Offense-CNN at SemEval-2018, Task A, B and C: GloVe embeddings and sentiment analysis to automate offensive Tweet detection using a Convolutional Neural Network classifier

Julian Mack
Imperial College, London
jfm1118@ic.ac.uk

Samuel Ogunmola
Imperial College, London
aso115@ic.ac.uk

Abstract

We designed a system to categorize offensive tweets as part of the Semeval-2018 challenge. We achieved our best performance with GloVe embeddings fed into a neural network consisting of a two convolutional layer feature extractor and five dense layers for classification. The fully connected layers also recieved the tweet's sentiment as input.

1 Preprocessing and embeddings

The size of the corpus in these tasks is small: just over 13,000 tweets, making it unlikely that a BOW approach would work well. Hence, we decided to use pretrained word embeddings in order to leverage the insight of those with more computing resources than ourselves. In doing so, we investigated a range of pretrained embeddings: BERT full sentence embeddings (Devlin et al., 2018), ELMO character embeddings (Peters et al., 2018) and GloVe word embeddings (Pennington et al., 2014).

Our expectation was that we would achieve the best results with BERT followed by ELMO followed by GloVe. In reality, as a result of memory issues, we were unable to train networks on the full dataset with ELMO (although received promising results with a subset of the data) and found it difficult to achieve good generalization performance with BERT (see Section 4 for a more lengthy discussion). In comparison, GloVe embeddings performed well and we found them considerably easier to use in PyTorch than their newer counterparts.

The GloVe model is trained by considering the number of co-occurrences of words within a large corpus and then extracting a lower dimensional dense representation of the co-occurrence matrix[1]. There are GloVe models pre-trained on a range of sources including Wikipedia and news data but, for this use case, it was natural to use the embeddings obtained from training on a corpus of 2 billion tweets[2]. We used 200-dimensional word vectors which were the largest available.

Group	Feature
A	sentiment
	subjectivity
B	count(words)
	count(punctuation)
	count(hashtags)
	count('@user')

Table 1: The engineered features added to our classifiers. The Group A features were created with Python's TextBlob. Here, 'subjectivity' is measured as a scale in the range [0, 1] where 0 is completely objective and 1 is completely subjective.

In addition to using these dense GloVe embeddings, we also used the pre-extracted features 'sentiment' and 'subjectivity' (Group A in Table 1) and observed a small increase in performance (1-2% increase in macro F1) as a result. We also investigated the effect of the features in Group B of Table 1 but did not observe any improvement. This was as expected; the sentiment and subjectivity regressors, implemented by the Python library TextBlob, were trained on a large corpus of IMDb movie reviews meaning these features introduce contextual information that was not in the GloVe tweet corpus, nor in our training set. In comparison, our classifier already had indirect access to the Group B features so we would expect it to extract them if they were relevant.

[1]The training objective is to learn vectors such that the dot product of two word embeddings is equal to the log of their probability of co-occurrence

[2]The twitter GloVe embedding gave a marked improvement over the Wikipedia-trained embeddings. A similar and more in-depth comparison is given in (Yang et al., 2018)

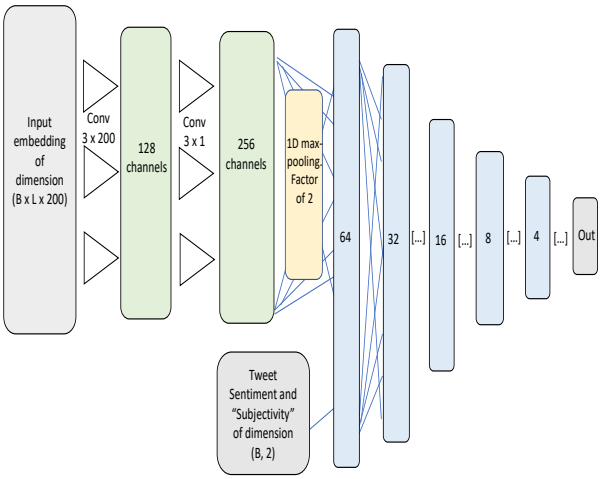

Figure 1: Our CNN architecture used for subtask A and subtask B. Inputs are in grey, convolutional layers are in green and fully connected layers are in blue. Input dimensions are (B, L, 200) where B is batch-size, L is the maximum sequence length for the batch and 200 refers to the GloVe embedding dimension. We used leaky ReLU activations between all layers and a 1D max pooling was applied after the 2 convolutional layers along the 3rd (and final) dimension. The architecture for subtask C is given in 2.

