Capstone Project Ahmed Eid
Machine Learning Nanodegree November 30th, 2019

Toxic Comment Detection in Online Discussion

Definition

Project Overview

Posting comments in online discussions has become an important way of sharing opinions and exercise one's right to freedom of expression on the web. This essential right is however under attack malicious users hinder otherwise respectful discussions with their toxic comments. A toxic comment is defined as a rude, disrespectful, or unreasonable comment that is likely to make other users leave a discussion. This problem is a sentimental classification problem under the NLP domain.
In this project, we will build a Recurrent model (LSTM) that will be able to classify the toxic comments and measure the degree of toxic ("severe_toxicity", "obscene", "threat", "insult", "identity_attack", and "sexual_explicit") in that comment.
In these approaches, we will use two types of data, first training data from Kaggle competition Jigsaw Unintended Bias in Toxicity Classification (data link), second-word embedding Glove(data link).

Problem Statement

Today, social media, blogs, online news platforms, and many other websites allow any web user to share his or her opinion on arbitrary content with a broad audience. The media business and journalists adapted to this development by introducing comment sections on their news platforms. Toxic comments are a problem for these platforms. First, they lower the number of users who engage in discussions and consequently, the number of visitors to their platform. As a result, an exchange of diverse opinions becomes impossible. With subscription models and ads as a way to earn money, a lower number of visitors means losing money. Second, legal reasons might require the platforms to deploy countermeasures against hate speech and to delete such content or not publish it at all. So here we are dealing with sequence classification problem that means we are going classify comments whether are toxic or not.
We will solve this problem by following sentimental analysis techniques such as clear data, convert object data to numerical, use a pre-trained model(Glove) as a word embedding, and build a recurrent neural network to solve this problem. Today there are more algorithms and techniques that make sentiment analysis more efficient like (LSTM) so I expected that will be a good solution for this problem.

Metric

Accuracy metric is one of the most important metric for evaluating classification model, informally, accuracy is the fraction of predictions our model got right.

$$\text{Accuracy} = \frac{\textit{Number of correct prediction}}{\textit{Total number of predictions}}$$

For binary classification, accuracy can be calculated in terms of positives and negatives as follows:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

where:
TP = true positive, TN = true negative, FP = false positive, FN = false negative

True Positive(TP): Reality: comment is toxicity Model said: comment is toxicity	False Positive(FP): Reality: comment is not toxicity Model said: comment is toxicity
False Negative(FN): Reality: comment is toxicity Model said: comment is not toxicity	True Negative(TN): Reality: comment is not toxicity Model said: comment is not toxicity

Analysis

Data Exploration

Data Background

At the end of 2017 the Civil Comments platform shut down and chose make their ~2m public comments from their platform available in a lasting open archive so that researchers could understand and improve civility in online conversations for years to come. Jigsaw sponsored this effort and extended annotation of this data by human raters for various toxic conversational attributes.

Data Description

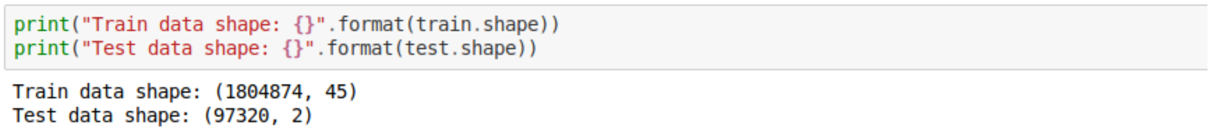
In the data, the text of the individual comment is found in the “comment_text” column. Each comment in Train has a toxicity label “target”, and models should predict the “target” toxicity for the Test data. This attribute (and all others) are fractional values which represent the fraction of human raters who believed the attribute applied to the given comment. For evaluation, test set examples with “target >= 0.5” will be considered to be in the positive class (toxic).

The data also has several additional toxicity sub type attributes, they are included as an additional avenue for research. But we will make it the second output for our model to measure the degree of toxic In the comment.(“severe_toxicity”, “obscene”, “threat”, “insult”, “identity_attack”, and “sexual_explicit”).

Comments have been labeled with a variety of identity attributes, representing the identities that are mentioned in the comment. The columns corresponding to identity attributes are (“white”, “black”, “homosexual_gay_or_lesbian”, “muslim”, “jewish”, “female”, “psychiatric_or_mental, “transgender”, “atheist”, “male”, “heterosexual”, “sexual_explicit”, “christian”, “latino”, “buddhist”, “asian”, and “hindu”, “bisexual”). We will sufficient with these highest toxic columns.

In addition the dataset also provides metadata from Jigsaw's annotation: “toxicity_annotator_count” and “identity_annotator_count”, and metadata from Civil Comments: “created_date”, “publication_id”, “parent_id”, “rating”, “funny”, “wow”, “sad”, “likes”, and “disagree”. In this project we will drop it from the data.

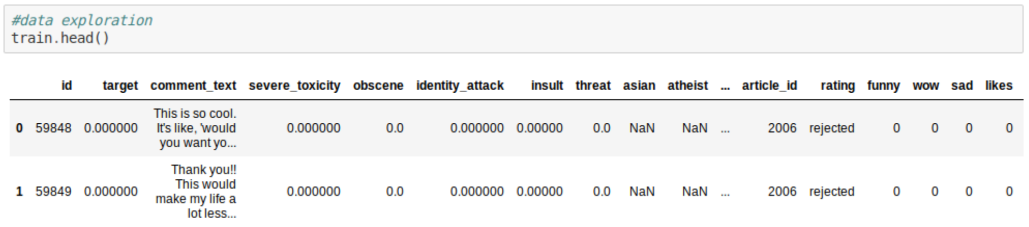
Data dimentionality



```
print("Train data shape: {}".format(train.shape))
print("Test data shape: {}".format(test.shape))
```

```
Train data shape: (1804874, 45)
Test data shape: (97320, 2)
```

