Machine Learning Engineer Nanodegree
Capstone Project
Lilit Sargsyan
April 11th, 2018

##TOXIC COMMENT CLASSIFICATION

I. Definition

Project Overview

This project comes from the problem that abusive language, threats, and harassment can poison online conversations and as a result, make it difficult to share your thoughts online about the things you care about. The threat of online abuse and harassment results on many people stop to express themselves and give up seeking different opinions. According to Pew Research Center[1], 27% of American internet users chose not to post something online after seeing someone being harassed. Toxic language makes it hard to discuss important issues.
The Conversation AI[2] team, a research initiative founded by Jigsaw[3] and Google are working on tools to help improve online conversation. Their research aims to help increase participation, quality, and empathy in online discussions. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful, or otherwise are likely to make participant to live the conversation). One of the approaches to solving this problem involves people facilitating discussions, but this is time-consuming and requires a large workforce, that's why we need machine learning methods to do it.

---

[1] http://www.pewinternet.org/2017/07/11/online-harassment-2017/
[2] https://conversationai.github.io/
[3] https://jigsaw.google.com/

Problem Statement

The goal of this project is connected with finding toxic comments in online discussions. The final model will be able to detect toxic comments and assign to it a probability for each type of toxicity which are the following:

- Toxic
- Severe_toxic
- Obscene
- Insult
- Identity_hate

For this, we will use data from Kaggle.com Toxic Comment Classification Challenge[1]. It is a public dataset of comments from Wikipedia's talk page edits. The dataset contains a large number of Wikipedia comments whit the id which has been labeled by 6 toxicity sub-types (reasons why something might be considered toxic). The labeled annotations are based on asking 5000 crowd-workers to rate Wikipedia comments according to their toxicity (likely to make others leave the conversation).

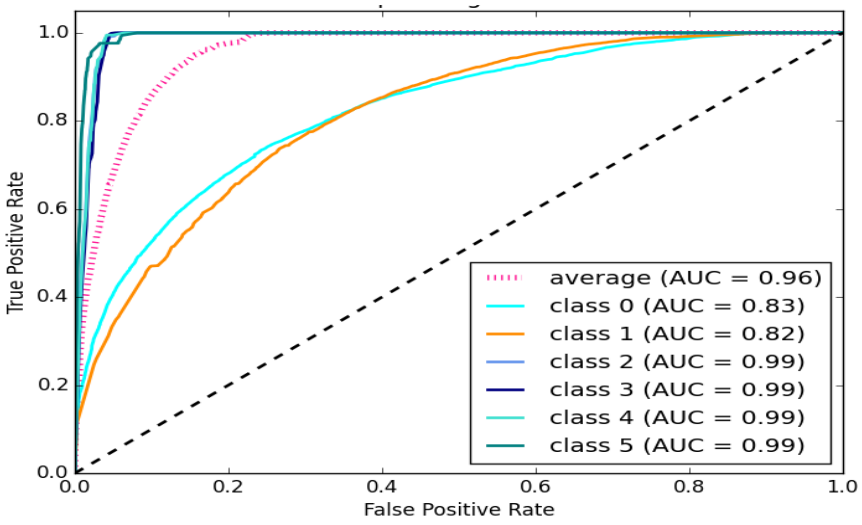
[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

Metrics

The evaluation metrics for my algorithm is ROC AUC[1] (area under the receiver operating characteristic curve). ROC is a set of {tp rate, fp rate} where tp is the true positive rate (positives correctly classified divided by total positives) and fp is the true negative rate (negatives incorrectly classified divided by total negatives).

