DoTheMath at SemEval-2020 Task 12 : Deep Neural Networks with Self Attention for Arabic Offensive Language Detection

Zoher Orabe*, Bushr Haddad*, Anas Al-Abood*, Nada Ghneim**
**Damascus University / Damascus, Syria
***AlSham Private University / Damascus, Syria
{zoherorabe999, bushr.haddad, anasabood3}@gmail.com
n.ghneim@aspu.edu.com

Abstract

This paper describes our team work and submission for the SemEval 2020 (Sub-Task A) "Offensive Eval: Identifying and Categorizing Offensive Arabic Language in Arabic Social Media". Our two baseline models were based on different levels of representation: character vs. word level. In word level based representation we implemented a convolutional neural network model and a bi-directional GRU model. In character level based representation we implemented a hyper CNN and LSTM model. All of these models have been further augmented with attention layers for a better performance on our task. We also experimented with three types of static word embeddings: word2vec, FastText, and Glove, in addition to emoji embeddings, and compared the performance of the different deep learning models on the dataset provided by this task. The bi-directional GRU model with attention has achieved the highest score (0.85% F1 score) among all other models.

1 Introduction

With the massive increase of social interactions on online social networks, and the expansion of opportunities provided by these networks, there has also been a rise of offensive and hateful content that exploits and violates such networks. Offensive language is defined as any explicit or implicit textual insult or attack against other people. Offensive language has many variants such as hate speech, which is defined as any targeted offensive language toward a particular person (a politician, a celebrity, etc.) or a particular group (a gender, a religion, a color). Offensive language is considered a critical problem to the social networks for its huge psychological negative impact on users.

This work has been prepared and organized as a participation at OffensEval2020 competition (Zampieri et al., 2020). Zampieri (2020) features a multilingual dataset with five languages (Arabic, Danish, English, Greek, and Turkish). We participated in the sub-task dedicated for Arabic language namely sub-task A (Arabic offensive language detection) (Mubarak et al., 2020). In this work, we present our deep learning models featured on word and character level and aimed to identify offensive language in Arabic tweets. In the rest of this paper, a brief of related works are presented in section 2. In section 3, we present our data preparation, and representation. Our proposed models are developed in section 4. In section 5, a brief discussion on the results is addressed. At the end, a short summary and some future insights are presented.

2 Related Works

Several works have addressed the problem of offensive language detection and its variants (Cambray and Podsadowski, 2019; Zhang and Luo, 2019; Gambäck and Sikdar, 2017; Badjatiya et al., 2017). Cambray (2019) competed on OffensEval 2019, and presented their best model as a bidirectional LSTM; followed by a two-branch bidirectional LSTM and GRU architecture. Zhang (2019) proposed two deep neural networks, CNN and GRU, to identify specific types of hate speech. Gamback (2017) evaluated CNN model on various word embeddings, and achieved their best score with CNN model trained on Word2Vec word embeddings. Badjatiya (2017) evaluated several neural architectures, and achieved their best score with a two-step approach using a short-term word-level memory (LSTM) model, tuning randomly initializing

Proceedings of the 14th International Workshop on Semantic Evaluation, pages 1932–1937
Barcelona, Spain (Online), December 12, 2020.

word embedding, and then training a gradient boosted decision tree (GBDT) classifier on the average of the tuned embedding in each tweet.

3 System Overview

3.1 Data Preprocessing and Balancing

Zampieri (2020) produced the largest Arabic dataset to date with special tags for vulgarity and hate speech, which is used in this task for the evaluation process. This dataset has the following format: an Arabic tweet followed by a label indicating its class whither it is offensive (OFF) or not-offensive (NOT_OFF), and was previously divided into 3 parts: 70% training data, 10% validation data, and 20% test data. Data preprocessing is an important step in building machine learning models and highly affects the performance of classification problems, especially when dealing with Arabic language, that contains a lot of informal dialects and colloquial words that differ by different countries. The first step of our preprocessing was to discard all meaningless tokens (some stop words, some punctuation marks, and other repeated consecutive characters). We also experimented our results keeping emojis within the texts and studying their effects on our classification task, as many researches including (Chen et al., 2018) showed the importance of emojis analysis in the sentiment analysis process. We have also experimented stemming the data and analyzed the results.

