OctoTweet – Tweeter sentiment analysis

András Ecker, Valentin Vasiliu, Ciprian I. Tomoiagă

***Abstract*—This project presents two modern approaches for predicting sentiment valence in tweets. The first one is based on the popular *doc2vec* framework with a conventional classifier on top, while the second and more performant one uses a convolutional neural network architecture with softmax classifier. Both methods received good evaluation on the final test, obtaining accuracy scores above 75% on the Kaggle competition platform, with the best of 86.5%. This placed our team in top 10, with a total of only 15 submissions. This report presents our research, the data processing, implementation details and comparisons of the methods.**

I. Introduction

Given a training set of 2.5 million tweets, equally split as positive and negative examples, we aim to train a classifier which correctly predicts the sentiment valence of new instances. The labeling of the data was done according to smileys it contained: '**:)**' for positive examples and '**:(**' for negative ones. For the solution we iterated through a number of ideas, from more classical ones using manually constructed features and an SVM classifier, to more modern ones using word embeddings and neural networks. Having spent more than a week on literature review, we settled on two of the modern approaches: distributed bag of words with `word2vec` embeddings [1] (**Doc2Vec**) and convolutional neural networks with `GloVe` word embeddings [2] (**CNN**). The following sections detail each of the approaches, illustrating the challenges we faced and how we mitigated them. It is important to note that both models are very powerful but computationally expensive to train, especially with the given data set, on a laptop. As such, we decided to follow best practices for *most* hyper-parameters to be tuned, and experiment with the more important ones on a smaller sample of the training data.

II. Data pre-processing

The training and test data sets were provided to us in a pre-processed form which put all text in lower-case, replaced all user mentions and hyperlinks with standard tokens, and standardised white space between tokens (words or punctuation marks). We decided to further process the texts to remove unnecessary variance and make it more suitable for the task.

A. Doc2vec

In the *doc2vec* case the pre-processing pipeline consists of further removing any stopwords present in a tweet, along with all non-letter characters and all tags such as <*url*> or <*user*>. An argument could be made for keeping characters such as punctuation or tags, as there have been indications in the literature that such elements might be helpful if kept, however it would require rebuilding the embeddings for each new preprocessing step and that takes a considerable amount of hardware resources. Improvements could be made to the *doc2vec* model if multiple pre-processing pipelines were to be tested in a reasonable amount of time.

B. CNN

For this task, we noted firstly the distribution of tweets' word count in the training set, which had a very long tail due to outliers (see Figure 1). As these were very few, it is safe to ignore them without losing important information. On the contrary, since the feature vector of a tweet is a concatenation of its word vectors (using zero padding), this simple step saved more than half of the memory used during training and also increased the variance of the inputs.

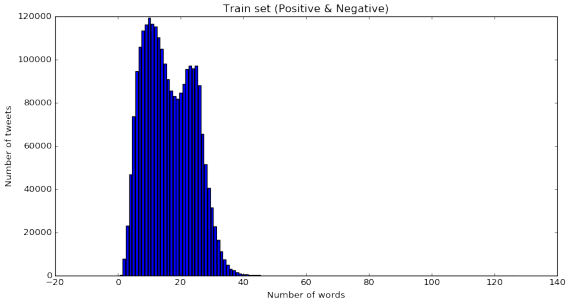

Figure 1. Distribution of tweets based on their word counts. A very fine right tail is present, with the longest tweet having 128 words.

Then, we noted there were numerous duplicated entries, accounting for $\sim 10\%$ of our training data; they were also ignored.

Finally, we introduced additional tokens to mark hashtags, additional smileys, repeated punctuation and elongated words.

Overall, the pre-processing improved our predictions with $\sim 2\%$. We also considered removing stop-words and infrequent punctuation as in the *doc2vec* case, but finally we let the algorithm learn and decide their importance.

III. Training implementation

Given that the problem is non-convex by its nature, accurate models require a long time to train for finding a so-

lution and beyond consumer-grade computational resources. Therefore we employ already tested architectures and tune them only where necessary. In the following part we present our studies, in parallel.

A. Doc2vec

The model's implementation followed closely Michael Czerny's article [3] and Linan Qiu's github repository [1]. Since their appearance a few years ago, word embeddings have become a popular choice in natural language processing and text analysis. We continue this trend by using the most recent version based on the established *word2vec*. The main advantage of *doc2vec* is the fact that it can represent not only words but also sentences and paragraphs, by preserving information on the word order. There are two main methods in the *doc2vec* implementation: Distributed Memory which attempts to predict a word given a series of preceding words and a paragraph vector, and Distributed Bag of Words which attempts to predict a series of words in a paragraph given only the paragraph vector.

