A Platform Agnostic Dual-Strand Hate Speech Detector

Johannes Skjeggestad Meyer and **Björn Gambäck**
Department of Computer Science
Norwegian University of Science and Technology
NO–7491 Trondheim, Norway
johannes@skmeyer.com, gamback@ntnu.no

Abstract

Hate speech detectors must be applicable across a multitude of services and platforms, and there is hence a need for detection approaches that do not depend on any information specific to a given platform. For instance, the information stored about the text's author may differ between services, and so using such data would reduce a system's general applicability. The paper thus focuses on using exclusively text-based input in the detection, in an optimised architecture combining Convolutional Neural Networks and Long Short-Term Memory-networks. The hate speech detector merges two strands with character n-grams and word embeddings to produce the final classification, and is shown to outperform comparable previous approaches.

1 Introduction

An increasing number of online arenas are becoming available for users worldwide to publish their opinions, from Internet fora and blogs, to microblog services like Twitter and social media such as Facebook and MeWe, and various chat rooms. However, in all arenas that are open to user generated content, there is a risk of some people misusing this opportunity to purposefully insult others, or even to convey hateful messages. This is often in breach of the given arena's terms and conditions, and sometimes, in some countries, illegal. Hence, there is a need for automatic detection of these messages across a multitude of online arenas, but without depending on any information specific to a given forum, so that the systems can be used across platforms without being changed.

Notably, information about the text's author, such as their usage history or their social network and activities, have been shown to be useful when categorising hate speech (Qian et al., 2018; Unsvåg and Gambäck, 2018; Mishra et al., 2018). In particular, on some occasions, an author belonging to an exposed group may use language that would normally be considered hateful towards that group, without the statement coming through as hateful. In such cases, disregarding user information may lead to misclassifications. However, what user metadata is stored may differ between services, and so using such information reduces the general applicability of a system. The research in this paper therefore aims at avoiding any such information, using exclusively text-based input in the detection. This is accomplished through a deep learning-based architecture combining Convolutional Neural Networks and Long Short-Term Memory-networks, and by utilising both character n-grams and word embeddings as input in a dual-strand methodology.

The rest of the paper is structured as follows: Section 2 describes prior work on hate speech detection. Section 3 then introduces the data set used in the experiments and Section 4 the proposed architecture. Section 5 presents experiments and results, while Section 6 discusses those. Finally, Section 7 concludes and presents ideas for future work.

2 Related Work

Research on hate speech detection has attempted many kinds of input features, and many different classification methods. In the early research, the input types used were highly language dependent, utilising specific syntax features and the presence of certain words. Later, these kinds of features were exchanged for more general text representa-

tions. Specifically, the approaches got more directed towards word- and character models, and in various alternations. Some researchers, such as Gambäck and Sikdar (2017), used both types at the same time, while others, e.g., Waseem and Hovy (2016) and Pavlopoulos et al. (2017), used only one of the types. Each kind of feature has its own advantage. The character n-gram approach is relatively resilient against misspellings, while word embeddings allow related words to produce similar output. In Mehdad and Tetreault (2016), word and character n-grams were used separately, in order to compare their performance, showing character n-grams to be more effective. Some systems, like that of Founta et al. (2018a), also apply various metadata and information about the author of the text. However, as the aim of this paper is to achieve classification more independent of the origin platform of the texts, such platform-dependent systems will largely be disregarded here.

Early research used traditional machine learning approaches, e.g., Support Vector Machines (SVMs) (Yin et al., 2009) and Naïve Bayes-based classifiers (Razavi et al., 2010). Some more recent research has also used traditional machine learning approaches, such as Logistic Regression (Waseem and Hovy, 2016). However, most recent work has focused on Deep Learning approaches: Gambäck and Sikdar (2017) and Park and Fung (2017) used Convolutional Neural Networks (CNNs), while Pavlopoulos et al. (2017) used Recurrent Neural Networks (RNNs) with Gated Recurrent Units (GRUs). Others have combined neural network-types, with Zhang et al. (2018) utilising a CNN followed by a GRU-based RNN, and Founta et al. (2018a) a two-part approach, with one part using word embeddings fed into an RNN-layer consisting of GRU-nodes, and the other, parallel part taking metadata as input to a feed-forward network.

Yet others have tried combining deep learners with more traditional methods: Badjatiya et al. (2017) tested both a CNN-based and a Long Short-Term Memory (LSTM)-based system (Hochreiter and Schmidhuber, 1997), in combination with Gradient Boosted Decision Trees (GBDT), while Gao et al. (2017) also used an LSTM, but running in parallel with a logistic regression system. In the SemEval 2019 OffensEval shared task (Zampieri et al., 2019b), the best performing systems utilised pretrained contextual embeddings such as BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018) and ELMo (Embeddings from Language Model) (Peters et al., 2018), hence in essence focusing on word-level n-grams (or *word pieces* as defined in BERT).

