Comment Analysis System

A project report submitted to

Delhi Technological University

For Engineering Analysis (Co-207) Innovative Project

Bachelor of Technology - Computer Engineering

Submitted by

B.Tech 3rd semester

Aman Mahendroo (2K19/CO/048)

Anshuman Raj Chauhan (2K19/CO/067)

Under the esteemed guidance of, Prof Anil Singh Parihar

DELHI TECHNOLOGICAL UNIVERSITY

Vision:

“To be a world class University through education, innovation and research for the service of humanity”

Mission:

1. To establish centres of excellence in emerging areas of science, engineering, technology, management and allied areas.
2. To foster an ecosystem for incubation, product development, transfer of technology and entrepreneurship.
3. To create an environment of collaboration, experimentation, imagination and creativity.
4. To develop human potential with analytical abilities, ethics and integrity.
5. To provide environment friendly, reasonable and sustainable solutions for local & global needs.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision:
Department of Computer science and engineering to be a leading world class technology department playing its role as a key node in national and global knowledge network, thus empowering the computer science industry with the wings of knowledge and the power of innovation.

Mission:
The mission of the department is as follows:
1. To nurture talent of students for research, innovation and excellence in the field of computer engineering starting from undergraduate level.
2. To develop highly analytical and qualified computer engineers by imparting training on cutting edge technology.
3. To produce socially sensitive computer engineers with professional ethics.
4. To focus on R&D environment in close partnership with industry and foreign universities.
5. To produce well-rounded, up to date, scientifically tempered, design oriented engineers and scientists capable of lifelong learning.

Contents

AIM

In this project, we aim to develop a Comment analysis system using an LSTM

INTRODUCTION

We use an artificial neural network, containing a sequence model (LSTM) to analyse text data with the following purposes:

1. Detect undignified content / swear words etc in the comment, in order to help filter the content in the website for users who may not wish to see them.

2. Quickly categorise comments into positive and negative in order to classify the post into various categories, such as good, if it contains mostly good comments, bad, if it contains mostly bad ones, controversial, if it contains a large mix of both etc.

Our purpose is to make the model scalable, so that it can find a variety of applications in even the most versatile social media websites.

The network runs on the server, so whenever a user posts a comment on a website, it is sent to the server for analysis. The system returns its response to the back-end of the website and further processing takes place.

The code is written in Python and using Keras machine learning library.

MOTIVATION:

In today's world of social media, there is a heavy influx of textual content, in the form of responses and text messages all over the internet. This calls for an automatic system for regulation of this content and filtering. For this purpose, we propose a scalable Recurrent Neural Network which helps in analysing such textual content with accuracy and efficiency.

Here is an example where this system finds purpose:

- Consider a content-hosting website such as YouTube, in which a user posts some content and other users often respond to it in the form of textual comments. In case such textual comments contain undignified material and do not adhere to maintaining the healthy environment of the website, our system can help analyse and detect them. Even though YouTube has a comment-reporting system, each comment has to be manually analysed, which is a time-consuming process and many comments may go unreported. Not to mention, it is subject to human-error. In such a scenario, our comment analysis system can help in analysing each thread efficiently.

THEORY

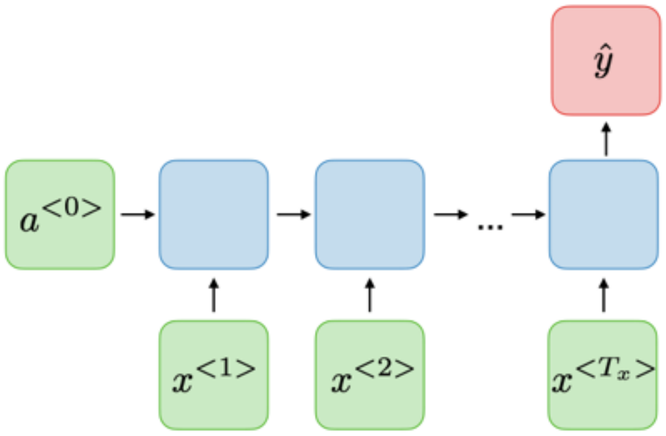
Deep Learning Theory

Recurrent Neural Network

A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. RNNs are designed to recognize a data's sequential characteristics and use patterns to predict the next likely scenario.

