
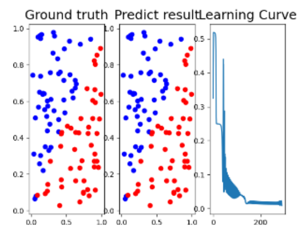
Linear	XOR
	

由上二圖得知，neural network對兩種input data的learning curve相似，到0.25左右會趨於平緩一段時間，才會再開始下降。

- Discussion (30%)

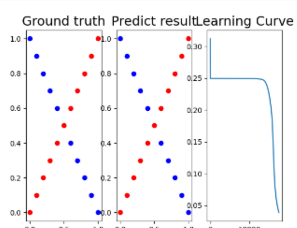
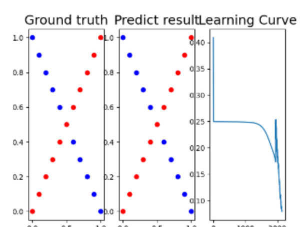
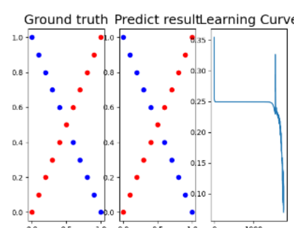
1. Try different learning rates

  - Linear

Learning Rate = 0.025	Learning Rate = 0.1	Learning Rate = 0.4
		
0.1216899959 0.5797512843 0.0378201360 0.5918966201 0.2076992395 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	0.0001486433 0.9999626536 0.0001623582 0.0001485052 0.0129768268 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.1 Accuracy: 1.0	0.9978103758 0.0013035194 0.0077800678 0.0073706503 0.0119268100 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.4 Accuracy: 1.0

當learning rate變大，會導致learning curve上下震盪，原因是儘管gradient方向是對的，但由於一次跨太大步，反而導致loss上升。

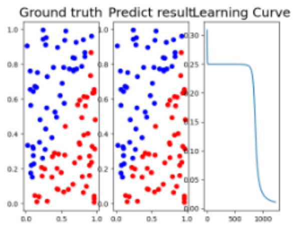
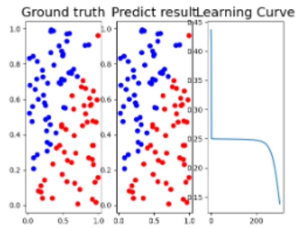
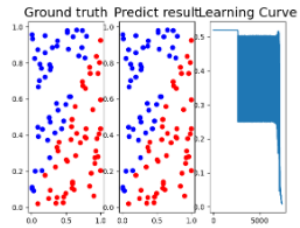
- XOR

Learning Rate = 0.025	Learning Rate = 0.1	Learning Rate = 0.4
		
0.9676236302 0.0603967904 0.9735008445 0.0383392316 0.9702133294 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	0.8876703087 0.1553639825 0.9371704038 0.1166118430 0.9549296992 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.1 Accuracy: 1.0	0.9410195909 0.1581666234 0.9580810654 0.1285851091 0.9631140764 ##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.4 Accuracy: 1.0

和linear input data的結果相似，當learning rate變大，會導致learning curve上下震盪。

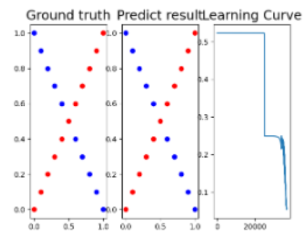
2. Try different numbers of hidden units

- Linear

Hidden Units = 2	Hidden Units = 8	Hidden Units = 24
 	 	
epoch: 1223 loss: 0.011191747069183654 epoch: 1224 loss: 0.011167337878149958 epoch: 1225 loss: 0.011143055076267562 epoch: 1226 loss: 0.011118897580068933 epoch: 1227 loss: 0.011094864318831022	epoch: 302 loss: 0.1498720723851483 epoch: 303 loss: 0.14690010791603123 epoch: 304 loss: 0.14394922288858306 epoch: 305 loss: 0.141029064887967 epoch: 306 loss: 0.13815308755229871	epoch: 7427 loss: 0.009378387707729098 epoch: 7428 loss: 0.009424739980691498 epoch: 7429 loss: 0.009312846430101988 epoch: 7430 loss: 0.009353611376193378 epoch: 7431 loss: 0.009249408238304786
##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 2 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 8 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 24 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0

當hidden units的數量太大或太小，在目標是accuracy為1.0的情況下，都會造成epoch數量增加，因此應根據情況，適度調整hidden units的數量，以節省training時間。