Several hate speech detection systems have been tested on the data set from Waseem and Hovy (2016) and can thus be compared more directly. Although the SVM-Naïve Bayes classifier of Mehdad and Tetreault (2016) outperformed their RNN-based system, deep learners seem to in general perform better than purely traditional machine learning classifiers on this dataset, with the CNN-based system of Gambäck and Sikdar (2017) outperforming the Logistic Regression-based system of Waseem and Hovy (2016), and with Badjatiya et al. (2017) claiming outstanding results for a hybrid system combining an LSTM with a GBDT. However, other researchers have failed to reproduce the experiments by Badjatiya et al., with Fortuna et al. (2019) indicating that Badjatiya et al.'s stated results rather were due to a faulty cross-validation process and with Mishra et al. (2018) noting that Badjatiya et al.'s decision tree-boosted version was tested on instances that the LSTM already had been trained on, leading to over-fitting.

3 Data Set

The largest data set used in research on inappropriate language is the one in Pavlopoulos et al. (2017), with 1.6 million comments from the Greek sports site *Gazzetta*. However, the labels in this data set are based on which comments the site's moderators found to be inappropriate in some way, including, but not restricted to, hate speech. The Twitter data set from Davidson et al. (2017) is also reasonably large and could have been an interesting option, but also somewhat lacks justifications for how each sample has been labelled: Davidson et al. attempted to differentiate between hate speech and other offensive content, but relied heavily on the crowd-sourced (CrowdFlower) annotators to make the distinction.

Version	Neutral	Racist	Sexist	Total
Original	11,559	1,972	3,383	16,914
Available	10,913	1,924	3,097	15,934

Table 1: Size of the Waseem and Hovy (2016) data set, as available at the time of data collection

On the other hand, Zampieri et al. (2019a), Golbeck et al. (2017), Founta et al. (2018b), and Waseem and Hovy (2016), all used extensive sets of rules when labelling their data. However, the first of those is aimed at offensive language, while the second is not straightforwardly available. And although the data set of Founta et al. (2018b) is substantially larger, the one by Waseem and Hovy (2016) has been used in more previous research, and was thus taken as the basis here, too, for reasons of easier comparison.

The data set of Waseem and Hovy (2016) originally contained 16,914 tweets labelled for racism and sexism. However, 980 of these tweets had been deleted by the time the data were collected, leaving 15,934 samples. As Table 1 shows, most of the deleted tweets were from the neutral group. As this is the largest group, it is also where the impact of deletion is the smallest. The smallest group, on the other hand, is where the loss is the lowest; more than 97% of the racist-labelled tweets were still available. The group with the greatest loss relative to size, is the sexist. Even here, though, more than 91% of the tweets still remained. In total, the loss constitutes less than 6% of the original tweets.

An issue with the data set is its representativeness. One aspect of this is the relatively high percentage of hate speech, at about 30%. In the data set of Pavlopoulos et al. (2017), too, about 30% of the samples were considered inappropriate, but there the 'positive' label was not restricted to just hate speech. In contrast, a study on the Facebook-pages of two Norwegian TV channels showed that every 10^{th} comment was hateful (Bjurstrøm, 2018), even after the media outlets had had 12 hours to moderate the debate. Similarly, Burnap and Williams (2015) found that 11% of tweets gathered in relation to a particularly hate-inducing event included offensive or antagonistic content, while Davidson et al. (2017), with a somewhat stricter definition, found 5% of their data to contain hate speech. This means that the propensity of hate speech is higher in the training data than what the system would face in real use. Furthermore, the Waseem and Hovy (2016) data was collected using bootstrapping, in particular of tweets related to an Australian TV cooking show, which could affect the results when applying a system trained on the data to arbitrary tweets.

4 Architecture

As discussed above, the input forms that have proven best for hate speech identification are word embeddings and character n-grams. Hence, the system described here uses both forms as input. However, the character- and word-based inputs are initially treated separately, in a dual-strand approach. Specifically, the system consists of a preprocessor and three main components. Two of those work in parallel, operating on the word and character-based inputs, respectively. The last component determines the final classification by combining the output of the previous two. Apart from the preprocessing, the system is implemented using TensorFlow.[1]

Text Preprocessing: The text samples (tweets) are first divided into mini batches, normally containing 20 samples each. Each tweet is then treated in two disjoint ways; one to create character representations, the other to create word representations. In both cases, Özcan's *tweet-preprocessor*[2] is used.

In the character-based preprocessing, each tweet is first cleared of emojis and lowercased, with each character transformed into a one-hot vector representation (a vector of length 31, with one slot each for the 26 letters of the English alphabet, four for space, number, '#', and '@', and one slot for any character that does not fall into any of the other categories). The samples of each mini batch are then zero-padded (post-data padded with only zero-valued vectors) to the length of the longest sample of that mini batch.

