Lab 01 Backpropagation

1. Introduction

   My program contains :

   1. generation_func.py : Create linear and xor datasets .
   2. Two_HL_net.py : Model of two hidden layers model .
   3. convolution_net.py : Model of convolution layers model .
   4. main.py : Implement training and testing .

2. Experiment setups

   A. Sigmoid function :

   Using sigmoid function for activation function (can choose tanh function or no using):

```
def activation(self, X):
    if self.act == 'sigmod':
        return 1.0/(1.0 + np.exp(-X))
    if self.act == 'tanh':
        return np.tanh(X)
    if self.act == 'no':
        return X

def derivative_activation(self, X):
    if self.act == 'sigmod':
        return X * (1.0 - X)
    if self.act == 'tanh':
        return 1.0 - X**2
    if self.act == 'no':
        return X
```

   B. Neural network :

   Design two hidden layers and one output layer , each hidden layer have n units (default is 3) , layer 1 expect two inputs , and output layer is n to one .

```
class two_hl_net(object):
    def __init__(self, n=3, lr=0.1, act='sigmod'):
        # initial neural networks layers
        self.hidden_weight_1 =  np.random.uniform(size=(2,n))
        self.hidden_weight_2 =  np.random.uniform(size=(n,n))
        self.output_weight =  np.random.uniform(size=(n,1))
        # initial output for each layer
        self.hidden_output_1 = None
        self.hidden_output_2 = None
        self.output = None
        # training parameters
        self.mode = 'train'
        self.lr = lr
        self.act = act
```

Forward propagation (X is input data) :

```
def forward(self, X):
    # forward propagation
    self.hidden_output_1 = self.activation(np.dot(X, self.hidden_weight_1))
    self.hidden_output_2 = self.activation(np.dot(self.hidden_output_1, self.hidden_weight_2))
    self.output = self.activation(np.dot(self.hidden_output_2, self.output_weight))
```

Loss function (x is input data , y is label) :

```
def backward(self,x,y):
    error_output = y - self.output
```

```
return abs(np.mean(error_output))
```

C. Backward propagation (x is input data , y is label) :
After getting the output loss , computing the gradient and loss of each layer . And then update the weight .

```
def backward(self,x,y):
    error_output = y - self.output
    if self.mode == 'train':
        # get loss and back propagation
        d_output = error_output*self.derivative_activation(self.output)

        error_h2 = d_output.dot(self.output_weight.T)
        d_hidden_output_2 = error_h2*self.derivative_activation(self.hidden_output_2)

        error_h1 = error_h2.dot(self.hidden_weight_2.T)
        d_hidden_output_1 = error_h1*self.derivative_activation(self.hidden_output_1)

        # update weight
        self.output_weight += self.hidden_output_2.T.dot(d_output)*self.lr
        self.hidden_weight_2 += self.hidden_output_1.T.dot(d_hidden_output_2)*self.lr
        self.hidden_weight_1 += x.T.dot(d_hidden_output_1)*self.lr

    return abs(np.mean(error_output))
```

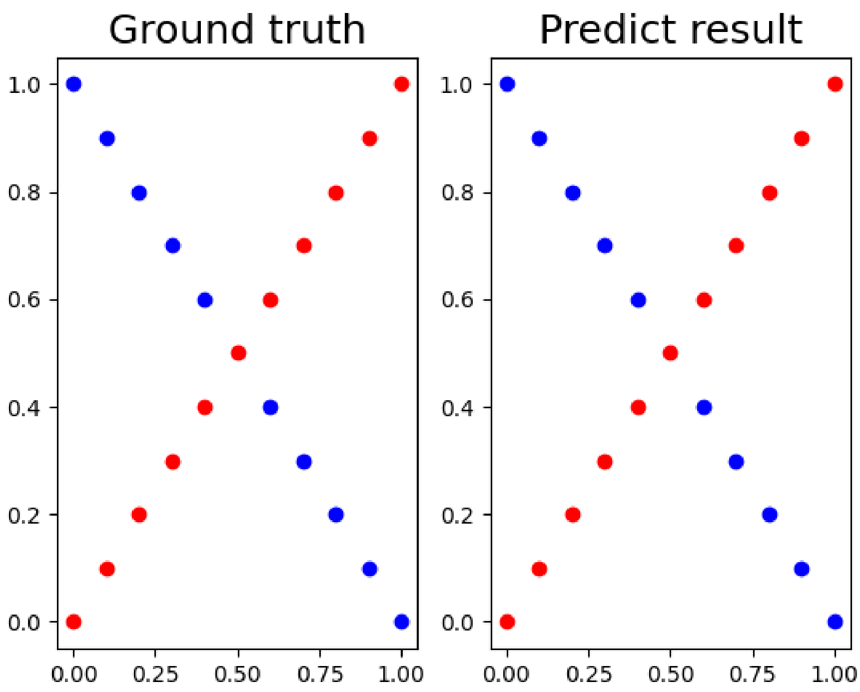
3. Result of testing :
(All batch size of training and testing are full data size)
XOR Train (epoch=100000 , n=3 , lr=0.1 , activation func=sigmoid) :

```
PS C:\Users\88697\nctu_hw\DL> python -u "c:\Users\88697\nctu_hw\DL\DL_hw1\main.py"
Start training ...
Epoch : 10000 || Train Loss : 0.0009156542 || Test Loss : 0.0009155180
Epoch : 20000 || Train Loss : 0.0003988270 || Test Loss : 0.0003988066
Epoch : 30000 || Train Loss : 0.0002693909 || Test Loss : 0.0002693826
Epoch : 40000 || Train Loss : 0.0002075968 || Test Loss : 0.0002075922
Epoch : 50000 || Train Loss : 0.0001706357 || Test Loss : 0.0001706327
Epoch : 60000 || Train Loss : 0.0001457468 || Test Loss : 0.0001457447
Epoch : 70000 || Train Loss : 0.0001277071 || Test Loss : 0.0001277055
Epoch : 80000 || Train Loss : 0.0001139572 || Test Loss : 0.0001139560
Epoch : 90000 || Train Loss : 0.0001030865 || Test Loss : 0.0001030855
Epoch : 100000 || Train Loss : 0.0000942494 || Test Loss : 0.0000942486
Finish training
Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]
```