For the purpose of text analysis, an RNN would process the encoding of each word in an input sentence, with the knowledge of all previous (and possibly, even the forthcoming) words. This allows us to extract meaningful features from the sentence.

For our purpose, we have a sentence input and a single binary output. This requires the use of a many-to-one RNN, which contains a series of chained inputs and only a single output.



The given figure illustrates a many-to-one Recurrent Neural Network:

1. We have a base input a<0> which serves as the initial hidden

state for the propagation of the network.

2. At every step, an encoded-word input x<t> is passed into the network.
3. Finally, when the entire sentence is evaluated, a single output ŷ is obtained.

Forward Pass-

The basic forward pass requires the implementation of the following formulae:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$
$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Here, the activation functions $g_1()$ and $g_2()$ are tanh and sigmoid respectively.
Dimensions of matrices are:

W_{aa}	d*d
W_{ax}	n*d
W_{ya}	d*k
$a^{<t>}$	1*d
$x^{<t>}$	1*n

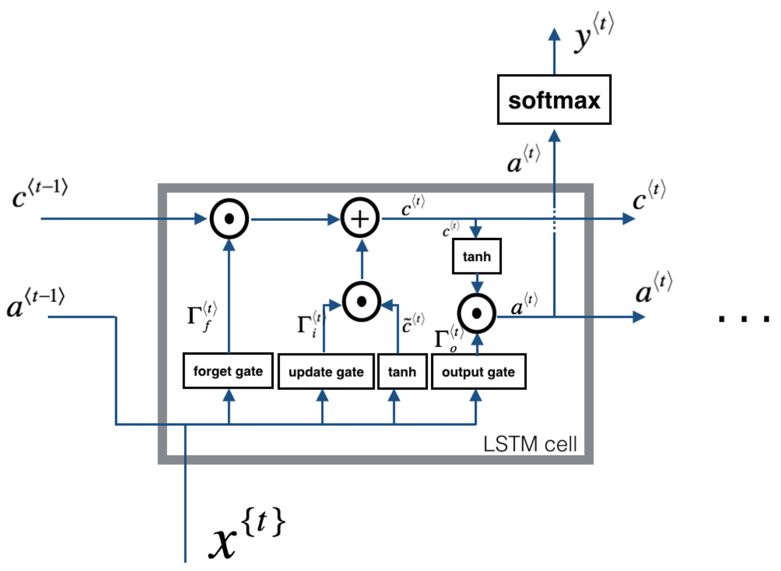
Here :
d- dimension of state vector(keras default)
n- dimensions of word embedding(64 in this project)
k- dimension of output vector at each time stamp(keras default)

Long Short Term Memory cell

Long short-term memory is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points, but also entire sequences of data.

It becomes essential to use LSTM networks for analysis of longer pieces of text. Due to the structure of an RNN, as we analyse words further into a sentence, the influence of words in the beginning of the sentence tends to diminish exponentially(vanishing gradient problem).

In order to get around this, we use feedback connections to maintain the influence of these initial words in the final output of the network.



$$\Gamma_f^{\langle t\rangle} = \sigma(W_f[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_f)$$
$$\Gamma_u^{\langle t\rangle} = \sigma(W_u[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$$
$$\tilde{c}^{\{t\}} = \tanh(W_C[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_C)$$
$$c^{\langle t\rangle} = \Gamma_f^{\langle t\rangle} \circ c^{\langle t-1\rangle} + \Gamma_u^{\langle t\rangle} \circ \tilde{c}^{\langle t\rangle}$$
$$\Gamma_o^{\langle t\rangle} = \sigma(W_o[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_o)$$
$$a^{\langle t\rangle} = \Gamma_o^{\langle t\rangle} \circ \tanh(c^{\langle t\rangle})$$

The LSTM cell requires a few more equations (mentioned above) for its implementation. Its different parts are described in brief:

1. Forget gate: the forget gate tells the cell state which information to forget by multiplying 0 to a position in the matrix.

2. Update gate: the update gate operates similar to a forget gate in the LSTM. It tells the cell state which information to update and which to forget in the matrix. However, it is multiplied with the modified cell state in the forward pass.