In the word-based preprocessor, emojis, URLs and Twitter-mentions are replaced with placeholders. Then hashtags are split into single words at capital letters, and the texts are

[1] `www.tensorflow.org`

[2] `pypi.python.org/pypi/tweet-preprocessor`

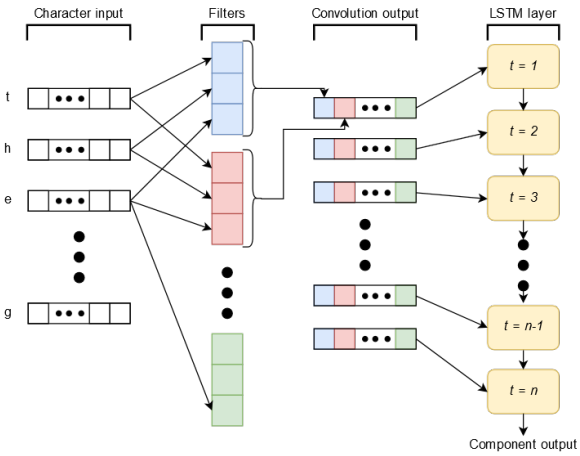

Figure 1: Character-handling component

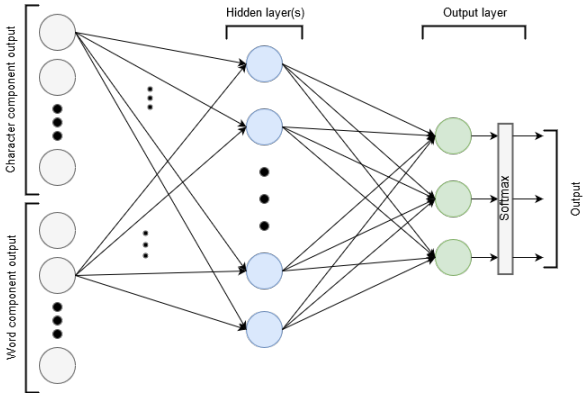

Figure 2: Architecture of the final classifier

lowercased, with punctuation and other symbols removed, and with all symbols that are not alphanumeric replaced by a space. The tweets are tokenised by splitting on white-spaces and the remaining words are transformed into their word embedding representations, with the batch samples zero-padded.

The word embeddings used here are pre-trained on external data sets, so as to avoid an additional source of overfitting due to the relatively small size of the Waseem and Hovy (2016) data set. Two different kinds of embeddings were used, word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). The word2vec-embeddings have a dimensionality of 300 and were trained on about 100 billion words from the Google News data.[3] The GloVe-embeddings on the other hand, were trained on Twitter, using 2 billion tweets.[4] The highest available dimensionality, 200, was used. Out of Vocabulary words were given a random value of corresponding dimensionality.

Word Input Component: The part of the system working on the word-based input, i.e., on word embeddings, is in the form of a Long Short-Term Memory (LSTM) network. The architecture allows for both unidirectional and bidirectional LSTM. The component's output for each sample should be the output state of the LSTM at the sample's last *relevant* (non-zero) time step. This is extracted by finding the non-padded lengths of the different samples, and collecting the LSTM-output at the time step corresponding to the last element.

[3] `code.google.com/archive/p/word2vec/`
[4] `nlp.stanford.edu/projects/glove/`

Character Input Component: The character-based portion of the system is divided into two parts; one convolutional and one recurrent, as shown in Figure 1. The architecture is inspired by that of Zhou et al. (2015) in how it combines these two elements. The first part takes the input and performs a 1-dimensional convolution, using multiple filters of size n, essentially treating the input as character n-grams. The output of this convolution is sorted by locations in the input, so that results of different filters at any given location appear together. This way, the results of the convolutions imitate the time steps of an LSTM. The architecture allows for several layers of convolution. In the second part of the component, the results of the convolution are input to an LSTM. The sample lengths of the LSTM are calculated from the output of the convolutions and used to extract the component output.

Final Classifying Component: Since different input samples vary in length, the above two components have to treat irregularity in input size, but the final component requires fixed-size inputs. Consequently, the outputs at the last relevant (last non-zero) time step for each of the first two components are combined by merging the two output vectors of each sample, with the result fed into a fully connected, feed-forward network, as Figure 2 shows. Note that the input layer simply provides data for subsequent layers, without applying any activation function.

In the output layer, the network has one node for each possible label (sexist, racist and neutral). The hidden nodes use ReLU as activation, but the output layer uses linear activa-

tion, with the weighted sum of a node's inputs used directly as output. This is run through a softmax layer, returning a probability distribution on which class a sample belongs to.

Training: The classification error of a sample during system training is calculated using cross entropy. The gradients of each weight's contributions to these losses are then calculated. After this, the gradients of the entire mini batch are accumulated, and used to update the system's weights according to the Adam optimiser (Kingma and Ba, 2014).

