SemEval-2019 Task 6: Offensive Language Analysis using Deep Learning Architecture

Ryan Ong
Faculty of Engineering, Department of Computing
Imperial College London
cmo18@ic.ac.uk

Abstract

SemEval-2019 Task 6 (Zampieri et al., 2019b) requires us to identify and categorise offensive language in social media. In this paper we will describe the process we took to tackle this challenge. Our process is heavily inspired by Sosa (2017) where he proposed CNN-LSTM and LSTM-CNN models to conduct twitter sentiment analysis. We decided to follow his approach as well as further his work by testing out different variations of RNN models with CNN. Specifically, we have divided the challenge into two parts: data processing and sampling and choosing the optimal deep learning architecture. In preprocessing, we experimented with two techniques, SMOTE and Class Weights to counter the imbalance between classes. Once we are happy with the quality of our input data, we proceed to choosing the optimal deep learning architecture for this task. Given the quality and quantity of data we have been given, we found that the addition of CNN layer provides very little to no additional improvement to our model's performance and sometimes even worsen our F1-score. In the end, the deep learning architecture that gives us the highest macro F1-score is a simple BiLSTM-CNN.

1 Introduction

In this paper we will describe the process we took to tackle SemEval-2019 Task 6 (Zampieri et al., 2019b). Zampieri et al. (2019a) describes the dataset for this task. We have divided the challenge into two parts: data processing and sampling and choosing the optimal deep learning architecture. Given that our datasets are unstructured and informal text data from social medias, we have decided to spend more time creating our text preprocessing pipeline to ensure that we are feeding in high quality data to our model. In addition, we realised that there's a high level of imbalance between classes in each of the subtasks. Therefore, we decided to experiment with two different techniques that tackle this imbalance; SMOTE and Class Weights. Once our data is clean and our data distribution among classes are balanced, we proceed to choosing our optimal deep learning architecture. We decided to use macro F1-score as our evaluation metrics due to the imbalance classes. Through searching for the optimal model architecture, we made two important findings. Firstly, the order of our layering in our models heavily affects our F1-score performance. We found that by feeding data into the LSTM layer first, then followed by CNN layer yields much better results than the alternative. Secondly, in this challenge, the addition of CNN layer provides very little to no additional improvement and sometimes even lead to a decrease in our F1-score. We suspect that by feeding inputs into the CNN layer, we lose the important sequential information in text data, thereby making our models less accurate. In the end, we found that the deep learning architecture that gives us the highest F1-score among subtasks is BiLSTM-CNN.

2 Deep learning architecture

In this paper, we experimented with different variations of CNN and LSTM layers. Our overall deep learning architecture is shown in Figure 1, where we initially feed our input text through an embedding layer to get our word embeddings. Depending on the variations of our CNN and LSTM layers, for example CNN-LSTM, we will feed these word embeddings to the convolution layer. The output will undergo MaxPooling layer (part of CNN), resulting in a smaller dimension output, which is then feed into the LSTM layer. We will then apply spatial dropout to the output of LSTM layer in an attempt to counter overfitting. This is

