German-Russian Institute of Advanced Technologies

TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

**Practice 2**

**«Tuning the handwritten digits recognition model by adding ReLU and**

**Changing number of layers»**

Student: Marushkai N. S.

Lecturer: Makmutova A.Z.

Kazan 2018

During this work, I have learned deeper how the algorythm of recognizing handwritten digits works and how to add layers and change activation functions for such kind of algorithms. During this work, the source program, which was given in presentation was modified, in order to use the ReLU activation function and work with one more hidden layer. This program gave such results of model evaluation as an output:

Accuracy: 0.9781

As we can see, the accuracy improved, according to use of the ReLU activation function and adding additional layer. Without this modifications, it was about 0.91

Here is the code listing:

**import** tensorflow **as** tf

**from** tensorflow.examples.tutorials.mnist **import** input\_data

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

**with** tf.name\_scope('input\_data') **as** input\_scope:

x = tf.placeholder(tf.float32, [None, 784])

keep\_probability = tf.placeholder(tf.float32,name='prob')

**with** tf.name\_scope('layers') **as** layers\_scope:

W = tf.Variable(tf.truncated\_normal([784, 784], stddev=0.1), name='weight\_1')

b = tf.Variable(tf.truncated\_normal([784], stddev=0.1), name='bias\_1')

*#relu*

hidden\_1 = tf.layers.dropout(tf.nn.relu(tf.matmul(x, W) + b))

W\_2 = tf.Variable(tf.zeros(shape=[784, 10]), name='weight\_2')

b\_2 = tf.Variable(tf.zeros(shape=[10]), name='bias\_2')

*#Softmax*

y = tf.nn.softmax(tf.matmul(hidden\_1, W\_2) + b\_2)

*#output*

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

cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y), reduction\_indices=[1]))

train\_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross\_entropy)

**with** tf.Session() **as** sess:

init = tf.global\_variables\_initializer()

sess.run(init)

**for** i **in** range(2000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100)

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys, keep\_probability: 0.5})

correct\_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y\_, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

**print**("Accuracy: %s" % sess.run(accuracy, feed\_dict={x: mnist.test.images, y\_: mnist.test.labels}))

prediction = (sess.run(y, feed\_dict={x: mnist.test.images[1:2]}))

**for** index, r **in** enumerate(prediction):

**print**(index, r)

**print**('Label is:', mnist.test.labels[1:2])