We performed a variety of preprocessing steps on the tweets. All tweets were lower-cased and tokenized using SpaCy. The following three steps were then optional in our pipeline:

1. We combined the nltk and sklearn collections of english stopwords and removed these from the tokens.
2. We removed all punctuation.
3. We replaced every word with its lemmatized form.

We treated these three steps as boolean hyperparameters for each subtask. We found that all three subtasks benefited from punctuation removal and lemmatization but only subtask A saw a boost when we removed stopwords. We hypothesised that some stopwords such as 'you' or 'they' are irrelevant in the detection of offensive tweets but crucial to finding the target of a tweet given that the tweet is offensive. GloVe contains embeddings for punctuation so we were surprised that its removal increased performance.

2 Classifier Design

We took a data-driven approach to designing a good classifier for offensive tweet detection. As such, we initially created a range of classifiers and then iteratively improved the models that performed best on our validation set. Our final classifier for the first two tasks (see Figure 1) consists of a two-layer CNN feature extractor followed by five fully connected hidden layers for classification. We will now undertake a brief summary of the other classifiers that we proposed in order to demonstrate the methodology by which we choose our final network.

Model[3]	Macro Avg F1
Task A	
Linear SVM with BOW	0.65
Linear SVM with GloVe	0.63
Task B	
Linear SVM with BOW	0.57
Linear SVM with GloVe	0.58
Task C	
Linear SVM with BOW	0.53
Linear SVM with GloVe	0.52

Table 2: Maximum performance achieved on a validation set for each model. BOW was performed with Term Frequency and stop-words removed and GloVe word embeddings were averaged to give a sentence embedding.

2.1 Explorations and methodology

We noted that the class distributions for all three tasks are unbalanced and, since the task of offense detection is non-trivial, there is a significant danger of classifiers simply predicting the majority class. We tackled this issue by only investigating classification algorithms where it was possible to avoid mono-class predictions by increasing the weighting on the loss for the minor class(es). An alternative route would have been to upsample the minor class by augmenting the data (e.g. replacing words with their synonyms) but this was not the route we took here.

In our exploration we decided to also look at "traditional" machine learning algorithms including as Support Vector Machines. We found that the use of the GloVe word embeddings does not appear to give any improvement over the Bag of Words approach when using SVMs (see Table 2). However, it should be noted that we averaged the GloVe word embeddings here to get a sentence embedding and thereby lost information about word order. The fact that our convolutional architectures performed better (see Section 3) sug-

gests the structure of a sentence is relevant in offensive language classification. The feature size for GloVe (200) is considerably lower than that of BOW (approximately 12,000) but we were able to obtain comparable results suggesting that the GloVe algorithm performs effective dimensionality reduction.

Our brief review of the NLP literature suggested that a bi-directional memory-based RNN feature extractor feeding into a fully connected network would produce state of the art performance (Young et al., 2017) so we produced a simple network of this type. For every tweet, the sequence of GloVe word embeddings was fed into the GRU units sequentially meaning selected features were outputted for each word. The GRU layer was bi-directional, meaning that these features were dependent on the subsequent words, as well as the prior ones. We used the feature output from the final word in the tweet as input to the fully connected layers on the basis that this should encode the information from all of the previous words. Note that we used GRU recurrent units instead of LSTMs as there are fewer parameters to train. Unfortunately, we achieved very poor performance with this network. There could be many reasons for this, but we suspect that the training corpus was insufficient in size for the GRU units to learn robust rules about the structure of a sentence. If we had had more time, we would have liked to pre-train this network on a next-word prediction task using a considerably larger Twitter corpus before adding the classification layer. Note that this is very similar to how BERT creates full sentence embeddings although it uses Transformers rather than GRUs.