Training set has a huge imbalance over its classes; 19% being offensive and 81% being inoffensive. Researches showed that any classifier trained over an imbalanced dataset tends to have a lower recall on its miner classes. In order to balance the dataset, we have experimented two types of augmentation techniques. The first one was to augment our data with some offensive and inoffensive samples collected from YouTube comments Alakrot (2018), and the other one is to random oversample the tweets by shuffling the words into the sentences and generate new samples.

3.2 Data Representation

Word embedding is a learning text representation where words that have the same meaning have a similar representation, and individual words are presented as real valued vectors.

3.2.1 Static Embedding Methods

- **Word2Vec:** Word2Vec is a famous text embedding algorithm that was developed by (Mikolov et al., 2013) with two different types of embedding techniques: CBOW (Continuous Bag of Word) and SkipGram (Continuous Skip Gram). We have trained these models on our dataset to create pre-trained word embedding vectors that will be used as an input to our deep learning models. Both models (CBOW and SkipGram) are trained with two different vector sizes (100 or 300)[1] and two different n-gram ranges (unigrams or bigrams).

- **Glove:** To marry both the global statistics of matrix factorization techniques such as LSA technique and the local context-based in word2vec (Pennington et al., 2014). So, we used the Glove algorithm which is developed by Pennington et al. (2014). Our Glove model is trained with these parameters: a vector size equals (100 or 300), a learning rate equals 0.05, a window size equals 10, number of epochs equals 10, and n-gram ranges are unigrams and bigrams.

- **FastText:** (Bojanowski et al., 2017) proposed FastText algorithm, which is an extension to Word2Vec model. Our FastText model treat each word as a composed of n-gram, and infer the whole vector as a sum of the vector representation of all the character n-grams. This model was also trained on two different vector sizes (100 or 300).

3.2.2 Character Embedding Methods

Character level models accept a sequence of encoded characters as input. Each character is quantized using a "One-hot encoding" with size equal to the number of unique characters that are used. We have padded each sequence of characters to a fixed sized vector (L0 length). The alphabet used in all of

[1] the vector size for all our static models are set empirically.

our models consists of a fixed number of characters, including (Arabic letter, digit, space, and other characters), which produce a small amount of vectors embedding that aimed to reduce model complexity and computation. Moreover, character-level models are very useful with misspelling words, and it also handles infrequent words. So which could be in perform well with new and unique offensive words and informal words that could be appeared in text.

3.2.3 Emoji Embedding Methods

Social networks have recently seen a wide spread of emojis. Over 10% of Twitter posts and 50% of texts on Instagram contain one or more emojis. So, we kept emojis within the text and take advantage of their meaning for a better semantic text representation. Chen (2018) used emoji embeddings for sentiment analysis and achieved the state of the art results. (Eisner et al., 2016) trained a model to learn emoji representation from their description in the Unicode emoji standard, they created emoji2vec embeddings of 1661 emoji symbols with 300-dimensional vector size for each emoji. We used their publicly available[2] pre-trained Emoji2Vec embeddings, where we combined each emoji vector with its initial static vector that has been trained by our static embedding models (Word2Vec, FastText, or Glove).

