Recurrent Neural Networks for Toxic Comments Classification

Shehab M. Mohamed[#1], Maged Guirguis[#2], Ahmed ElHussiny[*3]

Computer Science Department, The American University in Cairo
AUC Avenue, Cairo, Egypt

[1]shehab_m@aucegypt.edu

[2]magedrefaat@aucegypt.edu

[3]aelhussiny@aucegypt.edu

Prof. Ahmed Rafea
The American University in Cairo

rafea@aucegypt.edu

Abstract— **This paper attempts to tackle the problem of toxic language on the internet, through detecting and identifying it by applying recurrent neural networks.**

Keywords— **neural networks, natural language processing, toxic language, online behavior management**

I. Problem

The internet is a lot like a very advanced stick. Just as a stick can be used for good through building and supporting, it can be used for evil as a weapon. The internet is the same in that sense, it can be used for building knowledge, supporting others, sharing information, and guiding each generation to a place more knowledgeable and better than the previous generation. It can also be used as a weapon, a means of which there is no escape. Protected by their anonymity, cyberbullies thrive. The internet is a medium so easy to communicate, and so anonymous, that it became a non-safe medium where one can be subjected to verbal abuse, online harassment, or even death threats. This topic is so important it became the main target of First Lady of the United States Melania Trump. Ensuring that the internet is a safe place for all is the problem we aimed to tackle. Our main target was to detect the abnormal use of a platform, and stop the spread of this toxic behavior. This could be through the deletion of comments, accounts, groups, or even entire websites.

This is best done through applying the skills that we learned in Natural Language Processing. Our aim is to analyze actual online comments on the internet and categorize each comment based on the level of toxicity and label the type of this toxicity. We found an available dataset that we can work with, which is a collection of comments posted to Wikipedia with a label on each comment indicating the type of toxicity.

II. Methodology

Before we can start training our network, we first need to identify exactly how we will do that, and the ways through which the data will be preprocessed and massaged to be applied to the network.

Our data would be divided into two parts: X and Y. X will represent our actual input to the model, indicative of the comment, and Y will represent the expected output (label) from the model. The main idea is to train a neural network that can map a function of X to yield Y with ideally an error result of 0%. We also need to set our expectation that achieving 0% error is not realistic but the goal is to minimize the error as much as possible.

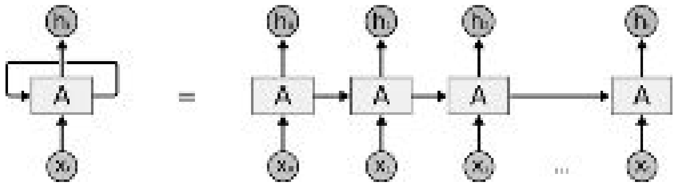
A. Recurrent Neural Networks

Neural networks come in a variety of shapes, sizes, and techniques. The type of network we chose for the purpose of this project is the recurrent neural network. This is most appropriate for solving problems that are made up of sequences of words.

To briefly explain RNN, it works by recursively feeding the output of a previous network into the input of the current network. But how far back should the network's memory go? For that, a popular layer of RNN was created, called Long Short Term Memory (LSTM). LSTM puts the constraint of how many previous layers a Recurrent Neural Network should remember [4].

The following diagram below (Figure I) shows the idea where X is fed to cell A to generate an output, and the output of that cell is also an input to the cell. So the output of each recursion is the input passed to the next layer. LSTM takes a tensor (array) of Batch size, time steps, and number of inputs. Batch size is the number of samples in a batch, time steps is the number of recursion it runs for each input, or as the example below shows, the number of "A"s, and finally the number of inputs is the number of words in each sentence. This whole tensor (3 dimensional) is considered a variable Xt in the picture.

FIGURE I
Representation of Input Tensor to RNN



III. Experiment

We conducted an experiment with a dataset of labelled comments obtained from Wikipedia. The target was to see if we could train a recurrent neural network (RNN) to properly recognize the labels.

A. Programming Libraries

Keras is an open source deep learning library in Python programming language [1]. It enables the fast prototyping and experimentation of deep learning models [1]. We used this library to build our CNN model, train it and validate its performance. Keras gave us the ability to perform transfer learning by initializing our model with weights learnt from Imagenet competition.