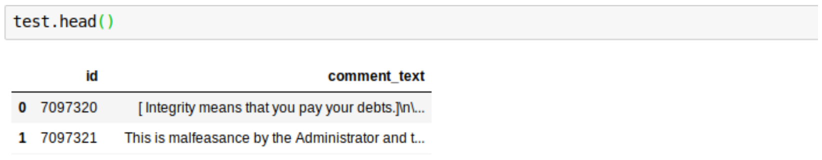
we have two data files, first(train.csv) contained training data its have ~1.8 M sample, each sample contained from id column, target column ,comment_text column, five toxicity degree columns, 25 identity columns, 12 metadata columns.



```
#data exploration
train.head()
```

	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat	asian	atheist	...	article_id	rating	funny	wow	sad	likes
0	59848	0.000000	This is so cool. It's like, 'would you want yo...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	...	2006	rejected	0	0	0	0
1	59849	0.000000	Thank you!! This would make my life a lot less...	0.000000	0.0	0.000000	0.00000	0.0	NaN	NaN	...	2006	rejected	0	0	0	0

Second(test.csv) it is have ~97K row, each row contain just id column, and comment_text column.

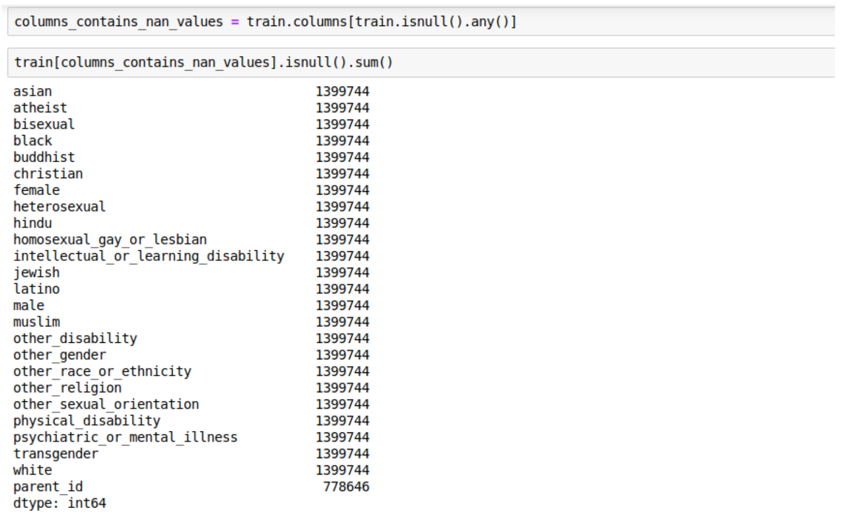


```
test.head()
```

	id	comment_text
0	7097320	[Integrity means that you pay your debts.]\n\...
1	7097321	This is malfeasance by the Administrator and t...

Due to this data belongs to Competetion, so the test file do not have labels, in this project we add it to use the comment_text column as a corpus with comment_text column in train file to fit the tokenizer on it, and we will split train file into three parts, one for training ,second for validation, third for testing.

Missing value checking



```
columns_contains_nan_values = train.columns[train.isnull().any()]
```

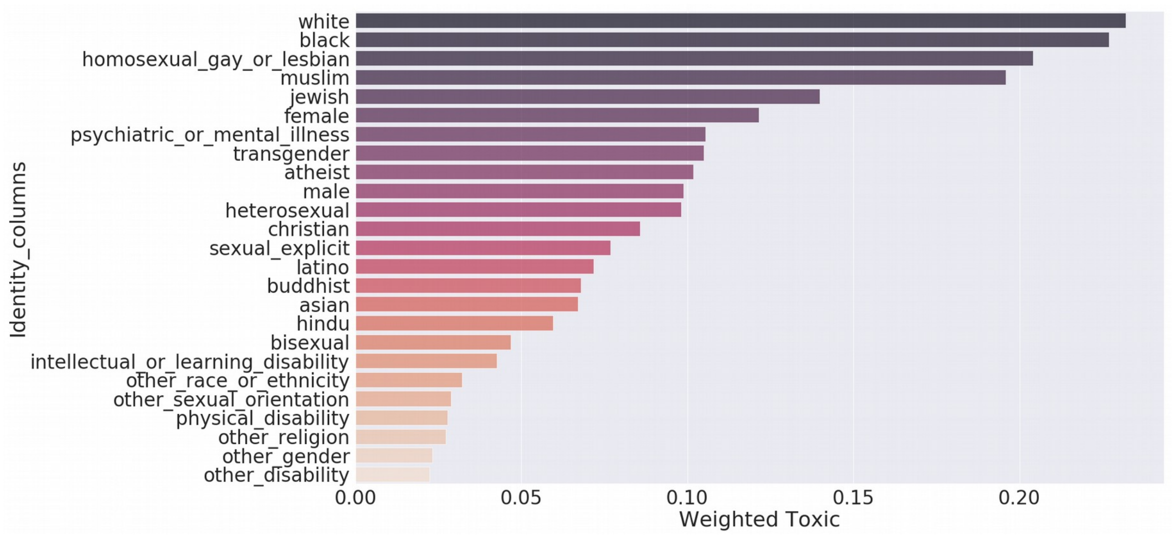
```
train[columns_contains_nan_values].isnull().sum()
```

```
asian                                    1399744
atheist                                  1399744
bisexual                                 1399744
black                                    1399744
buddhist                                 1399744
christian                                1399744
female                                   1399744
heterosexual                             1399744
hindu                                    1399744
homosexual_gay_or_lesbian                1399744
intellectual_or_learning_disability      1399744
jewish                                   1399744
latino                                   1399744
male                                     1399744
muslim                                   1399744
other_disability                         1399744
other_gender                             1399744
other_race_or_ethnicity                  1399744
other_religion                           1399744
other_sexual_orientation                 1399744
physical_disability                      1399744
psychiatric_or_mental_illness            1399744
transgender                              1399744
white                                    1399744
parent_id                                 778646
dtype: int64
```

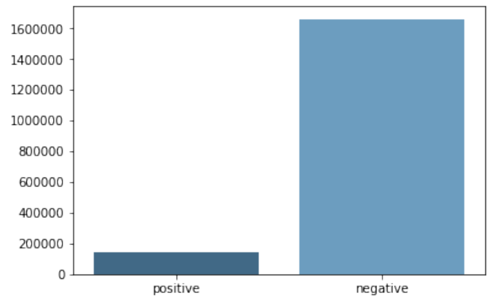
we note that from this checking no missing values in (target, comment_text, and toxicity_degree columns) and this is the major columns we are interested with, in the other hand all missing value found in identity columns which we will just use to build a sample_weight array that will determine the degree of importance of the sample for the model (only during training).