4 Our Proposed Models

4.1 Character Level Model

4.1.1 Hyper CNN and LSTM Model with Attention ((CNN+LSTM)_ATT)

Our convolutional network model is similar to the ConvNet Model presented by (Zhang et al., 2015). Researchers mentioned that their model produced the best performance on large-scale datasets in NLP downstream tasks with classification problems. We had recycled their model and mixed it with bi-directional LSTM layers, forward LSTM layer to cover characters from left to right and backward layer to cover them in the opposite direction. We have further augmented our model with a self-attention layer for a better performance. This model contains 6 layers of 1D convolutional layer with 3 max pooling layers between them, then, the output is fed to the attention layer. The same attention layer has been applied by (Zhou et al., 2016). This attention layer tries to reweight the input vectors and compute the "attention weights", by giving more importance for the important steps. The output of this attention layer is fed into a bi-directional LSTM layer, and then pass the output to a fully connected layers, and one neuron at the end to predict the output. To regularize the network, we have added two dropout layers with a dropout probability of 0.5.

4.2 Word Level Models

Word level models accept a sequence of encoded words as an input; each word of our sequence is embedded by one of the static embedding methods (Word2vec, FastText, or Glove).

4.2.1 CNN with Attention (CNN_ATT)

(Collobert and Weston, 2008) were among of the first researchers to apply CNN on NLP tasks. We relatively used the same model. Sentences are first tokenized into words, which are further transformed into words embedding matrix using static embedding techniques, then convolutional filters are applied on the embedding layer to produce feature maps, followed by an attention layer to create the attention weights that used to compute context vectors, this is then followed by a max-pooling operation to reduce the dimensionality of the output and accelerate the learning process computations. At the end, we have a dense layer of one neuron with a sigmoid activation function.

4.3 Bi directional GRU with Attention (BI_GRU_ATT)

Bi-directional GRU is considered a powerful model in many tasks thanks to its ability of recurrent and step dependencies. However, this architecture still treats all input steps as equal which badly affects the ability of connecting dependencies. In this model, we have experimented adding an attention layer after the bi-direction GRU. Firstly, we stack the steps vectors encoded by the bi-directional GRU, then calculate

[2]URL: https://github.com/uclnlp/emoji2vec.

the score of each step, normalize them and pass the obtained final scores into a folding layer to generate a context vector that indicates the importance of each input step. This attention layer is followed by a dense layer of multi neurons and another dense layer of one neuron with a sigmoid activation function.

