CHAPTER 10: RELU

Theory

In this chapter, we will discuss the ReLU function which can be used as an activation function instead of the sigmoid. And we will check the accuracy with several kinds of activation functions and then check the validity of ReLU function.

Recap

Let's remember the contents of the previous chapter. The most important thing in the last chapter is to have an understanding of XOR problems and multi-layer neural network (or MLP) for solving the XOR problem.

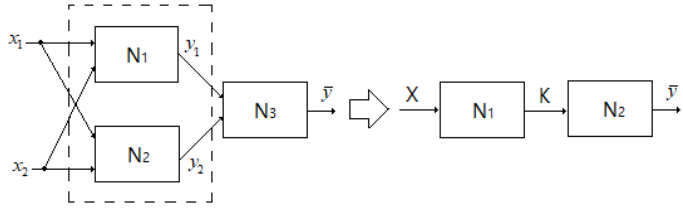
XOR Problem

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.

X1	X2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

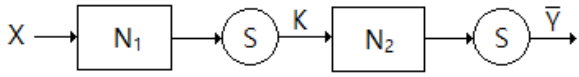
Multi-Layer Neural Network

It was proved that the XOR problem couldn't be solved using a single NN unit, so we have learned about the multi-layer NN for solving it.



The limitation of Sigmoid

Let's consider XOR problem again we discussed the previous chapter. In the previous chapter, we have configured NN as 2-layers such as following the diagram.



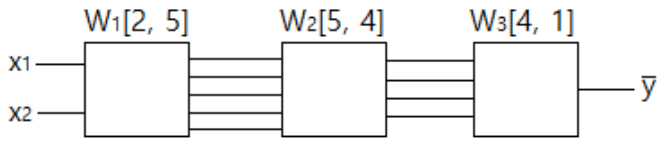
"S" is the sigmoid function which is called activation function. And this is implemented by Tensorflow with these codes.

```
W1 = tf.Variable(tf.random_uniform([2, 2], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))

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)
```

In this way, we can implement the 3-layers NN for XOR simply as following codes.




```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias3")

L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

In this NN, we call the first layer as **Input Layer**, and call the last layer as **Output Layer**, and call middle layers as **Hidden Layer**.

Then let's build the 9-hidden layers NN.

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W4 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W5 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W6 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W7 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W8 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W9 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W10 = tf.Variable(tf.random_uniform([5, 5], -1.0, 1.0))
W11 = tf.Variable(tf.random_uniform([5, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([5]), name="Bias2")
b3 = tf.Variable(tf.zeros([5]), name="Bias3")
b4 = tf.Variable(tf.zeros([5]), name="Bias4")
b5 = tf.Variable(tf.zeros([5]), name="Bias5")
b6 = tf.Variable(tf.zeros([5]), name="Bias6")
b7 = tf.Variable(tf.zeros([5]), name="Bias7")
b8 = tf.Variable(tf.zeros([5]), name="Bias8")
b9 = tf.Variable(tf.zeros([5]), name="Bias9")
b10 = tf.Variable(tf.zeros([5]), name="Bias10")
b11 = tf.Variable(tf.zeros([1]), name="Bias11")

L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)
L7 = tf.sigmoid(tf.matmul(L6, W7) + b7)
L8 = tf.sigmoid(tf.matmul(L7, W8) + b8)
L9 = tf.sigmoid(tf.matmul(L8, W9) + b9)
L10 = tf.sigmoid(tf.matmul(L9, W10) + b10)

hypothesis = tf.sigmoid(tf.matmul(L10, W11) + b11)
```

After that, we can train the neural network with the above model and check the accuracy.

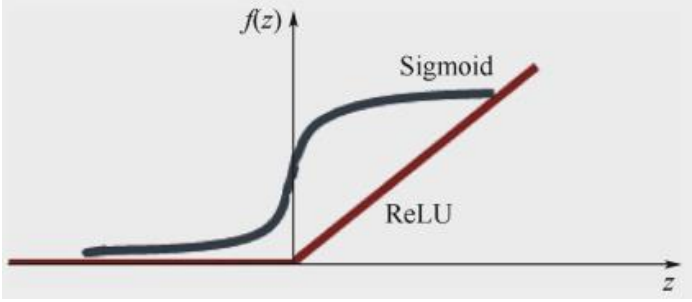
But we can know the accuracy is 0.5 with the 9-hidden layers although the accuracy was 1.0 with 2-layers NN in the previous chapter. From this, we can know the construction of NN isn't appropriate in this problem.