In order to avoid overfitting the network to the training data, some regularisation is necessary. The primary means of regularisation in this system is dropout (Srivastava et al., 2014), which is applied to the dense layers of the final component, as well as the LSTMs of the character- and word-based components. In the LSTMs, the dropout nodes vary from one time step to the next, and no dropout is applied to the states of the LSTM. In addition, the system uses L^2-regularisation, with the L^2-penalty calculated using all non-bias weights in the system, then added to the cross entropy classification error. Furthermore, the system uses early stopping, so that the training does not continue for too long. Combined, these three regularisers reduce overfitting in the system, thus increasing its general applicability.

In the experiments below, the hyperparameters of the Adam optimiser had the values suggested by Kingma and Ba (2014). The probability of "switching off" nodes due to dropout was set to 0.5, in accordance to the suggestions of Srivastava et al. (2014). All experiments were run using 10-fold cross validation, with stratified folds and size 20 mini batches.

5 Experiments and Results

To determine the optimal configurations of the system described above, experiments were carried out with varying layer sizes of the neural networks, as well as varying *number* of layers used in the different components. In addition, the system was tested using both bidirectional and unidirectional LSTMs.

In order to evaluate the effects of the variations consistently, the sizes of the word-based and the character-based components were changed separately. That is, when the sizes of the character-based component were changed, the word-based part was kept constant, and vice versa. This was done so that the best configuration of each component could be found independently, reducing the number of configurations to explore. As for variations in the number of layers, these were, for similar reasons, also made independently by component. Furthermore, in the character-based component, the number of convolutional and LSTM layers were changed separately. In the experiments with changes to the convolution, variations in the length of the convolutional filters were also made.

In addition, the system was tested using only character-based input, in order to evaluate the effectiveness of the CNN-LSTM combination on the character input. For comparative purposes, only using word-based input was also tested, disabling the character-based component. Beyond varying the setup configurations of the network itself, the effects of using different word embeddings were explored.

System Configuration Experiments: The first experiments separately tested variations to the components, with the unmodified part forming a baseline setup. In the first half of these experiments, each kind of nodes had one layer. Hence, the character-based component had one convolutional layer, followed by one LSTM layer; the word-based component had one LSTM layer; and the dense, feed-forward part had one hidden layer.

In the baseline system, the word-based component had one layer of 150 LSTM nodes. This dimensionality was chosen because it reduces the number of dimensions from the word embeddings, going down to half the size in the case of word2vec, without decimating the information carried through. The convolutional layer in the character-based part had 64 filters of length 3. The filter length here denotes the n in the character n-grams. This was set to 3 as trigrams have proven useful in prior work (Waseem and Hovy, 2016; Mehdad and Tetreault, 2016). 64 convolution filters were used since 64 is a power of 2 approximately twice the length of the character vectors. As such, it is significantly greater than the character vector size, while at the same time smaller than the size of each filter (i.e., 3×31).

Character		Word	*Unidirectional LSTM*			*Bi-LSTM*
Filters	LSTM	LSTM	**P**	**R**	**F$_1$**	**F$_1$**
64	100	150	79.12	**75.87**	**77.46**	**77.46**
100	100	150	79.06	74.90	76.93	77.08
50	50	150	79.42	74.94	77.11	77.31
512	256	150	79.59	75.24	77.35	77.06
64	100	50	**79.87**	74.58	77.13	77.11
64	100	100	79.30	74.67	76.92	77.04
64	100	200	79.10	75.38	77.20	77.21
64	100	250	79.24	74.67	76.89	77.10

Table 2: System configuration experiments.

The character-based component's LSTM layer had 100 nodes, a number chosen to balance the impact of the word- and character-based components on the final classifier, and since it should not be too much higher than the dimensionality of the convolution output (i.e., the number of filters used in the convolution). Hence, the final component had 250 input elements (150 from the word-based part and 100 from the character-based) and three output nodes; one for each class. Basheer and Hajmeer (2000) suggest that the number of nodes in a hidden layer should be between the numbers of input and output nodes. While such rules are not entirely reliable, the hidden layer size was set to 120; near the average of the input and output sizes.

In addition, the bidirectional version of this baseline configuration was tested, with each direction of the LSTMs having the dimensionality described above, thus giving the input to the final component twice the number of dimensions of the unidirectional case, so the hidden layer dimensionality was doubled.

The system was then tested with varying configurations in the character-based component, using 100 convolutional filters along with the 100 dimensions of the character LSTM. Then, the size was first cut down to 50 for both number of filters and LSTM layer size, and then increased to 512 filters and an LSTM layer size of 256. Finally, experiments were performed where the dimensionality of the word-based component was changed, while the character-based part had the default configuration, running the system with word-LSTM sizes of 50, 100, 200 and 250, respectively.