B. Dataset

Kaggle is an online machine learning competitions website [2]. It has a wealth of publicly available datasets in a variety of domains ranging from US home prices market to medical images datasets [2]. For our experiment, we used the dataset available in Toxic Comment Classification Challenge [3]. The aim of this competition is to build a classifiers that would classify the toxicity of comments online. This dataset contained 159,571 sentences. For our experiment, we divided this dataset into 143,613 sentences as a training set and 15,958 sentences as a validation set (90% training, 10% validation). The labels were one of six classes, according to toxicity: {"toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate" }.

C. Data Preprocessing

Unfortunately, we can't just put the words as is to the network. Before we, feed the comments in our neural network for it to learn, we need to apply some data preprocessing and NLP techniques before presenting it as X set. So we used the following:

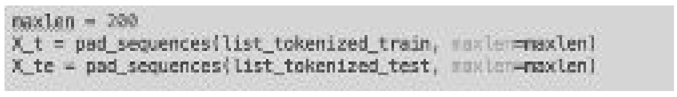
I) Tokenization: This is the method of taking an input sentence as a string to break it down to a list of unique words. For example, string "This is our NLP project" will become ["This", "is", "our", "NLP", "project"].

II) Indexing: After applying tokenization to a given sentence, we need to put the words into some kind of a dictionary where we index each word to have a better representation. The previous example would be {1: "This", 2: "is", 3: "our", 4: "NLP", 5: "project"}

III) Index representation: After getting this data structure, we chain the words together by representing the sentence as a sequence of words, so that when we feed the input to our model, we have some information which word follows the word before and after. An example of it would be [1, 2, 3, 4, 5, 2, 3, 1]

After all sentences have been tokenized and indexed, there still remained a problem of length consistency, since each comment had a different length than the other which means that not all sentences have equal number of features to extract and learn in the model. To remedy that, We used a method called pad_sequence() which makes all the sentences have the same length, so we put a constant value for a maximum size so that all comments have the same length. Short comments will be filled with zeros in order to reach the maximum length. Based on our dataset, we set our maximum length value to 200 as shown in the below screenshot (Figure II) to make sure we cover all different lengths even though they are all shorter than 200.

FIGURE II
Comment Padding

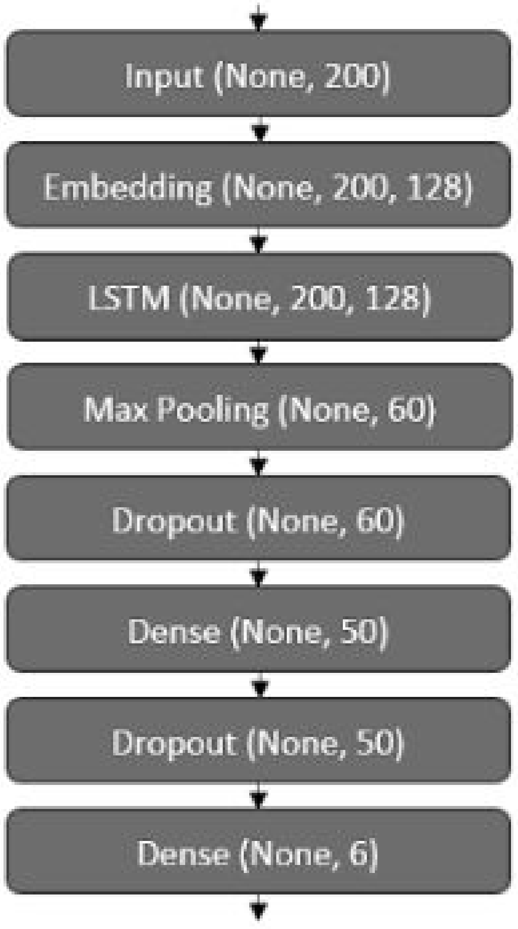


```
maxlen = 200
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
```

D. Models and Training

For this particular classification problem, we decided to use deep learning to solve this problem since it requires a complex architecture that learns not only an instant input, but a set of sequence for it to become a general classifier. We decided our architecture would be as shown in Figure III.

