ISYE 6740 Project Report: Russian Trolls

Eugenio Andres Beaufrand III

GTID: ebeaufrand3 - Team ID: 168 - July 2020

1 Introduction and Problem Statement

2016 was a year marked by the historic election between Hillary Clinton and Donald Trump. However, surrounding this election was a lot of controversy. Not least among claims that Russians were leveraging their technological powers to influence the election was the notion that they were using troll social media accounts to influence our people. A troll is defined as someone who intentionally upsets or provokes people on the Internet by posting inflammatory, digressive, extraneous, or off-topic messages in an online setting. It was said that the purpose of these trolls was to stir sentiments in our people that might have an effect on the outcome of the election.

This project will be an analysis of whether or not we can create software that can automatically identify Russian troll posts on Twitter. The datasets used will be a Russian trolls dataset from Kaggle, and a sample of regular tweets pulled from a large online Twitter archive containing millions of tweets.

As this is a continuation of a project done in the Spring, in which the methods used were Logistic Regression, Naive-Bayes, and basic Neural Networks, this project will be a deeper look into Neural Networks. It will be explored how the performance from last semester can be improved by including user features, LSTM layers, and GRU layers.

2 Data

The data used for this project came from two sources:

1. Kaggle dataset containing the usernames and tweets of almost 3000 Twitter accounts believed to be connected to Russia's Internet Research Agency, a company known for operating social media troll accounts. Twitter released these usernames and tweets as a part of a House Intelligence Committee investigation into how Russia may have influenced the 2016.

   This dataset is in CSV format, and is composed of two separate CSVs, one containing tweet data and another containing user data.

   The most notable features are: text (the tweet body), screen_name, name, follower count, favorites count, statuses count.

2. Twitter archive of regular user tweets from July of 2016. This archive is vast enough that only a small proportion of these tweets needed to be sampled for purposes of this project.

   This dataset is in the JSON format and contains hundreds of JSON Files. Each article in a JSON file contains a tweet and the user data corresponding to this tweet.

   The most notable features are: text (the tweet body), screen_name, name, follower count, favorites count, statuses count.

*In order to see what the data looks like, please follow these links:

1.)Kaggle Set: https://www.kaggle.com/vikasg/russian-troll-tweets?select=tweets.csv
2.)Archive Set: https://archive.org/search.php?query=collection%3Atwitterstreamsort=-publicdate

3 Data Reading and Cleaning

1. Regular Tweet Reading and Cleaning

   In order to read the Regular Tweet Data, a sub-repository named Regular Tweets was created in the project repository containing 42 randomly selected JSON files of Regular Tweet Data from the hundreds of JSON files in the archive for the month of July 2016.

   With this repository in place, the data reading script (TweetsRead.py) uses the os package to read each file in the repository in sequence. The tweet data in each file is fed into a Python 2 dimensional array which results in our Regular Tweet DataSet. From a list of many features the following are kept:

   Data[i] = [Text,screen_name,name,follower_count,favorite_count,status_count]

   Finally all duplicate entries and rows containing null values were dropped. Additionally a class column was introduced and set to '0' for all regular tweets, indicating not a troll tweet.

2. Russian Troll Tweet Reading and Cleaning

   In order to read the Russian Troll Tweet Data, pandas pd.read_csv() function was called on the csv file containing the Russian Troll Tweets.

   Using the csv file containing the Russian troll user data, a dictionary was created of each user in the dataset with keys set to user id number. With this dictionary, a mapping was created from the tweet data to the user data, which was used to combine user data features into the tweet data.

   Finally, all duplicate entries and rows containing null values were droppped. Additionally a class column was introduced and set to '1' for all russian troll tweets.

3. Combining Regular and Russian Tweets

   In this last step, 100,000 and 25,000 regular and Russian tweets were sampled using pandas DataFrame.sample function. Now, the two samples are concatenated and form our Final DataSet.

*In order to see the scripts that read and clean the data please see: TweetRead.py,DataCleaning.py

4 Text Preprocessing and Input Generation

1. OneHot Method

   For model implementations in which OneHot Vectors were used as input to the model, the DataSet generated in the last section and a maximum vocab size was passed to the TextProcess function located in the file Project_Helper.py with Embedding set to False.