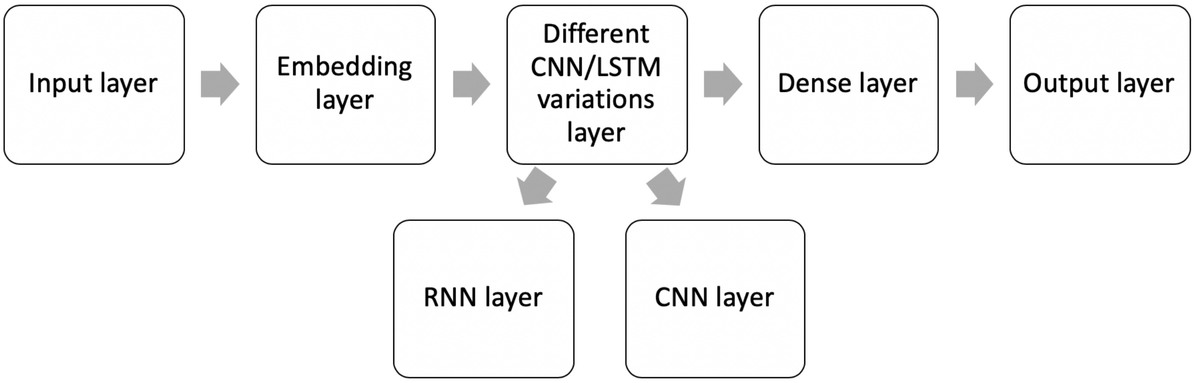

Figure 1: Overall model architecture

followed by a dense layer before our model architecture outputs the results through the output layer.

The model was implemented using the Keras libary with Tensorflow backend.

2.1 Pre-trained word embeddings

Word embeddings are widely used in different NLP tasks. In this paper, we decided to experiment with different kinds of word embeddings, including different dimensionality of the same embeddings, to see if different type and dimensionality of embeddings would affect the overall end performance of our models. Specifically, we decided to experiment with the following word embeddings (Stanford, 2014):

1. ***GloVe: Twitter (100d)*** - Trained on 2B tweets, which is very relevant to our tasks of analysing text data from social media. It has 27B tokens and 1.2M vocabulary of unique words

2. ***GloVe: Twitter (200d)*** - Same as Glove: Twitter (100d) except it's 200 dimension. This allows us to evaluate the increase in dimensionality on the performance of our models

3. ***GloVe: Common Crawl (300d)*** - Trained on 42B tokens, 1.9M vocabulary of unique words. We chose this embeddings for its relatively large vocabulary size and embeddings dimensions

2.2 Optimisation and Regularisation

Given our relatively small dataset, the network is trained using batch gradient descent with Adam optimiser. To counter overfitting, we have decided to utilise spatial dropout 1D regularisation which performs like a normal dropout regularisation except you drop the entire 1D feature maps instead of individual activation. This is because if adjacent frames within the same feature maps are highly correlated, then regular dropout will fail to regularise the activations.

3 Training

3.1 Data

To train and evaluate our models, we will be using the provided training and trial dataset. However, given the extremely small trial dataset, we have decided to combine both datasets as we aren't able to properly assess our models' predictions accurately with the trial dataset. Table 1 shows the label distribution of all the datasets. In addition, given the level of inbalance between classes in each subtasks, we have decided to focus more on the F1-scores, particularly the macro F1 score rather than just relying on the overall accuracy.

To train our model, we split the combined dataset randomly into 80% train-val and 20% test set and use the train-val set to perform k-fold cross validation (k = 5). Specifically we train each models using k-fold cross validation and use the validation set to do early stopping if the performance does not improve after 10 epochs with respect to average **macro F1-score**. Once we are happy with the performance of our final model, we do a final evaluation using the 20% test set.

3.2 Preprocessing

Our data preprocessing pipeline is as follows:

- *Remove @USER and URL token*

Dataset	Subtask A		Subtask B		Subtask C		
	NOT	OFF	TIN	UNT	IND	GRP	OTH
Train	8840	4400	3876	524	2407	1074	395
Trial	243	77	38	39	30	4	5
Combined	9083	4477	3914	563	2437	1078	400

Table 1: Benchmark dataset label distribution

- *Remove hashtags, twitter handles and hyperlinks*

- *Apostrophe contraction-to-expansion* - We used a dictionary to map contracted words to their corresponding expanded words. For example, *don't* will transform to *do not*. This preprocessing steps reveal ”hidden” negation words that are important for our models to detect offensive languages

- *Spelling corrections* - We used open-source Sympell (Wolfgarbe, 2018) which uses the Damerau-Levenshtein distance to find the closest correct spellings for any misspelled words. We chose the edit distance to be 3

- *Lemmatisation* - We used WordNetLemmatizer from NLTK to lemmatise all the words to their lemma form. For example, *saw* to *see*. By lemmatising the words, we only feed in words in their lemma form, therefore allowing the models to be able to capture the meaning of words regardless of their original forms.