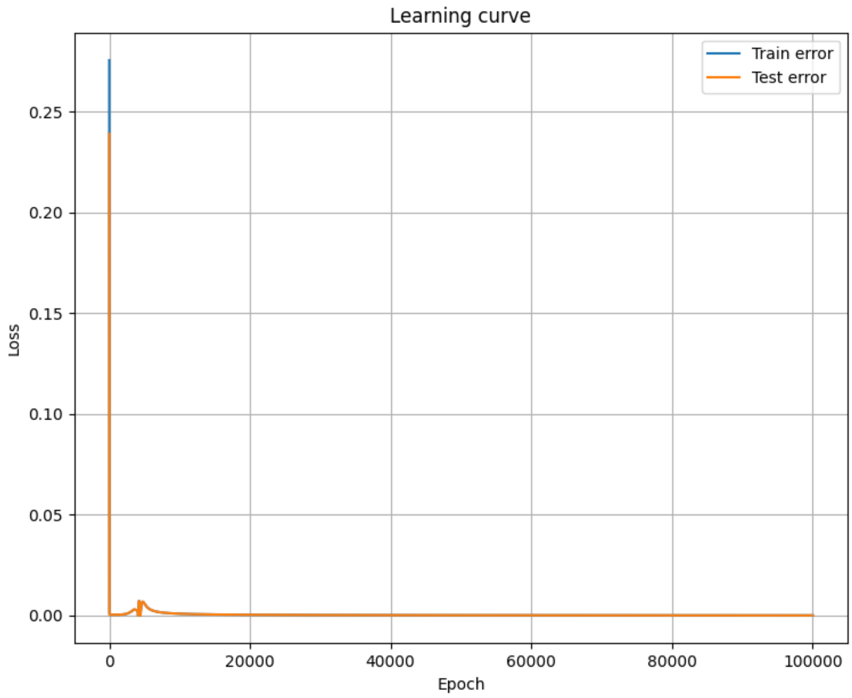
Predition result :

```
[[9.21533645e-02]
 [9.98992620e-01]
 [5.69983605e-03]
 [9.99391289e-01]
 [3.27714706e-04]
 [9.99493341e-01]
 [5.85676566e-05]
 [9.99522908e-01]
 [3.64634894e-05]
 [9.95903513e-01]
 [5.11102352e-05]
 [1.10366340e-04]
 [9.55496210e-01]
 [2.90471657e-04]
 [9.63415521e-01]
 [8.15906488e-04]
 [9.63658844e-01]
 [2.27243298e-03]
 [9.63709746e-01]
 [6.02484617e-03]
 [9.63727356e-01]]
```

Comparison figure :



Learning curve :



Linear Train (epoch=100000 , n=3 , lr=0.1 , activation func=sigmoid) :

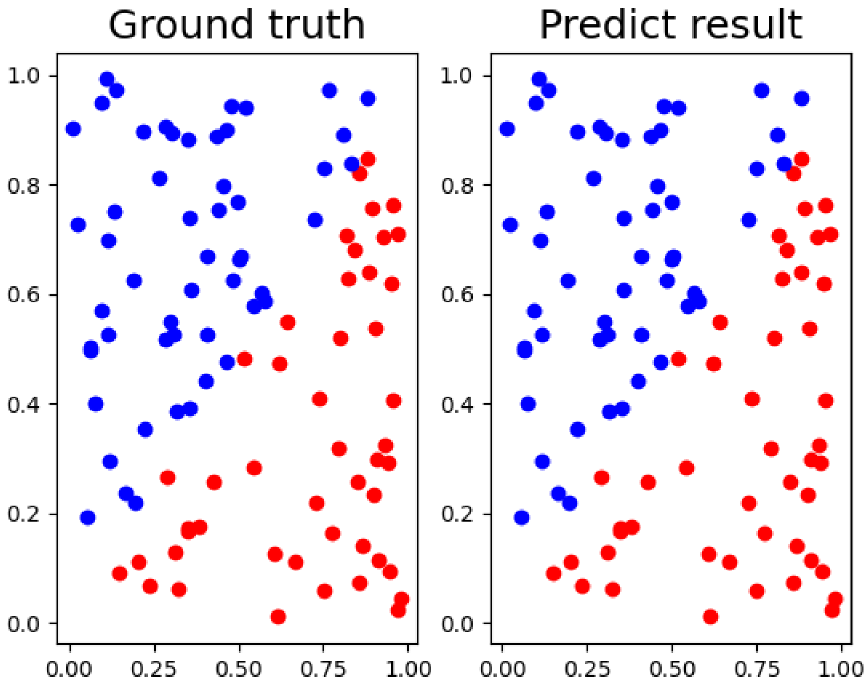
```
Epoch : 10000 || Train Loss : 0.0005584393 || Test Loss : 0.0025881151
Epoch : 20000 || Train Loss : 0.0002437875 || Test Loss : 0.0017019254
Epoch : 30000 || Train Loss : 0.0001536686 || Test Loss : 0.0013793765
Epoch : 40000 || Train Loss : 0.0001113417 || Test Loss : 0.0012183873
Epoch : 50000 || Train Loss : 0.0000864435 || Test Loss : 0.0011306698
Epoch : 60000 || Train Loss : 0.0000696822 || Test Loss : 0.0010865968
Epoch : 70000 || Train Loss : 0.0000572472 || Test Loss : 0.0010752245
Epoch : 80000 || Train Loss : 0.0000472337 || Test Loss : 0.0010943059
Epoch : 90000 || Train Loss : 0.0000385087 || Test Loss : 0.0011477531
Epoch : 100000 || Train Loss : 0.0000302654 || Test Loss : 0.0012458939
Finish training
```

```
Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]
```

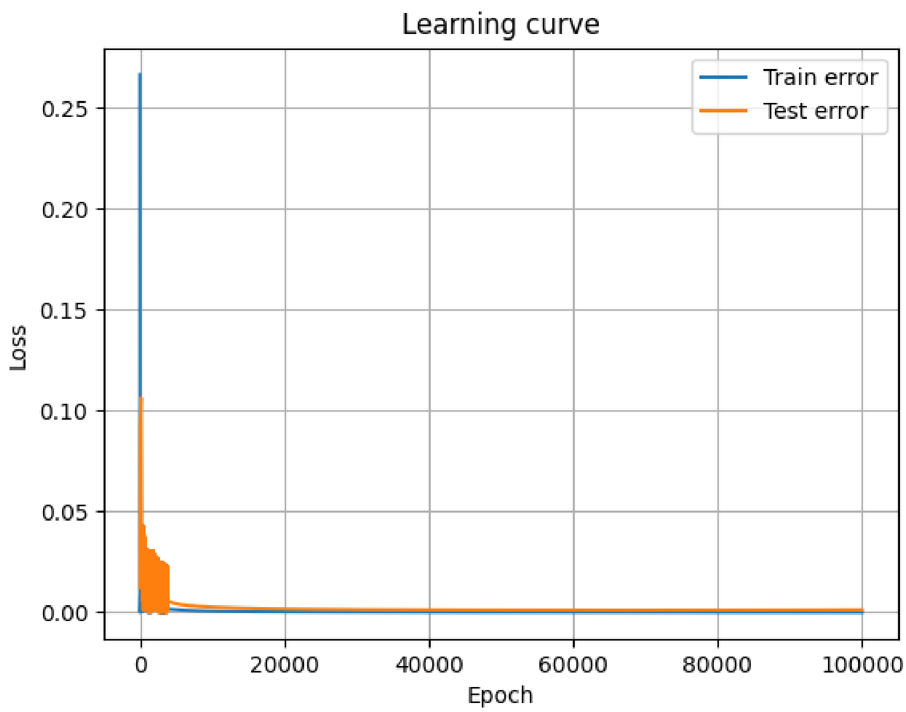
Predition result :

```
[[9.99999992e-01]  [3.83906377e-05]  [9.99999091e-01]  [3.64155408e-05]
 [9.99999530e-01]  [2.86410127e-05]  [4.10994754e-05]  [9.99999841e-01]
 [4.35283783e-05]  [5.24299941e-05]  [4.03197357e-05]  [9.99999969e-01]
 [9.99999139e-01]  [4.33507041e-05]  [9.99999990e-01]  [2.96533357e-04]
 [9.99999173e-01]  [3.89444091e-05]  [9.99999991e-01]  [2.19641295e-05]
 [9.87251451e-01]  [9.99999940e-01]  [3.81467939e-05]  [9.62853000e-01]
 [3.98353621e-05]  [1.71188751e-04]  [9.99999947e-01]  [9.99999990e-01]
 [9.99999577e-01]  [3.59400102e-05]  [9.99999966e-01]  [4.32021817e-05]
 [4.49056414e-05]  [9.99999991e-01]  [4.84624041e-05]  [2.38463698e-05]
 [9.99999990e-01]  [2.00260367e-05]  [9.99999934e-01]  [3.97697570e-05]
 [9.99999965e-01]  [2.93015366e-05]  [4.40598243e-05]  [9.99999959e-01]
 [3.74401102e-05]  [9.99999987e-01]  [9.99999995e-01]  [9.99999989e-01]
 [9.99999993e-01]  [9.99999989e-01]  [9.99999990e-01]  [9.99999892e-01]
 [9.99763262e-01]  [9.99999993e-01]  [9.99996629e-01]  [4.01696397e-05]
 [4.90122518e-05]  [2.73880176e-04]  [4.40784672e-05]  [9.99999991e-01]
 [9.99999171e-01]  [4.50809720e-05]  [9.85238564e-01]  [2.78249941e-05]
 [4.67196633e-05]  [4.97032814e-05]  [2.12914370e-05]  [1.96419601e-05]
 [4.68784749e-05]  [4.42237754e-05]  [9.99999043e-01]  [9.99999984e-01]
 [9.99999993e-01]  [3.80849465e-05]  [4.07396290e-05]  [9.99999974e-01]
 [9.99999986e-01]  [1.74828236e-04]  [2.35846303e-05]  [9.99999887e-01]
 [3.63255075e-05]  [9.99999961e-01]  [5.17420367e-05]  [9.99999990e-01]
 [9.99999993e-01]  [9.99999986e-01]  [9.99999968e-01]  [9.99999885e-01]]
 [9.99999847e-01]  [9.99991119e-01]  [3.94330982e-05]
                   [9.99999989e-01]  [9.37789177e-01]
                   [3.55336970e-05]  [9.99999988e-01]
                   [3.44349423e-05]  [2.18523090e-05]
                                     [9.99999980e-01]
                                     [9.99999917e-01]
```

