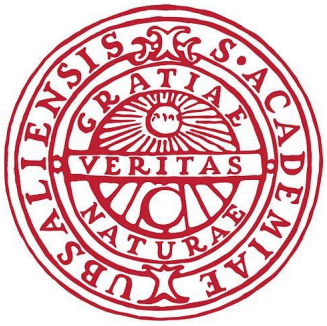

UPPSALA
UNIVERSITET

Classifying Hate Speech using Fine-tuned Language Models

By Erik Brorson

Department of Statistics

Uppsala University

Supervisor: Yukai Yang

2018

Abstract

Given the explosion in the size of social media, the amount of hate speech is also growing. To efficiently combat this issue we need reliable and scalable machine learning models. Current solutions rely on crowdsourced datasets that are limited in size, or using training data from self-identified hateful communities, that lacks specificity. In this thesis we introduce a novel semi-supervised modelling strategy. It is first trained on the freely available data from the hateful communities and then fine-tuned to classify hateful tweets from crowdsourced annotated datasets. We show that our model reach state of the art performance with minimal hyper-parameter tuning.

Contents

1 Introduction

We spend more time than ever on social media platforms. These platforms promotes increasing emphasis on users to participate, resulting in increasing amounts of content produced by the users. Much of this content is good, but not everything. We've lately been made aware of the dangers when these platforms are used with nefarious intent. In April 2018 a Canadian man killed 10 people in Toronto after making a facebook post pledging allegiance to the *Incel Rebellion*, an online movement and community which unites men in their in-ability to convince women to sleep with them (Beauchamp 2018). Incels, involuntary-celibacy, have had an active presence online on sites such as reddit.com spreading hateful content aimed towards women. A recent study has shown that the amount of hateful content towards migrants can predict the amount of reported violence against this group (Müller and Schwarz 2017). Hate speech online is wide spread and can be found in sectors such as gaming (Consalvo 2012), Wikipedia talk pages (Wulczyn, Thain, and Dixon 2016), reddit (Moreno, Pao, and Ohanian 2015), and twitter (Kwok and Wang 2013; Waseem and Hovy 2016; Davidson et al. 2017). A survey from the Pew Research Center showed that 73 percent of online internet users had witnessed online harassment, and 40 percent had been personally targeted (Duggan et al. 2014). This issue has gained the attention of governments such as the UK (Home Office 2016), and Germany who will fine social media companies up to €50 million if hate speech is not removed within 24 hours (The Guardian 2017). The European Union has, together with several major technology companies, released a Code of Conduct on online hate speech (European Commission 2016). In order to combat online hate we need to find scalable and transparent solutions.

The benefits of using automated techniques to detect hate speech are obvious. This explains the substantial amount of attention this problem has gained in recent years. A good overview of the current state of research can be found in Schmidt and Wiegand (2017). Current approaches all have their limits. Hate Speech cannot be understood as a monolithic problem, some empirical evidence of this can be found in Kwok and Wang (2013) who show that the ethnic background decides what one might consider hateful. Davidson et al. (2017) also find suggestions that cases of misogyny are often annotated as being offensive when cases of homophobia and racism is seen as hate speech. There seems to be an social dimensionality to ones perception of hate speech. It would not be surprising if it also depends on locality and culture (compare the usage of the n-word among white Americans and black Americans). Current attempts often use self sampled datasets that in most cases have been annotated using some crowd sourcing platform such as CrowdFlower or Amazon MTurk (Davidson et al. 2017; Waseem and Hovy 2016;

Kwok and Wang 2013; Wulczyn, Thain, and Dixon 2016). These datasets are in many cases limited in size, and annotated with unique definitions of hate speech. There are attempts to move past this limitation using data from alternate sources (Saleem et al. 2017) or training a classifier on more than one dataset (Waseem, Thorne, and Bingel 2018).

Annotating hate speech is an expensive and time consuming process, and machine learning models are limited by the size of the dataset. In theory, a very large and reliably annotated dataset would most likely produce powerful and reliable classifiers, however such datasets are very hard to produce. Rather than relying on one large dataset for the majority of the hate speech classification we propose a solution. Building on the work of Howard and Ruder (2018) we introduce *Fine-Tuned Language Model for Hate Speech Classification*. This is a semi-supervised text classifier that in three steps trains and fine-tunes a general language model to produce classifications. We expect this model to be more robust towards over-fitting if trained correctly and to reach acceptable performance using small specialised datasets. In this thesis we show that, with minimal hyper-parameter tuning, we can reach close to state of the art result or even beat it using two previously studied datasets. We use the same datasets as in Waseem, Thorne, and Bingel (2018).

The rest of this thesis is structured as follows: first we present earlier research and discuss definitions of hate speech used in earlier attempts to build hate speech classifiers. The following section we introduce a natural language transfer technique and discuss our implementation. After that we describe our data, and also our experimental results. We end with an analysis of the errors of our model and conclude with suggestions on further research. For an overview of our final classification model, see figure 3.

2 Earlier work

In order to train a model to classify hate speech we need to have a clear understanding of what it is. Human annotators are normally used to create training datasets that are used to evaluate models and build production models. But collecting and annotating hate speech corpora using this method leads to low agreement among annotators and as a result low reliability. In order to collect and reliably annotate cases of hate speech we need to have clear definitions and multiple annotators. (Ross et al. 2017; Waseem 2016)

In the literature the definition of hate speech varies. Saleem et al. (2017) talks about hateful speech and defines it as *speech which contains an expression of hatred on the part of the speaker/author against a person or people, based on their group identity.* Badjatiya et al. (2017) define hateful tweets as those containing abusive speech targeted against individuals, such as cyber-bullying or hate directed towards public figures, or on basis on group belonging, for example LGBTQ or gender. The common denominator is hate aimed against something or someone, often with incitement to discrimination, hostility or violence (Taylor, Peignon, and Chen 2017). Without having a clear definition it is impossible for the annotators to do a good job, resulting in low agreement among human annotators. Often this inter-annotator agreement is measured with the κ statistic, proposed by Fleiss (1979). This statistic is bounded by 0 and 1, and a value of 1 corresponds to a perfect agreement among the annotators.

Currently, attempts to classify hate speech can roughly be divided in two groups. The first which relies on crowdsourced datasets, whose annotation process is highly dependent on the definition of hate speech. The second tries to do away with the reliance of definitions by either using a transfer learning model (Waseem, Thorne, and Bingel 2018) or changing the objective by sampling from self-defined hateful communities.

2.1 Attempts using Crowdsourced Datasets

Kwok and Wang (2013) focuses on a narrow definition of hate speech against blacks. They first collected a sample of 100 tweets and let 3 students of similar age and gender but different ethnicity label each tweet as offensive or not. The three students did not annotate this sample reliably ($\kappa = 0.33$), Kwok and Wang concludes that this is indicative that it would be hard for a computer to reliably classify tweets. They annotate a sample of 24582 tweets, with tweets coming from self-identified racist accounts. After a manual review of the individual tweets they label those that were found to be racist and trained a Naive Bayes classifier to distinguish between the racist and the non-racist tweets. They achieve a mean accuracy of 77 percent. This methodology is highly dependent on manual feature engineering. The authors conclude that many of the tweets are racist because

they contain certain words, such as alternative forms of the n-word. Other tweets are however deemed offensive without the use of obvious terms, for example the tweet *Why did Obama's great granddaddy cross the road? Because my great granddaddy tugged his neck chain in that direction.* The authors of the study conclude that in order to classify racism on twitter additional techniques must be used, such as bigrams (using two word tokens instead of just one, for example *New York* can be considered a bigram), sentiment analysis etc.

