**In the name of GOD**

Neural Network Project Report

By Seyed Mohammad Eshagh Maibodi (40313414)

Teacher: Dr. Sadatee

Winter 1403

**Description:** In this project I develop a model that can recognize numbers from 0 to 9 using mnist dataset. In case of getting better results, I use deferent configuration to find the best model with highest accuracy.

**Step One: Import Libraries and Load Dataset**

# Import Library  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras.utils import to\_categorical  
from tensorflow.keras.layers import Input, Dense, Flatten  
from tensorflow.keras import Model  
from tensorflow.keras.optimizers import SGD  
import matplotlib.pyplot as plt

# Load MNIST Dataset  
(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data(path='mnist.npz')

\*This step will remain fixed in all cases.

As it’s shown in the code we need to import tensorflow library to use keras. In keras.utils we need “to\_categorical” function to make OneHot for outputs. From keras.layers we need “Input”, “Dense” and “Flatten”. I will explain the usage further in code. From keras we need “Model” to create the neural network model. The optimizer is actually the learning algorithm. In this case we use SGD(Stochastic gradient descent) that is based on derivative of the weights that shows the next step for minimize or maximizing. We can simply change the magnitude of this step by Multiply it. At the end we need to plot data of accuracy and loss by matplotlib library.

**Step Two: Normalizing Inputs**

For better results we need to normalize inputs. This action will lower the sensitivity of model on bigger amounts of inputs that outputs. For normalizing we use the following action. Notice that for grayscale image the value for each pixel is a number between 0 to 255. So by dividing the values ​​by 255, we can make it 0 to 1.

# Normalize Input Data (Min:0, Max:1)  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

**Step Three: OneHot Outputs**

As I explained we use “to\_categorical” method to OneHot outputs. There are 10 classes for 10 numbers(0 to 9) and an image cant be two number at the same time.

# Convert Labels to oneHot (10 Classes for 0 to 9 Numbers)  
y\_train = to\_categorical(y\_train, num\_classes=10)  
y\_test = to\_categorical(y\_test, num\_classes=10)

**Step Four: Define Neural Network Layers and Model**

Mnist dataset includes images with 28x28 dimension. So the model input must have a 28x28 shape. But we must make them flat (1 dimension) values. Function “Flatten()” will do this for us.

For layers we use “Dense” function to make a fully-connected network. In this function for each layer we must set 4 property: we have to define the number of neurons, then we have to specify the activation function(Sigmoid, ReLu, Softmax, …). Then we can use bias or not. At the end we have to determine the previous layer for current one. For example for the first hidden layer, the previous layer is input layer.

\*Further, we will change this property for better performance.

# Define Neural Network Layers  
input\_layer = Input(shape=(28, 28))  
flatten = Flatten()(input\_layer)  
h1 = Dense(64, activation='sigmoid', use\_bias=True)(flatten)  
h2 = Dense(64, activation='sigmoid', use\_bias=True)(h1)  
output\_layer = Dense(10, activation='sigmoid', use\_bias=True)(h2)

After declaring the layers we must give the input and output layer to the “Model” function.

# Define Neural Network Model  
model = Model(input\_layer, output\_layer)

Now we can see the model summary by “model.summary()” command. In figure-1, an example of a model summary is shown.

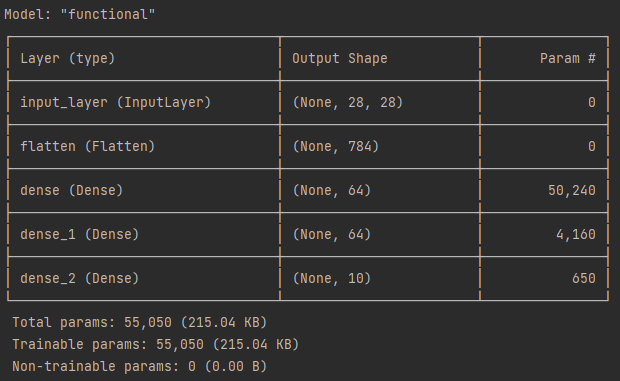


Figure example of a model summary

After all we have to compile the model using “model. compile()” function. In this method we must set 3 properties. Optimizer is the learning algorithm that we use SGD with 0.01 learning rate. Loss property is the loss function. We use min square error for that. In the end the metrics must set.

# Compile Model  
model.compile(optimizer=SGD(learning\_rate=0.1), loss='mse', metrics=['accuracy'])

**Step Five: Train and Plot**

The train progress will start with “model.fit()” method. In this method we must set the input and output(labels) for the network and number of epochs for Iterations. For validation we can use test split of our data.

# Train Model  
result = model.fit(x\_train, y\_train, epochs=100, validation\_data=(x\_test, y\_test))

For better demonstration we plot “Accuracy”, ” Validation Accuracy” and “Loss” values by epoch.

# Plot Training Progress  
plt.plot(result.history['accuracy'], label='Accuracy')  
plt.plot(result.history['val\_accuracy'], label='Validation Accuracy')  
plt.plot(result.history['loss'], label='Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.ylim([0, 1])  
plt.legend()  
plt.show()

**Step Five: Changing Hyper-Parameters For Better Result**