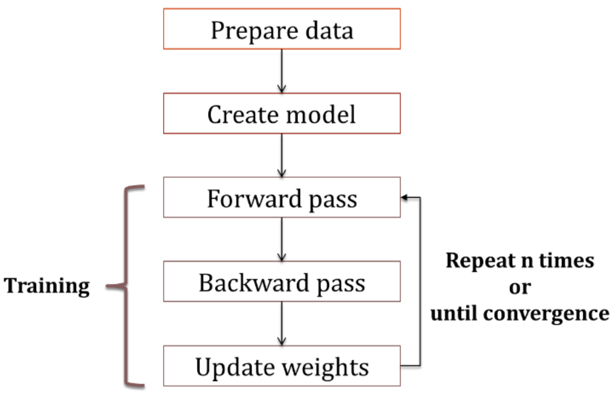
OASIS ML Group TRAINING 03

◆ Back Propagation

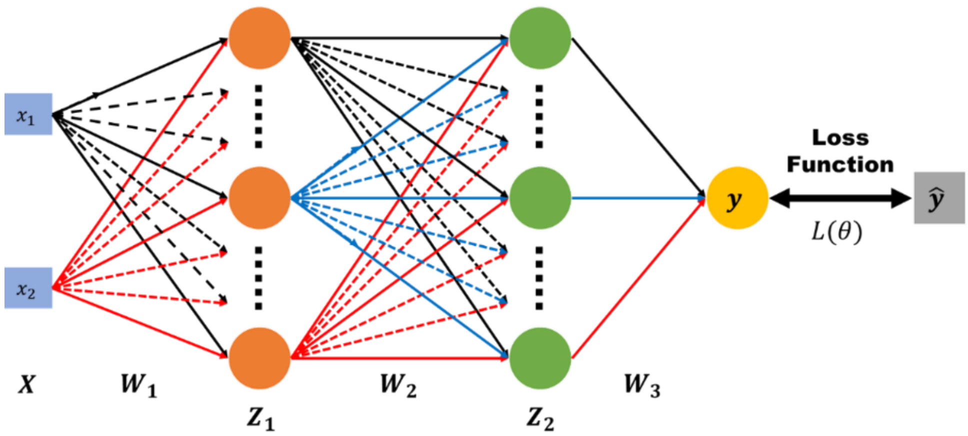
In this practice, you will need to understand and implement a simple neural network with forward and backward pass using two hidden layers. This practice is refer to Lab01 of DLP course at NCTUIOC.

▫ Practice 01：

■ Flow Chart：

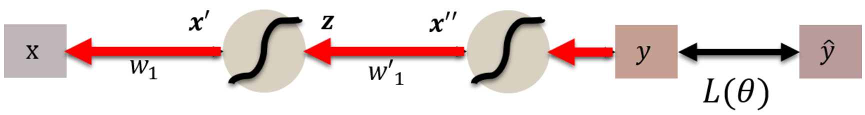


■ Forward propagation：



$Z_1 = \sigma(XW_1)$ $\quad$ $Z_2 = \sigma(Z_1W_2)$ $\quad$ $y = \sigma(Z_2W_3)$

■ Backward propagation：



$x' = xw_1$ $\quad$ $z = \sigma(x')$ $\quad$ $x'' = zw'_1$ $\quad$ $y = \sigma(x'')$

■ Propagation：

Each propagation involves the following steps：

1. Propagation forward through the network to generate the output value
2. Calculation of the cost L(θ) (error term)
3. Propagation of the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

■ Weight update：

For each weight-synapse follow the below steps：

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) of the gradient from the weight.
3. This ratio (percentage) influences the speed and quality of learning; it is called the **learning rate**. The greater the ratio, the faster the neuron trains; the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

Repeat propagation and weight update until the performance of the network is satisfactory.

■ Pseudocode：

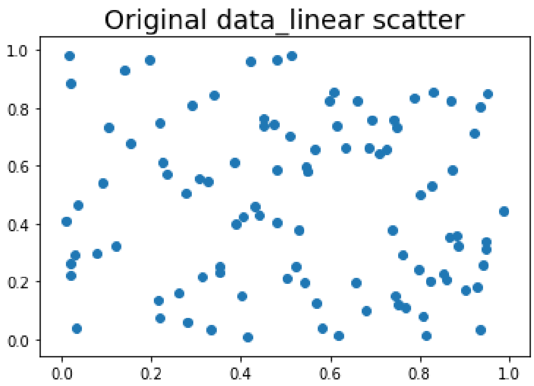
```
initialize network weights (often small random values)
do
   forEach training example named ex
      prediction = neural-net-output(network, ex)  // forward pass
      actual = teacher-output(ex)
      compute error (prediction - actual) at the output units
      compute Δw_h for all weights from hidden layer to output layer  // backward pass
      compute Δw_i for all weights from input layer to hidden layer   // backward pass continued
      update network weights // input layer not modified by error estimate
until all examples classified correctly or another stopping criterion satisfied
return the network
```