Separating hate speech from non-hate speech is achievable using linear classifiers with lexical features. For many people, hate speech is somewhat defined by the use of particular words such as homophobic or racial slurs. However, there are cases where a comment or tweet contains any of these words without the result constituting hate speech. This is the focus of Davidson et al. (2017). They begin with a lexicon of hate speech words and phrases defined by internet users, *Hatebase.org*. A total of 85.4 million tweets is gathered from 33,458 twitter users using the Twitter API. A sample of 25 thousand tweets is drawn and annotated by CrowdFlower users[1]. The tweets are labelled as either hate speech, offensive speech, or neither offensive speech nor hate speech by at least 3 annotators. The label is chosen from a majority decision. The inter-annotator agreement has a κ score of 0.92. By using a combination of TF-IDF weighted uni-grams, bi-grams, and tri-grams, sentiment scores as well as count indicators for hashtags, mentions, retweets, and URLs, they reach a F1 score of 0.90 when training on and evaluating on the full data. They used a logistic regression model with L2 regularisation and performed hyper-parameter tuning using cross-validation. Note that this data is used in the upcoming experiments. For a more in-depth analysis of sampling and data, see section 4.

Waseem and Hovy (2016) contributes with a dataset consisting of 16,000 annotated tweets that has been labelled as either sexist, racist, or neither. The definition used in the paper comprise an 11 point list (can be found in section 4.1) used to define hate speech. What is different in this definition is that it includes idiosyncratic features of Twitter such as the point *shows support of problematic hash tags. E.g. #BanIslam, #whoriental, #whitegenocide*. The inter-annotator agreement for this dataset is 0.84. A whole 85 percent of the disagreements among annotators occur in examples of sexism, with 98 percent of all reviewer changes being set to neither sexist nor racist. The remaining are set to racist. They conclude that most cases stems from the context, or rather lack thereof. In an effort to evaluate the importance of different features, they use a base model using n-grams. They then experiment with adding the additional features length of of tweet, gender of twitter user, and the locality of the twitter user. They find that gender is the

[1]CrowdFlower is a platform where you can outsource the annotation process, similar to Amazon Mechanical Turk. CrowdFlower has since been renamed figure eight and can be found on https://www.figure-eight.com/

only feature that improves the overall f1-score, they report a score of 73.89. To evaluate the models they used a logistic regression classifier and ran a 10-fold cross-validation.

Using n-grams and lexical features usually involves representing a document as a sparse vector with non-zero elements representing the count of some n-gram, or length of the document. This approach has obvious drawbacks as it throws away the structure of the text. This representation is commonly know as *bag-of-words* (BoW), it can be though of as taking the tokens, or words, and putting them in a bag without taking the order into consideration. In this representation the sentences *I love to hate you* and *I hate to love you* would be considered equivalent even though they have different semantic meanings. An alternative to the BoW representation is to view the sentence as a sequence of words, this is commonly done by representing each word as a dense vector trained on large amounts of data[2]. This approach is used by Badjatiya et al. (2017) who experiments on the dataset provided by Waseem and Hovy 2016, described above. They manage to achieve an average F1 score of 0.93 using a combination of randomly initialised word embeddings, an LSTM neural network (described in further detail in the upcoming section 3) and a gradient boosting classifier.

2.2 Other attempts

In an attempt to bridge the differences between different groups of annotators and different definitions Waseem, Thorne, and Bingel (2018) run experiments using the two earlier mentioned datasets from Waseem and Hovy (2016) / Waseem (2016), and Davidson et al. (2017) using a multi task framework. This is a transfer learning technique that in essence trains a model by solving two separate tasks simultaneously. A hand-wavy explanation goes like this: consider two tasks, an auxiliary task and a primary task. By solving these tasks simultaneously we hope to utilize similarities between the two tasks, in theory resulting in better generalisations. They switch between using the two datasets as auxiliary and primary datasets and finds that the multi-tasking model outperforms the model that is solely trained on a single dataset.

So far, we have only discussed cases where hate speech is defined using a supervised learning framework, complete with annotated datasets. This is however, not the only approach that can be found in literature. Saleem et al. (2017) treats the problem of hate speech differently. They propose a model of language that is defined by the community. Using historic data they sample comments from self-defined hateful subreddits (user run sub-communities on the popular link-sharing site reddit) and subreddits that are polar

[2]A few examples of these pre-trained dense vectors, also known as word embeddings, are fastText (Bojanowski et al. 2016), Word2Vec (Mikolov et al. 2013), and GloVE (Pennington, Socher, and Manning 2014)

opposites. One example is the subreddit *fatpeoplehate*, banned in 2015, that is defined by hate towards overweight people. The opposite subreddit in this case is *loseit* which is a community for people who are trying to lose weight. They show that these two communities share vocabulary and are discussing the same topics. They then train a classifier to explore the separability of the two communities. With accuracies around 80 percent, it is clear that the we can distinguish between the languages in the two hate groups. This data can be acquired with low cost, but the classifier will at best be able to indicate which community a comment is similar to. But to classify hate speech we need to be more specific.

This review is not comprehensive. A more thorough discussion on previous work can be found in Schmidt and Wiegand (2017). Considering previous work it becomes apparent that this problem is often framed as a supervised learning problem, with annotated datasets. These models are limited by the size of the dataset and often use features directly derived from the data itself, resulting in overfitting to the specific sample. Any claims of out-of-sample performance is therefore invalid as the out-of-sample data was used to create the models itself. In order to build a model with generalisable performance one needs to create a dataset so big that it encompasses large parts of the possible hateful language space: something that is not only expensive but also difficult to encompass with different definitions. A plausible solution would be to use a semi-supervised method as described in Saleem et al. (2017), but this lacks the precision to accurately distinguish text possibly stemming from a hateful community, from the actual comments. Instead we propose a model that draws from both approaches, by using a technique called transfer learning we can extract knowledge from easily available un-annotated data, subreddits, and use this to create predictions on annotated datasets.

3 Fine-tuned Language Model for Hate Speech Classification

Most machine learning techniques works well under a common assumption: that the train data and test data are drawn from the same feature space and the same distribution. In most cases when this is not the case, we need to sample fresh training data and retrain the model from scratch. This is often an expensive and tricky task, as in the case of labelling hate speech.

Just a quick note on some machine learning taxonomy, we usually divide the methodologies into three groups (excluding a fourth type of algorithms called reinforcement learning): supervised learning, unsupervised learning, and semi-supervised learning. Supervised learning are the cases when you observe some kind of label or dependent variable for each observation. We then try to find a function that maps the features to the label or dependent variable. Examples of techniques that falls into this category is regression based techniques, support vector machines, random forests, etc. Unsupervised learning deals with observations that has no label. Examples of unsupervised techniques are cluster analysis, principal components analysis, auto encoders, etc. A semi-supervised approach can loosely be understood as a technique that includes elements of both. In this thesis we are trying to build a semi-supervised hate speech classifier. We start off by building an unsupervised model that can learn the structure of the data. We then fine-tune this model using an annotated dataset. Finally we make changes to our model and we turn it into a supervised machine learning model that is trained to create predictions. By fine-tuning we mean updating weights using a new dataset to an already trained model.

As discussed in the previous section, current hate speech classifiers are lacking in a number of ways. Hate speech is hard to annotate, and classifiers trained using supervised learning models depends heavily on the annotation process. Machine learning models are also limited by the size of the dataset. This is one of the problems we want avoid by introducing the unsupervised element to our approach. This should give us acceptable performance even with small datasets. We based our approach on the work by Howard and Ruder (2018). They manage to obtain state-of-the-art performance on a five widely studied datasets. Their approach uses an unsupervised neural language model that is trained on some large corpus of documents. They then fine-tune their model and change the final neural network to produce classifications. Reusing their publicly available model weights is possible. But given the nature of hate speech, we are better off using a domain specific language model. Note that this is not the only possible semi-supervised text classifier, Dai and Le (2015) describes an alternative unsupervised part that consists of

sequential auto-encoder, which is a neural network where the input is the same as the output, rather than a neural language model.

The rest of this section explains the modelling steps in detail. It comprise three parts, the first is the neural language model; which is the unsupervised part of our approach. The second part is the fine-tuning where we continue to train the model that was obtained in step one using the annotated dataset. The third part consists of switching the final layers of our network to produce classifications. The whole model is visualised in figure 3. We report on our hyper-parameters in the results section.

