

# **Jigsaw Unintended Bias in Toxicity Classification**



Bachelor's thesis

Electrical and Automation Engineering

Spring 2019

Minh Quan Do

Electrical and Automation Engineering  
Valkeakoski

---

<b>Author</b>	Minh Quan Do	<b>Year</b> 2019
<b>Subject</b>	Jigsaw Unintended Bias in Toxicity Classification	
<b>Supervisor(s)</b>	Juhani Henttonen Xuan Hoai Nguyen	

---

#### ABSTRACT

The purpose of this thesis was to implement a Deep Learning Model to classify the toxicity of online comments. The main focus was on analysing the sentiment of sentences and on classifying this into categories. The data was collected from Jigsaw Unintended Bias in Toxicity Classification competition on Kaggle.

In this thesis, we will have a look at the following concept: Deep Learning, what is Natural Language Processing, what is Recurrent Network (RNN), Long-short Term Memory (LSTM) and Gated Recurrent Units (GRN). Background information was mostly collected from papers and journals from Google Scholar.

The method here was using pre-trained embeddings to represent the relationship of inputs from words into vectors. Then, the data went through Bidirectional LSTMs (GRUs) layer to extract the sentiment of inputs.

The project was implemented successfully and the result was favourable. The model achieved an accuracy of 93% on the public test set and ranked at 1314 out of 2172 contestants.

**Keywords** Deep Learning, Machine Learning, Natural Language Processing (NLP)

**Pages** 30 pages including appendices 5 pages

## CONTENTS

1	INTRODUCTION .....	3
2	BACKGROUND KNOWLEDGE .....	4
2.1	Machine Learning.....	4
2.2	Deep Learning .....	4
2.3	Neural Networks .....	4
2.4	Natural Language Processing (NLP) .....	6
2.4.1	Basic classification techniques .....	7
2.4.1.1.	Naïve Bayes Classifier .....	7
2.4.1.2.	Logistic Regression .....	7
2.4.2	Recurrent Neural Networks.....	7
2.4.2.1.	Backpropagation Through Time (BPTT) .....	10
2.4.2.2.	Long Short-Term Memory Units and Gated Recurrent Units .....	10
3	PROPOSED METHOD .....	17
3.1	Task Description .....	17
3.2	Model Architecture .....	17
3.3	Word Embedding .....	18
3.4	LSTMs and GRUs .....	19
3.5	Training .....	19
4	EVALUATION METRICS .....	20
4.1	Dataset Description.....	20
4.2	Evaluation .....	21
5	RESULTS AND DISCUSSION .....	22
6	CONCLUSIONS.....	23
	REFERENCES .....	24

### Appendices

Appendix 1 MODEL IMPLEMENTATION

Appendix 2 MODEL LOG

## 1 INTRODUCTION

This thesis discusses Machine Learning and Deep Learning. Deep Learning is a subfield of Machine Learning that teaches computers to learn by examples. Deep Learning is an important part of driverless cars, observation system in China, Alpha Go and Alpha Go Zero of OpenAI, etc.

If we take a brief look at the history, before Deep Learning became a worldwide phenomenon, Support Vector Machine (SVM) was preferred since the limitations in the computation power and the data reduced the accuracy of Deep Learning. Not until 2012, when Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton entered a submission that would halve the existing error rate to 16% using a Deep Convolutional Neural Network (AlexNet), Deep Learning has become a phenomenon, gaining back the interest of Data Engineers and Data Scientists. In 2014, DeepFace of Facebook and Generative Adversarial Networks (GAN) of Yann LeCun was introduced. Next, Google's AlphaGo beat Lee Sedol, a top-ranked international Go player from Korea in 2016. Lastly, in 2017, Fei-Fei Li launched the ImageNet (a free dataset of more than 14 million labelled images, which is very important for training neural nets in supervised learning) for researchers, educators and students.

In this thesis, we will focus on the Recurrent Neural Network for Natural Language Processing (NLP) to classify the toxicity of online comments in Kaggle's competition: Jigsaw Unintended Bias in Toxicity Classification.

With the development of social networks, people easily leave a comment online without knowing its consequences. Online comments or posts sentiment analysis can help to solve the problem and help us prevent some events such as: cyber bullying, negative news, negative feedbacks.

The challenge in the competition was the computing limitation and the lack of diversity in the dataset. Hence, it made the model easily got overfitted since there were significantly more zeros than ones. In addition, labels of the test set and cross validation set were hidden; thus, we could not know how good our model performed. Therefore, adjust the model to get less overfitted or underfitted.

However, the goal of the competition was not only classifying inputs as positive or negative but also predicting scores of the inputs. Therefore, it can later be used to predict the severity of a sentence.

## 2 BACKGROUND KNOWLEDGE

### 2.1 Machine Learning

Machine Learning is a subfield of Artificial Intelligence (AI) that is said “to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ ” (Mitchell, 1998, p. 2). That is, machine learning is a set of algorithm, which help the performance of a task better through training by minimizing errors using optimization methods. Some famous machine learning methods are: supervised learning algorithms (e.g. logistic regression, linear regression), unsupervised learning algorithms (e.g. K-nearest neighbourhood, K-mean clustering), semi-supervised learning algorithms (e.g. cluster assumption) and reinforcement learning algorithms.

### 2.2 Deep Learning

Deep Learning is a subfield of machine learning applying algorithms inspired by designs and functions of the brain called artificial neural networks.

Deep artificial neural networks are a collection of algorithms that have set breakthrough for several real-world problems: image recognition, sound recognition, image processing, recommender systems, healthcare services, autonomous vehicle, Artificial Intelligence, etc. (Mital, 2017).

Deep refers to the quantity of layers in a neural system. A shallow system has one purported concealed layer; on the other hand, a deep systems has multiple. Different concealed layers enable deep neural systems to adapt better highlights of information, since basic highlights recombine starting with one layer then onto the next, to frame increasingly complex highlights. Nets with numerous layers pass input information and highlight their attributes better than nets with less layers. However, Deep Learning requires a large amount of data to achieve a decent accuracy and avoid overfitting. Moreover, since Deep Learning takes matrices as input, it requires lots of computational power to make it faster; therefore, the deeper the network is, the more computational power it needed.

### 2.3 Neural Networks

Neural networks (Sarle, 1994) are a group of algorithms, imitating the human brain, translating sensory information using computer perception, classifying or clustering raw input; As a result, recognize a pattern and embedded the pattern as vectors or matrices; as a result, all real-world information such as: images, sounds, texts or statistics, have to be translated.

Neural networks facilitate us cluster and classify. They assist to cluster untagged information in keeping with similarities among the inputs, and classifying information after their labelled of training dataset. Neural networks may also extract features that are fed into learning algorithms for clustering and classification.

Deep Learning is the name we tend to refer to networks are composed of many layers; each layer is built from nodes. A node is the core where computation happens, imitating a neuron cell in a human brain, that activates once it encounters adequate impulses, combining inputs from the information to form a collection of coefficients (weights), that either magnify or lessen that input, thereby applying the trained matrices to inputs with relation to the task correspond to the algorithmic rule; e.g. that input is most useful is classifying information while not error? These input-weight product are totalled then the sum is responded to a node's supposed activation perform, to work out whether or not and to what extent that signal ought to progress additional through the network to have an effect on the final word outcome, say, associate degree act of classification. If the signal passes through, the neuron is considered as "activated."

