Paper 4: Final Paper
SemEval-2019 Task 6: Identifying Offensive Language through Embeddings and Classifiers

Gareth Harcombe
University of Canterbury
33186156
`grh102@uclive.ac.nz`

Abstract

This document contains a system evaluation for the shared task SemEval-2019 Task 6 (OffensEval) to identify offensive tweets from English tweets. This paper evaluates Bidirectional Encoder Representation from Transformer (BERT) and Skip-gram with Negative Sampling (SGNS) with different classification and embedding pooling methods to investigate what makes BERT successful on sub-task A of OffensEval. Although the performance of the systems discussed did not match the performance by the best teams using BERT, we found that the classification method used does not lead to a significant increase in performance, and that pooling methods can be inconsistent between embedding methods.

1 Introduction

Offensive language in social media is becoming increasingly prevalent, and it is desirable for the companies managing these sites to remove offensive language so as not to offend other users on the platform. Manual identification of offensive language is time-consuming, and so it is desirable to automate the process. Zampieri et al. (2019b) SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval) is a shared task that seeks to identify offensive language within a short piece of text from social media such as a tweet. The shared task has 3 sub-tasks: identifying offensive language; breaking down what category of offensive language is used; and who the offensive language is directed at. This system evaluation will only consider sub-task A, offensive language identification, in order to simplify model comparison. Sub-task A describes a one-label problem: to label each document as either offensive or not offensive. This task has been dominated by BERT models, with the top five teams all using a variation of BERT. This raises the question of what aspect of BERT leads to its high performance, the class token `[CLS]`, or the ability to generate contextual embeddings? Furthermore, to what extent does the classification network attached to BERT influence the model's performance? This system evaluation seeks to answer these questions by comparing the performance of BERT using the class token `[CLS]` against the raw embeddings, with different classification methods such as a linear classifier against a Long Short-Term Memory network classifier. The performance of embeddings produced by BERT will be compared against a baseline embedding model of Skip-Gram with Negative Sampling (SGNS), and the performance of other teams in OffensEval.

2 Related Works

Bidirectional Encoder Representation from Transformer (BERT) was developed by Devlin et al. (2019) and has proved to be accurate at many tasks, including this task. The *NULI* team (Liu et al., 2019) describes their approach using BERT with default parameters and a max sentence length of 64 and trained for 2 epochs. With this approach, *NULI* ranked first place in sub-task A with an F1 score of 0.829, 4th in sub-task B with an F1 score of 0.716, and 18th in sub-task C with an F1 score of 0.560. More generally, the top five teams for sub-task A used a BERT model. This clearly shows the dominance and success that BERT models have achieved on sub-task A.

However, BERT dominance was not completely transferred to sub-tasks B and C, whereby five of the top ten teams in each of the sub-tasks used ensemble methods. These included ensembles of deep neural networks such as Convolutional Neural Networks (CNNs), Bidirectional Long Short Term Memory and Bidirectional Gated Recurrent Unit (BLSTM+BGRU), and BERT by Team *MIDAS*

(Mahata et al., 2019) and Team *NLPR@SRPOL* (Seganti et al., 2019), but also non-neural machine learning techniques such as Random Forests, Naive Bayes and Support Vector Machine (SVM) by Team *vradivchev_anikolov* (Nikolov and Radivchev, 2019) and *NLPR@SRPOL* (Seganti et al., 2019).

The best team on sub-task B, *jhan014* achieved an F1 score of 0.755 using a rule-based approach (Han et al., 2019). This approach identified offensive words within the given tweet and the syntactic relationship of words to any offensive words within the tweet. This information was then combined into a probability of a tweet being offensive. Han et al. (2019) shows that handcrafted solutions can still outperform state-of-the-art models such as BERT.

3 Data

OffensEval supplies an annotated dataset for training and evaluation. The dataset contains an imbalance of offensive and non-offensive tweets.

3.1 Shared Task

This shared task uses English tweets from the Offensive Language Identification Dataset (OLID) dataset, presented by Zampieri et al. (2019a). The OLID dataset includes 14,100 annotated tweets, with 13,240 tweets used for training and 860 tweets used for testing. Two examples of tweets with labels from the dataset are found in Table 1.