3.1 Neural Language Model

A neural language model predicts the next words in a sequence. As an example, if we have the sentence *the quick brown fox jumped over the lazy dog* we would input the sequence from *the* to *lazy*, and predict the sequence from *quick* to *dog*. At each time point, we know the current, and the previous words. As an example, at time step 3 in figure 1 we input the word *brown* and know that the previous words were *quick* and *the*. This information is used to predict the word *fox*. What we are estimating is the probability distribution for each word in the dictionary, and we make a prediction of the next words using a maximum likelihood principle, i.e. the word with the highest probability mass.

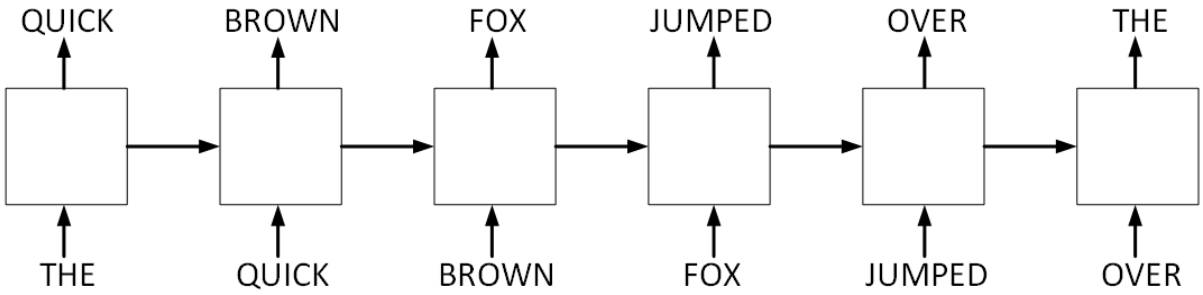

Figure 1: Illustration of a language model where we try to predict the next word in a sequence, given the previous words.

Each box in figure 1 is called a cell and each arrow represents some kind of dependence. If we consider the probability of the word at time step 3 being *fox* we might think of this like,

$$Pr(Y^{<3>} = \text{fox} | X^{<3>}, X^{<2>}, X^{<1>}, \boldsymbol{W})$$

The task we want to solve is to find a set of weights $\boldsymbol{W}$ so that the estimated probability distribution assign a high probability mass to observed words, and low to the rest. We model this probability distribution by using a multi-layer recurrent neural network.

In the next section, we will introduce this family of networks, recurrent neural networks. After that we discuss some implementation details of our model

3.1.1 LSTM and recurrent neural networks

Recurrent neural networks are a family of nets that are for processing of sequential or longitudinal data. The main feature is that they can process variable length sequences without altering the amount of parameters in the model, this comes down to something called weight sharing. The importance of this can be found in an example given by Goodfellow, Bengio, and Courville (2016), consider the two sentences "I went to Nepal in 2009" and "In 2009, I went to Nepal". If we want to have a machine learning model that outputs the year in which I went to Nepal, we would like identify 2009 as the interesting information regardless if it occurs in as the first word, or the last. In theory this can be performed using a regular network that inputs a fixed length sentence but then we would have to learn the whole structure of language at each position in the sequence.

We input a sentence to our model, represented by integers which corresponds to a position in our dictionary, a set of the n most common words. These words are then represented as dense vectors, commonly known as word vectors. These dense vectors are stacked in a matrix $\mathbf{U}$ that, in figure 1 would contain unique vectors for the words like *the, quick, brown* etc. Once trained, these vectors capture meaningful semantic meanings, similar words' vectors tend to be grouper closer together.

For the purpose of this thesis we consider a recurrent neural network that at each time step inputs the element a sequence and makes an output that is a function of the current input and previous inputs. This is illustrated in 2 where we see the net with its recurrent connection to the left and unrolled to the right. $X^{<t>}$ is the input at position t and $a^{<t>}$ is its output. Note that these might be vectors and matrices as well as scalars.

One of the main issues with this simple formulation of the recurrent network is its disability to model long time dependencies – consider if we want to model opening and closing a parenthesis. More modern architectures has been proposed, most notably the LSTM (Hochreiter and Schmidhuber 1997) and GRU (Cho et al. 2014). In the following experiments we will focus on the former, the *Long Short-Term Memory Network* or LSTM.

The most basic building block of an LSTM neural network is the LSTM cell itself. At each time step, the cell receives input data, $X^{<t>}$ and the previous output vector $\mathbf{a}^{<t-1>}$, and the previous cell state vector $\mathbf{c}^{<t-1>}$. Much of the power of this architecture lies in the gates, denoted as Γ. These are vectors whose values are bounded by 0 and 1. These gates controls the amount of information that flows between time steps, mimicking memory (hence the name). Below are equations that defines a single LSTM cell.

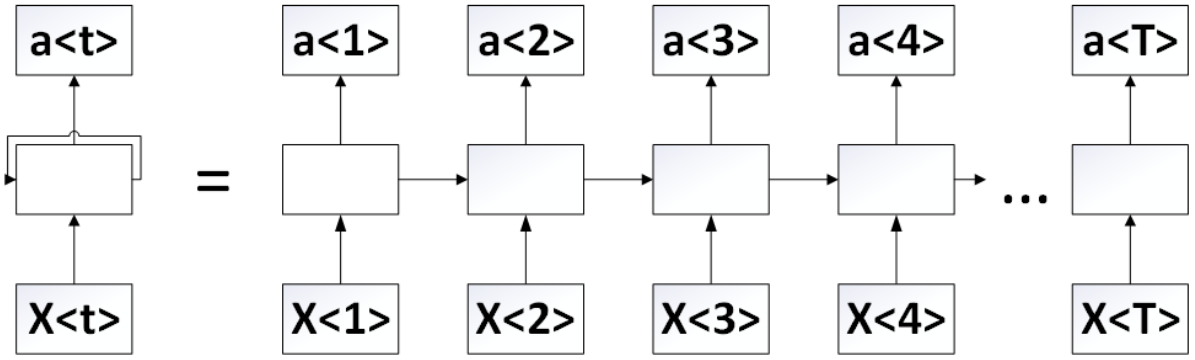

Figure 2: RNN illustration

$$\Gamma_f^{<t>} = \sigma\left(\mathbf{W}_{fa}\mathbf{a}^{<t-1>} + \mathbf{W}_{fx}\mathbf{X}^{<t>} + \mathbf{b_f}\right) \tag{1}$$

$$\Gamma_u^{<t>} = \sigma\left(\mathbf{W}_{ua}\mathbf{a}^{<t-1>} + \mathbf{W}_{ux}\mathbf{X}^{<t>} + \mathbf{b_u}\right) \tag{2}$$

$$\tilde{\mathbf{c}}^{<t>} = \tanh\left(\mathbf{W}_{ca}\mathbf{a}^{<t-1>} + \mathbf{W}_{cx}\mathbf{X}^{<t>} + \mathbf{b_c}\right) \tag{3}$$

$$\mathbf{c}^{<t>} = \Gamma_f^{<t>} \times \mathbf{c}^{<t-1>} + \Gamma_u^{<t>} \times \tilde{\mathbf{c}}^{<t>} \tag{4}$$

$$\Gamma_o^{<t>} = \sigma\left(\mathbf{W}_{oa}\mathbf{a}^{<t-1>} + \mathbf{W}_{ox}\mathbf{X}^{<t>}) + \mathbf{b_o}\right) \tag{5}$$