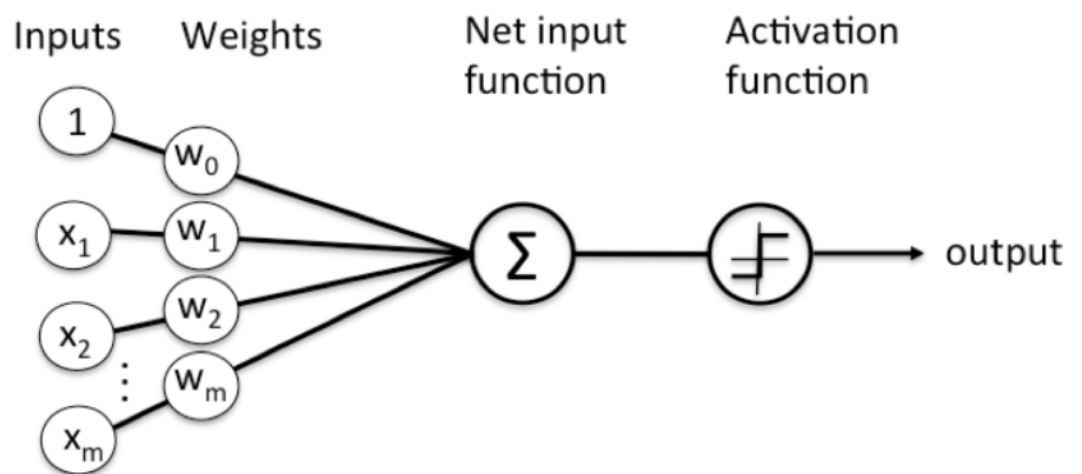


Figure 1: A node (SkyMind.ai, n.d.)

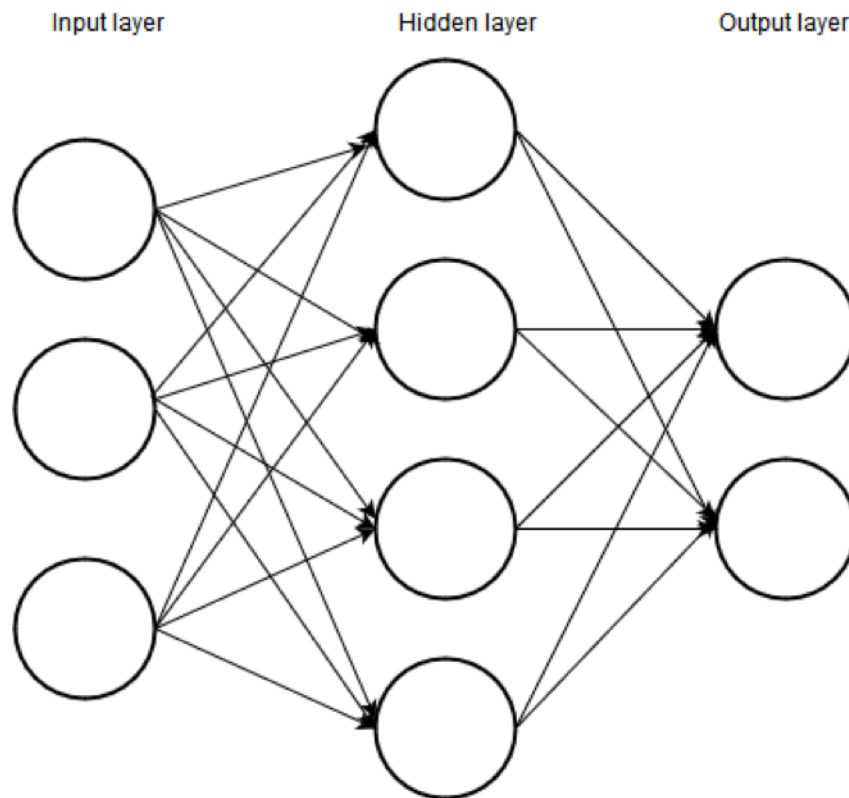


Figure 2: Simple neural networks layers

- Input layer: where data is fed into the network
- Hidden layer: containing activation functions, at this layer, the output of the previous layer is being processed (activated). In this case, the output of the input layer. A neural network can have more than one hidden layer.
- Output layer: where output from hidden layers is turned into probabilities of the target label.

## 2.4 Natural Language Processing (NLP)

Natural language processing (NLP) is a branch of Artificial Intelligence that helps computers understand, analyse and work on natural language. NLP follows many disciplines, including computer science and computational linguistics, in order to fill the gap between human communication and computer understanding. (Skymind.ai, n.d.).

Some applications of NLP are: Information Retrieval, Information Extraction, Machine Translation, Sentiment Analysis, Spam Filter, Speech Recognition, Natural Language Generation, etc. (Skymind.ai, n.d.).

## 2.4.1 Basic classification techniques

### 2.4.1.1. Naïve Bayes Classifier

The most basic and simple probabilistic classifier in machine learning based on Bayes' Theorem. Theoretically, naïve Bayes is a conditional probability model: probability of an event (some particular situation occurring) given that another event has occurred (Naïve Bayes Classifier, 2019).

- Bayes Theorem:

$$p(C_k | x) = \frac{p(C_k)p(x|C_k)}{p(x)} \quad (1)$$

- Gaussian Naïve Bayes

$$p(x = v | C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}} \quad (2)$$

### 2.4.1.2. Logistic Regression

Logistic Regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are using a logistic function (Logistic Regression, 2019).

- Linear Regression

$$f(x) = w^T x \quad (3)$$

- PLA

$$f(x) = \text{sgn}(w^T x) \quad (4)$$

In the formula, the activation function, the matrix  $w$  and the inputs  $x$  output a graph. Then, using a threshold, we can classify the input.

## 2.4.2 Recurrent Neural Networks

In deep learning, we can use Recurrent Neural Network to deal with NLP. The problem with traditional neural network is that they do not consider the relationship between inputs. Recurrent neural network can model dynamic systems, where to previous outputs depend on past values of outputs and inputs also. Ordinary neural network is a static model between inputs and outputs.



Recurrent neural networks address the issue above. They are networks with their own loops, allowing information to endure.

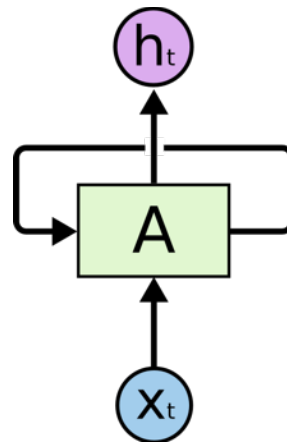


Figure 3: Folded Recurrent Neural Network (Do., 2017)