The results are shown in Table 2, for both the uni- and bidirectional configuration versions (only unidirectional precision and recall values are displayed, since the Bi-LSTM performance did not vary substantially). As the table shows, the baseline setup (row 1) worked best in terms of both recall and F$_1$-score. Several other configurations had better precision, such as the version where the word-based, unidirectional LSTM had a layer size of 50, but the corresponding recall values were comparatively lower than in the baseline setup.

Layer 1		Layer 2		**P**	**R**	**F$_1$**
Filters	Length	Filters	Length			
64	3	—	—	79.50	77.33	78.40
64	4	—	—	80.53	76.03	78.22
64	3	64	3	80.45	76.18	78.26
64	3	128	3	79.88	76.58	78.19
128	3	64	3	79.40	76.75	78.05
64	3	64	4	80.01	76.93	78.44
64	3	128	4	79.71	76.20	77.92
128	3	64	4	80.00	76.07	77.98
64	4	64	3	80.51	**77.73**	**79.10**
64	4	128	3	80.35	76.88	78.58
128	4	64	3	80.48	76.12	78.24
64	4	64	4	**80.57**	76.34	78.40
64	4	128	4	79.72	76.89	78.28
128	4	64	4	79.55	76.84	78.17

Table 3: Varying the convolutional segment of the character-based component. The setup columns show the number of filters at each consecutive layer, along with their corresponding filter lengths.

In these first experiments, the coefficient restricting the impact of the L^2-regularisation was given the commonly used value 0.001. However, the experiments showed that smaller values gave better results, so later experiments used a value of 0.0002 for this coefficient.

Convolution Experiments: In the next group of experiments, shown in Table 3, variations were made to the convolutional part of the character-based component (hence only unidirectional LSTMs were used, not bidirectional). Specifically, the system performance with filter length 4 was tested; then, an extra layer of convolution was added, with combinations of length 3 and length 4 filters being used. The standard number of filters in these experiments was 64, with the layers using a higher number having 128 filters. Next, the same three experiments were performed with the first convolutional layer using filters of length 3, and the second layer length 4. Then, the order was reversed, with the first layer filters having length 4 and the second layer length 3. Finally, the experiments were run with both layers using length 4 filters.

Setup	P	R	F_1
Baseline setup	79.50	**77.33**	**78.40**
Two character-LSTM layers	80.21	76.20	78.15
Two character-LSTM layers, bidirectional	79.73	76.59	78.13
Two word-LSTM layers	79.61	76.31	77.92
Two word-LSTM layers, bidirectional	79.69	76.38	78.00
Two convolutional layers $(64 \times 3, 64 \times 3)$ and two character-LSTM layers	79.39	76.09	77.71
Two convolutional layers $(64 \times 4, 64 \times 3)$ and two character-LSTM layers	**80.29**	76.36	78.27
Two LSTM layers each	79.40	76.38	77.86

Table 4: Using multiple LSTM layers

Since these experiments used a smaller value for the coefficient controlling L^2-regularisation, the first row of Table 3 reports a different baseline performance than row 1 in Table 2. The baseline setup still had the second highest score on recall, out-performed only by the best setup in these experiments. This configuration, with two layers of 64 convolutional filters where the first layer's filters were of length 4, and the second layer's of length 3, had a substantially better performance than the rest of the setups.

Two-layer LSTM Experiments: In addition to multilayer convolution, configurations using two-layer LSTMs were tested, with two same-sized layers in the LSTM part of the word- and character-based components, respectively. First, the character-based component's LSTM was given two layers of size 100, with the rest of the system having the settings of the baseline configuration. Then two 150-dimensional LSTM layers in the word-based component were used, reverting the character-based component back to the baseline. Further, the system was tested with both two convolutional layers *and* two LSTM layers in the character-based part, trying two settings of the convolutional section, one 'baseline-like' with the two convolutional layers each having 64 filters all of length 3, and the other version being the one which performed best in the convolution experiments above, i.e., two layers of 64 convolutional filters, with the first layer's filters having length 4, and the second layer's length 3. Finally, the baseline configuration was expanded to two LSTM layers in each of the system components holding LSTMs.

Table 4 shows the results and also includes the performance of the baseline setup, for comparison. Using two unidirectional LSTM layers in the character-based component of the baseline system setup and on the optimal convolution configuration (i.e., with filters of length 4 in the first convolutional layer) showed marked precision increases. However, recall in those cases was significantly weaker than in the baseline setup. Similar results, but with less marked precision increase, were found when using two LSTM layers in the word-based part, as well as in the bidirectional setup versions, and the equivalent two-character LSTM. Using two convolutional layers with all filters at length 3 and using two LSTM layers in each of the system components, gave lower precision than the baseline.