		True class			
		p	**n**		
Hypothesized class	**Y**	True Positive	False Positive		tp rate = $\frac{TP}{P}$
	N	False Negative	True Negative		fp rate = $\frac{FP}{N}$
Column totals:		**P**	**N**		

ROC graphs are two-dimensional graphs in which tp rate is plotted on the Y axis and fp rate is plotted on the X axis. AUC is the area under ROC curve. Because we have multiclass classification with 6 classes the ROC graph will be something like this:



We want to have a higher rate for recognizing toxic comments and lower rate assigning nontoxic comments as toxic. That's why we will use this score b it'll show how well our algorithm is doing determining toxic comments (tp) and not toxic comments (fp), also because ROC curves are insensitive to changes in class distribution. If the proportion of positive to negative instances changes in a test set, the ROC curves will not change. Higher ROC AUC score means more toxic comments recognize as toxic and less nontoxic comments as toxic.

---

[1] http://people.inf.elte.hu/kiss/13dwhdm/roc.pdf

II. Analysis

Data Exploration

First, we will have a look at our data:

```
data.head(7) # have a look at the data
```

Out[1]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\r\nWhy the edits made under my use...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\r\nMore\r\nI can't make any real suggestions...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\r\n\r\nCongratulations from me as well, use ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0

So our data consist of id column, which we don't need and will get rid of later, comment_text column, which is our training data, and 6 columns for toxicity types and these will be our labels. From the labels columns, we can see that one comment can be assigned to the several toxicity types or any type at all.

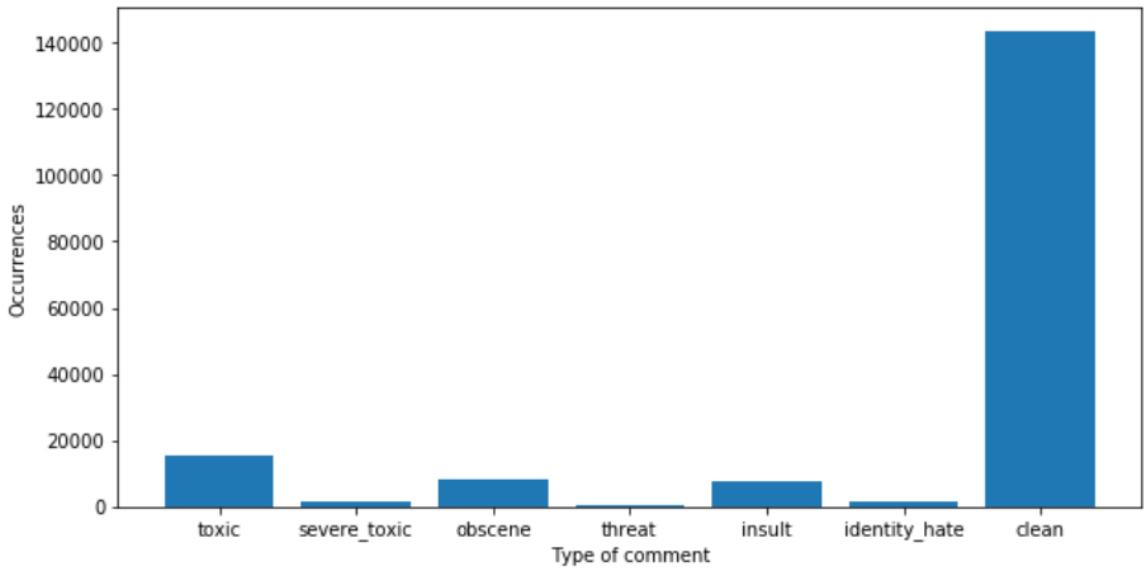
In the next step, we will have a look at the shape of our data and check if there are empty entries (rows with no information).
So as we can see from the image below there are 159571 rows with no missing values.

```
In [4]: data.shape #check how many rows are in data

Out[4]: (159571, 8)
```

```
In [5]: data.isnull().any() # check for empty entries

Out[5]: id              False
        comment_text    False
        toxic           False
        severe_toxic    False
        obscene         False
        threat          False
        insult          False
        identity_hate   False
        dtype: bool
```

