Predicting online toxic content

Master Business Analytics
Vrije Universiteit Amsterdam

Yuval Pappie: 2531716

Supervisor:
Jesper Slik

July 31, 2019

Abstract

A big problem for any major social media website today is how to handle toxic content such as hate speech, threats, racism, and online sexual harassment. Communities like Facebook, Reddit, and Quora rely on moderators and users to report offensive content, but there is a need for scalable solutions. This research investigates different state-of-the-art Machine Learning techniques for NLP and gives an experimental comparison of these methods to predict and flag toxic online content. To evaluate the performance, these algorithms are applied on the Quora questions dataset. Two different types of Recurrent Neural Networks (RNN), a Long Short-Term Memory Network (LSTM) and a Gated Recurrent Unit (GRU), are used and compared against a Logistic Regression baseline model. The results showed that the LSTM gives the best performance with a F_1 score of 0.671. Both the LSTM and GRU outperformed the baseline model. This paper concludes that deep learning models are well suited for NLP tasks.

Keywords: NLP, Text Mining, Deep learning, Long Short-Term Memory, Gated Recurrent Unit

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, we can not think of the absence of social media in our lives. According to the Global Digital Report of Hootsuite (2019), there are 3.84 billion active social media users. Communication has become much easier and with the use of social networks we can maintain relationships with people from all over the world. People can express their opinions, ask questions about any topic and can share their experiences easily by writing reviews. But there is also a downside. Due to the anonymity provided by social media, people are willing to say things they would not say in person. This leads to the increase of online toxic behaviour like hate speech, threats, racism, and online sexual harassment.

A big problem for any major social media website today is how to handle this toxic and inappropriate content. Communities like Facebook, Reddit, and Quora are relying on moderators and users to report offensive content. Due to the vast amount of information that is processed daily by these social media sites and the growth of online hateful speech (Guynn, 2019), there is a need for scalable and automated solutions to tackle this problem.

This is where Natural Language Processing (NLP) comes in. NLP is the a subfield of computer science and artificial intelligence that deals with the interaction between computers and humans using the natural language. Some common tasks in NLP are text classification, string matching, machine translation, and speech recognition. With the rise of the popularity of deep learning techniques which are yielding promising results in the fields of image recognition and speech processing, it can also be applied to natural language processing. Therefore, in this paper we will focus on these deep learning methods.

This research investigates different state-of-the-art Machine Learning techniques for NLP and gives an experimental comparison of these methods to predict and flag online toxic content. To evaluate the performance we apply these algorithms on the Quora questions dataset.

This paper begins with discussing related work in the field of natural language processing and text classification in Section 2. Section 3 describes the problem that we want to solve, the data we gather, exploratory data analysis and the data preprocessing steps. In Section 3.2 we will investigate the data in depth and conduct a sentiment analysis, and topic modeling. The models that we will use are extensively explained in Section 4. Next, Section 5 describes the features that we will create and the configuration of our models is explained. Furthermore, we discuss the evaluation measure in order to measure the best performing model. Section 6 will show the results and compare the performance of the models. Finally, Section 7 gives a conclusion and Section 8 a discussion on this research.

2 Related work

NLP and Text Mining are trending topics in recent data science research. In the past, the majority of methods used to study NLP problems were time consuming. With the increase of computing power, deep learning techniques can be trained more efficiently which tends to outclass traditional machine learning models like logistic regression, decision trees , or SVM. In this section, we will discuss relevant research that has been conducted on using both traditional supervised models and supervised deep learning models.

An overview of classical methods can be found in the research of Malmasi and Zampieri (2017) and Davidson et al. (2017). They examine methods to detect hate speech in social media using an annotated tweet dataset divided into three categories: hate speech, offensive speech, and none. Malmasi and Zampieri (2017) applied standard lexical features such as character n-gram, word n-grams, and word skip-grams to train a Support Vector Machine (SVM). The best result was obtained by a character 4-gram model achieving 78% accuracy. Davidson et al. (2017) used logistic regression, naive Bayes, decision trees, random forests , and SVMs models to classify the tweets.

Badjatiya et al. (2017) use multiple deep neural networks for hate speech detection. A benchmark dataset of 16k tweets was analyzed and labeled as sexist, racist or neither. They compare baseline methods such as character n-grams, TF-IDF, and Bag Of Words with neural networks such as CNN and LSTM. The results show that such deep learning methods outperform state-of-the-art char/word n-gram methods by 18 points on the F-measure. The best accuracy was obtained when combining deep neural networks with gradient boosted decision trees.

Joulin et al. (2016) explores a simple and efficient baseline for text classification that is often able to keep up with deep learning methods in terms of accuracy, and is significantly faster for training and evaluation. Their classifier is called fastText and supports training continuous bag of words (CBOW) or skip-gram models using negative sampling, softmax, or hierarchical softmax loss functions. FastText is evaluated on the tasks of sentiment analysis and on a tag prediction dataset. The result shows that FastText is able to compete with deep learning models such as LSTMs and CNNs while being much faster to train.

Park and Fung (2017) detected racist and sexist language through a two-step approach of combining two classifiers - one to classify abusive language and another to classify a specific type of sexist and racist comments given that the language is abusive. Then this is compared with a one-step approach of doing one multi-class classification. They used three CNN-based models: CharCNN, WordCNN, and HybridCNN, on 20k Twitter comments. The best performance was achieved with HybridCNN and the worst with CharCNN. They found that when two logistic regressions were combined, they performed similar to one-step HybridCNN, and performed better than the one-step logistic regression.

In the work of Zhou et al. (2015), both the strengths of CNN and LSTM are combined. They propose a novel and unified model called C-LSTM for sentence representation and text classification. C-LSTM utilizes the CNN to extract a sequence of higher-level phrase representations. These sequences are are fed into a long short-term memory recurrent neural network (LSTM) to obtain the sentence representation. The architecture is evaluated on a sentiment classification task on movie reviews and a question classification task on the TREC question dataset. The results show that the C-LSTM outperforms both CNN and LSTM and can achieve excellent performance on these tasks.