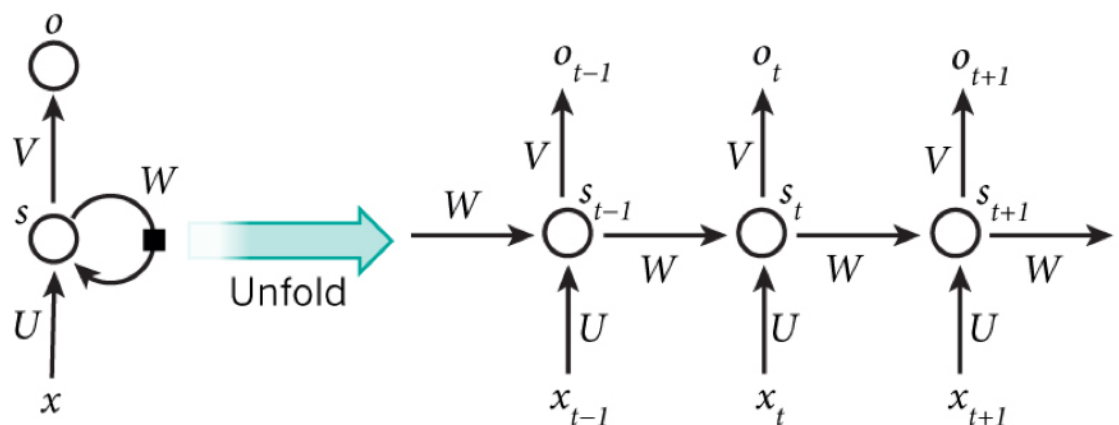


Figure 4: Unfolded Recurrent Neural Networks (Do, 2017)

Explanation of RNNs:

- $x_t$ : the input at the time step  $t$  (one hot vector or word embedding)
- $s_t$ : the hidden state at the time step  $t$ 
  - The memory of the network

$$s(t) = f(Ux_t + Ws_{t-1}) \quad (5)$$

- The function  $f$ : nonlinearity such as tanh or ReLU
- $s_{-1}$ : used to compute  $s_0$  (initialized by zero)
- $o_t$ : the output at the time step  $t$

- For example:

$$o_t = \text{softmax}(Vs_t)$$

With  $V$  is vocabulary

A chunk of neural network,  $A$ , takes some input  $x_t$  and produces an output  $o_t$  in the figure above.  $A$  allows the transmission of information from

one network step to the following. One can think of a recurrent neural network as multiple copies of the same network, each passing an information to a successor.

Recurrent networks are distinguished from feedforward networks by means of that it connected to their past selections, taking their very own previous outputs as input. It is regularly said that recurrent networks have reminiscence, including memory to neural networks has a reason: there is information inside the serie, and recurrent nets use it to carry out tasks that feedforward networks cannot.

Some notes on RNNs:

- $s_t$ : the memory of the network
  - $s_t$ : captures information from previous steps
- RNN uses the same parameters (U, W, V) for all steps
  - Performing the same task on different inputs
  - Reducing the number of parameters
- The output  $o_t$  is not necessary for all problems
  - Depending on each task
    - Sentiment prediction: only needs the final state
- The need of input  $x_t$ : also depends on each task
- Main feature of RNN
  - The hidden states
  - Capture hidden representation of a sequence

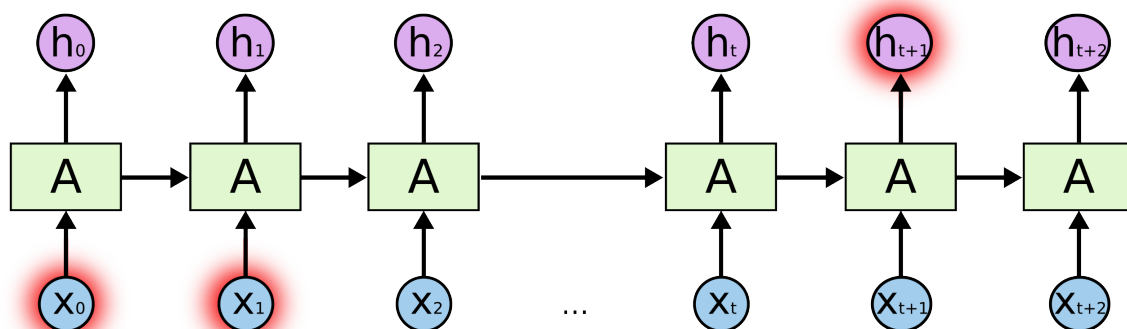


Figure 5: Long-term Dependencies (Do, 2017)

#### 2.4.2.1. Backpropagation Through Time (BPTT)

Backpropagation is an optimization algorithm in Machine Learning. It computes the portion of the error of the output by calculating the derivatives of the inputs, weights and output. After that, the derivatives are then used to adjust the weights matrix using other optimization method, e.g. Gradient Descent, Adam, Adadelta, etc. (Minh-Tien, N., 2019).

Recurrent networks depend on an expansion of backpropagation called backpropagation through time, or BPTT. Time, for this situation, is basically communicated by a well-characterized, ordered series of calculations connecting one time step to the following, which is all backpropagation needs to work (Minh-Tien, N., 2019).

#### 2.4.2.2. Long Short-Term Memory Units and Gated Recurrent Units

The problem of RNNs is Vanishing Gradients, which happens when a network has too many layers and the gradients of the loss function go toward zero, making no update on the weights matrix even though it has not reached the optimal point. This is somewhat because of the information going through many stages of multiplication (Minh-Tien, N., 2019).

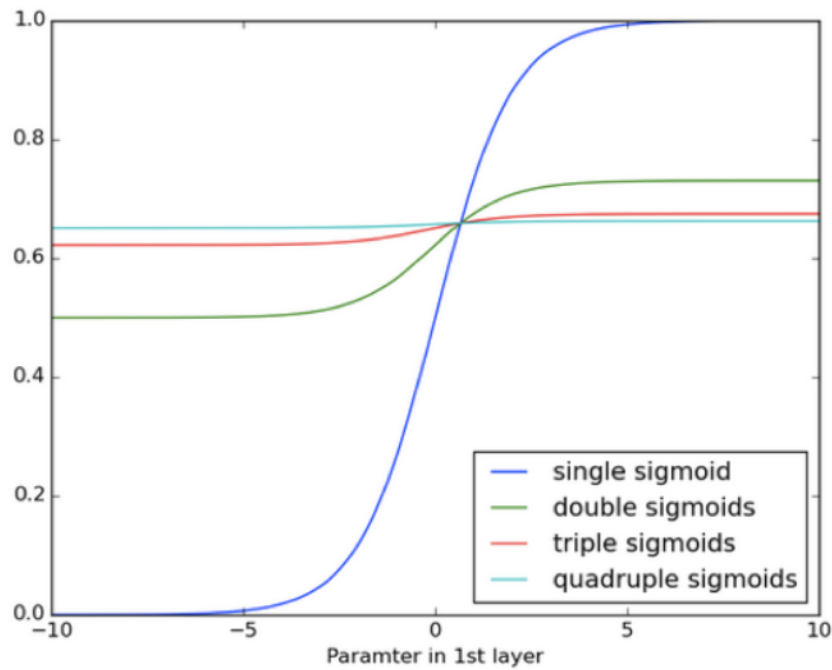


Figure 6: The relation between the number of layers and Gradient Vanishing

As a result, in the mid-90s, two German researchers Sepp Hochreiter and Jürgen Schmidhuber introduced the idea of Long Short-Term Memory Units (LSTMs) (Hochreiter & Schmidhuber, 2006) to encounter the Vanishing Gradient.

LSTMs help prevent the vanishing gradient that can be backpropagated through time and layers by keeping a small amount of error, it allows the RNN to continue learning. The information can be stored, read or written from a cell. Its mechanism let important feature passes through learning from data. That is, cells learn when and what data allow to enter, leave or be deleted through learning.