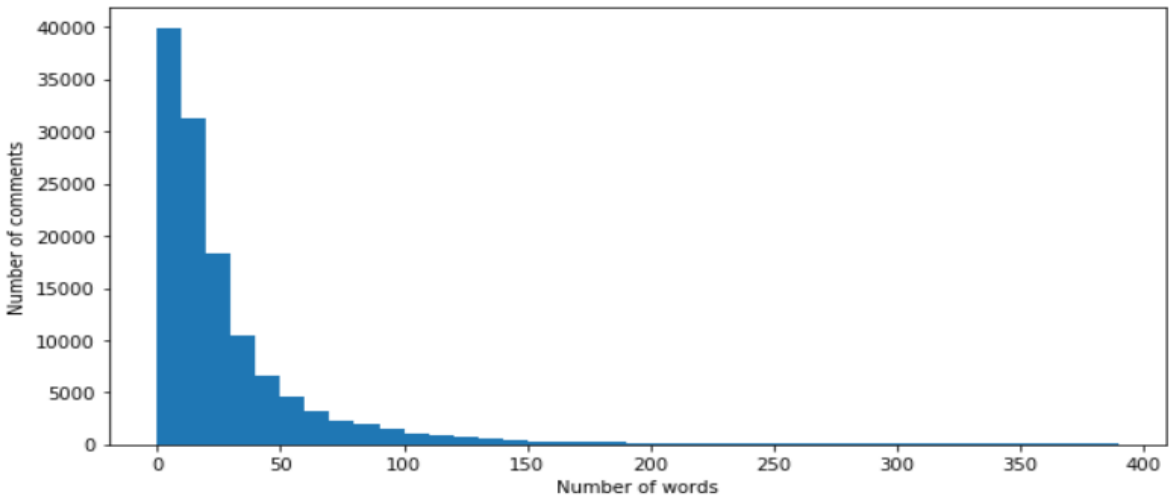
Exploratory Visualization

To visualize data distribution first we will add a new column for not toxic comments. This column will be one when all the other label columns are zero and zero otherwise. Now let's have a look at comments distribution
bars by their types:



As we can see from here our data is skewed and primarily consist of nontoxic comments. Most of the toxic comments are of type toxic that's because other types of toxic comments (like severe_toxic) can also be the type of toxic. So we have very unbalanced data even among toxic comments. This means if we predict clean for each comment our result still won't be so bad in term of accuracy that's why we using ROC AUC(how it's calculated is described in Metrics section).

Next visualization that we will need is for data preprocessing (we will describe it deeper in the III. Methodology part). We will plot a histogram to see how many words are there in each comment:



As we can see most of the comments contain less than 50 words, but because most of this comments are normal we will keep a length of 150 words for each comment to ensure that there is enough toxic word to train on our model.

Algorithms and Techniques

Deep Learning methods are proving very good in text classifications, achieving state of the art results for most of NLP(Natural language processing) tasks. The classifier is Word Embedding + RNN (Recurrent Neural Networks).

- The first layer will be hidden Embedding[1] layer.
  In an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from a text and is based on the words that surround the word when it is used.
  For Embedding layer 3 arguments required to be specified, these arguments are inpute_dim, output_dim, input_length (will go deeper here in Implementation section).
- The second layer will be LSTM (Long Short-Term Memory) layer.
  LSTM network trained using Back Propagation Through Time and overcomes the vanishing gradient problem. As such, it can be used to create large recurrent networks that can be used to address sequence problems (we not only need the words in comments but also their sequences to get better prediction results). Parameters to be

tuned are units (smart neurons), dropout, recurrent_dropout, return_sequences.

- The third layer is a GRU[3] layer.
  GRU is similar to LSTM but there are some differences one of which is that GRU exposes its full memory content that is seen or used by other units whereas LSTM uses controlled exposure. The parameters to be tuned is the same as with LSTM.(more about GRU layers can be found here)
- The fourth layer is a GlobalMaxPool1D layer.
- The last layer will be a Dense layer too with 6 output layers for each toxicity type and with the 'sigmoid' activation
- And depending on how well is our model doing we will add Dropout layers.
- After this next step will model compiling and fitting. hyperparameters to be tuned here are: optimizer, loss, metrics, epochs, batch_size, validation_split.
- We'll use callbacks to save weights with the best loss and print ROC AUC after each epoch.
- Then we'll do prediction on test data with the loaded best weights from the previous step.
- Get mean column wise ROC AUC for our model and keep tuning till we get our benchmark.

