Machine Learning with NumPy

Systems Programming

Department of Computer Science and Engineering

Sogang University

Machine Learning Terms

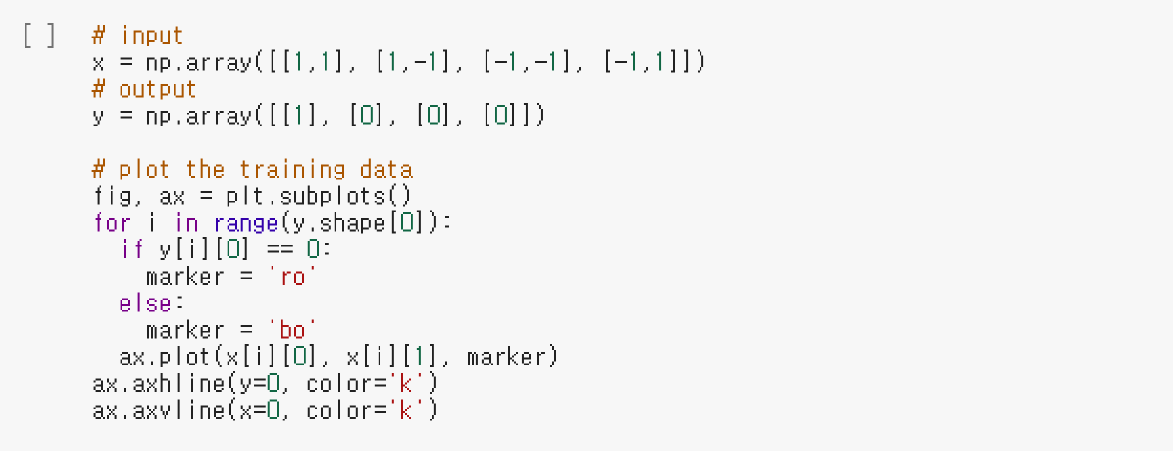
1 Machine learning terms

- Model: The process of training an ML model involves providing an ML algorithm (that is, the *learning algorithm*) with training data to learn from. The term *ML model* refers to the model artifact that is created by the training process.
- Epoch: An epoch is a single step in training a neural network; in other words when a neural network is trained on every training samples only in one pass we say that one epoch is finished.
- Learning rate: The learning rate is how quickly a network abandons old beliefs for new ones.
- Training dataset: A training dataset is a dataset of examples used for learning.
- Test dataset: A test dataset is a dataset that is independent of the training dataset.

AND/OR Problem

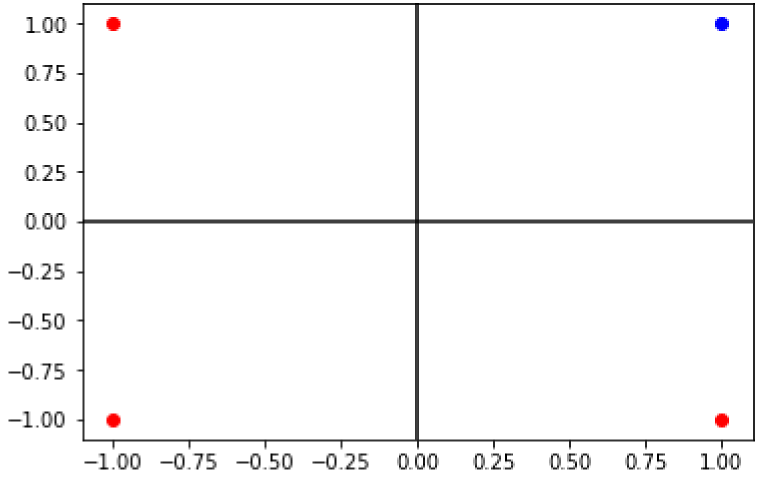
Use “01.AND_OR.ipynb”

1 AND Problem

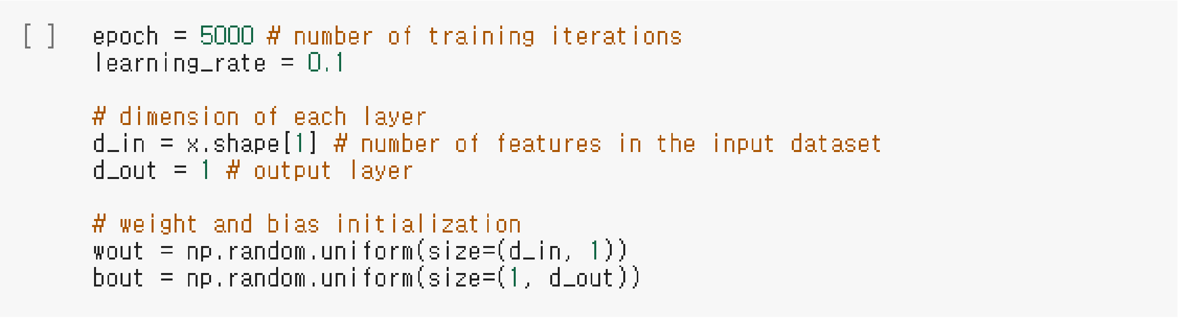


```
# input
x = np.array([[1,1], [1,-1], [-1,-1], [-1,1]])
# output
y = np.array([[1], [0], [0], [0]])

# plot the training data
fig, ax = plt.subplots()
for i in range(y.shape[0]):
  if y[i][0] == 0:
    marker = 'ro'
  else:
    marker = 'bo'
  ax.plot(x[i][0], x[i][1], marker)
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
```

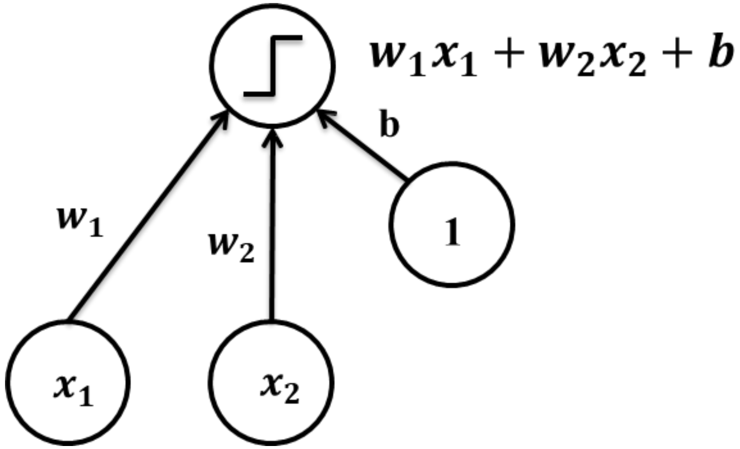
```
<matplotlib.lines.Line2D at 0x7f85bb925d30>
```



