Detecting offensive language using transfer learning

Alissa de Bruijn

September 2019

VU VRIJE UNIVERSITEIT AMSTERDAM

mobiquity

Master Thesis

Business Analytics

Detecting offensive language using transfer learning

Author:
Alissa de Bruijn

September 2019

Supervisors:
Dr. Vesa Muhonen
MSc. Richard Price

Mobiquity Inc.
Tommaso Albinonistraat 9
1083 HM Amsterdam

Supervisors:
Prof. dr. Wan Fokkink
Dr. Peter Bloem

Vrije Universiteit Amsterdam
Faculty of Sciences
Business Analytics
De Boelelaan 1081a
1081HV Amsterdam

Preface

This thesis is written as the graduation project for the Master Business Analytics at the Vrije Universiteit Amsterdam. Business Analytics is a multidisciplinary program aimed at improving and optimising business processes with mathematics, computer science, and economics.

This research project was part of a six months internship at Mobiquity. During my internship, I was part of the Data Science team. The project builds on the knowledge learned during previously taken courses. I am particularly interested in the field of Natural Language Processing. This research focuses on the detection of offensive language in social media.

First of all, I would like to thank Vesa Muhonen and Richard Price, my supervisors of Mobiquity, for their insights and guidance during the internship. Furthermore, I would like to thank Wan Fokkink, my supervisor at the VU, for his feedback, and Peter Bloem for being the second reader of my thesis.

Detecting offensive language using transfer learning

Abstract

Since 2004, the use of social media has been growing exponentially. Daily, there are 3.2 billion people who use social media, which is equal to 42% of the total population. With the rise of social media, there is also a growing amount of offensive content online. Anyone can use social media and there is no lack of people who spread hate speech to attack minorities or individual people on platforms like Twitter.

Facebook and Twitter have been criticised that they take insufficient action to remove offensive content on their platform. They are facing challenges as they try to remove content based on their policies. They try to improve it, Twitter announced last month that they will hide (but not remove) harmful tweets from public figures. While they are trying to fix it, it is still a difficult problem due to the variety of language used in social media.

This research focuses on detecting offensive language in social media using transfer learning and comparing different methods. Many NLP tasks share common knowledge about language, syntax and semantics. We will use transfer learning because it has shown state-of-the-art results for many supervised NLP tasks. We will use the Offensive Language Identification Dataset (OLID), which consists of 14K tweets to detect offensive content in social media.

Word can be represented by word embeddings. These word embeddings will be used as input for the models. We will use context-free embeddings (Word2Vec, GloVe, fastText) and contextual embeddings (BERT). Our work will compare different models. As a baseline, we will use a SVM. We will compare it to models which are often used in the context of language, LSTM, and CNN. CNNs are good at finding key phrases and LSTM networks work well for tasks like language modelling. Lastly, we will use a state-of-the-art language model BERT. BERT is a model which is trained on a large amount of text and therefore it can capture linguistic, syntactic and semantic features.

Analysis of the results shows that BERT and CNN are the best performing models. CNNs work well because tweets are short and they can pick up certain key phrases. BERT is a very complicated model but can still be used, also for tweets. Based on these results, we conclude that transfer learning, and especially, large pretrained models, can be very useful to detect offensive language.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations

BERT	**B**idirectional **E**ncoder **R**epresentations **f**rom **T**ransformers
BOW	**B**ag-**O**f-**W**ords
CBOW	**C**ontinuous **B**ag-**O**f-**W**ords
CNN	**C**onvolutional **N**eural **N**etwork
CV	**C**omputer **V**ision
FFNN	**F**eed-**F**orward **N**eural **N**etwork
GRU	**G**ated **R**ecurrent **U**nits
LM	**L**anguage **M**odel
LSTM	**L**ong **S**hort-**T**erm **M**emory
MLM	**M**asked **L**anguage **M**odel
MLP	**M**ulti**l**ayer **P**erceptron
ML	**M**achine **L**earning
NLP	**N**atural **L**anguage **P**rocessing
NN	**N**eural **N**etwork
OOV	**O**ut-**O**f-**V**ocabulary
RNN	**R**ecurrent **N**eural **N**etwork
SOTA	**S**tate-**o**f-**t**he-**A**rt
SVM	**S**upport **V**ector **M**achine

TFIDF **T**erm **F**requency–**I**nverse **D**ocument **F**requency

1 Introduction

Since 2004, social media has been growing exponentially (Maryam Mooshin, 2019). People can share their opinions and express their feelings. For example, Twitter users generate 500 million tweets per day and in 2019 they had a 14% year-over-year growth of daily usage (Twitter, 2019). With this growth, there is also an increase in offensive content. Recent research by Pew Research Center shows that a majority of the US adults say that social media companies have a responsibility to remove this offensive content. However, only one third are confident that these companies can determine what offensive content should be removed. Besides that, a majority says that people do not agree on what is seen as offensive language by other people (racist or sexist language) (Center, 2019).

These worries are not groundless, social media companies are facing challenges as they try to remove content based on their policies. Technology companies are under a lot of pressure to better monitor and police their platforms for hate speech, violence, abuse, and offensive language. The huge amount of user-generated content makes it impossible to monitor all these posts manually. To combat hate speech, Germany introduced a hate speech law. Since then German authorities say that they have issued Facebook with a €2 million fine (Wong, 2019). Also, France adopted a bill to give social media platforms 24 hours to remove hateful content (Kelly, 2019a).

For example, Facebook recently launched a new feature which asks users who try to post offensive comments to reconsider their choices before they place a comment. Instagram encourages people to undo their hurtful comments or share something less hurtful. They also launched a new feature which uses AI to notify people when their comment may be considered harmful or offensive before it is posted (Thompson, 2017). Twitter will hide but not remove harmful tweets from public figures. Users of the platform will see a notification if a tweet violates the platform rules (Kelly, 2019b).

Automatic detection of offensive content in forums, blogs, and social media can be a useful support for moderators of public platforms as well as users who could receive warnings or could filter unwanted content. Language usage in social media is exceptionally varied, it could contain incorrect spellings, grammar mistakes, acronyms, slang, or emojis. This makes understanding the context challenging.

In general, words can have multiple meanings. Meanings are dependent on the context which they are in for humans this is a lot easier to distinguish than for computers. Recent developments in NLP have made the once-insurmountable task of semantic and contextual inference more tractable. For example, in the sentence *'He ate an apple in front of his Apple computer.'*, the word *'apple'* can have two meanings. The first one refers to the brand of a laptop whereas the second one refers to a type of fruit. Earlier models can not distinguish this, contrary to newer models which understand the context in which the words are in.

We will use a Twitter dataset for the detection of offensive language. As a baseline, we will use a Support Vector Machine (SVM) model (Cortes & Vapnik, 1995). We will compare it to models which are often used in the context of language: a Long Term-Short Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and a Convolutional Neural Network (CNN) (Kim, 2014). At last, we will use a state-of-the-art model Bidirectional Encoder from Transformers (BERT). BERT is a language model trained on a large text corpus and therefore it can capture linguistic, syntactic, and semantic features.

Based on the objective of this research, the main goal of this research is as follows:

How can we use transfer learning to detect offensive content in social media?

We perform this research at Mobiquity, a consultancy company, and as such, they are doing a wide variety of tasks to solve issues for their clients. Mobiquity sees this internship as a knowledge-sharing task, bringing in new information to keep their service up-to-date. A better understanding of language models could help to build a chatbot, or a topic model, or help with other Natural Language Processing related tasks.

This paper is organized as follows. Section 2 describes the related work. We will explain in Section 3 how transfer learning can be used in NLP. Section 4 outlines the methods we will use. Next, Section 5 describes the dataset alongside with data analysis. Section 6 gives an overview of the experimental setup to find the optimal hyperparameters. The results and evaluation of the methods are presented in Section 7. The conclusion and discussion of possible directions for future work are given in Section 8.

2 Related Work

There is a lot of research done on offensive language. Most of this research is focused on a specific type of offensive content, such as aggression, abusive language (Nobata, Tetreault, Thomas, Mehdad, & Chang, 2016), bullying (Xu, Jun, Zhu, & Bellmore, 2012), hate speech (Wang, Chen, Thirunarayan, & Sheth, 2014), (Djuric et al., 2015), (Davidson, Warmsley, Macy, & Weber, 2017), (Zampieri et al., 2019) and offensive language. There are different methods used for detecting different types of offensive content. People have used machine learning models and different features such as character n-grams, word-n-grams with TFIDF, and different word embeddings like fastText and Glove.

However, there is a lack of consensus about the annotation of the different types of offensive content. Something may be considered as hate speech (Waseem, 2016) while others only define this as derogatory or offensive (Nobata et al., 2016). Therefore in (Waseem, Davidson, Warmsley, & Weber, 2017) they propose a typology which captures similarities and difference between hate speech, cyberbullying and online abuse. They present a two-fold typology whether (i) the abuse is directed at a specific target and (ii) the degree to which it is explicit.

Different tasks were published, TRAC (Trolling, Aggression and Cyberbullying) (Kumar, Ojha, Malmasi, & Zampieri, 2018) published a shared task on Aggression Identification to classify Facebook Posts and Comments in different types of aggression. The best performing team used a LSTM with a RNN and CNN as features. The GermEval (Wiegand, Siegel, & Ruppenhofer, 2018) shared task to identify offensive language in German tweets. The goal is to detect offensive versus non-offensive tweets. The second task is to distinguish if the offensive tweet is a profanity, insult or abuse. The classifiers used for this task consist of non-neural types of which SVM is the most common type; for the neural classifiers, CNN, LSTM, and GRU were used.

(Xu et al., 2012) did a study on detecting bullying in tweets. They used sentiment and topic models to identify bullying in tweets. (Dadvar, Trieschnigg, Ordelman, & de Jong, 2013) trained a cyberbullying classifier using content-based features, cyberbullying features and user-based features. Their work showed that context and user-based features cyberbullying improves the cyberbullying detection accuracy.

Hate speech is also a form of offensive language and is used to express hatred towards a targeted group or is intended to be derogatory, to humiliate or to insult the members of the group. It is difficult to distinguish hate speech from other offensive language because the differences are often based upon subtle linguistic distinctions. (Davidson et al., 2017) researched the detection of hate speech. For the classification, they used syntactic features (unigram, bigram trigram, Part-of-Speech) and non-linguistic features (gender, ethnicity). They showed that in their case supervised approaches (Logistic Regression and SVM) performed better than other models (naive Bayes, decision trees, random forests).

3 Background

Transfer learning is the transfer of knowledge from one task to another related task. It is related to how humans learn as we do not learn everything from scratch, but instead, we transfer knowledge from what we have learnt in the past for solving new tasks. Transfer learning has a huge influence on computer vision (CV). When developing neural network models it can save time and resources. It is common to use CV models as the Oxford VGG Model (Simonyan & Zisserman, 2015), Google Inception Model (Szegedy, Liu, Jia, Sermanet, & Reed, 2015) and Microsoft ResNet Model (?, ?), which have been trained on a large image dataset instead of training models from scratch. For example, ImageNet (Russakovsky, Deng, Su, Krause, & Satheesh, 2015) contains 1.2 million images with 1000 categories. This can be beneficial because these models can learn low-features, such as edges, shapes and corners from images, which can be shared across tasks.

For text, this is challenging, as the text is unstructured, noisy and very diverse. While deep learning models also showed SOTA results for different NLP tasks, these models are usually trained from scratch, which requires large datasets. Gathering and labelling data is expensive, often lacking high-quality annotated examples to train a model. Transfer learning could solve this problem, by extracting useful knowledge from different but related domains. In NLP, this is useful because many tasks share common knowledge about language, syntax and semantics.

Recent developments in this field are said to be the ImageNet moment for NLP (Sebastian Ruder, 2018). Empirically transfer learning showed SOTA for many supervised NLP tasks (e.g. classification, information extraction, Q&A, etc) (Martin, 2019). Because models do not have to learn from scratch, in general, it can reach a better performance with fewer data and computation time in comparison to models which do not use transfer learning.

For the classification task, there are two methods which can be used: traditional machine learning and transfer learning. This section will describe the transfer learning trends in NLP. Recent developments in the last year were in the field of language models, ELMo, ULMFit, OpenAI Transformer, and BERT, this section will describe the shift in NLP and will explain what transfer learning is and how it is used.