Comparison figure :



Learning curve :

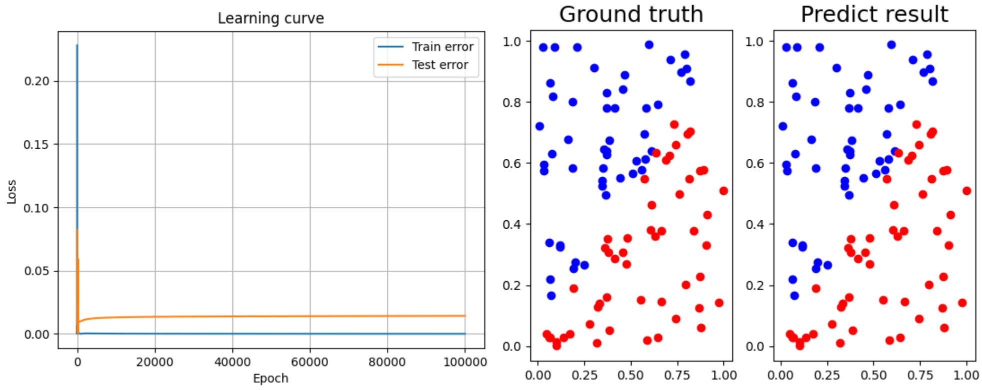


4. Discussion (each testing try two run for two different set of training and testing data) :

   Different learning rates (linear case , n=3 , activation func=sigmoid):

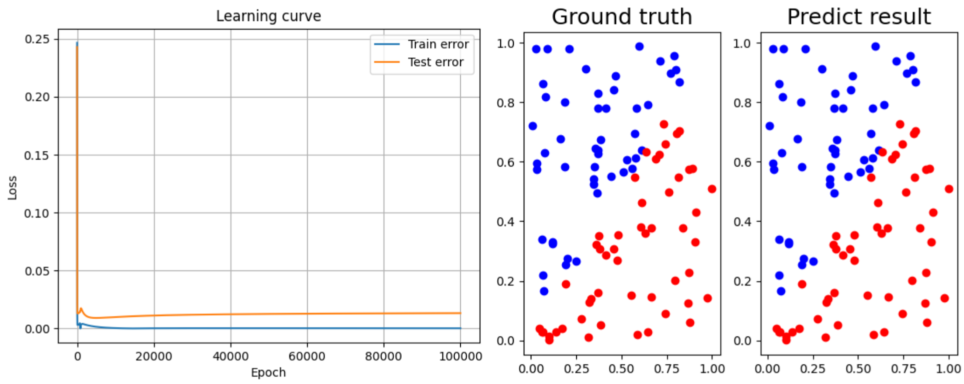
Run 1 :

lr = 0.1 :



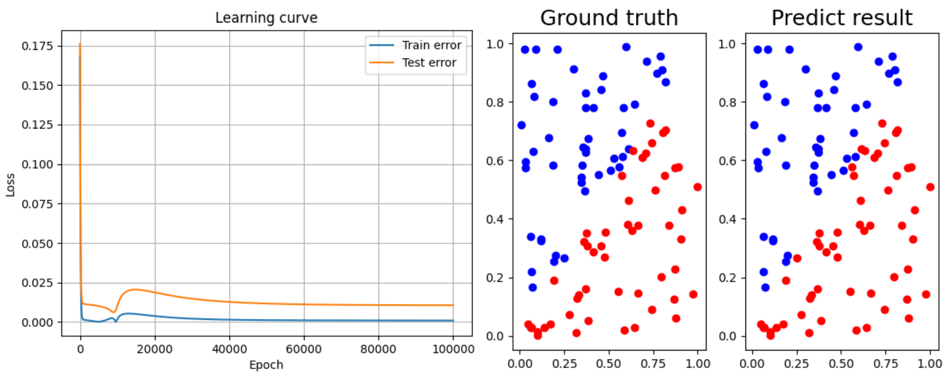
Accuracy rate : [0 : 100.0 percent , 1 : 100.0 percent]

lr = 0.01



Accuracy rate : [0 : 100.0 percent , 1 : 100.0 percent]

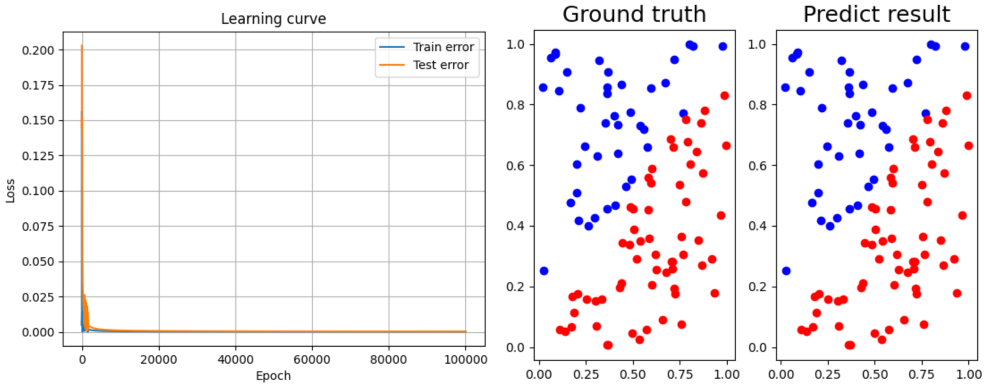
lr = 0.001



Accuracy rate : [0 : 100.0 percent , 1 : 93.9 percent]