   All the tweets in our DataSet are stored in a 2D array and using Tokenizer from keras.preprocessing.text a one hot vector representation of each tweet is created. This means that each index in a vector corresponds to a word in a tweet. If the word is present in the tweet, the value of that index is 1, and otherwise it is 0.

   With this method the possibility of appending extra user features to the end of the embedding was explored. After the one-hot embedding, the columns 'follower_count', 'favorite_count', and 'status_count' were appended to the data to provide extra feature information to our model input.

2. Text Embedding Method

   For model implementations in which a text embedding was used as input to the model, the DataSet generated in the last section and a maximum vocab size as well as a length to pad sequences to is passed to the TextProcess function located in the file Project_Helper.py with Embedding set to True.

   All the tweets in our DataSet are stored in a 2D array and after applying one_hot() from keras.preprocessing.text on each tweet and after padding each tweet to the size determined by our function parameter, this matrix of padded tweets becomes an input to a Keras embedding layer.

*In order to see the script that preprocesses text please see: Project_Helper.py

5 Models

In this section, some of the different models architectures utilized to classify the data will be introduced:

1. MLPs

   Multi Layer Perceptron is the classic neural network. It consists of an input layer with the number of units equal to the number of features, fully connected hidden layers, and an output layer. Each neuron contains an activation function and has weights that are tweaked as the training takes place to achieve minimum loss.

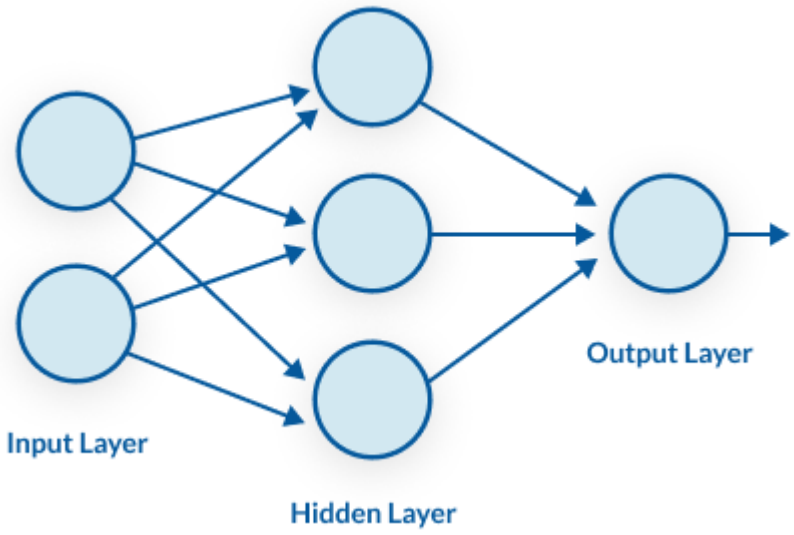


Figure 1: A standard Multi Layer Perceptron model.

   The process by which the gradient or maximum direction of decrease in the objective is found for the units is called backpropagation. The name of the algorithm which facilitates minimizing this loss function is called Stochastic Gradient Descent. Stochastic Gradient Descent is not guaranteed to find a global minimum, since the true function we are approximating is unlikely to be convex. However, if good results are found in practice, then the goal is achieved