3.1 Transfer Learning

Machine learning tasks can be learned from scratch, which takes a long time and in general, you need to have a lot of data. Instead of learning a model from scratch, transfer learning aims to extract and use the knowledge from one or more source tasks for a target task. Figure 1 shows the difference between traditional machine learning techniques and transfer learning techniques. Machine learning techniques train the model from scratch on one task in a certain domain; this information cannot be used

to train another task. With transfer learning, the aim is to transfer the knowledge learned from source tasks and use this for a target task. The target task often has less training data than the source tasks. (Pan & Yang, 2009) (Pitsilis, Ramampiaro, & Langseth, 2018)

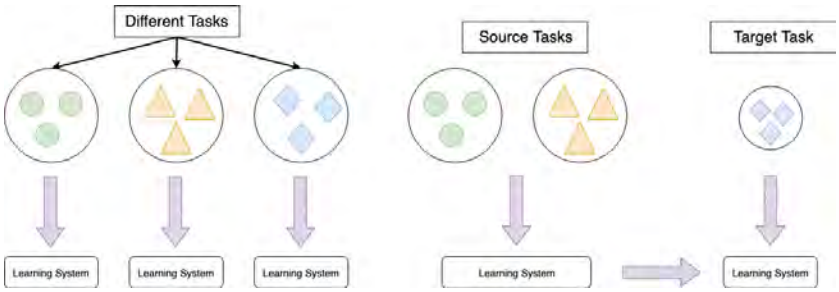


Figure 1: Traditional Machine Learning versus Transfer Learning

Notations:
In their paper, *A Survey on Transfer Learning*, (Pan & Yang, 2009) present a framework to understand transfer learning. They use the concepts of a domain and a task in their framework. The framework is defined as follows:

A domain $\mathcal{D}$, is defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$ where $\mathcal{X}$ is a feature space and $P(X)$ a marginal probability distribution, where X is a sample data point $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$. A task, T, can be defined $\mathcal{T} = \{\mathcal{Y}, P(Y)\}$, a label space $\mathcal{Y}$ and a conditional probability distribution $P(Y|X)$. The function is learned from the training data, which consists of pairs $\{x_i, y_i\}$, $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$.

With these definitions and representations, we can define transfer learning as follows: Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and target task $\mathcal{T}_T$, transfer learning aims to improve the learning of the target conditional probability distribution $P_T(Y_T|X_T)$ in $\mathcal{D}_T$ using the knowledge gained from $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.

Scenarios
Given the definition of the source and target domains $\mathcal{D}_S$ and $\mathcal{D}_T$ where $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and source and target tasks $\mathcal{T}_S$ and $\mathcal{T}_T$ where $\mathcal{T} = \{\mathcal{Y}, P(Y), P(Y|X)\}$. There are different transfer learning scenarios based on the source and target conditions.

To give some intuition about the definitions we will use binary document classification as a running-example. For document classification, $\mathcal{X}$ is the space of all term vectors, x_i is the i-th term vector corresponding some document and $\mathcal{X}$ is a particular learning sample. The set of labels, $\mathcal{Y}$, can be defined as $\{True, False\}$ where y_i is either *True* or *False*.

A domain is a pair $\mathcal{D} = \{\mathcal{X}, P(X)\}$, therefore either $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$ holds to meet the condition $\mathcal{D}_S \neq \mathcal{D}_T$. Which means that either the feature space of the source and target domain are different or the marginal probability distributions between the domains are different.

- $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the sources and target domain are different, e.g. the documents are described in different languages.
- $P(X_S) \neq P(X_T)$. The marginal probability distributions of the source and target tasks are different, e.g. the documents have different topics.

This also applies to a task. A task is defined as a pair $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. The condition $\mathcal{T}_S \neq \mathcal{T}_T$ either holds when $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$ where $Y_{S_i} \in \mathcal{Y}_S$ and $Y_{T_i} \in \mathcal{Y}_T$.

- $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces between the source and target task are different, e.g. the source domain and target domain have different labels.
- $P(Y_S|X_S) \neq P(Y_T|X_T)$ where $Y_{S_i} \in \mathcal{Y}_S$ and $Y_{T_i} \in \mathcal{Y}_T$. The conditional probability distributions between the tasks are different, e.g. the source and target documents are very unbalanced.

It is a traditional machine learning model when the domains are the same and the learning tasks are the same, $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. (Pan & Yang, 2009)

Transfer learning in NLP can be used for many tasks. The pretraining tasks and datasets are often unlabelled data. Target tasks are mostly supervised tasks and are ranging from sentence or document classification (sentiment), sentence pair classification, word level, structured prediction, and generation. For example, we could transfer knowledge from Wikipedia documents to Twitter text, from English documents to Chinese documents in a search engine, from webpages to images, etc.

Several pre-trained models used in transfer learning are based on NN with a multilayer architecture. We can use these models as feature-extractors or we can fine-tune the pre-trained model. The feature-based approach extracts fixed features from the pretrained model. This is mostly used when the data of the pre-trained model is similar to the data of the target task.

Secondly, the architecture of the pre-trained network can be used and then fine-tuned. The weights of the layers from the pre-trained model are discarded and the entire model is retrained on the target data. This approach is used when there is a large dataset for the given task but this data is not similar to the data were the pre-trained model was trained on. Earlier features of a model might be able to detect more generic features, for example in images it can detect edges and colour blobs. It is also possible to freeze a certain layer or certain layers of a pre-trained model and then train the other layers. This approach is used when the dataset is small and the datasets are not very similar.

3.2 Language Model

A Language Model (LM) is important in NLP. A LM can generate a probability distribution over sequences of words or provide word representations. This is important because, to do this, it needs a lot of world knowledge and understanding of grammar, semantics, and other elements of natural language. It can be used to predict the next word given the given context. (Jing & Xu, 2019) (Ruder, 2019) (Kamath, Liu, & Whitaker, 2019)

A traditional LM assigns probabilities to a sequence s of N words:

$$\begin{aligned} P(s) &= P\left(w_1 w_2 \cdots w_N\right) \\ &= P\left(w_1\right) P\left(w_2 | w_1\right) \cdots P\left(w_N | w_1 w_2 \cdots w_{N-1}\right), \end{aligned} \tag{1}$$

where w_i denotes the i-th word in sequence s.

The probability distribution over words or character sequences can be defined as the product of the conditional probability of the next word given the previous words or characters in the sequence (Jing & Xu, 2019) (Kamath et al., 2019).

The most used LM is the n-gram model. The probability of the sequence $\{w_1, .., w_t\}$ is given by the product of the conditional probabilities. The current state depends on the previous k states.

$$P\left(w_t | w_1 \cdots w_{t-1}\right) \approx P\left(w_t | w_{t-k} \cdots w_{t-1}\right) \tag{2}$$

The problem with LMs is the curse of dimensionality, a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. The traditional approach based on n-grams in general only takes the context of one or two consecutive words into account. It does not take into account the similarity between words. To overcome the problem of the curse of dimensionality, Neural Networks (NN), like the Feedforward Neural Network (FFNN), and RNN have been introduced for language modelling. They can also handle sequences which were not present in the training data. (Jing & Xu, 2019)

(Bengio, Ducharme, Vincent, & Jauvin, 2003) presented the first FFNN Language Model. It learns a distributed representation for each word, a word vector also called embedding. After FFNN, the RNN Language Model (RNNLM) was introduced by (Mikolov, Karafiát, Burget, Černock, & Khudanpur, 2010) and Long Short-Term Memory (LSTM) (Sundermeyer, Schlüter, & Ney, 2012) Language Model was introduced, see Section 4.

3.3 Word Embeddings

When using machine learning models words need to be processed in a form of numeric representation for the models to use them in the calculation. Word embeddings are used to encode and represent an entity (document, sentence, word, graph), a fixed-length vector. Word embeddings are typically used when applying NNs for NLP tasks. Each word w_i in the input sequence is mapped to a vector $\mathbf{x}_i$, which is called a word embedding of each word. The word embeddings are put in a word embedding matrix $\mathbf{X} \in \mathbb{R}^{|V| \times d}$, where V is the size of the vocabulary and d the dimension of the word embedding. These are then used as features for a NN.

The simplest method of creating a word embedding is one-hot encoding or the Bag of Words (BoW) approach. With one-hot encoding, each element in the vector corresponds to a word in the vocabulary of the corpus. If the word occurs in a given document, the word corresponding to an index is marked as 1, else it is marked as 0. As every element in the vector is associated with a word in the vocabulary, it will lead to huge sparse word vectors. These word embeddings also do not capture the semantic relationship between words. Instead of saying a word exists or not exists, BoW gives the counts of each word in each document. Words are evenly weighted independently of how frequent or in which context they occur. It only takes into consideration the frequency of words in a document. However, some words might be more relevant than others in most NLP tasks. To reflect how important a word is to a document in a collection of documents, Term Frequency-Inverse Document Frequency (TFIDF) is used. TFIDF gives weight to a word based on the context it occurs. It increases proportionally to the number of times a word appears in a document adjusting for the fact that some words appear more frequently than others.

$$w_{x,y} = tf_{x,y} \times log(\frac{N}{df_x}), \tag{3}$$

where $tf_{x,y}$ = frequency of x in y, df_x = number of documents containing x and N = total number of documents . In other words, high frequency words may not provide much information gain while rare words can contribute more weight to the model.

Another way to present words is with a fixed vector length which can capture context and semantics, such as Word2Vec, Glove, fastText and ELMo. The three most common models are Word2Vec, Glove and fastText. These models are based on the fact that words which occur and are used in the same context tend to be semantically similar to another and have a similar meaning. Those pre-trained word embedding techniques are used to initialise the first layer of a NN.

- **Word2Vec** Word2Vec uses a local context window, consisting of words in a defined window of neighbouring words. It uses two different methods to learn word embeddings: Continuous Bag-of-Words (CBOW) and Continuous Skip-Gram Model. CBOW learns an embedding by predicting the current word given the context, the surrounding words. The Skip-Gram Model learns an embedding by predicting the surrounding words based on the context, the current word. (Mikolov, Chen, Corrado, & Dean, 2013) (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013)
- **GloVe** GloVe is an extension of Word2Vec, it uses global corpus statistics for word representations. GloVe learns word embeddings by dimensionality reduction of the co-occurrence count matrix. Instead of learning raw co-occurrence probabilities, it learns ratios of co-occurrence probabilities to distinguish relevant words from irrelevant words. (Pennington, Socher, & Manning, 2014) (Huang, Socher, Manning, & Ng, 2012)
- **fastText** fastText is an extension of the Continuous Skip-Gram Model of Word2Vec. It learns word representations using subword level embeddings. The previous methods ignore the morphology of the words and cannot handle out-of-vocabulary words. This gives a disadvantage for languages with large vocabularies and many rare words. There are some languages with a lot of different inflected forms without morphology. For these words, it is difficult to learn a good word representation. fastText improves the vector representation and takes into account the morphology using subword units (character level information). The words are split into a bag of n-gram characters. Each word is represented as a bag of characters n-grams, plus a special boundary symbols <and> at the beginning and end of words, plus the word w itself in the set of its n-grams. A word is then represented by taking the sum of its character n-grams. (Joulin, Grave, Bojanowski, & Mikolov, 2016) (Bojanowski, Grave, Joulin, & Mikolov, 2017)

Although these word embeddings capture some meaning, there are often words which have different meanings depending on the context. These word embeddings can only capture one meaning. A lot of data is required to disambiguate words and learn words which were not seen before.

4 Methodology

This section describes the different methods we use to detect offensive language in social media. The baseline is a SVM model, see Section 4.1. The input features are a BOW model with TFIDF weighting, see Section 3.3. We use a MLP as a basic neural network, which we will describe in Section 4.2.1. We will expand this with a CNN in Section 4.2.2 and a sequential network, LSTM, a type of RNN network, see Section 4.2.3. CNNs are good at finding key phrases and RNNs work well for tasks like language modelling. The input features for these models are context-free word embeddings. Lastly, in Section 4.5 we will compare this to a state-of-the-art model, BERT, which uses contextual word embeddings.

4.1 Support Vector Machine

SVM (Cortes & Vapnik, 1995) is a model often used for classification. Every data item is a point in an n-dimensional space, where n is the number of features. Classification is performed by finding the hyper-planes that segregates the data in a given number of classes using maximum margin separation.

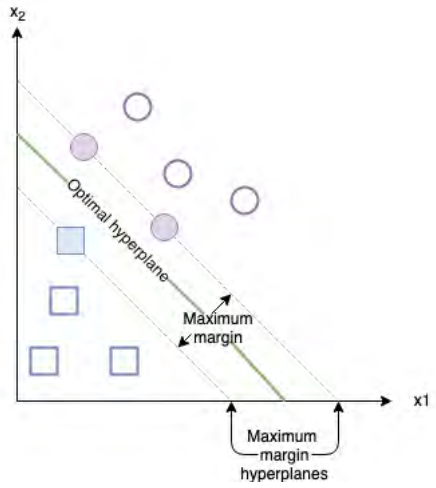


Figure 2: Support Vector Machine

The data points can be defined as pairs:

$$(y_1, x_1), \ldots, (y_\ell, x_\ell), \quad y_i \in \{-1, 1\} \tag{4}$$

This is linearly separable if there exists a vector $\mathbf{w}$ and scalar b such that the following inequalities hold:

$$\begin{aligned} \mathbf{w} \cdot x_i + b &\geq 1 \quad \text{if} \quad y_i = +1 \\ \mathbf{w} \cdot x_i + b &\leq -1 \quad \text{if} \quad y_i = -1 \end{aligned} \tag{5}$$

These equations can be written as:

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) - 1 \geq 0 \quad \forall \quad i \tag{6}$$

A hyperplane can be defined by:

$$f(x) = x^T \beta + \beta_0, \tag{7}$$

where β is a unit vector $\|\beta\| = 1$, with bias β_0.