3.2 Annotation

OLID's annotation scheme is a hierarchical scheme with three levels to include the target of the offensive language and the type of offensive language (Zampieri et al., 2019a). The first level is to classify offensive (OFF) and non-offensive (NOT) tweets. Offensive tweets can include profanity, and any kind of insult or threat, regardless of whether it is direct or veiled.

If the tweet is labelled as offensive, then the second level of annotation is whether the offensive language is targeted or not. This is a binary classification of whether the post includes targeted insults or threats to a group or individual, or whether the post contains profanity.

If the tweet contains targeted insults or threats, then the third level of annotation focuses on the target of the offensive language. There are three labels to this sub-task: individual, such as a famous person, named or unnamed individual; group, such as ethnicity, gender, or religion; and other, such as an organisation, event, or issue.

This evaluation focuses on labels from the first level, which annotates each document as either offensive (OFF) or not offensive (NOT).

Tweets were annotated using the crowd-sourcing platform Figure Eight[1]. In order to used reliable and accurate annotators, only annotators who were experienced and passed a threshold of test queries were used. All tweets were annotated by two annotators, with a third annotation and a majority vote being used in the event of a disagreement.

3.3 Class Imbalance

There is a significant class imbalance in the OLID data set, as the number of non-offensive documents outnumbers the number of non-offensive documents. 66.7% and 72.1% of the training and testing data respectively is labelled as not offensive. This is somewhat representative of all tweets, as the majority of tweets are not going to be offensive. However, this does also mean that a system can achieve reasonable level of accuracy by simply choosing the most common label, not offensive. In doing so, the system will not detect any offensive language, voiding the overall purpose of the shared task. To avoid this issue, the standard metric to report for this task is the macro-averaged F1 score to measure precision and recall as well as accuracy. This presents a more accurate representation of how well the system performs at detecting offensive language.

4 Embedding Systems

To provide an embedding system to compare BERT against, we develop both BERT and SGNS based models to evaluate on OffensEval.

4.1 SGNS

Skip-Gram with Negative Sampling is an embedding method developed by (Mikolov et al., 2013) that attempts to predict the surrounding words in a window given the current word. This is done by considering the current word and the neighbouring context words as positive examples, and randomly sampling other words from the vocabulary as negative examples, and distinguishing between the true context words and the negative samples. This is done by using logistic regression to train a classifier to determine whether words are commonly found

[1]https://figure-eight.com

Tweet	Sub-task A Label
@USER He is so generous with his offers.	NOT
IM FREEEEE!!!! WORST EXPERIENCE OF MY FUCKING LIFE	OFF

Table 1: Two tweets from the OLID dataset with labels for sub-task A

together in the positive examples, or whether they are not commonly found together in the negative examples. The weights of the classifier for each word then results in the embedding for each word. Negative sampling improves the efficiency of training the model, as the model parameters are only updated for a few negative samples instead of for the entire vocabulary.

4.2 BERT

BERT is a transformer-based model. The transformer architecture was developed by Vaswani et al. (2017) and uses only attention heads, removing Recurrent Neural Network and Convolutional Neural Network aspects of model architectures proceeding it.

The transformer architecture is comprised of multiple attention layers. Each attention head combines several vectors: Q, the query or current focus position; K, the preceding input; and V, the context vector which is used to compute Q. Combined, these predict the relationships between every word and the preceding word. In a bidirectional architecture, this becomes the prediction of the relationship between every word and every other word in the context window. The output from all attention heads is concatenated and fed through a linear layer to project the concatenated outputs into the original input dimension. Combined, this forms one attention layer.

Each layer of attention heads is connected to the previous layer and the original input. This introduces addition redundancy, whereby if one layer fails, the next layer may have more success, and prevents cascading error propagation.

The output of all attention layers is then fed through a final linear layer, and then through a softmax layer to output the probabilities for each word in the vocabulary.

Devlin et al. (2019) use the transformer architecture to build Bidirectional Encoder Representations from Transformers (BERT). The model is pre-trained using a masked language model objective, randomly masking tokens in the input and predicting the original token. BERT then outputs embedding vectors for all the tokens in the input sequence.

There are two sizes of BERT model, Base and Large. This implementation uses the Base model, with 12 layers, 12 attention heads, and 110 million total parameters (Devlin et al., 2019). The Base model was chosen to help prevent overfitting.

