Homework 8
Email classification with Neural Networks

University of Thessaly

Evangelou Sotiris 2159
Kalais Konstantinos 2146
Chatziefremidis Leuteris 2209

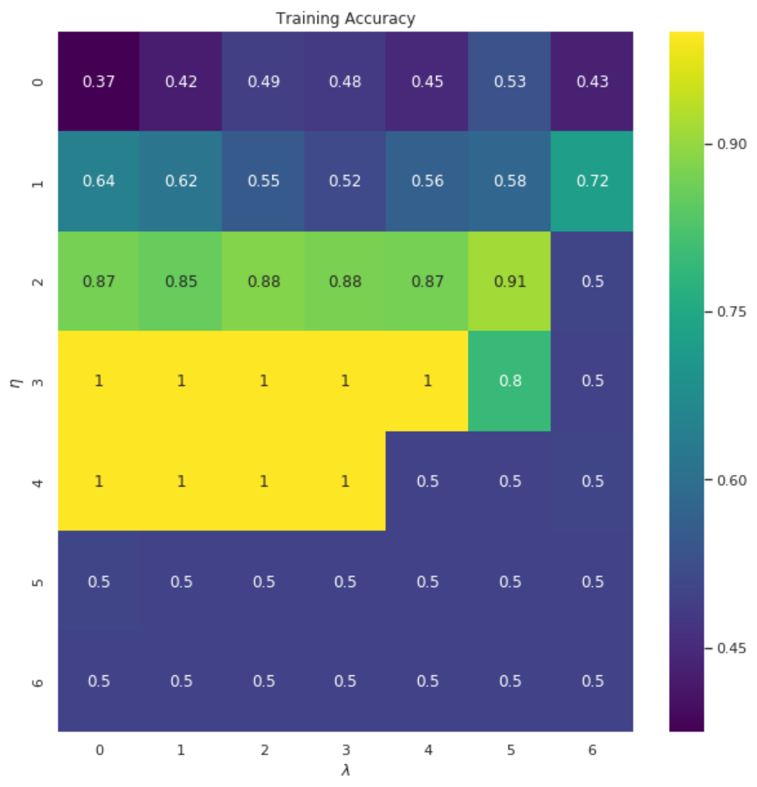
4. May 2019

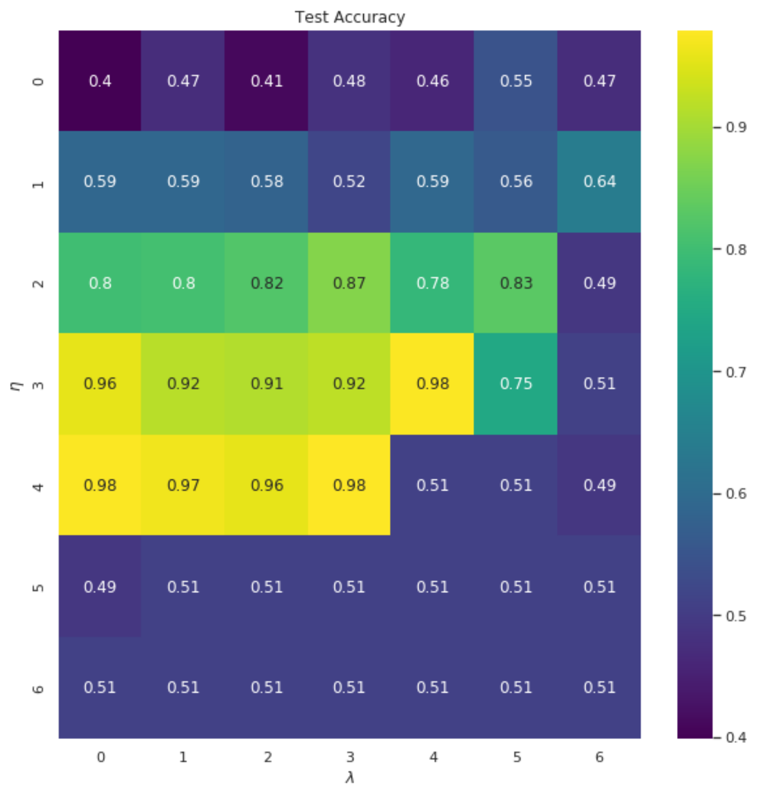
Abstract

In this project we will apply different types of neural networks to our dataset in order to classify emails as spam or not spam.We split our data to train(70%) and test(30%) in order to evaluate each type.We also tweak the parameters of each network and visualize the results.

