(1) Introduction

Back-propagation is the most important kernel function in neural network (NN). No matter how complex the NN is, we always can update parameters by the partial differentiation of loss value. However, implementing back-propagation may suffer from the difficult about partial derivative calculation. In Lab1, we will build a two-layer NN with four main functions, that is, ***forward***, ***loss computation***, ***backward***, and ***weights updating***, to show how to implement back-propagation in detail.

The architecture of the two-layer NN is shown in Fig. 1. The notations in Fig. 1 are described in Table 1.

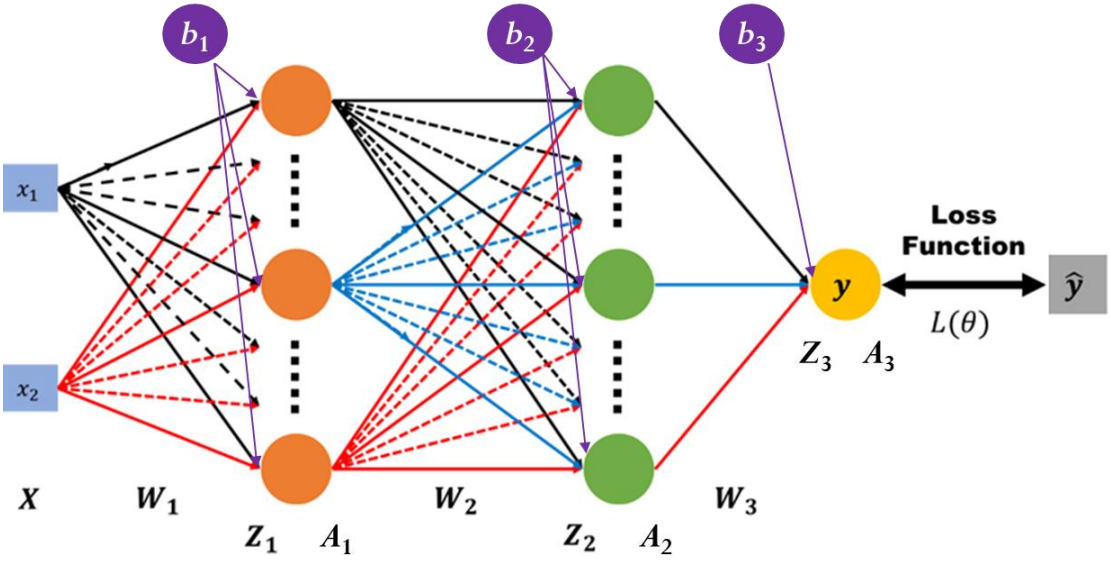


Fig. 1 The architecture of the two-layer NN

Table 1. The notations of NN in Fig. 1.

notation	description
x_1, x_2	neural network inputs
X	$[x_1, x_2]$
y	neural network outputs, y=A_3
$\hat{y}$	ground truth
$L(\theta)$	loss value
W_1, W_2, W_3	weight matrix of each network layers
b_1, b_2, b_3	bias vector of each network layers
Z_1, Z_2, Z_3	$[XW_1 \ \ or \ \ W_1X^T] + b_1, \ [A_1W_2 \ \ or \ \ W_2A_1^T] + b_2,$ $[A_2W_3 \ \ or \ \ W_3A_2^T] + b_3$
A_1, A_2, A_3	$\sigma(Z_1), \sigma(Z_2), \sigma(Z_3), where \ \ \sigma(x) = \frac{1}{1+e^{-x}}$

The flowchart of NN operation in training stage is shown in Fig. 2. After we created the model, we got the random initial weights with normal distribution. Then fixed the weights and calculate the output of NN, that is *Forward pass*. We calculate the errors, or we called loss value, between NN output from *Forward pass* and ground truth in *Loss computation*. Then calculated the partial derivative of weights from loss value (the detail is described in (2)-(c)), e.g., $\nabla L(\theta) = [\frac{\partial L(\theta)}{\partial W_1}, \frac{\partial L(\theta)}{\partial W_2}, \frac{\partial L(\theta)}{\partial W_3}, \dots]$. Finally, we updated weights by gradient descent. It means $\theta^{new} = \theta - \rho \nabla L(\theta)$.

