CHAPTER 9: NEURAL NETWORK FOR XOR

Theory

In this chapter, we will discuss the method about the implementation and training the neural network for XOR problem which is an important challenge in machine learning.

Recap

Let's remember the contents of the previous chapter. We should pay attention to the following rules; selection of proper learning rate, data preprocessing and overcome of overfitting when training the model.

Proper learning rate

- Observe the cost function
- Check it goes down at a reasonable rate

Data Preprocessing

- Zero-centered data
- Normalization

Overfitting

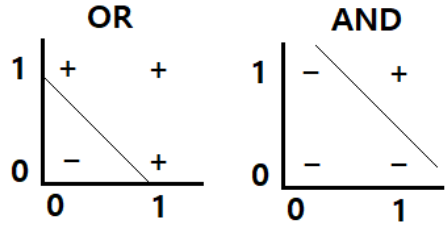
- Use more training data
- Reduce the number of features
- Regularization: don't use too big numbers in the weight.

XOR Problem

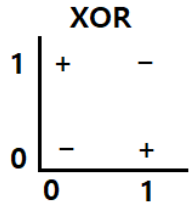
AND, OR, XOR is the most used in computer logic operation, so implementation these operations with NN is the very important problem.

X1	X2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

As we can know from the below graph, AND and OR problem can be solved with simple linear logistic regression unit.



But one logistic regression unit cannot separate XOR.

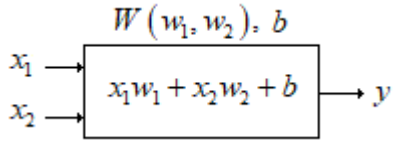


Dr. Marvin Minsky, founder of the MIT AI Lab said we need to use MLP, multilayer perceptron (multilayer neural nets) for solving of this XOR problem and proved this mathematically.

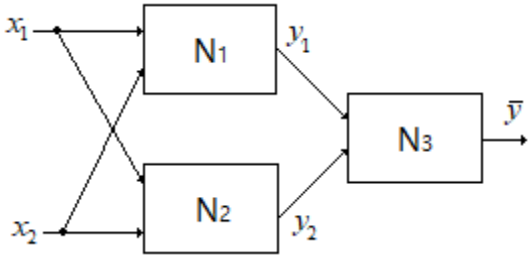
So let's try XOR problem with MLP in the next section.

NN for XOR

We will use 3 simple logistic regression units and each unit will be represented as below graph and equation.



Let's implement the NN with as following structure with 3 units.



First, let's set the weights as follows and check the output to see if the neural network can solve the XOR problem.

$$N_1: W_1 = \begin{bmatrix} 5 \\ 5 \end{bmatrix},\ b_1 = -8$$

$$N_2: W_2 = \begin{bmatrix} -7 \\ -7 \end{bmatrix},\ b_2 = 3$$

$$N_3: W_3 = \begin{bmatrix} -11 \\ -11 \end{bmatrix},\ b_3 = 6$$

Then let's calculate the output of NN.

First, let's consider in case of $x_1 = 0$, $x_2 = 0$ and check the output of NN, $\bar{y}$ is same as XOR(x_1, x_2) or not.

$$\begin{aligned} y_1 &= S(W_1 x + b_1) = \\ &= S\left(W_{11}x_1 + W_{12}x_2 + b_1\right) = \\ &= S\left(5\times 0 + 5\times 0 - 8\right) = \\ &= S\left(-8\right) = 0 \end{aligned}$$

$$
\begin{aligned}
y_2 &= S(W_2 x + b_2) = \\
&= S\left(W_{21}x_1 + W_{22}x_2 + b_2\right) = \\
&= S\left(-7\times 0 - 7\times 0 + 3\right) = \\
&= S\left(3\right) = 1
\end{aligned}
$$

As we have studied in the last chapter, the output of the sigmoid function for the positive value is 1, and the output for the negative value is 0.

Now we can calculate the $\bar{y}$ using y_1 and y_2.