1 Object-oriented implementation

In this part we create a neural network from scratch by implementing each part of it such as the feed forward phase and back propagation algorithm in order to adjust properly the weights.Moreover we tweak the learning rate and the lambda value in order to make our network more accurate.The results in both test and train are:





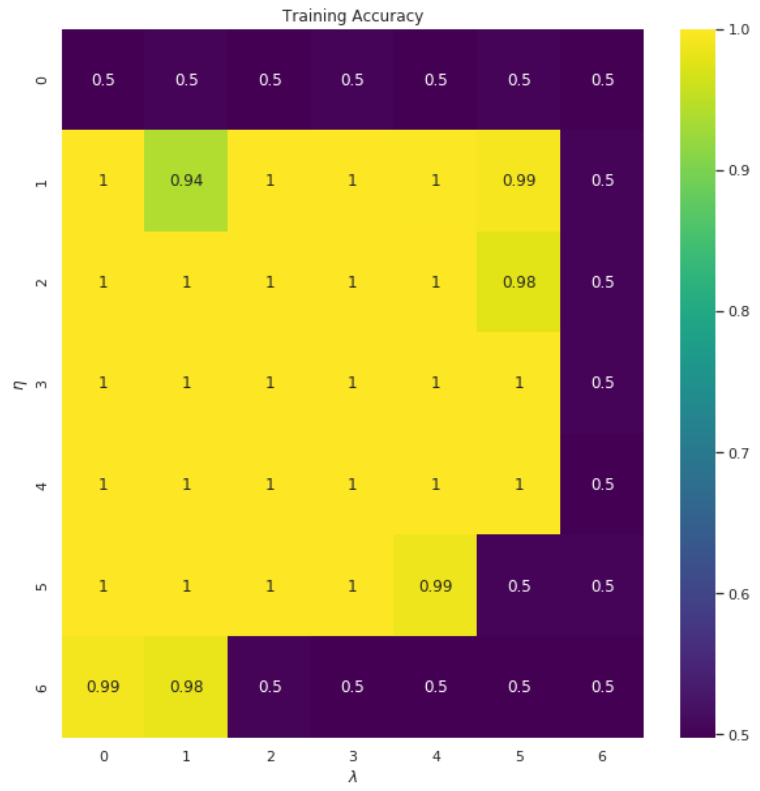
At the above figures we can see how the accuracy varies based on the learning rate and the lambda value.So we can choose properly the learning rate and the lambda value that fits our current dataset.