In the next section, let's discuss the new activation function for the resolve of this problem.

```
196000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
[array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 0.],
       [ 1.]], dtype=float32)]
Accuracy: 0.5
```

ReLU for Activation Function

The professor Geoffrey Hinton said we used the wrong type of non-linearity. This means the sigmoid we used as activation function isn't appropriate in this NN.



So we introduce a new function which is called as **ReLU** (Rectified Linear Unit).

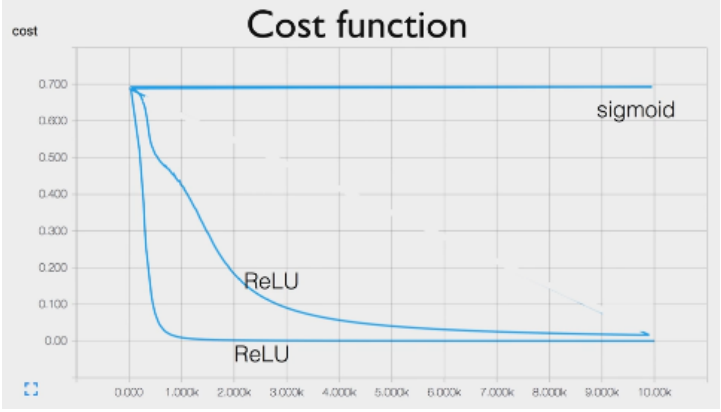
$$ReLU(x) = max(0, x)$$

We can replace the sigmoid function into ReLU function and try to train and check the accuracy again.

```
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L5 = tf.nn.relu(tf.matmul(L4, W5) + b5)
L6 = tf.nn.relu(tf.matmul(L5, W6) + b6)
```

```
L7 = tf.nn.relu(tf.matmul(L6, W7) + b7)
L8 = tf.nn.relu(tf.matmul(L7, W8) + b8)
L9 = tf.nn.relu(tf.matmul(L8, W9) + b9)
L10 = tf.nn.relu(tf.matmul(L9, W10) + b10)
```

```
196000 [2.6226094e-06, array([[  2.59195826e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.43454133e-06]], dtype=float32)]
198000 [2.607708e-06, array([[  2.55822852e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.40260101e-06]], dtype=float32)]
[array([[  2.52509381e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.37124709e-06]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 1.],
       [ 0.]], dtype=float32)]
Accuracy: 1.0
```



As we can know from the accuracy and cost function, ReLU function works very well.

Also, we can use another type of activation function besides of sigmoid or ReLU describes in the bellows.

- Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- tanh

$$f(x) = \tanh(x)$$

- ReLU

$$f(x) = \max(0, x)$$

- Leaky ReLU

$$f(x) = \max(0.1x, x)$$

- Maxout

$$f(x) = \max\left(w_1^T x + b_1, w_2^T x + b_2\right)$$

- ELU

$$f(x) = \begin{cases} x & if \ x > 0 \\ \alpha\left(\exp(x) - 1\right) & if \ x \le 0 \end{cases}$$

And the performance table is shown below.

maxout	ReLU	VLReLU	tanh	Sigmoid
93.94	**92.11**	92.97	89.28	n/c
93.78	91.74	92.40	89.48	n/c
–	91.93	**93.09**	–	n/c
91.75	90.63	92.27	**89.82**	n/c
n/c†	90.91	92.43	89.54	n/c

AIM

The aim of the following lab exercise is to implement the 8-hidden layers Neural Network with ReLU as the activation function for 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 using ReLU

