SWE2: SubWord Enriched and Significant Word Emphasized Framework for Hate Speech Detection

Guanyi Mou, Pengyi Ye, Kyumin Lee
Worcester Polytechnic Institute
{gmou,pye3,kmlee}@wpi.edu

ABSTRACT

Hate speech detection on online social networks has become one of the emerging hot topics in recent years. With the broad spread and fast propagation speed across online social networks, hate speech makes significant impacts on society by increasing prejudice and hurting people. Therefore, there are aroused attention and concern from both industry and academia. In this paper, we address the hate speech problem and propose a novel hate speech detection framework called *SWE2*, which only relies on the content of messages and automatically identifies hate speech. In particular, our framework exploits both word-level semantic information and sub-word knowledge. It is intuitively persuasive and also practically performs well under a situation with/without character-level adversarial attack. Experimental results show that our proposed model achieves 0.975 accuracy and 0.953 macro F1, outperforming 7 state-of-the-art baselines under no adversarial attack. Our model robustly and significantly performed well under extreme adversarial attack (manipulation of 50% messages), achieving 0.967 accuracy and 0.934 macro F1.

CCS CONCEPTS

• **Information systems** → *Information retrieval*; • **Computing methodologies** → *Natural language processing*.

KEYWORDS

hate speech detection; online social networks

ACM Reference Format:
Guanyi Mou, Pengyi Ye, Kyumin Lee. 2020. SWE2: SubWord Enriched and Significant Word Emphasized Framework for Hate Speech Detection. In *The 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411990

1 INTRODUCTION

Hate speech, "abusive speech targeting specific group characteristics" [45], has long been causing annoying disturbance to many people's lives, in terms of misleading the trends, shaping bias and discrimination, aggregating and aggravating conflicts among different religious, gender, and racial groups, etc. With the rapid growth of online social networks, hate speech is spreading faster and affecting a larger population than before in human history across the world[1]. Therefore, quickly and accurately identifying hate speech becomes crucial for keeping a harmonic and healthy online social environment, mitigating the possible conflicts, and protecting the diversity of our society. Hate speech detection is also helpful for public sentiment analysis and is useful as one of the pre-processing steps in content recommendation and chatterbot development [3].

Over the years, researchers have proposed various methods for detecting hate speech [2, 9, 15, 29, 33, 50], many of which have focused on feature engineering. New hand-crafted features were raised and checked from different perspectives to improve their overall performance. The content of hate speech was inevitably leveraged to generate features. The produced features vary from counting based features, sentiment, and semantic features to pre-trained word-level embedding and sentence-level embedding related features. However, the procedures of generating these features highly depend on two crucial presumptions: the sentences can be successfully tokenized into completely genuine atomized words, and these words can be recognized/categorized into bins. Thus, these methods/features are intuitively vulnerable in resisting character-level manipulation, which converts semantically significant known words to meaningless unknown words [27].

While in practice, it has been reported that intentionally or deliberately misspelled words as a kind of adversarial attacks are commonly adopted as a tool in manipulators' arsenal to evade detection. These manipulated words may vary in many ways but may not occur in normal natural language processing (NLP) related dictionaries [33, 45]. In fact, it is not practical to generate a vocabulary that includes everything, as the atmosphere of online social networks is dynamic: new words/variations are emerging almost every day. Even legitimate users can sometimes accidentally make typos [42]. The prior hate speech detection methods may not handle these cases properly.

To address the mentioned problem, we propose **SWE2**, the ***S**ub**W**ord **E**nriched and **S**ignificant **W**ord **E**mphasized* framework, which not only embraces the word-level semantics but also leverages subword information, to recognize typos/misspellings, resist character-level adversarial attacks and improve robustness and accuracy of hate speech detection. Our framework incorporates two types of subword embeddings: the phonetic-level embedding and the character-level embedding. In addition, we carefully designed an LSTM+attention based word-level feature extraction method, which extracts general content semantic information across the speech. For subword representations, we trained our domain-specific embeddings. While for word representations, we tested two pre-trained variations: the state-of-the-art generalized FastText embedding [24],

[1] https://www.cfr.org/backgrounder/hate-speech-social-media-global-comparisons

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411990

and the latest and most advanced generalized BERT embedding [14] to see which word representation complements our subword representations for hate speech detection. With the combination of word-level and subword-level representations, our framework can achieve high performance and robustness with/without the character-level adversarial attack (i.e., intentionally manipulating some characters of a message to evade hate speech detection).

To sum up, the contributions of this paper are listed as follows:

- We proposed to incorporate two new vital proportions of subword information in hate speech detection: character-level information and phonetic-level information. Both of them are non-trivial and work as complementary to each other. To the best of our knowledge, we are the first to incorporate pronunciation information in hate speech detection domain and show it makes a non-trivial contribution.
- We designed a novel hate speech detection framework, which utilized CNNs for subword information extraction, LSTMs for word-level information extraction, and attention mechanisms with statistical MAXes and MEANs for better feature generation. In the word-level information extraction, we compared FastText and BERT to see which one contributes more in the framework.
- We investigated the performance of our model without and with a black-box adversarial attack. We showed our model outperformed 7 state-of-the-art baselines, achieving minor reduction even under very extreme attack (i.e., manipulation of 100% messages in the test set), indicating robustness of our model.
- Our model only relies on the content of a hate speech message, not requiring other side information such as a user profile or message propagation status in the network. Therefore, it is more efficient than some of the prior methods in terms of feature utilization and prediction time.

2 RELATED WORKS

2.1 Hate Speech Detection

In the early literature [13], hate speech was defined as "the language used to express hatred against a specific target group or to be demeaning, insulting, or insult the team members". Mai ElSherief [16] specified hate speech into two classes: (1) directed hate – hate language towards a specific individual or entity; and (2) generalized hate – hate language towards a general group of individuals who share a common protected characteristic.

Despite the existing work dedicated to detecting hate speech [40], hate speech detection is still challenging in the NLP domain. The main reason is due to linguistic diversity and words' semantic ambiguity. Even though there have been several public corpora and automatic linguistic classifiers focusing on hate speech detection, the limitation and weakness of the existing classifiers based on text features are apparent. Intentional typos/character-level manipulation made by hate speech posters can easily avoid the prior hate speech detection methods based on texture features.

Ousidhoum et al. [34], Davidson et al. [13], Waseem and Hovy [47], and Elsherief et al. [16] released their annotated hate speech datasets in public. We made use of the latter three datasets in our research. Mathew et al. [31] and Chung et al. [11] provided counter speech datasets for better analysis of hate speech.