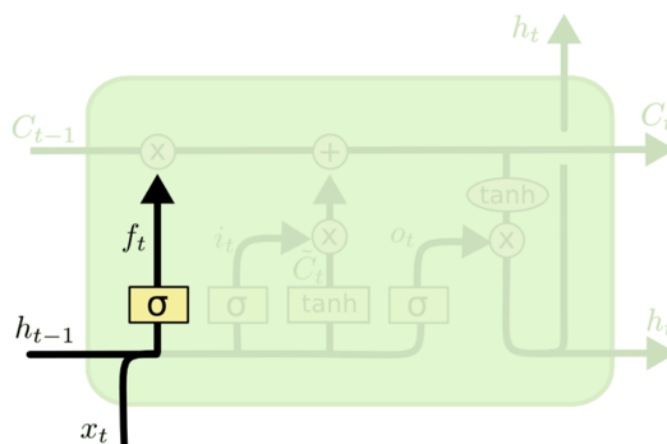


Figure 7: LSTM Forget Gate (Minh-Tien, 2019)

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (6)$$

- Looks at  $h_{t-1}$  and  $x_t$
- For each  $C_{t-1}$ , output
  - 1: completely keep the information
  - 0: completely forget the information
  - Sigmoid function

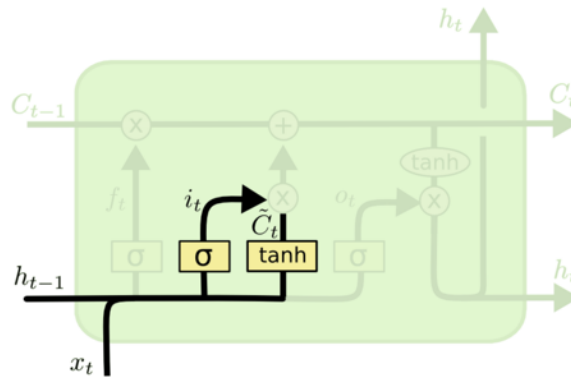


Figure 8: LSTM Storing Gate (Minh-Tien, 2019)

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (7)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (8)$$

- Deciding what new information to store
- Two steps:
  - Sigmoid layer: calls the “input gate layer”
    - Decide which value to be update
  - Tanh layer:
    - Create a vector of new candidate values

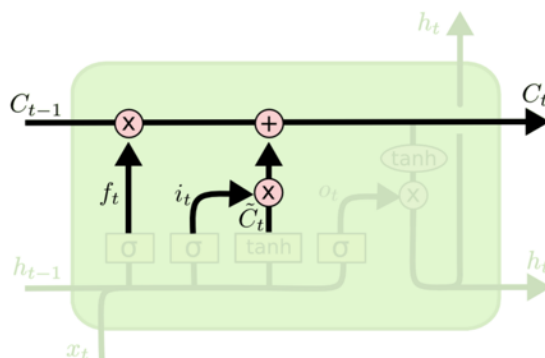


Figure 9: LSTM Update Gate (Minh-Tien, 2019)

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (9)$$

- Update the old cell state to the new cell state
  - Multiply  $C_{t-1}$  with  $f_t$ 
    - Forgetting information we want to forget
  - Adding:  $i_t * \tilde{C}_t$

- A new candidate value
- Selected by how much we decided to update each state value
- We actually drop information about the old subject' gender and add new information

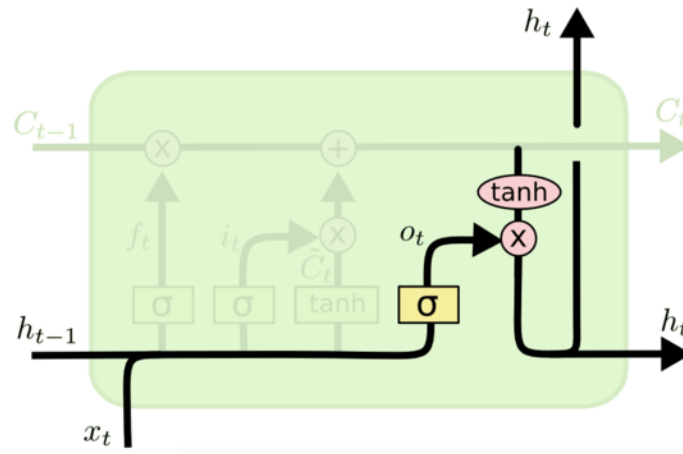


Figure 10: LSTM Output Gate (Minh-Tien, 2019)

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (10)$$

$$h_t = o_t * \tanh(C_t) \quad (11)$$

- Deciding the output of LSTM cell
- Sigmoid layer:
  - Deciding what parts of the cell state are going to output
- Tanh layer:
  - Put the cell state to  $\tanh(-1, 1)$
  - Multiply with the output of the sigmoid gate
  - Only output the parts we decided to
- The subject to the next word is a verb
- From the output of LSTM cell
  - Know the subject is plural or singular
  - Output correct verb

Using the same idea and mechanism with LSTMs, a GRU (Cho et al., 2014) can be considered as a LSTM with less gate and without output gate. As a result, it takes less computational time and fully writes the information from its memory cell to larger net at each time step.

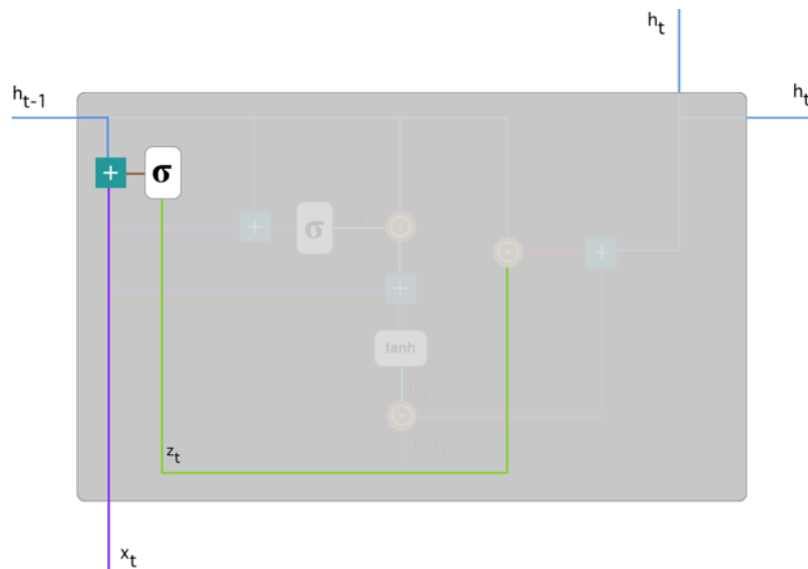


Figure 11: GRU Update Gate (Minh-Tien, 2019)

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \quad (12)$$

- Compute the update gate  $z_t$  for the time step  $t$ 
  - $x_t$  is combined with its weight
  - $h_{t-1}$  is combined with its weight
- Both the output are added together
  - Using sigmoid function
  - Between 0, 1