3. Output gate: the output gate in the LSTM is used to judge the extent to which a new value is used to evaluate the output in the network.

For simplicity, we won't mention here the exact dimensions of each of the entities in the expressions stated above.

Loss function

During back-propagation, we use a loss function which modifies the absolute error between the model's guess and the true answer for a particular element in the dataset.

Categorical cross-entropy loss: It is defined as:

$$CCE = \; - log \; q(x)$$

Where $q(x)$ is the one-hot encoded vector of the errors in the final output

Regularization technique

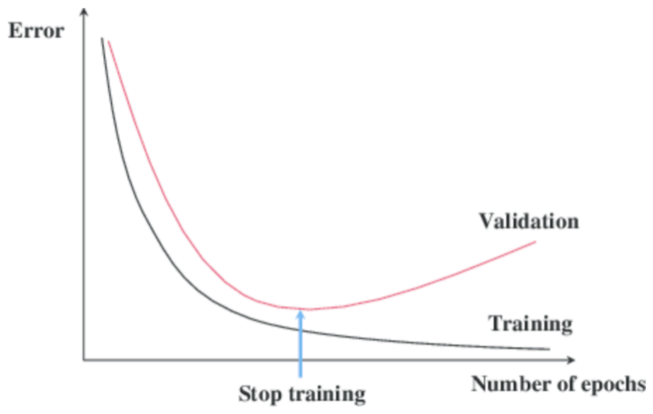
Bayes' error: Bayes' error in machine learning is defined as the lowest possible achievable error in a problem. This forms an important basis for analysing the performance of a model. We deem that the training error of any model is higher than the test error. On top of the bayes' error, we define the following terms:

1. Bias: the bias (different from bias used in weights and biases in forward propagation) is defined as the difference between the training error and (estimated) Bayes' error.
2. Variance: the variance is defined as the difference between the training error and test error in our model.

Analysis of bias and variance in a problem allows us to identify under-training and overfitting in our model and training techniques.

Early Stopping

In early stopping, we identify the point at which the accuracy in the test set begins to decrease in comparison to the training set and stop the execution of training at that point, as shown in the figure.



The patience of the early stopping algorithm is defined as the number of epochs, in which we see no improvement in the model, required to stop the execution of the training algorithm. For example, if the patience is set to 5, the training will stop if we encounter 5 epochs with 0 improvement in the accuracy.

Dropout

In dropout regularization, we disable some of the nodes in the artificial neural network in order to reduce their influence in training. This allows us to reduce overfitting of the model to the training set.

The Dropping probability is defined as the probability with which to randomly choose a neuron to disable in a particular layer. For example, if the dropout probability is set to 0.5, then, on average, half of the neuron will be disabled during one training cycle.

Optimization algorithm

An optimization algorithm is an algorithm which allows the neural network to learn faster and achieve better performance. We use optimization algorithms to alter the learning rate of the model dynamically.

- RMSProp: The RMSProp optimizer employs the following formula to calculate the updated weights and gradients.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

E[g] — moving average of squared gradients. dC/dw — gradient of the cost function with respect to the weight. n — learning rate. Beta — moving average parameter (good default value — 0.9)

This helps us to avoid exploding and vanishing gradients while taking into account the cost of the current iteration.

RMSProp works better than RProp in batch training because RProp uses averaging of gradients over successive mini-batches.

Mini-batch training

Mini-batch training is a popular technique in optimizing the training process. In order to understand mini-batch, we must understanding the following:

1. Stochastic training: in this method, we apply the learning algorithm to our model after passing just a single case from the dataset. This requires multiple iterations but each individual iteration is faster.

2. Batch training: in this method, we pass the entire dataset through the model and calculate the gradients over the error on the entire dataset. This requires very few iterations but each iteration takes a long time. The model may also not collapse to the optimum solution and simply hover around it.

Mini-batch training is in the middle of stochastic and batch training. Here, we divide the entire dataset into smaller chunks and train them individually. Due to this, we can leverage the advantages of both stochastic and batch training. By applying appropriate regularization and optimization techniques, we can avoid the disadvantages of both as well.