There are also papers focusing on in-depth analysis of hate speech: Warner and Hirschberg [45] did an overall analysis of hate speech detection. Waseem [46] wrote about the annotator's influence on hate speech detection, showing expert annotations contributed to a better detection rate. Arango et al. [1] analyzed a model validation problem of hate speech detection.

For classification tasks, Nobata et al. [33] tried various types of features and reported informative results. Recent papers leveraged CNNs, LSTMs and attention mechanisms for better detection results [2, 3, 9, 17, 29, 50]. Djuric et al. [15] experimented on using paragraph-level embeddings for hate speech detection.

Our approach also incorporates LSTM, CNNs, and an attention mechanism. However, our approach differs from the prior works in the following ways:

- First, we use word-level, character-level, and phonetic-level embeddings for hate speech detection, whereas the prior works only focused on one-level representation.
- Second, we apply different techniques for different parts of our framework to best utilize each method's advantage (i.e., CNNs for subword information extraction, and LSTMs for word-level information extraction), whereas the prior works used LSTMs and CNNs with simple concatenation or sequential addition.
- While other state-of-the-art methods relied on spelling correcting tools such as [19] or spellchecker[2] for word recovery (i.e., to fix typos), our framework is focusing on direct prediction without the word recovery (and without using spelling correcting tools) so that it can avoid possible new bias/error caused by the spelling correcting methods.

We discuss why and how our ideas of utilizing LSTMs are better than the prior approaches in Section 4.

2.2 Adversarial Attack

In the black-box attack, attackers do not know details of a detection framework, thus only generates the best possible guess to avoid exposure. In this paper, we focus on a black-box attack, especially, called a character-level adversarial attack. There is much clear evidence in character-level manipulations in online social networks. For instance, when a word[3] 'nigger' has a misspelling and is changed to 'n1gger' or 'nigga' which cannot be recognized by most word embedding models. There are also systematic methods for generating character-level manipulations such as [19, 27]. They reported general methods for attacking different existing frameworks and claimed success in almost all experiments.

Serra et al. [41] analyzed how out-of-vocabulary words can affect classification errors in detecting hate speech. Later on, Grondahl et al. [20] reported that character models are much more resistant to simple text-transformation attack against hate speech classification. Inspired by that, our framework incorporates the subword information to detect hate speech containing character-level manipulation and achieves better prediction accuracy than the prior methods.

[2] https://norvig.com/spell-correct.html

[3] **It is crucial to note that this paper contains hate speech examples (in Section 2.2, 4.2, 6.5, and Table 1), which may be offensive to some readers. They do not represent views of the authors. We tried to make a balance between showing less number of hate speech examples and sharing transparent information.**

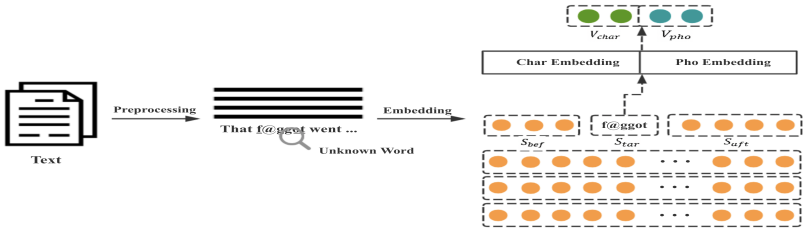

Figure 1: Process of extracting word-level and subword-Level embeddings in our framework.

3 BACKGROUND: EMBEDDING METHODS

Sentence-Level Embedding. Many existing works search for a higher level of text encodings, such as sentence-level or paragraph-level. Researchers proposed innovative ways to produce them [7, 12, 21, 26, 30, 38, 43].

Word-Level Semantic Embedding. There are many state-of-the-art semantic embedding methods such as context-independent Word2Vec [32], and GloVe combining counting based vectors [36], character-based embeddings such as FastText [24], shallow bidirectional LSTM generated context-related ELMo [37], and the most recent transformer generated BERT [14]. In our research, we utilized each of FastText and BERT to see which one performs well in our framework for hate speech detection.

Character-Level Embedding. The effectiveness of character-level information in the NLP domain was described in [28]. Similarly, a variety of works exploring the subword-based representations in this domain emerged consistently from several perspectives, such as part-of-speech tagging [39], parsing [4], and normalization[10]. [8] explored such linguistic patterns in Chinese and proposed a character-enhanced word embedding model.

It is reported that many traditional word representations ignore the morphology of words, by assigning a distinct vector to each word. Bojanowski et al. [6] note: "This is a limitation, especially for languages with large vocabularies and many rare words.". Inspired by their discovery and effort in emphasizing such information, we explore the insight of words by incorporating the character-level embedding in our research.

Phonetic-Level Embedding. Nowadays, phonetic recognition gains its popularity with the evolutions of hardware and software. CMU provided pronouncing dictionary[4] to translate characters of each word into phonetic symbols. The phoneme distribution and syllable structure of words in this dictionary have been explored by [48] and compared with the results obtained from the Buckeye Corpus [49]. Peng et al. [35] experimented with the strength of the pronunciation features of the text over sentiment analysis in Chinese by enriching the text representations. Based on the inspiration, we propose to incorporate the phonetic embedding method into our framework for hate speech detection, and further propose and develop a way to extract symbolic/phonetic representation of unknown words to learn their phonetic embeddings. The detailed approach of extracting phonetic embedding of unknown words is described in Section 5.3.1.

4 OUR FRAMEWORK

We introduce our novel hate speech detection framework, **SWE2** (pronounced as 'sweet'), which only relies on the text content of a message, identifies most essential words, and extracts their surrounding information to predict whether it is hate speech or not. Figure 1 shows the process of extracting word-level and subword-level embeddings in our framework, and Figure 2 shows the overall framework which uses these embeddings as a part of input, and outputs the final prediction.

4.1 Task and Procedure

Task. Given a text message/speech, the framework ought to make a prediction of whether such delivered speech is hate speech or not.

Cleaning. We clean the messages as follows: we intentionally remove all sentence punctuation and make the message to lower case since what we care about in hate speech is the content itself.

Tokenization and redundant information handling. Then, we tokenize the message, remove special characters for post tags and only keep their content, replace mentions of usernames with "USER" and links with "URL", and feed the result to our network. Assuming the message is a tweet obtained from Twitter. However, our framework can handle any other text messages from other web sites with minimum customization.

Most significant word recognition. Given the tokenized word (string) sequence, we aim to identify the most significant word (called *target* word). We first use VADER [22] for searching the most sentimentally strong word. If all words are sentimentally contributing similarly, we will compare each word with possible hate speech words in the given dictionary [16] to see whether hate speech words are having at most two-character difference compared with the word in the tokenized word sequence. We can quickly achieve such a goal by implementing a simple longest common subsequence method. If we find no similar word from the sequence, then the framework will randomly assign a word as the *target* word. Section 6.2 describes why we care about identifying a *target* word of each message and why our framework pays extra attention to the *target* word.