Python's *gensim* library contains a *doc2vec* implementation and it works in two steps. First step requires a dataset from which to build its vocabulary, which could be either imported from an external source or it could consist of the whole twitter corpus at our disposal. The second option was picked as it seemed the most natural choice and it prevented unwanted bias that could have been introduced from another corpus of tweets. Next step requires training over the corpus of sentences passed to the model (i.e. preprocessed tweets). According to the above sources multiple passes (epochs) through the data with a randomized sentence order improves the model's training.

The model was initialized with the following parameters: $min\_count = 1$, $window = 10$, $size = 100$, $sample = 10^{-4}$, where min_count is a threshold for minimum word frequency (as we are dealing with sentences it is set as 1), $window$ is the maximum distance between the current and predicted word within a sentence, $size$ is the length of the feature vector for each sentence (i.e. tweet), and $sample$ is the threshold for configuring which higher-frequency words are randomly down-sampled. It is worth knowing that these are only a few of the parameters that could be altered in the initialization of the model. After performing 10 runs (epochs) through the corpus of tweets we obtain 100 dimensional feature vectors for each tweet.

After obtaining the feature vectors for each tweet we can use standard classification algorithms to train a classifier based on the embeddings. For this part, we tested Logistic Regression, Naive Bayes and Random Forests. We also considered SVMs but soon realized that the *sklearn* implementation was not practical for the current task. [2]

B. CNN

Despite the well-established importance of feature engineering in machine learning tasks, it proves to be a non trivial task for language processing. CNN models, originally developed for image processing tasks, have subsequently been shown to outperform all manually developed models for this field [4]. By learning weights of convolutional filters that 'slide' through the sentences, the network does the feature engineering part automatically.

We based our architecture on the theory developed by [4] and we implemented it in *tensorflow* [3] following the model of Denny Britz's open source repository [4]. We extended that work to fit our task by adding pre-trained word embeddings and customised model parameters.

The architecture consists of 4 stages, schematised in Figure 2: 1) an input layer, feeding sets of pre-processed tweets; 2) a word embedding layer, mapping tokens in a sentence to their pre-trained, 200-dimensional `GloVe` representations [5]; 3) a convolutional layer with different filter sizes; 4) a *softmax* classification layer; 5) and a final output layer which gives our result. Note that the convolutional step includes a max-pooling layer which keeps only the best features from the filters. Also, our experiments showed the network achieves better results if we allow further training of the word embeddings in the second layer.

The *softmax* function appears often as the output layer of a neural network and it is a generalized version of the logistic one, having as outputs normalized class probabilities. It also has the desirable property of minimizing the cross-entropy between the predictions and the truth, as well as a nice probabilistic interpretation; namely it finds the Maximum Likelihood Estimate:

$$P(y = j|x) = \frac{exp(x^T w_j)}{\sum_{k=1}^{K} exp(x^T w_k)} \quad (1)$$

where, K is the number of clusters and j is the index of the "current" cluster.

For computing the weights of the network, we chose ADAM (Adaptive Moment Estimation) optimiser [5], which computes adaptive learning rates for each parameter and is shown to converge faster than other methods. We used the implementation in *tensorflow* with the default parameters.

In contrast with Kim's work, we did not use L_2 regularization, only dropout method, due to its popularity for

[1] https://github.com/linanqiu/word2vec-sentiments/blob/master/word2vec-sentiment.ipynb

[2] From the documentation of SVM sklearn: "The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples."

[3] An open source software library for numerical computation using data flow graphs https://www.tensorflow.org/