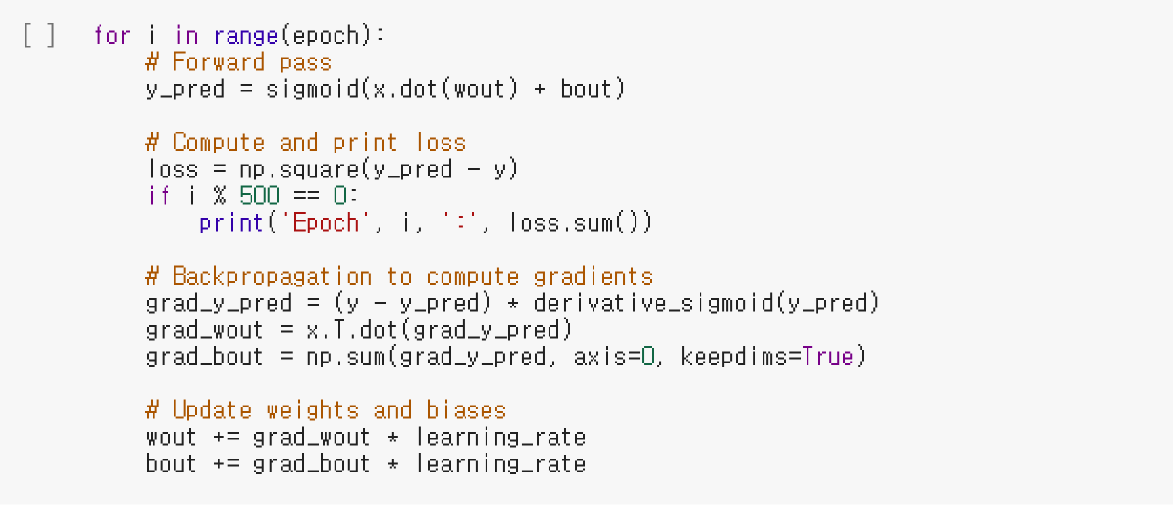
1 AND Problem



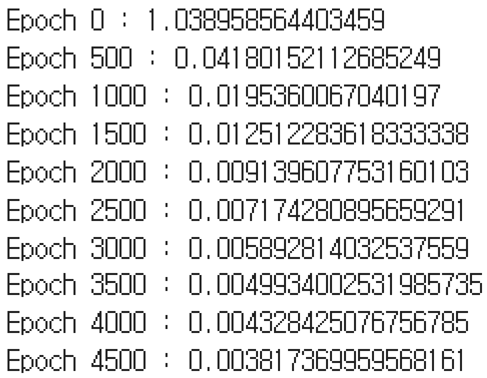
```
[ ]  epoch = 5000 # number of training iterations
     learning_rate = 0.1

     # dimension of each layer
     d_in = x.shape[1] # number of features in the input dataset
     d_out = 1 # output layer

     # weight and bias initialization
     wout = np.random.uniform(size=(d_in, 1))
     bout = np.random.uniform(size=(1, d_out))
```



1 AND Problem

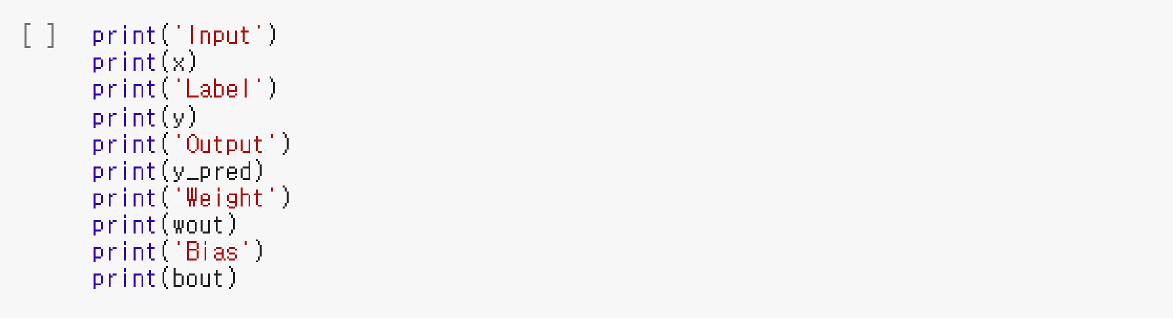


```
[ ]  for i in range(epoch):
         # Forward pass
         y_pred = sigmoid(x.dot(wout) + bout)

         # Compute and print loss
         loss = np.square(y_pred - y)
         if i % 500 == 0:
             print('Epoch', i, ':', loss.sum())

         # Backpropagation to compute gradients
         grad_y_pred = (y - y_pred) * derivative_sigmoid(y_pred)
         grad_wout = x.T.dot(grad_y_pred)
         grad_bout = np.sum(grad_y_pred, axis=0, keepdims=True)

         # Update weights and biases
         wout += grad_wout * learning_rate
         bout += grad_bout * learning_rate
```

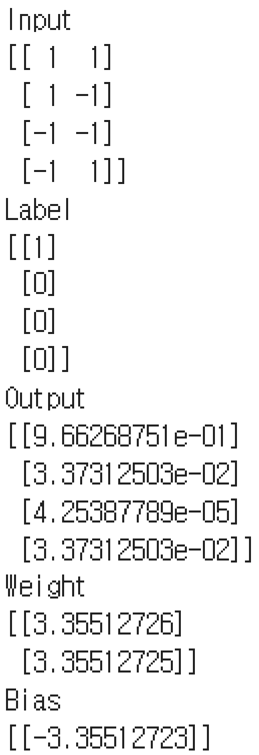


```
Epoch 0 : 1.038958564403459
Epoch 500 : 0.04180152112685249
Epoch 1000 : 0.019536006704019 7
Epoch 1500 : 0.012512283618333338
Epoch 2000 : 0.009139607753160103
Epoch 2500 : 0.007174280895659291
Epoch 3000 : 0.005892814032537559
Epoch 3500 : 0.004993400253198573 5
Epoch 4000 : 0.004328425076756785
Epoch 4500 : 0.003817369959568161
```