BERT has a specialised tokeniser for tasks. There are two special tokens that it adds to the input: the `[CLS]` or classification token, which is the start of every sequence; and the `[SEP]` token to determine which tokens belong to each sentence/context. In this problem, the `[CLS]` token will always be at the start of the sequence, and the `[SEP]` token will always be at the end of the sequence. The input for BERT is of fixed length, and so input sequences are either truncated to fit within the maximum length, or padded with `[PAD]` tokens.

The max sentence length was set to the default of 512. The model was fine-tuned for four epochs, with more training cycles resulting in overfitting and decreasing test error.

5 Classification Systems

The output from the embedding models needs to be processed for use in a classification problem. This can be done by passing the output from the embedding model into a neural network. All embedding methods produce an embedding for each token, which can be pooled or input directly into the classification model. BERT embeddings also include the `[CLS]` token, which can be used by itself for the classification task.

5.1 Linear Classifier

A linear classifier is a simple way of transforming input embeddings into class probabilities. A linear layer[2], also referred to as a fully connected layer or a dense layer, is used to connect every entry in the input embedding vector to two output nodes. This connection is in the form of applying a linear transformation to the input, multiplying by a weight

[2]`https://pytorch.org/docs/stable/generated/torch.nn.Linear.html`

and adding a bias. The network is trained using a version of stochastic gradient descent to minimise the cross entropy loss shown in Equation 1 between the actual class label, y_i, and the predicted class probability, p_i.

$$L = -(y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)) \quad (1)$$

Linear classifiers cannot take a variable number of inputs. Furthermore, concatenating many embeddings together to form the input for a linear classifier can result in a drastically increased classifier complexity. Hence, it is common to pool the embeddings into one single vector. There are several ways of doing this, such as taking the sum over all vectors, averaging over all vectors, or the maximum value over all vectors, for each dimension. For all of these pooling methods, `[PAD]`, `[CLS]` and `[SEP]` are ignored from the calculations.

When using BERT's `[CLS]` for classification, there is no pooling method as only the one token is used for the tweet's classification. In this case, all token embedding vectors are discarded except for the `[CLS]` token embedding, as described by Sun et al. (2020).

5.2 Recurrent Neural Network Classifier

Recurrent Neural Networks (RNNs) take the output of the previous token as an input for the current token. As a result, RNNs can take an input of any length, and so do not require any pooling methods and can take the full sequence of embeddings.

An example of an RNN is the Bidirectional Long Short Term Memory (Bi+LSTM) network (Sepp Hochreiter, 1997). An LSTM input takes the current input x_t, the previous hidden state h_{t-1}, and the previous context c_{t-1}, to produce a new hidden state h_t, and an updated context c_t (Staudemeyer and Morris, 2019). The LSTM consists of gates which attempt to retain important information in long-term memory and so improve the ability of models to "remember" the context of data from earlier in the sequence, as well as reducing the vanishing gradient problem. The Bidirectional RNN means that two models are trained, one with an RNN for the forward pass, and another with an RNN for the backward pass (starting at the last token moving backward through the sequence).

6 Experiment Set-up

The code for implementing these embedding systems can be found in Appendix A.

The BERT model was imported from PyTorch-Transformers[3], along with the BERT tokeniser. BERT parameters follow those used by the *NULI* team, of default parameters and a max-sentence length of 64.

The pretrained `word2vec-google-news-300` SGNS embeddings were used for evaluation. The embeddings have dimension 300, with a vocabulary of 3,000,000, and were chosen for the large corpus of relatively unbiased training data of Google News with approximately 100 billion words. The dataset is available from Gensim[4], and contains less bias to this task than other datasets such as a dataset of Twitter tweets directly, and hence provides a good baseline of non-contextual embedding performance. Additionally, these pretrained word embeddings were chosen because of the large vector size of 300. An embedding dimension of 768 to match the BERT embedding size would have allowed for a fairer comparison, but was not achievable given the computational resources available.

Text for evaluating SGNS was preprocessed by removing punctuation and numbers, and converting all text into lower case. The text was then tokenised using the NLTK `word_tokenize` function[5]. This tokeniser is based off the Treebank tokeniser, and splits text by words and splits standard contractions. For BERT systems, the default tokeniser was used.