[4] https://github.com/dennybritz/cnn-text-classification-tf

[5] Available on the library's website http://nlp.stanford.edu/projects/glove

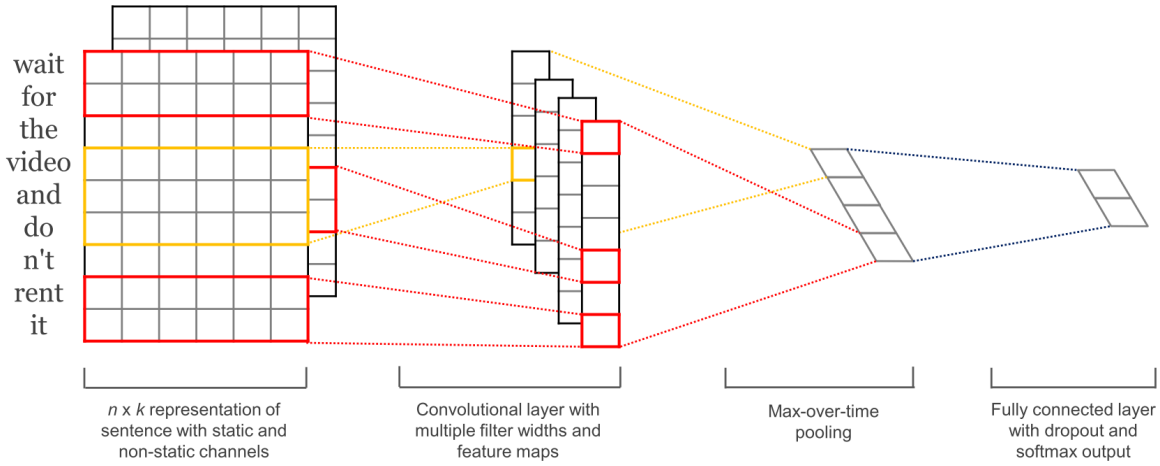

Figure 2. Model architecture (adapted from Kim 2014 [4])

convolutional neural networks. A dropout layer stochastically "disables" a fraction of its neurons, which prevents neurons from co-adapting and forces them to learn useful features individually [6]. This was applied with probability of dropping $p = 0.2$.

Finally, with the previously described network architecture in place (detailed in Figure 3), all that remains is to set and tune the parameters of the model. For the reasons motivated in the beginning of the section, we followed advice from [7] which presents a detailed sensitivity analysis for text classification. The settings that we **did** experiment with are: 1) embedding type (word2vec vs. GloVe); 2) embedding dimensions; 3) allowing or forbidding training of the embedding layer; 4) maximum number of filters and their sizes 5) batch size. The configuration that yielded best results is the following: 64 filters for each filter size in $3, 4, 5$ with max pooling over them, resulting in 192 features for the Softmax Classifier. The whole model, including the word vectors, was trained with ADAM optimizer in batches of 128 samples per iteration, for a total of 10000 steps.

IV. Evaluation

A. Doc2vec

Initial results ran on subsamples of the twitter datasets showed an accuracy between 62-64% for a 80/20 training and testing dataset split of the tweet corpus and using the classifiers mentioned in the previous section. On the full datasets the accuracy increased to 68-70% using Random Forests with 100 trees (the other metaparameters were kept as default). Given a testing framework which could run such models considerably faster significant improvements could be made in the future to the *doc2vec* model.

B. CNN

For assessing the performance of our model we applied cross-validation on a subset of 2000 tweets. We monitored the cross-entropy loss and accuracy on the validation set using TensorBoard. While they do slowly converge (Figure 4), small oscillations can be observed, as expected from a non-convex, stochastic optimisation. Increasing the batch size could have potentially stabilised the problem and increased convergence rate. In the end, the final measure for us was the Kaggle score in every case.

According to the accuracy estimation, the best result can be obtained after 9600 steps which corresponds to 86.38% accuracy on Kaggle, while the best model according to the loss function is the one trained in 9900 steps, achieving 86.45% on Kaggle. The accuracy after 9900 iterations is estimated to be lower, at 85.2% Kaggle score, which indicates that the dropout method successfully avoids overfitting.

V. Conclusion & Future work

We researched, implemented and compared two modern but different approaches for sentiment analysis on social media data and evaluated them comparatively on the same test scenario. Both performed well, with the CNN placing us in top 10 in the competition. Additionally, we looked at their their key features and parameters and used sensible validation metrics to analyse the two methods, as indicated by our low submission count in the competition.

Given more time and resources, it would be interesting to experiment with different input pre-processing and more robust cross-validation techniques.