Setup	P	R	F_1
Baseline setup	79.50	77.33	78.40
Baseline, characters only	**81.38**	77.18	**79.23**
Two conv. layers $(64 \times 4, 64 \times 3)$	80.51	77.73	79.10
Two conv. layers $(64 \times 4, 64 \times 3)$, char. only	80.23	**77.84**	79.01
Baseline, words only	79.99	77.07	78.50

Table 5: Using only character or only word input

Single Component Experiments: Finally, the baseline setup was used, with one convolutional layer and one LSTM layer, but with the word-based LSTM removed and the dense layer reduced to 50 nodes. Then the equivalent was done using the best-performing configuration above, the system having two convolutional layers of 64 filters each, with lengths 4 and 3. For comparison, the system was then tested using just the word-based input. Here, too, the baseline setup was used as basis, meaning one LSTM layer of size 150.

The results are shown in Table 5. Interestingly, both of the character-only systems outperformed the baseline. Furthermore, the characters-only version of the baseline setup showed the highest precision of all the experiments in this research. As for the characters-only version of the configuration with two convolutional layers, the recall was higher than in the version including word-based input, but the precision was lower. Notably, it still outperformed the word-inclusive baseline setup on all measures. The word-only configuration was outperformed by the character-only systems, but still performed better than the baseline using all inputs.

All the above experiments utilised pre-trained word2vec embeddings. For comparison, the baseline and optimal configurations

were also evaluated using GloVe embeddings. In terms of F_1-score, both of the tested configurations improved when changing to GloVe. The baseline setup improved on all measures, though the improvement in precision was very slight. In the configuration with two convolutional layers, the precision got worse when changing to GloVe-embeddings. However, the recall of this setup using GloVe was the highest recorded throughout this research (78.28), outperforming the second best (the same configuration with the word-based component disabled) by more than 0.4%. In addition, the precision, while lower than the equivalent word2vec-performance, was still acceptably high (80.22). Hence, the resulting macro average F_1-score was 79.24 (84.14 micro average), which is higher than any other configuration in these experiments.

6 Discussion

The experimental results showed several consistencies. Notably, the recall values of all system configurations were lower than the corresponding precision. Furthermore, the recall had much greater variations between the different classes. Specifically, all the setups had the best performance on the recall of neutral samples, and the worst on sexist. The recall of sexist samples was also where the main difference from the change in value of the L^2-coefficient occurred. Using the original value of this coefficient, the recall on sexist samples was mostly in the range 53–58%, whereas with a lower coefficient value, the averages were mainly in the range 60–65%. In general, the performance on neutral samples was the most stable. The performance on the sexist class was mainly higher than on the racist one, although they tended to display opposite variations, so that when one class performed better, the other performed worse.

As Table 5 shows, using only one type of input in the default setting improved performance compared to using both. This is likely due to a difference in convergence rates between the two strands of the system, similar to the findings of Founta et al. (2018a). Word embeddings are inherently more informative than the one-hot vectors used for character input, and so the word-based strand is likely to have a significantly higher convergence rate than the character-based one. Such a discrepancy in convergence rates may cause one of the strands to dominate the other, hampering the training and resulting in an overall suboptimal performance. This issue may also have affected the experiments on variations in layer sizes and number of layers, as changes in the size of a system component will change its rate of convergence. These variations would work to the advantage of some configurations and the disadvantage of others. The results indicate that this may be the case. However, they are not sufficient to draw a conclusion.

The difference in performance between using word2vec- and GloVe-embeddings may to some extent be explained by the fact that the word2vec-embeddings were trained after removing stop words from the training data. Hence, in word2vec-embeddings, the stop words were considered Out of Vocabulary terms and given a random value. With the average number of words in the samples being 15, the impact of not having a meaningful representation of stop words could be significant. GloVe-embeddings, on the other hand, include representations of typical stop words, and thus have an advantage in the classification.

Several other researchers have tested their hate speech detection systems on the Waseem and Hovy (2016) data set. Table 6 shows the performance of some of these. Notably, Waseem (2016) introduced another, but related, data set, which Gambäck and Sikdar (2017) used, while Park and Fung (2017) used both data sets combined. A problem with the results shown in Table 6 is that different papers have used different methods to calculate the performance, with some using micro averaging (or weighted macro averaging) and others macro averages. Hence, Table 6 includes both the macro and micro averaged performance of the optimal configuration found in Section 5 (GloVe-embeddings and two convolutional layers: 64×4, 64×3).