Task 3: Running script and get results

LAB EXERCISE 9: NEURAL NETWORK FOR XOR

1. Preparation of the development environment
2. Making script using ReLU
3. Running script and get results

Task 1: Preparation of the development environment

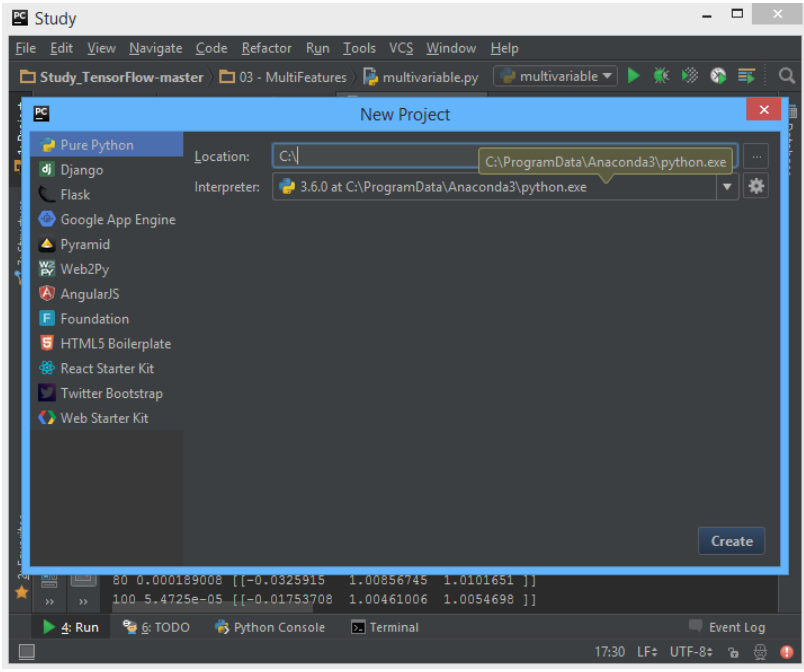
STEP 1: Problem Statement

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.

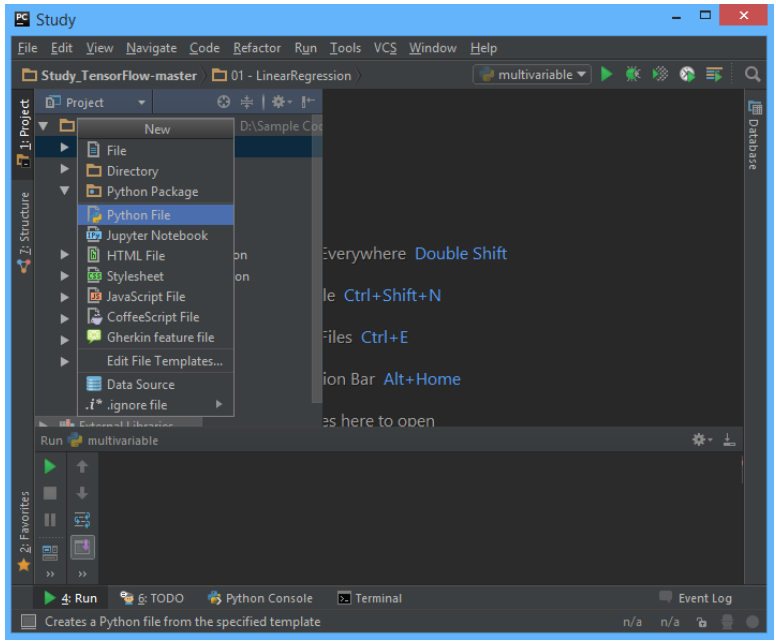


Select the project location and interpreter there and click Create Button. Then the empty project will be created.

STEP 3: Create a python script file

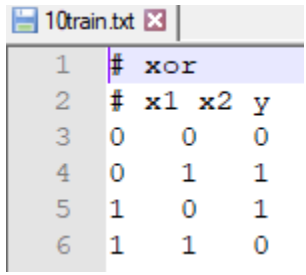
Create a new python file using File/New/Python File.

And give the file name as “NN_relu.py”

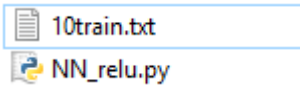


STEP 4: Create a training data file

Let’s make training data as the txt file format. Create “10train.txt” file in the same folder with python script and type as bellows using Notepad or EditPlus and so on.