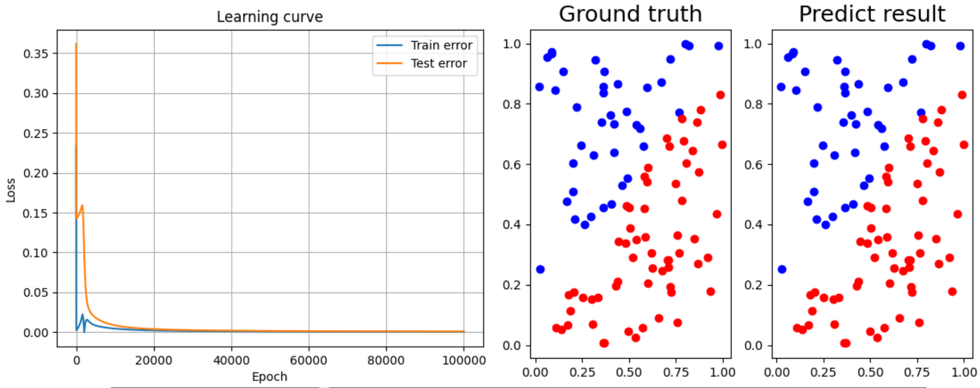
Run 2 :

lr = 0.1



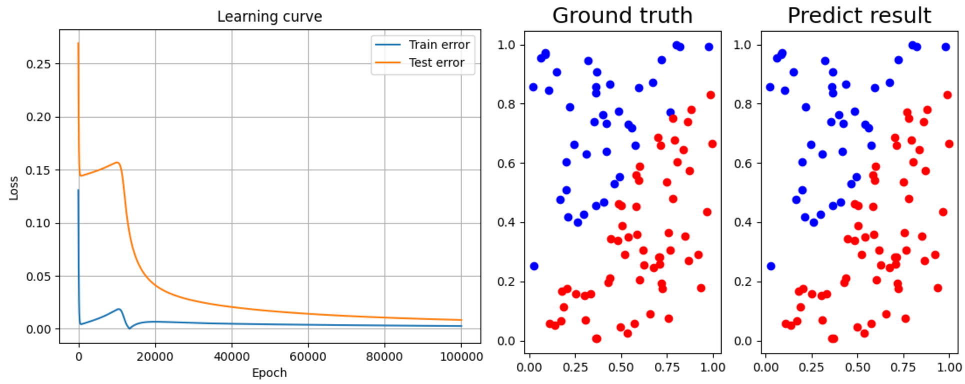
Accuracy rate : [0 : 100.0 percent , 1 : 100.0 percent]

lr = 0.01



Accuracy rate : [0 : 100.0 percent , 1 : 100.0 percent]

lr = 0.001



Accuracy rate : [0 : 100.0 percent , 1 : 97.5 percent]

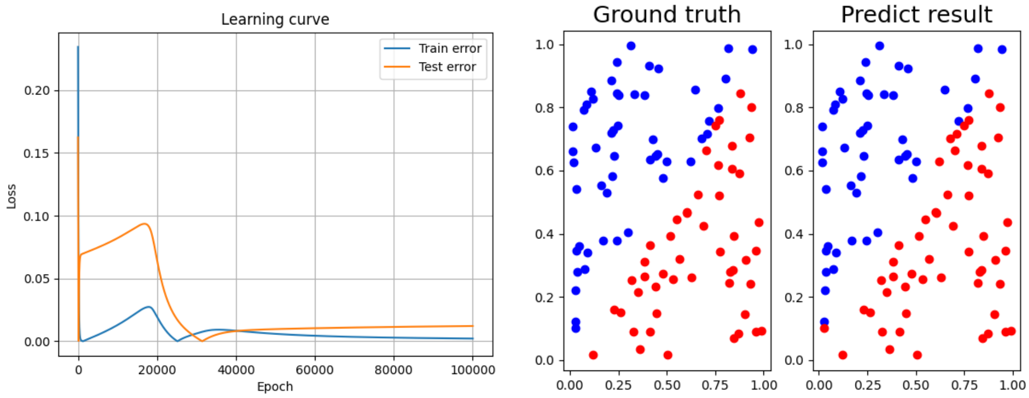
In the case of a lower learning rate , the loss curve will have a greater oscillation before epoch = 20000 .

Different numbers of hidden units (linear case , lr=0.001 , activation func=sigmoid):

Set lr = 0.001 , to see if using more hidden unit can reduce the oscillation .

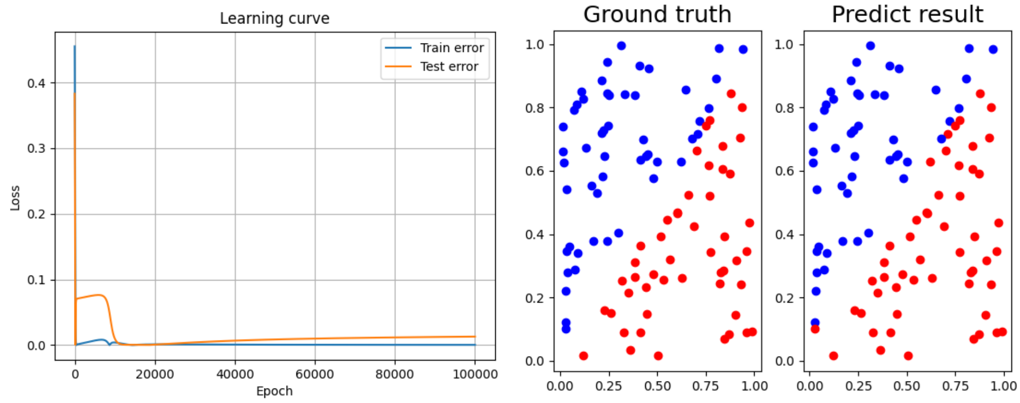
Run 1 :

n = 3 :



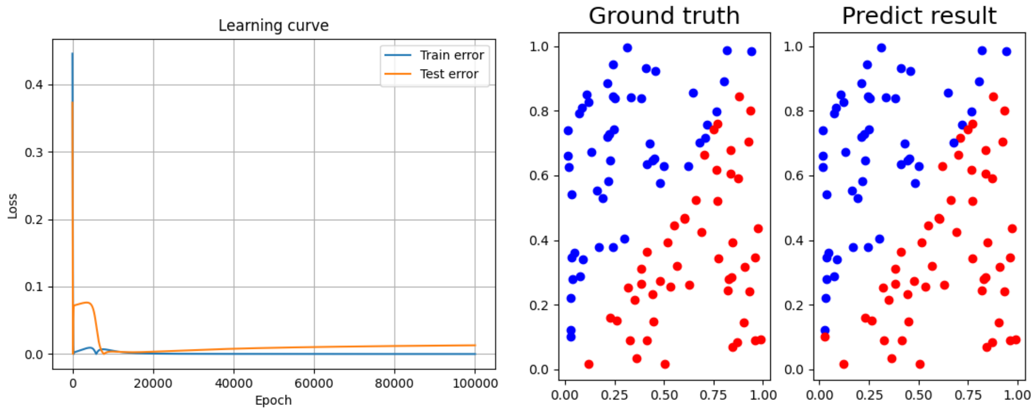
Accuracy rate : [0 : 100.0 percent , 1 : 92.2 percent]

n = 4 :



Accuracy rate : [0 : 100.0 percent , 1 : 94.1 percent]