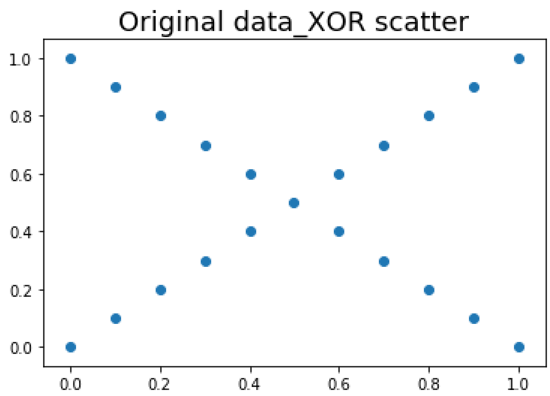
■ Recommend initial parameter：

- Epoch：100000, learning rate：try by yourself
- Activation function：sigmoid function

▪ **Hint 01**：Import library you needed, but do not call API.

▪ **Hint 02**：Generate the dataset showed below.





Data # of linear scatter：100 Data # of XOR scatter：21

(You should try to generate by yourself first, after that, you can use the following generate functions show below to create inputs x, y.)

```python
def generate_linear( ):
    pts = np.random.uniform(0, 1, (n, 2))
    inputs = []
    labels = []
    for pt in pts :
        inputs.append([pt[0], pt[1]])
        distance = (pt[0] - pt[1])/1.414
        if(pt[0] > pt[1]):
            labels.append(0)
        else:
            labels.append(1)
    return np.array(inputs), np.array(labels).reshape(n, 1)
```

```python
def generate_XOR_easy():
    inputs = []
    labels = []

    for i in range(11):
        inputs.append([0.1*i, 0.1*i])
        labels.append(0)

        if 0.1*i == 0.5:
            continue
        inputs.append([0.1*i, (1 - 0.1*i)])
        labels.append(1)
    return np.array(inputs), np.array(labels).reshape(21, 1)
```

▪ **Hint 03**：Define class, function to finish training & testing flow.

For example：

```python
class Neural_Network(object):
    def __init__(self):
    #parameters
        self.learning_rate = ???
        self.inputSize  = 2
        self.hiddenSize = ???
        self.outputSize = 1
    #weights
        self.W1 = np.random.randn(self.inputSize,  self.hiddenSize)
        self.W2 = np.random.randn(self.hiddenSize, self.hiddenSize)
        self.W3 = np.random.randn(self.hiddenSize, self.outputSize)
```

```python
#forward propagation
def forward(self, X):
    ### TODO ###
    return ???

# backward propagation
def backward(self, X, y, o):
    ### TODO ###
    #update eights

def train_flow(self, X, y):
    ### TODO ###
```

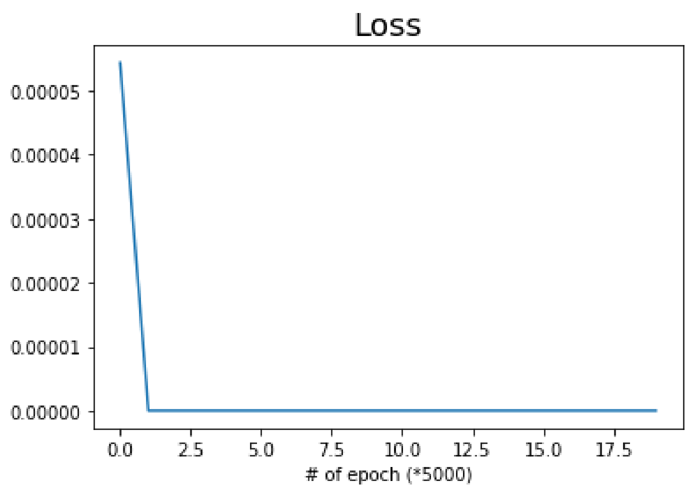
- **Hint 04**：In training, print the **loss values** and the **execution time** as shown below.

```
start training
epoch 0 Loss: 0.30179620862766027
epoch 10000 Loss: 0.0020527808935772974
epoch 15000 Loss: 0.001174151579035179
epoch 20000 Loss: 0.000782217592492378
epoch 25000 Loss: 0.0005687344719030468
epoch 30000 Loss: 0.0004373905450591l965
epoch 35000 Loss: 0.00034984193755013674
epoch 40000 Loss: 0.00028808017980564457
epoch 45000 Loss: 0.0002426203404691321
epoch 50000 Loss: 0.000208033288718607
epoch 55000 Loss: 0.00018100882641505604
epoch 60000 Loss: 0.00015942583006825386
epoch 65000 Loss: 0.0001418692683240151
epoch 70000 Loss: 0.0001273631214461876
epoch 75000 Loss: 0.00011521495044074177
epoch 80000 Loss: 0.00010492150985651163
epoch 85000 Loss: 9.610932494251336e-05
epoch 90000 Loss: 8.849610268256983e-05
epoch 95000 Loss: 8.186498669647466e-05
epoch 100000 Loss: 7.604696865607952e-05
Excution time：19.284762 sec
```

- **Hint 05**：In testing, show the accuracy and result of prediction as shown below.