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 using ReLU

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('10train.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.

```
w1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0),
name='weight1')
```

```
w2 = tf.Variable(tf.random_uniform([5, 10], -1.0, 1.0),
name='weight2')
w3 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight3')
w4 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight4')
w5 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight5')
w6 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight6')
w7 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight7')
w8 = tf.Variable(tf.random_uniform([10, 1], -1.0, 1.0),
name='weight8')

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b3 = tf.Variable(tf.zeros([10]), name="Bias3")
b2 = tf.Variable(tf.zeros([10]), name="Bias2")
b4 = tf.Variable(tf.zeros([10]), name="Bias4")
b5 = tf.Variable(tf.zeros([10]), name="Bias5")
b6 = tf.Variable(tf.zeros([10]), name="Bias6")
b7 = tf.Variable(tf.zeros([10]), name="Bias7")
b8 = tf.Variable(tf.zeros([1]), name="Bias8")

L2 = tf.nn.relu(tf.matmul(X, w1) + b1)
L3 = tf.nn.relu(tf.matmul(L2, w2) + b2)
L4 = tf.nn.relu(tf.matmul(L3, w3) + b3)
L5 = tf.nn.relu(tf.matmul(L4, w4) + b4)
L6 = tf.nn.relu(tf.matmul(L5, w5) + b5)
L7 = tf.nn.relu(tf.matmul(L6, w6) + b6)
L8 = tf.nn.relu(tf.matmul(L7, w7) + b7)
```

STEP 8: Define the softmax hypothesis

Next, let's define the hypothesis.

```
hypothesis = tf.sigmoid(tf.matmul(L8, w8) + b8)
```

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:
        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('10train.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, 5], -1.0, 1.0),
name='weight1')
w2 = tf.Variable(tf.random_uniform([5, 10], -1.0, 1.0),
name='weight2')
w3 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight3')
w4 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight4')
w5 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight5')
w6 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight6')
w7 = tf.Variable(tf.random_uniform([10, 10], -1.0, 1.0),
name='weight7')
w8 = tf.Variable(tf.random_uniform([10, 1], -1.0, 1.0),
name='weight8')

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b3 = tf.Variable(tf.zeros([10]), name="Bias3")
b2 = tf.Variable(tf.zeros([10]), name="Bias2")
b4 = tf.Variable(tf.zeros([10]), name="Bias4")
b5 = tf.Variable(tf.zeros([10]), name="Bias5")
b6 = tf.Variable(tf.zeros([10]), name="Bias6")
b7 = tf.Variable(tf.zeros([10]), name="Bias7")
b8 = tf.Variable(tf.zeros([1]), name="Bias8")

L2 = tf.nn.relu(tf.matmul(X, w1) + b1)
L3 = tf.nn.relu(tf.matmul(L2, w2) + b2)
L4 = tf.nn.relu(tf.matmul(L3, w3) + b3)
L5 = tf.nn.relu(tf.matmul(L4, w4) + b4)
L6 = tf.nn.relu(tf.matmul(L5, w5) + b5)
L7 = tf.nn.relu(tf.matmul(L6, w6) + b6)
L8 = tf.nn.relu(tf.matmul(L7, w7) + b7)

hypothesis = tf.sigmoid(tf.matmul(L8, w8) + b8)

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)

w1_hist = tf.histogram_summary("weights1", w1)
w2_hist = tf.histogram_summary("weights2", w2)

b1_hist = tf.histogram_summary("biases1", b1)
b2_hist = tf.histogram_summary("biases2", b2)

y_hist = tf.histogram_summary("y", Y)

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:
            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.

```
196000 [2.6226094e-06, array([[  2.59195826e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.43454133e-06]], dtype=float32)]
198000 [2.607708e-06, array([[  2.55822852e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.40260101e-06]], dtype=float32)]
[array([[  2.52509381e-06],
       [  9.99999642e-01],
       [  9.99994874e-01],
       [  2.37124709e-06]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 1.],
       [ 0.]], dtype=float32)]
Accuracy: 1.0
```

LAB CHALLENGE

Challenge

In this lab, we have implemented NN using 8-hidden layers neural network for the XOR problem.

Copy the MNIST dataset from the Chapter8 lab and then implement the multi-layer neural network for MNIST dataset.

SUMMARY

The sigmoid function is the good activation function for linear regression and logistic problem, but it has limitation for the multi-layer neural network. We have confirmed it using several NNs which have different depth of layers with its accuracy.

ReLU is one of the good candidates for resolving this issue.

REFERENCES

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

INDEX