- *All text are lowercased*

3.3 Class Imbalance

The provided dataset has a high level of class imbalance (shown in Table 1) and we have decided to use two different approaches to counter this: **class weights** and **SMOTE** (Chawla et al., 2011). Class weights involves computing the class weights and use it to re-scale the loss function when performing back-propagation. SMOTE, on the other hand, is an oversampling technique whereby it generates new data points using the existing minority data that we supply as input. The algorithm takes samples of the feature space for each target class and its nearest neighbors and generates new examples that combine features of the target case with features of its neighbors (Kim, 2018).

4 Experiments and results

4.1 Experiment environment

In order to find the optimal architecture for this task, we have decided to experiment with CNN and different variations of RNN, which includes LSTM and GRU (each either unidirectional or Bidirectional). Each variation of model will follow the same overall structure as mentioned in Section 2.

We will be making performance comparisons between the models below:

1. CNN
2. LSTM
3. BiLSTM
4. GRU
5. BiGRU
6. CNN-LSTM
7. CNN-BiLSTM
8. CNN-GRU
9. CNN-BiGRU
10. LSTM-CNN
11. BiLSTM-CNN
12. GRU-CNN
13. BiGRU-CNN

Each model will be train using 5-fold cross validation and will be evaluated on the average accuracy and macro F1-score.

4.2 Results analysis - Subtask A

The results on Table 2 shows the average accuracy and macro F1-score of each architecture after 5-fold cross validation.

Models (Subtask A)	Avg Acc	Avg Macro F1
CNN	71%	0.63
LSTM	**78%**	**0.73**
BiLSTM	**78%**	**0.73**
GRU	**79%**	**0.74**
BiGRU	**78%**	**0.74**
CNN-LSTM	71%	0.66
CNN-BiLSTM	72%	0.65
CNN-GRU	72%	0.66
CNN-BiGRU	72%	0.67
LSTM-CNN	74%	0.67
BiLSTM-CNN	**78%**	**0.74**
GRU-CNN	73%	0.67
BiGRU-CNN	**77%**	**0.72**

Table 2: Average accuracy and macro F1-score of different model architecture (k-fold = 5) - **Subtask A**

Given our inbalanced datasets, we will primarily be evaluating our models using the average macro F1-score. A standalone CNN model yields the lowest average macro F1-score of 0.63. Through adding an LSTM or GRU (either unidirectional or bidirectional) layer after the CNN layer, thereby forming a LSTM-CNN or GRU-CNN, our model scores on average 0.02 - 0.04 higher than standalone CNN model. However, this is 0.06 lower than standalone LSTM (0.73). A possible reason for this could be that although CNN layer is great at extracting local features and learn to emphasise or disregard certain n-grams in the input data, it still looses some of the important sequential information in our text input.

On the other hand, a standalone LSTM or GRU model yields the highest average macro F1-score of 0.73-0.74. Our results show that there's no significant difference between unidirectional and bidirectional LSTM or GRU. Intuitively, the benefit of a LSTM or GRU layer is that the network will be able to remember what was read previously, therefore can develop a better understanding of future inputs. We found that a normal unidirectional LSTM-CNN or GRU-CNN underperformed relatively to standalone LSTM/GRU models and only outperforms standalone CNN marginally by 0.04. BiLSTM-CNN/BiGRU-CNN achieve average macro F1-score similar to standalone LSTM/GRU. Our results show that adding a CNN layer after LSTM/GRU provides no benefits or worsen the score.