As the macro averaged performance (upper part of Table 6) shows, the system using the optimal configuration with two convolutional layers and GloVe-embeddings outperformed the Waseem and Hovy (2016) system, and also had a higher performance, in

	System	P	R	F_1
macro avg	64×4, 64×3, GloVe	80.22	**78.28**	**79.24**
	Waseem and Hovy (2016)	72.87	77.75	73.89
	Waseem (2016), multiclass	—	—	53.43
	Waseem (2016), binary	—	—	70.05
	Gambäck and Sikdar (2017)	**85.66**	72.14	78.29
	Fortuna et al. (2019)	—	—	78
weighted / micro avg	64×4, 64×3, GloVe	**84.14**	84.14	**84.14**
	Zhang et al. (2018)	—	—	82
	Park and Fung (2017)	82.7	82.7	82.7
	Founta et al. (2018a)	84	83	83
	Badjatiya et al. (2017)	83.9	84.0	83.9
	Mishra et al. (2018) (WS)	82.86	83.10	82.37
	Mishra et al. (2018) (LR)	84.07	**84.31**	83.81
	Mishra et al. (2018) (HS)	83.50	83.71	83.54

Table 6: System performance comparison

terms of F_1-score, than the system of Gambäck and Sikdar (2017). However, that paper used a slightly different data set, and so the comparison is not entirely valid. In the case of the system by Waseem (2016), the approach described in Section 4 performed significantly better, particularly compared to the multiclass version, although these results are not for the primary data set of Waseem (2016), which had markedly higher performance.

Based on micro averaged performance values, the system clearly outperforms those of Zhang et al. (2018) and Park and Fung (2017). It also outperforms the system of Founta et al. (2018a), when this is restricted to using text as input, and the best non-GBDT version reported by Badjatiya et al. (2017).

Since Badjatiya et al.'s GBDT performance and cross-validation have been found to be questionable, Fortuna et al. (2019) give the macro average results they reported obtaining using the Badjatiya et al. (2017) system with decision tree boosting. Furthermore, Mishra et al. (2018) reimplemented three other systems in order to use as baselines for testing the improvements that could be obtained when utilising author profiling features. Hence, Mishra et al. (2018) (WS) is essentially a reproduction of Badjatiya et al.'s results, but with a slightly different setup, while Mishra et al. (2018) (LR) reproduces the LR-based approach taken by Waseem and Hovy (2016), and Mishra et al. (2018) (HS) is their implementation of the RNN approach used by Pavlopoulos et al. (2017).

7 Conclusion and Future Work

The dual-stranded CNN-LSTM combination for hate speech detection outlined here, which uses both word embeddings and character n-grams as input, performed relatively well on the Waseem and Hovy (2016) data set. Specifically, the system did well when using two layers of convolution on the character input, with diminishing filter lengths, combined with single layer LSTMs in both strands. Using multiple layers of LSTMs, on the other hand, actually reduced performance. With a macro averaged F_1-score of 79.24, the architecture performed better than all comparable, state-of-the-art systems on the data set.

It is possible that the different convergence rates in the architecture's word-based and character-based components may have reduced performance. A way to avoid this could be to train the system using an interleaving technique, as done by Founta et al. (2018a) — or take the similar multi-task learning approach suggested by Waseem et al. (2018) — so that only one of the two parallel system components is trained at any given time.

Another idea for further research would be to test the impact of using the architecture described here in combination with other top-level classifiers, such as the Gradient Boosted Decision Trees used by Badjatiya et al. (2017). It could also be interesting to investigate utilising dynamic convolutions for classifying hate speech, since Wu et al. (2019) report those as out-performing approaches based on self-attention, such as BERT (Devlin et al., 2018), on other language processing tasks.

Acknowledgements

Thanks to Zeerak Waseem and Dirk Hovy for providing the data set used here — and to all other researchers and annotators who contribute publicly available data.

Thanks also to all the anonymous reviewers for many useful comments, and to Elise Fehn Unsvåg, Vebjørn Isaksen, Steve Durairaj Swamy, Anupam Jamatia, and Amitava Das for insightful discussions on hate speech detection approaches, features and data sets.

References

Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760, Perth, Australia. International World Wide Web Conferences Steering Committee.

Imad A. Basheer and Maha Hajmeer. 2000. Artificial neural networks: Fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3 – 31.

Hanne Bjurstrøm. 2018. Hatefulle ytringer i offentlig debatt på nett (Hateful utterances in the public debate on the Internet). Likestillings- og diskrimineringsombudet, Norwegian Government, Oslo, Norway.

Pete Burnap and Matthew L. Williams. 2015. Cyber hate speech on Twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy & Internet*, 7(2):223–242.

Thomas Davidson, Dana Warmsley, Michael W. Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. *arXiv e-print*, abs/1703.04009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Paula Fortuna, Juan Soler-Company, and Nunes Sérgio. 2019. Stop PropagHate at SemEval-2019 Tasks 5 and 6: Are abusive language classification results reproducible? In *Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval)*, pages 741–748, Minneapolis, Minnesota, USA. ACL.

Antigoni-Maria Founta, Despoina Chatzakou, Nicolas Kourtellis, Jeremy Blackburn, Athena Vakali, and Ilias Leontiadis. 2018a. A unified deep learning architecture for abuse detection. *arXiv e-print*, abs/1802.00385.

Antigoni-Maria Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, Jeremy Blackburn, Gianluca Stringhini, Athena Vakali, Michael Sirivianos, and Nicolas Kourtellis. 2018b. Large scale crowdsourcing and characterization of Twitter abusive behavior. *CoRR*, abs/1802.00393.