Exploratory Visualization

select the identity column with highest toxic weight to build sample_weight array, the plot below show the toxic weight for each identity column.



In the plot below we show the balanced between toxic class and not toxic class and it show that the different classes are not balanced, which mean that we must treatment this large gap between the two classes.



Algorithms And Techniques

Word Embedding:

embedding is mapping words to a high dimentionality semantic space, and the core problem that embedding solve is generalization, instead of the network needing to learn many disparate ways to handle disconnected input, we instead let similar words “share” parameters and computational paths.

word embedding is a class of approaches for representing words and documents using a dense vector representation. and their relative meanings. In word embedding, Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding. Word embedding techniques such as GloVe and Word2Vec have proven to be extremely efficient for converting words into corresponding dense vectors. Example:

	animal	human	plant	dangerous	nice
dog	0.95	0.0052	0.0422	0.76	0.30
boy	0.20	0.979	0.0324	0.042	0.53
lion	0.98	0.0471	0.023	0.89	0.07
man	0.06	0.98	0.003	0.08	0.78
tree	0.12	0.09	0.97	0.001	0.67
king	0.16	0.98	0.041	0.25	0.72

One-hot-encoding have many problem like very high dimentionality, and no relative meanings between words. So in this project we used “Glove.6B.300d” 300 dimensional word embedding.(data link)

Recurrent Neural Network:

Recurrent neural network is a deep learning technique use to learn from sequence data and make sentimental analysis, its decision at time step (t) affected by the decision taken in time step (t-1) and time step (t+1) if the network is bi-directional(which mean that is use backpropagation) which mean that the parameter which used in each time step is shared, so the recurrent neural network have two source of inputs “present”, “recent past”, and “future” which combined to determine the output(new input). Its often said that the recurrent nets have memory and adding this memory to neural networks for an important reason which is there is information in the sequence itself, and recurrent nets use it to perform a task that the feed-forward network can’t. But the simple RNN suffers from vanishing gradient problems when language can have very long-term dependencies, where it worked at this much earlier can affect what needs to come much later in the sentence. simple RNN not very good at capturing very long-term dependencies.

So in this project, we will use the SimpleRNN algorithm from Keras to build the simple benchmark algorithm.

And use the Long Short-Term Memory, or LSTM to build our model. LSTM is an updated version from SimpleRNN that is comprised of internal gates. Unlike another recurrent neural network, the network’s internal gates allow the model to be train successfully using backpropagation through time, or BPTT and avoid vanishing gradient problem.

The following parameter can be tuned to optimize the classifier:

- number of units
- number of layers
- activation function
- optimizer, loss function and metric
- number of epochs, batch size

additionally depend on the data Preprocessing.

Benchmark

I well use two option as a benchmark model firs is the Kaggle Competition’s Private Leaderboard score (“0.947”). to compare my model score with.
And I will add the second option because “testing” data provided by this competition has no label and I will evaluate my model with different test data, so I will create a simple model with SimpleRNN from keras and train it on the same data and epochs number, then compare my model score with it.

Methodology

Data Preprocessing

first I would like to notify you that the only data we will used in train, test, and validation is the data provided in “train.csv” file.
the processing of the data done be the following steps:

- drop unneeded columns(comment_id, metadata columns, and identity columns with low toxic weighted)
- select the identity columns that will used to build the sample_weight array
- due to the missing value just exist in identity columns, and the sample_weight will used only in the training process then we will take all sample that doesn’t has missing values as a training set(~405k samples) and use the remaining data(1.4M samples) to create the evaluation and testing set, because these two sets doesn’t effected by identity columns.
- due to the data is not balanced in training set I will split it into positive class(~46k) and negative class(~359k), then concatenate all samples in positive class with equivalent number (~46k) from negative class, resulting (~92k samples) in training set.
- Also the data not balanced in remaining set I will split it into positive class(~98k) and negative class(1.3M), then concatenate all samples in positive class with equivalent number (~98k) from negative class, resulting (~196k samples) in remaining set.
- Shuffle training and remaining set in random way.
- Take (20k samples) for evaluation set and another (20k samples) for testing set, from remaining set.
- Split every data set training, evaluation, and testing into features and labels:
  - ✔ features : x_train, x_validate, x_test
  - ✔ first output : y_train, y_validate, y_test
  - ✔ second output : aux_y_train, aux_y_validate, aux_y_test
- create text corpus from all comment in train.csv file and test.csv file, and fit the tokenizer on this corpus, with num_words = 100k.

