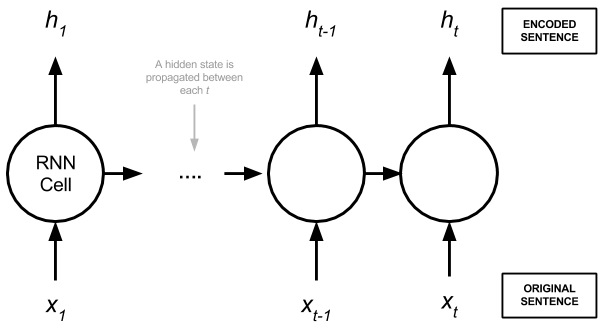
**CSE485-DATA MINING PROJECT**

**Movie Review Analysis in Turkish With RNN**



**Şuayb Talha Özçelik**

**215CS2018**

**03.05.2018**

**Data from Merve Yapnaz in Sentiment Analysis**

**Introduction and Problem Description**

The Movie Review Dataset in Turkis contains 19,999 highly-polar movie reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given movie review has a positive or negative sentiment.

We use pandas read\_csv method to Access the dataset and read it. The reviews is in the csv file with their consecutive numbers.We use RNN deep learning method train and test it.

We try to learn the review is positive or negative Our purpose is “Can we predict the the true result of review’s positivity or negativity with test(predict)?”.If we see the result is true , our hypothesis is correct.

**Data Description**

The Movie Review Dataset in Turkis contains 19,999 highly-polar movie reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given movie review has a positive or negative sentiment.

We will map each word onto a 50 length real valued vector. We will also limit the total number of words that we are interested in modeling to the 5000 most frequent words, and zero out the rest. Finally, the sequence length (number of words) in each review varies, so we will constrain each review to be 100 words, truncating long reviews and pad the shorter reviews with zero values.

Downloaded dataset in this page: <https://github.com/merveyapnaz/Sentiment-Analysist>

**Model Description**

The first layer is the Embedded layer that uses 50 length vectors to represent each word. The next layer is the LSTM layer with 64 memory units (smart neurons). Finally, because this is a classification problem we use a Dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (good and bad) in the problem.

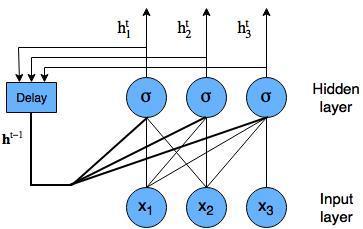
Because it is a binary classification problem, log loss is used as the loss function (**binary\_crossentropy** in Keras). The efficient ADAM optimization algorithm is used. The model is fit for only 2 epochs because it quickly overfits the problem. A large batch size of 64 reviews is used to space out weight updates. However,in ADAM the learning rate gradually decreases and at some point in time the system stops learning.So maybe we can use RMSprop optimization for reduce this problem.

**My RNN model:**

def RNN():  
 inputs = Input(name=**'inputs'**,shape=[100])  
 layer = Embedding(5000,50,input\_length=100)(inputs)  
 layer = LSTM(64)(layer)  
 layer = Dense(256,name=**'FC1'**)(layer)  
 layer = Activation(**'relu'**)(layer)  
 layer = Dropout(0.5)(layer)  
 layer = Dense(1,name=**'out\_layer'**)(layer)  
 layer = Activation(**'sigmoid'**)(layer)  
 model = Model(inputs=inputs,outputs=layer)  
 return model  
  
model = RNN()  
model.summary()  
model.compile(loss=**'binary\_crossentropy'**,optimizer=RMSprop(),metrics=[**'accuracy'**])  
history= model.fit(sequences\_matrix,y\_train,batch\_size=16,epochs=100,  
 validation\_split=0.2)

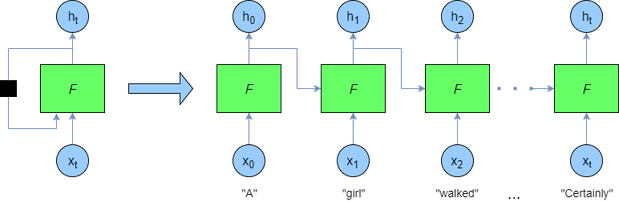
**What is RNN?**

A recurrent neural network is a neural network that attempts to model time or sequence dependent behaviour – such as language, stock prices, electricity demand and so on. This is performed by feeding back the output of a neural network layer at time *t* to the input of the same network layer at time *t + 1*. It looks like this:



Recurrent neural network diagram with nodes shown

Recurrent neural networks are “unrolled” programmatically during training and prediction, so we get something like the following:



Unrolled recurrent neural network

Here you can see that at each time step, a new word is being supplied – the output of the previous *F* (i.e. ht−1ht−1) is supplied to the network at each time step also.