2. LSTMs and GRUs

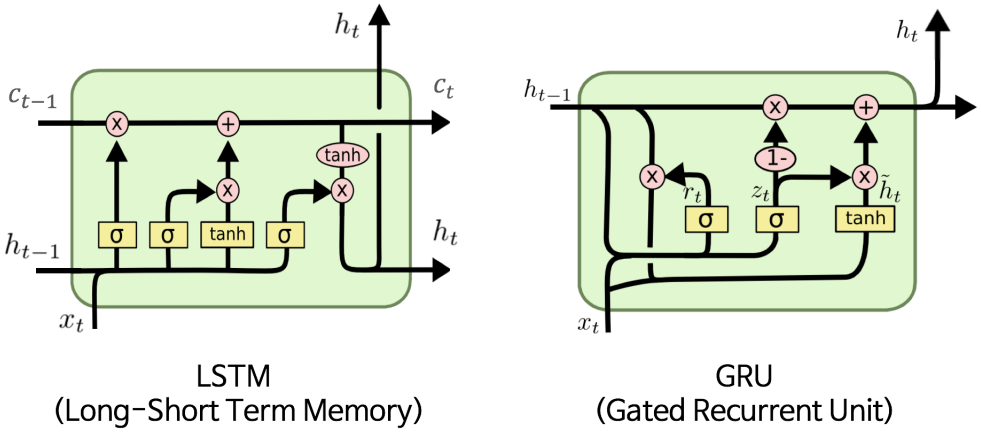
   Long short-term memory (LSTM) neural and Gated Recurrent Unit (GRU) networks are a type of recurrent neural network (RNN). LSTM and GRU networks aim to solve the short-term memory problem of many RNN model architectures. The short-term memory problem of RNNs is that as the length of a sequence increases, the model has difficulty carrying information from earlier in the sequence and keeping it in later steps. This is also called the vanishing gradient problem.

   LSTMs and GRUs attempt to keep the information at all parts of the sequence relevant by having internal mechanisms called gates that control information flow. These gates can learn which data in a sequence is relevant or important and keep that data reflected in the weights of the model.

   LSTMs make use of what is called a cell-state, which is the memory of the network, and various gates to control the carrying of relevant information throughout the entire sequence. The gates of an

LSTM are the forget gate, the input gate, and the output gate.

GRUs make use of what is called a hidden state and various gates to control the carrying of relevant information throughout the entire sequence. The gates of a GRU are the update gate, and the reset gate.



LSTM
(Long-Short Term Memory)

GRU
(Gated Recurrent Unit)

6 Model Performance

In this section the various types of models with given hyperparameter configurations which performed highly will be listed in order of ascending accuracy. In order to see the implementation of these Models please see: Main.py and Models.py

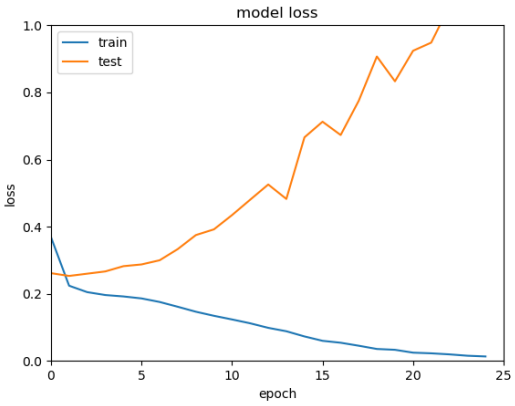
1. Embedding Layer-> 128 unit LSTM Layer->
   256 unit Dense Layer-> Output Layer

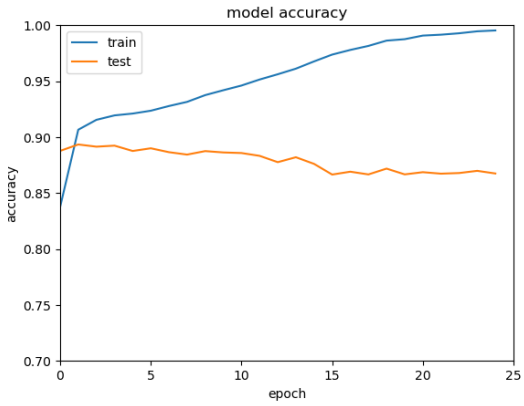
   vocab_size = 10,000, ExtraFeatures = False
   Embedding_dim = 30, pad_sequences_to = 20
   activations: LSTM - tanh, Dense - relu, Output - sigmoid
   optimizer, loss: adam, binary crossentropy

   Train Accuracy: 0.998
   Test Accuracy: 0.868

Confusion Matrix

actual value	0	1	total
0′	18348	1600	19948′
1′	1710	3342	5052′
total	20058	4942	





Comments: While the training accuracy for this model was good, the model suffered clear overfitting. The model assigned the correct label in 91.5% of cases for regular tweets and 67% of the time when it was a Russian tweet. Not a bad starting point but definitely with room for improvement.