2.2 Final architectures and optimization

In our preliminary investigation, we found that a simpler version of the network in Figure 1 with just one convolutional layer and two fully connected layers was performing better than either the RNN or the traditional ML classifiers. As a result, we decided to improve on this basic CNN-FC architecture to produce the best classifier possible. We were aware that the optimal architecture might be different depending on the text pre-processing decisions (such as whether to remove punctuation) so we included these in our hyperparameter optimization. The full list of hyperparameters (see Table 3) was too large for grid search so we used

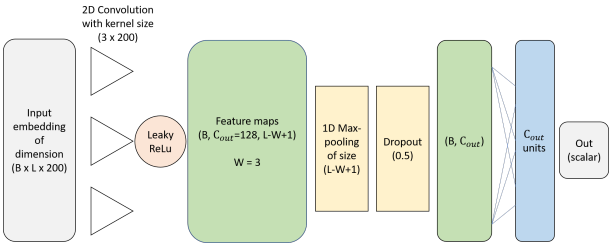

Figure 2: Our CNN architecture used for subtask C. Inputs are in grey, convolutional layers are in green and fully connected layers are in blue. Input dimensions are (B, L, 200) where B is batch-size, L is the maximum sequence length for the batch and 200 refers to the GloVe embedding dimension. We used a cross-entropy loss as with task A and B.

Hyperparameter	Range
Lemmatize words	{True, False}
Remove punctuation	{True, False}
Remove stopwords	{True, False}
Add sentiment (and subjectivity)	{True, False}
Type of model	{simple CNN, Deep CNN, Bidirectional GRU}
Window size (in CNN only)	{2, 3, 4, 5, 6, 7, 12, 20, 30}
Learning rate	[0.0001, 0.0015]
weight decay	[0.0,0.1]

Table 3: Hyperparameters explored in Bayesian Optimization of subtask A.

the sckit-optimize implementation of Bayesian optimization with gaussian processes. In an ideal world, we would have conducted a full scale hyperparameter search for all three subtasks but unfortunately, this was not possible due to time and resource constraints. As a result, we attempted to use the classifier architecture and parameters that were successful for subtask A on both of the subsequent tasks with a small number of ad-hoc tweaks. This approach was successful for subtask B, but we found that we could not get good results with this architecture for subtask C and had to opt for the simple model given in Figure 2. In-line with our earlier decision to create models

Hyperparameter	A	B	C
Lemmatize	TRUE	TRUE	TRUE
Rem. punctuation	TRUE	TRUE	TRUE
Rem. stopwords	TRUE	FALSE	FALSE
Add sentiment	TRUE	TRUE	FALSE
Type of model	Deep CNN	Deep CNN	CNN
Window size	2 x 3	2 x 3	5
Learning rate	0.00017	0.00017	0.00017
Weight decay	0.0054	0.0054	0.0054

Table 4: Hyperparameters used for all subtasks.

Subtask	Macro F1	Best† Macro F1
A	0.758644	0.785731
B	0.666048	0.692457
C	**0.55178**	**0.55178**

Table 5: Macro F1 performance on test sets. †The SemEval leaderboard has not been made public so here 'best' refers to Imperial's NLP course entrants as of midday on March 1st.

capable of predicting both classes, in the Bayesian optimization, we maximized the product of the F1 scores for both classes when the model was run on the validation set. We chose this metric instead of the macro F1 on the basis that it will be zero if a model simply outputs the majority class. This means it gives a stronger signal when exploring hyperparameter regions that create useless classifiers. Different combinations of hyperparameters alter the speed at which the network learns and begins to overfit, so we took the maximum value of the F1 product across all training epochs (on the basis that we could choose an appropriate number of epochs at a later point).

Our results are in Table 4. Interestingly, we found that, for this architecture and dataset, weight decay was preferred to dropout as a method of regularization. We also found that convolutional window sizes of 5 words were preferred[4].

Inspired by the imaging domain, we knew that it is often better to have a series of small kernel convolutions than a single convolution with a large kernel as this reduces the number of parameters and, in theory, gives more opportunity to learn more abstract features. As such, we investigated the effect of replacing a single convolutional layer of window size 5 with two convolutional layers of window size 3. We found that there was a small increase in performance as the result of this change so we incorporated it into our final architecture.

[4]In the Bayesian optimization process, we explored convolutional window-size values in $\{x \in \mathbb{Z} | 2 \leq x \leq 5\}$. The results suggested a kernel width of 5 was best but as this was at the boundary of our domain, we were worried that the optimal value could be larger. We conducted a quick line search on larger values of this parameter whilst keeping the other optimal hyperparemeters constant and found that a window size = 5 was best under these conditions. However, it is possible that we may have found a better combination of parameters than those given in Table 4 if we had allowed a larger search domain in our original optimization.