---

[1] https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
[2] https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[3] https://arxiv.org/pdf/1412.3555v1.pdf

Benchmark

So far there are publicly available models served through Perspective API for this problem to determine toxic comments but they don't show types of toxicity. So benchmarks for this project are:

- Be able to distinguish toxic comments from normal ones and show probability for each toxicity type for that comment.
- The leaderboard on Kaggle shows achieving ROC AUC score of 0.97 and higher which will be the target of our model.

III. Methodology

Data Preprocessing

We keep to kind of data one for our model and one for evaluating our data in kaggle.
For our model first we divide 'comment_text' column of our data into training and testing subsets and 'toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate' columns as training and testing labels using train_test_split from sklearn.model_selection library keeping 20% of data for testing. Now we have following subsets:

- comments_train (80% of data to train)
- labels_train (80% of labels to train)
- comments_text (20% of data to test)
- labels_test(20% labels to test)

For kaggle, we keep all data for training and then use testing data from kaggle to do predictions and check our score at kaggle. We can't use testing data from kaggle to evaluate our model because it doesn't have labels.
Farther we preprocessing comments_train, comments_test, kaggle_train and kaggle_test data sets.
We start data preprocessing by data cleaning. Split data using text_to_words_sequence() function from keras.preprocessing_text library which by default does three things:

- Splits words by space.
- Filters out punctuation.
- Convert text to lower case.

After this remove numbers and common words using stopwords from the nltk.corpus library.
Next step after data cleaning is tokenizing the comments i.e. turning sentences into numbers sequences where each number is word's index representation. WE using Tokenizer from keras.preprocessing_text library for this. Here we have to define the number of unique words (num_words) in our dictionary, we set this to 20000.For this, we first fit comments_train
to Tokenizer (to get an index for each word) then use text_to_sequences to turn sentences into numbers sequences.
The last step in data preprocessing is to make all comments of the same length and we already have discussed what length will be good for our data in exploratory visualization section. For this we use pad_sequences from keras_preprocessing_sequence with maxlen = 150. This will fill short sentences with zeroes in the end and cut long sentences from the end to make both short and long sentences of length 150.