The algorithm tries to find the margin M, the hyperplane that gives the largest minimum distance of the data points of the classes. The optimal separating hyperplane maximises the margin of the data. The data points that separate the hyperplane and lie on the margin are known as support vectors (Kamath et al., 2019). Figure 2 shows the optimal separating hyperplane when two classes are linearly separable (Fletcher, 2008) (Cortes & Vapnik, 1995).

Maximising the margin, the largest distance between the hyperplane and the data instances, is equivalent to minimising a function $\mathrm{L}(\beta)$ subject to given constraints.

$$\min_{\beta,\beta_0} \tfrac{1}{2}\|\beta\|^2, \text{ subject to } y_i \left(x_i^T \beta + \beta_0\right) \geq 1, \quad i = 1, \ldots, N \tag{8}$$

Ideally, there is a hyperplane that separates classes in a way that they are non-overlapping. This might not be possible, or it will result in an undesirable number of different classes. In this case, the SVM tries to find a hyperplane that maximises the margin and minimises the misclassifications. A slack variable is introduced which allow data points to fall off the margin but penalises them when that happens. (Cortes & Vapnik, 1995)

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i, \text{ subject to } \begin{cases} \mathrm{y}_i \left(\boldsymbol{w} \cdot x_i + b\right) \geq 1 - \xi_i, & \forall \quad i \\ \xi_i \geq 0 \end{cases} \tag{9}$$

where C is the trade-off between the margin width and the misclassifications.

4.2 Neural Networks

A neural network (NN) is a network of interconnected perceptrons. A perceptron is a one-unit neural network and used a building block for more complex neural networks. Each perceptron unit has an input (x), an output (y), and a set of weights (w), a bias (b), and an activation function (f).

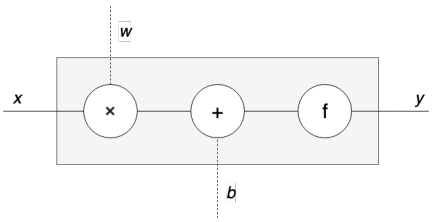


Figure 3: Computational Graph for a Perceptron

The weights and the bias are learned from the data. We can define a perceptron as follows:

$$y = f(\mathbf{w}x + \mathbf{b}) \tag{10}$$

In a general case, there are multiple inputs for a perceptron, x and w are vectors and the product of x and w is replaced with a dot product. The activation function, denoted by f, of typically a nonlinear function.

$$y = f(\mathbf{w}\mathbf{x} + \mathbf{b}) \tag{11}$$

An example of a NN is shown in Figure 4. A NN has 3 components, input layer, hidden layer(s) and an output layer. In Figure 4, the input layer is fed with input data. This is passed to the hidden layer(s). The nodes in the hidden layer(s) each compute an activation based on the input, and pass this to the output layer. The node(s) in the output layer computes the final output, based on the input from the previous hidden layer.

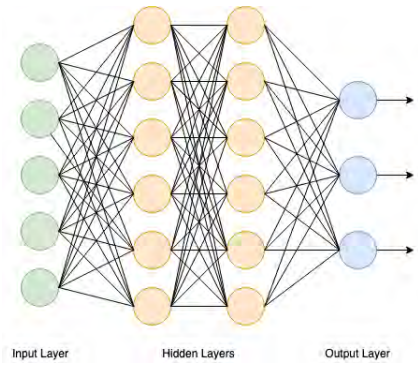


Figure 4: Neural Network

There are different types of activation functions, the most used activation functions are: sigmoid, softmax, hyperbolic tangent, and rectified linear unit.

Sigmoid or *logistic* activation function σ, maps any arbitrary range of values to a value between 0 and 1. This allows the output to be used as probabilities.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

Softmax activation function, like the softmax function, transforms the output between 0 and 1. However, the softmax function also divides each output by the sum of all the outputs, it calculates the probabilities of the event over n possible classes.

$$f\left(x_i\right) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{13}$$

Hyperbolic tangent (*tanh*) activation function outputs, maps a set of real values from $(-\infty, +\infty)$ to a value in a range from -1 to 1.

$$f(x) = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}} \tag{14}$$

Rectified linear unit (ReLU) activation function, clips the negative values to zero. It outputs values between the range from 0 to infinity.

$$f(x) = \max(0, x) \tag{15}$$

The way various layer are connected is also called the architecture of a network. The variables that define the architecture of a network are called hyperparameters. The number of hidden layers, the activation function, learning rate and batch size, are a couple of examples of hyperparameters. When training these networks the hyperparameters have to be optimised. Training the network is obtaining the parameters such as the network weights and bias. The training goal is to minimise a certain loss function. The loss function depends on the predicted output and the labels or values. The training algorithm uses back-propagation to update weights and biases. The error in the prediction is propagated backwards through the network. For the classification tasks, we will use the cross-entropy loss. Cross-entropy computes the softmax probabilities of the given classes. (Osinga, 2018) (Arumugam & Shanmugamani, 2018) (McMahan & Rao, 2019)

4.2.1 Multilayer Perceptron

MLP is a FFNN and consists of at least three layers: an input, an output and one or more hidden layers. Each node in the layer, except for the input nodes, is a neuron with a nonlinear activation function. The network is also fully connected, which means that in a layer each node is connected to the other nodes in the network with a certain weight w_{ij}.

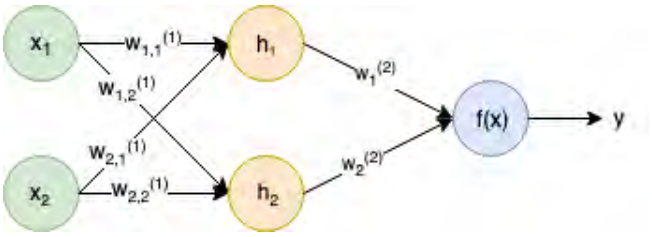


Figure 5: Illustration of a MLP Network with two neurons

We can describe the network as:

$$\begin{aligned} \mathbf{h} &= g\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}_1\right) \\ \mathbf{y} &= f\left(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}_2\right), \end{aligned} \tag{16}$$

where layer is parameterized with their weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$. With h, the hidden layer where $g(x)$ is the activation function. The output of the network is y, with $f(x)$ the output function (Deng & Liu, 2018) (Ruder, 2019).

4.2.2 Convolutional Neural Network

CNN is similar to an ordinary NN, but consists of multiple hidden layers and a filter called a convolution layer. They are good in, for example, identifying objects, and faces. CNNs were important for breakthroughs in image classification. An image can be converted into an array of pixel values. Often it is described as a $X \times Y \times Z$ array of numbers. For example, a colour image with a size of 480×480 is represented by a $480 \times 480 \times 3$ array, where 3 is the RGB value of the colour. Using a CNN to detect faces in an image, the features you could extract are features such as a nose, mouth, or pair of eyes in the image.

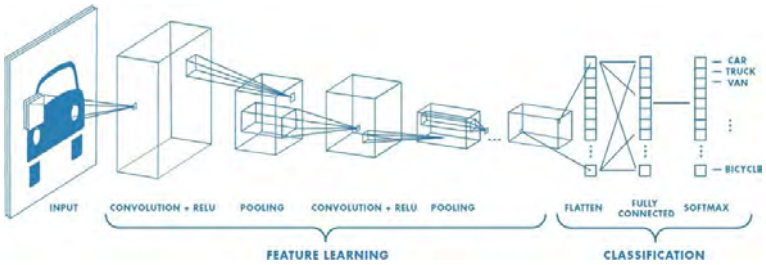


Figure 6: Architecture of a CNN for CV

The hidden layers of a CNN consist of different steps. The first step is a convolution, convolution is a sliding window function applied to a matrix. The convolution, also called filters, extracts features from the input image. A new matrix is formed by sliding a filter over the image and multiplying this with another matrix. A CNN automatically learns the values of its filters during the training process. After a convolution, a non-linear activation function is used. Pooling is used to decrease the dimensionality of the feature without losing important information. Figure 7 uses a 2×2 window and slides over the image and takes the maximum value in each region. The last layer of the CNN is a dense layer with as input the features. The fully connected layer takes the input from the layer and gives out an N-dimensional vector where N is the number of classes. In Figure 6 the softmax function is used to convert the N-dimensional vectors into a probability for each class. (Kulkarni & Shivananda, 2019)

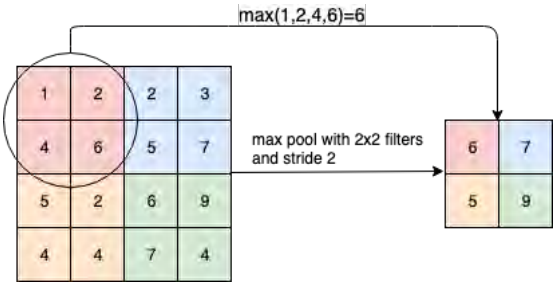


Figure 7: Pooling CNN

CNNs are also used for text classification and other NLP tasks. For text, the filters can look over sequential words in a piece of text represented as n-grams with a 1xn filter. A pooling method extracts the relevant n-grams for making a decision and the rest of the network classifies the text based on this information.

A document can be represented as a real matrix $A \in \mathbb{R}^{n \times d}$, a concatenation of the input word embeddings where n is the document length, the number of tokens in the document. The dimension of a word embedding vector is denoted by d. The document length is fixed, longer documents are truncated and the shorter documents are padded with zeros. Therefore, the input layer is a sequence x containing n entries: $x_1, x_2, ..., x_n$. Each entry is represented by a d-dimensional vector: $x_1, .., x_n \in \mathbb{R}^{n \times d}$. For text a CNN model can be used with multiple filters of different sizes where you can look at bi-grams (a 1x2 filter), tri-grams (a 1x3 filter), or n-grams (a 1xn filter) within the text.

To summarise, a common architecture for a CNN for text classification is an architecture where each word in a document is represented by an embedding vector. A convolutional layer with m filters is applied which produces an m-dimensional vector for each document n-gram. After the convolutional layer, a pooling strategy is used. The extracted features from the filters are then passed to a fully connected softmax layer. The output of this layer is the probability distribution over labels, see Figure 8.

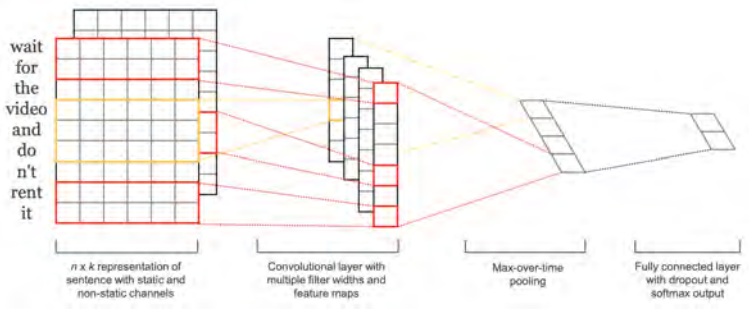


Figure 8: Architecture of a CNN for NLP

4.2.3 Recurrent Neural Network

With CNNs long-range dependencies are not captured. However, CNNs can detect patterns of multiple adjacent words (e.g. bigram, trigram and fourgram). These patterns could be expressions like 'He hates' (a bigram) and 'You are awesome' (a trigram). RNN is a FFNN developed to work with sequences. Sequential input in a text can, for example, be a sequence of words (sentences) or sequences of characters (words). A RNN is a chain of simple neural layers that share the same parameters, as shown in Figure 9. The output depends on the previous computations, the RNN maintains a hidden state h_t, which stands for the memory of the sequence at time step t.