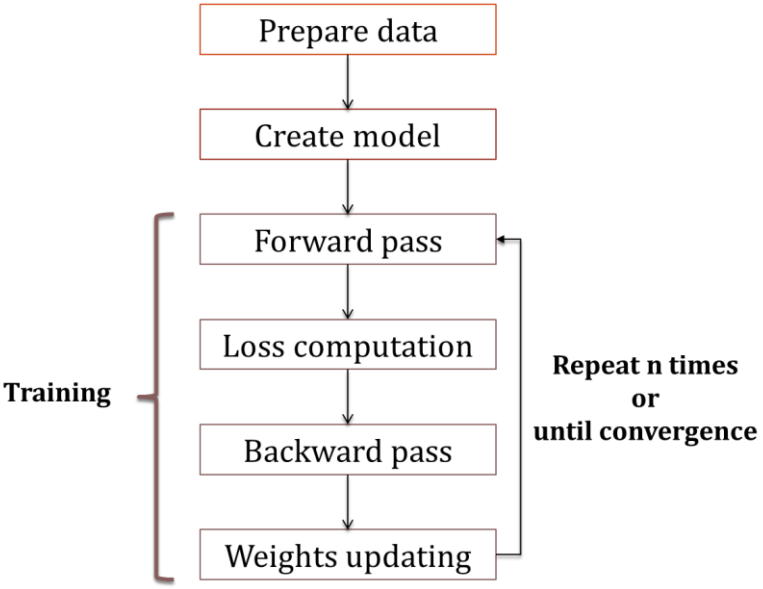


Fig. 2 The flowchart of NN operation in training stage

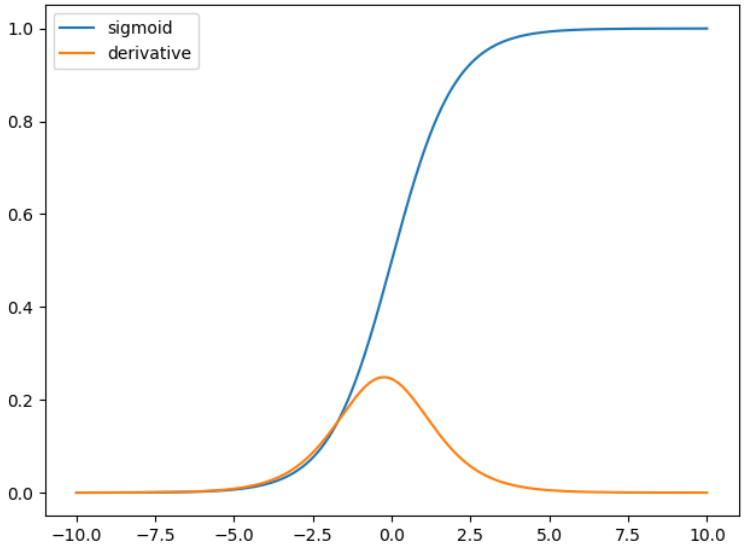
(2) Experimental Setup

(A) Sigmoid functions

The formula of sigmoid function is $\sigma(x) = \frac{1}{1+e^{-x}}$. It can map a real number to (0, 1) space which included the solution space of binary classification. Its partial derivative of x can be described by itself.

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)) \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad (1)$$

Here, we used sigmoid function as our activation function after Z_i, i = 1,2,3. The visualization curve of sigmoid function and its derivative is shown below.



(B) Neural network

The architecture of neural network is shown in Fig. 1. Here, we set the neural numbers of hidden layer 1 and hidden layer 2 be the same, e.g, 20 in linear dataset and 100 in XOR dataset. The shapes of weights are shown in Table 2.

weight	shape
W_1	n_h_1 x 2
b_1	n_h_1x 1
W_2	n_h_2 x n_h_1
b_2	n_h_2 x 1
W_3	1 x n_h_2
b_3	1 x 1

The neural network includes:

(i) Initial function:

We initialized weights with normal distribution and set learning rate = 0.1.

```
# Normal distribution
W1 = np.random.randn(n_h1, n_x)
b1 = np.zeros([n_h1, 1])
W2 = np.random.randn(n_h2, n_h1)
b2 = np.zeros([n_h2, 1])
W3 = np.random.randn(n_y, n_h2)
b3 = np.zeros([n_y, 1])

self.parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2, "W3": W3, "b3": b3}
```

(ii) Forward function:

Calculate NN output.

```
Z1 = np.dot(W1, inputs) + b1
A1 = sigmoid(Z1)  # Sigmoid
Z2 = np.dot(W2, A1) + b2
A2 = sigmoid(Z2)  # Sigmoid
Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)  # Sigmoid

self.cache = {"Z1": Z1, "A1": A1, "Z2": Z2, "A2": A2, "Z3": Z3, "A3": A3}
```

(iii) Backward function:

Calculate gradients $\nabla L(\theta) = \left[\frac{\partial L(\theta)}{\partial W_1}, \frac{\partial L(\theta)}{\partial b_1}, \frac{\partial L(\theta)}{\partial W_2}, \frac{\partial L(\theta)}{\partial b_2}, \frac{\partial L(\theta)}{\partial W_3}, \frac{\partial L(\theta)}{\partial b_3}\right]$.

```
L = self.cost  # L = (Y - A3)
temp_s = sigmoid(Z3)
dZ3 = L * der_sigmoid(temp_s)  # Sigmoid (back propagation)

dW3 = 1 / m * np.dot(dZ3, A2.T)
db3 = 1 / m * np.sum(dZ3, axis=1, keepdims=True)

dA2 = np.dot(W3.T, dZ3)
temp_s = sigmoid(Z2)
dZ2 = dA2 * der_sigmoid(temp_s)  # Sigmoid (back propagation)

dW2 = 1 / m * np.dot(dZ2, A1.T)
db2 = 1 / m * np.sum(dZ2, axis=1, keepdims=True)

dA1 = np.dot(W2.T, dZ2)
temp_s = sigmoid(Z1)
dZ1 = dA1 * der_sigmoid(temp_s)  # Sigmoid (back propagation)

dW1 = 1 / m * np.dot(dZ1, X.T)
db1 = 1 / m * np.sum(dZ1, axis=1, keepdims=True)

grads = {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2, "dW3": dW3, "db3": db3}
```

(iv) Loss computation:

Calculate loss value.

```
def compute_cost(A3, Y):
    cost = (A3 - Y)
    return np.squeeze(cost)
```

(v) Weights updating:

Update weights by gradient descent.

```
def update_parameters(parameters, grads, learning_rate=1.2):
    W1 = W1 - learning_rate * dW1
    b1 = b1 - learning_rate * db1
    W2 = W2 - learning_rate * dW2
    b2 = b2 - learning_rate * db2
    W3 = W3 - learning_rate * dW3
    b3 = b3 - learning_rate * db3

    parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2, "W3": W3, "b3": b3}
    return parameters
```

(vi) Train:

The processing steps in training stage. It responses to the flowchart in Fig. 2.

```
def train(self, inputs, labels):
    for epochs in range(self.num_step):
        for idx in range(n):
            # operation in each training step:
            #   1. forward passing
            #   2. compute loss
            #   3. propagate gradient backward to the front
            #   4. update weights
            self.inputs = inputs[:, idx:idx+1].copy()
            self.labels = labels[:, idx:idx+1].copy()
            A3 = self.forward(inputs[:, idx:idx+1])
            self.cost = compute_cost(A3, labels[:, idx:idx+1])
            grads = self.backward()
            self.parameters = update_parameters(self.parameters, grads,
self.learning_rate)
```

```
        if epochs % self.print_interval == 0:
            print('Epoch {} loss : {}'.format(epochs, abs(self.cost)))
```

(vii) Test:

After getting the training model, just calculate the forward results from input data. Then calculate the accuracy by average of error values.

```
def test(self, inputs, labels):
    n = inputs.shape[1]
    pred_result = np.zeros(n)

    error = 0.0
    for idx in range(n):
        result = self.forward(inputs[:, idx:idx+1])
        pred_result[idx] = result
        error += abs(result - labels[:, idx:idx+1])

    with np.printoptions(precision=8, suppress=True):
        print(np.reshape(pred_result,(n,1)))
    error_avg = error / n
    print('accuracy: %.2f' % ((1 - error_avg) * 100) + '%')
    print('')
```

(C) Back-propagation

Here we use “Mean Square Error” as our objective function.

Therefore the error of *i*-th prediction is $L^{(i)} = \frac{1}{2}\sum_{j=0}^{1}(\hat{y} - y)^2$. The total error is $L = \sum_{i=1}^{n} L^{(i)}, where\ n\ is\ the\ data\ number$. If we can minimize *L* with the specific $W = \{W_1, b_1, W_2, b_2, W_3, b_3\}$, *W* is our solution. Therefore, solve $\frac{\partial L}{\partial W} = 0$ is the simple way to get *W*.

However, it is difficult to calculate unique solution directly while there are many variances in the equation. So, gradient descent is the best way to find optimal solutions.

$$W_k = W_k - \eta \Delta W_k, \quad where\ \eta\ is\ learning\ rate$$
$$b_k = b_k - \eta \Delta b_k, \quad where\ \eta\ is\ learning\ rate$$

If we can calculate ΔW_k and Δb_k, the optimal solutions will be got

after several iterations.

$$\Delta W_k = \frac{\partial L}{\partial W_k} = \frac{\partial \sum_{i=1}^{n} L^{(i)}}{\partial W_k} = \sum_{i=1}^{n} \frac{\partial L^{(i)}}{\partial W_k}$$

$$\Delta b_k = \frac{\partial L}{\partial b_k} = \frac{\partial \sum_{i=1}^{n} L^{(i)}}{\partial b_k} = \sum_{i=1}^{n} \frac{\partial L^{(i)}}{\partial b_k}$$

However, $\frac{\partial L^{(i)}}{\partial W_k}$ and $\frac{\partial L^{(i)}}{\partial b_k}$ cannot be calculated directly, we use chain rule to solve these two formulas.