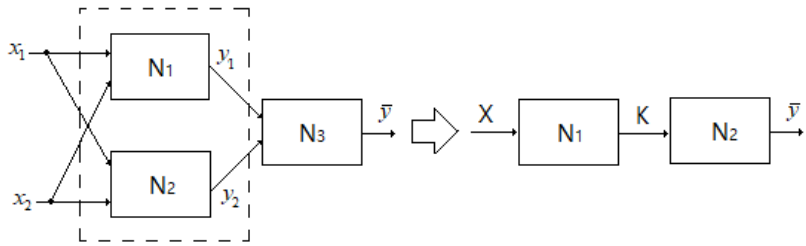
$$
\begin{aligned}
\bar{y} &= S(W_3 y + b_3) = \\
&= S\left(W_{31}y_1 + W_{32}y_2 + b_3\right) = \\
&= S\left(-11\times 0 - 11\times 1 + 6\right) = \\
&= S\left(-5\right) = 0
\end{aligned}
$$

In this way, we can calculate the $\bar{y}$ when x_1 and x_2 are of different value.

In the following table we can compare the output of NN and XOR result.

x_1	x_2	y_1	y_2	$\bar{y}$	XOR	Check
0	0	0	1	0	0	Ok
0	1	0	0	1	1	Ok
1	0	0	0	1	1	Ok
1	1	1	0	0	0	Ok

As we can know from the table, all the output of NN is same as XOR operation. And here we can merge N1 and N2 into multinomial NN.



Then the weights and bias can be following values.

$$W_1 = \begin{bmatrix} 5 & -7 \\ 5 & -7 \end{bmatrix}, \qquad B_1 = \begin{bmatrix} -8 & 3 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} -11 \\ -11 \end{bmatrix}, \qquad b_2 = 6$$

Let's consider the method to implement the NN for XOR with the Tensorflow in the next lab.

AIM

The aim of the following lab exercise is to implement the Neural Network could solve the XOR problem with Pycharm and Tensorflow and then train the model and evaluate the trained result.

Following steps are required.

Task 1: Preparation of the development environment

Task 2: Making script for XOR problem

Task 3: Running script and get results

LAB EXERCISE 9: NEURAL NETWORK FOR XOR

1. Preparation of the development environment
2. Making script for XOR problem
3. Running script and get results

Task 1: Preparation of the development environment

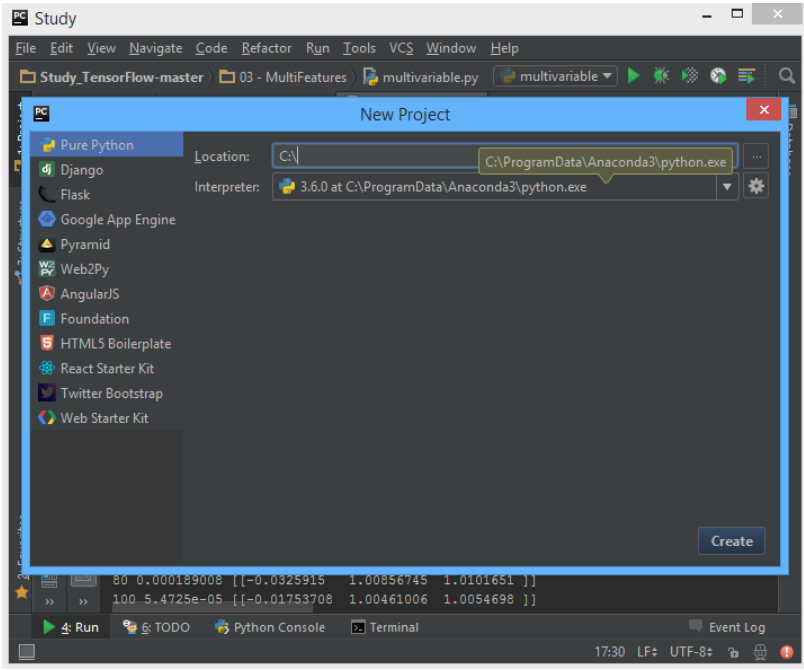
STEP 1: Problem Statement

In this lab, let's create a python script to build and train a neural network for the following XOR training materials.