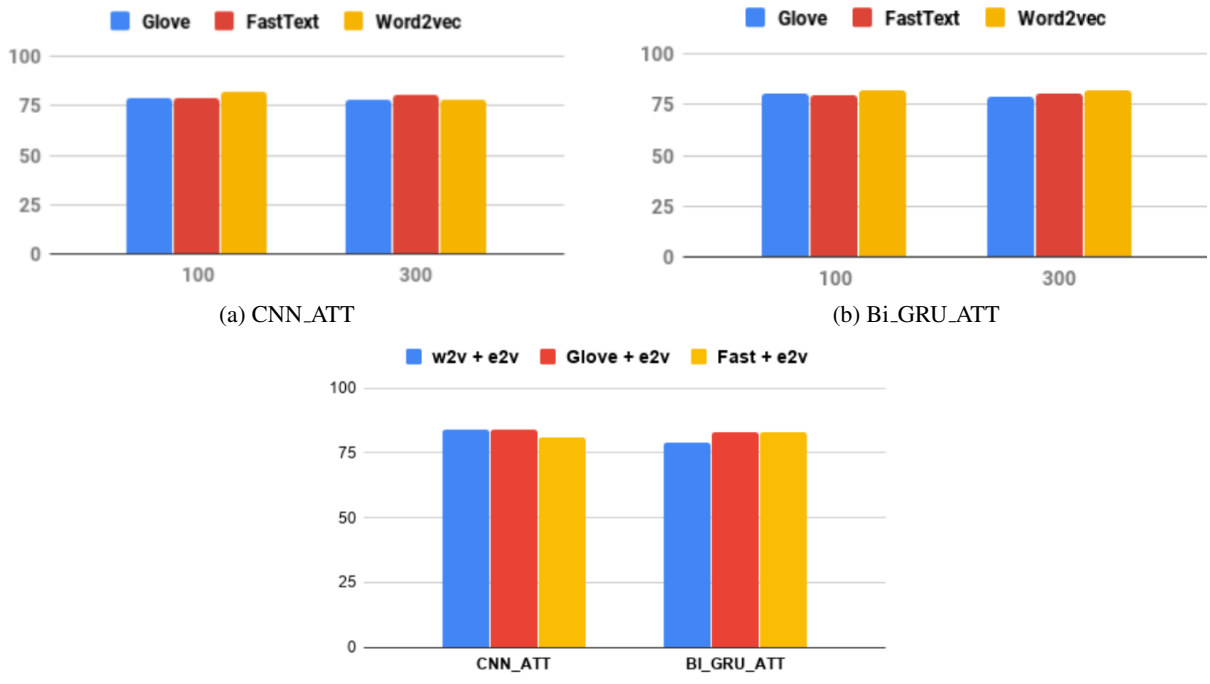
5 Results and Analysis

5.1 Word-Level Models Results

Our word-level models were trained on our dataset before and after the dataset balancing. These results shows the evaluation process on the validations set before balancing our dataset. The hyperparameters that used in our word-level models are tuned empirically like (dorpout, filter size, max pooling size, batch_size, n_epochs, etc..), and we used the default learning rate 0.001 with 'Adam' optimizer.

- **Results without Emojis Embedding:** We present the evaluation results of our word-level model being embedded firstly on three different embedding techniques (FastText, Glove, or Word2vec) and by using different embedding sizes (100 and 300). After embedding, we trained our word-level models to minimize a binary cross-entropy function and by using Adam optimizer. Figure 1 shows a comparison between our embedding algorithms, unigram models shows a better result than bigram ones. It is also clear that the results from both models were approximately similar, however the Word2Vec model was more efficient compared to Glove and FastText models (82% F1 score).

- **Result with Emojis Embedding:** We used the same word-level models but with emoji embeddings, where each emoji in our dataset has been directly trained on Unicode description of emojis as we have explained in section 3. The choice of vector size used in this model is 300. We mixed between emojis embeddings matrix with static embedding matrix (word2vec, FastText, or Glove), where the dimension of vectors is 300 for each of them. We obtained a better result (F1 score of 0.84), that means it is highly recommended to handle emojis with such a problem and It is clear that the meaning behind these emojis in the text are considered as extra features Which make our classifiers have abilities to extract more useful features from the text. These results are also shown in figure 1.



(a) CNN_ATT

(b) Bi_GRU_ATT

(c)

Figure 1: Comparison our word-level models with static embeddings in (a) and (b), and with emoji embeddings in (c). The results are evaluated using the macro F1 score.

We also present our results after balancing the dataset, here we used the publically available word embedding vectors AraVec (Soliman et al., 2017). These pre-trained embeddings were trained on Twitter dataset with skipGram architecture and vector size of 100. We notice that the BI_GRU_ATT is better than CNN_ATT model with 86% F1 score as shown in table 1, this may be due to the fact that GRU captured longer time dependencies between words.

Models	Avg. F1-score	Avg. Recall	Avg. Precision	Avg. Accuracy
CNN_ATT	0.82	0.81	0.83	0.90
BI_GRU_ATT	**0.86**	**0.83**	**0.91**	**0.93**

Table 1: Word-level models after balancing dataset

5.2 Character-Level Model Results:

Our (CNN+LSTM_ATT) has also been trained on our dataset before and after balancing, using the same loss function and optimizer used in word-level models, in addition to max length character with 2028 size, with the number of epochs training steps 30, and batch size 512. The results are shown in table 2 .We notice that the results of character model is less efficient about 10% compared with word level models. By looking at recall and f1-score on our balanced dataset we noticed that it decreased, which means that the augmentation technique that we used by replicating the same words in the minority class to be balanced with other classes isn't very effective. insteadly, we can experiment with other augmentation techniques by adding some new words from WordNet which hold a similar meaning to our words.