BP-1	$$\frac{\partial L^{(i)}}{\partial W_3} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_3}$$ $$\frac{\partial L^{(i)}}{\partial b_3} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial b_3}$$ $$\frac{\partial L^{(i)}}{\partial Z_3} = \frac{\partial \frac{1}{2}\sum_{j=0}^{1}(\hat{y}-y)^2}{\partial Z_3} = \frac{\partial \frac{1}{2}(\sigma(Z_3^{(i)}) - y^{(i)})^2}{\partial Z_3}$$ $$= \left(\hat{y}^{(i)} - y^{(i)}\right) \cdot \sigma'(Z_3^{(i)})$$ $$\frac{\partial Z_3}{\partial W_3} = \frac{\partial (W_3 \cdot A_2 + b_3)}{\partial W_3} = A_2$$ $$\frac{\partial Z_3}{\partial b_3} = \frac{\partial (W_3 \cdot A_2 + b_3)}{\partial b_3} = 1$$ $$\frac{\partial L^{(i)}}{\partial W_3} = \left(\hat{y}^{(i)} - y^{(i)}\right) \cdot \sigma'(Z_3^{(i)}) \cdot A_2$$ $$\frac{\partial L^{(i)}}{\partial b_3} = \left(\hat{y}^{(i)} - y^{(i)}\right) \cdot \sigma'(Z_3^{(i)}) \cdot 1$$
BP-2	$$\frac{\partial L^{(i)}}{\partial W_2} = \frac{\partial L^{(i)}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2}$$ $$\frac{\partial L^{(i)}}{\partial b_2} = \frac{\partial L^{(i)}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2}$$ $$\frac{\partial L^{(i)}}{\partial Z_3} = \left(\hat{y}^{(i)} - y^{(i)}\right) \cdot \sigma'(Z_3^{(i)})$$ $$\frac{\partial Z_3}{\partial Z_2} = \frac{\partial (W_3 \cdot A_2 + b_3)}{\partial Z_2} = W_3 \cdot \sigma'(Z_2^{(i)})$$

	$$\frac{\partial Z_2}{\partial W_2} = \frac{\partial (W_2 \cdot A_1 + b_2)}{\partial W_2} = A_1$$ $$\frac{\partial Z_2}{\partial b_2} = \frac{\partial (W_2 \cdot A_1 + b_2)}{\partial b_2} = 1$$ $$\frac{\partial L^{(i)}}{\partial W_2} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot W_3 \cdot \sigma'(Z_2^{(i)}) \cdot A_1$$ $$\frac{\partial L^{(i)}}{\partial b_2} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot W_3 \cdot \sigma'(Z_2^{(i)}) \cdot 1$$
BP-3	$$\frac{\partial L^{(i)}}{\partial W_1} = \frac{\partial L^{(i)}}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1}$$ $$\frac{\partial L^{(i)}}{\partial b_1} = \frac{\partial L^{(i)}}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial b_1} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial b_1}$$ $$\frac{\partial L^{(i)}}{\partial Z_3} = \left(\hat{y}^{(i)} - y^{(i)}\right) \cdot \sigma'(Z_3^{(i)})$$ $$\frac{\partial Z_3}{\partial Z_2} = W_3 \cdot \sigma'(Z_2^{(i)})$$ $$\frac{\partial Z_2}{\partial Z_1} = \frac{\partial (W_2 \cdot A_1 + b_2)}{\partial Z_1} = W_2 \cdot \sigma'(Z_1^{(i)})$$ $$\frac{\partial Z_1}{\partial W_1} = \frac{\partial (W_1 \cdot X + b_1)}{\partial W_1} = \mathrm{X}$$ $$\frac{\partial Z_1}{\partial b_1} = \frac{\partial (W_1 \cdot X + b_1)}{\partial b_1} = 1$$ $$\frac{\partial L^{(i)}}{\partial W_1} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot W_2 \cdot \sigma'(Z_1^{(i)}) \cdot \mathrm{X}$$ $$\frac{\partial L^{(i)}}{\partial b_1} = \frac{\partial L^{(i)}}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \cdot W_2 \cdot \sigma'(Z_1^{(i)}) \cdot 1$$

(3) Experimental Result

(A) Screenshot and comparison figure

(i) <u>Linear</u>

- Data number: 70
- Neural number in both hidden layer: 20
- num_step: 5000

- Learning rate: 0.1
- The loss of each epoch are shown below.