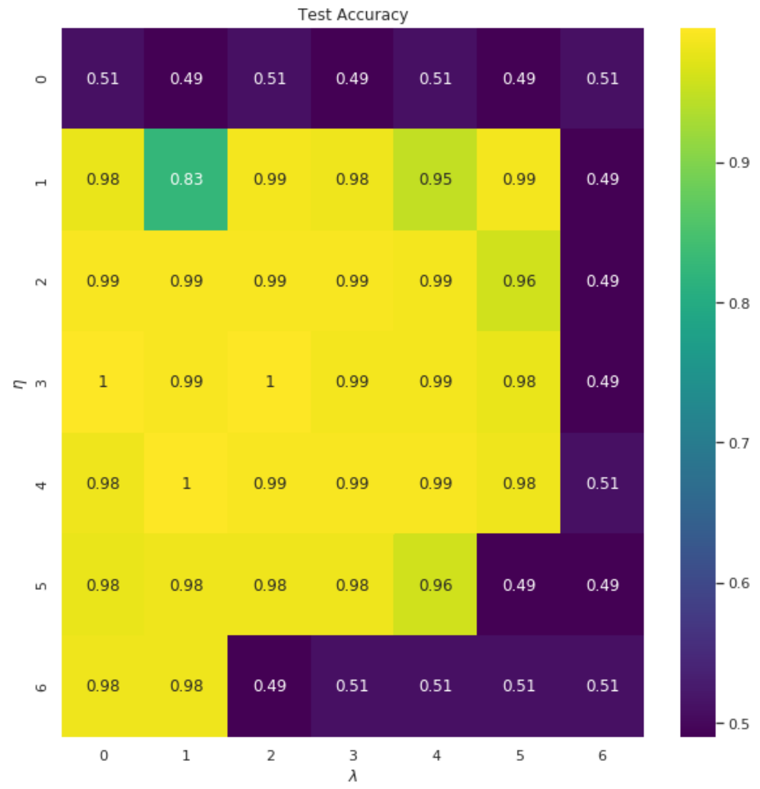
2 Scikit-Learn implementation

Scikit-Learn focuses more on traditional machine learning methods, such as regression, clustering, decision trees, etc. As such, it has only two types of neural networks: Multi Layer Perceptron outputting continuous values, MLPRegressor, and Multi Layer Perceptron outputting labels, MLPClassifier. We will see how simple it is to use these classes.

Scikit-Learn implements a few improvements from our neural network, such as early stopping, a varying learning rate, different optimization methods, etc. We would therefore expect a better performance overall.

In this case we will use the logistic as activation function and after tweaking again the learning rate and the lamba value we get the results for each phase:



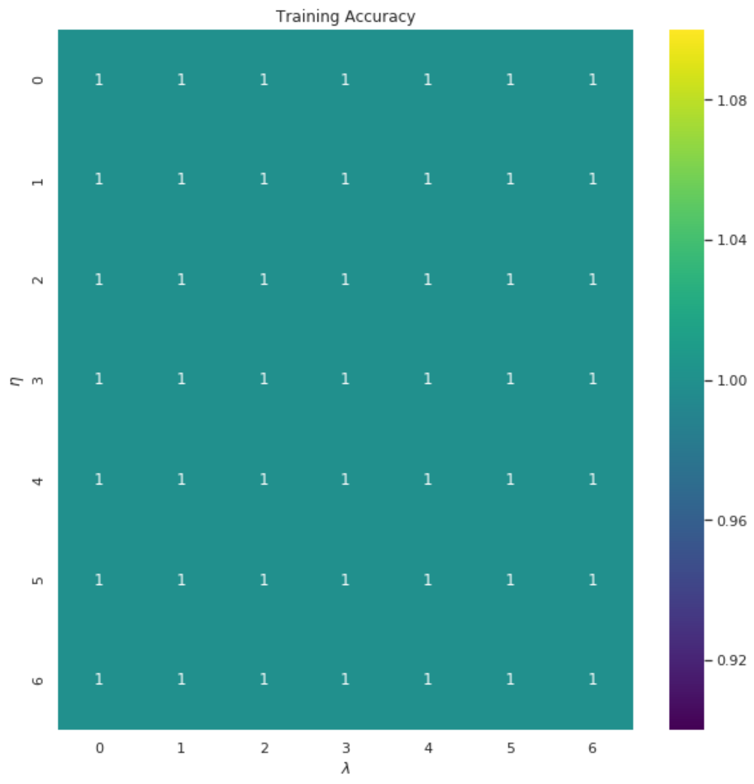


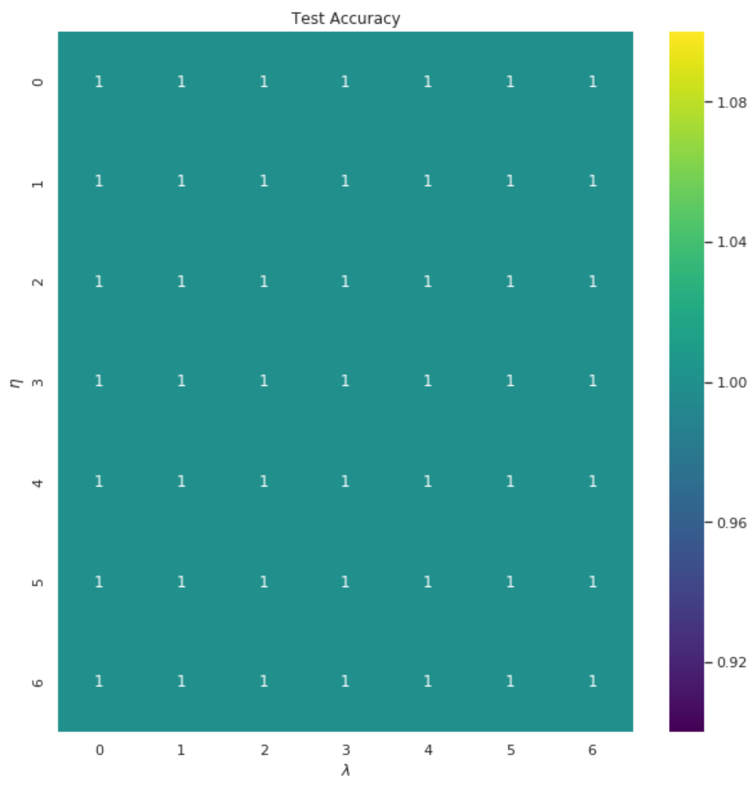
We observe that the results we got from the sklearn module are far better that the previous implementation with the same tweaking.But that was expected since the sklearn MLP Classifier differs from our implementation and is more sophisticated.

3 Tensorflow-Keras

Tensorflow is a computational framework that allows you to construct machine learning models at different levels of abstraction, from high-level, object-oriented APIs like Keras, down to the C++ kernels that Tensorflow is built upon. The higher levels of abstraction are simpler to use, but less flexible, and our choice of implementation should reflect the problems we are trying to solve.Keras is a high level neural network that supports Tensorflow backend.

First we use the modules and functions that tensorflow provides for us in order to implement from scratch a neural network.We build up all the methods that are needed to develope,train and test the neural network using only the components and the philosophy of the tensorflow.Then after tweaking the parameters we find the ones that fits our dataset.The results we get are:



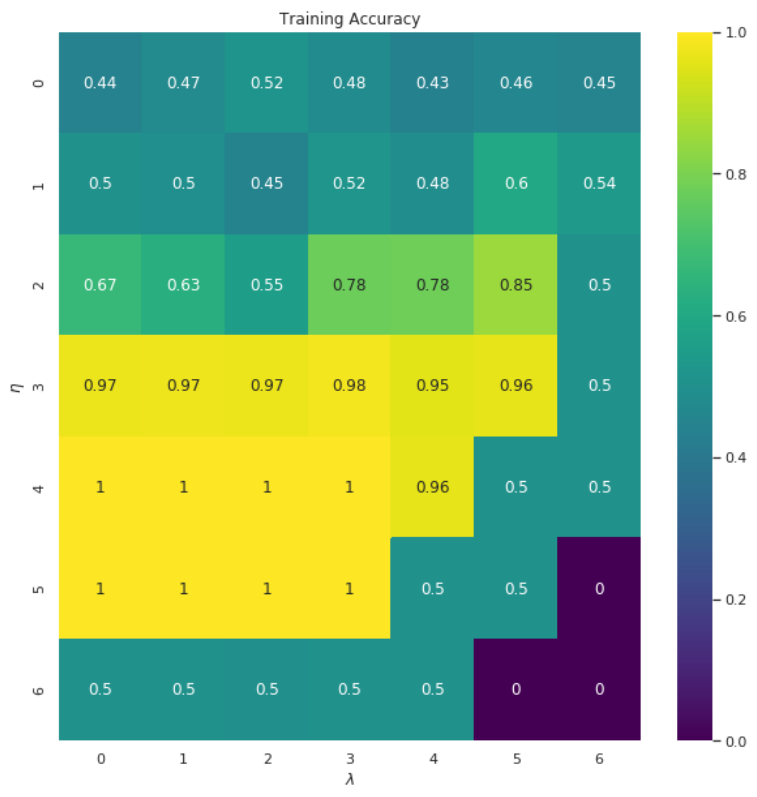


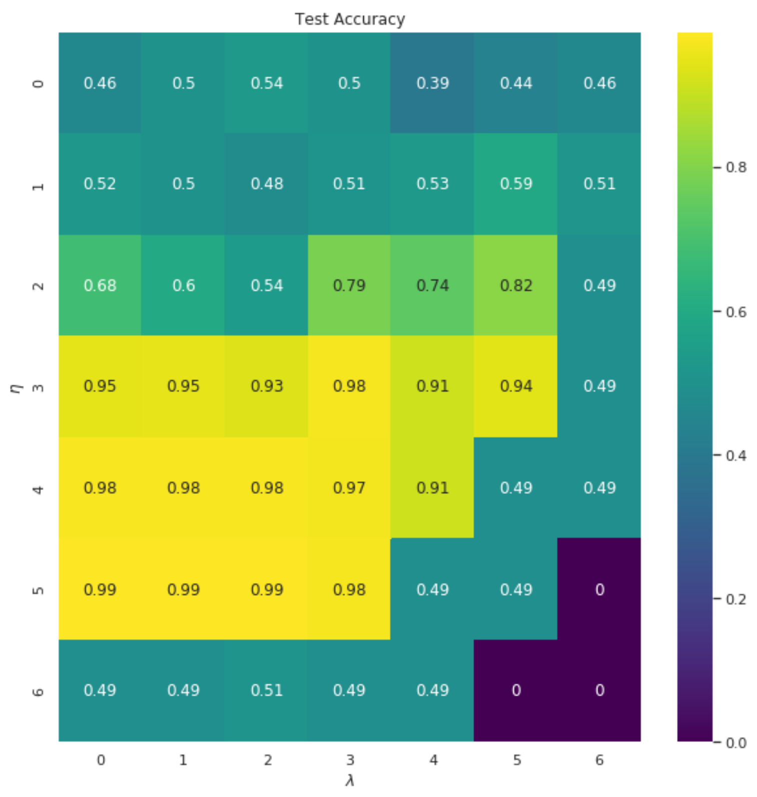
As we can observe the impementation with tensorflow fits our model perfectly.This is due to the fact that our dataset is not large enough.So our network can be 100 % accurate regardless of the parameters.

But if we use the API that Keras provides us in order to make the neural network the results are not so similar with the above.We create the below model:

```
model = Sequential()

model.add(Dense(n_neurons_layer,input_dim=trainX.shape[1]
    ,activation='relu', kernel_regularizer=l2(lmbd)))

model.add(Dense(n_categories, activation='sigmoid'))

sgd = SGD(lr=eta)

model.compile(loss='binary_crossentropy', optimizer=sgd
    ,metrics=['accuracy'])
```

The results that are exported from the above keras model are:





4 Convolutional Neural Networks

For the last part we can use two methods.The first includes only one 1D convolutional layer and the second two 1D convolutional layers and also one extra embedding layer with preprocessed data.

The model of the first method is the below:

```
model = Sequential()

model.add(Conv1D(5,1,input_shape=(trainX.shape[1]
      ,trainX.shape[2]),activation='relu',padding='same'))

model.add(Flatten())

model.add(Dense(5,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam'
      ,metrics=['acc'])

model.fit(trainX,trainY,validation_data=(testX,testY),epochs=30
      ,batch_size=1)
```

The other methods includes also and embedding layer that filters the data that we insert.Both have the same results which is an model with 100 % accuracy.

5 Comparing to other algorithms

If we compare our results with other algorithms that we used in older projects we can see that even if the neural networks are more sophisticated and complicated,they are also more accurate.Thats because they can adjust their weights in order to approach non linear behaviours that other algorithms fail to achieve.