Splitting and Embedding. Given the *target* word in a sentence/message, we split the original sentence S_{Ori} into three parts, namely part before *target* word S_{Bef}, *target* word S_{Tar}, and the part after *target* word S_{Aft}.

$$S_{Ori} = [S_{Bef}, S_{Tar}, S_{Aft}] \tag{1}$$

We use the character-level and phonetic-level embeddings trained by ourselves to represent the *target* word S_{Tar}, deriving representations of V_{Char} and V_{Pho} as shown in Figure 1.

$$V_{Char} = EmbC(S_{Tar}) \tag{2}$$

$$V_{Pho} = EmbP(S_{Tar}) \tag{3}$$

We feed these two matrices to separate CNNs, trying to fetch important information of the *target* word as shown in Figure 2.

[4] http://www.speech.cs.cmu.edu/cgi-bin/cmudict#phones

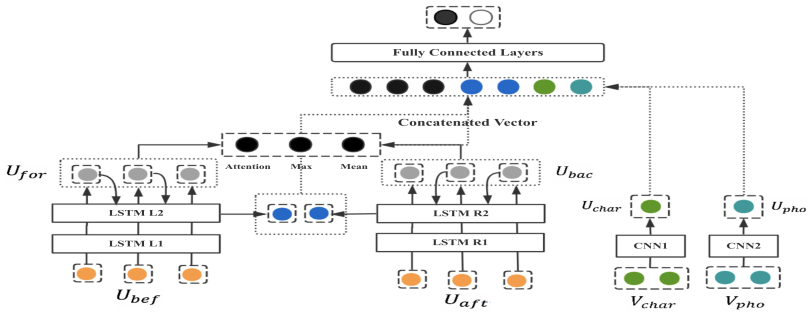

Figure 2: The overall framework.

$$U_{Char} = CNN1(V_{Char}) \tag{4}$$

$$U_{Pho} = CNN2(V_{Pho}) \tag{5}$$

We use word-level embedding methods to represent each word in S_{Bef} and S_{Aft} separately, stacking word vectors together and receive representations of two matrices $U_{Bef} \in R^2$ and $U_{Aft} \in R^2$.

$$U_{Bef} = EmbW(S_{Bef}) \tag{6}$$

$$U_{Aft} = EmbW(S_{Aft}) \tag{7}$$

Now two matrices representing the part of the message before and after the *target* word are fed to two separate LSTM models, to extract useful information as shown in Figure 2. Inspired by the ELMo design [37], which had two layers of LSTM to learn syntax and grammar of the sentence in the first layer and to address semantic meanings of words and disambiguation in the second layer, each of our two LSTM models has two layers. We call the first LSTM model as forward LSTM and the second one as backward LSTM because U_{Bef} is fed to forward LSTM, and U_{Aft} is reversed (i.e. reversing the order of word vectors of U_{Aft}), and then is fed to backward LSTM. In this way, the last outputs of the two LSTMs can be treated as the prediction of the *target* word, while the other outputs of the two LSTMs can be seen as global side information. All of the outputs from the second layers of the two LSTMs are collected and formed as two matrices $U_{For} \in R^2$ and $U_{Bac} \in R^2$.

$$U_{For} = LSTM_{Forward}(U_{Bef}) \tag{8}$$

$$U_{Bac} = LSTM_{Backward}(Reverse(U_{Aft})) \tag{9}$$

Now we split each output matrix into two parts: the last outputs of two LSTMs can easily be explained as the predicted vector representation of the *target* word (i.e., $U_{ForLast}$ and $U_{BacLast}$), so we separate them from the other outputs (i.e., $U_{ForRest}$ and $U_{BacRest}$).

$$U_{For} = U_{ForLast} \oplus U_{ForRest} \tag{10}$$

$$U_{Bac} = U_{BacLast} \oplus U_{BacRest} \tag{11}$$

where $\oplus$ denotes concatenation.

Eventually we form the global information $U_{Glo} \in R^2$, and also the focused local representation $U_{Loc} \in R^1$. Notice that global information is everything except the *target* word, while the focused local representation is only about the *target* word.

$$U_{Glo} = U_{ForRest} \oplus U_{BacRest} \tag{12}$$

$$U_{Loc} = U_{ForLast} \oplus U_{BacLast} \oplus U_{Char} \oplus U_{Pho} \tag{13}$$

Now the global information is too large and may contain much redundancy, so we only use self-attention, max and mean information extracted from it and concatenate them, namely $U_{Glo2} \in R^1$

$$U_{Glo2} = Attn(U_{Glo}) \oplus Max(U_{Glo}) \oplus Mean(U_{Glo}) \tag{14}$$

Finally, we combine the global information and local information altogether, feed them to multiple fully connected layers, and then make the final prediction.

$$Pred(S_{Ori}) = argmax(MultiFC(U_{Glo2} \oplus U_{Loc})) \tag{15}$$

4.2 Character-Level Manipulation for the Adversarial Attack

In this subsection, we describe how to simulate the character-level adversarial attack. Due to the diversity of text combinations and varieties of typos, it is labor-consuming to manually collect large-scaled hate speech data, which contains deliberate typos as well as recovering their original perfect spelling. To the best of our knowledge, there is no existing publicly available large scale dataset, which includes both real-world hate speech with deliberate typos and the recovered ones.

So we turned to apply scalable simulation methods to generate spelling errors. We used *TEXTBUGGER* framework [27], which can create utility-preserving adversarial texts against the state-of-the-art text classification systems effectively and efficiently, focusing on the *target* word manipulation and mimicking evasion behavior of hate speech posters as we described them in Section 6.2.

According to [27], we could consider five types of bug generations (i.e., manipulation) inside one word (i.e., the *target* word):

- Insertion: insert an extra space in the word
- Deletion: delete a character in the word
- Swapping: switch the position of two neighbor characters
- Sub-c: substitute a character with a similar character
- Sub-w: replace the word with its closest meaning neighbor

From the description of hate speech on *Wikipedia*[5], the domain of the hate speech is deeply nested in the offensive and hostile speech. However, the boundary of hate speech is so strict, so even the nearest word in semantics or word embedding representations can unlikely be hate speech. For instance, the most similar word[6] of "limey", which is an insulting word for a British person, is "yeasty", which might be offensive in some scenarios but far from being hate speech. Therefore, in our experiments, we choose not to do any word-level manipulation to avoid any labeling bias introduced to the manipulated data. Based on that, *sub-w* is not selected.