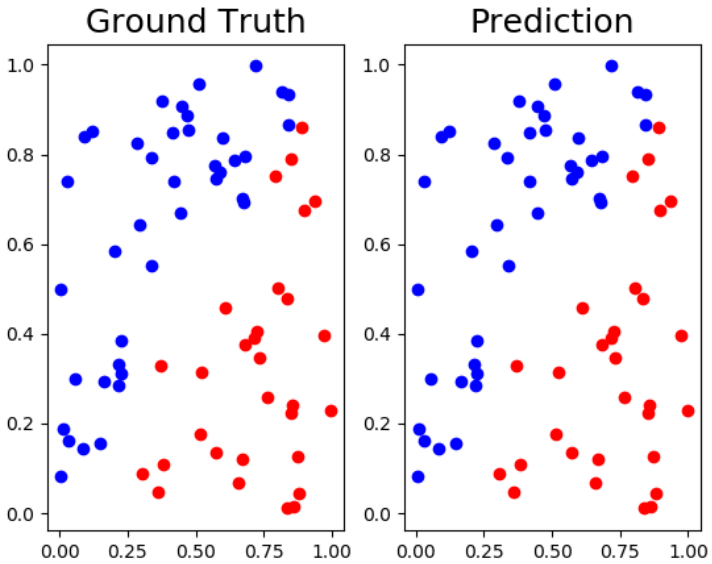
```
Epoch 0 loss : 0.5594048248811484
Epoch 100 loss : 0.14648840074255223
Epoch 200 loss : 0.0912776475669248
Epoch 300 loss : 0.07112411427683785
Epoch 400 loss : 0.05976260360560357
Epoch 500 loss : 0.05209378841589319
Epoch 600 loss : 0.04637417199849525
Epoch 700 loss : 0.04184626643621658
Epoch 800 loss : 0.03812292933642745
Epoch 900 loss : 0.03498229133035179
Epoch 1000 loss : 0.03228589417413705
Epoch 1100 loss : 0.029941233913778474
Epoch 1200 loss : 0.027882915826154972
Epoch 1300 loss : 0.026062511209074306
Epoch 1400 loss : 0.024442795507246762
Epoch 1500 loss : 0.022994311825205987
Epoch 1600 loss : 0.021693222840428964
Epoch 1700 loss : 0.020519905733085025
Epoch 1800 loss : 0.019457993022169013
Epoch 1900 loss : 0.018493691886672967
Epoch 2000 loss : 0.01761528434891612
Epoch 2100 loss : 0.01681274939587131
Epoch 2200 loss : 0.01607747023187139
Epoch 2300 loss : 0.015402002888295985
Epoch 2400 loss : 0.014779890328005368
```

```
Epoch 2500 loss : 0.014205511127594876
Epoch 2600 loss : 0.013673955000195762
Epoch 2700 loss : 0.013180919523557677
Epoch 2800 loss : 0.0127226238663731
Epoch 2900 loss : 0.012295736302685935
Epoch 3000 loss : 0.011897313018343455
Epoch 3100 loss : 0.011524746237672846
Epoch 3200 loss : 0.011175720092083176
Epoch 3300 loss : 0.010848172953616349
Epoch 3400 loss : 0.010540265191175055
Epoch 3500 loss : 0.010250351492630788
Epoch 3600 loss : 0.009976957044343822
Epoch 3700 loss : 0.009718756979420533
Epoch 3800 loss : 0.009474558603553257
Epoch 3900 loss : 0.009243285987195378
Epoch 4000 loss : 0.009023966578653472
Epoch 4100 loss : 0.008815719547150743
Epoch 4200 loss : 0.008617745610162878
Epoch 4300 loss : 0.008429318137034831
Epoch 4400 loss : 0.00824977535240252
Epoch 4500 loss : 0.008078513489352271
Epoch 4600 loss : 0.007914980764434424
Epoch 4700 loss : 0.007758672065325388
Epoch 4800 loss : 0.007609124257689488
Epoch 4900 loss : 0.007465912031115972
```

- The predicted results in testing stage are shown below.

```
[[0.99994298] [0.99997815] [0.99999852]
 [0.00006974] [0.9999791 ] [0.99999947]
 [0.93635817] [0.99999925] [0.97082947]
 [0.99992208] [0.99999194] [0.10886948]
 [0.99999465] [0.00000173] [0.00000179]
 [0.00000069] [0.9999946 ] [0.00921008]
 [0.00001037] [0.0000002 ] [0.00000022]
 [0.99999929] [0.00000015] [0.00000032]
 [0.99999946] [0.9998153 ] [0.90448067]
 [0.00000024] [0.99786534] [0.99999064]
 [0.08032081] [0.99999962] [0.00000132]
 [0.99999929] [0.99999918] [0.99992531]
 [0.99997547] [0.00000026] [0.00000013]
 [0.99999881] [0.99936095] [0.99999912]
 [0.99999926] [0.00000014] [0.99997158]
 [0.99988605] [0.00000113] [0.99999885]
 [0.99878411] [0.99924072] [0.00000021]
 [0.00000047] [0.99999685] [0.0000016 ]
 [0.99999962] [0.00000139] [0.00001225]
 [0.00000489] [0.99986334] [0.99999959]
 [0.00000295]
 [0.00000092]
 [0.99999322]
 [0.99999817]
 [0.04106662]
 [0.00000015]
 [0.92087254]
 [0.99958209]
 [0.00000014]
 [0.00000751]]
accuracy: 99.27%
```

– The visualization of the predictions and ground truth at the end of the training process.