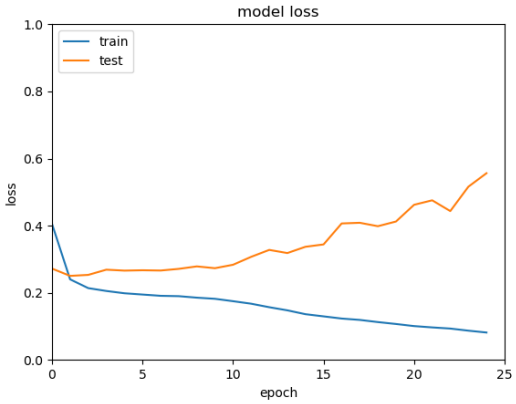
2. Embedding Layer-> 128 unit GRU Layer ->
256 unit Dense Layer -> Output Layer

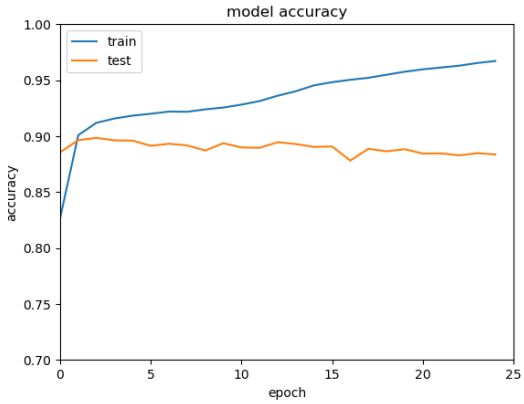
vocab_size = 10,000, ExtraFeatures = False
Embedding_dim = 30, pad_sequences_to = 20
activations: GRU - tanh, Dense - relu, Output - sigmoid
optimizer, loss: adam, binary crossentropy

Train Accuracy: 0.996
Test Accuracy: 0.871

Confusion Matrix

actual value	0	1	total
0′	18465	1548	20013′
1′	1672	3315	4987′
total	20137	4863	





Comments: Similar to the above LSTM with the same unit sizes. Again good training accuracy and overfitting. The model assigned the correct label in 92.2% of cases for regular tweets and 66.47% of the time when it was a Russian tweet.

3. Embedding Layer-> 128 unit GRU Layer (dropout(0.3),recurrent dropout(0.3))
->256 unit Dense Layer (dropout(0.3)) -> Output Layer

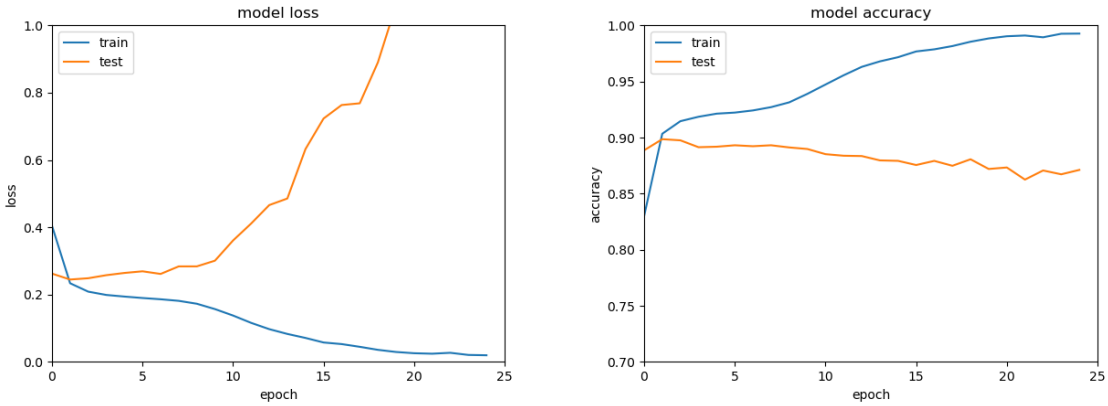
vocab_size = 10,000, ExtraFeatures = False
Embedding_dim = 30, pad_sequences_to = 20
activations: GRU - tanh, Dense - relu, Output - sigmoid

optimizer, loss: adam, binary crossentropy

Train Accuracy: 0.981
Test Accuracy: 0.884

Confusion Matrix

actual value	0	1	total
0′	18660	1348	20008′
1′	1559	3433	4992′
total	20219	4781	



Comments: This network was constructed after noticing the overfitting in the previously shown GRU. While it did have a modest impact on training accuracy, it was slight at only a percentage point.The model assigned the correct label in 93.2% of cases for regular tweets and 68.8% of the time when it was a Russian tweet.