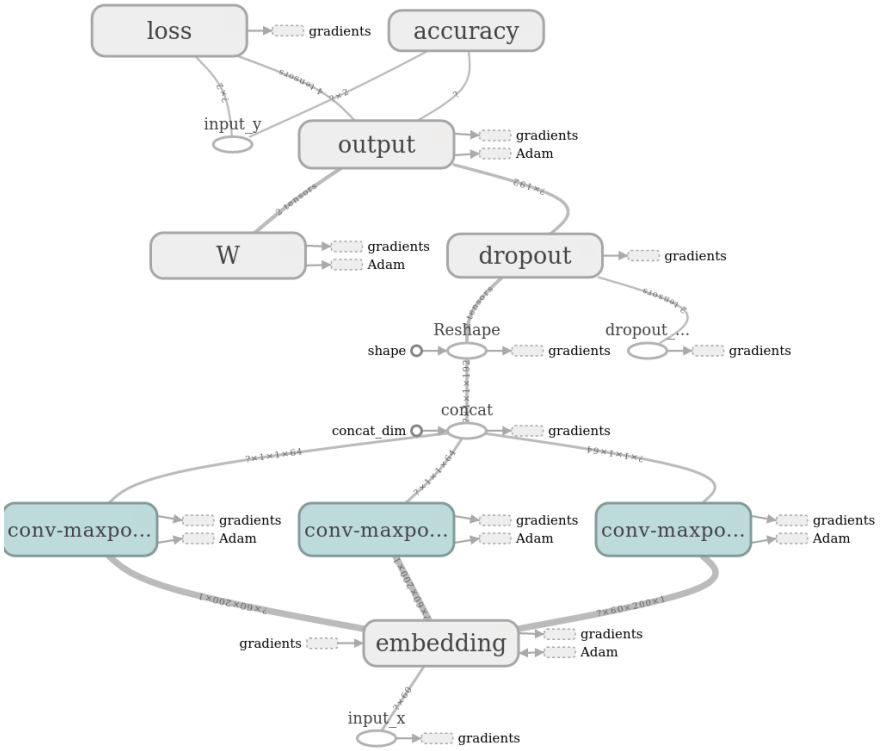

Figure 3. Graph of the final model (produced with TensorFlow's visualization tool: TensorBoard)

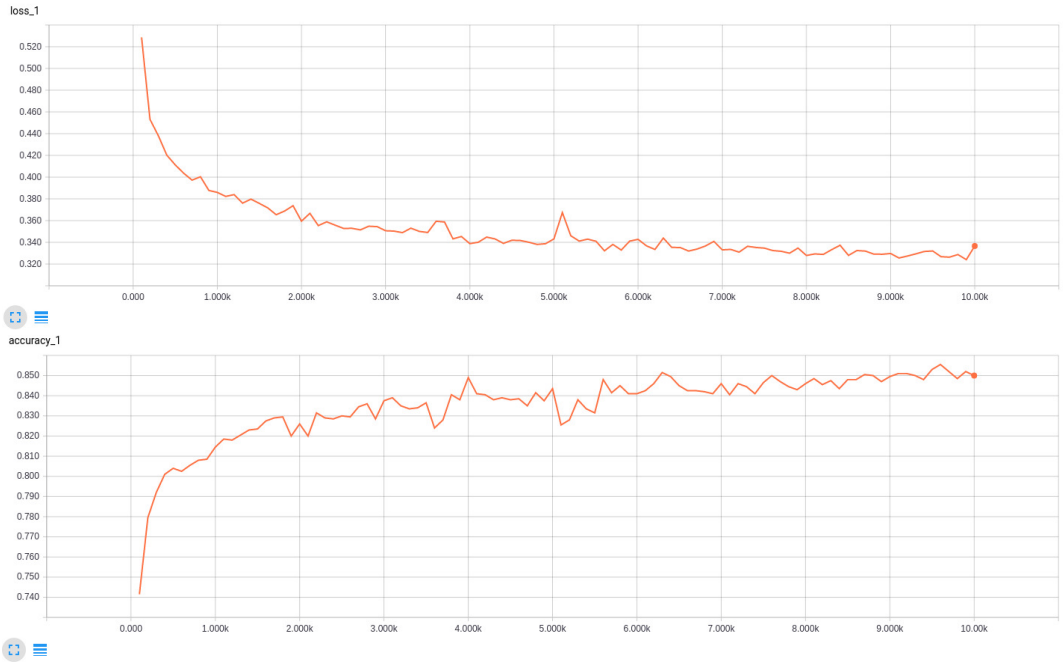

Figure 4. Cross-entropy loss and accuracy of the final CNN architecture - tested on 2000 tweets (produced with the TensorBoard visualization tool)

REFERENCES

[1] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents." in *ICML*, vol. 14, 2014, pp. 1188–1196.

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[3] M. Czerny, "Modern methods for sentiment analysis," https://districtdatalabs.silvrback.com/modern-methods-for-sentiment-analysis, 2015, [Online; accessed 21-December-2016].

[4] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014, pp. 1746–1751. [Online]. Available: http://emnlp2014.org/papers/pdf/EMNLP2014181.pdf

[5] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," in *International Conference on Learning Representations 2015*, 2015, pp. 1–15.

[6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[7] Y. Zhang and B. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," 2016. [Online]. Available: http://arxiv.org/abs/1510.03820