Researchers from Google AI Language introduced in 2018 a new state-of-the-art language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers (Devlin et al., 2018). This model is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context on all layers. This technique can be used for a wide range of tasks, such as question answering, language inference, and text classification, without configuring specific architecture modifications. To evaluate their model performance, they tested BERT on the General Language Understanding Evaluation (GLUE) benchmark , the SQuAD v1.1, and the SQuAD v2.0. GLUE is a collection of diverse natural language understanding tasks which include multiple datasets. SQuAD stands for Stanford Question Answering dataset and is a collection of 100k crowdsourced question/answer pairs. The results showed that BERT was outperforming other state-of-the-art models. The GLUE score was pushed to 80.5% (7.7% absolute improvement) and SQuAD v1.1 got a F1 score of 93.2 (1.5 point absolute improvement).

A team of the Facebook AI Research introduced a new architecture (VDCNN) for text processing which operates directly at the character level and uses only small convolutions and pooling operations (Conneau et al., 2016). They show that that the performance of this model increases with the depth: using up to 29 convolutional layers, they report improvements over the state-of-the-art models on several public text classification tasks. The model is evaluated on eight large-scale datasets which cover several classification tasks such as sentiment analysis and topic classification. The number of classes of the datasets varies between the 2 and 14. The model is compared against other architectures of CNNs. The VDCNN outperformed other CNNs on all datasets except of the two smallest datasets.

Researchers from the Shandong University and the National University of Singapore, have proposed another new model for text classification (Du et al., 2018). Deep learning methods have achieved promising performance in the text classification task but ignore the fine-grained (matching signals between words and classes) classification clues since the classification mainly rely on text-level representations. To tackle this problem Du et al. (2018) introduced the interaction mechanism that incorporates word-level matching signals into the text classification task. The model is called Explicit interAction Model (EXAM). To evaluate EXAM, six text classification datasets corresponding to sentiment analysis, news classification, question answering, and ontology extraction were used. To demonstrate the effectiveness of EXAM, they compared it with several state-of-the-art baselines. The baselines have three variations: models based on feature engineering, char-based deep models, and word-based deep models. The results showed that EXAM outperformed the other models based on precision, recall, and F1 score.

3 Data

The dataset used for this research is provided by Quora through their "Insincere Questions Classification" challenge on Kaggle [1]. Quora is a platform to gain and share knowledge where you can ask any question and get answers from different people with unique insights. At the same time, it is important to handle toxic content by removing insincere questions. According to Quora's policy, insincere questions are defined as ones that meet any of the following criteria:

- **Has a non-neutral tone**: such as an exaggeration to emphasize a comment about a particular group of people
- **Is disparaging or inflammatory**: such as an attempt to seek confirmation of a stereotype or present a discriminatory remark, a comment based on an outlandish premise about a group of people, or the disparaging of a natural, not fixable, or immeasurable characteristic
- **Is not grounded in reality**: such as a reference to false information or absurd assumptions
- **Uses sexual content for shock value**: such as references to pedophilia, incest, or bestiality

The dataset consists of 1,306,122 rows and 3 columns. A snapshot of the data can be found in Figure 1. The three features are:

- **qid** - unique question identifier
- **question_text** - Quora question text
- **target** - questions labeled "insincere" have a value of 1, otherwise 0

	qid	question_text	target
1	00013ceca...	Which babies are more sweeter to their parents? Dark skin babies or light skin babies?	1
2	0000412ca...	Why does velocity affect time? Does velocity affect space geometry?	0
3	0000455df...	Can I convert montra helicon D to a mountain bike by just changing the tyres?	0
4	0000e9157...	Has the United States become the largest dictatorship in the world?	1
5	000021653...	How did Quebec nationalists see their province as a nation in the 1960s?	0

Figure 1: Dataset snapshot

From the 1.306.122 observations 1.225.312 are classified as 0 and 80.810 as 1. In other words, only 6% of the questions is labeled as insincere which makes this dataset highly imbalanced.

3.1 Data preprocessing

Since we are working with text data, different preprocessing steps need to be taken before we can feed it to the algorithm. These steps include:

- **Tokenization:** This is the process of breaking up a sequence of strings into pieces such as words, n-grams, keywords, phrases, symbols, and other elements called tokens
- **Removing white space, punctuation, numbers, and symbols**
- **Transform the words to lower case words**
- **Removing stopwords:** Words such as "is", "am", "are", "this", or "a" are considered as noise in the text and should be removed

[1] https://www.kaggle.com/c/quora-insincere-questions-classification

- **Stemming:** This is the process of linguistic normalization, which reduces words to their word root or cuts off the derivational affixes. For example: "jumps", "jumped", and "jumping" become "jump"

3.2 Data analysis

In this section, further data analysis is conducted to explore the questions. First, we look at the most occurring bigrams. Then, we perform a sentiment analysis on the text. Lastly, we model the different topics of the questions.

After committing the preprocessing steps steps, we can look at n-grams. This are sequences of n items from a given sample of the text. In Figure 2 we can see the most occurring bigrams for both target variables.

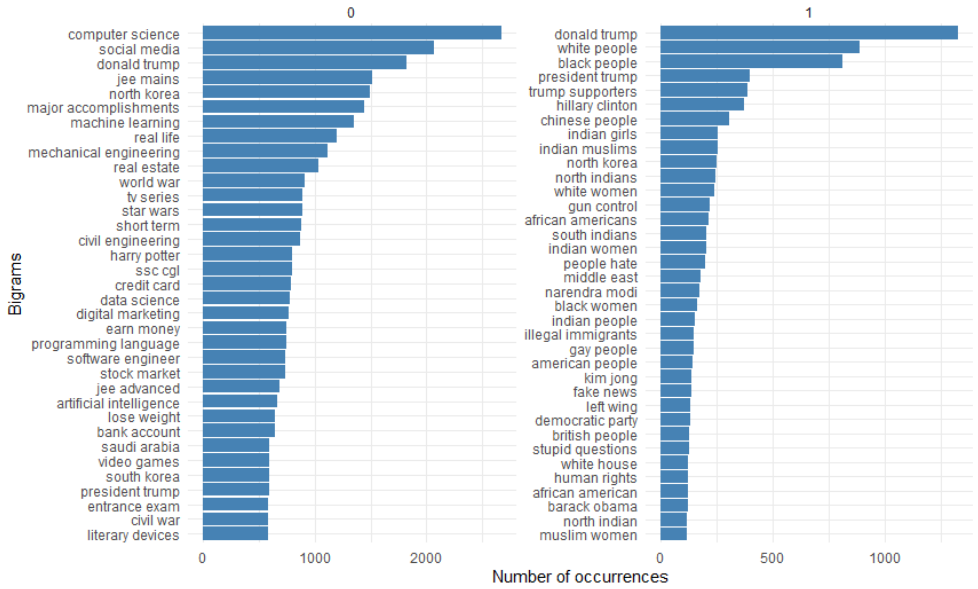

Figure 2: Top occurring bigrams for both targets: 0 (left) and 1 (right)