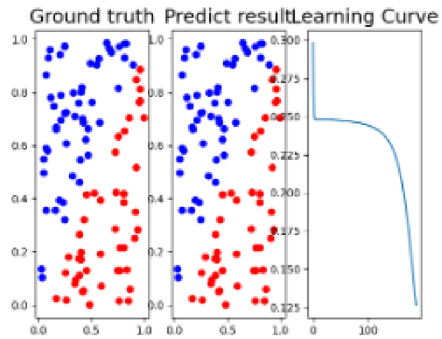
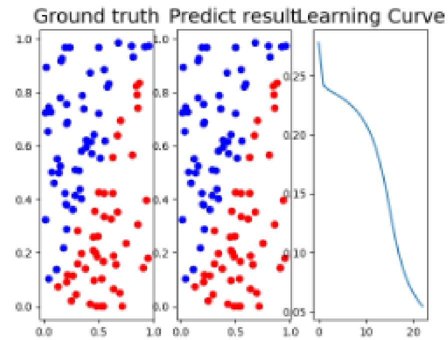
- XOR

<table>
<tr><th>Hidden Units = 2</th><th>Hidden Units = 8</th><th>Hidden Units = 24</th></tr>
<tr><td>
</td><td>
</td><td>
</td></tr>
<tr><td>epoch: 40555 loss: 0.057163908151987325
epoch: 40556 loss: 0.057146160462631844
epoch: 40557 loss: 0.057128419299621015
epoch: 40558 loss: 0.057110684663810245
epoch: 40559 loss: 0.05709295655605229</td><td>epoch: 10523 loss: 0.09172553764321324
epoch: 10524 loss: 0.09162143039004034
epoch: 10525 loss: 0.09151754361768202
epoch: 10526 loss: 0.09141387674585005
epoch: 10527 loss: 0.09131042919433197</td><td>epoch: 36752 loss: 0.05405959843664789
epoch: 36753 loss: 0.05570632819920152
epoch: 36754 loss: 0.05395424976437237
epoch: 36755 loss: 0.05558850856642315
epoch: 36756 loss: 0.053849376983899466</td></tr>
<tr><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 2
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0</td><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 8
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0</td><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 24
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0</td></tr>
</table>

和linear input data的結果相似，在目標是accuracy為1.0的情況下，都會造成epoch數量增加。

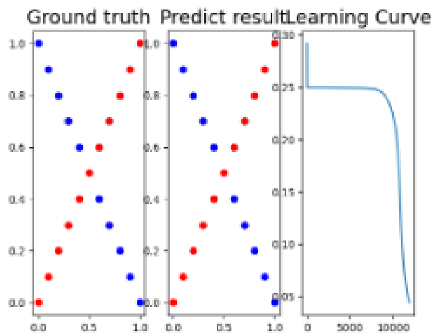
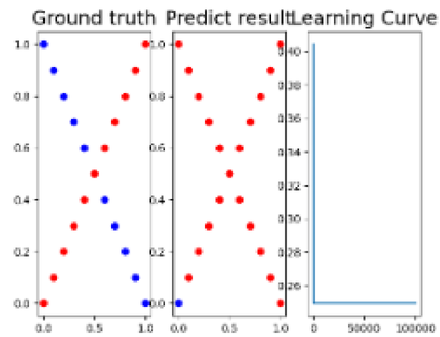
3. Try without activation functions

- Linear

with Activation Function	without Activation Function
 	
epoch: 183 loss: 0.1411674240106308 epoch: 184 loss: 0.13745244003303872 epoch: 185 loss: 0.13375793825199012 epoch: 186 loss: 0.1300956537025607 epoch: 187 loss: 0.1264766140976646	epoch: 19 loss: 0.08466033527346221 epoch: 20 loss: 0.07425754093511439 epoch: 21 loss: 0.0662438399444001 epoch: 22 loss: 0.059908386042964216 epoch: 23 loss: 0.05478242731874495
##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: none Learning rate: 0.025 Accuracy: 1.0

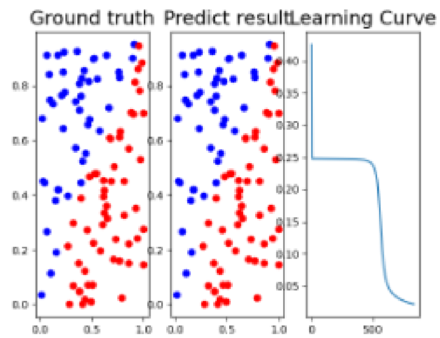
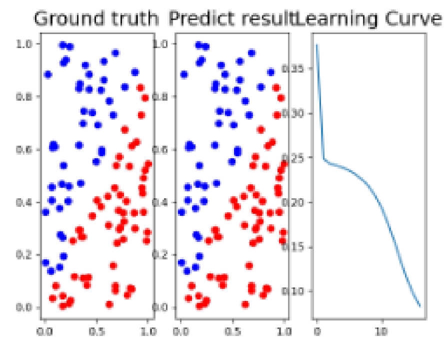
沒有activation function，在目標是accuracy為1.0的情況下，具有更快的收斂速度，原因可能是Sigmoid將每一層的output限縮在0~1之間所導致的。