Modelling - Simulation Theory

Discrete Event simulation

Discrete Event simulation concerns the modelling of a system as it evolves over time by a representation in which the state variables change at discrete points in time.

Next-event time advance Mechanism

Because of the dynamic nature of discrete-event simulation models, a track of the current value of simulated time as the simulation proceeds is kept and also a mechanism to advance simulated time from one value to another is required.

In the next-event time-advance approach, the simulation clock is initialised to zero and the times of occurrence of future events are determined.

The simulation clock is then advanced to the time of occurrence of the most imminent of these future events at which point the state of the system is updated to account for the fact that an event has occurred, and our knowledge of the times of occurrence of future events is also updated.

This process of advancing the simulation clock from one event time to another is continued until eventually some pre-specified stopping condition is satisfied.
Since all state changes occur only at event times for a discrete event simulation model, periods of inactivity are skipped over by jumping the clock from event time to event time.

SIMULATION STEPS INVOLVED IN PROJECT

1. Formulation of problem-

The problem formulated was an unhealthy environment on social media websites and lack of meaningful insights on the available data. Further, a survey was conducted in which comments on youtube and twitter were analysed so as to get a general idea of the comments i.e., average length of comments, language used, etc. Also the rate at which comments are received on these sites was analysed.

2. Collection of data

A dataset of 50000 movie reviews from kaggle was chosen to be used for the classification and filtering model as the reviews were very similar to the analysed comments.

3. Assumption Document Verification

The document (dataset) was validated by Prof. Anil Singh Parihar for usage in our model.

4. Construction of program and verification

The classification and filtering model was built using LSTM networks(details given in the next section) . The program for the same was debugged and validated.
Further, this model was trained on 25000 movie reviews to a training accuracy of 92.46%.

5. Pilot Run

The classification and filtering model was tested on another 25000 reviews and a testing accuracy of 83.18% was obtained.

6. Program Validation

Since the model had a testing accuracy of 83.18% on a very healthy testing data, we concluded our model was working extremely well.

7. Design Experiments

We designed an experiment to check how efficient our classification and filtering model will be in a real world scenario (using single server queuing system).

We first ran our model on different input sizes. The results obtained were as follows
(logarithmic because of tensorflow's efficient matrix multiplication)

Number of comments	Time taken for input
1	1.48 sec
50	1.49 sec
100	1.50 sec
500	1.52 sec
1000	1.72 sec
5000	2.42 sec

Further we decided to choose a server traffic of 50 comments/sec. Ie - 43.2 million comments per day (equal to youtube, one of the busiest servers on the planet).

Our model had a performance of 50 comments in 1.49 seconds but it had to handle the same in 1 sec.

So we chose to run our model every-time the number of comments in the queue exceeded 100. This way the received 100comments/2sec could be processed in 1.5sec , thus making a stable server.

Running our model after 500 comments in the queue would also yield a stable server but this would increase the expected average delay for comments so this plan was discarded.

8. Make production runs

Then we made a single server queuing system for analysing our model's performance.

The model received 50 comments each second and every time the 'number in queue' exceeded 100, the comments were sent into the server(classification and filtering model).

The system was run until 1000 comments completed their delay in the model.

9. Analyse output data

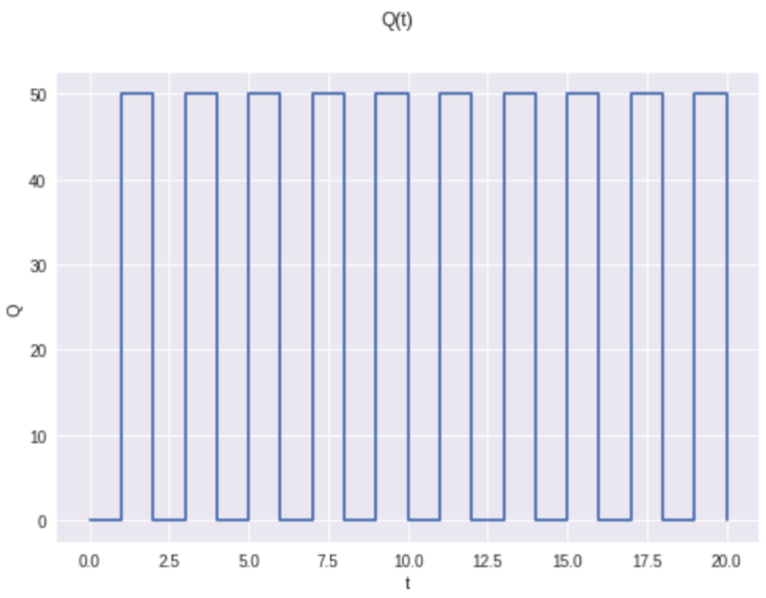
Our model performance obtained were as follows-