(ii) XOR

– Data number: 69
– Neural number in both hidden layer: 100
– num_step: 7000
– Learning rate: 0.1
– The loss of each epoch are shown below.

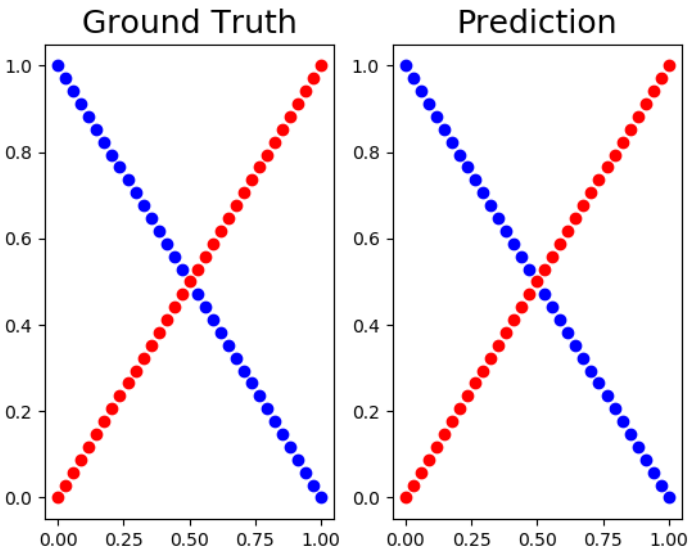
```
Epoch 0 loss : 0.49273293895353926
Epoch 100 loss : 0.5072425016251048
Epoch 200 loss : 0.5072394206004984
Epoch 300 loss : 0.5072210623147878
Epoch 400 loss : 0.5072137669375383
Epoch 500 loss : 0.5071890075286833
Epoch 600 loss : 0.4927059261455068
Epoch 700 loss : 0.4926621631875011
Epoch 800 loss : 0.506130306853991
Epoch 900 loss : 0.49266445246096463
Epoch 1000 loss : 0.49255599498471087
Epoch 1100 loss : 0.4896987794105178
Epoch 1200 loss : 0.11551978162019252
Epoch 1300 loss : 0.08120485398371925
Epoch 1400 loss : 0.11374933892784055
Epoch 1500 loss : 0.09675535379229114
Epoch 1600 loss : 0.07040399294466561
Epoch 1700 loss : 0.07177812425722684
Epoch 1800 loss : 0.05515717174197685
Epoch 1900 loss : 0.045509240721012305
Epoch 2000 loss : 0.044911937645845855
Epoch 2100 loss : 0.042801370144916716
Epoch 2200 loss : 0.03952405597679824
Epoch 2300 loss : 0.0381799066224073
Epoch 2400 loss : 0.04003612164807095
Epoch 2500 loss : 0.041764352137333406
Epoch 2600 loss : 0.033245346520242226
Epoch 2700 loss : 0.03548912623254518
Epoch 2800 loss : 0.03218031392806329
Epoch 2900 loss : 0.03403412631303748
Epoch 3000 loss : 0.02982130671871064
Epoch 3000 loss : 0.02982130671871064
Epoch 3100 loss : 0.027866455386495303
Epoch 3200 loss : 0.02701812148422447
Epoch 3300 loss : 0.026050561673924083
Epoch 3400 loss : 0.025312281177587152
Epoch 3500 loss : 0.02439501193060794
Epoch 3600 loss : 0.023455732050934583
Epoch 3700 loss : 0.022491043970411576
Epoch 3800 loss : 0.021509248614783923
Epoch 3900 loss : 0.020569741376858842
Epoch 4000 loss : 0.019915187680552095
Epoch 4100 loss : 0.020028604505972648
Epoch 4200 loss : 0.020184476360601347
Epoch 4300 loss : 0.019594791765862065
Epoch 4400 loss : 0.018971036142197955
Epoch 4500 loss : 0.018376193938682815
Epoch 4600 loss : 0.01781067854425657
Epoch 4700 loss : 0.017273722931411266
Epoch 4800 loss : 0.01676410475882142
Epoch 4900 loss : 0.016280438572495885
Epoch 5000 loss : 0.01582130281352947
Epoch 5100 loss : 0.015385297137290635
Epoch 5200 loss : 0.014971069187947803
Epoch 5300 loss : 0.014577327180546988
Epoch 5400 loss : 0.014202845438870551
Epoch 5500 loss : 0.013846466196683085
Epoch 5600 loss : 0.0135070992942155
Epoch 5700 loss : 0.013183720633124991
Epoch 5800 loss : 0.012875369880786785
Epoch 5900 loss : 0.012581147723010024
Epoch 6000 loss : 0.012300212858601731
Epoch 6100 loss : 0.012031778866630403
Epoch 6200 loss : 0.011775111037512229
Epoch 6300 loss : 0.01152952323222067
Epoch 6400 loss : 0.01129437481493013
Epoch 6500 loss : 0.011069067690506314
Epoch 6600 loss : 0.01085304346787708
Epoch 6700 loss : 0.010645780762491317
Epoch 6800 loss : 0.010446792645161346
Epoch 6900 loss : 0.01025562424014188
```

