INTRODUCTION TO ARTIFICIAL INTELLIGENCE

EXERCISE 5

Full names: Andreas Kalavas, Panagiota Chita

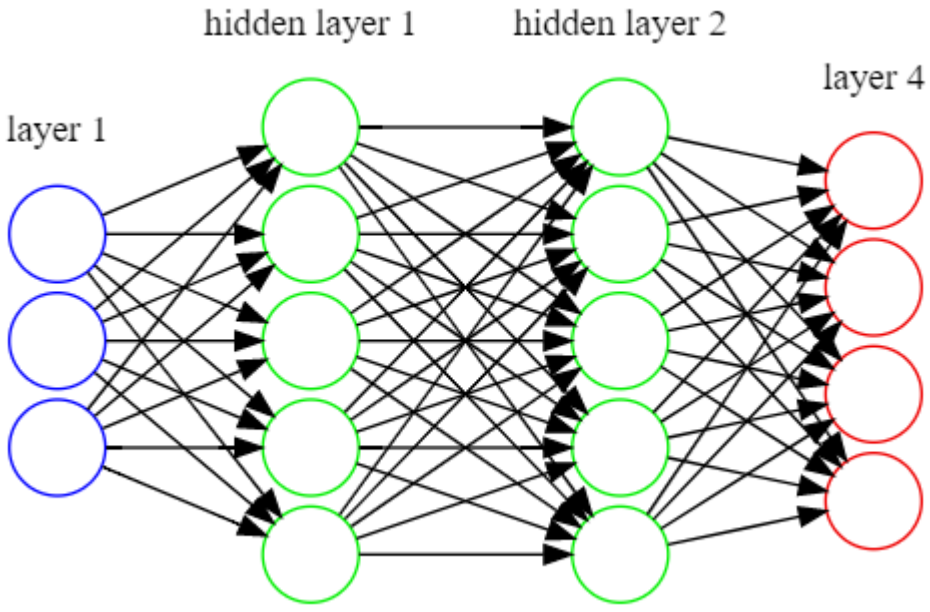
Emails: andreas.kalavas.stud@pw.edu.pl , panagiota.chita.stud@pw.edu.pl

In this report, we will explain how we implement an N-layer neural network for the classification of Handwritten Arabic characters with the aid of specific libraries such as TensorFlow. The Keras Sequential model is defined as a *linear stack of layers*. This is perfect for what we need to create an Artificial Neural Network consisting of Input, Output, and Hidden nodes

Data Process

Our data are already split into training and validation sets. At first, we input our data and we shuffle the train values randomly. After that, we imply normalization on our data trainx and we encode the trainy values to binary form.

Then the program asks the user to insert the number of layers that the neural network will be consists of (The number should be more than 2 in order to have hidden layers as well).



Hyperparameter

The parameters that we will try to optimize are the number of neurons in each layer and the activation functions we will use (either sigmoid or relu or tanh). We have to note that the number of epochs will be set constant to 20 when the hyperparameter process is executing and then it will be set at 80. That technique was chosen due to the fact that during the implementation of the hyperparameter we wanted to reduce the computational time. After this process, we change the number of epochs to a bigger one (80) as by increasing them, the performance is improving too.

We apply two methods to perform the hyperparameter optimization and the user has to choose one. Insert 1 for the first method in which we separately run the optimization in every parameter and insert 2 to randomly combine them.

First Method:
We run the process first for optimizing the activation function and after we find the best solution, we continue by applying optimization to the number of neurons.

```
Epoch 78/80
188/188 [==============================] - 4s 24ms/step - loss: 0.2185 - accuracy: 0.9250
Epoch 79/80
188/188 [==============================] - 4s 24ms/step - loss: 0.0741 - accuracy: 0.9718
Epoch 80/80
188/188 [==============================] - 4s 24ms/step - loss: 0.0228 - accuracy: 0.9849
```

Second Method:
The algorithm runs simultaneously in both parameters in order to find the optimal solution (biggest accuracy). Approximately accuracy = 0.875

Below the statistics illustrate the significant of the epochs factor and its impact on the results:

1. case: neurons = 60, function = relu, epochs 100, layers = 6

```
Epoch 93/100
380/380 [==============================] - 1s 4ms/step - loss: 0.4926 - accuracy: 0.8370
Epoch 94/100
380/380 [==============================] - 1s 4ms/step - loss: 0.5069 - accuracy: 0.8294
Epoch 95/100
380/380 [==============================] - 2s 4ms/step - loss: 0.5020 - accuracy: 0.8327
Epoch 96/100
380/380 [==============================] - 1s 4ms/step - loss: 0.4881 - accuracy: 0.8351
Epoch 97/100
380/380 [==============================] - 1s 4ms/step - loss: 0.5091 - accuracy: 0.8305
Epoch 98/100
380/380 [==============================] - 1s 4ms/step - loss: 0.5444 - accuracy: 0.8206
Epoch 99/100
380/380 [==============================] - 1s 4ms/step - loss: 0.5255 - accuracy: 0.8267
Epoch 100/100
380/380 [==============================] - 2s 4ms/step - loss: 0.5044 - accuracy: 0.8331
```

2. case: neurons = 60, function = relu, epochs 20, layers = 6

```
Epoch 15/20
380/380 [==============================] - 2s 4ms/step - loss: 1.1773 - accuracy: 0.6164
Epoch 16/20
380/380 [==============================] - 2s 4ms/step - loss: 1.1483 - accuracy: 0.6290
Epoch 17/20
380/380 [==============================] - 2s 4ms/step - loss: 1.1164 - accuracy: 0.6371
Epoch 18/20
380/380 [==============================] - 2s 4ms/step - loss: 1.0920 - accuracy: 0.6438
Epoch 19/20
380/380 [==============================] - 2s 4ms/step - loss: 1.0689 - accuracy: 0.6513
Epoch 20/20
380/380 [==============================] - 2s 4ms/step - loss: 1.0424 - accuracy: 0.6594
```

Later, we evaluate our model by calculating the accuracy.

Finally, we evaluate our model and we create a predict y set. We convert our predictions to label and later we also convert the one-hot encoded test label to label.

We can notice that the accuracy is rising as the number of epochs increases.

Evaluation of results:

As for the assessment of the performance of our model we use some metrics like:

1. Accuracy = Total number of correctly predicted samples / total number of samples
2. Confusion Matrix
3. Precision = True Positives(TP)/ (TP + False Positives)
4. Recall = TP / (TP + FN)

Predictions and True values:

```
[8, 0, 0, 0, 0, 0, 0, 28, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Precision and Recall values:

```
Precision Score of the classifier is: 0.947387242856312
Recall Score of the classifier is: 0.9464504536485254
```