1 AND Problem

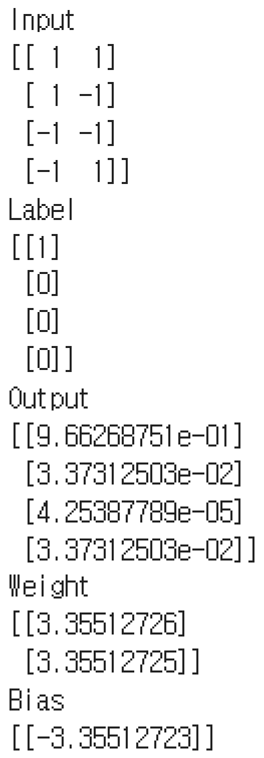


```
[ ]  print('Input')
     print(x)
     print('Label')
     print(y)
     print('Output')
     print(y_pred)
     print('Weight')
     print(wout)
     print('Bias')
     print(bout)
```



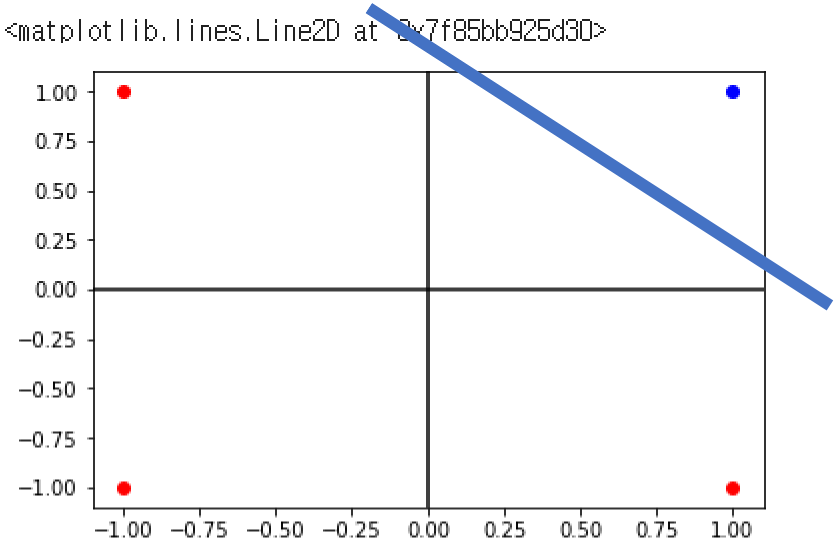
```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[1]
 [0]
 [0]
 [0]]
Output
[[9.66268751e-01]
 [3.37312503e-02]
 [4.25387789e-05]
 [3.37312503e-02]]
Weight
[[3.35512726]
 [3.35512725]]
Bias
[[-3.35512723]]
```

1 AND Problem



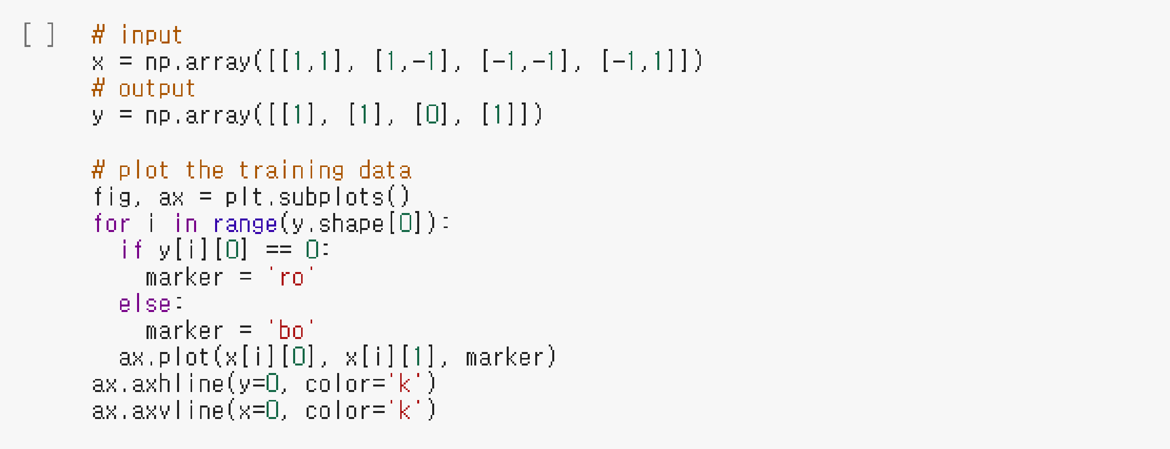
```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[1]
 [0]
 [0]
 [0]]
Output
[[9.66268751e-01]
 [3.37312503e-02]
 [4.25387789e-05]
 [3.37312503e-02]]
Weight
[[3.35512726]
 [3.35512725]]
Bias
[[-3.35512723]]
```

$$f(x_1, x_2) = x_1 + x_2 - 1$$



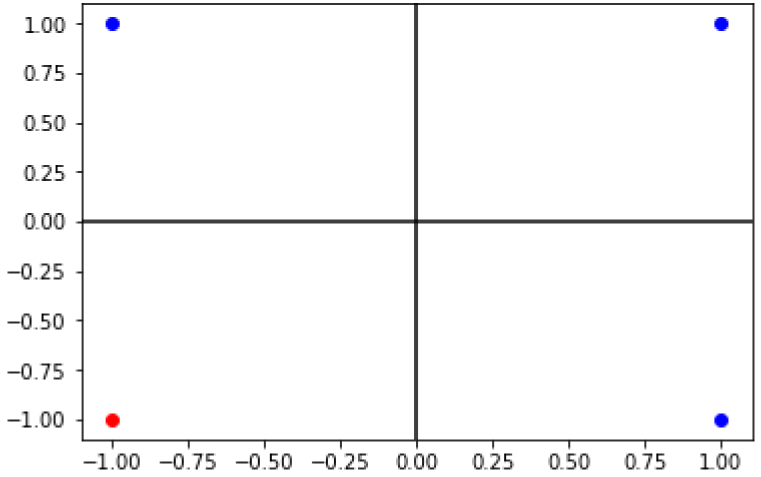
```
<matplotlib.lines.Line2D at 0x7f85bb925d30>
```