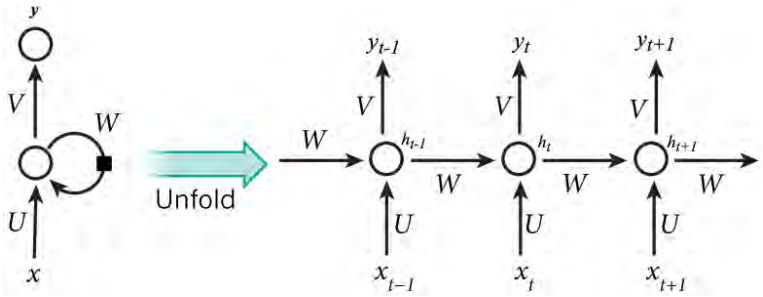


Figure 9: Unrolled RNN

A T length input sequence for a RNN can be defined as X, where $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, with $\mathbf{x}_t \in \mathbb{R}^N$ a vector input at time t. The output is an ordered list of hidden states, the 'short-term' memory of the RNN, $h_0, .., h_T$, with an initial hidden state h_0 which is initialised to all zeros. And it returns the output vectors $\{y_1, \ldots, y_T\}$, which can be used as input for the other RNN units. (Deng & Liu, 2018) (Ruder, 2019) (Tixier, 2018)

For every time step a RNN performs the following operations:

$$\begin{aligned} \mathbf{h}_t &= \sigma_h \left(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h \right) \\ \mathbf{y}_t &= \sigma_y \left(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y \right) \end{aligned} \tag{17}$$

where σ_h and σ_y are activation functions. The RNN modifies the previous hidden states $\mathbf{h}_{t-1}$ by applying a transformation $\mathbf{U}_h$. The new hidden state is obtained by a transformation $\mathbf{W}_h$ to the current input $\mathbf{x}_t$. The output $\mathbf{y}_t$ is produced for every time step t. (Ruder, 2019)

In practice, RNNs have problems with learning longer-range dependencies due to the vanishing/exploding gradient problem. When training the network, the gradient values are used when updating the weights of the neural network. The recurrent multiplication in the back-propagation step can cause either very large weights or very small weights which will respectively cause the gradients to grow exponentially

or shrink exponentially. When this is the case, the network will stop training and does not learn anymore. If that happens, a RNN can forget what it has seen at the beginning of longer sequences and therefore has a short-term memory.

There are many methods to combat the vanishing/exploding gradient problem, most of them are focused on the initialisation or controlling the size of the propagated gradients. The most common methods add additional gates to the RNNs. Two methods are the LSTM or the GRU which can retain information over longer periods. (Nguyen, 2019a) (Nguyen, 2019b) (Ruder, 2019) (Deng & Liu, 2018)

4.2.4 Long Short-Term Memory

LSTM (Hochreiter & Schmidhuber, 1997) has a cell state, which can decide what should be remembered and forgotten. The hidden state in the vanilla RNNs is computed with a single layer $\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$. Instead of a single layer, a LSTM unit computes the hidden state using four interacting layers that give the network the ability to remember or forget specific information about the preceding elements in the sequence. The LSTM has a forget gate $\mathbf{f}_t$, an input gate $\mathbf{i}_t$, and an output gate $\mathbf{o}_t$. They are functions of the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_t$. The gates interact with the previous cell state $\mathbf{c}_{t-1}$, the current input, the current cell state $\mathbf{c}_t$. (Ruder, 2019) Figure 10[1] shows the operations in the LSTM cell.

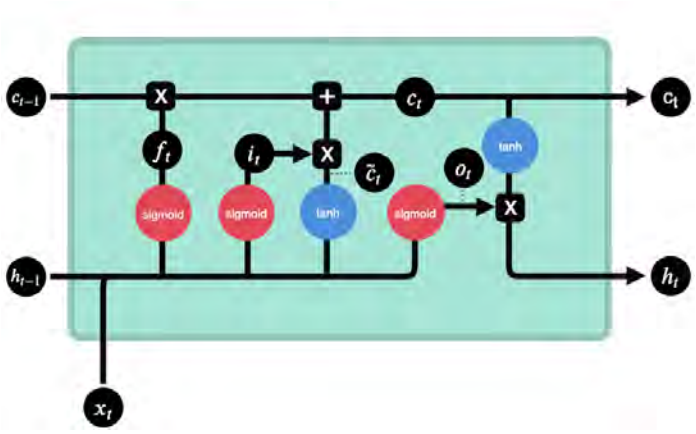


Figure 10: LSTM Cell

[1] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

The LSTM cell is formally defined as:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(\mathbf{W}_g \mathbf{x}_t + \mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{b}_g\right) \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh\left(\mathbf{c}_t\right),
\end{aligned} \tag{18}$$

where $\mathbf{W}$ and $\mathbf{U}$ are learned weight matrices, σ is the sigmoid activation function, tanh the tanh activation function, $\circ$ the elementwise multiplication (Hadamard product), and $\mathbf{b}$ the bias vectors.

The input, forget and output are called gates because of the sigmoid activation function. The values are passed through a sigmoid function which transforms the values between a range 0 and 1. The forget gate ($\mathbf{f}_t$), that means that it will decide how much the memory cell should forget (the previous state), is based on the previous hidden state, and the current input. The input gate ($\mathbf{i}_t$) determines what information is relevant to add from the current step, the current input you want to let through. The output gate ($\mathbf{o}_t$) determines what the next hidden state should be.

The 'candidate' hidden state ($\tilde{\mathbf{c}}_t$) is computed based on the current input and the previous hidden state.

The internal memory, ($\mathbf{c}_t$) combines the previous memory ($\mathbf{c}_{t-1}$) and the new input. The old memory could be ignored (forget gate all 0's) or the new cell state could be ignored (input gate set all to 0's), but probably something in between this.

The output hidden state h_t is computed based on the updated memory ($\mathbf{c}_t$) and the output gate.

4.2.5 Gated Recurrent Unit

GRU (Cho et al., 2014) is another gating structure for a RNN and similar to a LSTM unit, only with fewer gates. GRU has two gates, a reset gate ($\mathbf{r}_t$) and a update gate ($\mathbf{z}_t$) without internal memory $\mathbf{c}_t$, see Figure 11[2].

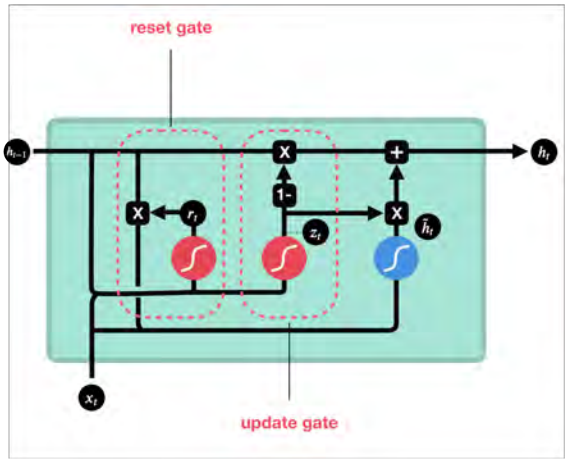


Figure 11: GRU Cell

The equations for the GRU are:

$$\begin{aligned}
\mathbf{r}_t &= \sigma\left(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r\right) \\
\mathbf{z}_t &= \sigma\left(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z\right) \\
\tilde{\mathbf{h}}_\mathbf{t} &= \tanh\left(\mathbf{W}_\mathbf{h} \mathbf{x}_\mathbf{t} + \mathbf{U}_\mathbf{h}\left(\mathbf{r}_\mathbf{t} \circ \mathbf{h}_{\mathbf{t}-\mathbf{1}}\right) + \mathbf{b}_\mathbf{h}\right) \\
\mathbf{h}_t &= \left(1 - \mathbf{z}_t\right) \circ \tilde{\mathbf{h}} + \mathbf{z}_t \circ h_{t-1}
\end{aligned} \tag{19}$$

The reset gate $\mathbf{r}_t$ decides whether it ignores the previous hidden state. When the reset gate is close to 0, the hidden state ignores the previous hidden state and only takes the current input. The update gate $\mathbf{z}_t$ decides whether the hidden state is updated with a new hidden state, $\tilde{\mathbf{h}}_t$, which defines how much of the previous memory should be kept. (Nguyen, 2019a) (Ruder, 2019)

[2] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

4.3 Attention

RNNs cannot capture information of very long sequences. Attention mechanisms address this issue and are loosely based on how humans pay attention. For example, if we would translate a long sentence from one language to another language, we will focus on important parts of sequences or regions that we are translating. Attention works similarly for NN. (Deng & Liu, 2018) (Tixier, 2018).

The attention mechanism was developed in the context of the encoder-decoder architectures for Neural Machine Translation (NMT) (Bahdanau, Cho, & Bengio, 2014) and is also applied to other tasks such as image captioning. The traditional methods for neural machine translation are methods based on an encoder-decoder architecture, both of them are RNNs.

The encoder encodes the input into a fixed-length context vector. The fixed-length context vector is the representation of the text. This context vector is then decoded into the output sequence by the decoder, see Figure 12. (Bahdanau et al., 2014) (Deng & Liu, 2018)

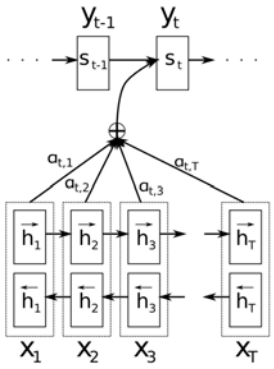


Figure 12: Neural Machine Translation by Jointly Learning to Align and Translate

The hidden states at time i takes three inputs:

- the previous hidden state of the decoder s_{i-1},
- the prediction from the previous time step y_{i-1}, and
- a context vector c_i which weighs the appropriate hidden states for the given time step.

For each hidden representation, an attention score α_{ji} is calculated. Based on the attention scores for each hidden representation a context vector c_i is calculated. The hidden state of decoder, s_i, is computed by:

$$s_i = f\left(s_{i-1}, y_{i-1}, c_i\right) \tag{20}$$

The context vector c_i depends on the annotations $(h_1, .., h_{T_x})$. Each annotation h_i contains information about the whole input sequence, with a focus on the parts surrounding the i-th word of the input sequence.

The context vector c_i is computed as the weighted sum of the annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{21}$$

The attention weights α_{ij} for each annotation h_j is computed by:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)} \tag{22}$$

where

$$e_{ij} = a\left(s_{i-1}, h_j\right), \tag{23}$$

is an alignment model and scores how well the inputs around the position j and the output at position i match. The alignment model a is a FFNN.

With attention, instead of encoding the full source sentence into a fixed-length vector decoder, it allows attending to different parts of the source sentence at each step of the output generation. The models learn what to attend to based on the input sentence and what it has produced so far.

4.4 Transformer Network

Sequence-to-sequence tasks are based on RNNs or CNNs with an encoder and a decoder structure. The best performing models use the attention mechanism to connect the encoder and the decoder. Instead of using either recurrence or convolutions, the Transformer (Vaswani et al., 2017) applies attention directly to the input. The Transformer achieves a higher performance for both the recurrent and the convolutional models for machine translation. It does not rely on the memory of the RNNs from the previous states but instead uses 'multi-headed' attention directly on the input embeddings. This allows the model to perform parallel computations and is, therefore, faster to train. (Vaswani et al., 2017)

The Transformer network extends the mechanism of the traditional attention mechanism. Instead of giving only the decoder access to the entire input sequence it processes the input and output sentences as well. It allows the encoder and decoder to directly model these dependencies of the encoder and the decoder, instead of going from left-to-right using RNNs. The Transformer consists of an encoder and a decoder, the architecture of the Transformers is given in Figure 13.

The encoder and the decoder have multiple layers of multi-head attention with residual connections and fully connected layers. A residual connection takes the input and

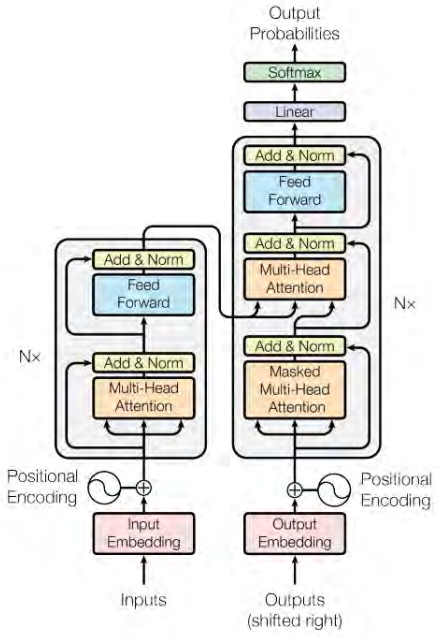


Figure 13: Transformer Architecture

adds it to the output of the sub-network. Because the computations happen in parallel, they use a masking technique in the decoder to prevent positions from attending to subsequent positions. The output embeddings are also offset by one position, to make sure that the predictions for position i can only depend on the known outputs at positions smaller than i. The transformer achieves state-of-the-art results while significantly improving the computation time because it is faster to train and more parallelizable. (Vaswani et al., 2017) (Kamath et al., 2019)

Figure 14 shows the multi-head attention, which is defined by three input matrices. Let $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ be matrices of vectors of the queries, keys and values. The attention functions used in Transformer is identical to dot-product attention, a commonly used attention function, with an additional scaling factor of $\frac{1}{\sqrt{d_k}}$. The attention mechanism used is called the 'Scaled Dot-Product Attention', Figure 14.