- XOR

<table>
<tr><th>with Activation Function</th><th>without Activation Function</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>epoch: 11899 loss: 0.04460671605603022
epoch: 11900 loss: 0.044573208834464924
epoch: 11901 loss: 0.044539734053686456
epoch: 11902 loss: 0.04450629162585623
epoch: 11903 loss: 0.044472881463415874</td><td>epoch: 99996 loss: 0.24953706217711666
epoch: 99997 loss: 0.24953706217711666
epoch: 99998 loss: 0.24953706217711666
epoch: 99999 loss: 0.24953706217711666
epoch: 100000 loss: 0.24953706217711666</td></tr>
<tr><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0</td><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: none
Learning rate: 0.025
Accuracy: 0.47619047619047616</td></tr>
</table>

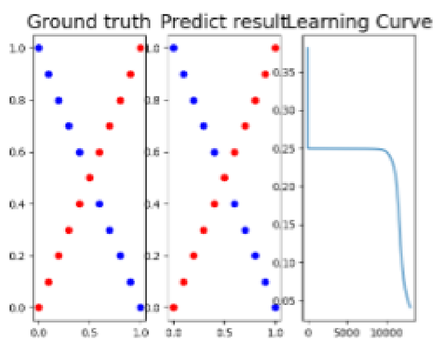
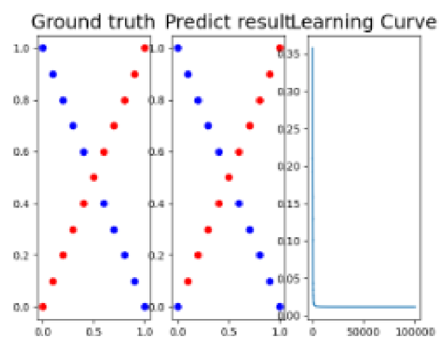
在XOR data points中的結果，與linear data points的結果差異慎大，儘管epoch已經來到了100000，仍舊無法收斂。因此可判斷，with activation function可以增加training穩定性，儘管在一些input dataset會導致更長的training時間，但同時也避免在某些dataset無法收斂的問題。

- Extra
  1. Implement different activation functions. (3%)
     - Linear

<table>
<tr><th>Sigmoid</th><th>ReLU</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>epoch: 834 loss: 0.021743386046934807
epoch: 835 loss: 0.021695127212803086
epoch: 836 loss: 0.0216471223400041
epoch: 837 loss: 0.021599369260363863
epoch: 838 loss: 0.02155186583373257</td><td>epoch: 13 loss: 0.1560404675504458
epoch: 14 loss: 0.13385886982307849
epoch: 15 loss: 0.11294815330628215
epoch: 16 loss: 0.09585747732498538
epoch: 17 loss: 0.08287752058749294</td></tr>
<tr><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: sigmoid
Learning rate: 0.025
Accuracy: 1.0</td><td>##### Statistics
Number of hidden layers: 2
Number of input neurals: 2
Number of hidden neurals: 4
Number of output neurals: 1
Activation function: relu
Learning rate: 0.025
Accuracy: 1.0</td></tr>
</table>

ReLU在目標是accuracy為1.0的情況下，具有更快的收斂速度，原因可能是Sigmoid將每一層的output限縮在0~1之間所導致的。

- XOR

Sigmoid	ReLU
 	
epoch: 13003 loss: 0.04120868890310717 epoch: 13004 loss: 0.0411808893115604 epoch: 13005 loss: 0.04115311034863617 epoch: 13006 loss: 0.041125351971475724 epoch: 13007 loss: 0.04109761413735052	epoch: 99996 loss: 0.011911420905913541 epoch: 99997 loss: 0.01191142042248732 epoch: 99998 loss: 0.011911420222302242 epoch: 99999 loss: 0.011911420322073416 epoch: 100000 loss: 0.011911420578684979
##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: sigmoid Learning rate: 0.025 Accuracy: 1.0	##### Statistics Number of hidden layers: 2 Number of input neurals: 2 Number of hidden neurals: 4 Number of output neurals: 1 Activation function: relu Learning rate: 0.025 Accuracy: 0.9523809523809523

在XOR data points中的結果，與linear data points的結果差異慎大，儘管epoch已經來到了100000，ReLU的accuracy仍未達到1.0。因此可判斷，Sigmoid相對ReLU可以增加training穩定性，儘管在一些input dataset會導致更長的training時間。

- Reference

1. What Should I Use for Dot Product and Matrix Multiplication?
2. Neural Network - 3Blue1Brown
3. Backpropagation - Hung-yi Lee