- Due to the the embedding layer accept fixed length and it expect the sentence with equal size, however our encoded sentence are of different size. One way to make all the sentences of uniform size is to increase the length of all the sentences and make it equal to the length of the largest sentence. First calculate the length of the largest sentence(347) to pad all sentence to these length. I think that 240 is good.
- Convert text in features(x_train, x_validate, x_test) to sequence(comment to numerical vector).
- Pad sequence to fixed length(240)
- load word embedding file(Glove) as a “glove_file” and open it to create a dictionary that will contain words as keys and the corresponding 100 dimensional vectors as values, in the form of an array.
- Create an embedding_matrix, we want the word embedding for only those words that are present in our corpus,We will create a two dimensional numpy array of 409328 (size of vocabulary) rows and 300 columns. The matrix will initially contain zeros.
- Generate the Sample Importance (Sample_weight) array using IDENTITY_COLUMNS (Note: I’m trying to build this array but I failed so I take the code for this section from Kaggle and use it)

Implementation

In the implementation we will use the Keras functional API because we need to create a model with two output main output (target: toxic or not) second output(degree of toxic).
The Implementation process can be split into two stages:

1. Benchmark Model (SimpleRNN) implementation
2. Our main Model (LSTM) implementation

Each stage split to:

- build the model
- compile model
- train the model

1. Benchmark Model(SimpleRNN):

In this stage we build a very basic and simple model that is contained from:

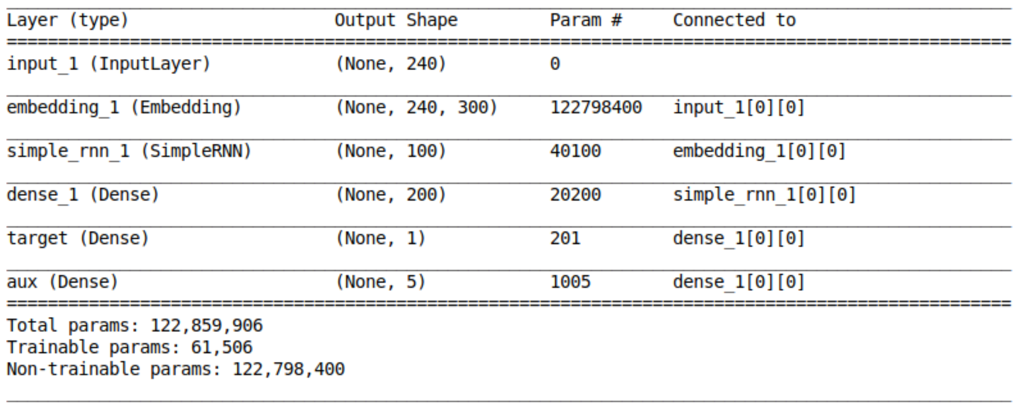
- input layer with shape=240
- embedding layer can be only used as a first layer, take positive integer(index of the word) and turn it to its related vector in embedding_matrix, and take following parameter:
    - input_dim: size of the vocabulary, in this project(409328)
    - output_dim: dimension of dense embedding, (300)
    - weights: embedding matrix
    - trainable: false
    - input length: take this parameter from the Input_layer(240,)
- SimpleRNN layer which is fully-connected RNN where the output is to be fed back to input. take the following parameter:
    - units: Positive integer, dimentionality of the output space.(100)
    - activation: activation function,it is a node added to the output layer or between two layers of any neural network it is also known as the transfer function it is used to determine the output of neural network layer in between 0 to 1 or -1 to 1, default “tanh” and we make it as default.
    - Input: it is input is the output tensor of last layer(embedding)

- Dense Layer it is just a regular layers of neurons in neural nets each neuron receives input from all the previous layer and take the following parameter:
  - units (200), activation (“tanh”), and take its input tensor from the last layer.
- Two output dense layer:
  - units: 1 for target output, 5 for toxic degree output
  - activation: I’m understand from here that softmax() helps when you want a probability distribution, which sums up to 1. but sigmoid() is used when you want the output to be ranging from 0 to 1, but need not sum to 1. so in my problem I will use sigmoid() as activation function with output layers.

Compile stage:
- Optimizer: “Adam” used to minimize the loss functional
- Loss function: “binary_crossentropy” to calculate the error in classification
- metric: “accuracy” to evaluate the preforming of the model

Train Stage:
- training data: I will train this model in all train data we have, features(pad_x_train), target([y_train, aux_y_train])
- validation data: features(pad_x_validate), target([y_validate, aux_y_validate])
- epochs: 20
- verbose: 1 used to show info during training processes
- batch size: 512, In each epoch model bring random batch data to GPU, or CPU to training

Model Summary:



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 240)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 240, 300)     122798400   input_1[0][0]
__________________________________________________________________________________________________
simple_rnn_1 (SimpleRNN)        (None, 100)          40100       embedding_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 200)          20200       simple_rnn_1[0][0]
__________________________________________________________________________________________________
target (Dense)                  (None, 1)            201         dense_1[0][0]
__________________________________________________________________________________________________
aux (Dense)                     (None, 5)            1005        dense_1[0][0]
==================================================================================================
Total params: 122,859,906
Trainable params: 61,506
Non-trainable params: 122,798,400
__________________________________________________________________________________________________
```

2. Main Model (LSTM):

In this stage I will build the main model using Long Short-Term Memory layer which is treatment the vanishing gradient problem happens in SimpleRNN.

This model contained from 10 layers:

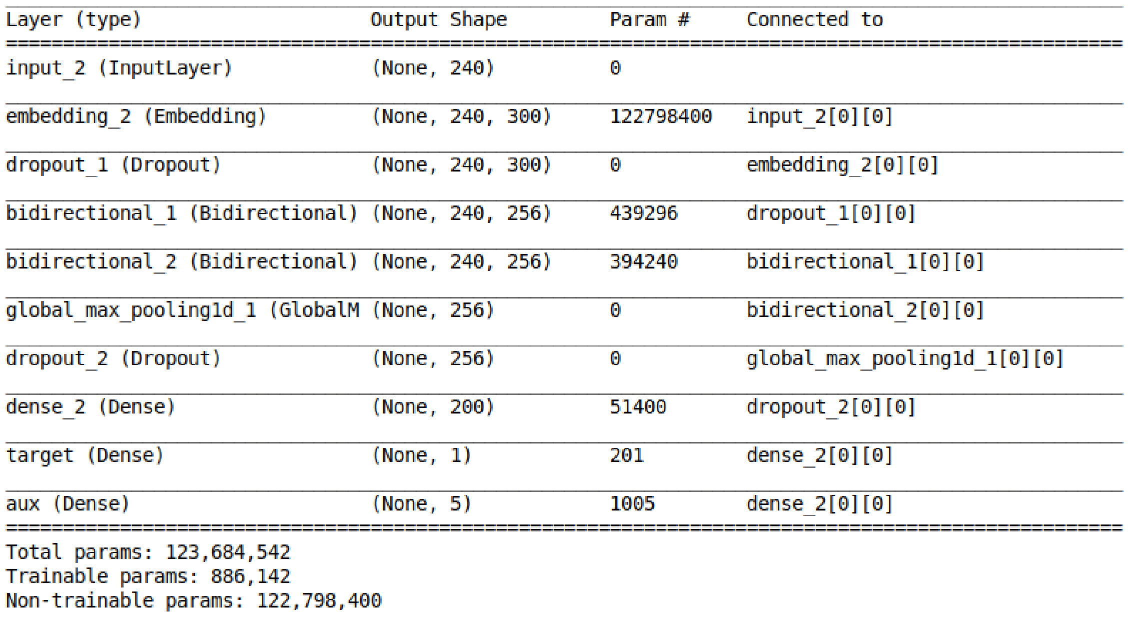
- Input layer with shape 240(Input length)
- Embedding layer: (discussed before)
- Two dropout layers: it is a simple way to prevent the overfitting with with rate 0.2 of the input units to drop.

- Two Bidirectional LSTM layers: are an extension of traditional LSTM that can improve model performance on sequence classification problems. Bidirectional LSTM train two instead of one LSTM on the input sequence. The first train on the input sequence as-is and the second on the reverse copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem. we have two layer from this type takes parameters:
    - units: 128
    - return_sequence: true, return the hidden state output for each input time step.
    - Its input take it from last layer output.
- GlobalMaxPool1D: pooling layer use for reduce feature map.
- Dense layer: (discussed before)
    - units(200), activation(tanh)
- two output layer as we discussed before with same parameters.

Compile Stage: as before Compiled Stage.

Train Stage:

- training data: pad_x_train, [y_train, aux_y_train]
- evaluation data: pad_x_evaluate, [y_evaluate, aux_y_evaluate]
- epochs: 20, verbose: 1
- sample_weight: [sample_weights, np.ones_like(sample_weight)]



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_2 (InputLayer)             (None, 240)           0
____________________________________________________________________________________________________
embedding_2 (Embedding)          (None, 240, 300)      122798400   input_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 240, 300)      0           embedding_2[0][0]
____________________________________________________________________________________________________
bidirectional_1 (Bidirectional)  (None, 240, 256)      439296      dropout_1[0][0]
____________________________________________________________________________________________________
bidirectional_2 (Bidirectional)  (None, 240, 256)      394240      bidirectional_1[0][0]
____________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM  (None, 256)           0           bidirectional_2[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 256)           0           global_max_pooling1d_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 200)           51400       dropout_2[0][0]
____________________________________________________________________________________________________
target (Dense)                   (None, 1)             201         dense_2[0][0]
____________________________________________________________________________________________________
aux (Dense)                      (None, 5)             1005        dense_2[0][0]
====================================================================================================
Total params: 123,684,542
Trainable params: 886,142
Non-trainable params: 122,798,400
```

Refinement

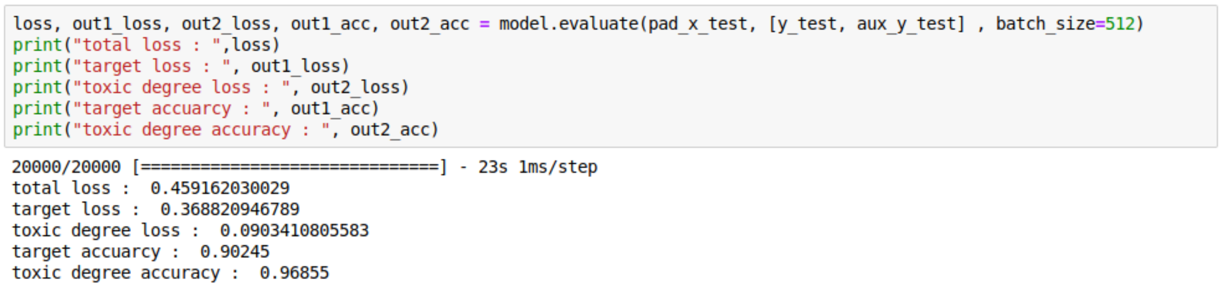
- In the first time of trained(on epochs=5) the model I get the aux_accuracy (0.84) but bad target_accuracy (0.65).
- I’m trying to improve the model by increase the number of epochs to 20 and the number of units in LSTM layer from 120 to 128 the model take near an hour to train, I get more good accuracy in aux_accuracy (0.93) and target_accuracy(0.79) which effected negatively by overfitting accrued between target_accuracy and val_target accuracy.

- I’m trying to decrease this overfitting by add more dropout layer, and increase the training sample and the result is so excited aux_accuracy(0.96), target_accuracy(0.90) with some overfitting between the target_accuracy and val_target_accuracy