2 OR Problem

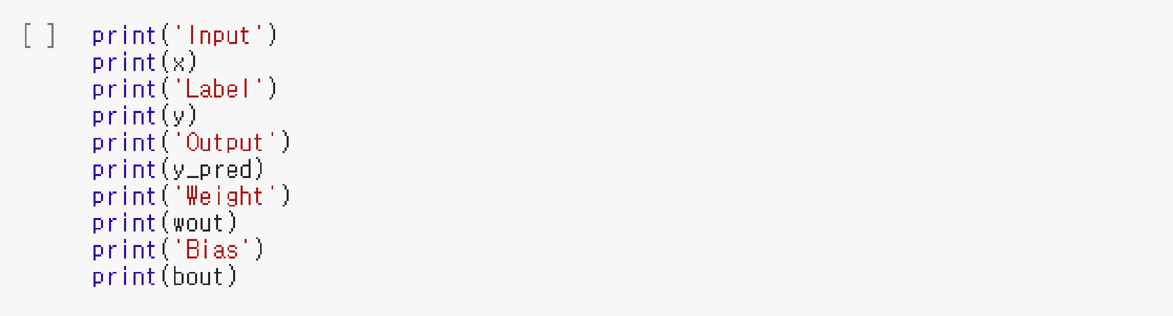


```
# input
x = np.array([[1,1], [1,-1], [-1,-1], [-1,1]])
# output
y = np.array([[1], [1], [0], [1]])

# plot the training data
fig, ax = plt.subplots()
for i in range(y.shape[0]):
  if y[i][0] == 0:
    marker = 'ro'
  else:
    marker = 'bo'
  ax.plot(x[i][0], x[i][1], marker)
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
```

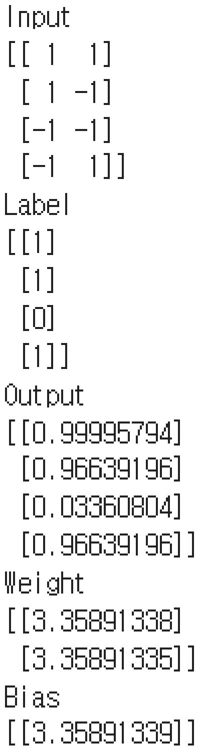
```
<matplotlib.lines.Line2D at 0x7f85bb89eac8>
```



2 OR Problem

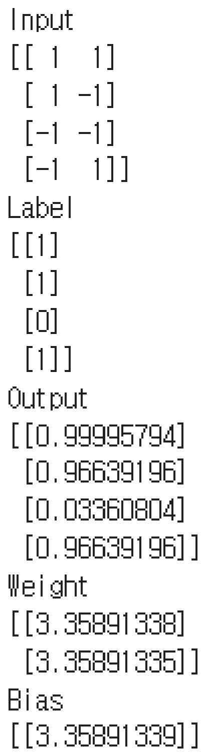


```
[ ]  print('Input')
     print(x)
     print('Label')
     print(y)
     print('Output')
     print(y_pred)
     print('Weight')
     print(wout)
     print('Bias')
     print(bout)
```



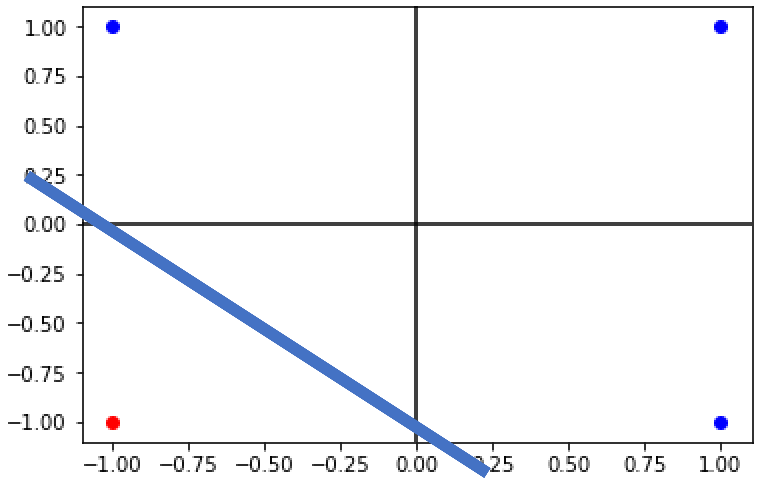
```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[1]
 [1]
 [0]
 [1]]
Output
[[0.99995794]
 [0.96639196]
 [0.03360804]
 [0.96639196]]
Weight
[[3.35891338]
 [3.35891335]]
Bias
[[3.35891339]]
```

2 AND Problem



```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[1]
 [1]
 [0]
 [1]]
Output
[[0.99995794]
 [0.96639196]
 [0.03360804]
 [0.96639196]]
Weight
[[3.35891338]
 [3.35891335]]
Bias
[[3.35891339]]
```

```
<matplotlib.lines.Line2D at 0x7f85bb89eac8>
```