Overall, our results show that the ordering of layers significantly affect the performance of our models. Our results indicate that the optimal ordering of layers is LSTM/GRU follow by CNN, thereby forming a LSTM-CNN/GRU-CNN architecture. The initial LSTM/GRU layer will be able to capture sequential information unlike having CNN layer as the first layer. The output is then pass to the CNN layer to extract local features.

4.3 Subtask B and C

Given our findings on the optimal ordering of layers and the fact that BiLSTM-CNN and BiGRU-CNN significantly outperformed normal LSTM-CNN and GRU-CNN in subtask A, we have decided to only apply BiLSTM, BiGRU, BiLSTM-CNN and BiGRU-CNN to subtask B and C. The holdout results for subtask B and C are shown in Table 3 and 4 respectively. We decided not to use cross validation for subtask B and C due to computationally intensive to run. The results show that SMOTE is the best technique to tackle the class imbalance issue. With the exception of BiLSTM, we performed top macro F1-score for the other three models. However, for subtask C, it seems that our results has got worse since applying SMOTE/Class Weights to the datasets, with the exception of BiGRU-CNN. Our results indicate that it is better off keeping the original datasets.

Taking the results from our experiments, we conclude that the optimal deep learning architecture to tackle SemEval-2019 Task 6 offensive language analysis is BiLSTM-CNN as it consistently outperforms every other model variations. We decided to not apply SMOTE or class weights to datasets in subtask A as the level of imbalance in

Models	Imbalanced Data		SMOTE		Class Weights	
(Subtask B)	Acc	Macro F1	Acc	Macro F1	Acc	Macro F1
BiLSTM-CNN	87.39%	0.51	81.25%	**0.59**	68.86%	0.55
BiGRU-CNN	87.83%	0.47	78.68%	**0.57**	44.64%	0.40
BiLSTM	87.50%	**0.56**	79.80%	0.53	43.86%	0.40
BiGRU	87.95%	0.53	75.22%	**0.55**	55.24%	0.48

Table 3: Evaluation of different techniques to tackle class imbalance. Table displays accuracy and macro F1-score of different model architecture (holdout method) - **Subtask B**

Models	Imbalanced Data		SMOTE		Class Weights	
(Subtask C)	Acc	Macro F1	Acc	Macro F1	Acc	Macro F1
BiLSTM-CNN	69.99%	**0.48**	66.16%	0.45	59.13%	0.44
BiGRU-CNN	71.14%	0.42	68.20%	**0.45**	63.09%	0.35
BiLSTM	69.48%	**0.45**	67.82%	0.45	61.30%	0.45
BiGRU	71.39%	**0.46**	64.11%	0.43	62.58%	0.43

Table 4: Evaluation of different techniques to tackle class imbalance. Table displays accuracy and macro F1-score of different model architecture (holdout method) - **Subtask C**

Epochs	BiLSTM-CNN	BiGRU-CNN
5	**0.74**	**0.75**
10	0.70	0.70
20	0.71	0.73

Table 5: Macro F1-score for BiLSTM-CNN trained with different epochs - **Subtask A**

subtask A is mild. In terms of subtask B and C, it is clear that we should apply SMOTE to balance our data among classes in order to yield the highest possible macro F1-score.

4.4 Hyperparamter Tuning and Findings

Once we finalise our model to be BiLSTM-CNN, we conducted manual search for some of the key hyperparameters of the model using subtask A. This include optimal number of epochs to train our model, the spatial dropout probability and the use of different types and dimensions of word embeddings. We included BiGRU-CNN as a comparison. The results is as follows:

1. **Level of Epochs** - As shown in Table 5, the optimal number of epochs to train our model is 5. Our model managed to reach high F1-score of 0.74/0.75 (relative to our experiments) after 5 epochs. The F1-score starts to plateau/drop as we increase the number of epochs beyond 5, showing signs of overfitting