**Used Optimizer in Algorithm (RNN)**

**Adam**

keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=**None**, decay=0.0, amsgrad=**False**)

Adam optimizer.

Default parameters follow those provided in the original paper.

**Arguments**

* **lr**: float >= 0. Learning rate.
* **beta\_1**: float, 0 < beta < 1. Generally close to 1.
* **beta\_2**: float, 0 < beta < 1. Generally close to 1.
* **epsilon**: float >= 0. Fuzz factor. If None, defaults to K.epsilon().
* **decay**: float >= 0. Learning rate decay over each update.
* **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

### RMSprop

keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=**None**, decay=0.0)

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

**Arguments**

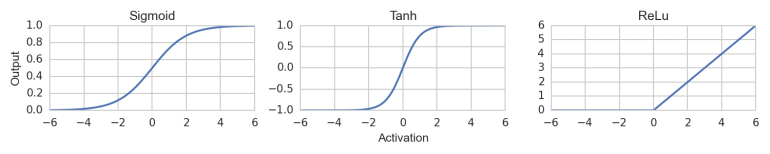
* **lr**: float >= 0. Learning rate.
* **rho**: float >= 0.
* **epsilon**: float >= 0. Fuzz factor. If None, defaults to K.epsilon().
* **decay**: float >= 0. Learning rate decay over each update.

**References**

* [rmsprop: Divide the gradient by a running average of its recent magnitude](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf)

**Used Activation for Normalizing (Sigmoid)**

In hidden layers (hidden layers) to be taken back (gradient decent to be calculated) (difference in learning back with the derivative) hidden layers (hidden layer) output is normalized with some activation functions. Some of these activation functions are sigmoid, tanch, ReLu, PreLu etc. given in figures. The most useful in them is ReLu. According to sigmoid, parameters can be learned more quickly. PReLu catches the negative values that ReLU misses; PReLu should be preferred if negative values are important for us.

We also use activation functions when we go back with a gradient descent. Our purpose in this kind of use is to be able to derive easy derivatives.

**Code Sources:**

<https://keras.io/layers/recurrent/>

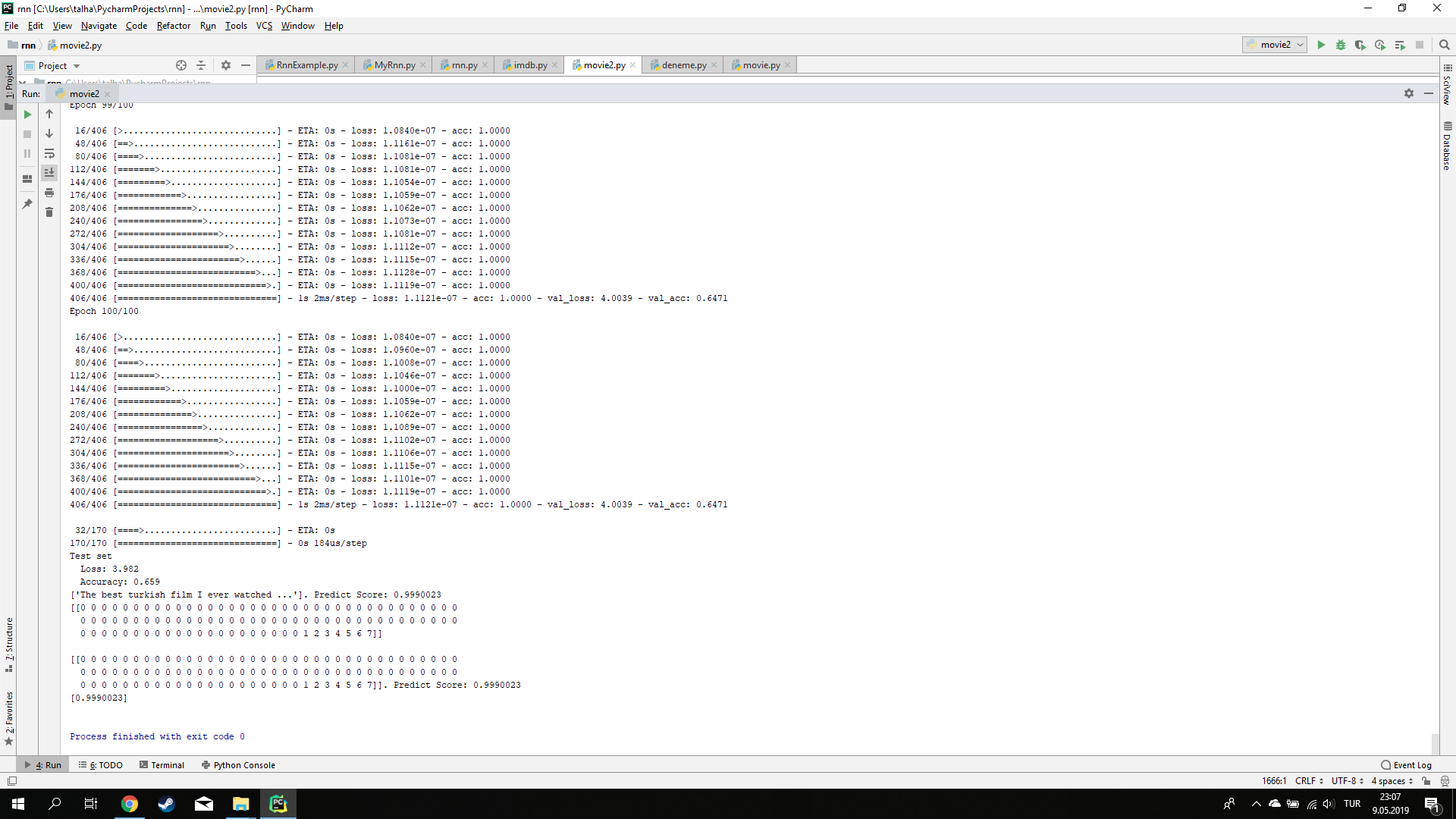
<https://medium.com/datadriveninvestor/multivariate-time-series-using-rnn-with-keras-7f78f4488679>