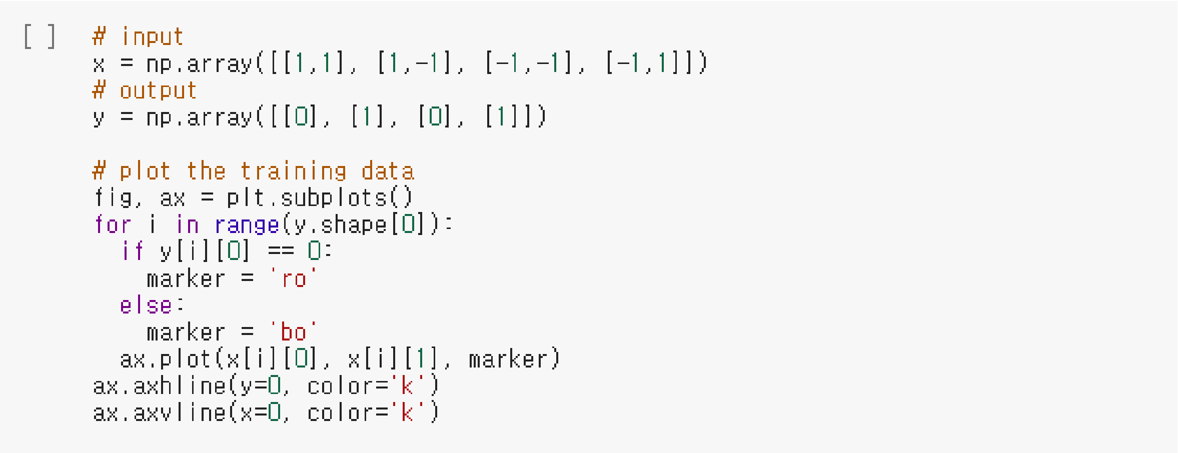


$$f(x_1, x_2) = x_1 + x_2 + 1$$

XOR Problem

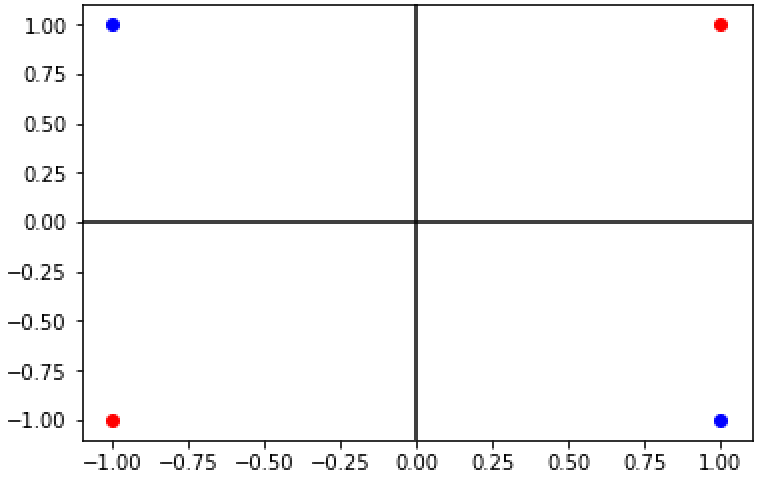
Use “02.XOR.ipynb”

1 XOR Problem

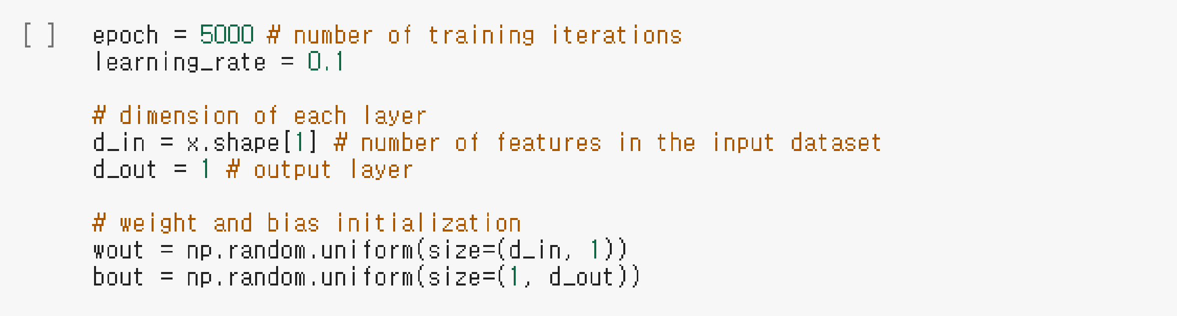


```
[ ]  # input
     x = np.array([[1,1], [1,-1], [-1,-1], [-1,1]])
     # output
     y = np.array([[0], [1], [0], [1]])

     # plot the training data
     fig, ax = plt.subplots()
     for i in range(y.shape[0]):
       if y[i][0] == 0:
         marker = 'ro'
       else:
         marker = 'bo'
       ax.plot(x[i][0], x[i][1], marker)
     ax.axhline(y=0, color='k')
     ax.axvline(x=0, color='k')
```

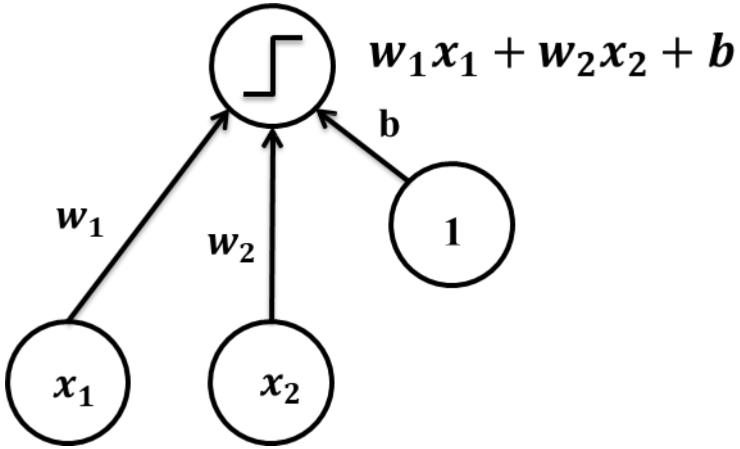
```
<matplotlib.lines.Line2D at 0x7fec7e15e320>
```