- The predicted results in testing stage are shown below.

```
[[0.00248542]  [0.00213091]  [0.99996227]
 [0.99999607]  [0.99999677]  [0.01125579]
 [0.0021915 ]  [0.00297389]  [0.99998443]
 [0.99999667]  [0.99999529]  [0.00499253]
 [0.00195431]  [0.00485275]  [0.99998849]
 [0.99999715]  [0.99999165]  [0.00231031]
 [0.00176634]  [0.00943869]  [0.99998961]
 [0.9999975 ]  [0.99997999]  [0.00115176]
 [0.00162274]  [0.02101914]  [0.99998991]
 [0.99999775]  [0.99992161]  [0.00062464]
 [0.00152181]  [0.04592095]  [0.99998989]
 [0.99999792]  [0.99914567]  [0.00036734]
 [0.00146603]  [0.07910102]  [0.99998971]
 [0.99999799]  [0.88081427]  [0.00023214]
 [0.00146433]  [0.09474627]  [0.99998945]
 [0.99999796]  [0.07929689]  [0.00015602]
 [0.00153709]  [0.87881056]  [0.99998913]
 [0.99999781]  [0.04933065]  [0.00011043]
 [0.00172757]  [0.99951569]  [0.99998878]
 [0.99999747]  [0.02488667]  [0.00008161]
```

```
 [0.99998839]
 [0.00006255]
 [0.99998799]
 [0.00004943]
 [0.99998756]
 [0.00004009]
 [0.99998711]
 [0.00003324]
 [0.99998665]]
accuracy: 98.99%
```

– The visualization of the predictions and ground truth at the end of the training process.