<https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17>

<https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/>

**Conclusion and Results:**

We try to understand RNN(recurrent neural network) and its structure. We try to classify the texts, reviews to positive or negative.We create a model, we choose our algorithms, algorithms’ methods and we train our model with these reviews then we test it.After then we predict the model.We try to see the results of these reviews according to positivity or negativity. If we can see the result is true, our hypothesis is correct or not.

**-Positive Sentence Example-**

Last two epochs are these:

16/406 [>.............................] - ETA: 0s - loss: 1.0840e-07 - acc: 1.0000

48/406 [==>...........................] - ETA: 0s - loss: 1.1161e-07 - acc: 1.0000

80/406 [====>.........................] - ETA: 0s - loss: 1.1081e-07 - acc: 1.0000

112/406 [=======>......................] - ETA: 0s - loss: 1.1081e-07 - acc: 1.0000

144/406 [=========>....................] - ETA: 0s - loss: 1.1054e-07 - acc: 1.0000

176/406 [============>.................] - ETA: 0s - loss: 1.1059e-07 - acc: 1.0000

208/406 [==============>...............] - ETA: 0s - loss: 1.1062e-07 - acc: 1.0000

240/406 [================>.............] - ETA: 0s - loss: 1.1073e-07 - acc: 1.0000

272/406 [===================>..........] - ETA: 0s - loss: 1.1081e-07 - acc: 1.0000

304/406 [=====================>........] - ETA: 0s - loss: 1.1112e-07 - acc: 1.0000

336/406 [=======================>......] - ETA: 0s - loss: 1.1115e-07 - acc: 1.0000

368/406 [==========================>...] - ETA: 0s - loss: 1.1128e-07 - acc: 1.0000

400/406 [============================>.] - ETA: 0s - loss: 1.1119e-07 - acc: 1.0000

406/406 [==============================] - 1s 2ms/step - loss: 1.1121e-07 - acc: 1.0000 - val\_loss: 4.0039 - val\_acc: 0.6471

Epoch 100/100

16/406 [>.............................] - ETA: 0s - loss: 1.0840e-07 - acc: 1.0000

48/406 [==>...........................] - ETA: 0s - loss: 1.0960e-07 - acc: 1.0000

80/406 [====>.........................] - ETA: 0s - loss: 1.1008e-07 - acc: 1.0000

112/406 [=======>......................] - ETA: 0s - loss: 1.1046e-07 - acc: 1.0000

144/406 [=========>....................] - ETA: 0s - loss: 1.1000e-07 - acc: 1.0000

176/406 [============>.................] - ETA: 0s - loss: 1.1059e-07 - acc: 1.0000

208/406 [==============>...............] - ETA: 0s - loss: 1.1062e-07 - acc: 1.0000

240/406 [================>.............] - ETA: 0s - loss: 1.1089e-07 - acc: 1.0000

272/406 [===================>..........] - ETA: 0s - loss: 1.1102e-07 - acc: 1.0000