$$\mathbf{a}^{<t>} = \Gamma_o^{<t>} \tanh(\mathbf{c}^{<t>}) \tag{6}$$

where Γ_f is the forget gate, Γ_u is the update gate, and Γ_o is the output gate. The two most important parts of the architecture are the cell state $\mathbf{c}^{<\mathbf{t}>}$ and the output $\mathbf{a}^{<\mathbf{t}>}$. In step (1) above we compute the forget gate Γ_f as a function of the previous output, $\mathbf{a}^{<\mathbf{t}>}$ and the current input, $\mathbf{X}^{<\mathbf{t}>}$ which are matrix multiplied with two weight matrices. This weighted sum is then put through a sigmoid function, $\sigma(a) = \frac{1}{1+e^{-a}}$, which is $\sigma \in (0, 1)\ \forall a$. This matrix is later element-wise multiplied in equation (4) to decide which information we keep from the previous cell state, hence the name *forget* gate. The second gate is like the forget gate calculated as a weighted sum of the previous output and the current input. In equation (3) we have what is called the candidate cell-state, $\tilde{\mathbf{c}}^{<\mathbf{t}>}$ which is calculated by matrix multiplying the previous output, and input with weights and put them through another activation function, the hyperbolic tangent function. This function is defined as $\arctan(a) = \frac{\sinh(a)}{\cosh(a)}$ and $\arctan(a) \in (-1, 1)\ \forall a$. In equation (4) we start to see how everything comes together. We calculate the current cell state $\mathbf{c}^{<\mathbf{t}>}$ by doing two element wise multiplications and then summing them. The first part is using the forget gate $\Gamma_f^{<t>}$ calculated in (1) and the previous cell state. The forget gate decides *what to keep* from the previous state. The second part is using the update gate $\Gamma_u^{<t>}$ to update the current state by using the candidate state. Operations (1) to (4) calculates the current state as a function of the previous cell state, the previous output and the

current input. The information in the current state is governed by the values of the two gates. Operation (5) and (6) is straight forward, the third gate is calculated like the two earlier gates. In the final operation where we decide the output of the current cell, we multiply the tanh of the cell state with the output gate.

At each time step we project our, outputs also called, activation vector $\mathbf{a}^{<\mathbf{t}>}$ by multiplying it with a matrix $\mathbf{V}$, which is then squished by using a softmax activation layer so that,

$$\text{softmax}(\mathbf{a}^{<t>T}\mathbf{V}) = \hat{\mathbf{y}}^{<t>}$$

where element j of $\hat{\mathbf{y}}^{<\mathbf{t}>}$ is $\frac{e^{\mathbf{a}^{<\mathbf{t}>}\mathbf{T}\mathbf{v}_j}}{\sum_{i=1}^{k} e^{\mathbf{a}^{<\mathbf{t}>}\mathbf{T}\mathbf{v}_i}}$. The effect of the softmax function gives us a vector of values between 0 and 1 where all elements sum to 1. This is a valid estimation of the probability distribution over the dictionary for each position in the sequence. The exponential element can intuitively be understood to force a few elements to be much larger than others. We then end up with a matrix $\hat{\mathbf{Y}} = \left[\hat{\mathbf{y}}^{<1>}, \hat{\mathbf{y}}^{<2>}, \ldots, \hat{\mathbf{y}}^{<N>}\right]$ which is of dimension (vocabulary size, timesteps) where each column is the estimated probability distribution of words given context.

One very common approach to avoid overfitting when using neural networks is dropout (Srivastava et al. 2014) which before each epoch randomly puts some elements to equal zero. We apply dropout to the inputs of the LSTM layers to avoid overfitting. We also apply something called dropconnect which is dropout applied to the weight matrix rather than the activations, this is also know as weightdrop (Merity, Keskar, and Socher 2017; Wan et al. 2013). We apply dropconnect to the hidden-to-hidden weight matrices, $[\mathbf{W}_{fa}, \mathbf{W}_{ua}, \mathbf{W}_{ca}, \mathbf{W}_{oa}]$ in the LSTM layer.

We trained our general language model using the common stochastic gradient descent optimizer with learning rate decay. Consider the general parameter matrix θ, which will be updated at step k as:

$$\boldsymbol{\theta_k} = \boldsymbol{\theta_{k-1}} - \eta \nabla J(\boldsymbol{\theta})$$

where $\nabla J(\boldsymbol{\theta})$ is the gradient of the loss function with respect to the parameter matrix and η is the learning rate. After each update we shrink η exponentially with some predefined amount, so that

$$\eta_u = \eta_{u-1}(1 + \text{decay} \times u)^{-1}$$

where η_u is the learning rate at update u and decay is the decay hyper-parameters.

3.1.2 Weight tying

Each token in the input sequence is represented as a vector that is zero everywhere except at the index which represents that unique word, a so-called one-hot encoding. A sequence is an ordered set of tokens, which is represented as matrix that is of dimension (length, number of tokens). The first step in the language model projects this sequence matrix to a dense representation using a word embedding matrix $\mathbf{U}$. The dense representation is of dimension (length, embedding size). The model then pass this matrix through the recurrent net and produces a matrix of activations $\mathbf{A} = [\mathbf{a}^{<1>}, \mathbf{a}^{<2>}, \ldots, \mathbf{a}^{<n>}]$. $\mathbf{A}$ is then projected to produce matrix $\mathbf{Z}$ by multiplying $\mathbf{A}$ with a second matrix $\mathbf{V}$. $\mathbf{Z}$ has the same dimension as our first one-hot encoded sequence matrix and is squished to model the probability distribution over each token in the sequence. The matrices $\mathbf{U}$ and $\mathbf{V}$ has the same dimensions, and both can be considered valid word embeddings (dense representation of words, i.e we represent each word or token as a vector). Recent work on language modelling using neural networks has proposed to *tie* these matrices together (so that $\hat{\mathbf{U}} = \hat{\mathbf{V}}$) and has shown that this improves the performance (Press and Wolf 2016; Inan, Khosravi, and Socher 2016).

In our experiments we tie the these weights together. This has the additional benefit of cutting the number of parameters in our model used for the word embeddings by half.

3.1.3 Bidirectional language model

We train a two language models, one from each *direction*. That means that we train one model from start to end, and one from end to start. This means that we train and fine-tune two language models. Finally these models will be put concatenated, this is explained more in an upcoming section.

3.2 Fine-tuning the Language Model using the Classification Corpus

Each dataset contains some unique features that we cannot learn from the big unlabelled dataset. Therefore we fine-tune the language model on the sentences in our classification dataset. In order to avoid destroying the ingrained knowledge in our model, we train for two epochs per layer for the Waseem and Hovy, gradually unfreezing layer after the epochs (explained below) and two epochs per layer in the Davidson et al. case.

Fine-tuning is one of the most important aspects of transfer learning as it is in this step that the transferring is happening. One risk of doing this is called *catastrophic interference*, or *catastrophic forgetting*, which is the tendency of neural networks to forget previously learned knowledge when faces with a new task. We understand this concept in

the framework of fine-tuning language models as no longer having the knowledge learned in the general training.

In order to combat this problem we gradually unfreeze the layers on our network (which is also done by Howard and Ruder 2018). We train our network for a predefined number of epochs per layer whilst gradually unfreezing them one-by-one. By freezing we simply mean we stop updating the weights of that specific layer.

Like Howard and Ruder we used a trick when performing the fine-tuning called cosine annealing of the learning rate where we change it after each update according to,

$$\eta = \eta_{min} + (\eta_{max} - \eta_{min}) \left(1 + \cos \frac{t}{T}\pi\right)$$

where η is the learning rate and η_{max}, η_{min} are hyper-parameters and T is the maximum updates for one epoch, and t is the current update. We train for using cosine annealing for two epochs, one with increasing learning rate (only difference to the expression above is a minus instead of the plus in the second part with the cosine).