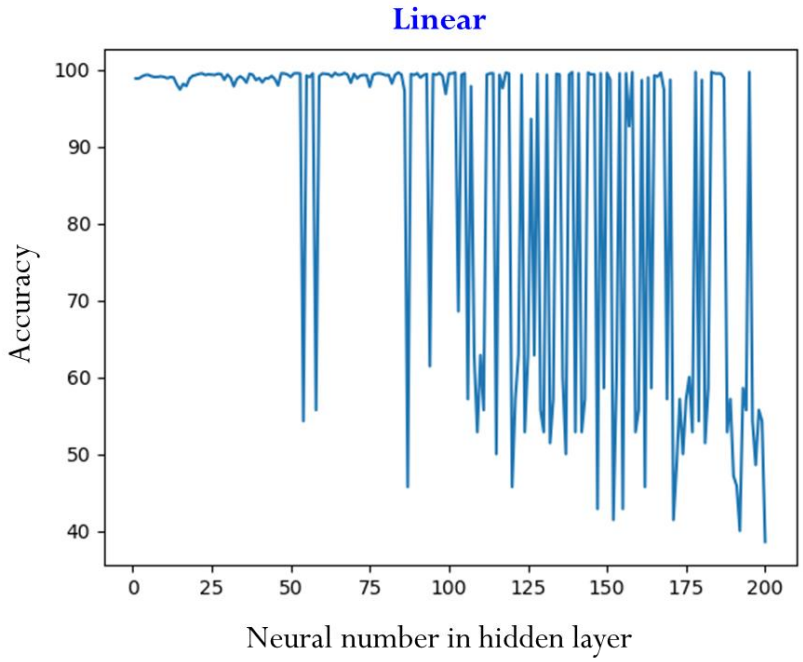


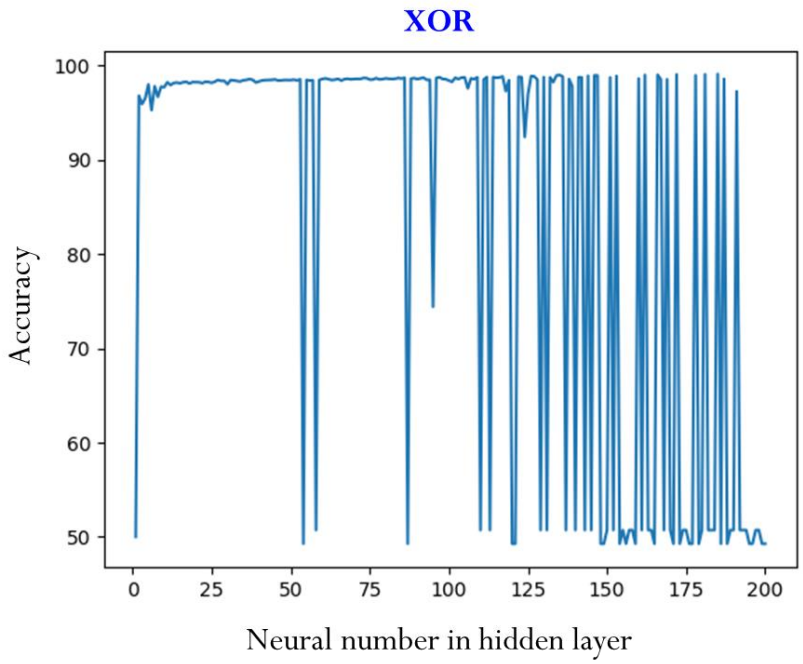
(B) Anything you want to share

(i) Neural numbers in hidden layer

How to choose the best numbers in hidden layer may be problems. Therefore, I changed neural numbers from 1 to 200 while fixing learning rate (0.1) and num_step (5000) to

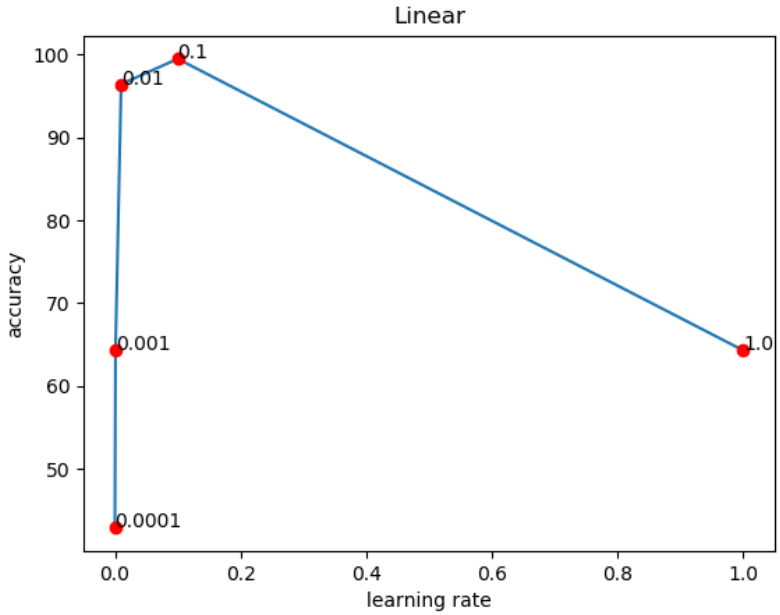
observe the phenomenon.

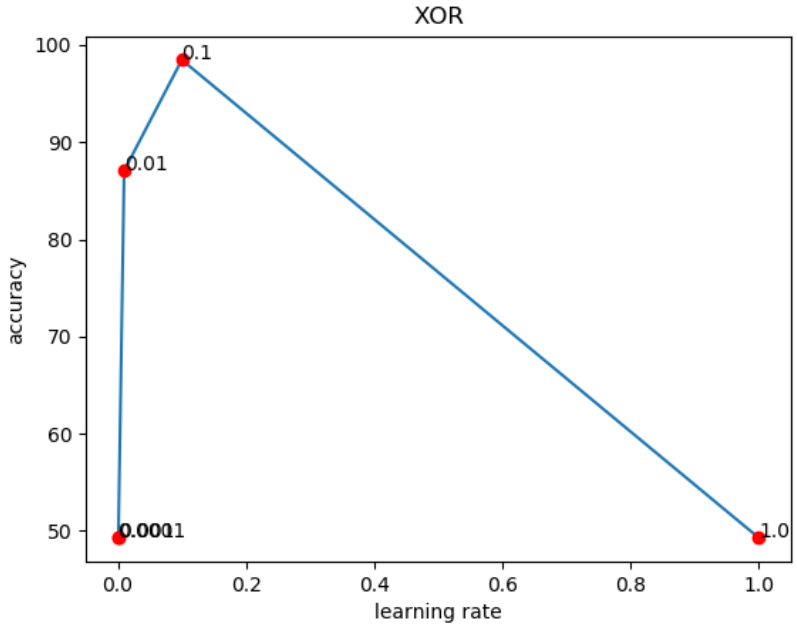




(ii) Learning rate

Different learning rate will cause different results. Here, we fixed neural numbers (50) in hidden layer and num_step (5000) then change learning rate by 0.0001, 0.001, 0.01, 0.1, and 1.0.





(4) Discussion and extra experiments

We also try Xavier Initialization, however, the result is not always better than random initialization with normal distribution.

	Linear	XOR
Normal distribution	99.25 %	98.46 %
Xavier	99.56 %	97.56 %