From Figure 2, we can observe that there are numerous bigrams popular in both target groups. In order to get an idea of which words occur more often among the target class, the frequencies are compared in Figure 3. Words near the red line such as "people", "India", "country", "guy", and "talk" are used with about equal frequencies in both sentences.

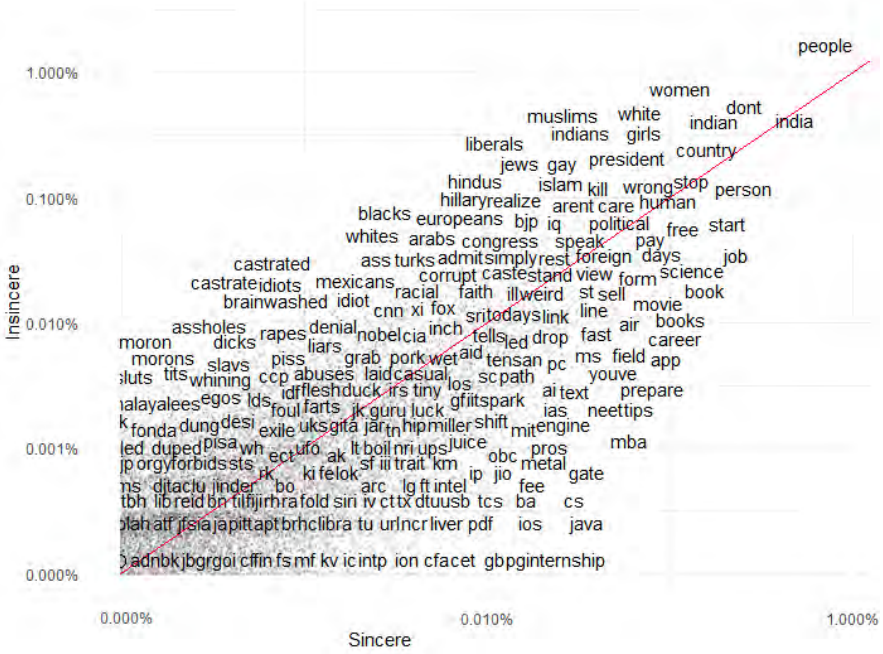

Figure 3: Frequency of the words among target class

3.2.1 Sentiment analysis

Sentiment analysis is the process of identifying and extracting subjective information from text or speech. It is a way to evaluate written or spoken language to determine if an expression is positive, negative, or neutral. With the help of the AFINN (Nielsen, 2011) lexicon we can extract the sentiment of the words. The AFINN lexicon is a list of English terms that is manually rated for positive/negative sentiment with an integer between -5 (negative) and +5 (positive). In Figure 4, the distribution of the sentiments is shown among the target classes. The dashed line indicates the mean of the frequency. As we can expect, in the insincere questions the sentiment tends to be more negative. To give an indication of the most positive and negative questions among the target classes, we can display them graphically in Figure 5.