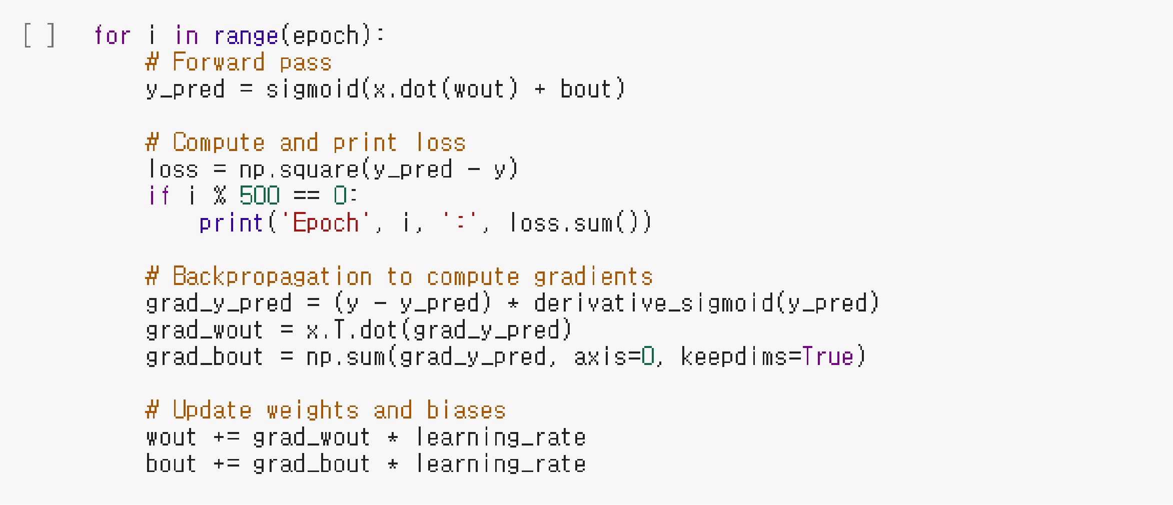
2 XOR Problem with Single layer perceptron



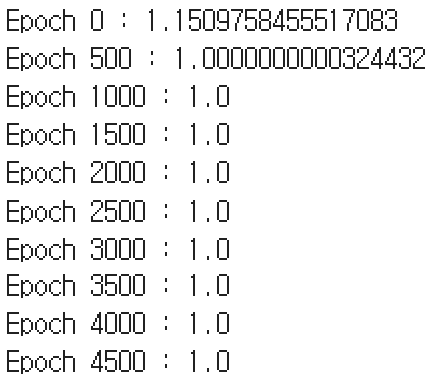
```
[ ]  epoch = 5000 # number of training iterations
     learning_rate = 0.1

     # dimension of each layer
     d_in = x.shape[1] # number of features in the input dataset
     d_out = 1 # output layer

     # weight and bias initialization
     wout = np.random.uniform(size=(d_in, 1))
     bout = np.random.uniform(size=(1, d_out))
```



2 XOR Problem with Single layer perceptron

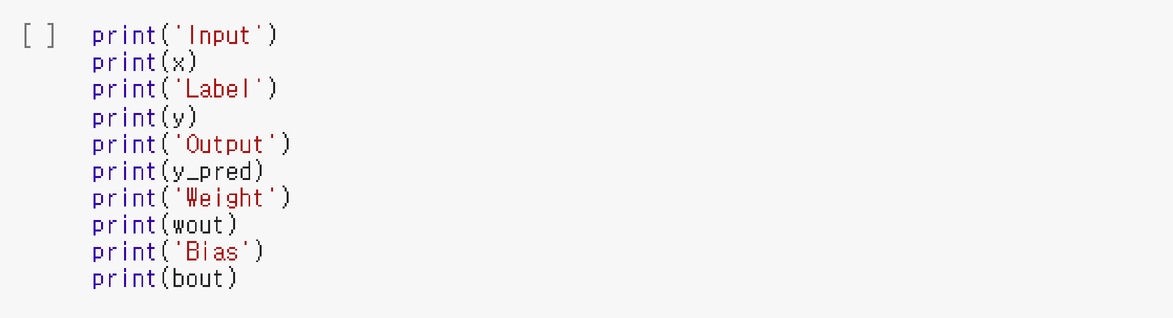


```
for i in range(epoch):
    # Forward pass
    y_pred = sigmoid(x.dot(wout) + bout)

    # Compute and print loss
    loss = np.square(y_pred - y)
    if i % 500 == 0:
        print('Epoch', i, ':', loss.sum())

    # Backpropagation to compute gradients
    grad_y_pred = (y - y_pred) * derivative_sigmoid(y_pred)
    grad_wout = x.T.dot(grad_y_pred)
    grad_bout = np.sum(grad_y_pred, axis=0, keepdims=True)

    # Update weights and biases
    wout += grad_wout * learning_rate
    bout += grad_bout * learning_rate
```

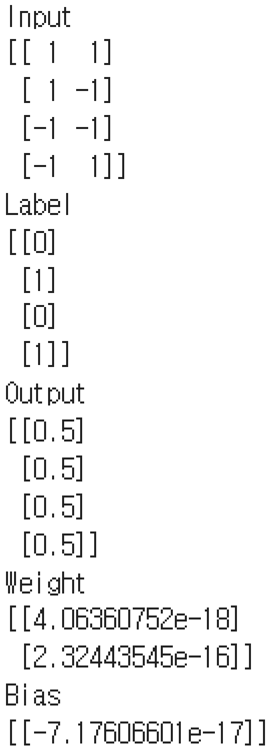


```
Epoch 0 : 1.1509758455517083
Epoch 500 : 1.0000000000324432
Epoch 1000 : 1.0
Epoch 1500 : 1.0
Epoch 2000 : 1.0
Epoch 2500 : 1.0
Epoch 3000 : 1.0
Epoch 3500 : 1.0
Epoch 4000 : 1.0
Epoch 4500 : 1.0
```

2 XOR Problem with Single layer perceptron

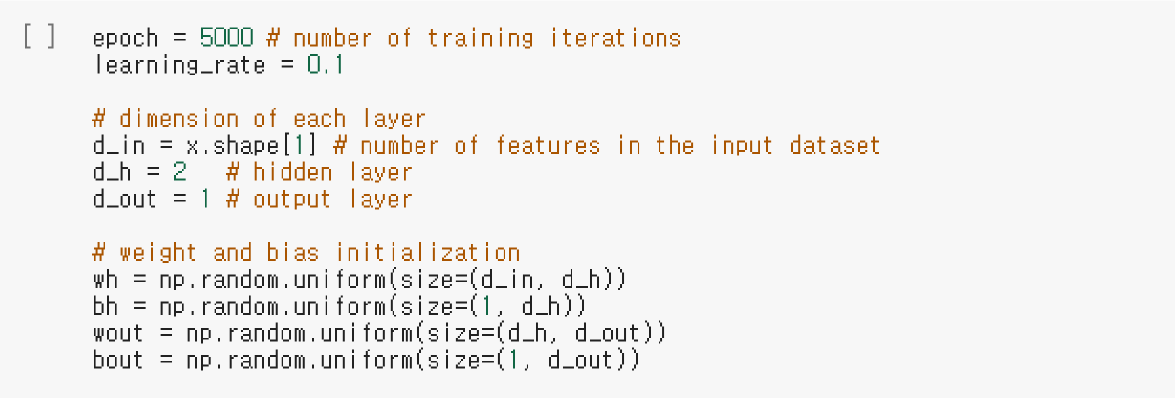


```
[ ]  print('Input')
     print(x)
     print('Label')
     print(y)
     print('Output')
     print(y_pred)
     print('Weight')
     print(wout)
     print('Bias')
     print(bout)
```