The linear and LSTM classifiers were built using PyTorch Neural Network models[6]. The linear classifier uses PyTorch's Linear layer[7], with input feature dimension equal to the dimension of the embedding space (768 for BERT and 300 for SGNS), and output feature dimension equal to two, one for each of the OFF and NOT tags. The LSTM classifier uses PyTorch's LSTM layer[8], with input feature dimension equal to the dimension of the embedding space, and output feature dimension of 256, and bidirectional set to `True`. The output

[3]https://pytorch.org/hub/huggingface_pytorch-transformers/

[4]https://github.com/RaRe-Technologies/gensim-data

[5]https://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.word_tokenize

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

[7]https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

[8]https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

from the LSTM layer is then input into a linear layer with input dimension 512 (256 for the last hidden layer in the LSTM for each direction) and an output dimension of two.

All the classification models were trained for 2 epochs, using cross entropy loss as the loss function, with the Adam optimiser.

There is inherent randomness in training using stochastic gradient descent, and so retraining models with the same hyperparameters can lead to different model performance. In order to get an estimate of model performance without this randomness, each model was trained five times and the F1 scores averaged.

7 Results

The results of testing BERT and SGNS with different classification and pooling methods on OffensEval is shown in Table 2 and Table 3. Table 2 displays results using linear classifiers for each permutation for BERT and SGNS, and the three pooling methods used. Table 3 displays results for systems using all tokens with a LSTM classifier, and for BERT systems using the `[CLS]` token. Table 4 shows the full results of five replications, and confidence intervals for the results with $\alpha = 0.05$.

Embedding	Classifier	Pooling	F1
BERT	Linear	Average	0.7639
BERT	Linear	Sum	0.4186
BERT	Linear	Max	0.7153
SGNS	Linear	Average	0.4307
SGNS	Linear	Sum	0.4838
SGNS	Linear	Max	0.4474

Table 2: OffensEval task results with linear classifier and three pooling methods for BERT and SGNS.

Embedding	Classifier	Tokens	F1
BERT	Linear	CLS Token	0.7629
BERT	LSTM	CLS Token	0.7726
BERT	LSTM	All Tokens	0.7770
SGNS	LSTM	All Tokens	0.4036

Table 3: OffensEval task results with linear and LSTM classifiers, and which are used as input tokens.

The best performing system was BERT with LSTM classifier on all embedded tokens, with an F1 score of 77.7%. This would place the system 24-27th out of 104 entrants.

7.1 Statistical Significance

The confidence intervals in Table 4 show that few of the methods have significantly better performance.

All of the confidence intervals for the SGNS based systems with linear classifier overlap, showing no statistical significance in the results. Sum and max pooling methods with a linear classifier are significantly better than using a LSTM classifier on all tokens.

The BERT based system with linear classifier and sum pooling is significantly worse than all the other BERT systems, and performed significantly worse than SGNS with linear classifier and sum pooling.

7.2 Embedding Performance

All models using output from BERT performed significantly better than the models using SGNS, except for using the sum pooling method with BERT. This suggests that the contextual embeddings produced by BERT overall perform better than the non-contextual embeddings produced by SGNS for the OffensEval task, regardless of the pooling method or classification network used.

Comparing BERT models gives some insight into what is useful output from the BERT model. Using all of the embedded tokens available did not lead to a significant improvement in performance in either of the classification methods, and saw a significant decrease in performance under some pooling methods using a linear classifier. This implies that the `[CLS]` token sufficiently represents tweets for a classification task, and including all tokens may add additional noise or make the classification problem harder to learn.

The BERT systems performed worse than the top BERT systems discussed by Zampieri et al. (2019b). The *NULI* team (Liu et al., 2019) achieved a F1 score of 82.6%, 4.90 points higher than the best system discussed in this paper. Whilst *NULI* does not include details on how BERT outputs were translated into classifications, *NULI* did use different preprocessing techniques such as emoji substitution and hashtag segmentation that may have led to the increase in performance.

The performance of the SGNS models were only just above the baseline of labelling all tweets as offensive, showing poor performance. All of the SGNS systems would have placed behind 97th place out of 104 entrants.