Insertion is also not chosen for two reasons. First, inserting a space can be interpreted as a method of word-level manipulation,

[5]https://en.wikipedia.org/wiki/Hate_speech

[6]Here we used Word2Vec vector and measured with cosine similarity

Table 1: Examples of character-level manipulation.

	Char		Phonetic	
Method	Original	Manipulated	Original	Manipulated
Swap	fucking	fukcing	limey	liemy
Delete	wigger	wiger	coonass	coonas
Sub-C	trash	tr@sh	nigger	neegeer

where one word is split into two words if we add whitespace inside of the word. Secondly, splitting one word into two words may severely impact the readability of the whole speech. Taking these scenarios into consideration, we use character-level *sub-c*, *deletion*, and *swapping* to keep the original meaning of a message for human readers. We show examples of the character-level manipulations in Table 1. Sometimes hate speech posters do character-level manipulation with/without considering phonetic similarity. The examples consider both cases.

To generate the character-level manipulation, our attack system automatically searches for hate speech candidate words in the aforementioned hate speech words dictionary. Then, it selects one of them for the manipulation. If there was no hate speech candidate words, the manipulation happens in a randomly selected word.

To decide which attacking method among *sub-c*, *deletion*, and *swapping* is the most effective for a given message, we used Universal Sentence Encoder [7], a sentence-level embedding framework, to encode the whole message into a fixed-length vector. Then we used cosine similarity to compare the distance of the original message (before the manipulation) and the manipulated message (after the manipulation). We chose the attacking method that produces the longest distance among them.

5 EXPERIMENTS

5.1 Data Collection

To conduct experiments, we used some portion of three existing datasets (i.e., Waseem [47], Davidson [13] and HateLingo [16] datasets) and collected legitimate tweets from Twitter as follows:

Waseem dataset [47] includes 17,325 tweets which were manually labeled into sexism, racism, offensive, and neither. The labels of the messages were automatically identified, and the reliability and consistency of labels were manually investigated and verified. As we are aware of the fact that offensive speeches do not necessarily lead to hate speech, we filtered out the offensive messages; however, we kept the sexism and racism as hate speech. Eventually, we fetched out 2,778 hate speech and 7,133 legitimate speech from the dataset.

Davidson dataset [13] includes 24,783 tweets, consisting of offensive speech, hate speech, and neither. Similar to what we did in the previous dataset, we removed offensive speech, eventually using 1,294 hate speech and 3,925 legitimate messages.

HateLingo dataset [16] contains only hate speech messages crawled via Twitter Streaming API with specific keywords and hashtags defined by Hatebase[7]. To recognize the anti-hate tweets which may also contain hate speech terms, the authors cleaned the dataset by using Perspective API[8], and conducted manual checking during the experiment. As they only provided TweetID, we had to fetch the actual messages through Twitter API. We were able to collect 12,631 hate speech messages.

[7]https://www.hatebase.org/

[8]https://github.com/conversationai/perspectiveapi/blob/master/api_reference.md

Table 2: Dataset.

source	|hate speech|	|legitimate|
Waseem [47]	2,778	7,133
Davidson [13]	1,294	3,925
HateLingo [16]	12,631	0
Legitimate	0	72,457
Total	16,703	83,515

Legitimate Messages: To balance the proportion of hate speech and legitimate messages, we randomly collected 1% real-time tweets by Twitter API, and then selected 800,000 tweets in English. To guarantee these tweets are legitimate messages, we followed thorough labeling process: filter out the messages which contain the aforementioned hate speech keywords or have negative sentiment scores by following the rule proposed in [16]. Finally, 72,457 messages were chosen, and we manually sampled 2,000 messages to see whether there is any hate speech message or not. All of them were legitimate. Therefore, we labeled them as legitimate messages. Dataset can be found at https://web.cs.wpi.edu/~kmlee/data.html.

Our dataset was carefully selected from various datasets to avoid possible bias in any single data collection and labeling method. We incorporated hate speech messages which contain *target* words and those which do not. All of these efforts were made to avoid the possibilities that a model only learned to identify particular words or particular hashtags. As the world is evolving, word meanings can sometimes be ambiguous. A message containing certain words may not necessarily be absolute hate speech [2]. Overall, our dataset consists of 16,703 hate speech messages and 83,515 legitimate messages as presented in Table 2. Note that all of our datasets are related to Twitter. We chose these datasets as they are easy to trace and verify. However, our framework is not designed for only Twitter but designed for any online social system because it only requires a text message without any other additional information (e.g., user profile, social network, temporal/activity information) for the hate speech detection.

5.2 Preprocessing

We preprocessed the collected dataset in the following ways to provide cleaner data to our framework as well as to guarantee the capability of generalizing our model in any online social media:

- As mentioned in Section 4.1, we removed all punctuations and irrelevant characters. We converted all letters to lower case.
- Privacy information such as user mention (i.e., @username) was substituted with "USER". Domain-specific labels such as hashtags had their starting character removed (e.g., for Twitter, hashtags start with '#') and content kept. Specific website links are kept anonymous as "URL". In this way, we guarantee the generalized ability of models.

5.3 Embeddings

As we mentioned in the previous section, three types of embeddings were generated in our framework: phonetic-level embedding, character-level embedding, and word-level embedding.

5.3.1 ***Phonetic-level embedding.*** We trained our domain-specific phonetic-level embedding in the following way:

(1) ***Data Collection.*** We randomly collected 80,000 tweets from Twitter API for training our embeddings and further guaranteed that these tweets do not have any overlap with the

Table 3: Baseline Information.

Model	Domain Specific	Deep Learning	Machine Learning	Description
Davidson'17	✓		✓	Linear SVC trained on a combination of useful handcrafted features.
Text-CNN'14		✓		CNN-based model with dynamic window size for text classification.
Badjatiya'17	✓	✓		LSTM-based network designed for hate speech detection.
Waseem'16	✓		✓	Logistic regression model trained on n-gram counting-based features.
Zhang'18	✓	✓		CNN followed by GRU, then fully connected layers for classification.
Fermi'19	✓	✓	✓	USE for sentence embedding, then SVM for classification.
DirectBert'19		✓		Pooling BERT embeddings into sentence embeddings, concatenate with MLP.

experiment dataset. These tweets then went through the same data preprocessing procedure described in Section 5.2. Among these tweets, there were 40,000+ unique words.