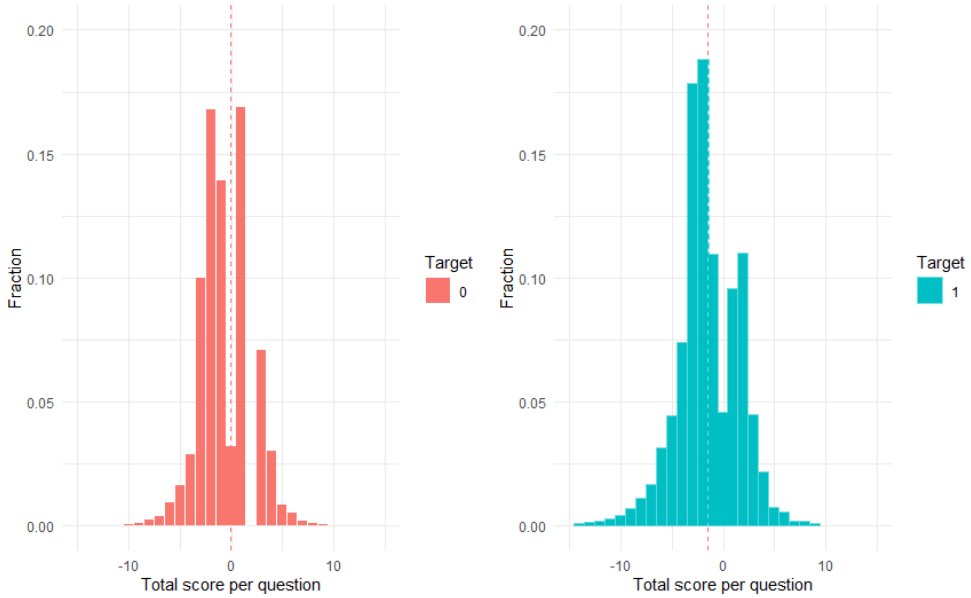

Figure 4: Distribution of the sentiment score

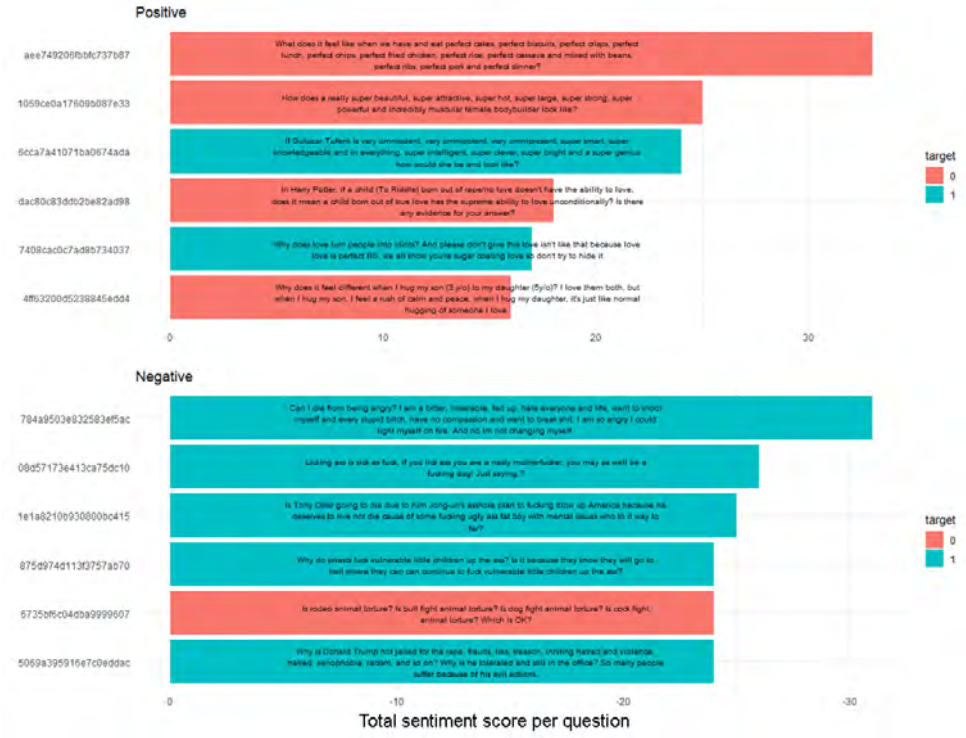

Figure 5: Most positive and negative questions

3.2.2 Topic modeling

Topic modeling is a text mining tool for the discovery of semantic structures in a text body. Often, we want to divide documents in groups or topics so that we can understand them separately. Topic modeling is a method for unsupervised classification of documents, similar to clustering on numerical data.

Latent Dirichlet Allocation (LDA) is one of the most common algorithms for topic modeling. This is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words (Blei et al., 2003). The algorithm tries to find the mixture of words that is associated with each topic, while determining the mixture of topics that describes each document (Silge and Robinson, 2017). The most relevant terms for each topic found in the data are: "people", "India", "time", and "life".

4 Models

This section describes the different types of neural networks that we are going to use. First, the Artificial Neural Network (ANN) will be be explained and thereafter we will expand this to more complicated networks like the Recurrent Neural Network (RNN), Long Short-Term Memory Networks (LSTM), and Gated Recurrent Units (GRU).

4.1 Artificial Neural Networks

Artificial neural networks are systems inspired by the neural networks in a biological brain. Neural networks consists of input, hidden, and output layers, which contains nodes that transform the input into useful output. Each node in the layer is connected to the node in the next layer. In Figure 6, a simple example of an ANN is shown.

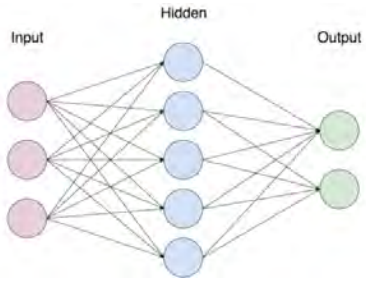

Figure 6: ANN

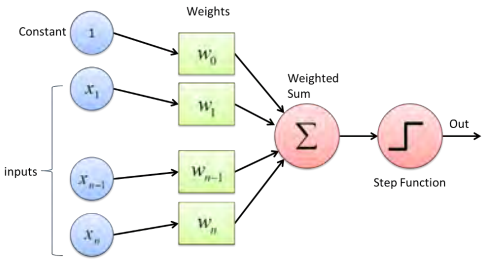

Figure 7: A perceptron

In order to understand ANNs we need to understand what a perceptron is. A perceptron is a single layer neural network and is used as a linear or binary classifier (Figure 7). All the inputs $x_i, i = 1, .., n$ are multiplied with their weights $w_i, i = 1, .., n$ and a bias $b_i, i = 1, .., n$ is added. The bias value allows us to shift the activation function to fit the data better. Next, the sum of the weighted inputs and bias are passed through an activation function. This activation function is used to determine the output of the perceptron. A popular activation function is the sigmoid function. Other common activation functions are shown in Figure 8.

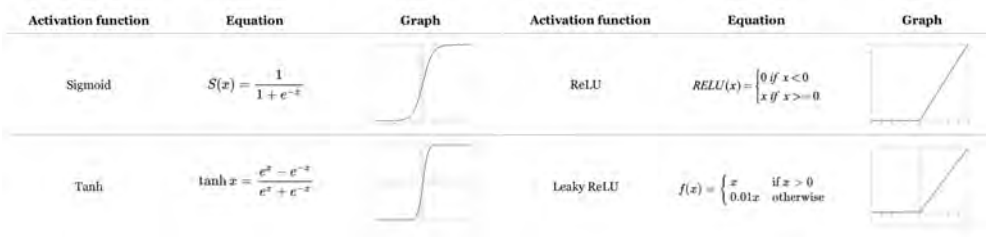

Figure 8: Activation functions

In order to train a neural network and make predictions, we need to adjust the weights in such a way that the predicted value of the network is the closest to the actual value. Therefore, we need to state a loss function to quantify how far off the prediction is from the actual values. The goal of training the neural network is to minimize this loss. This can be done with stochastic gradient descent (SGD) using backpropagation as a gradient computing technique. Stochastic gradient descent is an optimization method that tells us how to change the weights and biases in order to minimize the loss function. The update equation of SGD is shown in Formula 1.

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i} \tag{1}$$

In this formula, η is the learning rate that controls how much the weights are adjusted with respect to the loss gradient. For more information on SGD and backpropagation,

please refer to the book of Kriesel (2007).

4.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a subclass of the ANNs and are popular in many NLP tasks. In traditional feed forward neural networks we assume that all inputs an outputs are independent of each other. But if we want to predict the next word of a sentence, the previous words are required and need to be remembered. RNNs are capable of predicting such sequential input accurately because of the presence of the feedback loops in the network. In Figure 9, an unrolled recurrent network is shown. It can be seen as multiple copies of the same network where x_t represents the inputs at time t, the A's represent a chunk of the network, and h_t represents the output at time t. This network allows the previous outputs to be used as inputs while having hidden states. In this way, the network keeps remembering the context while training (Olah, 2015).