Embedding	Classifier	Pooling	F1					Conf. Interval
BERT	Linear	CLS Token	0.7869	0.7483	0.7548	0.7462	0.7783	(0.7399, 0.7859)
BERT	Linear	Average	0.7540	0.7828	0.7562	0.7489	0.7778	(0.7450, 0.7829)
BERT	Linear	Sum	0.4185	0.4189	0.4181	0.4189	0.4189	(0.4182, 0.4191)
BERT	Linear	Max	0.7786	0.7848	0.8056	0.4403	0.7674	(0.5236, 0.9070)
BERT	LSTM	CLS Token	0.7662	0.7661	0.7646	0.7928	0.7729	(0.7579, 0.7872)
BERT	LSTM	All Tokens	0.7602	0.7728	0.7808	0.7863	0.7847	(0.7636, 0.7903)
SGNS	Linear	Average	0.4251	0.4257	0.4889	0.4036	0.4100	(0.3885, 0.4728)
SGNS	Linear	Sum	0.4473	0.4382	0.5254	0.4945	0.5139	(0.4351, 0.5326)
SGNS	Linear	Max	0.4512	0.4311	0.4122	0.4880	0.4546	(0.4122, 0.4827)
SGNS	LSTM	All Tokens	0.4036	0.4036	0.4036	0.4036	0.4036	(0.4036, 0.4036)

Table 4: Full OffensEval task results with different classification and pooling methods before averaging F1 scores, including confidence intervals with $\alpha = 0.05$.

7.3 Classifier Methods

The outlier to BERT's good performance is the BERT model that uses a linear classifier and sum pooling method, which achieved an F1 score of 0.4186 and performed significantly worse than all other BERT systems, and worse than the majority of SGNS systems. By inspecting the results confusion matrix, we can see that this model always predicts tweets as offensive, explaining why the F1 scores of 0.418 is so close to the All Offensive baseline of 0.420. It is not directly clear why this model always predicts tweets as offensive+. The code used to sum embeddings is the same code used to average embeddings before dividing by the number of embeddings present, suggesting that the poor performance of the sum pooling method is not due to a coding error. It may be that because a summation can produce results with a large range of values[9], the linear classifier could not learn to classify embeddings in 2 epochs. Further testing of hyperparameters and significance testing would be useful to investigate this outlier.

The two BERT systems using LSTM classifiers did not show significantly different results between using all tokens and just using the `[CLS]` token. The LSTM systems using the `[CLS]` token did show a marginal improvement over the linear system using the `[CLS]` token. This may imply that using an LSTM classifier results in a minor improvement in performance, but the difference in performance between the two methods is not significant, and would require further testing to investigate whether using LSTM results in an increase in performance.

As seen in the full results in Table 4, the BERT system with linear classifier and max pooling showed the most variance. The system achieved the highest score out of all the systems of 0.8055, but also had a very poor outlier performance of 0.4403. This may be a result of instability in max pooling, whereby a few vectors with large values can dominate and ignore the majority of other vectors, especially vectors with strongly negative values that would be taken into account by the Average and Sum methods.

There was no pooling method which showed consistently higher performance in both embedding systems. The best pooling method for SGNS was sum pooling, by a margin of 0.04, but the best pooling method for BERT was average pooling by a margin of 0.05, with sum pooling performing significantly worse than all other methods. Interestingly, sum pooling from BERT performed the worst out of all pooling methods, but sum pooling from SGNS performed the best out of all pooling methods. Combined, this suggests that choosing the best pooling method may be specific to the embedding vector space and method used, and that there is no universal best pooling method.

The SGNS embeddings showed better performance when using a linear classifier than the more complex LSTM classifier, in contrast to the BERT systems which showed slightly higher performance with LSTM classifiers.

8 Discussion

When evaluating systems, it is important to consider any issues with the shared task results, what challenges might occur when adapting them to

[9] i.e. the magnitude of a single embedding will likely be significantly smaller than the magnitude of 64 embeddings summed together

other languages, and contexts where the performance of the systems may vary.

8.1 Shared Task Details

Zampieri et al. (2019b) presents the confusion matrix for the NULI team, along with the associated macro-averaged F1 score of 0.829. However, this F1 score could not be replicated from the confusion matrix, with the macro-average F1 score from the matrix equally 0.756. This is a significant difference, and is not fixed by using different weightings for the F1 score, such as micro or class weighted. Furthermore, the confusion matrix presented in the paper for the NULI team is very similar to the confusion matrix produced by BERT systems in this paper, see Appendix B. This implies that the confusion matrix is correct, as similar models of BERT would produce similar outputs, but does not explain why the F1 scores differ, especially as there are multiple models with F1 scores above 0.800, making a typo unlikely. Additionally, the *NULI* team reported an F1 score of 0.7826 from their testing, a significant difference to the final score of 0.829. Hence, the F1 scores presented by Zampieri et al. (2019b) may not be reliable.