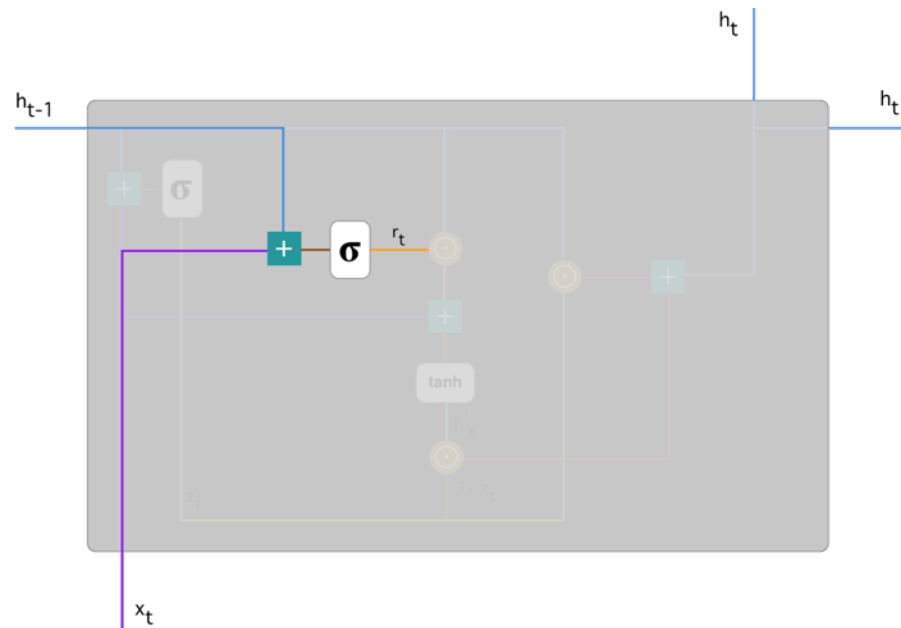


Figure 12: GRU Reset Gate (Minh-Tien, 2019)

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \quad (13)$$

- Compute the reset gate  $r_t$  for the time step  $t$ . Similar to the update gate
  - $x_t$  is combined with its weight
  - $h_{t-1}$  is combined with its weight
- Both the output are added together
  - Using sigmoid function
  - Between 0, 1

The reset gate and the update gate are computed by the 2 formulas (11) and (12). “When the reset gate is close to 0, the hidden state is forced to ignore the previous hidden state and reset with the current input only” (Cho et al., 2014, p. 3). This allows the hidden state to exclude information that is irrelevant in the future. On the other hand, the update gate decides how much information will be forwarded to the current hidden state.

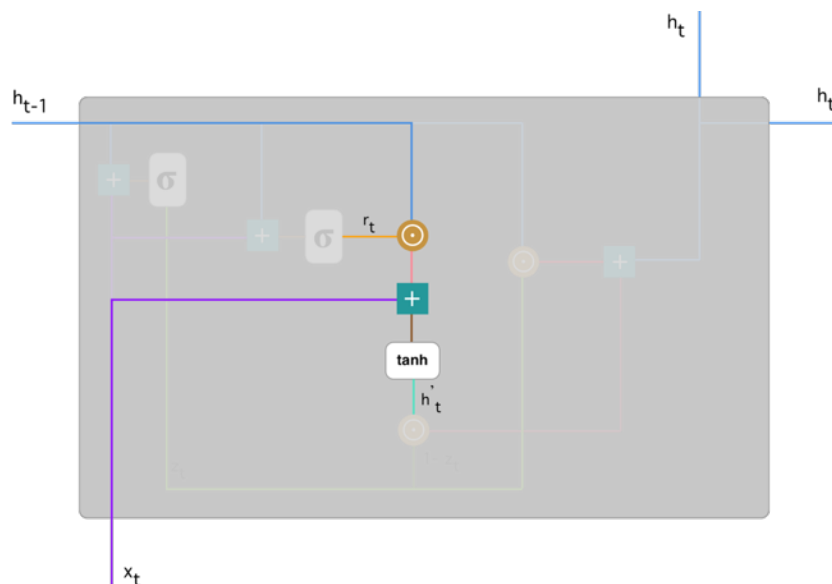


Figure 13: GRU Current Memory Content (Minh-Tien, 2019)

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1}) \quad (14)$$

- Multiply
  - $x_t$  with  $W$  and  $h_{t-1}$  with  $U$
- Hadamard (element wise)
  - Determine what to remove from the previous time steps
- Sum up
- Apply tanh function



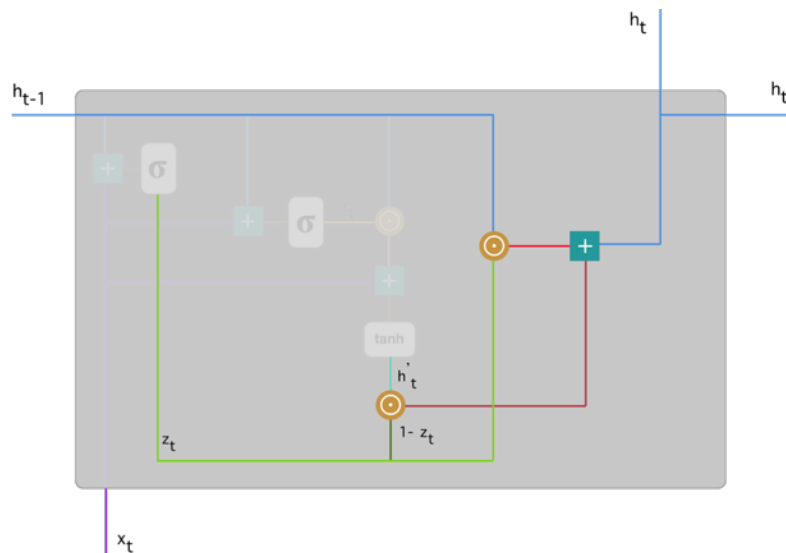


Figure 14: GRU Final Memory at current time step (Minh-Tien, 2019)

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{15}$$

- Need to compute  $h_t$  which holds information for the current time step
  - Apply element wise of  $z_t$  and  $h_{t-1}$
  - Apply element wise of  $z_t$  and  $(1-z_t)$
  - Sum up two parts

After computed all the step, the final memory sums up to give the matrix and the output, this is a result of the update, reset and the shared weights matrix.

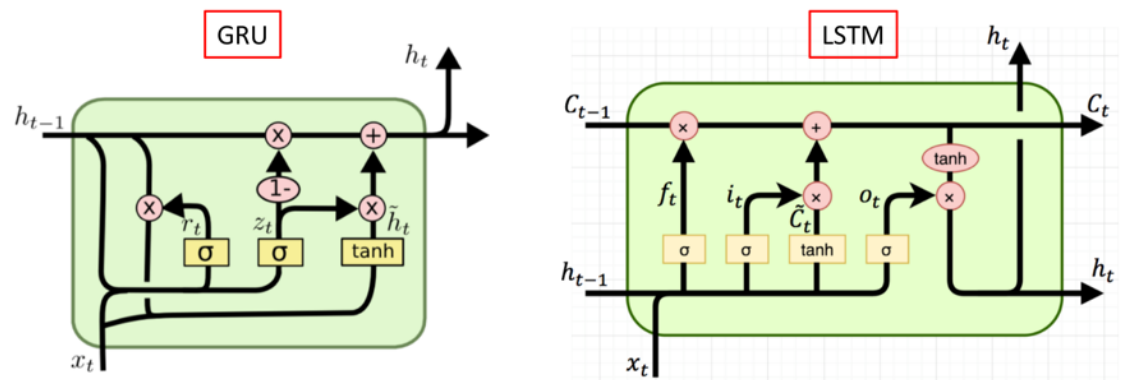


Figure 15: GRU vs LSTM (Minh-Tien, N., 2019)

Table 1: Comparison table between GRU and LSTM

GRU	LSTM
An Update gate which decide whether to pass previous information to the next cell	Two more gates for forget and output
A Forget gate is nothing but has a new set of weight ( $W_t$ )	Two additional math operations with two new sets of weights

### 3 PROPOSED METHOD

#### 3.1 Task Description

Kaggle is an online network of data scientists and machine learning, claimed by Google LLC. Kaggle enables clients to discover and distribute public datasets, investigate and fabricate models in an online information science condition, work with other information researchers and AI designs, and enter rivalries to unravel data science challenges. Kaggle got its start by offering AI competitions and now likewise offering an open data platform, a cloud-based workbench for data science, and short structure AI training.