3 Results

Our submission results are in Table 5 and our validation set results are summarized in Table 6.

SUBTASK A				
	Precision	Recall	F1-score	Support
NOT	0.84	0.80	0.82	1733
OFF	0.62	0.68	0.65	875
Micro av.	0.76	0.76	0.76	2648
Macro av.	0.73	0.74	0.73	2648
Weight av.	0.77	0.76	0.76	2648

SUBTASK B				
	Precision	Recall	F1-score	Support
TIN	0.91	0.80	0.86	768
UNT	0.26	0.48	0.34	112
Micro av.	0.76	0.76	0.76	880
Macro av.	0.59	0.64	0.6	880
	0.83	0.76	0.79	880

SUBTASK C				
	Precision	Recall	F1-score	Support
IND	0.81	0.76	0.79	476
GRP	0.52	0.68	0.59	207
OTH	0.29	0.17	0.22	92
Micro av.	0.67	0.67	0.67	775
Macro av.	0.54	0.54	0.53	775
Weight av.	0.67	0.67	0.67	775

Table 6: Validation set performance.

4 Challenges and obstacles

We initially envisioned using either ELMO or BERT embeddings because, unlike the Torchtext implementation of GloVe, the ELMO and BERT weights can be fine-tuned during training to tailor them to a specific task. In addition, BERT and ELMO embeddings are contextualised meaning that the same token in two different sentences will have different embeddings. For ELMO, we made use of the AllenNLP implementation[5] but found fine-tuning ELMO to be GPU memory intensive, even with small batch sizes. Nevertheless,

[5]See https://allennlp.org/elmo

we were able to train a convolutional network using a small subset of the data. The engineering challenge of integrating ELMO with our convolutional architecture is a possible future extension of this work.

We utilised the huggingface implementation of BERT[6] and were able to run a fully connected network with the BERT full sentence embedding ('pooled output' in BERT terminology) and a convolutional network over individual word embeddings but both networks gave results that were only marginally better than the baseline. With the aforementioned difficulties in mind and with the added bonus of GloVe providing word embeddings that were trained on a data-set similar to our use case, we decided to focus on GloVe embeddings.

The other difficulty we faced was getting our GRU-based RNN architecture described in Section 2 to give good predictions. Like with BERT, our implementation gave results that were only slightly better than the baseline.

5 Summary

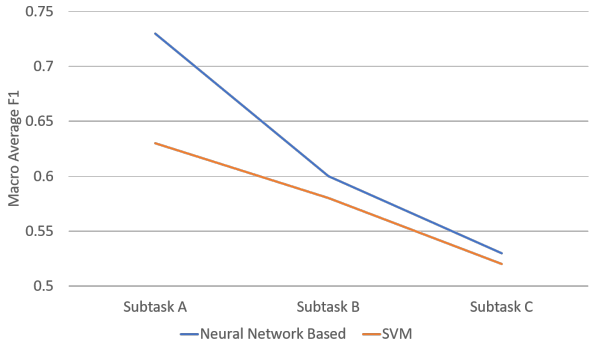

Figure 3: This shows a comparison of the performance of our neural network against Support Vector Machines (both using GloVe embeddings) across the three tasks. The size of training data decreases from left to right.

In conclusion, our experiments and results show that while it is possible to improve the classification of offensive language using deep convolutional networks instead of traditional methods, the size of improvement reduces as the amount of data available decreases. This can be seen in Figure 3 and in the fact that we needed to settle for a much simpler CNN architecture in task C. Our results, further emphasise the need for a large labeled corpus when tackling NLP classification problems.

6 Supplementary Information

For more details, please see our code in the following repository: https://github.com/adesam146/nlpcw. It is recommended that the code is viewed in colab rather than directly on GitHub.

[6]See https://github.com/huggingface/pytorch-pretrained-BERT

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Xiao Yang, Craig Macdonald, and Iadh Ounis. 2018. Using word embeddings in Twitter election classification. *Information Retrieval Journal*, 21(2-3):183–207.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2017. Recent trends in deep learning based natural language processing.