(2) ***Known Word Pronunciation Conversion.*** We used the CMU pronouncing dictionary to translate characters in each word into phonetic symbols. Such a dictionary can only handle known words in a given fixed size vocabulary.
(3) ***Unknown Word Pronunciation Prediction.*** To overcome the limitation of the CMU pronunciation dictionary and extract symbolic (phonetic) representation of unknown words, the known words' sequences of symbols were fed to an attentive LSTM model[9] for training. Eventually, the model can predict any given unknown word's symbolic representation.
(4) ***Embedding.*** Given any word's symbolic representation, we trained the embedding with the same design of Word2Vec CBOW method [18]. Thus eventually each word is embedded into a 2D matrix, with its height as the number of phonetic symbols and the width as the vector dimension.

5.3.2 ***Character-level embedding.*** We also trained a domain-specific character-level embedding in the following way:

(1) ***Data Collection.*** We used the same dataset collected and used for the phonetic-level embedding.
(2) ***Embedding.*** We trained our own character-level embedding model to directly predict embedding using the same design of Word2Vec CBOW method, thus getting a 2D-Matrix similar to phonetic-level embedding.

5.3.3 ***Word-level embedding.*** To understand which word-level embedding under our framework performs the best, we applied two popular word embedding models: FastText and BERT. FastText is a character-based embedding model trained by large corpus of data, which is capturing more subword information. BERT [14] is a context related model that achieved outstanding performance in many NLP tasks. BERT also used a word-piece tokenizer, which enabled it to capture subword information effectively and is thus intuitively less vulnerable to character-level adversarial attacks.

5.4 Baselines

To compare the performance of our model against baselines, we chose the following **7** state-of-the-art baselines as shown in Table 3:

Davidson'17 [13]: This model used the linear support vector classifier trained by TPO features (i.e., TF-IDF and POS), and other text features such as sentiment scores, readability scores and count indicators for # of mentions and # of hashtags, etc.

Text-CNN'14 [25]: Inspired by Kim's work, we implemented Text-CNN for hate speech detection trained by using GloVe as default word embedding. This is a general purpose classification framework widely applied in many text classification tasks.

Badjatiya'17 [3]: It is a domain specific LSTM-based network for detecting hate speeches.

Waseem'16 [47]: This baseline is a logistic regression model trained on the bag of words features.

Zhang'18 [50]: The authors of this work proposed C-GRU model, which combines CNN and GRU to not only fit in small-scale data but also accelerate the whole progress. Such framework is domain specific to hate speech detection.

Fermi'19 [23]: *Fermi* was proposed for Task 5 of SemEval-2019: HatEval: Multilingual Detection of Hate Speech Against Immigrants and Women on Twitter. The authors participated in the subtask A for English and ranked the first in the evaluation on the test set. Their model used pretrained Universal Encoder sentence embeddings for transforming the input, and SVM for classification.

Directly using BERT for sentence encoding (DirectBERT'19): As an additional baseline, we applied BERT in directly generating sentence encodings with two linear projection layers and dropout. We used REDUCE_MEAN, which takes the average of the hidden state of the encoding layer on the time axis for pooling strategy. It maps word-piece embeddings to the whole sentence embedding.

5.5 Experiment Setting

We randomly split our dataset into 80% training, 10% validation, and 10% test sets. For subword information embeddings, we chose the number of dimensions of phonetic-level embedding as 20, and the number of dimensions of character-level embedding as 20. For word-level embeddings, the pre-trained FastText and BERT_Base had 300 dimensions and 768 dimensions, respectively. To ensure consistency of all deep learning models (including baselines), we manually fixed the batch size as 128. Other than that, we applied grid search for determining the best hyperparameters for all models (including baselines). All weights/parameters of all models were fine-tuned to achieve each one's best result. We used ReLU as activation functions, cross-entropy as loss measurement, and Adam as the optimizer.

In the character-level attack scenario, all manipulated misspelling was created in only the test set, while we kept the training and validation sets without any change, avoiding any model seeing or remembering the adversarial attack. The proportion of manipulated data varies from 0% to 100%, at a step size of 10%. We report the result of them in Section 5.6.2.

5.6 Experimental Results

5.6.1 ***Performance under no adversarial attack.*** Table 4 shows performance of our models (*SWE2 w/ BERT* and *SWE2 w/ FastText*) and baselines under no-attack scenario. Our models outperformed

[9] https://github.com/repp/big-phoney

Table 4: Performance of our SWE2 models and baselines without the adversarial attack.

MODEL	Overall Acc.	Macro F1	Leg. F1	Hate S. F1
Davidson'17	.904	.764	.946	.583
Text-CNN'14	.935	.894	.960	.829
Waseem'16	.950	.913	.970	.857
Zhang'18	.957	.927	.974	.879
Badjatiya'17	.933	.892	.959	.826
Fermi'19 SVM	.821	.740	.885	.595
DirectBERT'19	.942	.902	.965	.839
SWE2 w/ BERT	**.975**	**.953**	**.985**	**.921**
SWE2 w/ FastText5	.974	.950	.984	.915

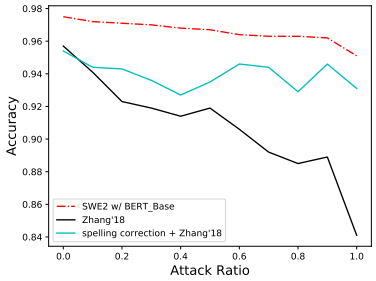

Figure 3: Accuracy of our SWE2 model and the best baseline under the adversarial attack.

all baseline models. In particular, *SWE2 w/ BERT* achieved 0.975 accuracy and 0.953 macro F1. The macro F1 balances the performance of both classes as the dataset is not balanced.

In addition, one would observe that baseline models' performance in Hate Speech class is not as good as the Legitimate Class (at least 10% gap between F1 in two classes). This is also reported in [47], as they found detecting offensive language and legitimate speech is easier than hate speech detection in their experiments. Our model, however, did exceptionally well in the Hate Speech class in terms of F1 score. Especially, our *SWE2 w/ BERT* achieved 0.921 F1, improving 4.8% compared with the best baseline (i.e, *Zhang'18*). Although *SWE2 w/ BERT* is slightly better than *SWE2 w/ FastText*, the performance of different embeddings does not differ much, showing contribution of our overall framework design.

5.6.2 ***Performance under the adversarial attack.*** To measure the robustness of our model under the character-level adversarial attack, we change an attack ratio from 0 to 1 (i.e., manipulating 0% to 100% messages in the test set). For example, 0.1 attack ratio means 10% of hate speeches and 10% of legitimate messages are manipulated by the attack model as described in Section 4.2. We intentionally manipulated both hate speeches and legitimate messages to avoid possible bias that models learned to judge hate speech based on a higher ratio of unknown words rather than understanding actual hate speech content (manipulation actually changed the spellings of words, so they may lead to out of vocabulary (OOV) cases for some embeddings). Another important reason is that as we reported earlier, even legitimate users often make typos. We are interested to push the case to an extreme, to see whether models are still capable to tell the difference between legitimate speeches against hate speeches. As *Zhang'18* was the best baseline in the previous experiment, we compare the performance of our model (*SWE2 w/ BERT*) against it. In addition, we also applied auxiliary spelling correction tools [5] as an preprocessing technique into *Zhang'18* to check the effectiveness of spelling correction under the adversarial attack.