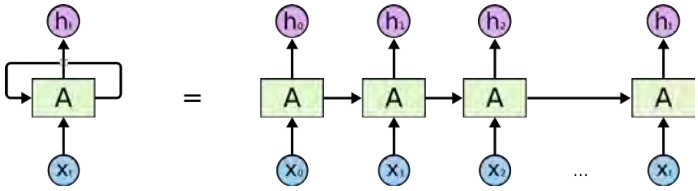

Figure 9: An unrolled recurrent neural network (Olah, 2015)

Even though, RNNs are capable of remembering short term dependencies, they have difficulties of learning long-term dependencies. For example, when a model is trying to predict the next word in the following text: "The clouds are in the ...". It can easily been seen that the last word should be *"sky"*. But when the model is trying to predict the last word in the sentence: "I grew up in the Netherlands... I speak fluent *Dutch*". The model needs the context of the word "Dutch" from further back. This means that when the gap between the relevant information and the word that needs to predicted grows, RNNs becomes unable to learn and connect this information. This is called the vanishing gradient problem for RNNs and is graphically shown in Figure 10.

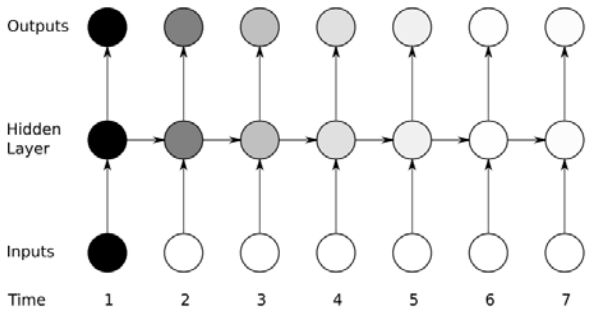

Figure 10: The vanishing gradient problem for RNNs. (Graves, 2008)

We can observe from Figure 10 that the shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time, as new inputs overwrite the activations of the hidden layer and the network 'forgets' the first inputs (Graves, 2008).

4.3 Long Short-Term Memory

Long Short Term Memory networks (LSTMs) are a special subclass of RNNs and were introduced by Hochreiter and Schmidhuber (1997). These networks are capable of processing sequential data and learning long-term dependencies. The basic unit of an LSTM network is the memory block. This memory block contains one or more memory cells and three adaptive multiplicative gating units shared by all cells in the block (Gers et al., 2003). The three gates are the forget, input, and output gate. In Figure 11, a LSTM cell is shown.

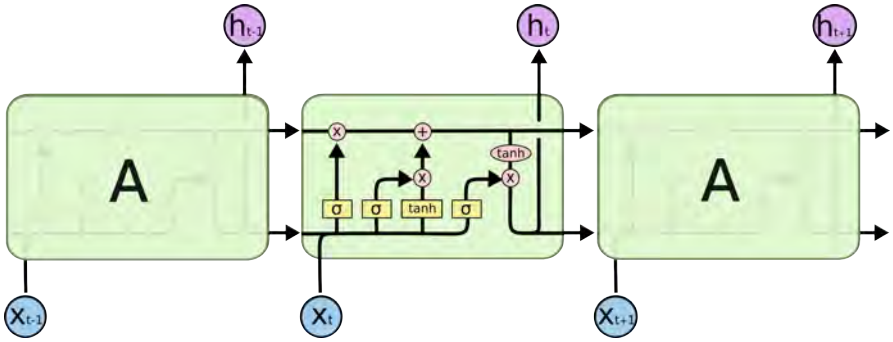

Figure 11: LSTM cells. (Olah, 2015)

Forget gate

The forget gate decides what information should be forgotten or memorized. Information from the previous hidden state and information from the current input is passed through the sigmoid function. The output of the sigmoid function is bounded between 0 and 1. The closer to 0 means to forget, the closer to 1 means to memorize. In figure 11, we see that the forget gate gets the input of the current time step t, x_t, and the output from the previous time step, h_{t-1}. The weighted sum of these inputs is taken, a bias b_f is added, and the resulting value is passed into a sigmoid activation function (Graves, 2013). This results in Formula 2:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2}$$

Input gate

The input gate regulates the information that flows into the cell. It gets the same input as the forget gate and decides what new information is going to be stored in the cell state. Again, the weighted sum of the inputs is taken, a bias is added and passed into a sigmoid function. Next, a tanh layer creates a vector of new candidate values $\tilde{C}_t$, that could be added to the state (Olah, 2015). Both functions are combined to create an update of the state and are shown in Formula 3.

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_f h_{t-1} + b_i) \\ \tilde{C}_t &= tanh(W_c x_t + U_t h_{t-1} + b_C) \end{aligned} \tag{3}$$

The old cell state C_{t-1} is updated to the new cell state C_t. This is done by multiplication of the old state f_t and adding $i_t * \tilde{C}_t$ to obtain the following formula:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

Output gate

The output gate decides the amount of information inside the cells that is exposed to the external network. First, the information is passed through a sigmoid layer. Then, it goes through a tanh layer and is multiplied by the output of the input gate. The equations are shown in Formula 6.

$$\begin{aligned} o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{5}$$

4.4 Gated Recurrent Units

Gated Recurrent Units (GRU) is a variation of the LSTM model and was introduced by Cho et al. (2014). A GRU has a similar architecture as the LSTM, but has fewer parameters as it lacks an output gate. The GRU has two gates, a reset gate r and an update gate z. The reset gate determines how to combine the new input with the previous memory and the update gate defines how much of the previous memory needs to be memorized (Britz, 2015). In Figure 12, a GRU is graphically shown. The equation for the GRU are as follows:

$$\begin{aligned} z &= \sigma(x_t U^z + s_{t-1} W^z) \\ r &= \sigma(x_t U^r + s_{t-1} W^r) \\ h &= tanh(x_t U^h + (s_{t-1} * r) W^h) \\ s_t &= (1 - z) * h + z * s_{t-1} \end{aligned} \tag{6}$$