---

https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

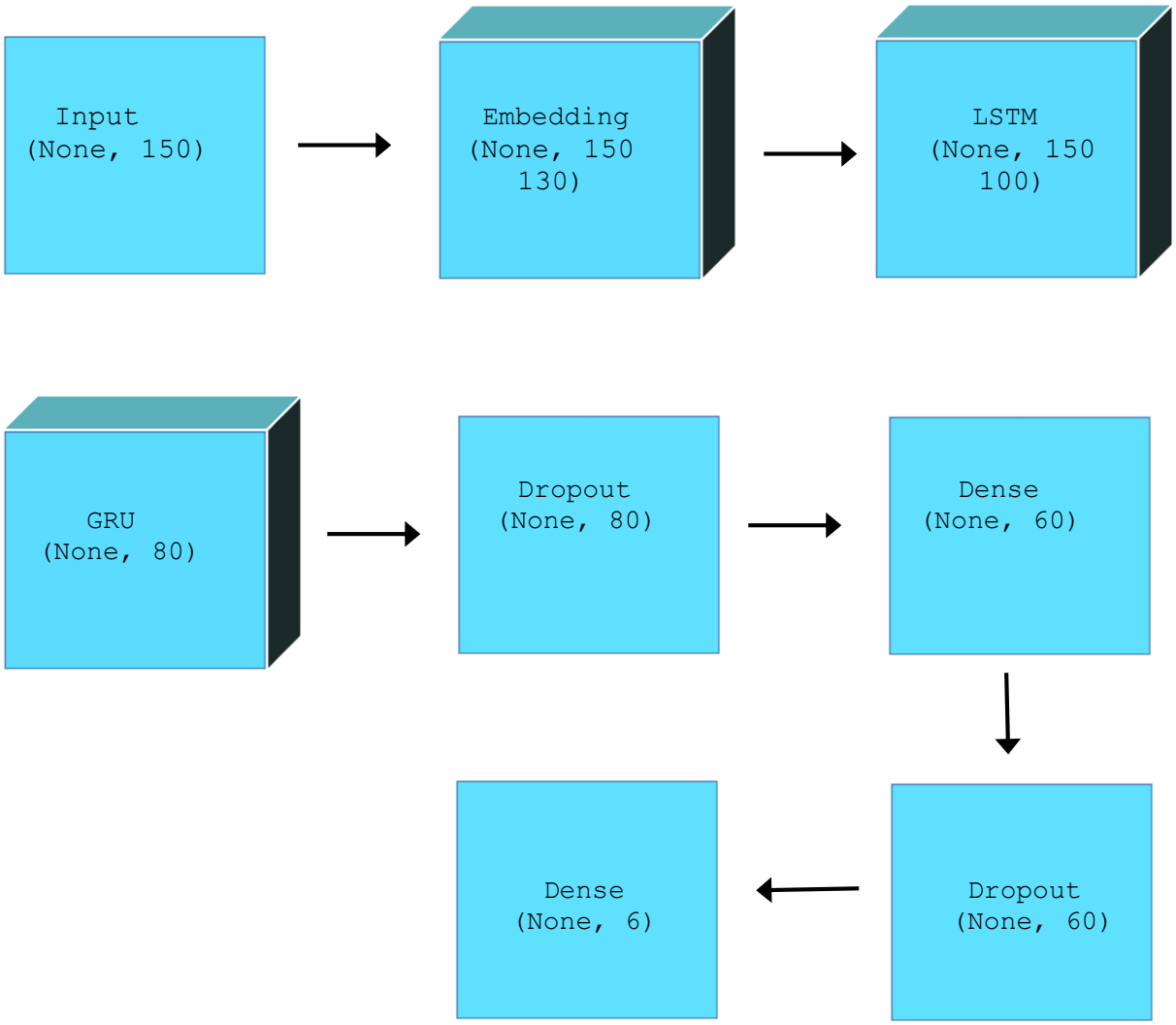
Implementation

Load training data into the data described in previous sections and testing data for kaggle submission into the test_data (we can't use this data for our model we can only make predictions on it for submission). After data loading, data exploration and exploratory visualizations (these steps are thoroughly described in Analyses section) we do data preprocessing as described in above section. After this data is ready to be trained.
We implement our architecture as described in Algorithms and Techniques section but we will use Bidirectional LSTM and Bidirectional GRU for second and third layers respectively. Bidirectional LSTM and GRU are extension of traditional LSTM and GRU that can improve model performance on sequence classification problems. These are the steps we take:

- The first layer of our neural network is Embedding layer. For this layer we need to give three parameters which are:
  - inpute_dim: This is the size of the vocabulary in the text data, in our case, it's 20000(look data preprocessing section when we chose the maximum number of words for tokenizing)
  - output_dim: This is the size of the vector space in which words will be embedding (we set this parameter equal to 130)
  - input_length: This is the length of the input, in our case, this is 150 (look data preprocessing section when we pad comments to the same length)
- Next comes Bidirectional LSTM layer with 100 units (dimensionality of output layer), return_sequences set to True, dropout set to 0.3, and reccurent_dropout set to 0.2.
- Next Bidirectional GRU layer with similar arguments as LSTM layer exempt with units of 80.
- Next MaxPool1d with pool_size = 2
- Next is GlobalMaxPool1D layer.
- After this Dropout layer.
- And the last layer is a Dense layer with output dimension of 6 because of our toxicity types are 6. Activation is 'sigmoid'. We use 'sigmoid' instead of 'softmax' because softmax forces the total predicted probabilities to some up to 1, meaning it won't be able to predict not toxic comments or multiple toxicity comments.
- We create the roc_auc_callback function to use in callbacks to show ROC AUC score after each epoch.
- Declare checkpointer to save weights with the best val_loss. For this, we use ModelCheckpoint from keras.callbacks
- Compile the model using 'adam' for the optimizer, 'binary_crossentropy' for loss function, because it independently optimizes each class and we have multi-label scenario, and for the metric, we use 'accuracy'.