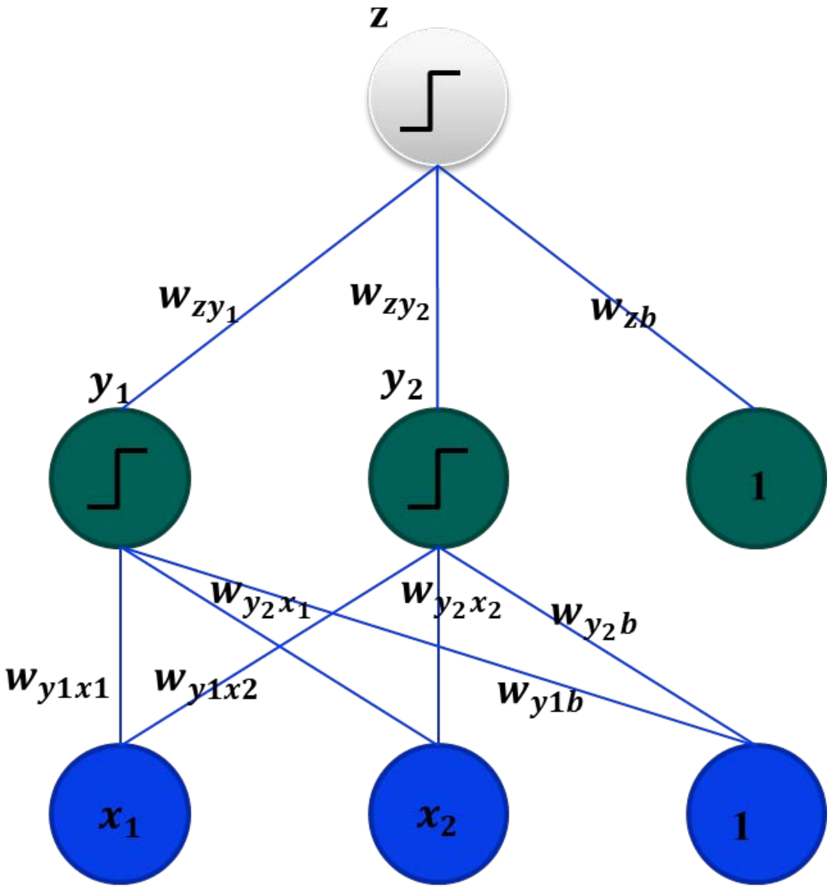


```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[0]
 [1]
 [0]
 [1]]
Output
[[0.5]
 [0.5]
 [0.5]
 [0.5]]
Weight
[[4.06360752e-18]
 [2.32443545e-16]]
Bias
[[-7.17606601e-17]]
```

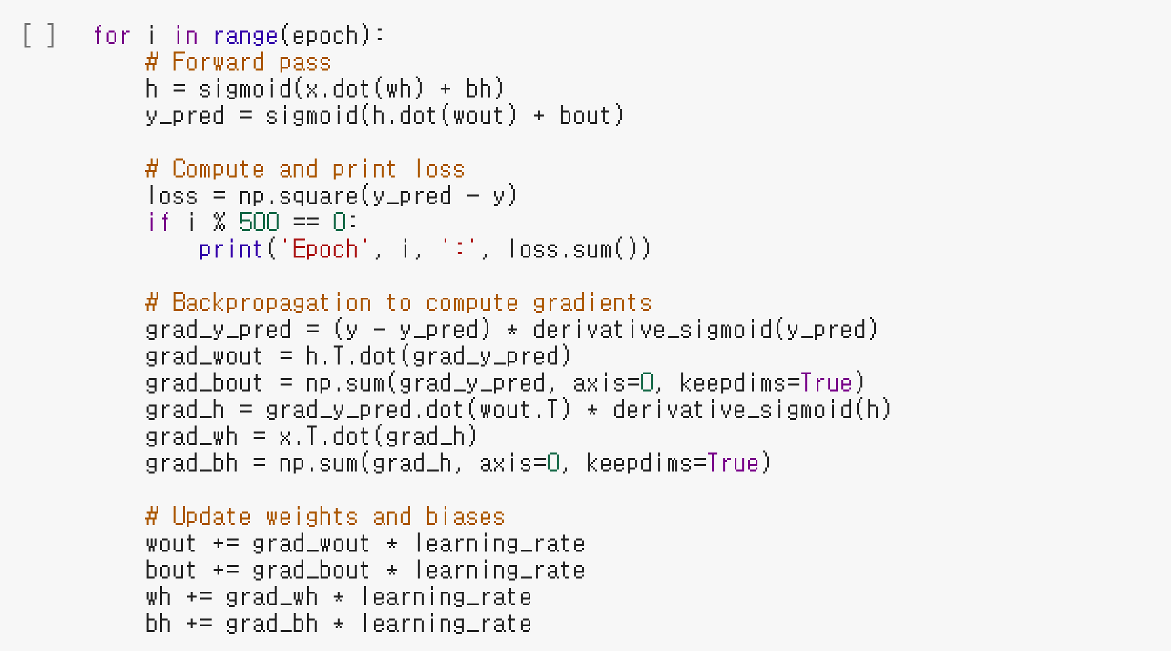
3 XOR Problem with Multi layer perceptron