The main differences between the GRU and LSTM are:

- A GRU has two gates, an LSTM has three gates
- GRUs do not possess and internal memory (c_t) that is different from the exposed hidden state. They don't have the output gate that is present in LSTMs
- The input and forget gates are coupled by an update gate z and the reset gate r is applied directly to the previous hidden state. Thus, the responsibility of the reset gate in a LSTM is split up into both r and z

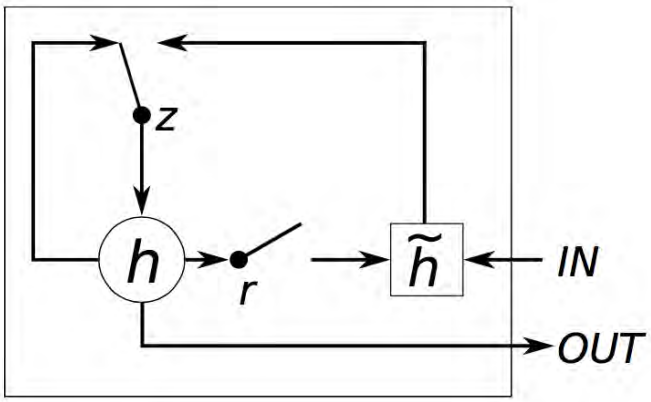

Figure 12: GRU (Cho et al., 2014)

5 Experiment

The goal of this experiment is to predict the insincere questions as accurately as possible. For this prediction, the LSTM and GRU are used. To do a righteous evaluation, the dataset is split into a train, validation-, and test set of 65% , 15%, and 20% respectively. The validation set is used during training to check the model performance. When the model is trained, we can make predictions on the unseen test data to evaluate a final performance. If the performance of the validation set and the test set are close to each other, we know that the model is not overfitting. Table 1 describes the distribution of observations between the train- and validation set.

Table 1: Distribution of observations between train- and validation set

	train	**validation**	**test**	**total**
%	65	15	20	100
observations	888,163	156,735	261,224	1,306,122

5.1 Feature engineering

Besides the data preprocessing steps described in Section 3, extra count features are added. These features are common in Text Mining classification tasks and count the number of: sentences, words, capital letters, and punctuation in the text. Next to this, the length of the text is also added.

5.2 Configuration

Within Deep Learning, there are a vast amount of choices to make for building a Neural Network. Different architectures, hyperparameters, and optimization methods can be chosen. In this section, the configurations of the GRU and LSTM are explained.

Word embeddings

In both models, an Embedding layer is added to the input layer. The Embedding layer is used to create numerical vectors for incoming words, since Neural Networks require numerical input. A general approach for converting words to numerical vectors is to one-hot encode the text. This will lead to substantial data sparsity and computations will be inefficient since most values will be zero. To prevent this, a dense distributed representation for each word is used. 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. This representation is called a word embedding. The weights for the Embedding layer can either be initialized with random values, or more commonly, they are initialized with pre-trained word embeddings such as: Word2Vec, Glove or Wiki-news. In this experiment we use the Wiki-news word embedding. For more information about word embeddings please refer to (Mandelbaum and Shalev, 2016).

Loss function

A loss function, or a cost function is, a method to quantify how far off the prediction is from the actual values. The goal of training a model is to minimize this loss. The most common loss functions in machine learning are the Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the binary cross entropy.

In this research, the binary cross entropy is used. This is one of the most common loss function for binary classification. The categorical cross entropy loss measures the dissimilarity between the true label distribution y_i and the predicted label distribution

$\hat{y}_i$, and is defined as cross entropy (Koidl, 2013).

$$L_{cross-entropy}(y_i, \hat{y}_i) = -\sum_i y_i log(\hat{y}_i) \tag{7}$$

5.2.1 GRU

Architecture

The GRU has the following architecture:

- Input layer: The shape of the input layer is the length of the question text that we set equal to 80. This means that there are 80 neurons in the input layer
- Embedding layer
- GRU layer: with 80 neurons
- Dense layer: this is a fully connected layer with 128 neurons and a ReLu activation function
- Drop out layer: this is a regularization method that randomly drops neurons to prevent overfitting. The drop out rate is set to 0.25
- Dense layer: 64 neurons and a ReLu activation function
- Drop out layer with rate 0.25
- Output layer of 1 neuron with a sigmoid function

Hyperparameters

This model is trained with 25 epochs and a batch size of 4096. The optimization method for this model is Adam. Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iteratively based in training data. It is a popular optimization method for deep learning. Next to this, we use early stopping that will stop training the model when the performance is not improving anymore. This prevents overfitting. The loss function that we use is the binary cross entropy which is explained in Subsection 5.3.

5.2.2 LSTM

Architecture

The LSTM has the following architecture:

- Input layer: 80 neurons
- Embedding layer
- LSTM layer
- Drop out layer with rate 0.25
- Dense layer: 128 neurons and a ReLu activation function
- Output layer: 1 neuron with a sigmoid function

Hyperparameters

This model is trained with 25 epochs and a batch size of 4096. The optimization method is again Adam and the loss function is the binary cross entropy. Also early stopping is used.

5.3 Evaluation measures

In order to evaluate the models, different evaluation measures are discussed. The most common evaluation methods for binary classification are based on the Confusion Matrix. This matrix is shown in Figure 13. We can define the terms as follows:

- True Positives (TP): The model predicted positive and it is true.
  E.g.: The model predicted the question is toxic which is the case.
- True Negatives (TN): The model predicted negative and it is true.
  E.g.: The model predicted the question is not toxic which is the case
- False Positives (FP): The model predicted positive and it is false.
  E.g.: The model predicted that the question is toxic but it was not
- False Negatives (FN): The model predicted negative and it is false.
  E.g.: The model predicted that the question is not toxic but the question is toxic

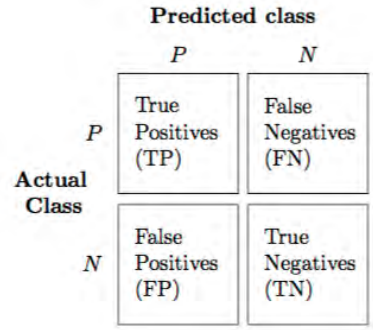

Figure 13: Confusion matrix

From this confusion matrix, some popular performance measures such as accuracy, precision, recall, and f-measure can be derived.