Björn Gambäck and Utpal Kumar Sikdar. 2017. Using convolutional neural networks to classify hate-speech. In *Proceedings of the 1st Workshop on Abusive Language Online*, pages 85–90, Vancouver, Canada. ACL.

Lei Gao, Alexis Kuppersmith, and Ruihong Huang. 2017. Recognizing explicit and implicit hate speech using a weakly supervised two-path bootstrapping approach. *arXiv e-print*, abs/1710.07394.

Jennifer Golbeck, Zahra Ashktorab, Rashad O. Banjo, Alexandra Berlinger, Siddharth Bhagwan, Cody Buntain, Paul Cheakalos, Alicia A. Geller, Quint Gergory, Rajesh Kumar Gnanasekaran, Raja Rajan Gunasekaran, Kelly M. Hoffman, Jenny Hottle, Vichita Jienjitlert, Shivika Khare, Ryan Lau, Marianna J. Martindale, Shalmali Naik, Heather L. Nixon, Piyush Ramachandran, Kristine M. Rogers, Lisa Rogers, Meghna Sardana Sarin, Gaurav Shahane, Jayanee Thanki, Priyanka Vengataraman, Zijian Wan, and Derek Michael Wu. 2017. A large labeled corpus for online harassment research. In *Proceedings of the 2017 ACM on Web Science Conference*, WebSci '17, pages 229–233, Troy, New York, New York, USA. ACM.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Yashar Mehdad and Joel Tetreault. 2016. Do characters abuse more than words? In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 299–303, Los Angeles, California, USA. ACL.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv e-print*, abs/1301.3781.

Pushkar Mishra, Marco Del Tredici, Helen Yannakoudakis, and Ekaterina Shutova. 2018. Author profiling for abuse detection. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1088–1098, Santa Fe, New Mexico, USA. ACL.

Ji Ho Park and Pascale Fung. 2017. One-step and two-step classification for abusive language detection on Twitter. In *Proceedings of the 1st Workshop on Abusive Language Online*, pages 41–45, Vancouver, Canada. ACL.

John Pavlopoulos, Prodromos Malakasiotis, and Ion Androutsopoulos. 2017. Deep learning for user comment moderation. In *Proceedings of the 1st Workshop on Abusive Language Online*, pages 25–35, Vancouver, Canada. ACL.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Processdings of the Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. ACL.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Jing Qian, Mai ElSherief, Elizabeth Belding, and William Yang Wang. 2018. Leveraging intra-user and inter-user representation learning for automated hate speech detection. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 2 (Short Papers), pages 118–123, New Orleans, Louisiana, USA. ACL.

Amir H. Razavi, Diana Inkpen, Sasha Uritsky, and Stan Matwin. 2010. Offensive language detection using multi-level classification. In *Advances in Artificial Intelligence: Proceedings of the 23rd Canadian Conference on Artificial Intelligence, Canadian AI 2010*, pages 16–27, Ottawa, Canada. Springer.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Elise Fehn Unsvåg and Björn Gambäck. 2018. The effects of user features on Twitter hate speech detection. In *Proceedings of the 2nd Workshop on Abusive Language Online*, pages 75–85, Brussels, Belgium. ACL.

Zeerak Waseem. 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*, pages 138–142, Austin, Texas, USA. ACL.

Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? Predictive features for hate speech detection on Twitter. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 88–93, San Diego, California, USA. ACL.

Zeerak Waseem, James Thorne, and Joachim Bingel. 2018. Bridging the gaps: Multi task learning for domain transfer of hate speech detection. In Jennifer Golbeck, editor, *Online Harassment*, pages 29–55. Springer, Cham, Switzerland.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *CoRR*, abs/1901.10430.

Dawei Yin, Brian D. Davison, Zhenzhen Xue, Liangjie Hong, April Kontostathis, and Lynne Edwards. 2009. Detection of harassment on Web 2.0. In *Proceedings of the Content Analysis in the Web 2.0 Workshop at WWW2009*, Madrid, Spain.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019a. Predicting the type and target of offensive posts in social media. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1415–1420, Minneapolis, Minnesota, USA. ACL.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019b. SemEval-2019 Task 6: Identifying and categorizing offensive language in social media (OffensEval). In *Proceedings of the 13th International Workshop on Semantic Evaluation (SemEval)*, pages 75–86, Minneapolis, Minnesota, USA. ACL.

Ziqi Zhang, David Robinson, and Jonathan Tepper. 2018. Detecting hate speech on Twitter using a convolution-GRU based deep neural network. In *European Semantic Web Conference*, Lecture Notes in Computer Science, pages 745–760, Heraklion, Greece. Springer.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. 2015. A C-LSTM neural network for text classification. *arXiv e-print*, abs/1511.08630.