- Fit the model using 10% of training data as validation data, epochs equal to 4 and batch_size of 30.

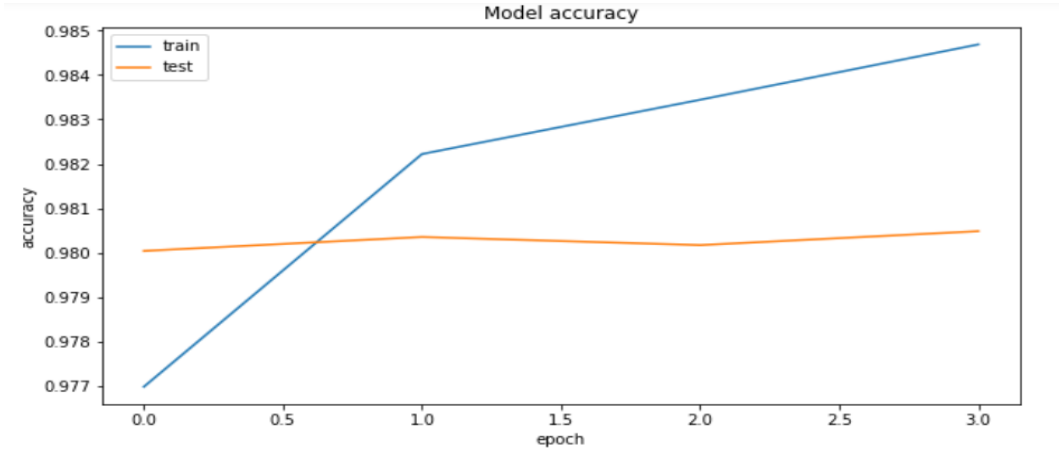
This is the architecture of the model:

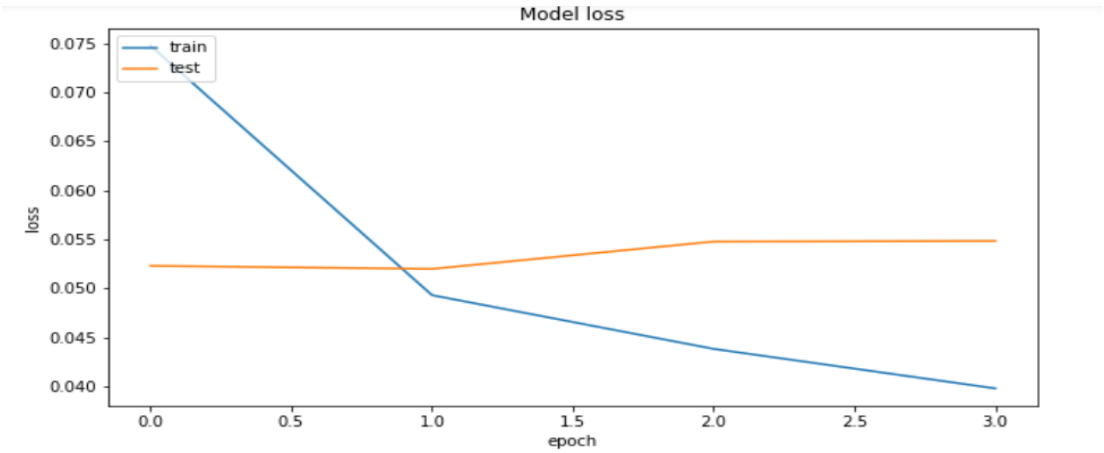


After this comes model tuning and evaluation steps which we will discuss in the next sections.