The accuracy is defined as: how often is the classifier correct? This is a poor measure for imbalanced data. Let's say we try to predict the number of fraud cases from 100 observations where there is only 1 fraud case. If our model predicts that there is no fraud at all, the model will have an accuracy of 99%. So even though the model did not predict the fraud it will let you believe that the model performed well. It is calculated by:

$$Accuracy = \frac{TP}{TP + FN + FP + TN} \tag{8}$$

The precision is defined as: out of all the predicted positive classes, how much is predicted correctly. This is a good measure when the costs of False Positive is high. For example: If a email spam detector identifies a mail as spam while it is not (False positive), the user can loose important mails. It is calculated by:

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

The recall is defined as: out of all the actual positive classes, how much is predicted correctly. This is a good measure when the costs of False Negative is high: For example: If a sick patient (Actual Positive) goes through the test and predicted as not sick (Predicted Negative). The cost associated with False Negative will be extremely high if the sickness is deadly. It is calculated by:

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

The F-measure or F_1 score is the harmonic mean of the precision and recall. The F_1 Score is a better measure to use if we need to seek a balance between Precision and Recall and there is an uneven class distribution.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{11}$$

Since we want to have a balance between the precision and recall and we have highly imbalanced dataset, we will use the F_1 score as our performance metric. The objective of the model is to maximize this score.

Threshold

Since the models are returning probabilities that a question is sincere or insincere, we have to convert these probabilities to binary values. An easy way to do this is to say that if the probability is higher than 0.5, the question is insincere. But since this method does not maximize the F_1 score, we have to make a function that looks for the best threshold that does maximizes the F_1 score. This function checks for each threshold in a range from 0.01 to 0.99 what the best F_1 score gives on the evaluation set. We run this function for both the GRU and the LSTM to find the optimal threshold.

5.4 Implementation

The models are implemented in R by using the Keras and Tensorflow packages. Keras is a high-level API to build and train deep learning models on top of Tensorflow. Tensorflow is a backend library for machine learning and is developed by Google. Deep learning models train significantly faster on GPUs, but since the lack of a powerful GPU, the models are trained on the CPU. The runtime of the LSTM is approximately 12 hours and the runtime for the GRU is approximately 15 hours.

6 Results

6.1 Performance

Before we test the model on the test set, it is important to see how the models perform on the validation set. This will give an indication whether the hyperparameters need to be tuned and if the model is overfitting on the training data. In Figure 14 and Figure 15, we can see the loss of the models for both training- and validation set of the GRU and LSTM. We can observe that for both models, the train loss and validation loss does not vary substantially. This means that both models are not overfitting on the train data.