The attention is calculated as follows:

$$\text{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right)\mathbf{V},$$

computes the dot product of the query and the keys, scaled by the $\sqrt{d_k}$ and normalised by a softmax function to obtain the weights on the values. The weighted sum is calculated by applying the weights of the values to the values.

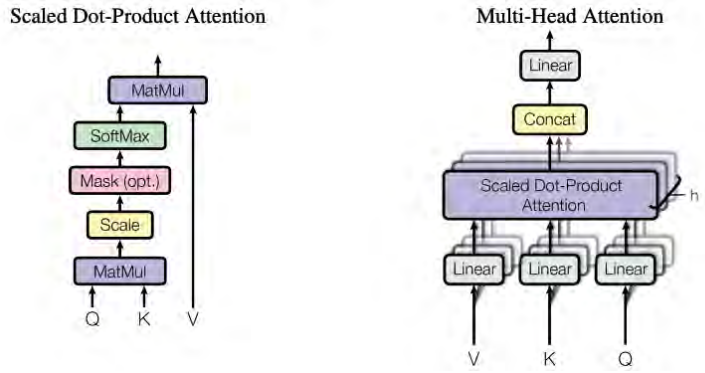


Figure 14: Scaled Dot-Product Attention (left) / Multi-Head Attention (right)

Instead of one attention function, the Transformer Network uses parallel attention functions. The output values are then concatenated and projected.

The multi-head attention is defined as:

$$\begin{aligned} \text{MultiHead}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right) &= \text{Concat}\left(\text{head}_1,\ldots,\text{head}_\text{h}\right)\mathbf{W}^O \\ \text{where head}_\text{i} &= \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V\right), \end{aligned} \tag{24}$$

where the projections are parameters matrices $\mathbf{W}_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

4.5 Bidirectional Encoder Representations from Transformers

BERT is a state-of-the-art model for many NLP tasks. It is built upon recent work in pre-training contextual representations, like ELMo (Peters et al., 2018) and ULMFit (Howard & Ruder, 2018). Contextual representations from context-free representations. Context-free representations create a single word embedding representation for each word in the vocabulary. Contextual representations, however, generate an embedding representation of each word based on the context of the surrounding words. BERT uses both the previous context and the next context of the word. It is the first deeply bidirectional, unsupervised language representation, pretrained only using a plain text corpus. (Jacob Devlin & Chang, 2018)

BERT is based on a language model, see Section 3.2. But instead of predicting the next word after a sequence of words. BERT randomly masks words in the sentence and predicts them.

Language models are typically from left-to-right:
'the man went to a store'

$$P(\text{the} \mid \text{<s>}) \cdot P(\text{man}|\text{<s> the}) \cdot P(\text{went}|\text{<s> the man}) \cdot \ldots$$

For downstream tasks, you mostly do not want to use a language model but get the best representation for the words. If the contextual representation of each word is only based on the context of the left words, information is missing. Additional to a language model from left-to-right, a model from the right-to-left model can be trained. The two representations for each word can be concatenated to use for the downstream tasks.

$$P(\text{store}|\text{</s>}) \cdot P(\text{a}|\text{store </s>}) \cdot \dots$$

Instead of concatenation two representation of each word, one left-to-right and one right-to-left, it would be better to train a single deeply bidirectional model. However, it is not possible to train deep a bidirectional model as a language model. Doing so would create a cycle, in which words can indirectly see themselves. See themselves refers to the fact that it creates a circular reference where a word's prediction is based on the word itself, they can see themselves via the context of another word.

To train a deeply bidirectional LM and prevent words from seeing themselves, the model was trained with a method which is called Masked LM. This is often called a Cloze task. The Masked LM predicts, instead of the next for a sequence of words, random words from within the sequence. A percentage of the words from the input is masked and the task is to reconstruct those words from the context. BERT forces the model to learn how to use information from the sentences when trying to guess the missing words.

Input Embeddings
The input for BERT consists of token embeddings, segment embeddings, and position embeddings. A sentence is tokenized and two extra tokens are added, one at the start ([CLS]) and one at the end of the sentence ([SEP]). The tokenization method uses word pieces. (e.g. playing -> play + ##ing) instead of words. This helps to reduce the size of the vocabulary and also maps the out-of-vocabulary words. Each word piece token is converted into a 768-dimensional vector representation.

Transformers do not encode the sequential nature of the input. To address this problem, positional embeddings are added to let the model learn the sequential ordering of the input. BERT learns a vector representation for each position, which helps to determine the position of each word.

Because BERT can be trained on pairs of sentences, the segment embeddings are embeddings, which can distinguish between the first or the second sentences. It can, therefore, learn a unique embedding for the first and the second sentence.

The tokenised input sequence of length n will have three distinct representations:

- Token Embeddings
- Segment Embeddings
- Position Embeddings

To produce a single representation, these embeddings are summed element-wise with shape $(1, n, 768)$, see Figure 15.

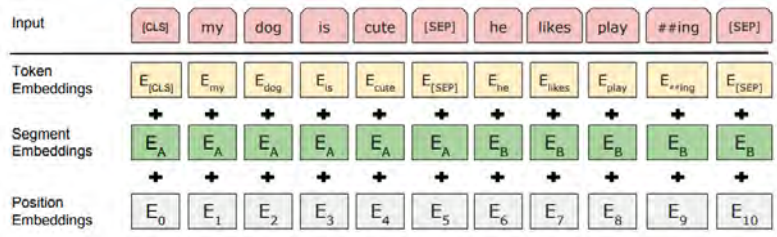


Figure 15: BERT Input Representation

Masked Language Model
BERT uses the masked language model (MLM) to address the unidirectional constraints. It randomly masks a percentage of the input tokens and then predicts these masked tokens. To prevent that the model only tries to predict when a [MASK] token is present it replaces some masked tokens with random words to add some noise. A data generator chooses 15% of the tokens at random and then it follows the following procedure:

- 80%: Replace the word with the [MASK] token
- 10%: Replace the word with a random word
- 10%: Keep the word unchanged

Input: the man [MASK1] to [MASK2] store
Label: [MASK1] = went; [MASK2] = a

A Transformer encoder is used to predict the masked word, the final hidden states corresponding to the masked position. To understand the relationships between sentences, they use a binary classification task to pre-train a sentence relationship model. The classification task is given two concatenated sentences, A and B, to predict if sentence B comes after sentence A in the text.

Next Sentence Prediction
Next, to the MLM, BERT uses Next Sentence Prediction to model the relationship between two sentences. If the input of the BERT model is two sentences, it separates these two sentences with a [SEP] token.

BERT gets then as input either a random next sentence or the actual next sentence and has to try to predict whether the second sentence is random or not random.

Input: the man went to the store [SEP] he bought a gallon of milk
Label: IsNext

Input: the man went to the store [SEP] penguins are flightless birds
Label: NotNext

The model is trained on a big corpus, English Wikipedia (2,500M words) and the BooksCorpus (800M words). To use the model for a downstream task, the model can be fine-tuned using three or four epochs. BERT is based on a multiple layer self-attention model, and then this model is fine-tuned by adding a classification layer.

Attention Patterns
While BERT and other large NNs can achieve high performance, it is not clear what aspects of language these models learn. Recent papers aim to investigate what aspects of language, linguistic features BERT learns from the unlabelled data. A paper by (Clark, Khandelwal, Levy, & Manning, 2019) shows that BERT learns specific linguistic features by examining the output of the language models. Figure 16 gives examples of different attentions patterns.

BERT's attention heads show patterns such as attending to delimiter tokens, specific positional offsets, or broadly attending over the whole sentence, with heads in the same layer often showing similar behaviour. They also show that certain attention heads capture syntax and co-reference. For example, they find heads that attend to the direct objects of verbs, determiners of nouns, objects of prepositions, and co-referent mentions. Lastly, they show that syntactic information is also captured in BERT's attention (Clark et al., 2019).

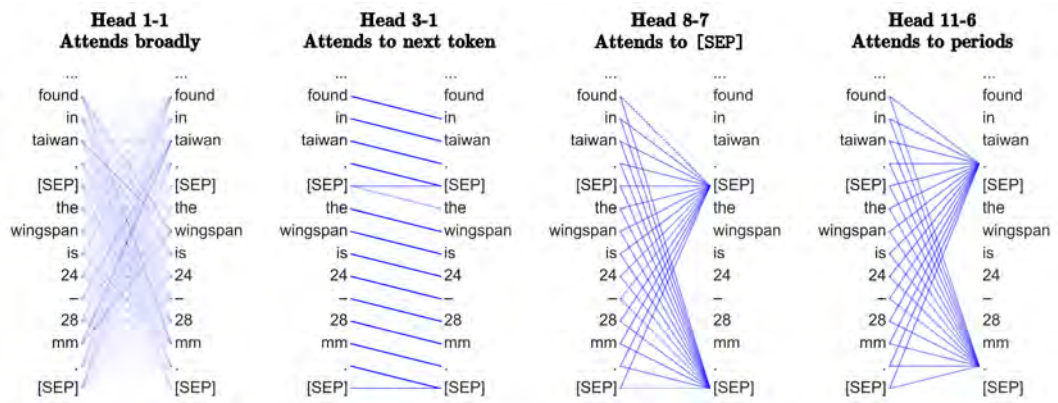


Figure 16: Example of Attention Patterns for Different Heads

5 Data

In this section, we will describe the dataset in Section 5.1, a data analysis is given in Section 5.2. The pre-processing steps will be discussed in Section 5.3.

5.1 Description

We will use the Offensive Language Identification Dataset (OLID) provided by (Zampieri et al., 2019). OLID was created using a Twitter API searching for tweets containing certain selected keyword patterns which are often used in offensive posts. These were keyword patterns such as 'she is', 'you are', 'he is', and to:BreitBartNews. Also, tweets which are marked as unsafe are used. The data has been annotated using a crowd-sourcing platform. If there was 100% agreement, they considered it as an acceptable annotation. If there was no annotator agreement, they asked for more annotations until an agreement above 66% was reached. More information about the collection and creation of the dataset can be found in (Zampieri et al., 2019).

OLID has a three-layer annotation scheme. The first layer of the annotation scheme is the Offensive Language Detection, which identifies whether a tweet is offensive (OFF) or non-offensive (NOT). The second layer of the annotation scheme is the Categorisation of Offensive Language, the type of offence is categorised in targeted (TIN) and untargeted (INT) insults and threats. The third layer is the Offensive Target Identification, the targets are categorised in individual (IND), group (GRP), and other (OTH). (Zampieri et al., 2019)

The terms used for the categorisation of the tweets are defined as:

- *Not Offensive (NOT)*: Posts that do not contain offence or profanities.
- *Offensive (OFF)*: Posts that contain any form of non-acceptable language (profanity) or a targeted offence, veiled or direct. Including insults, threats, posts containing profane language or swear words.

- *Target Insult (TIN)*: Posts that contain insults/threats to an individual, group, or others.
- *Untargeted (UNT)*: Posts that contain insults/threats but are not targeted to a certain individual, group, or others.

- *Individual (IND)*: (Cyberbullying) Posts which are targeted to an individual (named individual, a famous person, or an unnamed participant).

- *Group (GRP)*: (Hate Speech) Posts that are targeted to a group of people based on ethnicity, gender, sexual orientation, political, religious belief, or other common characteristics.
- *Other (OTH)*: Posts that are not targeted to an individual or a group (e.g. organisation, situation, event, issue).

5.2 Exploratory Data Analysis

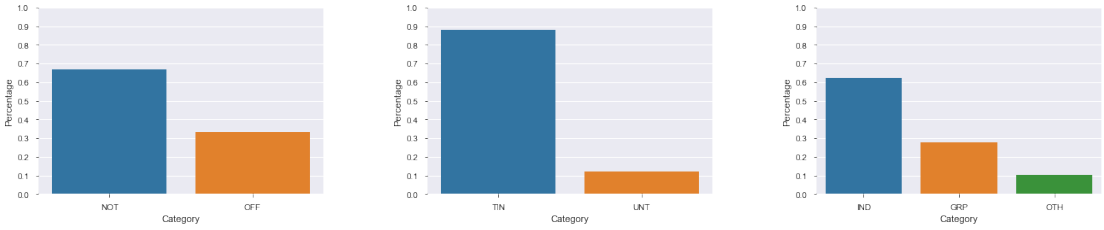
The dataset consists of 14,100 tweets with 13,240 as training data and the remaining 860 as test data. The columns in the dataset are id, tweet, subtask_a, subtask_b, and subtak_c. Table 1 shows an example of the tweets in the dataset.