Table 5: Performance of ablation study.

MODEL	Attack 0%		Attack 50%	
	Acc.	Macro F1	Acc.	Macro F1
SWE2 w/ BERT	.975	.953	.966	.934
−Char	.959	.928	.956	.923
−Pho	.960	.931	958	.926
−Char&Pho	.957	.923	.956	.923
−LSTMs	.940	.863	.915	.821

Figure 3 shows accuracy of our model and the best baseline under the adversarial attack (macro F1 has almost same pattern). We observe that our model consistently performed well under the adversarial attack. In particular, in the 50% (0.5) attack ratio, our model still achieved high performance, returning 0.967 accuracy and 0.934 macro F1 score while *Zhang'18* produced 0.919 accuracy and 0.887 macro F1. Even under 100% attack ratio (very extreme scenario), our model achieved 0.951 accuracy and 0.902 macro F1, while *Zhang'18* reached 0.841 accuracy and 0.783 macro F1. The performance of the baseline dropped rapidly as we increased an attack ratio. Another observation is that spelling correction in general helped mitigating character level attacks to *Zhang'18* but its performance was not stable. The spelling correction helped partly with some misspellings while not working in others. For example, given 'this how you brign the city out url' text, it was then wrongly recovered as 'this how you brain the city out url' (i.e., recovered 'brign' into 'brain' instead of 'bring'). Overall, our model still outperformed *Zhang'18* with and without spelling correction. We also tried manipulating only hate speeches in the test set without manipulating legitimate speeches. In the experiment, we saw the similar pattern in which our model consistently and robustly performed well, but the best baselines rapidly dropped its performance as increasing an attack ratio. These experiments confirm robustness of our model under the attack scenario.

5.6.3 ***Ablation Study.*** We show the results of the ablation study in Table 5. We used the BERT_Base as the default word embedding method and tested the model's performance without certain parts to see their contribution. We show results of removing *target* word's character embedding, phonetic embedding, both of them (i.e., no explicit *target* word information is given), or our two two-layer LSTMs (i.e., no explicit global information and no implicit prediction for the *target* word. It also means we remove/not use all words except the *target* word in a message). In running these experiments, we not only show each part of embeddings' contribution but also examine the actual effect of the located *target* word. Note that we only show the results of 0% attack (i.e., no attack) and 50% attack due to the limited space, but results of all other ratios are similar.

Under no attack scenario, character-level *target* word embedding contributed 1.6% accuracy and 2.5% macro F1 improvement. Phonetic-level *target* word embedding contributed 1.5% accuracy and 2.2% macro F1 improvement. Our LSTM architecture contributed 3.5% accuracy and 9% macro F1 improvement. Under the 50% attack scenario, character-level *target* word embedding contributed 1.0% accuracy and 1.1% macro F1 improvement. Phonetic-level *target* word embedding contributed 0.8% accuracy and 0.8%

Table 6: SWE2 w/ BERT under various class ratios.

Leg.:Hate S.	Overall Acc.	Macro F1	Leg. F1	Hate S. F1
1:1	.953	.953	.953	.953
2:1	.961	.956	.970	.941
3:1	.965	.953	.977	.930
4:1	.972	.958	.983	.930
5:1	.975	.953	.985	.921

macro F1 improvement. our LSTM architecture contributed 5.1% accuracy and 11.3% macro F1 improvement.

The results make sense. LSTMs function as the backbone of the framework, as losing LSTMs deprives most semantics as well as all synthetic information from the sentence. In other words, keeping only one word from a sentence and eliminating the other words are not sufficient to judge whether a speech is legitimate or not. On the other hand, dropping the target word would deprive the advantage of our framework, so it makes our model's accuracy lower. Overall, all of the components in our framework positively contributed in terms of improving the performance and keeping the robustness under the adversarial attack.

5.7 Varying a class ratio

To understand how a class ratio in the dataset affects performance of our model (as an additional experiment to measure robustness of our model), we varied a class ratio of the dataset, by downsampling legitimate messages. In particular, we tested a ratio from 1:1 to 5:1. 5:1 ratio means the original dataset without downsampling.

Table 6 shows performance of our model under various class ratios. As we increase the number of legitimate messages (from 1:1 to 5:1), accuracy has increased but macro F1 has been consistent. The result makes sense since our models were trained with different respective class weights, depending on a class ratio (i.e., if more legitimate messages are in a dataset, the model will try not to misclassify the legitimate messages in the training process). The consistent macro F1 indicates the robustness of our model/framework regardless of a class ratio. In practice, since there would be more legitimate messages, our model would achieve better accuracy and similar macro F1.

6 DISCUSSION AND ANALYSIS

6.1 How many hate speech words do hate speeches contain?

We further investigate the hate speeches distribution to answer the following questions:

- Do all hate speeches contain some hate speech words?
- As our framework puts extra emphasis on one most significant word (i.e., *target* word) in each speech/message, can it handle speeches with no hate speech word or speeches with multiple hate speech words?

We used the hate speech keyword dictionary provided by [16] to identify hate speech words in hate speeches of our dataset. Out of 16,703 hate speeches in our dataset, 12,378 hate speeches contained hate speech words. Among them, 11,835 hate speeches contained only one hate speech word; 443 hate speeches contained multiple hate speech words. The remaining 4,425 hate speeches did not contain any hate speech word.

Our *SWE2 w/ BERT* detected 69.0%, 99.1%, 100% correctly for hate speeches without hate speech word, hate speeches with single hate speech word, and hate speeches with multiple hate speech words, respectively. Although it is generally hard to detect hate speeches without any specific hate speech word, our model still reasonably identified them. On the other side, our model did exceptionally well in detecting hate speeches with one or multiple hate speech words, indicating that our model is effective in detecting hate speeches.