The confidence intervals for the majority of the systems is large, with some systems having a confidence interval width of 5% F1 score. This makes it hard to draw any meaningful conclusions from the experiment results, as almost all confidence intervals from systems using the same embedding method overlap. This could be improved by running more replications for each system, but there is inherently a large amount of variance for each system, such as the BERT with Max pooling system which has an F1 score range of 0.3653. Varying other hyperparameters such as the number of epochs or the classifier layer structure (i.e adding dropout or ReLU) could be explored to give more consistent results.

8.2 Pacific Languages

At a basic level, Pacific languages have different words, grammar and structure, and have less data available. This presents issues for the BERT model, as it has been pre-trained on English language texts including over 3 billion words before being fine-tuned for this task. In order to be trained on Pacific language, a similarly large corpus of text would be to be obtained, and would take a large amount of resources to train. The number of native speakers of Pacific languages is significantly less than those of English speaking, making it significantly harder to collate a data set in a Pacific language that is as large, if possible at all. The SGNS embeddings were trained on an even larger dataset, as the `word2vec-google-news-300` embeddings were trained on a dataset with 100 billion words. However, it is possible to train embeddings with a smaller corpus, at the cost of accuracy. As a result, in languages with significantly less training data, SGNS may perform better than BERT based systems, the opposite of the results seen in this paper where BERT outperformed almost every SGNS based system.

Adapting SGNS and BERT systems for Pacific languages may also present challenges due to the range of dialects that can exist within a language. This is less of an issue as the above challenge, as the English language already has dialects such as British and American English which do not pose major issues. However, the presence of dialects in Pacific languages will still likely require a large training corpus in order to accurately learn and classify offensive tweets regardless of dialects.

A deeper issue with adapting systems for Pacific language is that Pacific languages may have unique beliefs, customs, and values that may be reflected in their language use. Issues with different words, grammar, and dialects could theoretically be overcome by translating the Pacific tweets into English, and then processing them the same as an ordinary English tweet. However, this may not prove to be effective due to the different way that the language is used. For example, what is classified as an insult in a Pacific language may not be classified as an insult in the English language due to what is fundamentally considered to be offensive in the two languages.

We would expect that more complex classifiers such as LSTM would be able to better learn what makes a tweet offensive, including different beliefs and values of different cultures. However, this was not reflected in the results in this paper, with LSTM classifiers showing no significant advantage over linear classifiers. Further research is required to investigate if there are any tasks or situations where complex classifier methods have a significant advantage over simple classifier methods, such as whether complex classifier methods can be more effective at learning belief and value systems across different cultures. Similarly, investigating whether there are any situations where using all of BERT's embedded tokens presents a significant advantage over just using the [CLS] token, such as for long passages of text rather than short tweets used in the OffensEval shared task which did not not show a significant improvement over either method, could give more insight into the usefulness of BERT's [CLS] token compared to all the embeddings produced.

8.3 Performance in Different Context and Populations

Offensive language depends heavily on context. A comment with vulgar language directed at friends can be considered not offensive, and instead be a form of trusting joking around. However, a comment with vulgar language directed at a stranger can be considered offensive. Without the context associated with each tweet, such as whom the author is tweeting about or tweeting to, it is inherently hard to classify the tweet as offensive or not. If the context is a general tweet aimed at the public, then the systems will likely perform better, as it is unlikely that the author is making an inside joke or communicating with friends. Conversely, if the context is a tweet between associates, then the systems will likely perform worse, as it becomes more difficult to determine whether any offensive language is joking between friends or actually intended to be offensive.

Similarly, offensive language also depends on whether the individual making the comment is inside the group being offended. If an individual is part of the group targeted by the offensive language, then it is usually accepted by society to not be offensive, as otherwise the individual would be inherently offending themselves. However, if an individual is not part of the group targeted by the offensive language, it is more likely to be offensive language.

Contextual embeddings used by BERT can help reduce this problem, and may be one of the main reasons that BERT performs significantly better than the non-contextual SGNS embeddings on the OffensEval task. However, BERT still fails to identify the wider context of whom the tweet is aimed at, which would require meta data about the tweet such as who sent and received the tweet, and connection between the sender and receiver.