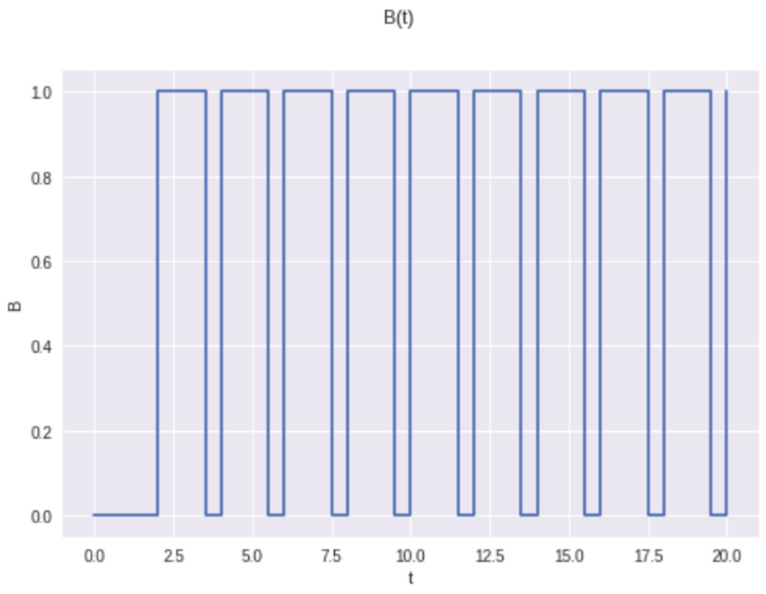
Expected average delay = 0.5 units (here seconds)

Expected average number in queue = 25

Expected utilization of server = 0.675

The plots of Q(t) (number in queue) and B(t) (server status) are shown in figure below





10. Document, Present and use results

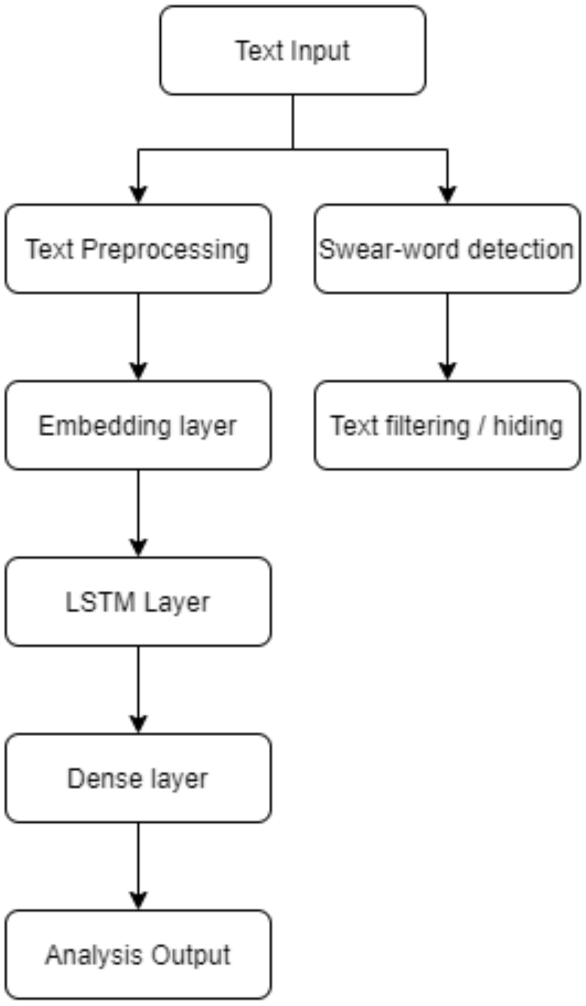
We obtained an accuracy of 83.18% on the test data and the 3 system performance measures told us that our model is working pretty well for a server traffic as high as 43.2 million per day.