Result

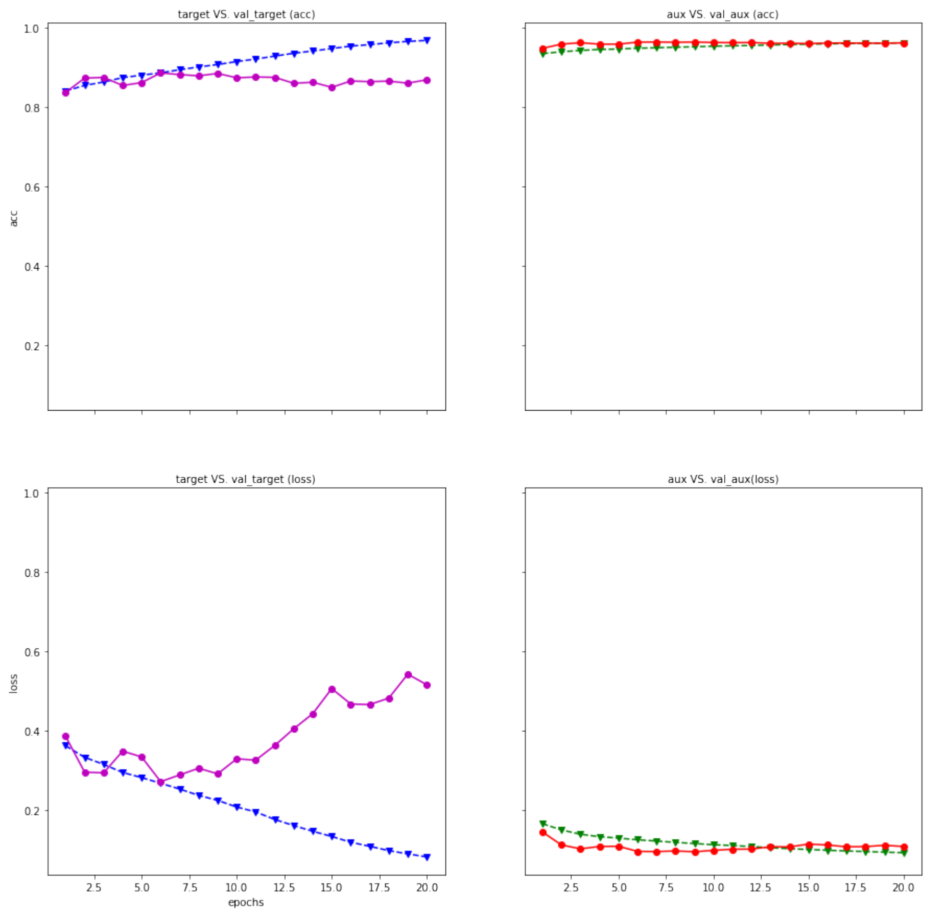
Model Evaluation And Validation

final architecture will chosen because it the best compared by the latter architecture and hyperparameter and reasonable and aligning with solution expectations.
The model was evaluated with unseen data before(testing data = ~20k sample) and it preform well, and it is more accurately with seconder output which measure the degree of toxic with accuracy(~0.96) and very good with target accuracy which determine if comment is toxic or not with accuracy(~0.90).



```
loss, out1_loss, out2_loss, out1_acc, out2_acc = model.evaluate(pad_x_test, [y_test, aux_y_test] , batch_size=512)
print("total loss : ",loss)
print("target loss : ", out1_loss)
print("toxic degree loss : ", out2_loss)
print("target accuarcy : ", out1_acc)
print("toxic degree accuracy : ", out2_acc)
```

```
20000/20000 [==============================] - 23s 1ms/step
total loss :  0.459162030029
target loss :  0.368820946789
toxic degree loss :  0.0903410805583
target accuarcy :  0.90245
toxic degree accuracy :  0.96855
```

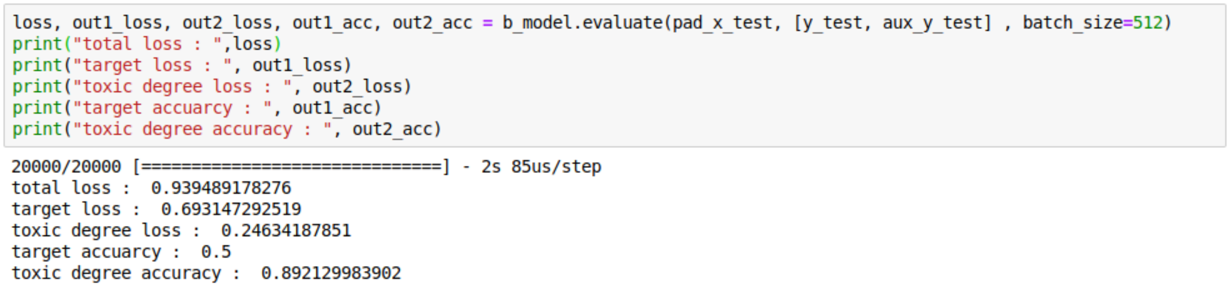


We note that the model doing well with aux_target and suffer from little overfitting main_target. But doing well with test data which mean that the model is good and robust enough for the problem. And we can trusted from model's result.

Justification

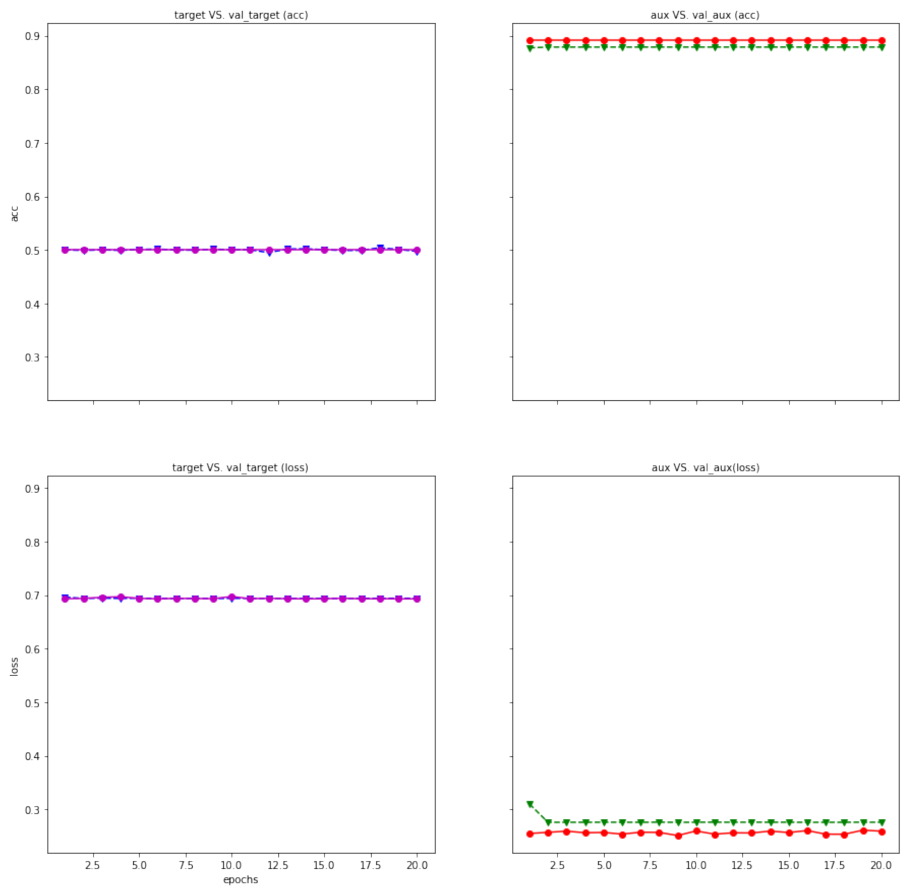
The result of the first benchmark(SimpleRNN) model:

target_accuracy = 0.5



```
loss, out1_loss, out2_loss, out1_acc, out2_acc = b_model.evaluate(pad_x_test, [y_test, aux_y_test] , batch_size=512)
print("total loss : ",loss)
print("target loss : ", out1_loss)
print("toxic degree loss : ", out2_loss)
print("target accuarcy : ", out1_acc)
print("toxic degree accuracy : ", out2_acc)
```

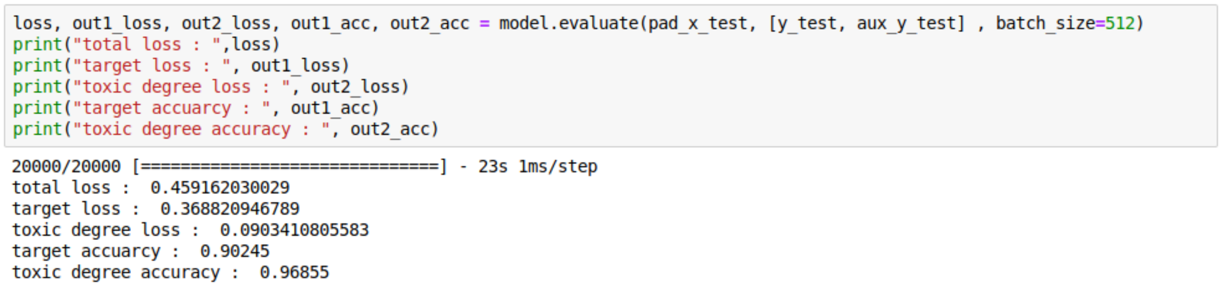
```
20000/20000 [==============================] - 2s 85us/step
total loss :  0.939489178276
target loss :  0.693147292519
toxic degree loss :  0.24634187851
target accuarcy :  0.5
toxic degree accuracy :  0.892129983902
```



the result of the second benchmark(Private Leaderboard):

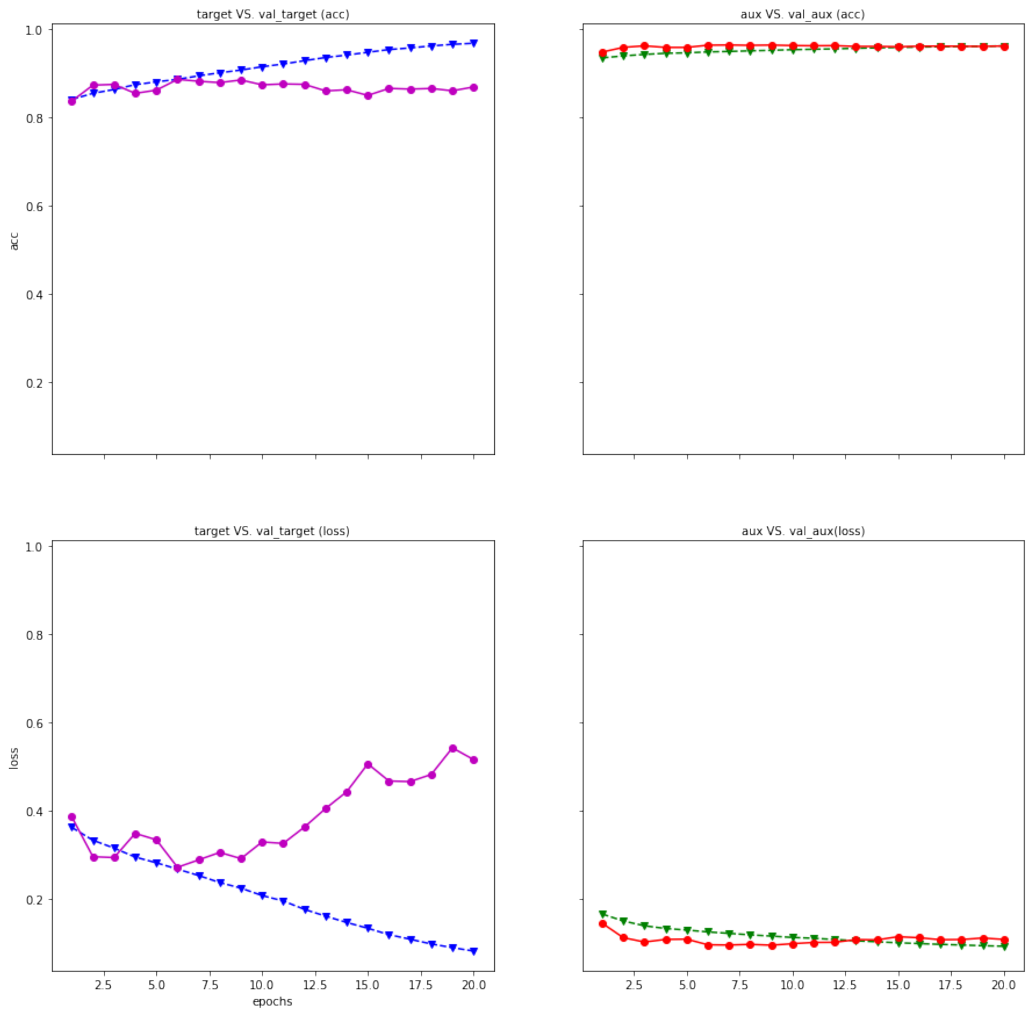
target_accuracy = 0.94

the result of Main Model:
target_accuracy = 0.90



```
loss, out1_loss, out2_loss, out1_acc, out2_acc = model.evaluate(pad_x_test, [y_test, aux_y_test] , batch_size=512)
print("total loss : ",loss)
print("target loss : ", out1_loss)
print("toxic degree loss : ", out2_loss)
print("target accuarcy : ", out1_acc)
print("toxic degree accuracy : ", out2_acc)
```

```
20000/20000 [==============================] - 23s 1ms/step
total loss :  0.459162030029
target loss :  0.368820946789
toxic degree loss :  0.0903410805583
target accuarcy :  0.90245
toxic degree accuracy :  0.96855
```