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

STEP 2: Run PyCharm and create the project

First, run PyCharm and create a new project using File/New Project menu.

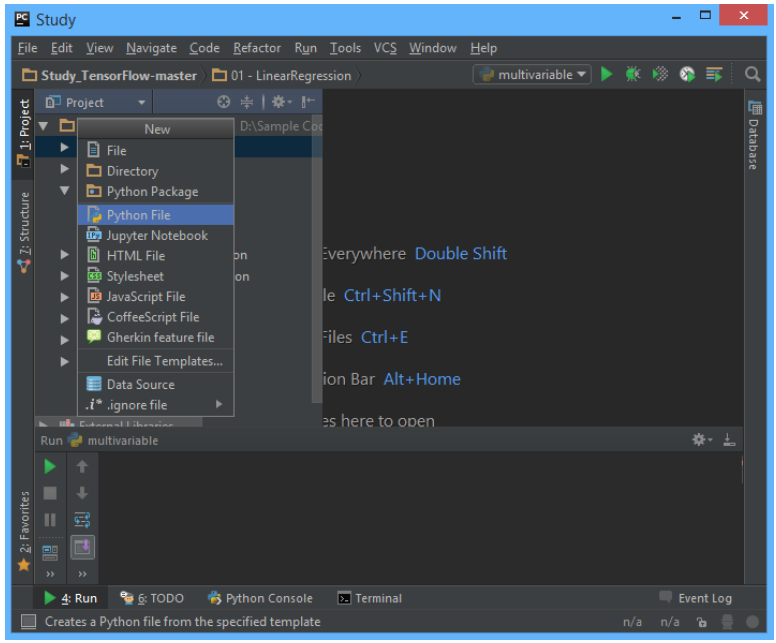


Select the project location and interpreter there and click Create Button. Then the empty project will be created.

STEP 3: Create a python script file

Create a new python file using File/New/Python File.

And give the file name as “NN.py”



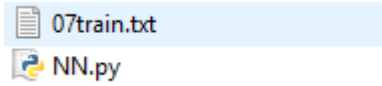
STEP 4: Create a training data file

Let’s make training data as the txt file format. Create “07train.txt” file in the same folder with python script and type as bellows using Notepad or EditPlus and so on.

07train.txt

```
1  # xor
2  # x1  x2  y
3  0    0   0
4  0    1   1
5  1    0   1
6  1    1   0
```

And then we can both of python script file and training data text file are in the same location in Windows Explorer.



Task1 is completed.

Task 2: Making script for XOR problem

STEP 5: Import packages

First, import the Tensorflow and Numpy package in order to use in Python script using the code.

```
import tensorflow as tf
import numpy as np
```

STEP 6: Load training data

We can load the training data from the text file using numpy function.

```
xy = np.loadtxt('07train.txt', unpack=True)
x_data = np.transpose(xy[0:-1])
y_data = np.reshape(xy[-1], (4, 1))
```

The first command invokes text except for the comment with # and converts it to float32 type, and stores it in xy variable. And then using last 2 commands split the training data into x_data and y_data.

After the above 3 commands, x_data and y_data should be as following

x_data = [[0. 0. 1. 1.]

[0. 1. 0. 1.]]

y_data = [[0. 1. 1. 0.]]

STEP 7: Define model weights

And define X, Y, W as tensorflow variable.

```
X = tf.placeholder(tf.float32, name='x-input')
Y = tf.placeholder(tf.float32, name='y-input')
```

```
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0),
name='weight1')
W2 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0),
name='weight2')

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
```

STEP 8: Define the softmax hypothesis

Next, let's define the hypothesis.

```
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)
```

Then the hypothesis will be implemented as follow function.

$$hypothesis = S\left(W_2 \times S\left(W_1 x + b_1\right) + b_2\right)$$

STEP 9: Define the cost function

And cost function could be defined as follows.

```
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1-Y) *
tf.log(1 - hypothesis))
```

STEP 10: Define the learning rate and optimizer

We can define the learning rate and optimizer as the same as the previous chapter.

```
a = tf.Variable(0.1)
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

Task2 is completed.

Task 3: Running script and get results

Let's complete the code and run it.

STEP 11: Initialize the tensorflow variables

Initialize all of the tensorflow variables.