304/406 [=====================>........] - ETA: 0s - loss: 1.1106e-07 - acc: 1.0000

336/406 [=======================>......] - ETA: 0s - loss: 1.1115e-07 - acc: 1.0000

368/406 [==========================>...] - ETA: 0s - loss: 1.1101e-07 - acc: 1.0000

400/406 [============================>.] - ETA: 0s - loss: 1.1119e-07 - acc: 1.0000

406/406 [==============================] - 1s 2ms/step - loss: 1.1121e-07 - acc: 1.0000 - val\_loss: 4.0039 - val\_acc: 0.6471

32/170 [====>.........................] - ETA: 0s

170/170 [==============================] - 0s 184us/step

Test set

Loss: 3.982

Accuracy: 0.659

['The best turkish film I ever watched ...']. Predict Score: 0.9990023

[[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

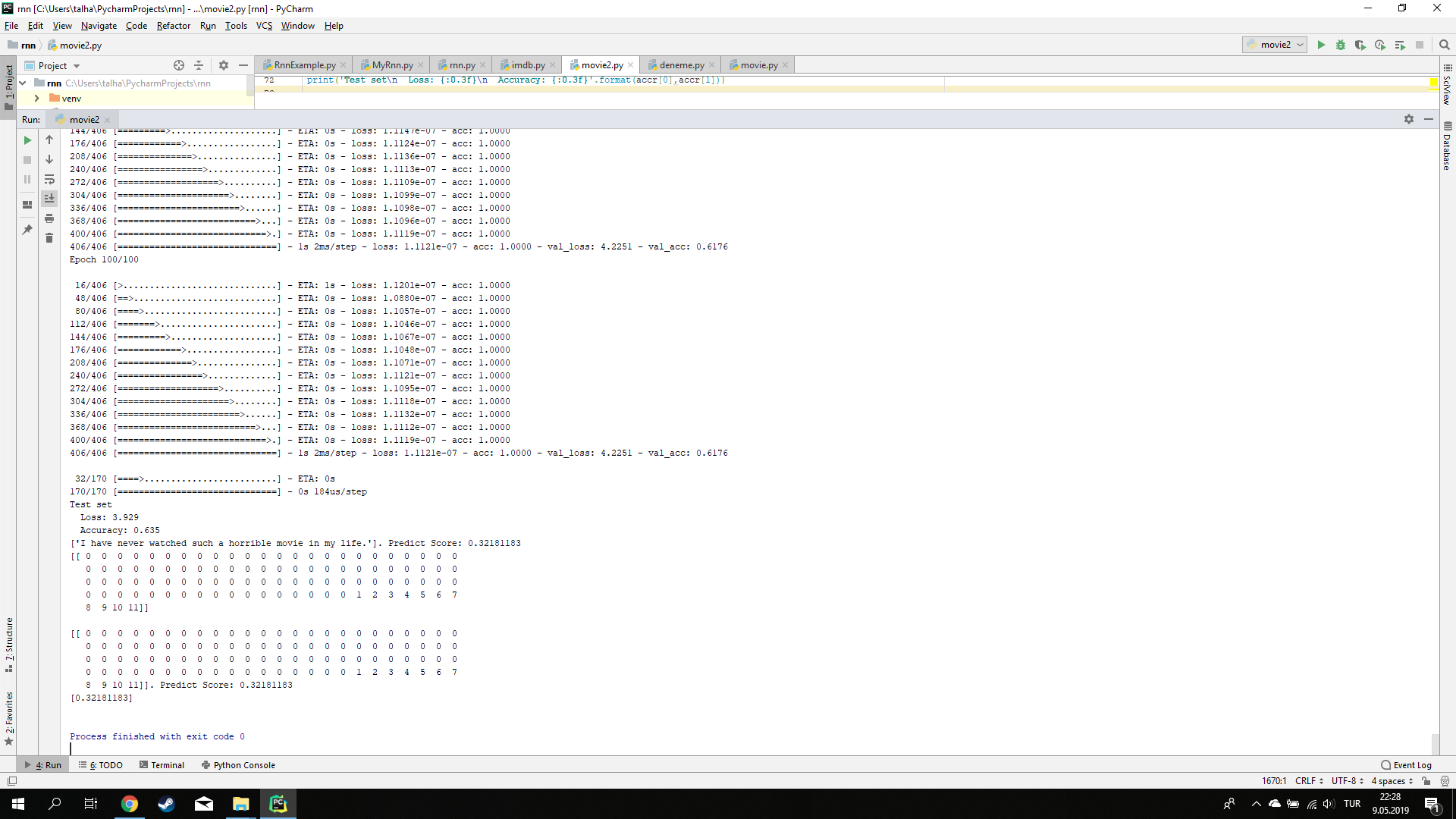
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7]]. **Predict Score: 0.9990023**