The challenge of the competition was to detect toxic comments and minimized unintended model bias. Unintended bias in Machine Learning can be seen as systemic differences in performance for different demographic groups, potentially compounding existing challenges to fairness in society at large.

#### 3.2 Model Architecture

Due to the kernels time limitation and the hidden test set labels. We are unable to make a complex models. Therefore, I made tests on different basic models to get the overview which model give the best result.

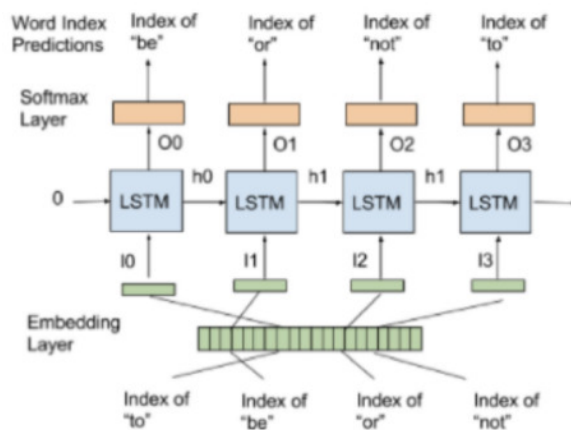
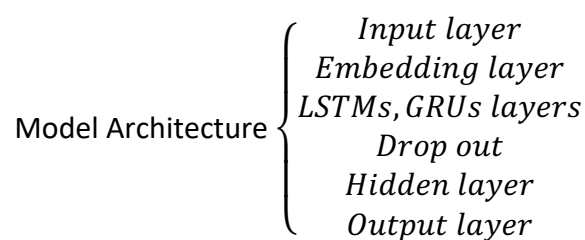


Figure 16: An illustration on the RNNs model (Solomon, n.d.)



Firstly, comments are taken through pre-processing function to remove punctuations and emoticons. The return text is cleaned and lowercased. Secondly, comments are converted into sequence using keras.preprocessing library. Finally, they are sliced and padded to fit the input length.

After that, inputs went through a pre-trained embedding layers, so their relationship can be better represented. In this thesis, I used Facebook's Fasttext with 600 billions tokens embedded in a square matrix of size 300 and Stanford's Glove embedded in a square matrix of the same size using 6 billions tokens. In addition, for model using both word embedding and char embedding, I used Standford's char embedding using 840 billions tokens embedded in a 300 dimension matrix. Then, the input will go through the LSTMs or GRUs to extract their features. Bidirectional recurrent networks allow the networks to extract sentiment from the word before and after of the input. Next, drop out layer removes an entire 1 dimension feature map randomly, preventing model from overfitting. After going through two hidden layers using ReLU activation function, the sigmoid, tanh or softmax function outputs the probabilities of the inputs.

### 3.3 Word Embedding

Word embedding is one of the most popular representations of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. "A fundamental problem that makes language modelling and other learning problems difficult is the curse of dimensionality." (Bengio et al., 2001, p. 1).

In word embedding, each word mapped to one vector illustrate by a real-value 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 and its value learned through training.

The representation is learned based on the usage of words. This allow to represent words are used in a similar ways to have similar representations; therefore, capture their meanings. This could be summarized as: words that have similar context will have similar meanings; as a result, have the similar representations.

Many NLP systems and traditional techniques treats words as atomic units, that is, there is no consideration between similarity of words since they are represented as indices in vocabulary (Mikolov et al., 2013, p. 1); therefore, in 2013, Tomas Mikolov introduced the Word2Vec.

### 3.4 LSTMs and GRUs

As mentioned earlier, when dealing with sequence data, people prefer RNN over CNN (Convolution Neural Network) since it can preserve the history information. And the most common RNNs networks in NLP are LSTMs and GRUs.

### 3.5 Training

An important algorithm of machine learning and deep learning is the loss function. For this project, I used Binary Cross-Entropy loss, this is a popular loss function in binary classification problems:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \quad (16)$$

*where  $y$  is the label (1 or 0) and  $p(y)$  is the predicted probability of the point being 1 for all  $N$  points.*

Another algorithm that makes a significant impact on Machine Learning and Deep Learning is Optimization algorithm. In this project, we used Adam optimizer algorithm (Kingma et al., 2017). The method is computational efficiency, required little memory. The computation of the decaying averages of past and past squared gradients  $m_t$  and  $v_t$  respectively as follow:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (17)$$

$$v_t = \beta_2 v_{t-2} + (1 - \beta_2) g_t^2 \quad (18)$$

$m_t$  and  $v_t$  are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. Additionally, as the  $m_t$  and  $v_t$  are initialize as vectors of 0's, the author of Adam: Kingma and Liu Ba, observed that they bias toward 0, especially during initial time steps or  $\beta_1$  and  $\beta_2$  are close to 1. Hence, they counteract these biases by computing bias-correct  $m_t$  and  $v_t$ :

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (19)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (20)$$

And then use these values to update the weights matrix:

$$\theta_{t-1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \quad (21)$$

The authors propose default value for  $\beta_1$  is 0.9,  $\beta_2$  is 0.999 and  $10^{-8}$  for  $\epsilon$ .

## 4 EVALUATION METRICS

### 4.1 Dataset Description

The dataset consisted of about 1.8 million online comments. Each comment in the training dataset had a toxicity label (target), and the model should predict the target of the test data. For evaluation, test set example with target  $\geq 0.5$  was considered as positive class (toxic) and vice versa.

Beside the target, the training dataset also provided us with many subtypes attributes such as: severe\_toxic, obscene, threat, insult, identity\_attack, sexual\_explicit. In addition, a subset of comments was labelled with a variety of identity attributes, representing the identities that were mentioned in the comment: male, female, homosexual\_gay\_or\_lesbian, christian, jewish, muslim, black, white, psychiatric\_or\_mental\_illness.