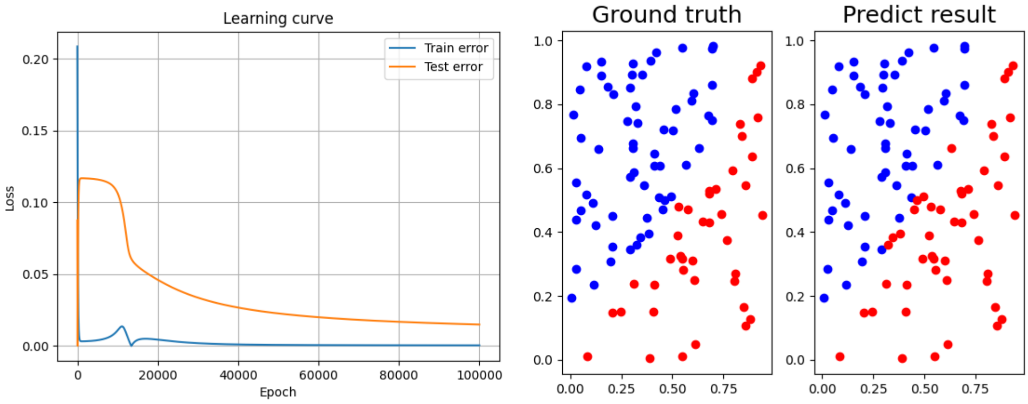
n = 5 :



Accuracy rate : [0 : 100.0 percent , 1 : 94.1 percent]

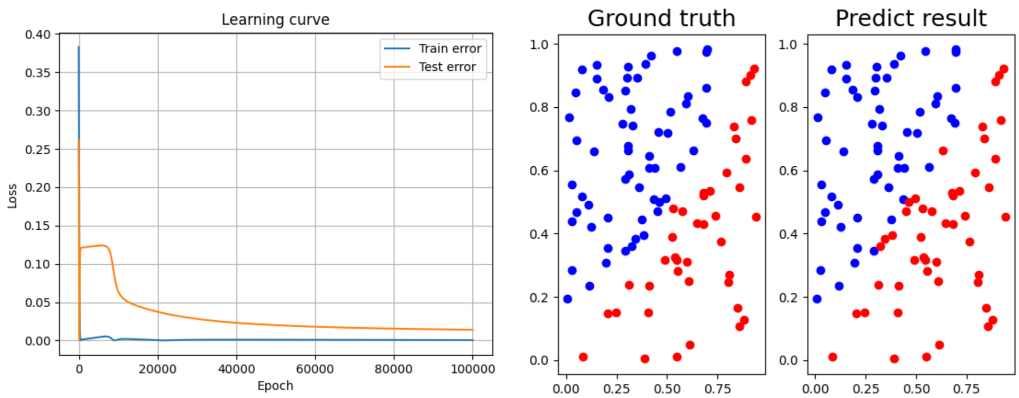
Run 2 :

n = 3



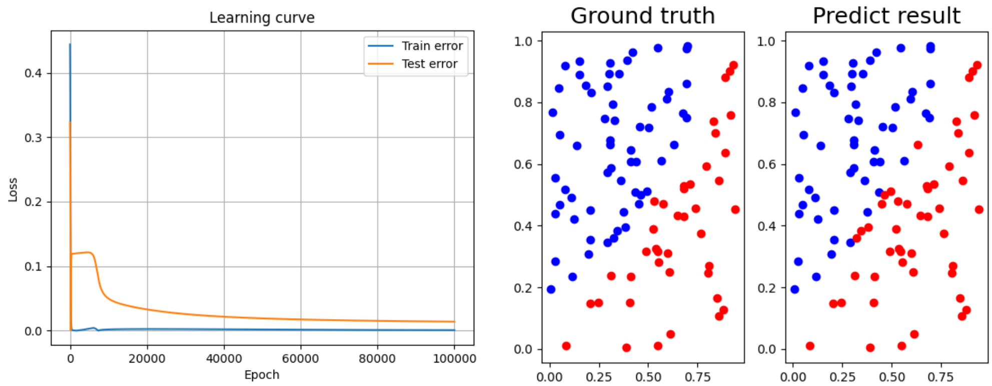
`Accuracy rate : [ 0 : 100.0 percent , 1 : 88.3 percent]`

n = 4



`Accuracy rate : [ 0 : 100.0 percent , 1 : 88.3 percent]`

n = 5



`Accuracy rate : [ 0 : 100.0 percent , 1 : 88.3 percent]`

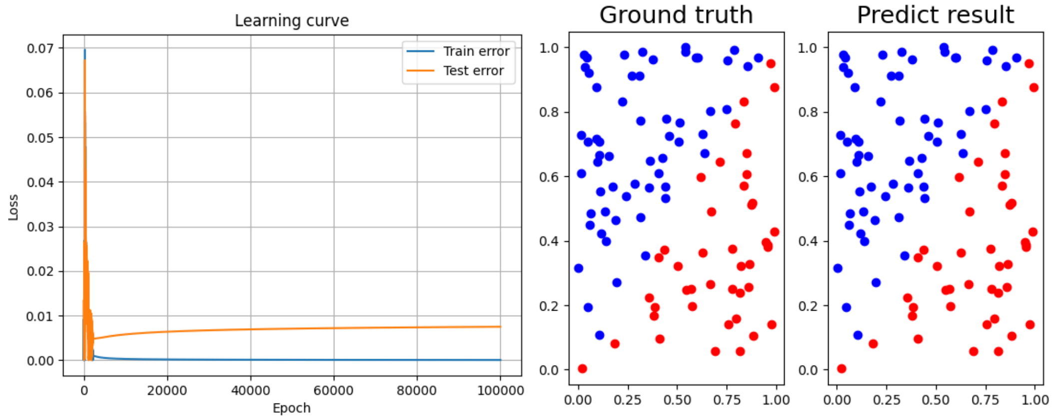
For more hidden units , it can reduce the oscillation . And look at the case of n = 4 , 5 , it seem the more hidden unit is used , the early loss of testing reduce .

Training without sigmoid function :

(linear case , n = 3 , lr=0.1) :

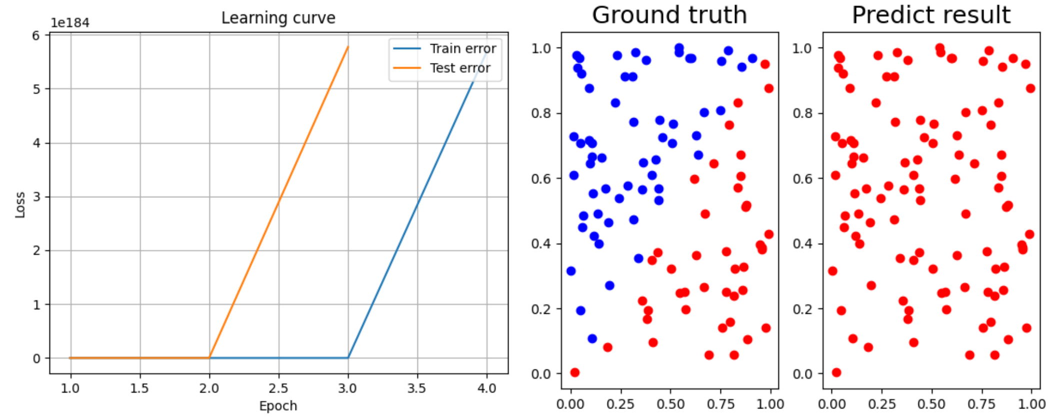
Run 1 :

Use sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 100.0 percent]