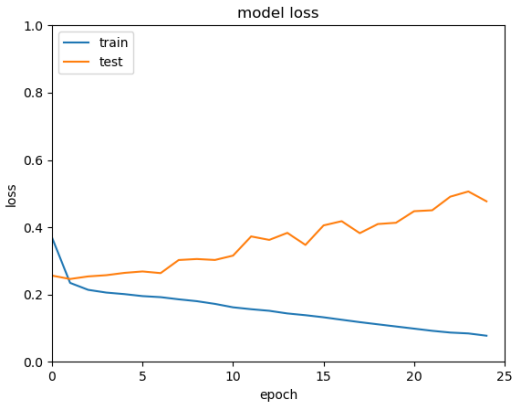
4. Embedding Layer-> 128 unit LSTM Layer (dropout(0.3),recurrent dropout(0.3))
->256 unit Dense Layer (dropout(0.3)) -> Output Layer

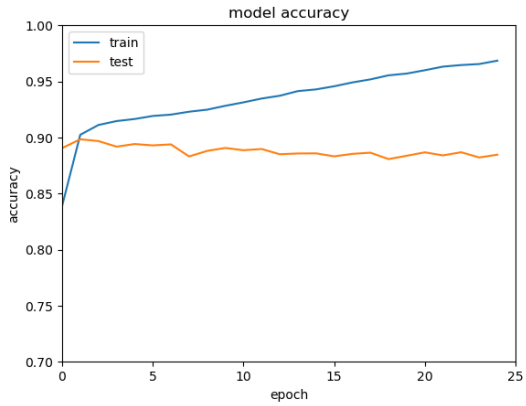
vocab_size = 10,000, ExtraFeatures = False
Embedding_dim = 30, pad_sequences_to = 20
activations: LSTM - tanh, Dense - relu, Output - sigmoid
optimizer, loss: adam, binary crossentropy

Train Accuracy: 0.982
Test Accuracy: 0.885

Confusion Matrix

actual value	0	1	total
0′	18661	1193	19854′
1′	1688	3258	4946′
total	20349	6139	





Comments: This network was constructed after noticing the overfitting in the previously shown LSTM. As with the GRU with dropout, while it did have a modest impact on training accuracy, it was slight at only a percentage point.The model assigned the correct label in 93.9% of cases for regular tweets and 65.8% of the time when it was a Russian tweet.

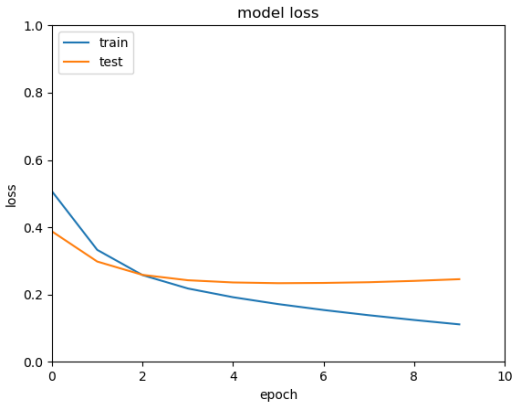
5. Embedding Layer -> Output

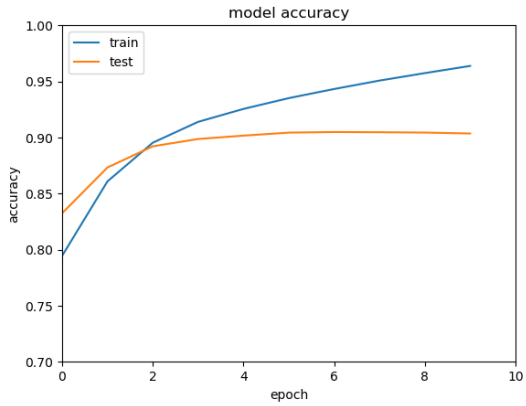
   vocab_size = 10,000, ExtraFeatures = False
   Embedding_dim = 30, pad_sequences_to = 20
   activations: Output - sigmoid
   optimizer, loss: adam, binary crossentropy