6.2 Why choose the most significant word?

We reason the choice of focusing the one most significant word with the following facts from different perspectives:

- **Only focusing on random words does not work.** Previous work in [27] showed that random attack on words is not successful. A successful attack has to choose certain words rather than randomly chosen words to attack. Consequently, focusing on random words to defend attacks is also not useful.
- **The most important word contributes most in both sentiment and semantic meaning of the message.** Figure 4 shows how the most significant word almost dominated a sentiment score of each message. In addition, we observe that some hate speech posters tend to manipulate the important word to evade existing detection approaches.
- **Focusing on one single most important word succeeded with detecting hate speech with multiple hate speech words or without hate speech word.** As we analyzed before, there are many hate speeches with multiple hate speech words in the dataset, and also over four thousand hate speeches without any specific hate speech word. However, the strategy of focusing on the single most important word still succeeded to detect hate speeches correctly overall.
- **We show a real evidence in Case Study, indicating the necessity of the focusing strategy.** Section 6.5 shows a real hate speech that was correctly predicted by our model, but the best baseline misclassified it to a legitimate message because only the most important word in the message was strongly negative while the other words were positive. The best baseline failed to put extra emphasis on the most important word. Thus other words' meanings mitigated the impact of the *target* word, eventually leading to misclassification.

6.3 How is sentiment of a message changed under character-level manipulation?

We try to reason and give out evidence of why sentiment-based features are vulnerable against character-level manipulation/adversarial attack. As sentiment-based features' atomized elements are words, the manipulated words lost their sentiment significance. Thus, it will affect the actual sentiment measure of each message.

To prove the correctness of the hypothesis, we used VADER [22] as the sentiment analysis tool and showed a sentiment change between before and after the character-level manipulation in Figure 4. For each given message, we generate the compound sentiment score generated by the VADER. Such a score varies in a range of $[-1, 1]$. The closer the score gets to 1 indicates the message is more sentimentally positive, while the closer the score gets to *-1* means the message is more sentimentally negative. In the left subfigure,

we show sentiment score distribution of hate speech and legitimate messages before applying the character-level manipulation in our dataset. The curve reflects the kernel density estimation of the given population distribution observations. In the right subfigure, we show the sentiment score distribution of hate speech and legitimate messages after applying the character-level manipulation.

We observe that sentiment based features contribute to separating hate speech against legitimate speech under no attack/manipulation in the left subfigure. However, they lost power when manipulations occur in the right subfigure. This result explains why the baselines using sentiment features (e.g., *Davidson'17*) experienced a performance drop when they face the character-level adversarial attack. This figure also points out two interesting facts: 1) The sentiment does not change much for legitimate speech with the manipulated misspellings; and 2) The most significant word contributes the most in each speech's overall sentiment score.

6.4 Humans vs. machine learning models under the character-level adversarial attack

Why can humans still correctly perceive the meaning of a message with typos, but machine learning models consider it as a hard problem[10]? The way humans memorize words is different from machine learning models in three ways:

- Humans' reading is sequential, and they can tolerate some errors/typos[11], while usually machine learning models compare strings of characters, so the result is strictly boolean, which excludes any tolerance.
- When reading, humans encode the information they receive mainly from eyes, graphically recognizing words. Thus they can recognize similar characters[12] such as 's' and '$', or 'l' and '1'. While inside the machine learning models, every character is encoded differently and independently, thus there is no such correlation between different characters.
- Humans incorporate more side information than machine learning models during the reading and memorizing. They not only read words quietly but also explicitly or unintentionally associate pronunciations with the words [44]. Thus, the words 'jews' and 'jooz' can be recognized as if they are the same thing even when we never actually learned 'jooz' before. However, if machine learning models do not have the term 'jooz' in their vocabulary, they would treat the word as an unknown word, failing to extract useful information from it.

Even though some hate speech messages are manipulated, the ultimate goal of these hate speech messages will never be changed. In other words, they have to be understood by humans, to make an actual impact on society. Based on the reasoning and analysis, in our models, we incorporated both word-level and subword level information not only to enrich the content of a message but also help them better recognize/recover the original meaning of the message. Our experimental results in the previous section confirmed the effectiveness of our framework and models, achieving high hate speech detection accuracy and F1.

[10]https://en.wikipedia.org/wiki/Typoglycemia
[11]https://bit.ly/33zupxw and https://bit.ly/3idUXIR
[12]https://bit.ly/33wCHX9

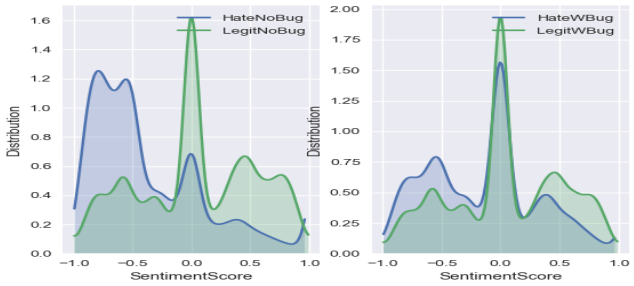

Figure 4: Sentiment score before and after the character-level manipulation.

6.5 Case Study

In this subsection, we conduct a micro-scale case study, where we show examples[13] that our models correctly predicted and that our models misclassified. By doing this analysis, we can shed some light on designing better hate speech detection approaches. For convenience, we compare our *SWE2 w/ BERT* against the best performing baseline – *Zhang'18*.

6.5.1 ***Advantage Analysis.*** We first show two examples randomly sampled from instances that were correctly predicted by our model, while the best baseline misclassified it.

- An example without the adversarial attack: ***"... win the faggot award congrats ..."***

When we read this message, we can find that all the words except 'faggot' are positive words. Thus, the baseline does not emphasize the most important word, 'faggot', which does not have enough influence on the overall sentiment score. Thus, its negative impact is mitigated by its surrounding positive words, leading to misclassification. However, in our model, 'faggot', is recognized as an important keyword. It puts extra attention and emphasis on the important word. The character-level information is captured by our model and further guarantees correct prediction.

- A real-world example before applying the adversarial attack: ***"... look like a redneck ... confederate flag tattoo on ... ass."***
- The same example after applying the adversarial attack: ***"... look like a rednecj ... confederate flag tattoo on ... ass."***

By comparing the above examples, we noticed that 'rednecj' was manipulated from 'redneck' by the *sub-c* manipulation. In the baseline, this misspelled word is roughly identified as an unknown word. However, our model still learns features from it by using character embedding and phonetic embedding, mimicking how humans read it as described in Section 6.4.

6.5.2 ***Error Analysis.*** Next, we show an example message that even our model did not correctly identify.

- ***"... fucking hate you ... but thank you ... dick van dyke."***

The example was manually labeled as a legitimate tweet [46] but was misclassified by both of our model and the best baseline. The message itself contains several aggressive and sentimentally negative words such as 'fucking' and 'dick'. Besides, it has 'dyke', which is listed as a hate speech keyword. However, the real meaning is to express thanks in a joking way. It is still challenging for our

[13]All examples in the case study belong to the public datasets released by other researchers described in Section 5.

model to further deeply understand and detect the text with complicated sentiment and mood swings. Using better domain-specific embedding may help a model to predict its label correctly.