3.3 Training the Classifier

The outputs of the final recurrent layer are fed into a block of various layers that outputs a prediction. The first part of the linear block is used the extract useful information that can be used to distinguish between our classed. We do this in a similar way as described in (Howard and Ruder 2018). We take the outputs from the third LSTM layer, stacked in tensor

$$\boldsymbol{A} = \left[\boldsymbol{a}^{<1>}, \boldsymbol{a}^{<2>}, \ldots, \boldsymbol{a}^{<N>}\right]$$

where $\boldsymbol{a}^{<i>}$ is the output at position i in the sequence $i = 1, 2, \ldots N$. The first two operations we use are called *global max pooling* and *global average pooling*. As in the name, max pooling takes the max along the temporal dimension, so if $\boldsymbol{A}$ is of dimension $(\text{batch size}, \text{time}, \text{number of units})$, max pooling outputs a matrix of dimensions (batch size, number of units). Average pooling is the same as max-pooling but takes the average rather than the max. Finally we also flatten our the output of our final LSTM layer. This is done by stacking tensor $\boldsymbol{A}$ so that it has the dimension $(\text{batch size}, \text{time} \times \text{number if units})$. These three matrices are concatenated and fed into a dense layer with a ReLU activation. The rest of the block is made up of a batch normalisation layer (Ioffe and Szegedy 2015) and dropout (Srivastava et al. 2014). Finally we feed the outputs into a final dense layer with a softmax-activation which produces the predictions.

Every weight in the network gets updated in the final step. In order to reduce risk

of catastrophic forgetting we use different learning rates for different layers. The code implementation used in the experiments multiplies the learning rate by a predefined factor that can be set layer by layer. In our experiments we lower the learning rate based on the order of the layer. We used the Adam optimizer (Kingma and Ba 2014) in this step.

3.3.1 Making a unidirectional model bidirectional

So far we have only discussed the modelling from a single-model perspective. But since we train two neural language models, one from each direction, we need a way to ensemble these models. This is what we are discussing in this section.

The simplest sampling technique would be the one used in Howard and Ruder (2018) where they train each model separately and then simply average the predictions. This is easy to implement and does not require any extra memory. Another idea would be to create the predictions of both models and then use these as the features for some other machine learning algorithm such as a support vector machine or logistic regression. This idea is slightly more involved than the one before as we need to consider a whole new set of hyper-parameters. If we however want to maximize performance by creating an ensemble with more than the model discussed here this is a viable option. We would however advice to pay some attention to how the hyper-parameters of this ensemble model are tuned as it is easy to cause data leakage between splits.

The approach we used in our experiments is however slightly different. By removing the final classification layers in the linear block of the two models we concatenate the outputs of the final dense layers, creating a single bidirectional model. This has the nice property that we don't have to consider two models in parallel but can focus on a single, big model. However, as we are effectively doubling the amount of parameters in our model, we need to have a large memory in our GPU to make training feasible. In our experiments we went from 30 million to 60 million. The choice of ensembling technique most likely plays little role in the grand scheme of things.

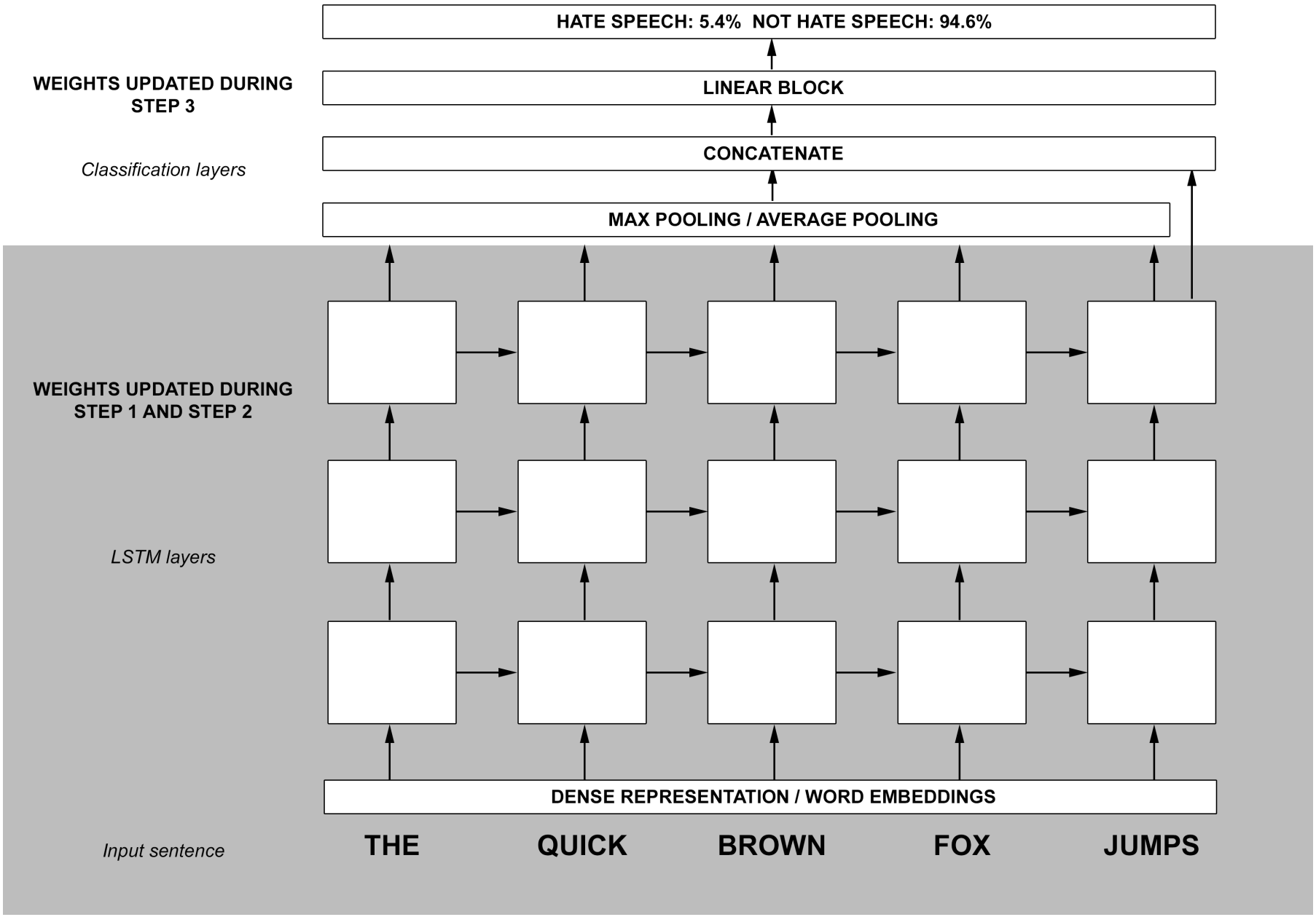

Figure 3: Schematic of final classification model. The lowest level represents an input sentence that then gets turned into a sequence of word embeddings. These are fed into the three LSTM-layers. The output of the third LSTM layer feeds into both an average pooling operator and an max pooling operator. These are concatenated with the final raw output of the final output. They are then fed into a linear block who in turn feeds into the classification layer that outputs probabilities. The grey part is trained in step 1 and 2, the white part is then added to the end of LSTM-layers, replacing a layer that was used to create probability distributions for the neural language model. The weights in the white are only trained during the final step.

4 Data

4.1 Sampling

4.1.1 Waseem and Davidson

Waseem and Hovy (2016) sampled 16,914 tweets out of which 3,383 were labelled as sexist and 1,972 were labelled as racist. The data was collected during a period of two months and collected tweets that fulfilled any of the 11 criteria. This means that 20 percent of the tweets in the dataset are labelled as sexist and almost 12 percent are labelled as racist. These are (cited from their paper):

1. Uses a sexist or racial slur
2. Attacks a minority
3. Seeks to silence a minority
4. Criticizes a minority (without a well founded speech or violent crime)
5. Promotes, bot does not directly use, hate speech or violent crime
6. Criticizes a minority and uses a straw man argument
7. Blatantly misrepresents truth or seeks to distort views on a minority with unfounded claims.
8. Shows support of problematic hash tags. E.g. *#BanIslam*, *#whoriental*, *#whitegenocide*
9. Negatively stereotypes a minority
10. Defends xenophobia or sexism
11. Contains a screen name that is offensive, as per previous criteria, the tweet is ambiguous (at best), and the tweet is on a topic that satisfies any of the above criteria.