Beyond Pacific languages, any population other than English will see a decrease in performance for the BERT and SGNS systems. This is because both systems has been pre-trained on an English text corpus with English text embeddings. In order for the systems to perform well in other languages, the embeddings and transformers would need to be retrained on a large corpus of text in that language with a potentially different character scheme. It would be challenging to both collect a data set of such size, and to retrain the large BERT and SGNS systems. Notably however, SGNS embeddings can achieve acceptable results with less training data than transformer models, which typically require large text corpora. As a result, even though SGNS performed worse than BERT based systems on this task, SGNS does have advantages over BERT when only a small dataset is available.

9 Conclusion

The best systems evaluated in this paper used BERT with an LSTM classifier on all embedded tokens, and achieved an average F1 score of 77.7% on subtask A of OffensEval. This is 4.90 points behind the best team, *NULI*, who also used BERT. However, despite not matching the top teams, this paper gives several insights into BERT and the classification and pooling methods used. Using all of the embedded tokens gave no significant performance improvement over using the [CLS] token, and with certain pooling methods, saw a significant decrease in performance. Using a complex classification method such as LSTM did not lead to a significant increase in performance over simpler linear classification methods. However, it did allow for all the embedded tokens to be used without a decrease in performance.

Additionally, this paper highlights the performance increase of BERT over non-contextual embeddings such as SGNS, with SGNS performing only slightly higher than the baseline of predicting all tweets as offensive, the majority class.

9.1 Future Work

Additional investigation into why some pooling methods such as sum and max had poor performance and high variance respectively could provide interesting insight into why pooling methods have differing performance on different embeddings. Varying hyperparameters such as training epochs and preprocessing methods could both lead to increased performance, and a more thorough understanding of how embeddings can be used in classification tasks.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Technical report, Google. ArXiv:1810.04805 [cs] type: article.

Jiahui Han, Shengtan Wu, and Xinyu Liu. 2019. jhan014 at SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 652–656, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Ping Liu, Wen Li, and Liang Zou. 2019. NULI at SemEval-2019 Task 6: Transfer Learning for Offensive Language Detection using Bidirectional Transformers. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 87–91, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Debanjan Mahata, Haimin Zhang, Karan Uppal, Yaman Kumar, Rajiv Ratn Shah, Simra Shahid, Laiba Mehnaz, and Sarthak Anand. 2019. MIDAS at SemEval-2019 Task 6: Identifying Offensive Posts and Targeted Offense from Twitter. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 683–690, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. Technical report, Google. ArXiv:1301.3781 [cs] type: article.

Alex Nikolov and Victor Radivchev. 2019. Nikolov-Radivchev at SemEval-2019 Task 6: Offensive Tweet Classification with BERT and Ensembles. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 691–695, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Alessandro Seganti, Helena Sobol, Iryna Orlova, Hannam Kim, Jakub Staniszewski, Tymoteusz Krumholc, and Krystian Koziel. 2019. NLPR@SRPOL at SemEval-2019 Task 6 and Task 5: Linguistically enhanced deep learning offensive sentence classifier. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 712–721, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Jurgen Schmidhuber Sepp Hochreiter. 1997. Long short-term memory. In *Neural Computation*, volume 9, pages 1735–1780.

Ralf C. Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. Technical report, Schmalkalden University of Applied Sciences. ArXiv:1909.09586 [cs] type: article.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to Fine-Tune BERT for Text Classification? Technical report, Fudan University. ArXiv:1905.05583 [cs] type: article.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *31st Conference on Neural Information Processing System*.

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, Volume 1 (Long and Short Papers)*, pages 1415–1420, Minneapolis, Minnesota. Association for Computational Linguistics.

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*, pages 75–86, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

A Code

All the code to implement the SGNS and BERT models with classification methods and evaluate them against OffensEval (including the data used for testing and training) can be found in the following repo: `https://github.com/GarethHarcombe/unsupervised-systems`

B Confusion Matrices

A confusion matrix from a BERT system using linear classification on the `[CLS]` token is listed in Table 5.

		Prediction outcome	
		NOT	**OFF**
actual value	**NOT**	560	60
	OFF	92	148

Table 5: BERT System Confusion Matrix