Overall we conclude that we have built a very accurate, scalable, robust and reliable model for comment analysis on social media.

CLASSIFICATION AND FILTERING ALGORITHM

SubModel

Text preprocessing and classification are the two major components of this algorithm.
The steps are discussed in detail below-



1. The input text is preprocessed to enable it to pass through the LSTM. The following steps are executed for the same:
    - Lower casing
    - Removal of punctuations
    - Removal of stopwords
    - Stemming
    - Removal of emojis
    - Removal of emoticons
    - Removal of URLs
    - Removal of HTML Tags

2. The preprocessed message is then analysed for swear word detection. This is done using the same principle as used in step 1-c (Removal of stopwords)

3. Further, a dictionary of size 10000 is created by the list of existing words in the present corpus. This yields the tokens(words) being represented by one hot vector of size 10000.

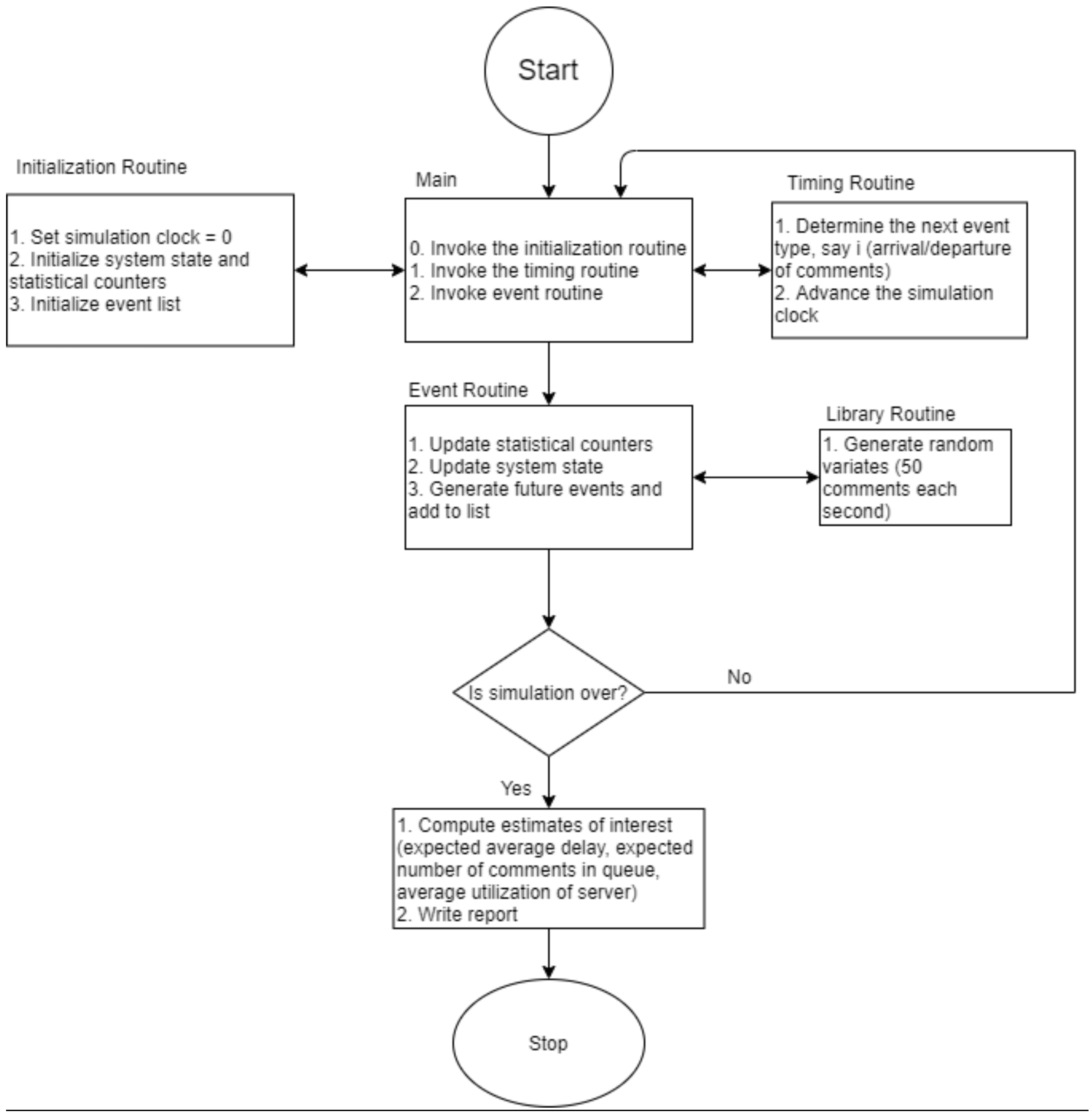
4. The sentences in corpus are 0-padded (or truncated) to a size of 500 characters.
5. The dictionary in step 3 is then encoded to its corresponding (64-dimensional) vector, word by word, using a trained word embedding.