The inter-rater agreement is 0.84. Waseem (2016) extended this dataset with about 4000 observations using the same criteria. Their annotation process was done using a combination of expert and amateur annotators. Among the amateur annotators the inter-rater agreement is 0.57, among the expert annotators the agreement is 0.34 if we would label according to a majority vote and 0.7 if we would label according to a full agreement.

Subreddit	Number of comments
TwoXChromosomes	2 366 474
fatpeoplehate	1 437 861
MensRights	966 959
loseit	761 489
islam	541 864
CoonTown	334 176
Feminism	88 459
racism	11 110
ShitN*ggersSay	50
Total	6 508 442

Table 1: Dataset statistics for Reddit data extracted from 2014-2015

Among all groups the agreement is low, just 0.14. We merged these datasets in the same way as Waseem, Thorne, and Bingel (2018).

The approach taken by Davidson et al. (2017) used a hate speech lexicon containing words and phrases identified by internet users as hate speech. They then scrape the entire history of the accounts that appears in the search. A sample of 25000 tweets where then sampled from this population and annotated by humans using the CrowdFlower platform. Each tweet is labelled according the a majority rule. The raters were given the definition of hate speech given by the authors: *[hate speech is] language that 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.* The reported inter-rater agreement is 0.92.

4.1.2 Reddit data

In 2015 a Reddit user by the name of *Stuck_in_the_matrix* posted a huge dataset consisting of the entire history of reddit. It has since been updated and realised to the public in large torrent files. We downloaded the files spanning 2014 and 2015 and grabbed the comments from nine subreddits: *mensrights*, *fatpeoplehate*, *coontown*, *beatingwomen*, *racism*, *twoxchromosomes*, *loseit*, *transgays* and *shitniggerssay*. This list includes several of the controversial communities on the website that were banned in 2015 (Moreno, Pao, and Ohanian 2015) in an attempt to combat hate and harassment. The choice of subreddits is to some extent arbitrary, but since hate speech is very dependent on specific words or tokens it made sense to pick a corpus that probably includes many of these.

4.2 Processing

With processing we mean the process of representing the raw texts as vectors. We start off with a short discussion of the processing strategies used in the benchmarks we are

comparing against. Then we discuss the processing pipeline used in our experiments. We also discuss how we split the data and prepared in for the experiments.

Davidson et al. (2017) rely on features extracted from the text, chosen in relation to the whole corpus. The tweets were first lower cased and stemmed. They then extracted n-grams that were weighted by the TF-IDF. They also introduced several counted features such as hashtags, mentions, retweets, and URLs. They also included syntactic features, reading ease scores, and sentiment scores.

Waseem and Hovy (2016) also builds their model on extracted features. They remove stop words except for the word *not*. They keep tokens for retweets, screen names and punctuations. Unigrams, bigrams, trigrams, and fourgrams of characters are extracted together with the length of each text, the gender of the user, and the users locality.

Waseem, Thorne, and Bingel (2018) compare two different representation techniques, the first is a bag-of-words like approach described above. The second is a different way of using word embeddings. Details can be found in the paper, but they choose not to use any architecture, like a recurrent neural network, that can process variable length sequences. Instead they take the element-wise mean of the word embeddings in a sequence and use that fix-length vector to represent a document. Like a bag-of-words representation, this does away with the sequential information and shares its drawbacks. Using word embeddings, especially pre-trained, should result in better predictions. Imagine a sentence containing the word *football*, using a bag-of-words approach our representation would be different if we swapped the word with *soccer*. As embeddings of words tends to be closer to words that are semantically similar, the distance between the words *soccer* and *football* is small. So, using word embeddings would give the two sentences similar representation. The texts were preprocessed, i.e. lower cased, removing stop words etc. in the same way as in Waseem and Hovy (2016).

Our preprocessing strategy is that it requires much less feature engineering. This means that we want to tweak our preprocessing pipeline as little as possible between datasets. Our preprocessing function adds tokens for several common emoticons, numbers, and mentions of users. It also adds tokens for elongated words – as an example, *laaaame* would be shortened to *lame ⟨elong⟩* – and repeated use of delimiting characters, such as exclamation marks or question marks. We also add a specific token for hashtags. In addition to this, we remove punctuation and extra delimiting characters, we lower case the words and split the comments into equal sized sequences. A common preprocessing technique often used when representing the documents as bag-of-words is to remove stop words [3], but since we model the sequences directly, we keep these words. Words that are

[3]Stop words are common words that usually doesn't carry any information. Words such as *I*, *you*, *are*, etc. are commonly regarded as stop words

Dataset split	Hate speech	Offensive	Neither	Total
Train	1144	15352	3331	19826
Validation	143	1919	416	2478
Test	143	1919	416	2478
Total	1430	19190	4163	

Table 2: Dataset statistics for Davidson et al. 2017

Dataset split	Racism	Sexism	Neither	Total
Train	1653	3371	10733	15757
Validation	207	421	1342	1970
Test	207	421	1342	1970
Total	2067	4213	13417	

Table 3: Dataset statistics for Waseem and Hovy 2016 and Waseem 2016

not in the vocabulary where replaced by an unknown-word token.

We also included a few extra tricks for the preprocessing. As earlier mentioned, there is an hate-speech online lexicon known as *hatebase.org*. To make sure that our model has capacity to read all of these words we added those that weren't among the first 50 000 most common words in the reddit data to the tokenizer, resulting in 50 105 words. The words that were in the hate speech lexicon, but occurred less than 10 times were given a hateful-unknown word token. The quintessential hateful word is the derogatory term *n*gger* and variations of this, unique words that begins with *nig** are given the unknown-n-word token.

We split the classification datasets in three parts:

- Training data: Used to update the weights
- Validation data: Used to measure out-of-sample performance of the model during training
- Test data: Used to measure out-of-sample performance after training. All results reported are from the test dataset

For both Waseem and Davidson data we created stratified samples consisting of approximately 10 percent of the overall number of documents each. The sizes of these datasets can be found in table 2 and table 3.

5 Experimental results

In this section we present the models and how they stack up against the current benchmarks. First we define the metrics that we use to evaluate our models, we then report our hyper-parameters, and then we go through the performance of the models.

5.1 Metrics

We will report on 3 different metrics: *precision, recall*, and *F1*. It is easy to mix some of these metrics up, so below is a quick recap on the different metrics.

We start off by considering a two class classification problem, an observation can either be 0 or 1. If we then have some model that produces predictions we get four different possibilities for each observation. We start with the two most obvious cases: *true positives*(tp) and *true negatives*(tn). The first one refers to the observations that are 1 and was predicted to be 1. The second refers to the observations that are 0 and that were predicted as 0. The last two cases refers to when our model is wrong. A *false positive*(fp) is an observation that is predicted to be 1 but that is 0. A *false negative*(fn) is an observation that is predicted as 0 but that are 1. This is summed up in the table below. For notational simplicity, we drop the sum-operator below when we define our metrics. So, we write the number of false positives as $\text{fp} = \sum \text{false positives}$.

	Predicted	
Label	0	1
0	True positive	False positive
1	False negative	True negative

Table 4: The different possible outcomes of a binary classification problem

We may now define our metrics. The first metric we are interested in is called *precision* and is defined as proportion of correctly classified examples among the one that are predicted to be 1.

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

The second metric is called *recall* and is the proportion of the observations that are 1 that are correctly classified as such.

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

And finally, the third metric is called the *F1-score* and is a harmonic mean of the *precision* and *recall*.

$$\text{f1} = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

All of these metrics are bound $[0, 1]$, the higher the better. Even though that we are dealing with more than two classes in our task, this argument is still valid. We can think of our problem in a one-vs-rest way. As in the case of the Waseem data, we have three classes: *sexist, racist* or *neither*. We can then consider each label at a time, calculate the metrics as if the problem were binary, and then report the weighted average of each metric.