	Avg. F1-score	Avg. Recall	Avg. Precision	Avg. Accuracy
Before Balancing	**0.76**	**0.79**	0.75	0.85
After Balancing	0.75	0.72	**0.81**	**0.87**

Table 2: (CNN+LSTM_ATT) model on dataset before and after balancing.

The best model that achieved the highest macro F1 score on the test set in (sub-task A) for SemEval2020 was BI_GRU_ATT with average macro F1 score of 0.85, and table 3 shows the results.

BI_GRU_ATT	F1-score	Recall	Precision	Acc
Subtask (A)	0.85	0.83	0.88	0.91

Table 3: Bi_GRU_ATT model performance on test dataset.

6 Conclusion and Future Work

This paper describes our team contribution to the offensiveEval 2020 contest (SemEval 2020- Sub Task A). Two different levels of representation for deep learning models (character-level, word-level) are used with attention mechanisms. We also tested our word-level models with three types of standard static embeddings such as (FastText, Word2vec, and Glove) in addition to emoji embeddings. The bi-directional GRU model with word2vec pre-trained embeddings and attention mechanism has achieved the highest macro F1 score with 85% as the best result on the test set among all other models. In the future we hope to make our dataset more balanced by collecting some offensive and inoffensive tweets from Twitter and use it to augment our dataset. Furthermore, we also can use some oversampling and undersampling techniques to make our classes balanced. So we hope to use the SMOTE (Synthetic Minority Over-sampling Technique) to oversampling the offensive samples in our dataset to be equal to the not-offensive samples. In contrast, we can use a simple undersampling technique to under-sampling the majority class (NOT_OFF) randomly and uniformly to be equal to minority class (OFF) which can improve the generalizability and flexibility to our models.

References

Azalden Alakrot, Liam Murray, and Nikola S Nikolov. 2018. Dataset construction for the detection of anti-social behaviour in online communication in arabic. *Procedia Computer Science*, 142:174–181.

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

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *arXiv:1609.04606*.

Aleix Cambray and Norbert Podsadowski. 2019. Bidirectional recurrent models for offensive tweet classification. *arXiv:1903.08808*.

Yuxiao Chen, Jianbo Yuan, Quanzeng You, and Jiebo Luo. 2018. Twitter sentiment analysis via bi-sense emoji embedding and attention-based lstm. In *Proceedings of the 26th ACM International Conference on Multimedia*, pages 117–125.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. 2016. emoji2vec: Learning emoji representations from their description. *arXiv:1609.08359*.

Björn Gambäck and Utpal Kumar Sikdar. 2017. Using convolutional neural networks to classify hate-speech. In *Proceedings of the First Workshop On Abusive Language Online*, pages 85–90.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv:1301.3781*.

Hamdy Mubarak, Ammar Rashed, Kareem Darwish, Younes Samih, and Ahmed Abdelali. 2020. Arabic offensive language on twitter: Analysis and experiments. *arXiv:2004.02192*.

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.

Abu Bakr Soliman, Kareem Eissa, and Samhaa R El-Beltagy. 2017. Aravec: A set of arabic word embedding models for use in arabic nlp. *Procedia Computer Science*, 117:256–265.

Marcos Zampieri, Preslav Nakov, Sara Rosenthal, Pepa Atanasova, Georgi Karadzhov, Hamdy Mubarak, Leon Derczynski, Zeses Pitenis, and Cagrı Cöltekin. 2020. Semeval-2020 task 12: Multilingual offensive language identification in social media (offenseval 2020). *Proceedings of SemEval*.

Ziqi Zhang and Lei Luo. 2019. Hate speech detection: A solved problem? the challenging case of long tail on twitter. *Semantic Web*, 10(5):925–945.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657.

Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (volume 2: Short papers)*, pages 207–212.