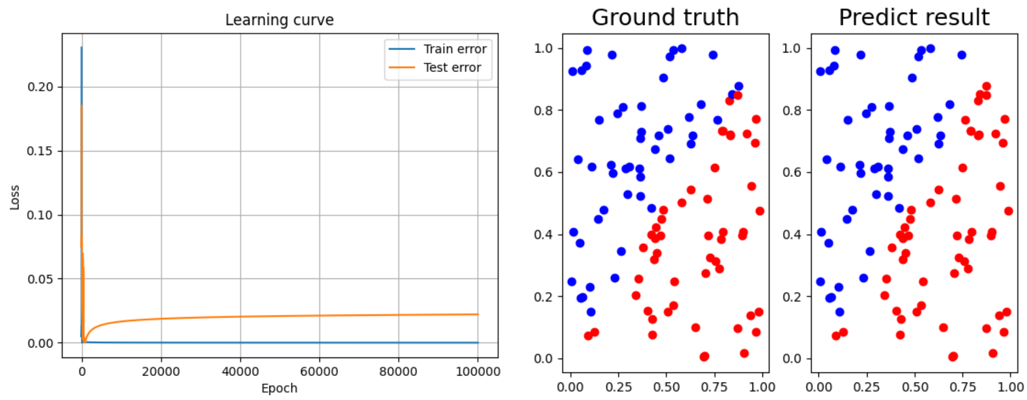
Without sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 0.0 percent]

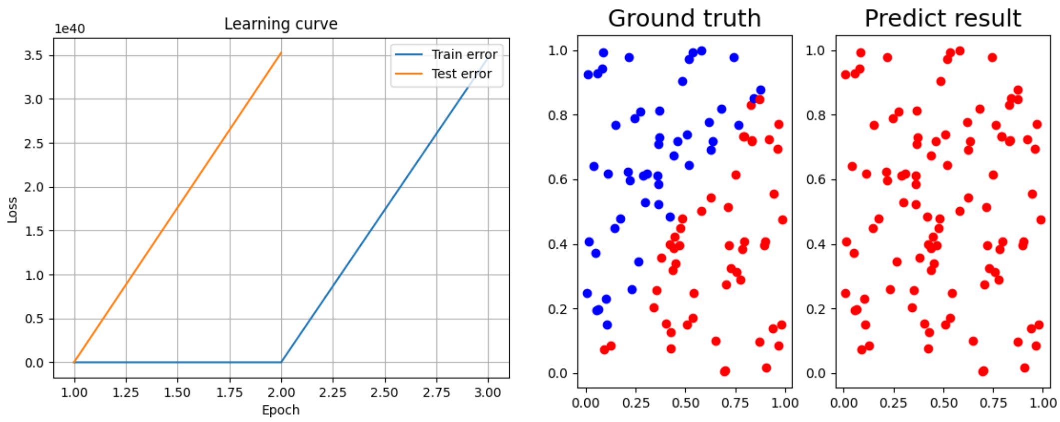
Run 2 :

Use sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 93.9 percent]

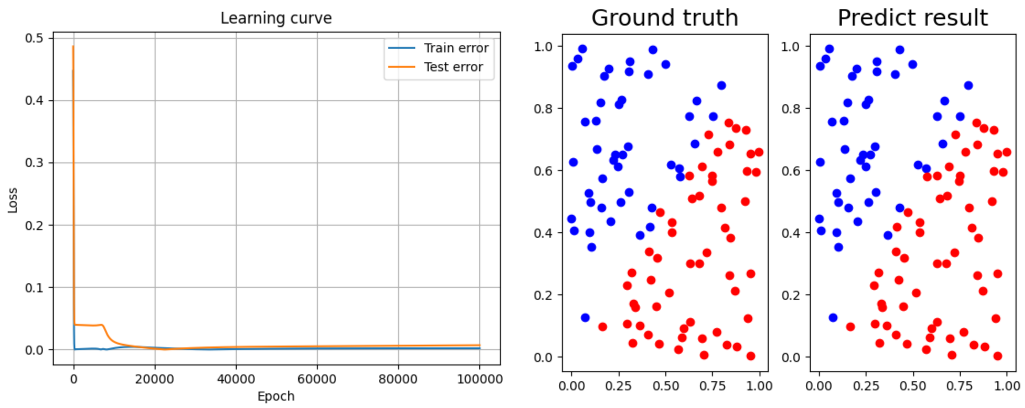
Without sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 0.0 percent]

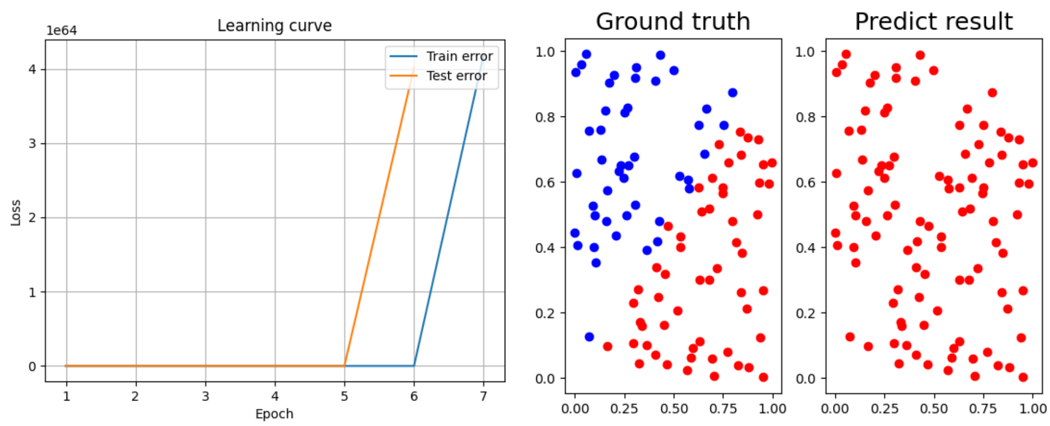
Try using n=5 , lr = 0.001 :

Use sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 95.6 percent]

Without sigmoid function :



Accuracy rate : [0 : 100.0 percent , 1 : 0.0 percent]

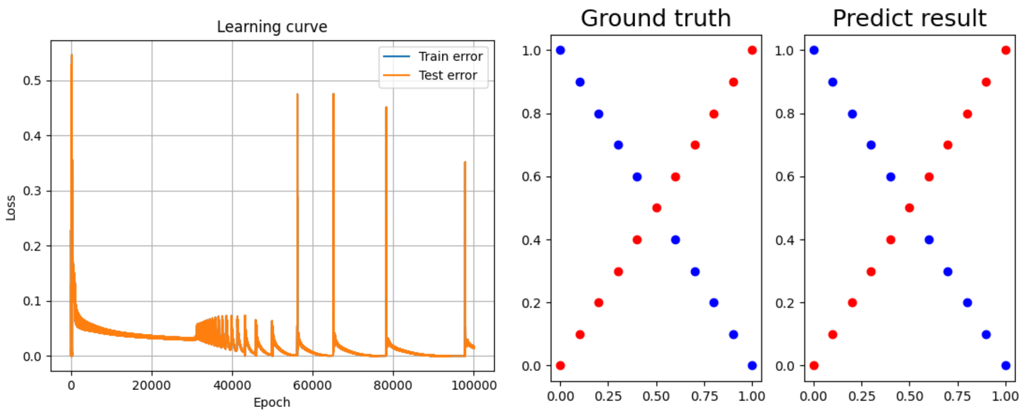
Without sigmoid function , it is totally unable to learn .

