# **Artificial Intelligence and Machine Learning Fundamentals**

**Activity 14**: Written Digit Detection

1. This section will discuss how to provide more security for the cryptocurrency traders via the detection of hand-written digits. We will be using assuming that you are a software developer at a new Cryptocurrency trader platform.

The latest security measure you are implementing requires the recognition of hand-written digits. Use the MNIST library to train a neural network to recognize digits. You can read more about this dataset on <https://www.tensorflow.org/tutorials/>.

1. Improve the accuracy of the model as much as possible. And to ensure that it happens correctly, you will need to complete the previous topic.
2. Load the dataset and format the input

import tensorflow.keras.datasets.mnist as mnist

(features\_train, label\_train),

(features\_test, label\_test) = mnist.load\_data()

features\_train = features\_train / 255.0

features\_test = features\_test / 255.0

def flatten(matrix):

return [elem for row in matrix for elem in row]

features\_train\_vector = [

flatten(image) for image in features\_train

]

features\_test\_vector = [

flatten(image) for image in features\_test

]

import numpy as np

label\_train\_vector = np.zeros((label\_train.size, 10))

for i, label in enumerate(label\_train\_vector):

label[label\_train[i]] = 1

label\_test\_vector = np.zeros((label\_test.size, 10))

for i, label in enumerate(label\_test\_vector):

label[label\_test[i]] = 1

1. Set up the Tensorflow graph. Instead of the sigmoid function, we will now use the relu function.

import tensorflow as tf

f = tf.nn.softmax

x = tf.placeholder(tf.float32, [None, 28 \* 28 ])

W = tf.Variable( tf.random\_normal([784, 10]))

b = tf.Variable( tf.random\_normal([10]))

y = f(tf.add(tf.matmul( x, W ), b ))

1. Train the model.

import random

y\_true = tf.placeholder(tf.float32, [None, 10])

cross\_entropy = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(

logits=y,

labels=y\_true

)

cost = tf.reduce\_mean(cross\_entropy)

optimizer = tf.train.GradientDescentOptimizer(

learning\_rate = 0.5

).minimize(cost)

session = tf.Session()

session.run(tf.global\_variables\_initializer())

iterations = 600

batch\_size = 200

sample\_size = len(features\_train\_vector)

for \_ in range(iterations):

indices = random.sample(range(sample\_size), batchSize)

batch\_features = [

features\_train\_vector[i] for i in indices

]

batch\_labels = [

label\_train\_vector[i] for i in indices

]

min = i \* batch\_size

max = (i+1) \* batch\_size

dictionary = {

x: batch\_features,

y\_true: batch\_labels

}

session.run(optimizer, feed\_dict=dictionary)

1. Test the model

label\_predicted = session.run(classify( x ), feed\_dict={

x: features\_test\_vector

})

label\_predicted = [

np.argmax(label) for label in label\_predicted

]

confusion\_matrix(label\_test, label\_predicted)

The output is as follows:

array([[ 0, 0, 223, 80, 29, 275, 372, 0, 0, 1],

[ 0, 915, 4, 10, 1, 13, 192, 0, 0, 0],

[ 0, 39, 789, 75, 63, 30, 35, 0, 1, 0],

[ 0, 6, 82, 750, 13, 128, 29, 0, 0, 2],

[ 0, 43, 16, 16, 793, 63, 49, 0, 2, 0],

[ 0, 22, 34, 121, 40, 593, 76, 5, 0, 1],

[ 0, 29, 34, 6, 44, 56, 788, 0, 0, 1],

[ 1, 54, 44, 123, 715, 66, 24, 1, 0, 0],

[ 0, 99, 167, 143, 80, 419, 61, 0, 4, 1],

[ 0, 30, 13, 29, 637, 238, 58, 3, 1, 0]], dtype=int64)

1. Calculate the accuracy score:

accuracy\_score(label\_test, label\_predicted)

The output is as follows:

0.4633

1. By re-running the code segment responsible for training the data set, we can improve the accuracy:

for \_ in range(iterations):

indices = random.sample(range(sample\_size), batch\_size)

batch\_features = [

features\_train\_vector[i] for i in indices

]

batch\_labels = [

label\_train\_vector[i] for i in indices

]

min = i \* batch\_size

max = (i+1) \* batch\_size

dictionary = {

x: batch\_features,

y\_true: batch\_labels

}

session.run(optimizer, feed\_dict=dictionary)

Second run: 0.5107

Third run: 0.5276

Fourth run: 0.5683

Fifth run: 0.6002

Sixth run: 0.6803

Seventh run: 0.6989

Eighth run: 0.7074

Ninth run: 0.713

Tenth run: 0.7163

Twentieth run: 0.7308

Thirtieth run: 0.8188

Fortieth run: 0.8256

Fiftieth run: 0.8273

At the end of the fiftieth run, the improved confusion matrix looks as follows:

array([

[946, 0, 6, 3, 0, 1, 15, 2, 7, 0],

[ 0,1097, 3, 7, 1, 0, 4, 0, 23, 0],

[11, 3, 918, 11, 18, 0, 13, 8, 50, 0],

[3, 0, 23, 925, 2, 10, 4, 9, 34, 0],

[2, 2, 6, 1, 929, 0, 14, 2, 26, 0],

[16, 4, 7, 62, 8, 673, 22, 3, 97, 0],

[8, 2, 4, 3, 8, 8, 912, 2, 11, 0],

[5, 9, 33, 6, 9, 1, 0, 949, 16, 0],

[3, 4, 5, 12, 7, 4, 12, 3, 924, 0],

[8, 5, 7, 40, 470, 11, 5, 212, 251, 0]

],

dtype=int64)

Not a bad result. More than 8 out of 10 digits are accurately recognized.