5.2 Hyperparameters

Our language model has three recurrent layers, the first and third with 400 units in each, and the middle with 1150 units. We applied dropout to the hidden-to-hidden weight matrices in the middle lstm layer with a probability of 0.5. We also applied conventional dropout to the inputs of the first LSTM layer with a probability of 0.5, to the inputs of the second LSTM layer of 0.3 and to the inputs of the third LSTM layer of 0.1. During training we also had a validation sample of 5000 sequences. We trained each model for three epochs, one epoch took about 5-6 hours to run. We trained the forward model for six epochs and the reverse model for four epochs.

We fine-tuned the model gradually unfreezing the layers. We trained for two epochs per layer with $\eta_{min} = 0$ and $\eta_{max} = 0.002$, the first epoch for each layer with increasing learning rate and the second with decreasing. We used a batch size of 100 observations in the fine-tuning. We used the Adam (Kingma and Ba 2014) optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.7$. The language model has 30 million trainable parameters.

The classification models where trained using a learning rate multiplier of 0.7 using the Adam optimizer with β values the same as above. We used a fixed learning rate of 0.0005 and a batch size of 128. The two models were then concatenated and trained for two more epochs. In both cases we trained the combined model with a learning rate of 0.001 with multipliers of 0.001, using the same Adam optimizer as before.

The weights of the recurrent layers were initialised using a uniform distribution where the limits are varying with the number of outputs and inputs to the specific layer[4]. The embeddings were initialised using a uniform distribution with limits ± 0.05.

[4]See keras.io for details, we used the glorot_uniform initialiser

5.3 Performance on the Waseem data

The performance metrics for the Waseem data can be found in table 5. The bidirectional model managed to achieve an weighted average precision score of 0.83, an weighted average recall score of 0.83 which results in an weighted average f1-score of 0.83. The performance varies over the different classes, the *neither* class had by far the highest f1-score of 0.88, with precision at 0.85 and recall at 0.91. *Racism* has a slightly higher f1-score than *sexism* which is due to the low recall score of the latter. The gap between the precision, 0.78, and recall, 0.64, for the *sexism* class indicates that we have a relatively low amount of false-positives. This means that if our model classifies a text as sexist, we can be relatively certain that it actually is. Low recall indicates that our model missed many of the *sexism* examples, i.e. we observe a high amount of false-negatives. *Racism* has lower precision than *sexism* but higher recall.

Class	**F1**	**Precision**	**Recall**	**Support**
Neither	0.88	0.85	0.91	1342
Racism	0.73	0.74	0.72	207
Sexism	0.70	0.78	0.64	421
Weighted Average	0.83	0.83	0.83	

Table 5: Experimental results from Waseem / Waseem & Hovy data

If we consider the confusion matrix in table 6 we find that the model managed to separate sexism from racism. There were only two cases of racism that was classified as sexism, and only one example of sexism that was classified as racism. A large majority of the errors made by the model was when it misclassified any of the two hateful categories with neither and vice versa

	Predicted		
Label	Neither	Sexism	Racism
Neither	1217	50	75
Sexism	56	150	1
Racism	151	2	268

Table 6: Confusion matrix for Waseem / Waseem & Hovy data

In order to make sense of the errors the model made, and also to inform further research, we sampled three examples of every combination in the confusion matrix. Many of the examples are hard to make any sense of, for example the tweet

> Raw: @User People were making scientific discoveries, including Algebra, before Islam.

Processed: ⟨unk⟩ ⟨user⟩ people were making scientific discoveries including algebra before islam

Which was correctly classified as racist. Or the tweet

Raw: RT @User @User she is blond what do you want #notsexist
Processed: rt ⟨allcaps⟩ ⟨user⟩ ⟨user⟩ she is blond what do you want ⟨hashtag⟩ ⟨unk⟩

which was correctly classified as sexist. These examples indicates that the model learns to conflate use of some words with hate speech. In the first example the model most likely pick out the word *islam*. In the second case it would make sense to assume the model focus on the word *blond* or the hashtag that contained an unknown word. The tendency to have certain words being influential can also be seen in the tweet

Raw: Why We Need 'A Feminist Deck' LINK
Processed: why we need a feminist deck ⟨url⟩ ⟨unk⟩

which was mislabelled as sexist by our model. The tweet is annotated as *neither*. The word *feminist* is likely to play a big role in the classification. The model is not aware of the contents of hyperlinks, which in this case makes the tweet difficult to classify. Without any knowledge of the contents of the link this case is hard to classify.

In some cases the preprocessing choices are what makes our model misclassify, as in one of the two cases where we mistook sexism for racism.

Raw: RT @User: What #FemiNazi said young children are not sexual? "5 yr old son behaving sexually" LINK
Processed: rt ⟨allcaps⟩ ⟨user⟩ what ⟨hashtag⟩ ⟨unk⟩ nazi said young children are not sexual ⟨number⟩ yr old son behaving sexually ⟨url⟩ ⟨unk⟩

Which most likely was classified as sexism due to the misreading of the hashtag *#FemiNazi* as *⟨hashtag⟩ ⟨unk⟩ nazi*[5]

5.4 Performance on the Davidson data

The model does not manage to classify the hateful tweets very well. In table 7 we see that the f1-score for the *hate* class is 0.39, with a precision score of 0.30, and a recall score of 0.54. This is much lower than the f1-scores of the *offensive* and *neither* categories which

[5]Our preprocessing tries to split a hashtag that is a sentence, so that *#ThisIsASentence* would be *⟨hashtag⟩ this is a sentence*. This is achieved by splitting a hashtag by capital letters.

has f1-scores of 0.93 and 0.84. In turn, we observe low scores across the board for the *hate* class with a precision score of 0.54 and a recall score of 0.30. For the *neither* class we observe values of 0.84 for f1 and recall, and 0.83 for precision. The most common class *offensive* achieves a precision score of 0.93, a recall score of 0.96, which results in a f1-score of 0.94.

Class	**F1**	**Precision**	**Recall**	**Support**
Hate speech	0.39	0.54	0.30	143
Offensive language	0.94	0.93	0.96	1919
Neither	0.84	0.84	0.83	416
Weighted Average	0.89	0.89	0.90	2478

Table 7: Experimental results from Davidson data

In table 8 we see the confusion matrix for the Davidson data. It is obvious that we didn't manage to classify the *hate* class very well. In most cases, hate speech is conflated with offensive language. This is most likely a tendency of the model to base predictions on certain words.

	Predicted		
Label	Hate speech	Offensive language	Neither
Hate speech	43	87	13
Offensive language	29	1838	52
Neither	8	61	347

Table 8: Confusion matrix for Davidson data

Our model manages to correctly classify a few cases of the *hate speech* class. For easy examples such as,

> Raw: @whoswilly white boy talmbout Ight I'll see you tomorrow at school n*gger
> Processed: ⟨unk⟩ ⟨user⟩ white boy ⟨unk⟩ ⟨unk⟩ ill see you tomorrow at school n*gger

Or the tweet,

> Raw: @USER @USER @USER USER is Chi Sox Jew fag.
> Processed: ⟨unk⟩ ⟨user⟩ ⟨use⟩ ⟨user⟩ ⟨unk⟩ is chi sox jew fag

If we look at examples that were labelled *neither* but were predicted as *hate* we find some cases that are strange. For example,

Raw: @USER Marshall Law! Whatever you coon
Processed: ⟨unk⟩ ⟨user⟩ marshall law whatever you coon

which actually seems to be misclassified. The word *coon* is an insulting word, which basically can be translated to the n-word when used against blacks. So does the tweet

Raw: Do you have a nickname? What is it? — Tator, Tator tot, Tightey whitey, whitey Jr, Taylor whitey pants, agent ti... LINK
Processed: do you have a nickname what is it ⟨number⟩ ⟨unk⟩ ⟨unk⟩ tot ⟨unk⟩ whitey whitey jr taylor whitey pants agent ti ⟨repeat⟩ ⟨url⟩ ⟨unk⟩

which is difficult to classify, even for a human. The model classifies it as hate, which makes sense given earlier examples of the model depending on individual words in the tweets.