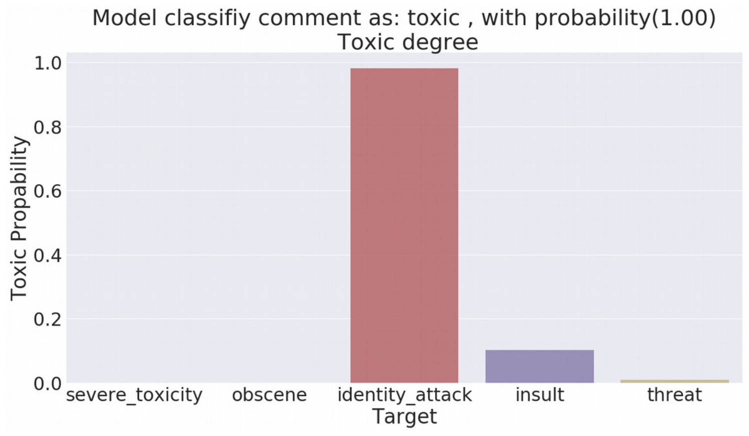


Conclusion

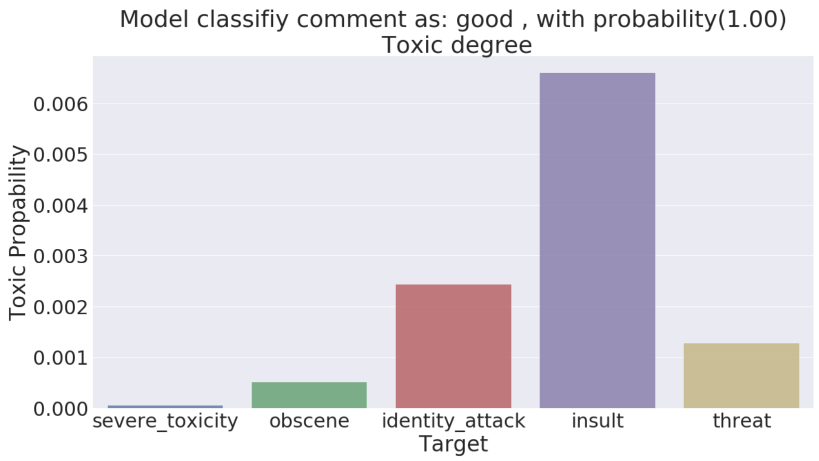
Free-Form Visualization

trying the model in new comments that we write as in real life to see if the model perform.

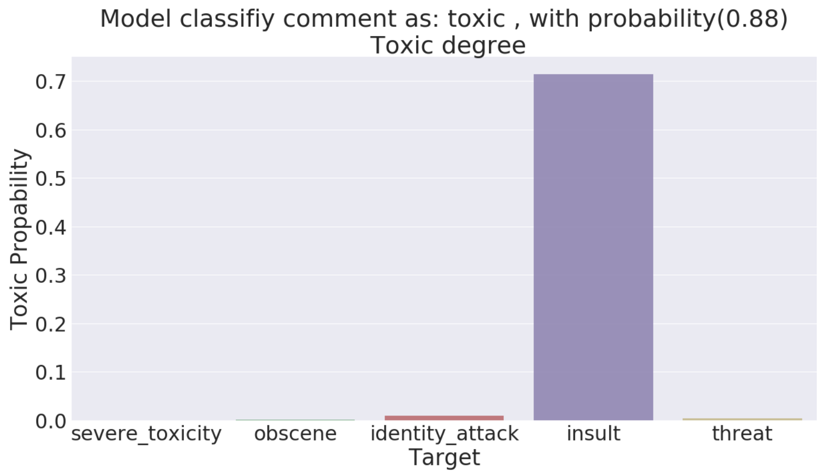
When we write “hello world i like to help poor people, but not black of them”



when we write “hello world i like to help poor people”



when we write “poor man is bad”.



Reflection

the entire process I followed to solve this problem:

1. downloaded data files and pretrained word embedding.
2. Explore and visualize data.
3. Preprocessing data.
4. Prepare data split it into train, test, and validation sets.
5. Applying embedding on the data and produce embedding_matrix
6. build simple model as a benchmark model
7. build the major model using LSTM
8. evaluate model

there are a lot of interested aspect in this project:

1. I’m learning to deal with keras functional API.
2. I’m learning a lot about very important recurrent nets like LSTM.
3. its so excited to deal with sequence data and convert it to numerical data.
4. I’m so happy to learn about word embedding techniques.

The difficult aspect of the project:

1. first I’m suffering to understand the LSTM Cell and how it works.
1. I’m facing some problems related with hardware the model need long time to train.
2. Build the sample_weight array

I think the final solution not bad to deal with these type of problem.

Improvement

I’m suggest some improvement to the model:

1. trying different number of optimizer (e.g “rmsprop”, “SGD”)
2. trying different number of units in hidden layers
3. trying to train the model in different numbers of epochs

I’m a beginner in the NLP domain and I spend a little time to learn some techniques, I think that if I have more time for studying this domain I will discover more techniques that will help me(e.g GRU recurrent network).
I believed that there is more improvement can make to model to get more better solution.