   Train Accuracy: 0.972
   Test Accuracy: 0.904

Confusion Matrix

actual value	0	1	total
0′	19149	890	20039′
1′	1518	3443	4961′
total	20667	4333	





Comments: In this network, simply the one-hot sequences were fed into the embedding layer to generate an output, and with more success than with the LSTM and GRU architectures attempted. It is worth noting that is validation accuracy of 90.4% is higher than the highest accuracy attained in last semesters project of 90.2%. The model assigned the correct label in 95.5% of cases for regular tweets and 69.4% of the time when it was a Russian tweet.

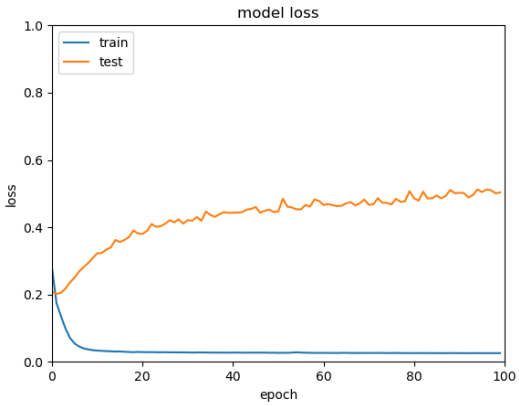
6. Input -> 4096 unit Dense Layer -> Output

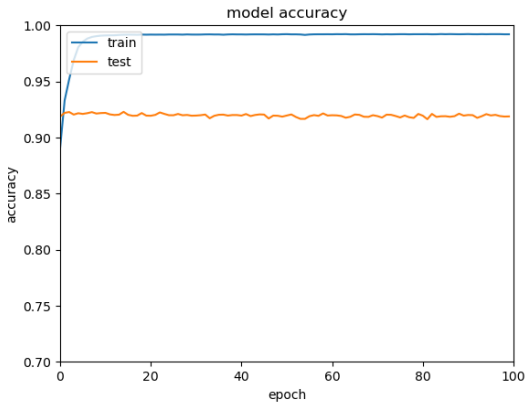
   vocab_size = 4,000, ExtraFeatures = False
   Embedding_dim = N/A, pad_sequences_to = N/A
   activations: Dense - relu, Output - sigmoid
   optimizer, loss: adam, binary crossentropy

   Train Accuracy: 0.993
   Test Accuracy: 0.919

Confusion Matrix

actual value	0	1	total
0′	19337	700	20037′
1′	1328	3635	4963′
total	20665	4335	





Comments: This network is the first network that departs from using word embeddings, and instead feeds one hot vectors directly into the network. A vocab size of 4,000 is used, and the network achieves the best results so far. There is still plenty of overfitting but the accuracy is moving in the right direction.The model assigned the correct label in 96.5% of cases for regular tweets and 73.2% of the time when it was a Russian tweet.

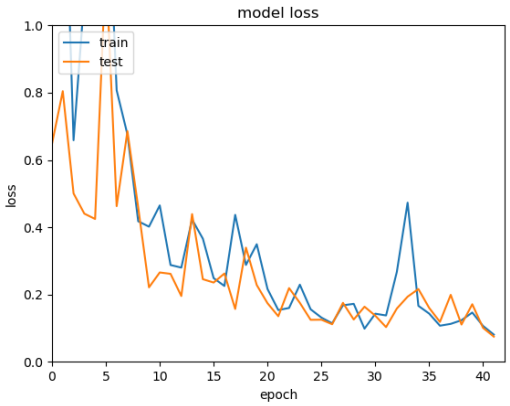
7. Input -> 2048 unit Dense Layer -> 512 unit Dense Layer -> Output