5.5 Comparing our results to the benchmark

In table 9 we list the results of other published metrics using the same data as us. Waseem, Thorne, and Bingel (2018) is the main source we compare our results to as they use similar data splits as we do and also utilize a transfer learning technique in their experiments. Davidson et al. (2017) reports the result they observe when training and measure performance using the same data. Even though they use a classifier that doesn't overfit like a neural network-based model might, it cannot be seen as a true estimate of out-of-sample performance.

Model	**F1**	**Precision**	**Recall**
Waseem / Waseem & Hovy data			
Base	0.80	.	.
Ours	0.83	0.83	0.83
Davidson data			
Base	0.90	0.89	0.91
Waseem, Thorne, and Bingel (2018)	0.89	.	.
Ours	0.89	0.89	0.90

Table 9: Hate speech classification benchmark results. The **Ours** refers to the model in this paper

When training with the Waseem data we observe a new state-of-the-art result with a weighted average f1-score of 0.83 to compare with the previously observed 0.80. We did

not manage to beat the benchmark set by Davidson et al. They observed a f1-score of 0.90 while we observe a score of 0.89 which interestingly is the same observed score as Waseem, Thorne, and Bingel (2018).

6 Conclusions

In this thesis we set out to build a scalable hate speech classification model. The model that we've describes is scalable in the way that we obtain state-of-the art, or close to it, on two previously published benchmark datasets. The model is scalable since we used the same preprocessing on the two different datasets, this gives our model a very nice plug-and-play feature. Given a new dataset, we can with minimal effort, train a classifier with good performance, or even state of the art performance.

Even though we didn't overcome the issue of depending on labelled datasets, and the issues that comes with it; such as varying definitions and low inter-rater agreement, we come close to a solution. We've shown that a semi-supervised approach to hate speech classification is the way forward to build good classifiers. The second part of the issue is coming up with the right annotation scheme, or even schemes depending on the platform and usage. Transfer learning lets us train complex architectures using minimal data, as shown in this thesis. This means that by using a transfer learning technique we don't need very large datasets to achieve good general performance.

Good areas of further research would be any of the following:

- Experiment with alternate unsupervised techniques, such as sequence auto-encoders.
- Experiment with different fine-tuning techniques and ensembling procedures.
- Research the unconscious biases that we learns from the annotated data. As shown in the results section, our model tends to see the occurrence of a certain hateful word as evidence enough. How does this affect the end user if the words that are learned to be hateful is words such as *homosexual*, *muslim*, or *feminism*?

References

Badjatiya, Pinkesh et al. (2017). "Deep learning for hate speech detection in tweets". In: *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, pp. 759–760.

Beauchamp, Zack (2018). "Incel, the misogynist ideology that inspired the deadly Toronto attack, explained". In: *Vox*. URL: https://www.vox.com/world/2018/4/25/17277496/incel-toronto-attack-alek-minassian (visited on 05/11/2018).

Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". In: *arXiv preprint arXiv:1607.04606*.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Consalvo, Mia (2012). "Confronting toxic gamer culture: A challenge for feminist game studies scholars". In: *Ada: A Journal of Gender, New Media, and Technology* 1.1.

Dai, Andrew M. and Quoc V. Le (2015). "Semi-supervised Sequence Learning". In: *CoRR* abs/1511.01432. arXiv: 1511.01432. URL: http://arxiv.org/abs/1511.01432.

Davidson, Thomas et al. (2017). "Automated Hate Speech Detection and the Problem of Offensive Language". In: *arXiv preprint arXiv:1703.04009*.

Duggan, Maeve et al. (2014). "Online Harassment". In: *Pew Research Center*. URL: http://assets.pewresearch.org/wp-content/uploads/sites/14/2014/10/PI_OnlineHarassment_72815.pdf (visited on 05/11/2018).

European Commission (2016). *European Commission and IT Companies announce Code of Conduct on illegal online hate speech*. Brussels, Belgium. URL: http://europa.eu/rapid/press-release_IP-16-1937_en.htm (visited on 05/11/2018).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Home Office (2016). *Action Against Hate: The UK Government's plan for tackling hate crime*. London, UK. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/543679/Action_Against_Hate_-_UK_Government_s_Plan_to_Tackle_Hate_Crime_2016.pdf (visited on 05/11/2018).

Howard, Jeremy and Sebastian Ruder (2018). "Fine-tuned Language Models for Text Classification". In: *arXiv preprint arXiv:1801.06146*.

Inan, Hakan, Khashayar Khosravi, and Richard Socher (2016). "Tying word vectors and word classifiers: A loss framework for language modeling". In: *arXiv preprint arXiv:1611.01462*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kwok, Irene and Yuzhou Wang (2013). "Locate the Hate: Detecting Tweets against Blacks." In: *AAAI*.

Merity, Stephen, Nitish Shirish Keskar, and Richard Socher (2017). "Regularizing and optimizing LSTM language models". In: *arXiv preprint arXiv:1708.02182*.

Mikolov, Tomas et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781. arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781.

Moreno, Jessica, Ellen Pao, and Alexis Ohanian (2015). *Removing harassing subreddits*. URL: https://www.reddit.com/r/announcements/comments/39bpam/removing_harassing_subreddits/.

Müller, Karsten and Carlo Schwarz (2017). "Fanning the Flames of Hate: Social Media and Hate Crime". In:

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

Press, Ofir and Lior Wolf (2016). "Using the Output Embedding to Improve Language Models". In: *CoRR* abs/1608.05859. arXiv: 1608.05859. URL: http://arxiv.org/abs/1608.05859.

Ross, Björn et al. (2017). "Measuring the reliability of hate speech annotations: The case of the european refugee crisis". In: *arXiv preprint arXiv:1701.08118*.

Saleem, Haji Mohammad et al. (2017). "A web of hate: Tackling hateful speech in online social spaces". In: *arXiv preprint arXiv:1709.10159*.

Schmidt, Anna and Michael Wiegand (2017). "A survey on hate speech detection using natural language processing". In: *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media. Association for Computational Linguistics, Valencia, Spain*, pp. 1–10.

Shrout, Patrick E and Joseph L Fleiss (1979). "Intraclass correlations: uses in assessing rater reliability." In: *Psychological bulletin* 86.2, p. 420.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Taylor, Jherez, Melvyn Peignon, and Yi-Shin Chen (2017). "Surfacing contextual hate speech words within social media". In: *arXiv preprint arXiv:1711.10093*.

The Guardian (2017). "Germany approves plans to fine social media firms up to €50m". In: *The Guardian*. URL: https://www.theguardian.com/media/2017/jun/30/germany-approves-plans-to-fine-social-media-firms-up-to-50m (visited on 05/11/2018).

Wan, Li et al. (2013). "Regularization of neural networks using dropconnect". In: *International Conference on Machine Learning*, pp. 1058–1066.

Waseem, Zeerak (2016). "Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter". In: *Proceedings of the first workshop on NLP and computational social science*, pp. 138–142.

Waseem, Zeerak and Dirk Hovy (2016). "Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter". In: *Proceedings of the NAACL Student Research Workshop*. San Diego, California: Association for Computational Linguistics, pp. 88–93. URL: http://www.aclweb.org/anthology/N16-2013.

Waseem, Zeerak, James Thorne, and Joachim Bingel (2018). "Multi-Task Learning for Domain Transfer of Hate Speech Detection". In: *Online Harassment*. Ed. by Jennifer Golbeck. London, UK: Springer.

Wulczyn, Ellery, Nithum Thain, and Lucas Dixon (2016). "Ex Machina: Personal Attacks Seen at Scale". In: *CoRR* abs/1610.08914. arXiv: 1610.08914. URL: http://arxiv.org/abs/1610.08914.