1. *id*: Unique identifier of a tweet
2. *tweet*: Tweet text
3. *subtask_a*: Offensive Language Detection (OFF, NOT)
4. *subtask_b*: Categorisation of Offensive Language (TIN, UNT)
5. *subtask_c*: Offensive Target Identification (IND, GRP, OTH)

Tweet	A	B	C
@USER Yes you are but I was asking what is it about the movie ? 👀	NOT		
@USER True point. Gun control is total bullshit.	OFF	UNT	
@USER @USER Go home you're drunk!!! @USER #MAGA #Trump2020 👊🇺🇸👊	OFF	TIN	IND
@USER @USER @USER @USER @USER You must be talking about hateful conservatives.	OFF	TIN	GRP
@USER Awful	OFF	TIN	OTH

Table 1: Examples of Tweets

The distribution of tweets in the three different classes is shown in Figure 17, which show that the classes are quite imbalanced. The tweets that contain offensive language are only around one third.



(a) Distribution Subtask A (b) Distribution Subtask B (c) Distribution Subtask C

Figure 17: Distribution Subtasks

We might distinguish offensive tweet from non-offensive tweets if we take a look at the content of the tweets. Figure 18 shows common words in the two given categories. On the one hand, the two categories have certain words in common. On the other hand, Figure 18a shows that offensive tweets contain more swear words and non-offensive tweets talk more about Antifa.

(a) Offensive Tweets (b) Non-Offensive Tweets

Figure 18: Wordclouds OFF / NOT Tweets

Furthermore, we can take a look at certain properties of tweets and check if these are different based on the category. Figure 19 shows that offensive tweets contain on average more characters than non-offensive tweets.

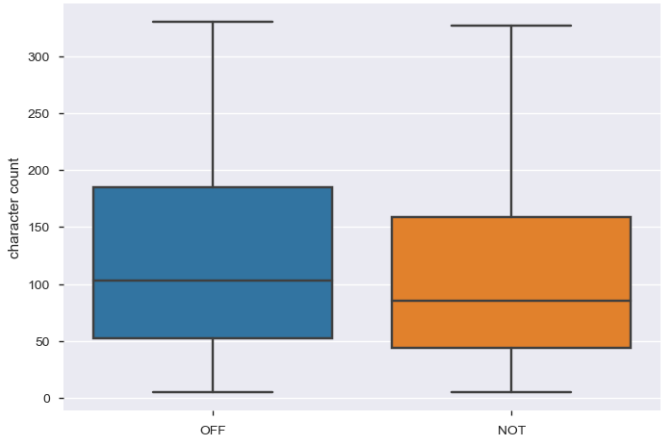


Figure 19: Character Count per Tweet

Hashtags
A hashtag (#) on Twitter is used to index keywords or topics on Twitter. A hashtag is a tag consisting of a string of characters with a "#" symbol as a prefix. There are no space delimiters in hashtags, which makes the segmentation not trivial. We will analyse the hashtags to examine if hashtags influence the classification of the tweet. Hashtags used in the Twitter dataset are given in Figure 20.

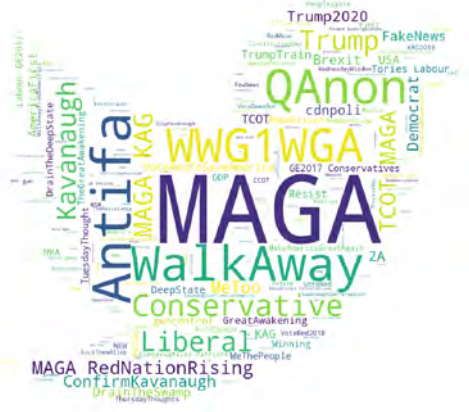
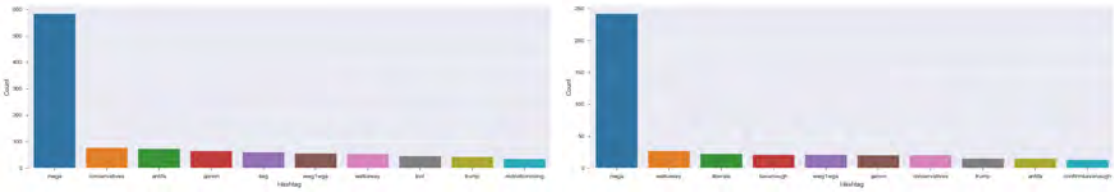


Figure 20: Wordcloud Hashtags

There are hashtags used where the meaning might not be clear. An overview of the meaning of the most used hashtags is given below:

- #MAGA - Make America Great Again
- #Kavanaugh - American lawyer and jurist who is an Associate Justice of the Supreme Court of the United States
- #WWG1WGA - Where We Go 1 We Go All
- #qanon - QAnon conspiracy theory that Deep State is determined to undermine the presidency of Donald J. Trump.
- #antifa - Anti-fascism
- #kag - Keep America Great
- #tcot - Top Conservatives on Twitter

We also wanted to compare the hashtags used in offensive tweets and non-offensive tweets. The hashtags which are used the most for respectively non-offensive and offensive tweets are given in Figure 21a and Figure 21b.



(a) Hashtags NOT (b) Hashtags OFF

Figure 21: Commonly Used Hashtags

We pre-process the hashtag and split them into proper words. For example, #DissolveTheUnion is split into 'dissolve the union'. We will use word segmentation, which decides where the word boundaries are, which is useful when parsing hashtags. WordSegment [3] is a module for the word segmentation of English words and is based on the book Beautiful Data (Segaran & Hammerbacher, 2009). This package returns word segments of the hashtags. It contains uni-gram data including the most common 333,000 words and bi-gram data including the most common 250,000 phrases.

Emojis and emoticons

Emojis are often used in social media. The Oxford Dictionary defines an emoji as 'A small digital image or icon used to express an idea or emotion'. Emoticons are defined as 'A representation of a facial expression such as a smile or frown, formed by various combinations of keyboard characters and used to convey the writer's feelings or intended tone'. The presence of emojis is interesting because they often contain some form of sentiment.

There are different emojis presented in the tweets. From the training set, 11% of the tweets contain emojis. The most used emojis are given in Table 2. If a tweet contains emojis, the tweet contains on average 2.4 emojis. There are 80 tweets which contain emoticons (0.60%). There are only 18 unique emoticons used in the tweets. From the offensive tweets, there are around 9.14% which contain emojis. From the non-offensive tweets, there are slightly more tweets which contain emojis (12.3%). The top emojis used in offensive tweets and non-offensive tweets are different. Therefore this might be an indicator for identifying a tweet. The top used emojis in the offensive tweets are emojis such as 😡, 👇, 🤦‍♂️. Whereas for the non-offensive tweets this the emojis ❤️, 👍, and 😍 are used.

[3] `http://www.grantjenks.com/docs/wordsegment/`

Emojis	Count
😂	546
🇺🇸	378
🤣	213
❤️	135
😭	102
👍	89
😍	83
🤔	79
👏	71
🙏	68

Table 2: Top 10 Most Used Emojis

There are different strategies to incorporate emojis. Emoji embeddings can be used alongside other word embeddings. Emoji2vec (Eisner, Rocktäschel, Augenstein, Bosnjak, & Riedel, 2016) pre-train emoji embeddings using positive and negative emoji descriptions. Another strategy is to replace emojis are replaced by their textual descriptions (Singh, Blanco, & Jin, 2019). They showed that this strategy outperforms previous methods for irony detection and sentiment analysis.

During the pre-processing of the tweets, we will, therefore, replace emojis by their textual description. We will use the emoji Unicodes with their CLDR Short Name. [4] To do this, we will use a customised version of the Python package emoji. [5] We did not take into consideration the images with different skin tones and removed distinctions based on the colour of the skin.

[4] https://www.unicode.org/emoji/charts-12.0/emoji-list.html
[5] https://github.com/carpedm20/emoji/

5.3 Pre-processing

A tweet contains unnecessary information which can be removed. We apply pre-processing steps to clean the tweet. Based on the word embedding type we use, we will apply different pre-processing steps.

SVM
For the SVM model we will use word count features (TFIDF), see Section 3.3, for the TFIDF we use the following pre-processing steps:

- Remove emojis and emoticons.
- Remove Twitter handles (@USER).
- Remove URLs.
- Shorten repeated characters, extra white space or lengthened words and remove repeated characters.
- Expand contractions by replacing the following words with the corresponding words for example, what's $\rightarrow$ what is, I'm $\rightarrow$ I am, etc.
- Convert words to lowercase.
- Remove stopwords.

MLP, CNN, LSTM
For the MLP, CNN, and LSTM we use pre-trained word embeddings (Word2Vec, GloVe, and fastText). The pre-processing steps for these word embeddings are based on the vocabulary from these word embeddings to reduce the out-of-vocabulary words. When using deep learning NLP models, the preprocessing steps stopword removal and stemming should not be used. When these steps are performed, valuable information is lost.

When using the word embeddings, we preprocess the tweets to get the vocabulary as close to the embeddings as possible. We based the preprocessing steps on minimising the list of out of vocabulary (OOV) words, which is the intersection between the vocabulary and the word embeddings.

For GloVe, pre-trained word embeddings for Twitter, we will use the preprocessing steps which they also use for preprocessing the tweets. The preprocessing steps for GloVe are similar to the one provided in the Ruby script [6] for preprocessing Twitter data.

[6] `https://nlp.stanford.edu/projects/glove/preprocess-twitter.rb/`

- Replace Twitter handles (@USER) by <user>.
- Replace URLs by <url>.
- Replace numbers by <number>.
- Replace emoticons by their description <smile>, <lolface>, <sadface>, or <neutralface>.
- Replace emojis with their textual description and remove skin tones.
- Segment hashtags in different words.
- Mark punctuation repetitions with <repeat>, e.g. "!!!" => "! <repeat>"
- Mark elongated words with <elong>, e.g. "wayyyy" => "way <elong>".
- Convert words to lowercase, and mark words in which all the letters are capital letters.
- Replace contractions and slang.

Word2Vec with pre-trained vectors trained on part the of Google News dataset does not have special pre-processing steps for Twitter data. We keep the same pre-processing steps for the hashtags, converting words to lowercase, and replace contractions and slang. The pre-processing steps for fastText, word trained on Common Crawl, are similar to the one for Word2Vec.

- Replace Twitter handles (@USER) by user.
- Replace URLs by url.
- Replace numbers by hashtags, e.g. 20 is replaced by ## and 120 is replaced by ###.
- Replace emojis with their textual description and remove emoticons.
- Remove punctuation, most of the punctuation is not contained in the vocabulary and therefore removed.

Table 3 shows the OOV coverage before preprocessing and after preprocessing.

	Before preprocessing		**After preprocessing**	
Embedding	Vocab	Text	Vocab	Text
Word2Vec	45.93%	68.58%	85.78%	98.17%
Glove	29.57%	61.73%	91.60%	99.24%
fastText	53.39%	77.94%	88.00%	98.92%

Table 3: Vocabulary Coverage for Word2Vec, GloVe, fastText

BERT

We took minimal preprocessing steps when using BERT. We only removed the USER and URL, handled the emojis and emoticons and expanded the contractions.

The pipeline of the whole classification process is given in Figure 22. We start with the text, perform tokenization and apply pre-processing steps, described in Section 5.3. After the pre-processing, we extract different features based on the model which we are going to use. We train the model with the given features. After model training, we predict and evaluate with the use of labels annotated by humans.

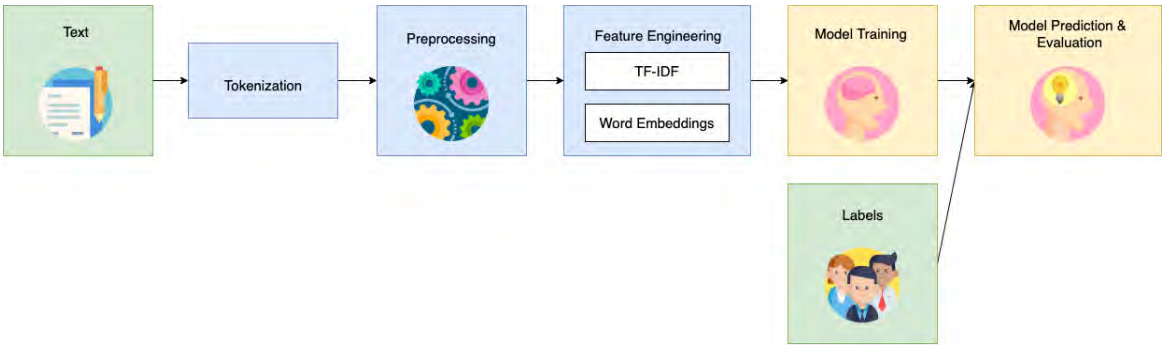


Figure 22: Pipeline Overview

6 Experimental Setup

This section describes the setup of the experiments. For each of the models, the parameters are tuned using Hyperopt. [7] The architectures of the models are described in Section 6.1. The hyper-parameters search is performed using Bayesian optimisation, which we will describe in Section 6.2.