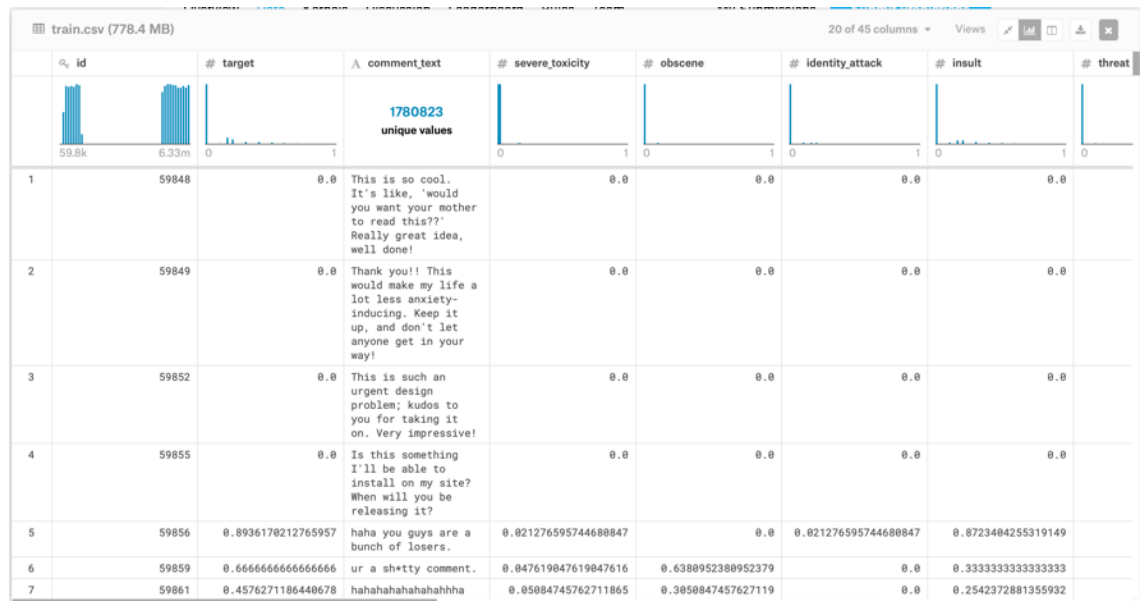


Figure 17: Description of the train data

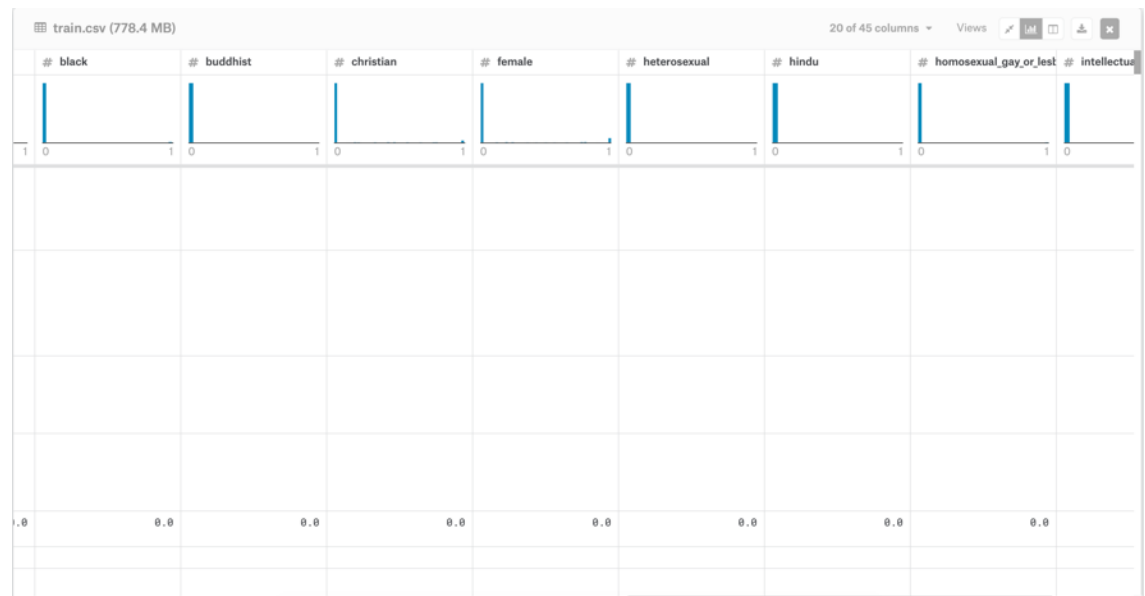


Figure 18: Identity attributes of train data

One drawback was that the data set was skewed, so that there were way more positive comments than negative ones. Therefore, it was easier to predict inputs as positive.

## 4.2 Evaluation

### Generalized Mean of Bias AUCs

- To combine the per-identity Bias AUCs into one overall measure, we calculate their generalized mean as defined below:

$$M_p(m_s) = \left( \frac{1}{N} \sum_{s=1}^N m_s^p \right)^{\frac{1}{p}} \quad (22)$$

- where:
  - $M_p$  = the  $p^{\text{th}}$  power-mean function

- $m_s$  = the bias metric  $m$  calculated for subgroup  $s$
- $N$  = number of identity subgroups
- For this competition, we use a  $p$  value of  $-5$  to encourage competitors to improve the model for the identity subgroups with the lowest model performance.

#### Final metric

- We combine the overall AUC with the generalized mean of the Bias AUCs to calculate the final model score:

$$score = w_0 AUC_{overall} + \sum_{a=1}^A w_a M_p(m_{s,a}) \quad (23)$$

- where:
  - $A$  = number of sub metrics
  - $m_{s,a}$  = bias metric for identity subgroup  $s$  using sub metric  $a$
  - $w_a$  = a weighting for the relative importance of each sub metric; all four  $w$  values set to 0.25

Having a set of comments on social networks, we need to build a model to classify the toxicity of the comments. However, the evaluation metrics is not just classified as positive or negative but the ROC-AUC score (Receiver Operating Curve – Area Under the Curve).

## 5 RESULTS AND DISCUSSION

Table 2: Accuracies of different models on public test set

Models	Results
2 GRUs with char + word embeddings layers (128 hidden states)	0.92264
1 LSTM with 2 word embeddings layers (128 hidden states)	0.92281
1 LSTM with 2 word embeddings layers (256 hidden states)	0.92308
2 LSTMs with char + word embeddings layers (128 hidden states)	0.92648
2 GRUs with 2 word embeddings layers (128 hidden states)	0.92892
2 LSTMs with 2 word embeddings layers (128 hidden states)	0.92938
Fine-tuned 2 LSTMs with word embedding and char embedding (128 hidden states)	0.92977



1314	Quan Do			0.92977	14	5d
------	---------	---	---	---------	----	----

Figure 19: Position on Kaggle's public leaderboard

The result of the model can be considered as successful as the author meets his objective for the project and broader knowledge on relevant

subject. Even though, on the way to get the result, I have met some problems, but in the end, I have successfully encountered it.

To conclude, we can observe that the model works better with two word embeddings than char embedding combine with word embedding, except for the fined tuned model. As a result, we can understand the importance of optimization method in Deep Learning and Machine Learning.

Due to the limitations of the hardware and the competition, the author could not test all the possibilities of the problems. XGBoost (Chen et al., 2016), Deeper Neural Network, train a new characters embedding, words embedding are proposed to make improvement on the model for further practical application.

## 6 CONCLUSIONS

In this thesis, we have focused on the defining concepts such as of Machine Learning, Deep Learning and their applications into real-world problems, especially Natural Language Processing. In addition, we successfully implemented a Deep Neural model to extract the semantics of sentences and output a score for inputs. Through theoretical and empirical work, a deeper understanding about Machine Learning in general and Deep Learning for NLP in specific: was gained Techniques involving data processing, simple optimization methods and classification methods became familiar to the author as well.

Further work to improve model performance was also discussed in the thesis such as using XGBoost, implement a new word or char embedding instead of using a pre-trained model, Deeper and Bigger Neural Networks, using Support Vector Machine (SVM) or Random Forest, which are both know for theirs robust characteristics and the fact that they work well for skewed dataset).