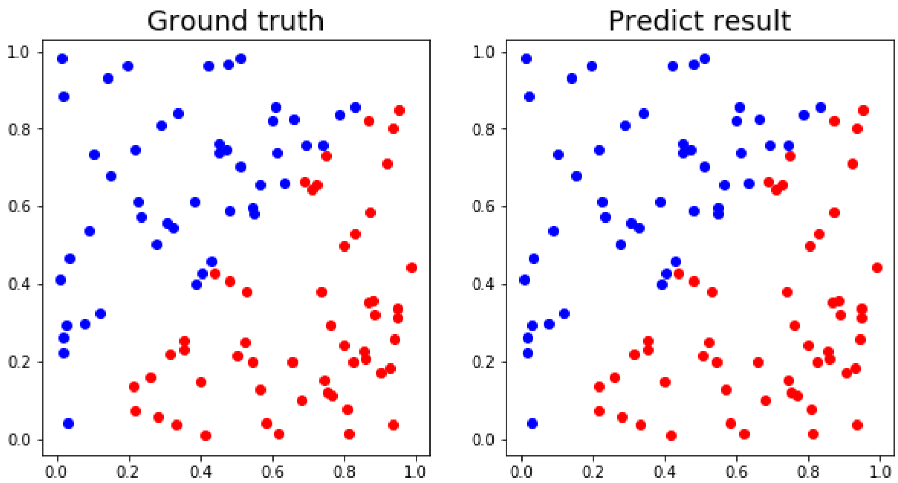
```
**-----Classification Result-----**
PASS : 100 || FAIL : 0
FAIL position : NONE
Accuracy : 1.0
**-------------------------------**
```

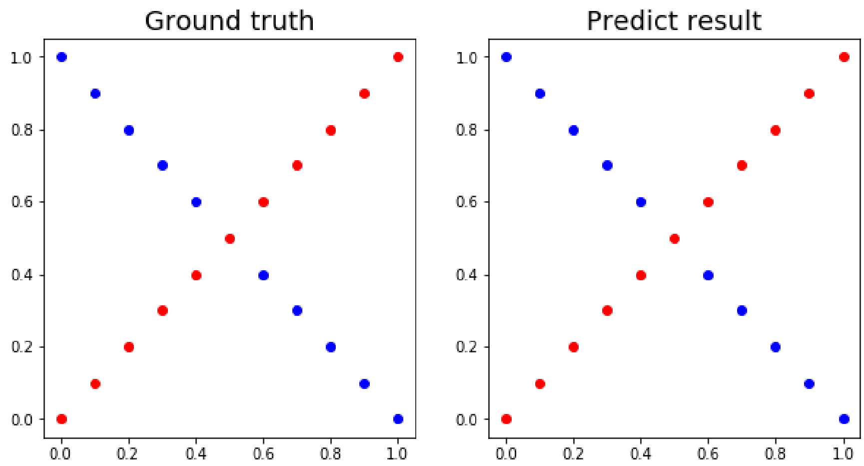
- **Hint 06**：Plot the plot learning curve (loss, epoch) as shown below.



- **Hint 07**：Plot comparison figure showing the predictions and ground truth.

For example：





(You can use the following show_result functions as shown below.)

```python
def show_result(x, y, pred_y):
    plt.figure(figsize=(10,5))
    plt.subplot(1, 2, 1)
    plt.title('Ground truth', fontsize=18)
    for i in range(x.shape[0]):
        if y[i] == 0:
            plt.plot(x[i][0],x[i][1],'ro')
        else :
            plt.plot(x[i][0],x[i][1],'bo')

    plt.subplot(1,2,2)
    plt.title('Predict result', fontsize=18)
    for i in range(x.shape[0]):
        if pred_y[i] == 0:
            plt.plot(x[i][0],x[i][1],'ro')
        else :
            plt.plot(x[i][0],x[i][1],'bo')

    plt.show()
```

Practice 02：

- **Try different activation functions：Tanh, Relu, Leaky Relu**
- **Try different numbers of hidden units**

Discuss the result.

Constraints

You can only use **numpy, matplotlib** and **Python standard libraries**