5. Extra

Implement different activation function :

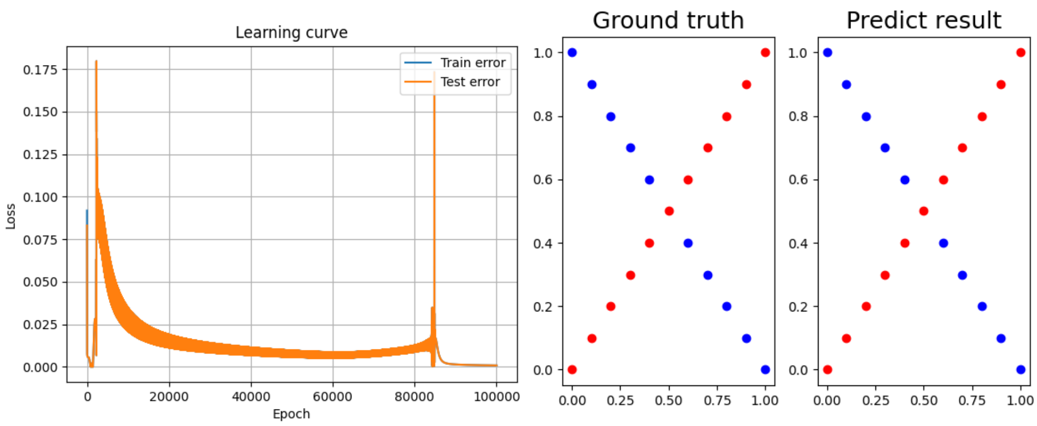
Using tanh function as activation function .

Xor (n = 3 , lr=0.1) :



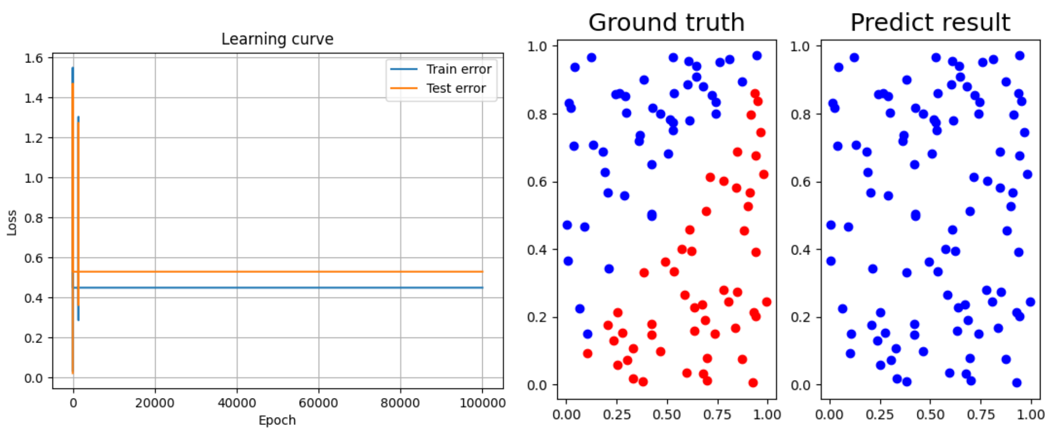

```
Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]
```

Xor (n = 3 , lr=0.01) :



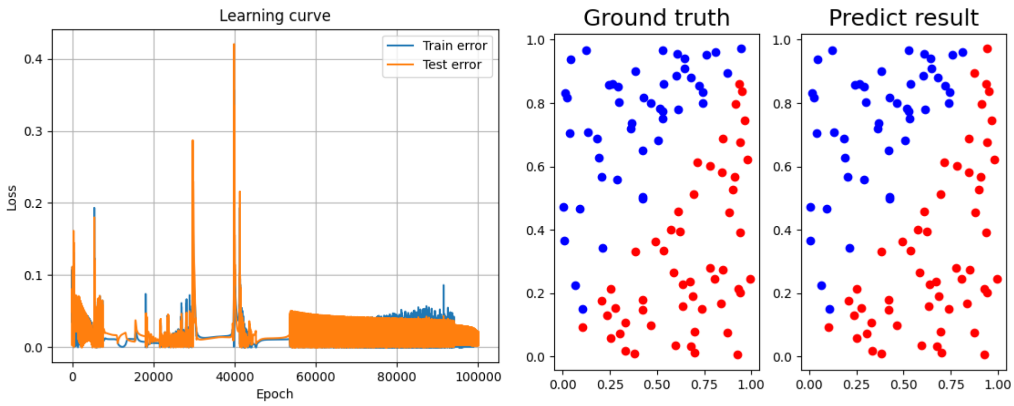

```
Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]
```

Linear (n = 3 , lr=0.1) :



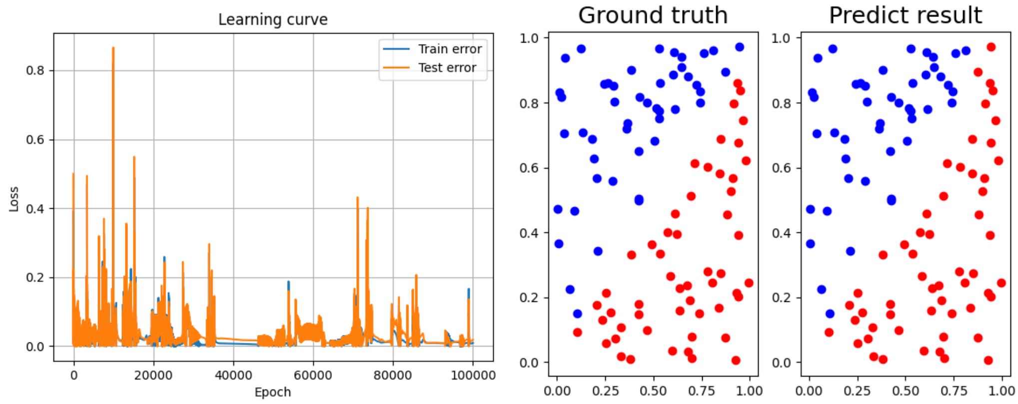

```
Accuracy rate : [ 0 : 0.0 percent , 1 : 100.0 percent]
```

Linear (n = 3 , lr=0.01) :




```
Accuracy rate : [ 0 : 100.0 percent , 1 : 95.7 percent]
```

Linear (n = 5 , lr=0.01) :




```
Accuracy rate : [ 0 : 100.0 percent , 1 : 95.7 percent]
```

Implement convolution layer :

Network :

Design the 1x2x1 dimension convolution layer (1 channels , each channel has 1x2 dimension) and one hidden layer (defalt 3 units) and one output layer . It can adjust M to change the numbers of channels of convolution layer 1x2xM , default M is 1 .

```
class Conv(object):
    def __init__(self, M=5, n=3, lr=0.01, act='sigmod'):
        # initial neural networks layers
        self.conv =  np.random.uniform(size=(2,M)) # M = 5
        self.hidden_weight_1 =  np.random.uniform(size=(M,n))
        self.output_weight =  np.random.uniform(size=(n,1))
        # initial output for each layer
        self.conv_output = None
        self.hidden_output_1 = None
        self.output = None
        # training parameters
        self.M = M
        self.mode = 'train'
        self.lr = lr
        self.act = act
```