Refinement

As mentioned in the benchmark our first goal is to predict the probability of each 6 type of toxicity. We achieve this by using a Dense layer with output dimension of 6 (for each type) and 'sigmoid' activation (for probabilities).
Second, achieving ROC AUC score 0.97 and higher. One of the improvements was connected with the architecture. Before we use CNN layer after Embedding but that was giving us ROC AUC not higher than 0.96. Instead, we use LSTM as the second layer and that gave us a better result.
These are the plots of training the training/validation accuracy and losses.





Divergence indicates overfitting. Which we address by adding Dropout layers and reducing model complexity.

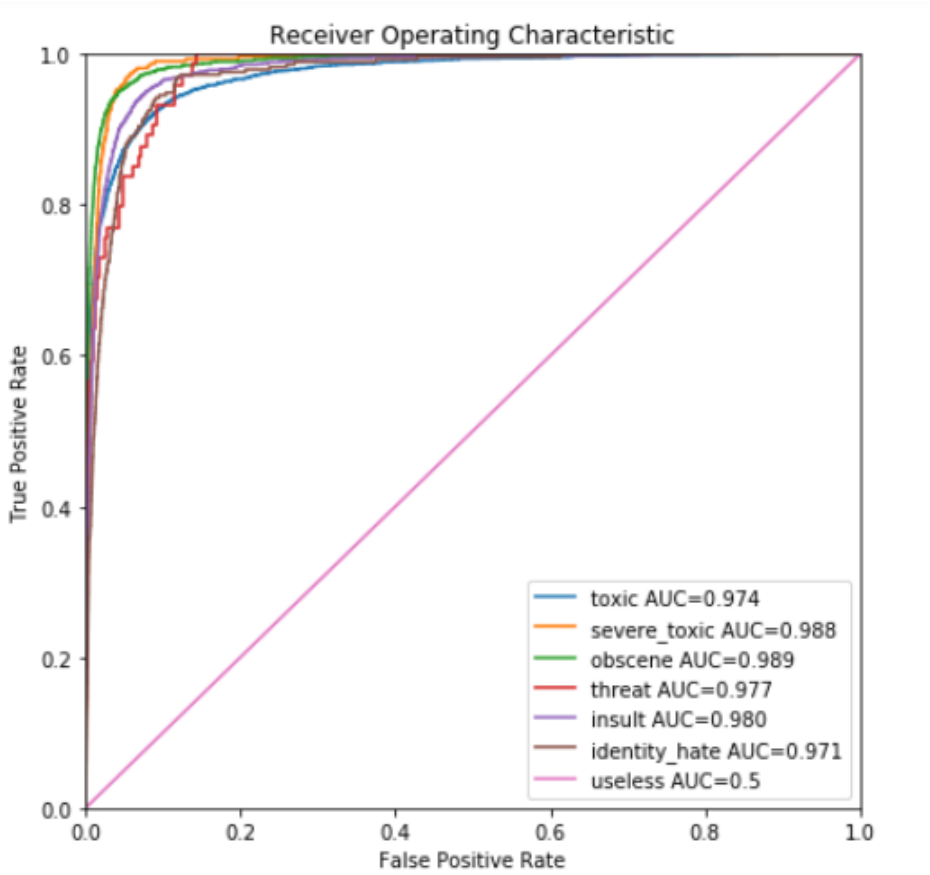
---

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

IV. Results

Model Evaluation and Validation

We used the validation set to evaluate our model during the training and testing set for final evaluation.
We achieve mean column wise ROC AUC score of 0.9797 using the architecture described in the Implementation section and preprocessed data described in the Data Preprocessing section. This architecture and hyperparameters where chosen because they performed best among the tried combination. The figure below displays ROC curves for each toxicity type for our model, along with corresponding AUC values. In contrast to this, the value of a useless model is 0.5 and displayed as a diagonal line. Model with this value (AUC=0.5) can be thought as it has the equivalent classification ability of flipping a coin.