[0.9990023]

**If predict score close to 1, it means that the review’s positivity is more than negativity. If predict score close to 0,it means that the review’s negativity is more than positivity.**

**'The best turkish film I ever watched ...’ sentence’s predict score is 0.999023 this means that the review is most probably positive.**



Last two epochs are these: **-Negative Sentence Example-**

16/406 [>.............................] - ETA: 0s - loss: 1.1081e-07 - acc: 1.0000

48/406 [==>...........................] - ETA: 0s - loss: 1.1121e-07 - acc: 1.0000

80/406 [====>.........................] - ETA: 0s - loss: 1.1033e-07 - acc: 1.0000

112/406 [=======>......................] - ETA: 0s - loss: 1.1098e-07 - acc: 1.0000

144/406 [=========>....................] - ETA: 0s - loss: 1.1147e-07 - acc: 1.0000

176/406 [============>.................] - ETA: 0s - loss: 1.1124e-07 - acc: 1.0000

208/406 [==============>...............] - ETA: 0s - loss: 1.1136e-07 - acc: 1.0000

240/406 [================>.............] - ETA: 0s - loss: 1.1113e-07 - acc: 1.0000

272/406 [===================>..........] - ETA: 0s - loss: 1.1109e-07 - acc: 1.0000

304/406 [=====================>........] - ETA: 0s - loss: 1.1099e-07 - acc: 1.0000

336/406 [=======================>......] - ETA: 0s - loss: 1.1098e-07 - acc: 1.0000

368/406 [==========================>...] - ETA: 0s - loss: 1.1096e-07 - acc: 1.0000

400/406 [============================>.] - ETA: 0s - loss: 1.1119e-07 - acc: 1.0000

406/406 [==============================] - 1s 2ms/step - loss: 1.1121e-07 - acc: 1.0000 - val\_loss: 4.2251 - val\_acc: 0.6176

Epoch 100/100

16/406 [>.............................] - ETA: 1s - loss: 1.1201e-07 - acc: 1.0000

48/406 [==>...........................] - ETA: 0s - loss: 1.0880e-07 - acc: 1.0000

80/406 [====>.........................] - ETA: 0s - loss: 1.1057e-07 - acc: 1.0000

112/406 [=======>......................] - ETA: 0s - loss: 1.1046e-07 - acc: 1.0000

144/406 [=========>....................] - ETA: 0s - loss: 1.1067e-07 - acc: 1.0000

176/406 [============>.................] - ETA: 0s - loss: 1.1048e-07 - acc: 1.0000

208/406 [==============>...............] - ETA: 0s - loss: 1.1071e-07 - acc: 1.0000

240/406 [================>.............] - ETA: 0s - loss: 1.1121e-07 - acc: 1.0000

272/406 [===================>..........] - ETA: 0s - loss: 1.1095e-07 - acc: 1.0000

304/406 [=====================>........] - ETA: 0s - loss: 1.1118e-07 - acc: 1.0000

336/406 [=======================>......] - ETA: 0s - loss: 1.1132e-07 - acc: 1.0000

368/406 [==========================>...] - ETA: 0s - loss: 1.1112e-07 - acc: 1.0000

400/406 [============================>.] - ETA: 0s - loss: 1.1119e-07 - acc: 1.0000

406/406 [==============================] - 1s 2ms/step - loss: 1.1121e-07 - acc: 1.0000 - val\_loss: 4.2251 - val\_acc: 0.6176

32/170 [====>.........................] - ETA: 0s

170/170 [==============================] - 0s 184us/step

Test set

Loss: 3.929

Accuracy: 0.635

['I have never watched such a horrible movie in my life.']. Predict Score: 0.32181183

[[ 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 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 0 0 0 0 0 1 2 3 4 5 6 7

8 9 10 11]]. Predict Score: 0.32181183

[0.32181183]

**'I have never watched such a horrible movie in my life.’ sentence’s predict score is 0.32181183** **this means that the review is most probably negative.**

I try same statements a lot , human can see the review is negative and the code’s predict score is close to 0.In the positive reviews the situations are same but now the code’s predict score is close to 1.

**References:**

<https://keras.io/layers/recurrent/>

<https://sorucevap.deeplearningturkiye.com/t/turkce-duygu-analizi-veri-kumeleri/1478>

<https://medium.com/@hamzaerguder/recurrent-neural-network-nedir-bdd3d0839120>

<https://www.academia.edu/11969217/Sentiment_Analysis_in_Turkish_Media>