6. The many-to-one LSTM is executed on the encoded vectors and the output is passed through a dense layer. We adjust the network to the following specifications:

   a. The categorical cross-entropy function is used to calculate the loss.
   b. All weights and biases are initialized to 0.
   c. We use mini-batch training with a batch size of 256.
   d. The RMSProp optimizer algorithm is used with the LSTM.
   e. The dense layer has 2 outputs and uses a sigmoid activation function.
   f. Our entire dataset, containing 50,000 samples is broken down to a training set of 25,000 samples and a test set of 25,000 samples.
   g. Dropout regularization is applied to the dense layer with a dropping probability of 0.4.
   h. Early stopping is applied to the entire network with a patience of 3.
   i. Total number of epochs was set to 10. However, the execution stops after 6 epochs due to early stopping.

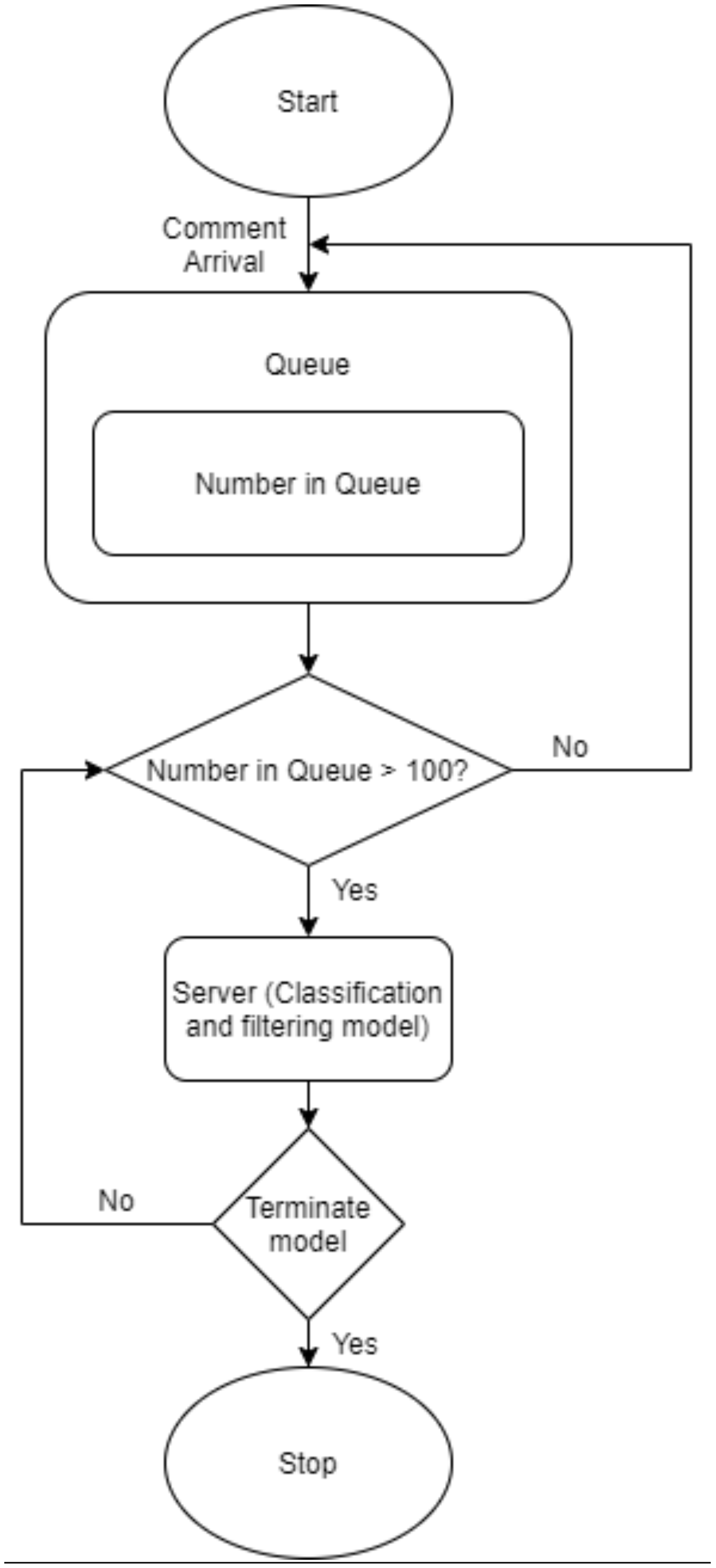
FLOWCHARTS

Single Server Queuing system (for testing model performance)


Start
Initialization Routine
1. Set simulation clock = 0
2. Initialize system state and statistical counters
3. Initialize event list
Main
0. Invoke the initialization routine
1. Invoke the timing routine
2. Invoke event routine
Timing Routine
1. Determine the next event type, say i (arrival/departure of comments)
2. Advance the simulation clock
Event Routine
1. Update statistical counters
2. Update system state
3. Generate future events and add to list
Library Routine
1. Generate random variates (50 comments each second)
Is simulation over?
No
Yes
1. Compute estimates of interest (expected average delay, expected number of comments in queue, average utilization of server)
2. Write report
Stop

- The only difference b/w a conventional single server queuing system and this one is that the comment go into the server(classification and filtering model) only when number in queue exceeds 100.
- Simulation is terminated after number delayed reaches 1000.

OVERALL MODEL



CODE

Due to the large size of the source code, it has been uploaded on GitHub. The link to the repository-