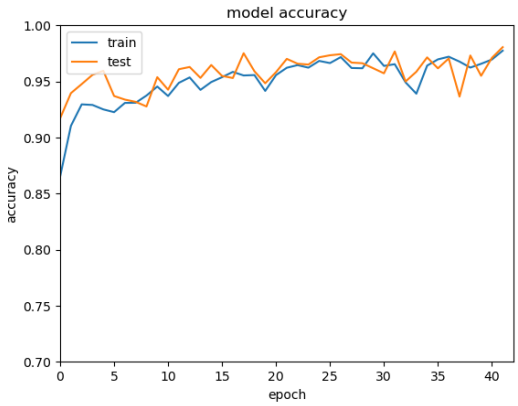
vocab_size = 5,000, ExtraFeatures = True
Embedding_dim = N/A, pad_sequences_to = N/A
activations: Dense - relu, Output - sigmoid
optimizer, loss: adam, binary crossentropy

Train Accuracy: 0.981
Test Accuracy: 0.981

Confusion Matrix

actual value	0	1	total
0′	19783	260	20043′
1′	224	4733	4957′
total	20007	4993	





Comments: This is the first network where the extra user features are finally introduced into the one hot vector input method. This is also where networks in a class of their own in terms of accuracy are found, with very high training and test accuracy and eliminated overfitting. The model assigned the correct label in 98.7% of cases for regular tweets and 95.5% of the time when it was a Russian tweet.

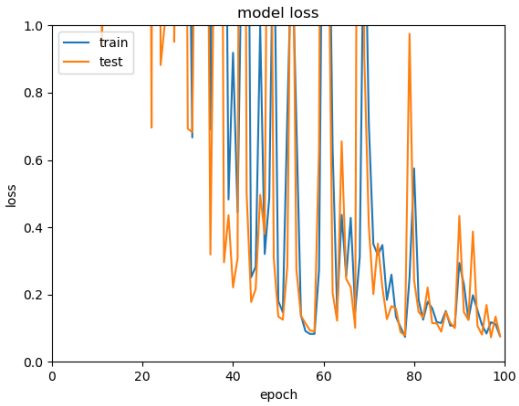
8. Input -> 4096 unit Dense Layer -> Output

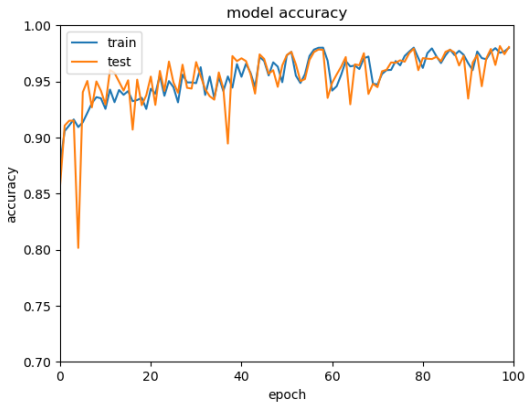
vocab_size = 4,000, ExtraFeatures = True
Embedding_dim = N/A, pad_sequences_to = N/A
activations: Dense - relu, Output - sigmoid
optimizer, loss: adam, binary crossentropy

Train Accuracy: 0.984
Test Accuracy: 0.981

Confusion Matrix

actual value	0	1	total
0′	19688	382	20070′
1′	99	4831	4930′
total	19787	5213	





Comments: This network just like the last performs extraordinarily well, with high training and test accuracy and overfitting eliminated. The model assigned the correct label in 98.1% of cases for regular tweets and 98.0% of the time when it was a Russian tweet.

*In order to see the implementation of these models, please see: Main.py, Models.py

7 Conclusions

The end results of training these models given the large sample size and high accuracy was finding that Russian troll tweets and regular user tweets can be discerned extraordinarily well, with likely even greater room for improvement. While it was expected that the Recurrent neural network architectures would perform more highly due to their reputation for being well suited for natural language processing tasks, this didn't turn out to be true. However, the expectation that implementing additional user features along with using text turned out to be highly effective, resulting in a final testing accuracy on the best models of 98.1%, which is a dramatic improvement from last semesters final testing accuracy of 90.2%.

Given the ability to yield such results, perhaps it is very reasonable to envision a future in which Twitter Data Scientists can effectively and rapidly delete the postings of troll accounts, and create a safer and higher quality platform for social interactions.

In the future it would be interesting to see if incorporating usernames and screen names as input features using either one hot encoding, text embedding, or perhaps some novel method would help continue pushing the accuracy of this model up.