FIGURE III
Representation of Input Tensor to RNN

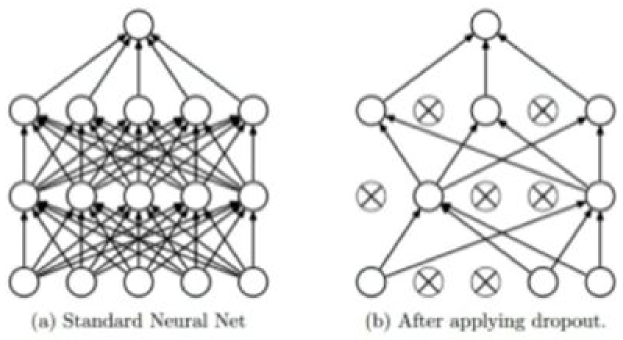


The rationale behind this selection was as follows:

Before we pass the 3D tensor (generated by the embedding and LSTM layers, as explained in section II.A) to one of the following layers of our architecture, we need to reshape this tensor into a 2D tensor without throwing away useful information about the original data. So we use a Global Max Pooling layer which is regularly used in Convolutional Neural Networks that reduce the dimensionality of image data without removing essential data. The idea of Global Max Pooling is we go through each batch of data, and take the maximum value of each batch to yield the collection of data with down-sized dimensionality.

After applying Global Max Pooling to reduce the dimensionality of the tensor, we pass the output of it to a Dropout layer which disables some neurons in the layers so that the next layer is forced to handle the representation of the missing data thus making the network more efficient at generating generalized results. In our architecture we have used two Dropout layers of 25% dropout which means forget 25% of the neuron weights to make it much harder to over fit while training. This concept is illustrated in Figure IV.

FIGURE IV
Visual representation of Dropout

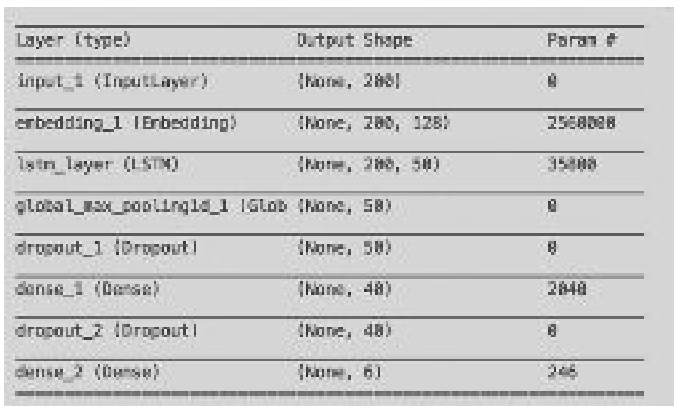


After Dropout layer, the output is connected to a regular layer with activation function "RELU" and then passed to another dropout layer of 25% then to final output layer of activation function "Sigmoid". The last output layer has the number of neurons equal to the number of classes in our classification problem, in that case it has 6 neurons in the last layer. The built model was compiled with a batch of size 32 so we don’t feed the entire training data in one batch. Each epoch is a cycle of training the whole network with all batches of the training data. With experimentation, we found it was enough to run it for 4 epochs only.

A summary of this architecture can be seen below in Figure V. It is important to note that this architecture was developed after experimenting with different values and fine-tuning specific parameters such as:

- number of layers,
- number of neurons,
- types of learning rate,
- percentages of dropout,
- embedding size,
- batch size, and
- number of epochs.