https://github.com/its7ARC/commentAnalysisSystem

(Sometimes GitHub shows "error in loading", In that case please open it again.. it surely opens up in 2-3 attempts)

RESULT AND CONCLUSION

We successfully created a comment classifier model with a good accuracy and performance, which also filters comments based on the presence of foul language. The system performs well on google cloud platform even with very high traffic and also gives decent accuracy on both training and test sets. It can also be expanded to give complex ratings (such as a score from 1-10) because of the use of Categorical Cross-entropy loss function.

This model can directly be applied to any major social media website to channelize content, filter unfriendly language and also analyse the category of posts based on the number and distribution of the comments it received.

RESOURCES USED

For our purpose, we use the following Python libraries to implement our system:

1. Keras framework
2. Numpy library
3. NLTK library
4. Matplotlib
5. SKLearn
6. Google Colab

To train our neural network, we used the following IMDB dataset containing 50000 samples of movie reviews:

Lakhsmipathi N - IMDB Dataset of 50k Movie Reviews - https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

REFERENCES

We referred to the following resources to aid us in our implementation of our project:

1. Greg Condit - The LSTM Reference Card - https://towardsdatascience.com/the-lstm-reference-card-6163ca98ae87 - [03-10-2020]
2. Vitaly Bushaev - Understanding RMSProp - https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a - [05-10-2020]
3. Artem Oppermann - Optimization Algorithms in Deep Learning - https://towardsdatascience.com/optimization-algorithms-in-deep-learning-191bfc2737a4 - [05-10-2000]
4. Sudalai Raj Kumar - Text Preprocessing Guide - https://www.kaggle.com/sudalairajkumar/getting-started-with-text-preprocessing - [12-10-2020]
5. Christopher Olah - LSTM - https://colah.github.io/posts/2015-08-Understanding-LSTMs/ - [15-10-2020]
6. Dhruvil Karani Word Embedding - https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa - [15-10-2020]