Another neural network that we have used is the Long Short-Term Memory or LSTM. LSTM is a type of recurrent neural network. The advantage of using LSTM is that it can remember the ordering between words in a sentence. This is very important because to understand a sentence you need the context of previous words. For this approach we used Keras with a TensorFlow backend.

First, we downloaded the training and testing data from Kaggle and put them into dataframes. The data consisted of a unique ID followed by the comment text and 6 binary labels (toxic, severe\_toxic, obscene, threat, insult, identity\_hate).

Next, we preprocessed the comments by tokenizing or splitting them up into individual words and putting them into a type of list. Keras comes with a built in function for this called Tokenizer which also indexes each new word into a vocabulary or dictionary and counts each instance of those words. For the tokenizer we had capped the number of words the vocabulary can hold to 50000 because training would take too long without a cap. With this cap it will only hold the 50000 most frequent words in the vocabulary.

After that, we transform each of the comments of the data frame into a list of numbers represented by the index of the vocabulary. For example: “This article is from Wikipedia” would be represented as [13, 23, 8, 31, 28].

There are many short comments and some very long comments. To be able to put this data into the neural network, we would have to find a max length. To do that we had to find the number of words each comment holds and find one that is not too long that there is too much empty words but also not too short that it cuts off some of the words. We have chosen the max words to be 300 based on some of the statistics of the data that we have collected.

When we finished with the preprocessing we moved on to building our LSTM model. Using Keras made this very simple as opposed to just using TensorFlow. To build this model we have used the functional model api. For this step we have chosen and tried many different arbitrary numbers for the amounts of hidden neurons, layers, activators, batch sizes, and epochs. After much testing, we have decided to use 64 hidden neurons, 32 batch size, and 2 epochs. For the layers we have: input, embedding, LSTM, global max pooling, dropout, ReLU activator, dropout, and sigmoid activator.

The result is a 98.36% accuracy.