FIGURE V
SUMMARY OF ARCHITECTURE

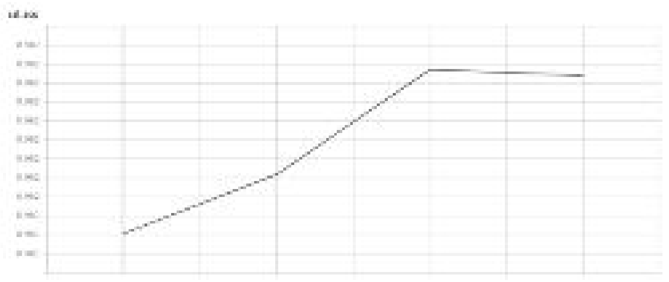


```
Layer (type)                 Output Shape              Param #
==============================================================
input_1 (InputLayer)         (None, 200)               0
______________________________________________________________
embedding_1 (Embedding)      (None, 200, 128)          2560000
______________________________________________________________
lstm_layer (LSTM)            (None, 200, 50)           35800
______________________________________________________________
global_max_pooling1d_1 (Glob (None, 50)                0
______________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
______________________________________________________________
dense_1 (Dense)              (None, 40)                2040
______________________________________________________________
dropout_2 (Dropout)          (None, 40)                0
______________________________________________________________
dense_2 (Dense)              (None, 6)                 246
==============================================================
```

E. Evaluation

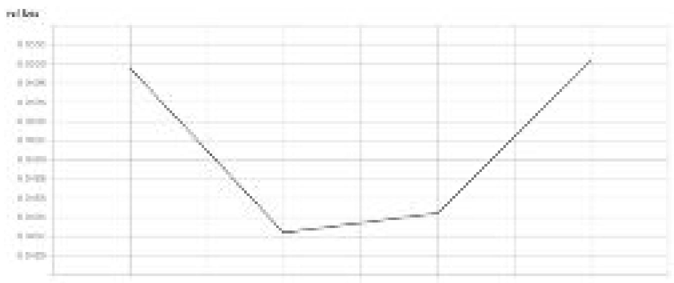
Using TensorBoard to visualize our training progress, this below diagram (Figure VI) shows the number of training cycles (epochs) we ran to learn the weights of our architecture for the toxic classification problem, the y-axis is the validation accuracy, and the x-axis is the number of epochs. Ideally the more epochs given for training, the higher the validation accuracy increases. However, to achieve an overall high validation accuracy we needed to make sure the validation loss is decreasing so that it doesn’t over fit the data. Because if we want to generalize our classifier to any new testing sentence, we need to make sure the validation loss is not increasing while training so that it doesn’t memorize the data by heart.

FIGURE VI
VALIDATION ACCURACY VS EPOCHS



Below (Figure VII) is the diagram for the validation loss where the y-axis is the validation loss and x-axis is the number of epochs. We chose the lowest value for the validation loss as a metric to determine the highest overall accuracy achieved by the model which was at epoch 2.

FIGURE VI
VALIDATION LOSS VS EPOCHS



IV. RESULTS AND DISCUSSION

After training this model on 143,613 data points (sentences) for only 4 epochs and then testing it using the validation set of 15,958 data points, we were pleasantly surprised to see that our model had yielded a classification accuracy of 98.22%.

While this is an excellent accuracy, we hope that in the future, better results can be achieved. When it comes to deep learning, there are so many parameters that can be experimented with to improve the accuracy of the classification problem. Some of these parameters and suggestions for improving on our work are:

- collecting more data points to increase the size of the training data and the variation of language used
- try different architectures

- rely upon transfer learning and load a significantly more complex pre-trained network and fine-tune the last few layers to achieve higher accuracy

It is important to note, however, that any of these suggestions would lead to a very serious trade-off between complexity, which is something that our project was trying to avoid.

Acknowledgment

Shehab Mohamed, Maged Refaat, and Ahmed ElHussiny would like to acknowledge Dr. Ahmed Rafea for the guidance he provided throughout the length of the project, and for teaching us the content necessary to provide us with the understanding needed to build such a project.

References

[1] Keras.io. (2018). Keras Documentation. [online] Available at: https://keras.io/ [Accessed 17 Apr. 2018].

[2] Kaggle.com. (2018). Kaggle: Your Home for Data Science. [online] Available at: https://www.kaggle.com/ [Accessed 10 Apr. 2018].

[3] Kaggle.com. (2018). Wikipedia Comments | Kaggle. [online] Available at: https://www.kaggle.com/ [Accessed 10 Apr. 2018].

[4] Moustafa, Mohamed. "Recurrent Neural Networks." 2018. Lecture.