2. **Spatial dropout rate** - We have spatial dropout layer immediately after the output of our BiLSTM-CNN model as well as after the dense layer (Figure 1). As shown in Table 6, the optimal spatial dropout rate is 20%. However, when taken out the spatial dropout layer, our macro F1-score was not affected. This might be due to our small network architecture and low overfitting, therefore dropout layer doesn't contribute much to our final performance

Dropout	BiLSTM-CNN	BiGRU-CNN
20%	**0.75**	0.73
35%	0.74	0.70
50%	0.72	0.72
No Dropout	0.74	**0.74**

Table 6: Macro F1-score for BiLSTM-CNN trained with different spatial dropout rates - **Subtask A**

3. **Pre-trained vs No pre-trained embeddings** - Our results in Table 7 aligns with the industry trend that by using pre-trained word embeddings, we yield a higher macro F1-score when compared trained without pre-trained word embeddings. In addition, we see an increase in the performance of our BiLSTM-CNN as we increased the dimensions of our word embeddings. However, due to the contrasting results from BiGRU-CNN, we aren't unable to draw a conclusion and further experiments is needed

Embeddings	BiLSTM-CNN	BiGRU-CNN
T - 100d	0.72	**0.74**
T - 200d	**0.75**	0.73
CC - 300d	**0.75**	0.71
No Embs	0.69	0.67

Table 7: T - GloVe Twitter, CC - GloVe Common Crawl. Macro F1-score for BiLSTM-CNN trained with/without pre-trained embeddings - **Subtask A**

Subtasks	Macro F1	Ranking
A	**0.75**	56
B	**0.65**	38
C	**0.46**	77

Table 8: Macro F1-score & Ranking - Hidden test set

5 Conclusion

From all our experiments, we concluded that our optimal model architecture is BiLSTM-CNN, trained with 5 epochs, no dropout layers (unless we decided to build a bigger model architecture) and use of pre-trained word embeddings, 42B GloVe-Common Crawl (300d). In this paper, we experimented with 13 model variations with the aim to find the optimal model architecture for offensive language analysis. Our findings show that the ordering of layers in our model are extremely important. By having CNN layer first followed by different types of RNN layers, our models perform 0.07 - 0.09 worse in terms of F1-score when compared to having RNN layers first followed by a CNN layer. We used BiLSTM-CNN to predict the labels for the hidden test set and our final macro F1-scores and rankings are shown in Table 8. **Our code is available at:** https://github.com/RyanOngAI/semeval-2019-task6

5.1 Future Work

1. **Systematic search** - Manual hyperparameter search limits the number of experiments I can carry out, for example, I wasn't able to manually test out different dropout and recurrent dropout rate within the RNN layers. This has been set to 35% randomly. Therefore it would be beneficial to implement different systematic search such as grid search or bayesian optimisation to optimise the hyperparameters for our models

2. **Contextualised word embeddings** - On top of tradition word embeddings, it would also be interesting to see how contextualised embeddings would affect the results of our models given the rise of BERT and ELMO

3. **Character-level** - Given the informal nature of our text data, it would be interesting to see the results of character level model variations of our experiments above seeing as the full power of pre-trained word embeddings is limited by the misspelled/slang words

References

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2011. Smote: Synthetic minority oversampling technique. *Journal Of Artificial Intelligence Research*, arXiv:1106.1813.

Ricky Kim. 2018. Yet another twitter sentiment analysis part 1 : tackling class imbalance. https://towardsdatascience.com/yet-another-twitter-sentiment-analysis-part-1-tackling-class-imbalance-4d7a7f717d44. Accessed: 2019-03-01.

Pedro M. Sosa. 2017. Twitter sentiment analysis using combined lstm-cnn models.

Stanford. 2014. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/. Accessed: 2019-03-01.

Wolfgarbe. 2018. Symspell: Spelling correction fuzzy search. https://github.com/wolfgarbe/SymSpell. Accessed: 2019-03-01.

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 NAACL*.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. 2019b. SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval). In *Proceedings of The 13th International Workshop on Semantic Evaluation (SemEval)*.