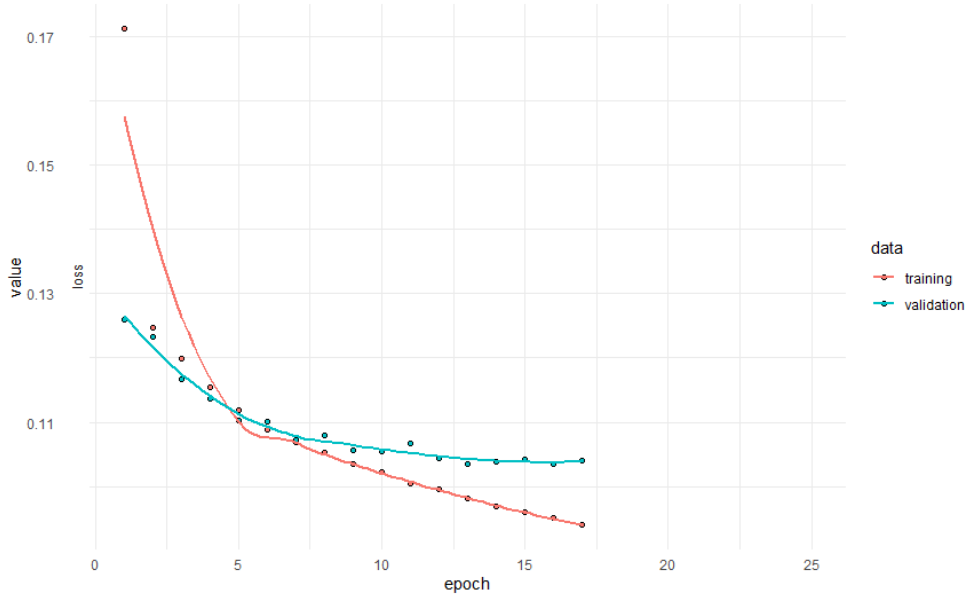

Figure 14: Loss per epoch GRU

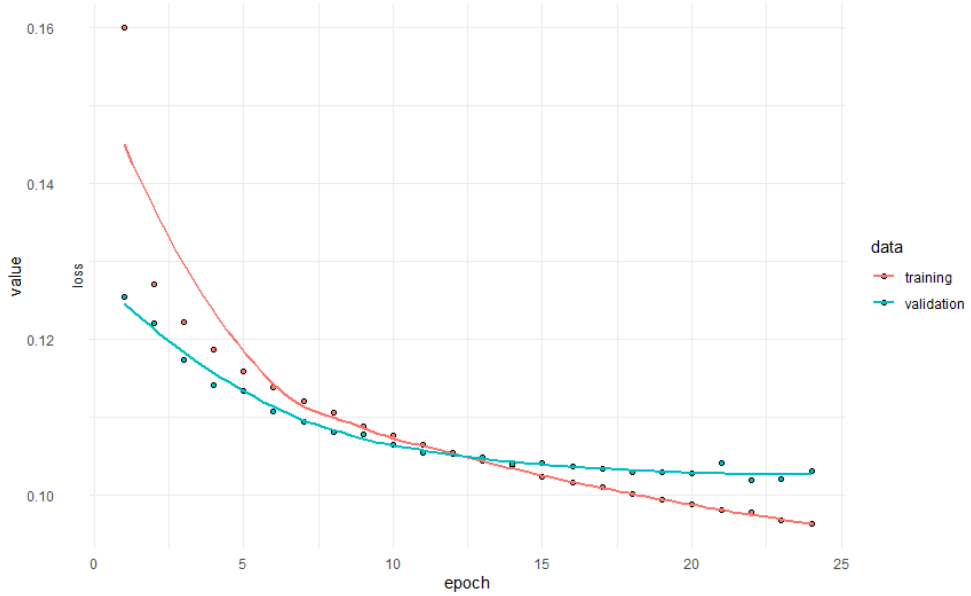

Figure 15: Loss per epoch LSTM

6.2 Model comparison

In order to compare the deep learning models with a more traditional machine learning model, we build a logistic regression model as our baseline. In Table 2, the results of the models on the test set are shown. We found that the LSTM model performed the best in terms of every metric except for the recall. it scores an accuracy of 0.958 a precision of 0.643, a recall of 0.701, and a F_1 score of 0.671. Both the LSTM and GRU outperform the baseline score of the logistic regression.

Table 2: Results

Model	**Accuracy**	**Precision**	**Recall**	F_1 **Score**	**Threshold**
GRU	0.956	0.627	0.713	0.667	0.33
LSTM	0.958	0.643	0.701	**0.671**	0.33
Logistic Regression	0.953	0.629	0.575	0.601	0.33

In Table 3, we compare the F_1 Score of the validation set and the test. As we can see, these scores are close to each other. This implies that the models give almost even performances on the unseen data, which means that the models have a good fit.

Table 3: F_1 score of validation and test set

Model	F_1 **score validation**	F_1 **score test**
GRU	0.663	0.667
LSTM	0.671	0.671

7 Conclusion

The aim of this research was to detect online toxic content using different deep learning models. This paper explained two different types of RNNs that showed promising results in related work in the field of NLP. To evaluate the performance of these deep learning models, a GRU and a LSTM model were trained on the Quora questions dataset. The data analysis gave us insights on the sentiment of the questions and the most common topics that occurred.

The final results showed that both models are performing accurately and are outperforming the baseline model. The LSTM showed the best performance with an F_1 score of 0.671. We can conclude that both deep learning models are capable of flagging an significant amount of toxic content in the questions dataset. Further research could investigate different models and experimental setups.

8 Discussion

The models showed a decent performance, but there are a large number of improvements possible. Due to the lack of time most of these improvements could not be conducted. In this section, these improvements will be discussed and could be contributing for future research.

Data preprocessing: An important step for reducing the amount of words in the dataset is to take care of common mispellings, contractions, acronyms, and replace words with a general term. This could be done by making a dictionary with all the possible words that you can think of and map these words to the right spellings. For example: 'colour':'color', 'isnt':'is not', 'instagram':'social media', 'wwii': 'world war 2'.

Feature engineering: More features could be made and tested. In this experiment, only the count and length features were used. Features such as the number of bad words in the questions, TF-IDF, n-grams and the sentiment scores could also be added.

Models: This paper conducted research on deep learning models and mainly on the variations of the Recurrent Neural Network. As seen in Section 2, related work showed promising results with Convolutional Neural Networks. Besides this, also ensemble models would potentially improve the results when combining a CNN with a LSTM or a GRU with a LSTM. Also, different word embeddings such as Glove or Word2Vec could make a difference in performance. A new kind of model developed by Google which is called BERT (Bidirectional Encoder Representations from Transformers) obtains state-of-the-art results and could be used for further research.

Training: Training deep learning models on a computer without a (powerful) GPU is extremely computational expensive and takes a significant long time to train. This was one of the major drawbacks that this research faced. Due to this problem there was no time to perform cross validation on the test data. Also, a grid search to search for the optimal hyperparameters could not be done. The same holds for testing different model architectures. Furthermore, the performance could be improved by after training the model on the train set, training the models on the train and validation set (80%) and testing on the test set (20%).

Error analysis: By manually investigating the predictions, you can get insights into what leads to certain errors. Checking this for both models would optimize the performance.

References

Badjatiya, P., Gupta, S., Gupta, M., and Varma, V. (2017). Deep learning for hate speech detection in tweets.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. J. Mach. Learn. Res., 3:993–1022.

Britz, D. (2015). Recurrent neural network tutorial, part 4 – implementing a gru/lstm rnn with python and theano. http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. pages 1724–1734.

Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2016). Very deep convolutional networks for natural language processing. KI - Künstliche Intelligenz., 26.

Davidson, T., Warmsley, D., Macy, M. W., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. CoRR, abs/1703.04009.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. CoRR, abs/1810.04805.

Du, C., Chen, Z., Feng, F., Zhu, L., Gan, T., and Nie, L. (2018). Explicit interaction model towards text classification. CoRR, abs/1811.09386.

Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2003). Learning precise timing with lstm recurrent networks. J. Mach. Learn. Res., 3:115–143.

Graves, A. (2008). Supervised sequence labelling with recurrent neural networks.

Graves, A. (2013). Generating sequences with recurrent neural networks. CoRR, abs/1308.0850.

Guynn, J. (2019). If you've been harassed online, you're not alone. https://eu.usatoday.com/story/news/2019/02/13/study-most-americans-have-been-targeted-hateful-speech-online/2846987002/.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9:1735–80.

Hootsuite (2019). The global state of digital in 2019 report. https://hootsuite.com/pages/digital-in-2019.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. CoRR, abs/1607.01759.

Koidl, K. (2013). Loss functions in classification tasks. https://www.scss.tcd.ie/~koidlk/cs4062/Loss-Functions.pdf.

Kriesel, D. (2007). A Brief Introduction to Neural Networks.

Malmasi, S. and Zampieri, M. (2017). Detecting hate speech in social media. CoRR, abs/1712.06427.

Mandelbaum, A. and Shalev, A. (2016). Word embeddings and their use in sentence classification tasks. CoRR, abs/1610.08229.

Nielsen, F. Å. (2011). Afinn.

Olah, C. (2015). Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Park, J. H. and Fung, P. (2017). One-step and two-step classification for abusive language detection on twitter.

Silge, S. and Robinson, D. (2017). Text Mining with R: A Tidy Approach.

Zhou, C., Sun, C., Liu, Z., and Lau, F. (2015). A c-lstm neural network for text classification.