```
init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)
```

STEP 12: Train the model

Then let's train the model and print the results.

```
for step in xrange(20000):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    if step % 200 == 0:
        summary = sess.run(merged, feed_dict={X: x_data,
Y: y_data})
        print step, sess.run(cost, feed_dict={X: x_data,
Y: y_data}), sess.run(w1), sess.run(w2)

correct_prediction = tf.equal(tf.floor(hypothesis+0.5),
Y)

accuracy = tf.reduce_mean(tf.cast(correct_prediction,
"float"))
print sess.run([hypothesis, tf.floor(hypothesis+0.5),
correct_prediction], feed_dict={X: x_data, Y: y_data})
print "accuracy", accuracy.eval({X: x_data, Y: y_data})
```

The coding is complete here. We print the value of hypothesis and accuracy.

Then the full code is below.

```
import tensorflow as tf
import numpy as np

xy = np.loadtxt('07train.txt', unpack=True)
x_data = np.transpose(xy[0:-1])
y_data = np.reshape(xy[-1], (4, 1))

print x_data
print y_data

X = tf.placeholder(tf.float32, name='x-input')
Y = tf.placeholder(tf.float32, name='y-input')

W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0),
```

```
                  name='weight1')
W2 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0),
                  name='weight2')

b1 = tf.Variable(tf.zeros([2]), name="Bias1")
b2 = tf.Variable(tf.zeros([1]), name="Bias2")

L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)

cost = -tf.reduce_mean(Y * tf.log(hypothesis) +
                       (1-Y) * tf.log(1 - hypothesis))

a = tf.Variable(0.1)
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)

init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)

    for step in xrange(20000):
        sess.run(train,feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            summary = sess.run(merged,
                               feed_dict={X: x_data,
                                          Y: y_data})
            print step, sess.run(cost,
                                 feed_dict={X:x_data,
                                            Y:y_data}),
                sess.run(w1), sess.run(w2)

    correct_prediction = tf.equal(tf.floor
                 (hypothesis+0.5), Y)

    accuracy = tf.reduce_mean(tf.cast(
                 correct_prediction, "float"))
    print sess.run([hypothesis,
                    tf.floor(hypothesis+0.5),
                    correct_prediction],
                   feed_dict={X: x_data, Y: y_data})
    print "accuracy", accuracy.eval({X: x_data,
                                     Y: y_data})
```

Let's run this code.

Then we can see the results such as below.

As we can know from the results, this neural network is working well for the XOR problem.

```
198000 cost 0.00878375
    predict [[ 0.01067145]
 [ 0.99207062]
 [ 0.99207932]
 [ 0.00845705]]
    W1, B1 [array([[ 6.84843874,  5.02159739],
       [ 6.83831406,  5.0197401 ]], dtype=float32), array([-5.00321722], dtype=float32)]
    W2, B2 [array([[ 10.83042145],
       [-11.59721661]], dtype=float32), array([-5.00321722], dtype=float32)]
[array([[ 0.],
       [ 1.],
       [ 1.],
       [ 0.]], dtype=float32), array([[ True],
       [ True],
       [ True],
       [ True]], dtype=bool)]
Accuracy: 1.0
```

LAB CHALLENGE

Challenge

In the task2, we have implemented NN using the 2-layer neural network which has dimension 2*2 and 2*1.

Please implement the 3-layer neural network which has dimension 2*3, 3*3, 3*1 with Tensorflow and confirm the result.

SUMMARY

XOR problem is one of the challenges in logistic regression problem and proved couldn't solve with a single unit, so we have implemented a multi-layer neural network and have studied the method of training.

Further, we can expand this method to multi-layer NN problem which has more than 3 layers.

REFERENCES

- https://en.wikipedia.org/wiki/multi-layer neural network
- https://en.wikipedia.org/wiki/TensorFlow
- https://en.wikipedia.org/wiki/back propagation

INDEX