To see how the models perform, we use a validation set. The training dataset was split into a training (80%) and a validation set (20%). The test set was already provided.

6.1 Architecture

We use the models SVM, MLP, LSTM, CNN and BERT for the classification. The models are implemented using Python and programmed using PyTorch. PyTorch is an open-source machine learning library for Python, based on Torch. It was developed by Facebook's AI research group and is used for applications such as computer vision and natural language processing. PyTorch is flexible and easy to write your layer types and run on a GPU. For the SVM model, we used scikit-learn.

Step	
Pre-processing	See pre-processing steps
Feature Extraction	TFID - Word Embeddings
Models	SVM, MLP, LSTM, CNN, BERT
Classification	OFF (Offensive), NOT (Not Offensive)

The pipeline for the classification is given in Figure 23.

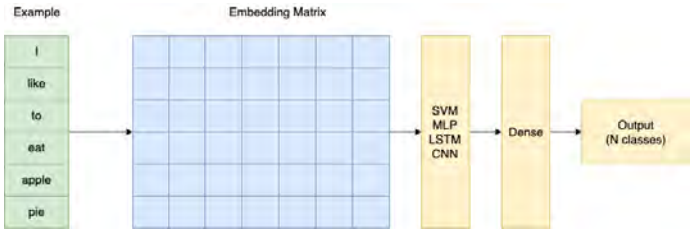


Figure 23: Pipeline Architecture

[7] https://hyperopt.github.io/hyperopt/

The architecture of the different models consists of different layers. [8] The layers we used are:

- **Embedding layer**
  The embedding layer is used to store the word embeddings of a fixed dictionary and size in a lookup table and retrieve them using the indices. The input of the embedding layer is a list of indices, the size of the dictionary of the embeddings. The output is the corresponding word embeddings, the size of each embedding vector.

  Parameters:

  - num_embeddings: Size of the dictionary of embeddings
  - embedding_dim: Size of each embedding vector

- **Dropout layer**
  The dropout layer randomly zeroes some of the elements of the input with probability p,m using samples from a Bernoulli distribution.

  Parameters:

  - p: Probability of an element to be zeroed.

- **Linear layer**
  The linear layer applies a linear transformation to the incoming data:
  $y = xA^T + b$.

  Parameters:

  - in_features: Size of each input sample
  - out_features: Size of each output sample

[8] `https://pytorch.org/docs/stable/nn.html`

- **Convolution layer**
  The convolution layer applies a convolution over the input.

  Parameters:

  - in_channels: Number of channels in the input image
  - out_channels: Number of channels produced by the convolution
  - kernel_size: Size of the convolving kernel

- **LSTM layer**
  The LSTM layer applies a multi-layer LSTM RNN to an input sequence.

  Parameters:

  - input_size: Number of expected features in the input x
  - hidden_size: Number of features in the hidden state h
  - num_layers: Number of recurrent layers
  - dropout: Uses a Dropout layer on the outputs of each LSTM layer, except the last layer.
  - bidirectional: If True, the LSTM becomes bidirectional

MLP Architecture
The MLP architecture consists of an embedding layer and multiple linear layers, where the last linear layer has an output dimension which is equal to the number of classes. There are ReLU activation functions after every linear layer and in the second linear layer, we use dropout. The dropout rate and the hidden dimension are both hyperparameters.

The architecture of the MLP is as follows:

- Embedding layer
- Linear Layer + ReLu: Embedding dimension (in_features), hidden dimension (out_features)
- Dropout layer: Dropout rate
- Linear layer + ReLu: Hidden dimension (in_features), hidden dimension (out_features)
- Linear layer + ReLu: Hidden dimension (in_features), hidden dimension/2 (out_features)
- Linear layer (Output layer): Hidden dimension/2 (in_features), 2 neurons (out_features)

LSTM / LSTM Attention Architecture
The LSTM and the LSTM with attention architecture consist of an embedding layer, followed by a LSTM layer and a linear layer, the output layer with a softmax activation function. A dropout layer is added to the hidden layer to prevent overfitting.

The architecture of the LSTM is as follows:

- Embedding layer
- LSTM layer: embedding dimension (in_features)
- Linear layer (Output layer): hidden dimension (in_features), 2 neurons (out_features)

The architecture of the LSTM with attention is similar, after the LSTM layer an attention layer is applied.

CNN
For the CNN we handle the tweets as images. We create a matrix with the number of rows equal to the maximum length of the tweet and the number of columns is equal to the embedding dimension. For an image, we move the convolution filter horizontally and vertically, for the text we only going to move vertically. The kernel size, size of convolving kernel is a list of [2, 3, 4]. The number of consecutive words we are looking over.

The CNN consist of an embedding layer, followed by a convolutional layer with different filter sizes to capture information from 2-grams, 3-grams and 4-grams. The feature maps produced by the convolutional layer are passed onto the Max pooling layer. We used a pooling layer to extract the maximum value from the filters. A dropout layer is added and lastly a linear layer to make a prediction.

The architecture of the CNN is as follows:

- Embedding layer
- Convolution layer + Max pooling: 1 (in_channels), embedding dimension (out_channels), 2/3/4 (kernel_size)
- Dropout: dropout
- Linear layer: embedding dim * filter sizes (in_features), 2 neurons (out_features)

BERT
We explored two different approaches, the fine-tuning approach and feature-based approach. The fine-tuning approach is to fine-tune the parameters of the model. With the feature-based approach, you extract fixed features from the pretrained BERT model. It extracts activations from one or more layer without fine-tuning any parameters. We tried different feature-based approaches, took the second-to-last hidden, last hidden,

weighted sum last four hidden, concatenate last four hidden, and a weighted sum of all the 12 layers.

For the fine-tuning approach, we experimented with different classifiers architectures on top of BERT, see Figure 24. [9]

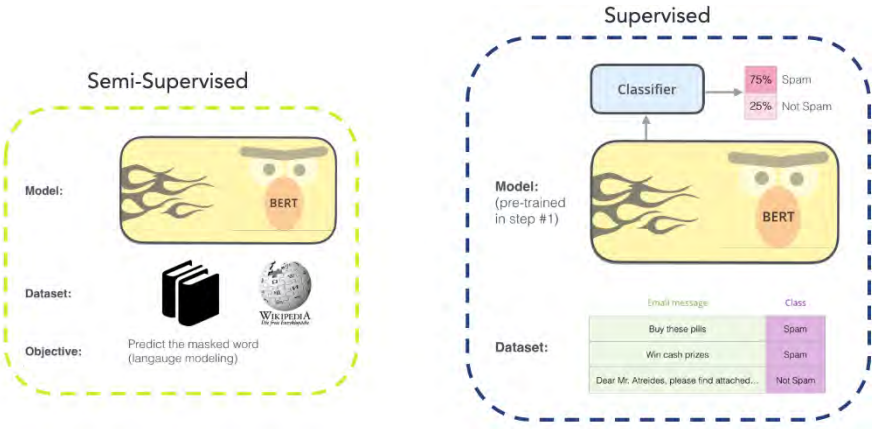


Figure 24: BERT Fine-Tuning

- Bert
  Consists of BERT without any additional model on top.
- BertLinear
  Consists of four linear layers with dropout and ReLU activation functions between the linear layers.
- BertNorm
  Consists of a fully connected linear layer, a batch normalisation, a ReLU layer and after that a linear layer.
- BertLSTM
  Consists of a LSTM model on top of BERT.

[9] https://jalammar.github.io/illustrated-bert/

6.2 Hyperparameters

We use Hyperopt for hyperparameter optimisation. Hyperopt a Python library for optimising over awkward search spaces with real-valued, discrete, and conditional dimensions. Hyperopt provides an implementation for Tree-of-Parzen-Estimators (TPE) which is a Bayesian optimisation method.

Hyperparameter optimization or tuning is choosing a set of optimal hyperparameters for learning an algorithm. The goal of hyperparameter search is to find a set of hyperparameters that gives an optimal model which minimises some predefined loss function on a given test data. There are different approaches for hyperparameter optimization. We will use Bayesian optimisation to obtain better results in fewer evaluations compared to grid search and random search. Bayesian optimisation chooses parameters based on previous performance of the chosen parameters. Grid search and random search spend a lot of time evaluating hyperparameters which are not efficient. Bayesian hyperparameter optimisation builds a probability model of the objective function and uses this to select the most promising hyperparameters to evaluate the objective function. (Snoek, Larochelle, & Adams, 2012)

Table 4 shows the parameter grid used for optimising the parameters. The models are trained with 50 epochs with early stopping after 10 epochs and the batch size is set to 100. The loss function used is Adaptive Moment Estimation (Adam).

Model	**Parameter Grid**
SVM	C: [0.001, 0.01, 0.1, 1, 10] kernel: ['linear', 'sigmoid', 'poly', 'rbf'] degree: [1, 15, 1] gamma: [0.001, 0.01, 0.1, 1]
MLP	learning_rate: [0.00001, 0.01, 0.00005] hidden_dim: [50, 200, 10] dropout: [0.01, 0.5, 0.005]
CNN	learning_rate: [0.00001, 0.01, 0.00005] dropout: [0.01, 0.5, 0.005]
LSTM / LSTMAttention	learning_rate: [0.00001, 0.01, 0.00005] hidden_dim: [50, 200, 10] hidden_layers: [2, 10, 2] dropout: [0.01, 0.5, 0.005]
BERT	learning_rate: [5e-5, 3e-5, 2e-5]

Table 4: Hyperparameter Grid

6.3 Evaluation

We evaluate the classification performance of the classifiers using the evaluation measures *precision*, *recall*, and *F1-score*. The evaluation measures are calculated for each of the individual classes. The classes of the dataset are fairly imbalanced, therefore we also compute the *macro-average* precision, recall, and F1-score. The macro-average calculates the metrics for each class and takes the unweighted mean. We do not use accuracy as it can be misleading for imbalanced data. Accuracy typically awards the correct classification of the majority class.

These metrics are based on the confusion matrix given in Table 5.

		Prediction	
		P	**N**
Actual	**P**	True Positives (TP)	False Positives (FP)
	N	False Negatives (FN)	True Negatives (TN)

Table 5: Confusion Matrix

The terms used in the confusion matrix are defined as:

1. True Positives (TP): Prediction is positive and actual is true.
   I.g.: Prediction tweet is offensive and actual tweet is also offensive.
2. True Negatives (TN): Prediction is negative and actual is true.
   I.g.: Prediction is non-offensive and the actual tweet is offensive.
3. False Positives (FP): Prediction is positive and actual is false.
   I.g.: Prediction is offensive and the actual tweet is non-offensive.
4. False Negatives (FN): Prediction is negative and actual is false.
   I.g.: Prediction is non-offensive but the actual tweet is offensive.

Based on the confusion matrix, we will calculate the accuracy, precision, recall, and F1-score.

The accuracy is the ratio of the correct predictions to the total predictions.

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{25}$$

The precision is the probability that given the predicted class of a sample, the sample is correctly classified and shows how precise/accurate the model is.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{26}$$

The recall is the probability that given a sample, the sample will be correctly classified by the classifier.

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{27}$$

The F1-score is the harmonic mean of precision and recall. It takes both the false positives and false negatives into account.

$$F1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{28}$$

7 Results

This section provides the results of the different models. First, we give an overview of the optimal hyperparameters used for each model. Then, we will evaluate the performance of the models using the found parameters on the test set.

7.1 Hyperparameters

After the preprocessing steps, the average number of words was around 45. Therefore we chose to fix the sequence length of the tweets at 45. The number of features is equal to the vocabulary. The models are trained with 50 epochs with early stopping after 10 epochs. Which means that if the loss did not decrease after 10 epochs we considered the optimisation process as finished. The batch size was set to 100 and the Adam optimisation was used to update the network weights.

We performed Bayesian hyperparameter optimisation using 30 evaluations and the given parameter grid given in Table 4. Table 6 shows the optimal hyperparameters found using the *hyperopt* package. The results given in Section 7.2 are based on the optimal parameters found.

Model	**Parameter Grid**
SVM	C: 10 kernel: linear
MLP	learning_rate: 5.26e-4 hidden_dim: 120 dropout: 0.5
CNN	learning_rate: 4e-4 dropout: 0.31
LSTM	learning_rate: 1.17e-4 hidden_dim: 50 number_of_layers: 2 dropout: 0.5
LSTMAttention	learning_rate: 0.000179 hidden_dim: 100
BERT	learning_rate: 3e-5

Table 6: Optimal Hyperparameters

For the SVM model, we will use a linear kernel. Which means that we will separate the offensive from the non-offensive tweets in a linear way. We will use C=10 as the penalty parameter of the error term. It is the parameter that tells the SVM optimisation how much you want to avoid miss classifying each training values of C. For large values of C, the optimisation will choose a smaller-margin hyperplane.