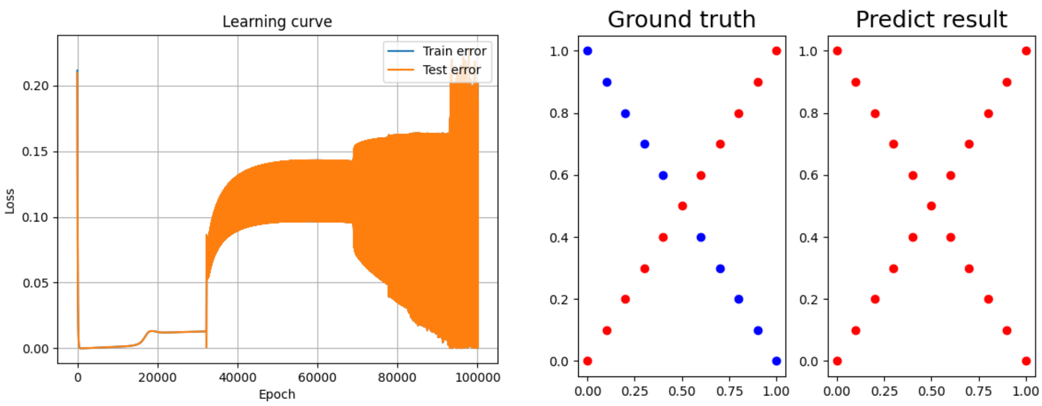
Forward propagation :

```python
def forward(self, X):
    # forward propagation
    self.conv_output = np.zeros((X.shape[0], self.M))
    # do convolution
    for i in range(X.shape[0]):
        for j in range(self.M):
            self.conv_output[i,j] = np.sum(X[i]*self.conv[:,j])

    self.hidden_output_1 = self.activation(np.dot(self.conv_output, self.hidden_weight_1))
    self.output = self.activation(np.dot(self.hidden_output_1, self.output_weight))

    return self.output
```

Back propagation :

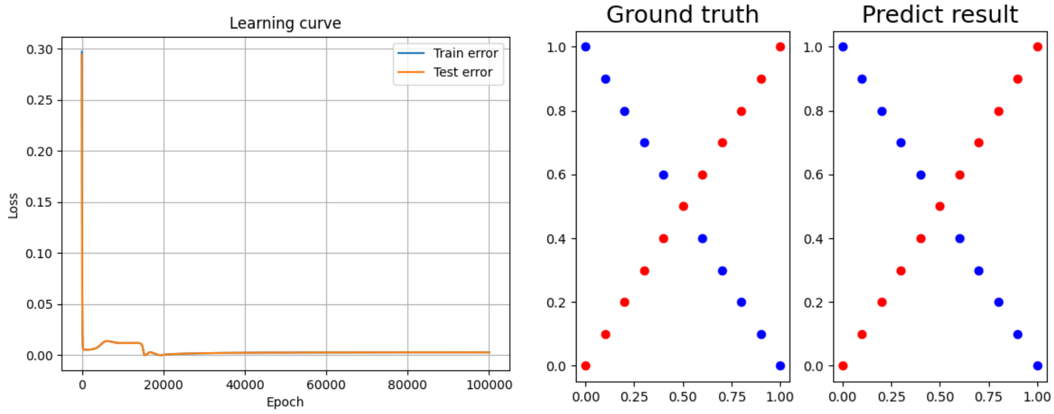
```python
def backward(self, x, y):
    error_output = y - self.output
    if self.mode == 'train':
        d_output = error_output*self.derivative_activation(self.output)

        error_h1 = d_output.dot(self.output_weight.T)
        d_hidden_output_1 = error_h1*self.derivative_activation(self.hidden_output_1)

        error_conv = d_hidden_output_1.dot(self.hidden_weight_1.T)
        d_conv_out = np.zeros((x.shape[0],self.M))
        for i in range(x.shape[0]):
            d_conv_out[i,:] = np.sum(x[i])
        d_conv = error_conv*d_conv_out

        # update weight
        self.output_weight += self.hidden_output_1.T.dot(d_output)*self.lr
        self.hidden_weight_1 += self.conv_output.T.dot(d_hidden_output_1)*self.lr
        self.conv += x.T.dot(d_conv)*self.lr

    return abs(np.mean(error_output))
```

Xor (M=1 , n=3 , lr=0.01 , activation func=sigmoid) :



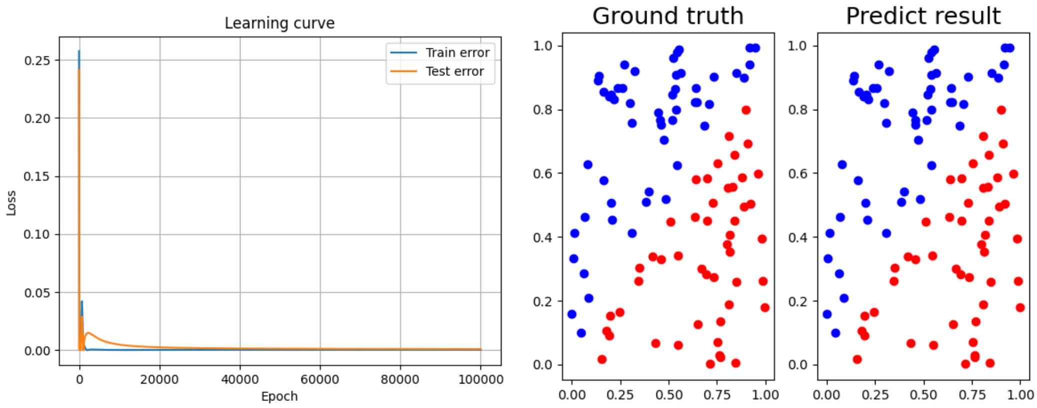

```
Accuracy rate : [ 0 : 100.0 percent , 1 : 0.0 percent]
```

Xor (M=3 , n=3 , lr=0.01 , activation func=sigmoid) :



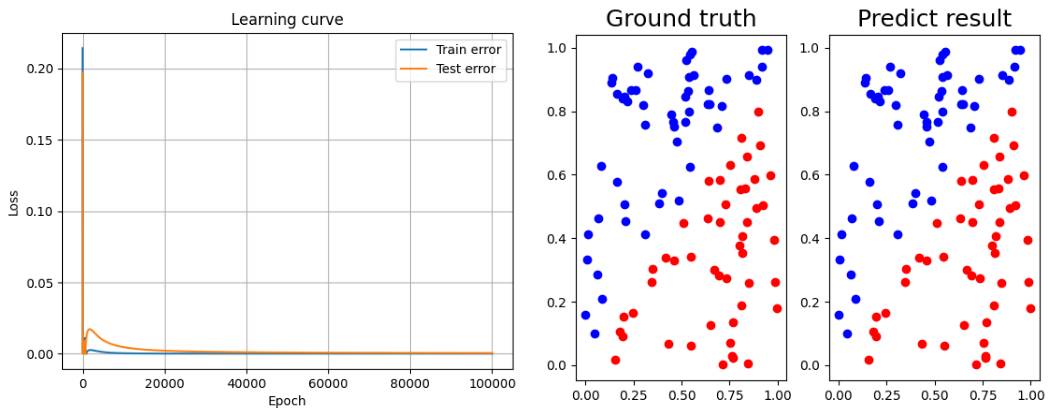
`Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]`

Linear (M=1 , n=3 , lr=0.01 , activation func=sigmoid) :



`Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]`

Linear (M=3 , n=3 , lr=0.01 , activation func=sigmoid) :



`Accuracy rate : [ 0 : 100.0 percent , 1 : 100.0 percent]`