7 CONCLUSION

In this paper, we proposed a novel hate speech detection framework that incorporates subword information and word-level semantic information. By addressing the importance of successfully identifying manipulated words and focusing on the most significant word in the message, and by using attention mechanisms to extract side information, our models outperformed all 7 state-of-the-art baselines. Under no attack, our *SWE2 /w BERT* achieved 0.975 accuracy and 0.953 macro F1, and under 50% attack, our *SWE2 /w BERT* still achieved 0.967 accuracy and 0.934 macro F1, showing effectiveness and robustness of our model.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant CNS-1755536, AWS Cloud Credits for Research, and Google Cloud.

REFERENCES

[1] Aymé Arango, Jorge Pérez, and Barbara Poblete. 2019. Hate Speech Detection is Not as Easy as You May Think: A Closer Look at Model Validation. In *SIGIR*.
[2] Pinkesh Badjatiya, Manish Gupta, and Vasudeva Varma. 2019. Stereotypical Bias Removal for Hate Speech Detection Task using Knowledge-based Generalizations. In *WWW*.
[3] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. Deep learning for hate speech detection in tweets. In *WWW*.
[4] Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2015. Improved Transition-based Parsing by Modeling Characters instead of Words with LSTMs. In *EMNLP*.
[5] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis. In *SemEval-2017*.
[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.
[7] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).
[8] Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015. Joint learning of character and word embeddings. In *IJCAI*.
[9] Arijit Ghosh Chowdhury, Aniket Didolkar, Ramit Sawhney, and Rajiv Shah. 2019. ARHNet-Leveraging Community Interaction for Detection of Religious Hate Speech in Arabic. In *ACL Student Research Workshop*.
[10] Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *ACL*.
[11] Yi-Ling Chung, Elizaveta Kuzmenko, Serra Sinem Tekiroglu, and Marco Guerini. 2019. CONAN-COunter NArratives through Nichesourcing: a Multilingual Dataset of Responses to Fight Online Hate Speech. In *ACL*.
[12] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *EMNLP*.
[13] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *ICWSM*.
[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.
[15] Nemanja Djuric, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavljevic, and Narayan Bhamidipati. 2015. Hate speech detection with comment embeddings. In *WWW*.
[16] Mai ElSherief, Vivek Kulkarni, Dana Nguyen, William Yang Wang, and Elizabeth Belding. 2018. Hate lingo: A target-based linguistic analysis of hate speech in social media. In *ICWSM*.
[17] Björn Gambäck and Utpal Kumar Sikdar. 2017. Using convolutional neural networks to classify hate-speech. In *First workshop on abusive language online*.
[18] Fréderic Godin. 2019. *Improving and Interpreting Neural Networks for Word-Level Prediction Tasks in Natural Language Processing*. Ph.D. Dissertation. Ghent University, Belgium.
[19] Hongyu Gong, Yuchen Li, Suma Bhat, and Pramod Viswanath. 2019. Context-sensitive malicious spelling error correction. In *WWW*.
[20] Tommi Gröndahl, Luca Pajola, Mika Juuti, Mauro Conti, and N Asokan. 2018. All You Need is "Love" Evading Hate Speech Detection. In *AISEC*.
[21] Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. In *ACL*.
[22] Clayton J Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *ICWSM*.
[23] Vijayasaradhi Indurthi, Bakhtiyar Syed, Manish Shrivastava, Nikhil Chakravartula, Manish Gupta, and Vasudeva Varma. 2019. Fermi at semeval-2019 task 5: Using sentence embeddings to identify hate speech against immigrants and women in twitter. In *SemEval*.
[24] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).
[25] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*.
[26] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NIPS*.
[27] J Li, S Ji, T Du, B Li, and T Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *NDSS Symposium*.
[28] Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. 2015. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586* (2015).
[29] Han Liu, Pete Burnap, Wafa Alorainy, and Matthew L Williams. 2019. Fuzzy Multi-task Learning for Hate Speech Type Identification. In *WWW*.
[30] Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *ICLR*.
[31] Binny Mathew, Punyajoy Saha, Hardik Tharad, Subham Rajgaria, Prajwal Singhania, Suman Kalyan Maity, Pawan Goyal, and Animesh Mukherjee. 2019. Thou shalt not hate: Countering online hate speech. In *ICWSM*.
[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[33] Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *WWW*.
[34] Nedjma Ousidhoum, Zizheng Lin, Hongming Zhang, Yangqiu Song, and Dit-Yan Yeung. 2019. Multilingual and Multi-Aspect Hate Speech Analysis. In *EMNLP-IJCNLP*.
[35] Haiyun Peng, Yukun Ma, Soujanya Poria, Yang Li, and Erik Cambria. 2019. Phonetic-enriched Text Representation for Chinese Sentiment Analysis with Reinforcement Learning. *arXiv preprint arXiv:1901.07880* (2019).
[36] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.
[37] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.
[38] Andreas Rücklé, Steffen Eger, Maxime Peyrard, and Iryna Gurevych. 2018. Concatenated power mean word embeddings as universal cross-lingual sentence representations. *arXiv preprint arXiv:1803.01400* (2018).
[39] Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *ICML*.
[40] Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. In *SocialNLP-W*.
[41] Joan Serra, Ilias Leontiadis, Dimitris Spathis, J Blackburn, G Stringhini, and Athena Vakali. 2017. Class-based prediction errors to detect hate speech with out-of-vocabulary words. In *Abusive Language Workshop*.
[42] Richard Sproat, Alan W Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer speech & language* 15, 3 (2001), 287–333.
[43] Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J Pal. 2018. Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning. In *ICLR*.
[44] K TOUTANOVA. 2001. Pronunciation modeling for improved spelling correction. In *ACL*.
[45] William Warner and Julia Hirschberg. 2012. Detecting hate speech on the world wide web. In *Second workshop on language in social media*.
[46] Zeerak Waseem. 2016. Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In *NLP+CSS*.
[47] Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *NAACL Student Research Workshop*.
[48] Byunggon Yang. 2016. Phoneme distribution and syllable structure of entry words in the Carnegie Mellon University Pronouncing Dictionary. *The Journal of the Acoustical Society of America* 140, 4 (2016), 3113–3113.
[49] Byung-Gon Yang. 2012. Reduction and frequency analyses of vowels and consonants in the Buckeye Speech Corpus. *Phonetics and Speech Sciences* 4, 3 (2012), 75–83.
[50] Ziqi Zhang, David Robinson, and Jonathan Tepper. 2018. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *ESWC*.