The learning rate for the models is fairly low, which means that the learning process of the process is also slower. The dropout rate for the models is similar except for the LSTM model with attention. A dropout of 0.5 means that we will randomly drop out 50% by randomly of the nodes during training to reduce overfitting.

7.2 Model Comparison

Table 7 shows the results in terms of precision, recall and macro F1-score for the classification task. We used the macro averaged F1 as a metric for the classification of offensive language accounting for the high-class imbalance.

We experiment with different models and used different features with BOW with TIFDF features for SVM model and we used the GloVe Twitter word embeddings for the other four models. After experimenting with different word embeddings Glove Twitter word embeddings gave the best or comparable results than the other word embeddings.

We will evaluate the models based on the measures explained in Section 6.3. We found that the best performing model is a CNN model, the second-best model is the BERT model. The CNN model has a learning rate of 4e-4 and a dropout of 0.31. And the BERT with a linear layer and batch normalisation has a learning rate of 3e-5.

If we would only use BERT without any other model on top, the recall of the offensive tweets is low, 0.52. This means that it is not good at detection offensive posts. It did, however, have a very high recall for the non-offensive tweets. This would mean that BERT would classify most of the tweets as non-offensive. BertLinear, a model which adds linear layers on top of BERT also had a very low recall for the offensive tweets, 0.48. The model BertNorm, the BERT model with a linear layer and batch normalisation gave the best performance in terms of the F1 score, and higher recall.

	NOT			**OFF**			
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**Macro-F1**
SVM	0.82	0.78	0.80	0.57	0.63	0.60	0.69
MLP	0.90	0.80	0.85	0.53	0.73	0.61	0.71
LSTM	0.89	0.82	0.85	0.60	0.73	0.66	0.74
CNN	0.91	0.87	0.89	0.72	0.79	0.75	**0.81**
BERT	0.92	0.86	0.89	0.62	0.78	0.69	0.78

Table 7: Evaluation Metrics

The performance of the CNN model in terms of the macro-F1 score is slightly higher than the BERT model. We could explain this because tweets are short and for a CNN model it is then easier to find certain key phrases. BERT does not perform as well as we would expect. This could be explained by the fact that the language used in tweets is very different from the language used for training BERT. If we take a look at the difference in performance between the non-offensive and offensive tweets, it shows that the performance of the non-offensive tweets is similar. The precision and recall are fairly similar. However, if we take a look at the offensive tweets we can see that there is a drop of precision while using BERT. Which means that if we get an offensive tweet it is often not correctly classified. Therefore, the BERT model is less accurate than the CNN model but can still be used to detect offensive language.

8 Conclusion

Detecting offensive language in social media is becoming more important. Because of the nature of language, it is difficult to achieve high performance when detecting it as such. Social media companies face a lot of challenges when detecting offensive language.

In this paper, we use different transfer learning methods to detect the offensive language in social media. The dataset we used was provided by (Zampieri et al., 2019) and consists of a three-layer annotation scheme. Our approach focused on the recent developments in NLP, which are based on language models. BERT is one of the new state-of-the-art language models. With this research, we wanted to see if BERT will give a big improvement over using word embeddings with a LSTM or CNN model. BERT can identify offensive language, however, it performs slightly worse than a CNN model.

Detecting offensive language is difficult. To tackle this problem, different state-of-the-art methods are used. BERT and the CNN have shown to perform the best. BERT might not perform as well as we would expect beforehand. Tweets are very short and different from where BERT was trained on. The dataset is very imbalanced, furthermore, the dataset is quite small. Adding additional data to this dataset could improve the performance.

We showed that transfer learning gives promising results. In reality, there does not exist a lot of labelled data because it requires a lot of expensive human labour. In that case, transfer learning is a good method to use for the task. Tuning the parameters and exploring other more SOTA language models could be explored, which could possibly lead to better performance.

Future Work
Offensive language contains a lot of emotions. Tweets also contain emojis. To use the emoji we replaced the emoji with their meaning. We could do more research on how to incorporate the emojis and if they really influence a lot whether something is offensive or not.

BERT is one of the best language models created, and it performs 'surprisingly well' on a lot of different datasets. It requires world knowledge and common sense reasoning. Only, there is little evidence that language models can learn such features of high-level natural language understanding. It could be that these methods are solving a task by learning the wrong thing, also called the Clever Hans effect. We could be sceptical about a near-human performance in high-level natural language understanding tasks.

We can expect that models will just become bigger and bigger. After BERT there were other models released such as the GPT-2, which is 24x the size of BERT. There is a lot of new research done in the area of NLP. Newer methods such as XLNet (Yang et al.,

2019), RoBERTa (Liu, Ott, Goyal, & Du, 2019) could be further explored.

The dataset is not really large. We could augment this dataset with other data. We could also add more additional features and explore this more. Besides that, more linguistic features could be added.

References

Arumugam, R., & Shanmugamani, R. (2018). *Hands-on natural language processing with python: A practical guide to applying deep learning architectures to your nlp applications*. Packt Publishing.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, *3*(Feb), 1137–1155.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Center, P. R. (2019). *Public Highly Critical of State of Political Discourse in the U.S.* Retrieved 2019-06-19, from `https://www.people-press.org/2019/06/19/public-highly-critical-of-state-of-political-discourse-in-the-u-s/`

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (p. 1724-–1734). Association for Computational Linguistics.

Clark, K., Khandelwal, U., Levy, O., & Manning, C. D. (2019). What Does BERT Look At? An Analysis of BERT's Attention. *arXiv preprint arXiv:1906.04341*.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*, 273–297.

Dadvar, M., Trieschnigg, D., Ordelman, R., & de Jong, F. (2013). Improving cyberbullying detection with user context. In *European Conference on Information Retrieval* (pp. 693–696). Springer.

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Proceedings of the Eleventh International Conference on Web and Social Media* (p. 512—515).

Deng, L., & Liu, Y. (2018). *Deep Learning in Natural Language Processing* (1st ed.). Springer.

Djuric, N., Zhou, J., Morris, R., Grbovic, M., Radosavljevic, V., & Bhamidipati, N. (2015). Hate speech detection with comment embeddings. In *Proceedings of the 24th international conference on world wide web* (pp. 29–30). Association for Computational Linguistics.

Eisner, B., Rocktäschel, T., Augenstein, I., Bosnjak, M., & Riedel, S. (2016). emoji2vec: Learning Emoji Representations from their Description. *arXiv preprint arXiv:1609.08359*.

Fletcher, T. (2008). *Support vector machines explained.*

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural computation*, *9*(8), 1735–1780.

Howard, J., & Ruder, S. (2018). Fine-tuned Language Models for Text Classification . *arXiv preprint arXiv:1801.06146*.

Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). Improving Word Representations via Global Context and Multiple Word Prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics* (pp. 873–882). Association for Computational Linguistics.

Jacob Devlin, & Chang, M.-W. (2018). *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing [blog post].* Retrieved 2019-08-15, from `http://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html`

Jing, K., & Xu, J. (2019). A Survey on Neural Network Language Models. *arXiv preprint arXiv:1906.03591*.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (p. 427—431). Association for Computational Linguistics.

Kamath, U., Liu, J., & Whitaker, J. (2019). *Deep learning for nlp and speech recognition*. Springer.

Kelly, M. (2019a). *France wants to fine Facebook over hate speech.* Retrieved 2019-07-19, from `https://www.theverge.com/2019/7/4/20682513/french-parliament-facebook-google-social-network-hate-speech-removal`

Kelly, M. (2019b). *Twitter will now hide — but not remove — harmful tweets from public figures [Blog post].* Retrieved 2019-08-01, from `https://www.theverge.com/2019/`

6/27/18761132/twitter-donald-trump-rules-violation-tweet-hide-remove-political-figures

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (p. 1746—1751). Association for Computational Linguistics.

Kulkarni, A., & Shivananda, A. (2019). *Deep learning for nlp: Unlocking text data with machine learning and deep learning using python*. Apress.

Kumar, R., Ojha, A. K., Malmasi, S., & Zampieri, M. (2018). Benchmarking aggression identification in social media. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)* (pp. 1–11).

Liu, Y., Ott, M., Goyal, N., & Du, J. (2019). Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Martin, S. (2019). *What Is Transfer Learning? | NVIDIA [Blog post].* Retrieved 2019-07-07, from https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/

Maryam Mooshin. (2019, March). *10 Social Media Statistics You Need to Know in 2019 [Infographic].* Retrieved 2019-08-28, from https://www.oberlo.com/blog/social-media-marketing-statistics

McMahan, B., & Rao, D. (2019). *Natural language processing with pytorch*. O'Reilly Media, Inc.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Černock, J., & Khudanpur, S. (2010, 1). Recurrent neural network based language model. In *Proceedings of the 11th annual conference of the international speech communication association* (pp. 1045–1048).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

Nguyen, M. (2019a). *Illustrated Guide to LSTM's and GRU's: A step by step explanation [Blog post].* Retrieved 2019-07-21, from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Nguyen, M. (2019b). *Illustrated Guide to Recurrent Neural Networks - Towards Data Science [Blog post].* Retrieved 2019-07-21, from https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016). Abusive language detection in online user content. In *Proceedings of the 25th international conference on world wide web* (pp. 145–153). International World Wide Web Conferences Steering Committee.

Osinga, D. (2018). *Deep learning cookbook*. O'Reilly Media, Inc.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of NAACL* (pp. 2227–2237). Association for Computational Linguistics.

Pitsilis, G. K., Ramampiaro, H., & Langseth, H. (2018). Detecting offensive language in tweets using deep learning. *arXiv preprint arXiv:1801.04433*.

Ruder, S. (2019). *Neural Transfer Learning for Natural Language Processing* (Ph.D. Thesis). National University of Ireland, Galway.

Russakovsky, O., Deng, J., Su, H., Krause, J., & Satheesh, S. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252.

Sebastian Ruder. (2018). *NLP's ImageNet moment has arrived [Blog post].* Retrieved 2019-08-01, from https://thegradient.pub/nlp-imagenet/

Segaran, T., & Hammerbacher, J. (Eds.). (2009). *Beautiful Data: The Stories Behind Elegant Data Solutions*. Beijing: O'Reilly. Retrieved from https://www.safaribooksonline.com/library/view/beautiful-data/9780596801656/

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*.

Singh, A., Blanco, E., & Jin, W. (2019). Incorporating Emoji Descriptions Improves Tweet Classification. In *Proceedings of NAACL-HLT 2019* (pp. 2096–2101). Association for Computational Linguistics.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems 25* (pp. 2951–2959). Curran Associates, Inc.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In *Interspeech* (p. 194—197).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., & Reed, S. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1–9).

Thompson, N. (2017). Instagram Launches An AI System to Blast Away Nasty Comments [Blog post]. *Wired*. Retrieved 2019-07-19, from `https://www.wired.com/story/instagram-launches-ai-system-to-blast-nasty-comments/`

Tixier, A. J.-P. (2018). Notes on Deep Learning for NLP. , *arXiv preprint arXiv:1808.09772*.

Twitter, I. (2019). *Q2 2019 letter to shareholders [PDF file].* Retrieved 2019-07-26, from `https://s22.q4cdn.com/826641620/files/doc_financials/2019/q2/Q2-2019-Shareholder-Letter.pdf`

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., & Gomez, A. N. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (p. 6000—6010).

Wang, W., Chen, L., Thirunarayan, K., & Sheth, A. P. (2014). Cursing in English on Twitter. In *Proceedings of the 17th ACM conference on Computer Supported Cooperative Work & Social Computing* (pp. 415–425).

Waseem, Z. (2016). Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In *Proceedings of 2016 EMNLP Workshop on Natural Language Processing and Computational Social Science* (pp. 138–142).

Waseem, Z., Davidson, T., Warmsley, D., & Weber, I. (2017). Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online* (p. 78—84).

Wiegand, M., Siegel, M., & Ruppenhofer, J. (2018). Overview of the germeval 2018

shared task on the identification of offensive language. In *Proceedings of GermEval.*

Wong, Q. (2019). *Facebook fined $2.3 million for violating Germany's hate speech law [Blog post].* Retrieved 2019-07-19, from `https://www.cnet.com/news/facebook-fined-2-3-million-for-violating-germanys-hate-speech-law/`

Xu, J.-M., Jun, K.-S., Zhu, X., & Bellmore, A. (2012). Learning from bullying traces in social media. In *2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 656–666). Association for Computational Linguistics.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., & Kumar, R. (2019). Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL-HLT 2019* (p. 1415—1420). Association for Computational Linguistics.