## REFERENCES

- Bengio, Y., Ducharme, R., Vincent, P. (2001). *A Neural Probabilistic Language Model*. Retrieved from <https://papers.nips.cc/paper/1839-a-neural-probabilistic-language-model.pdf>
- Chen, T., Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Retrieved 11 May from <https://dl.acm.org/citation.cfm?id=2939785>
- Cho, K., Bahdanau, D., Bougares, F., Shwenk, H., Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. Retrieved from <https://arxiv.org/pdf/1406.1078v3.pdf>
- Do, H. [1] (2017). *LSTM là gì?*. Blog publication 20 October 2017. Retrieved 11 May 2019 from <https://dominhhai.github.io/vi/2017/10/what-is-lstm/>
- Do, H. [2] (2017). *[RNN] RNN là gì?*. Blog publication 19 October 2017. Retrieved 11 May 2019 from <https://dominhhai.github.io/vi/2017/10/what-is-rnn/>
- Kingma D., P., Lei Ba, J. (2017). *ADAM: A method for Stochastic Optimization*. Published as a conference paper at International Conference on Learning Representation (ICLR) 2015. Retrieved from <https://arxiv.org/pdf/1412.6980.pdf>
- Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. Retrieved from <https://arxiv.org/pdf/1301.3781.pdf%5D>
- Minh-Tien, N. (2019). Online Deep Learning course, Moodle. AI Academy Vietnam. Retrieved 13 May 2019 from [http://103.216.114.222/moodle/pluginfile.php/1103/mod\\_resource/content/2/RNN.pdf](http://103.216.114.222/moodle/pluginfile.php/1103/mod_resource/content/2/RNN.pdf)
- Mital, V. (2017). *Top 15 Deep Learning applications that will rule the world in 2018 and beyond*. Retrieved from <https://medium.com/@vratulmittal/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01>
- Naïve Bayes Classifier (2019). Retrieved 16 May 2019 from [https://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

Logistic Regression (2019). Retrieved 16 May 2019 from [https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)

Sepp, H., Jürgen, S. (2006). *Long Short-Term Memory*. The MIT press.  
Skymind.ai (n.d.). Retrieved 13 May 2019 from <https://skymind.ai/wiki/neural-network#concept>

Skymind.ai (n.d.). A Beginner's Guide to Natural Language Processing (NLP). Retrieved 13 May 2019 from <https://skymind.ai/wiki/natural-language-processing-nlp>

Solomon, F. (n.d.). Going deeper with recurrent networks: Sequence to Bag of Words Model. Retrieved 11 May from <https://www.kdnuggets.com/2017/08/deeper-recurrent-networks-sequence-bag-words-model.html>

Tom, M. (1998). *Machine Learning*. 1<sup>st</sup> edition. McGraw-Hill Science/Engineering/Math publisher.

Warren, S. (1994). *Neural Networks and Statistical Models*. Proceedings of the Nineteenth Annual SAS Users Group International Conference.

## Appendix 1 MODEL IMPLEMENTATION

```
import numpy as np
import pandas as pd
import re
from keras.models import Model
from keras.layers import Input, Dense, Embedding, SpatialDropout1D, Dropout, add,
concatenate
from keras.layers import CuDNNGRU, CuDNNLSTM, GlobalMaxPooling1D,
GlobalAveragePooling1D
from keras.layers.wrappers import Bidirectional
from keras.preprocessing import text, sequence
from keras.callbacks import LearningRateScheduler
from keras import optimizers

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the
input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.

EMBEDDING_FILES = [
    # '../input/fasttext-crawl-300d-2m/crawl-300d-2M.vec',
    '../input/glove6b300dtxt/glove.6B.300d.txt',
    '../input/glove840b300dchar/glove.840B.300d-char.txt'
]

NUM_MODELS = 2
BATCH_SIZE = 512
GRN_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * GRN_UNITS
EPOCHS = 4
MAX_LEN = 220
```

```
DROP_OUT = 0.3

def get_coefs(word, *arr):
    return word, np.asarray(arr, dtype='float32')

def load_embeddings(path):
    with open(path) as f:
        return dict(get_coefs(*line.strip().split(' ')) for line in f)

def build_matrix(word_index, path):
    embedding_index = load_embeddings(path)
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in word_index.items():
        try:
            embedding_matrix[i] = embedding_index[word]
        except (KeyError, UnicodeDecodeError, UnicodeEncodeError) as e:
            pass
    return embedding_matrix

def build_model(embedding_matrix, num_aux_targets):
    input_s = Input(shape=(MAX_LEN,))
    grn_s = Embedding(*embedding_matrix.shape, weights=[embedding_matrix],
trainable=False)(input_s)
    grn_s = SpatialDropout1D(0.3)(grn_s)
    grn_s = Bidirectional(CuDNNGRU(GRN_UNITS, return_sequences=True))(grn_s)
    grn_s = Bidirectional(CuDNNGRU(GRN_UNITS, return_sequences=True))(grn_s)

    hidden = concatenate([
        GlobalMaxPooling1D()(grn_s),
        GlobalAveragePooling1D()(grn_s),
    ])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
```

```

result = Dense(1, activation='sigmoid')(hidden)
aux_result = Dense(num_aux_targets, activation='sigmoid')(hidden)

model = Model(inputs=input_s, outputs=[result, aux_result])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

def preprocess(text):
    text = re.sub(r"<[^>>", "", text)
    emoticons = re.findall(r"(?:|;|=)(?:-)?(?:\)|\(|D|P)", text)
    text = re.sub(r"[\W]+", " ", text.lower()) + " ".join(emoticons).replace('-', '')
    text = re.sub(r"\n", ' ', text)
    return text.lower()

train = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/train.csv')
test = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/test.csv')

train = train.fillna(0.0)
x_train = train['comment_text'].apply(preprocess)
y_train = np.where(train['target'] >= 0.5, 1, 0)
y_aux_train = train[['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threat']]
x_test = test['comment_text'].apply(preprocess)

tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(list(x_train) + list(x_test))

x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_test = sequence.pad_sequences(x_test, maxlen=MAX_LEN)

```

```
embedding_matrix = np.concatenate(
    [build_matrix(tokenizer.word_index, f) for f in EMBEDDING_FILES], axis=-1)

model = build_model(embedding_matrix, y_aux_train.shape[-1])
model.fit(
    x_train,
    [y_train, y_aux_train],
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=2,
    callbacks=[
        LearningRateScheduler(lambda epoch: 1e-3 * (0.6 ** 1))
    ]
)

predictions = model.predict(X_test, batch_size=2048)[0].flatten()

submission = pd.DataFrame.from_dict({
    'id': test['id'],
    'prediction': predictions
})

submission.to_csv('submission.csv', index=False)
```

Appendix 2  
MODEL LOG

Using TensorFlow backend.

['glove6b300dtx', 'fasttext-crawl-300d-2m', 'jigsaw-unintended-bias-in-toxicity-classification']

Epoch 1/4

- 774s - loss: 0.2365 - dense\_3\_loss: 0.1289 - dense\_4\_loss: 0.1076 - dense\_3\_acc: 0.9491 - dense\_4\_acc: 0.8547

Epoch 2/4

- 775s - loss: 0.2154 - dense\_3\_loss: 0.1128 - dense\_4\_loss: 0.1026 - dense\_3\_acc: 0.9541 - dense\_4\_acc: 0.8550

Epoch 3/4

- 777s - loss: 0.2077 - dense\_7\_loss: 0.1061 - dense\_8\_loss: 0.1015 - dense\_7\_acc: 0.9563 - dense\_8\_acc: 0.8550

Epoch 4/4

- 777s - loss: 0.2011 - dense\_7\_loss: 0.1002 - dense\_8\_loss: 0.1009 - dense\_7\_acc: 0.9582 - dense\_8\_acc: 0.8550