Justification

Our first benchmark is to predict probabilities for each toxicity type. Let's see how our model will deal with this task by using test set:

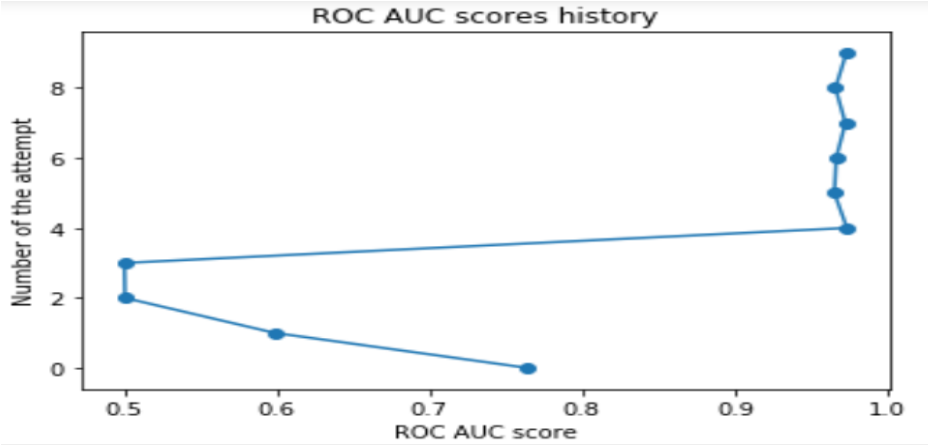
id	toxic	severe_toxic	obscene	threat	insult	identity_hate
00001cee341fdb12	0.994555	0.446482	0.987973	0.086714	0.859173	0.186006
0000247867823ef7	0.002174	0.000082	0.000505	0.000043	0.000540	0.000121
00013b17ad220c46	0.016832	0.000704	0.004090	0.000523	0.003863	0.000909
00017563c3f7919a	0.001495	0.000036	0.000279	0.000016	0.000294	0.000065
00017695ad8997eb	0.016660	0.000402	0.002618	0.000353	0.003385	0.000727

So as we can see the model gives us the probabilities for each type (this is an improvement from being able to just recognizing the toxic comments).
Next benchmark is mean column wise ROC AUC score higher or equal to 0.97. Our model gets 0.9797 and 0.972 in kaggle leaderboard when bigger test set is used.

V. Conclusion

Free-Form Visualization

As my ultimate goal was to get as high ROC AUC score as possible in kaggle leaderboard when predictions are made in the bigger test set with no labels (pad_kaggle_test), I'll use my scores from kaggle leaderboard for different submissions where different models where used. This is the history of my scores aggle leaderboard:



Reflection

The processes we went during this project can be summarized in the following steps:

1. Find a problem and relevant public data to solve it
2. Explore data and visualize
3. Based on previous step's conclusions clean and preprocess data
4. Set the benchmark for our model
5. Create an architecture and train using the data (multiple time until satisfied parameters where fount for the benchmark)
6. Get ROC AUC score as expected in the benchmark and a model capable of predicting probabilities for each toxicity type

I found steps 5 and 6 a bit difficult as I had to familiarize myself how different kind of neural networks layers are working and which one will be more appropriate. This steps where the most challenging and the most interesting, I've discovered many interesting properties of neural networks.

Improvement

One of the aspects of this model that can be improved are:

- To be able to recognize and predict probabilities for toxic comments in other languages as well
- This model can be used as a pre-trained model and be adjust to specific fields(healthcare, politics, science …), by doing this it can yield higher accuracy
- Another improvement can be done if we train our model to read text from the images and then predicted toxicity because toxic comments can be in the form of images as well.