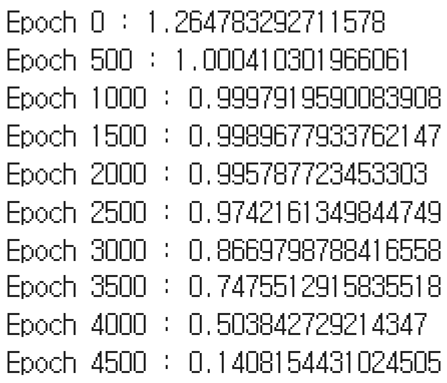
```
[ ]  epoch = 5000 # number of training iterations
     learning_rate = 0.1

     # dimension of each layer
     d_in = x.shape[1] # number of features in the input dataset
     d_h = 2   # hidden layer
     d_out = 1 # output layer

     # weight and bias initialization
     wh = np.random.uniform(size=(d_in, d_h))
     bh = np.random.uniform(size=(1, d_h))
     wout = np.random.uniform(size=(d_h, d_out))
     bout = np.random.uniform(size=(1, d_out))
```



3 XOR Problem with Multi layer perceptron

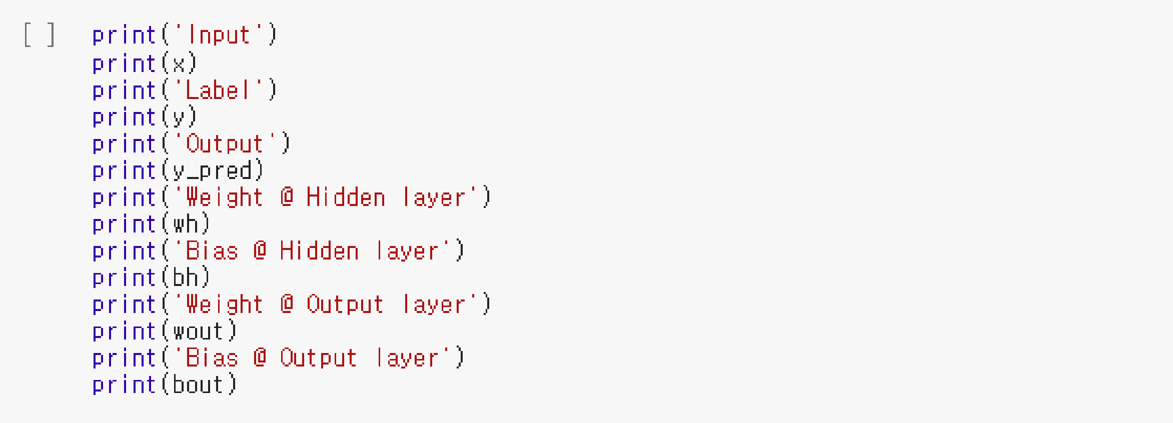


```
[ ]  for i in range(epoch):
         # Forward pass
         h = sigmoid(x.dot(wh) + bh)
         y_pred = sigmoid(h.dot(wout) + bout)

         # Compute and print loss
         loss = np.square(y_pred - y)
         if i % 500 == 0:
             print('Epoch', i, ':', loss.sum())

         # Backpropagation to compute gradients
         grad_y_pred = (y - y_pred) * derivative_sigmoid(y_pred)
         grad_wout = h.T.dot(grad_y_pred)
         grad_bout = np.sum(grad_y_pred, axis=0, keepdims=True)
         grad_h = grad_y_pred.dot(wout.T) * derivative_sigmoid(h)
         grad_wh = x.T.dot(grad_h)
         grad_bh = np.sum(grad_h, axis=0, keepdims=True)

         # Update weights and biases
         wout += grad_wout * learning_rate
         bout += grad_bout * learning_rate
         wh += grad_wh * learning_rate
         bh += grad_bh * learning_rate
```

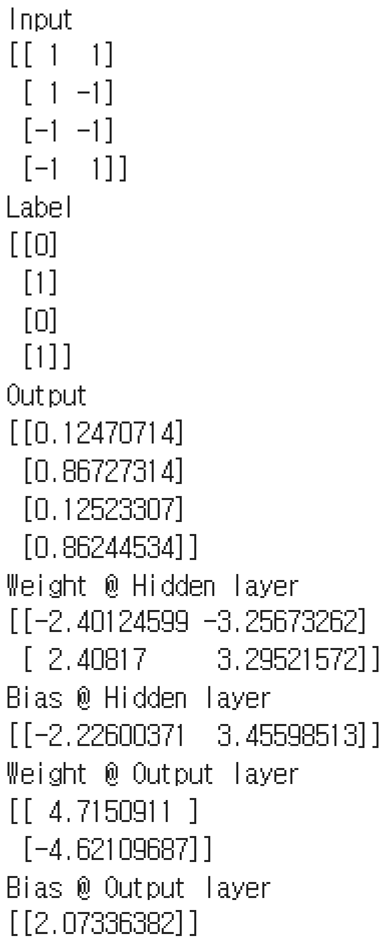


```
Epoch 0 : 1.264783292711578
Epoch 500 : 1.000410301966061
Epoch 1000 : 0.9997919590083908
Epoch 1500 : 0.9989677933762147
Epoch 2000 : 0.995787723453303
Epoch 2500 : 0.9742161349844749
Epoch 3000 : 0.8669798788416558
Epoch 3500 : 0.7475512915835518
Epoch 4000 : 0.503842729214347
Epoch 4500 : 0.1408154431024505
```

3 XOR Problem with Multi layer perceptron



```
[ ]  print('Input')
     print(x)
     print('Label')
     print(y)
     print('Output')
     print(y_pred)
     print('Weight @ Hidden layer')
     print(wh)
     print('Bias @ Hidden layer')
     print(bh)
     print('Weight @ Output layer')
     print(wout)
     print('Bias @ Output layer')
     print(bout)
```



```
Input
[[ 1  1]
 [ 1 -1]
 [-1 -1]
 [-1  1]]
Label
[[0]
 [1]
 [0]
 [1]]
Output
[[0.12470714]
 [0.86727314]
 [0.12523307]
 [0.86244534]]
Weight @ Hidden layer
[[-2.40124599 -3.25673262]
 [ 2.40817     3.29521572]]
Bias @ Hidden layer
[[-2.22600371  3.45598513]]
Weight @ Output layer
[[ 4.7150911 ]
 [-4.62109687]]
Bias @ Output layer
[[2.07336382]]
```