CIL 2018: Text Sentiment Classification

Aryaman Fasciati, Nikolas Göbel, Philip Junker, Pirmin Schmid
The Optimists
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—In this work we consider the task of classifying tweets by sentiment. In particular we want to determine whether a given tweet is associated with a positive smiley ":)" or a negative smiley ":(" after having been stripped of these emoticons. We present two baseline models, a random forest classifier and a simple stacked GRU/RNN model, and a third more complex neural network combining GRU/RNN with convolutional layers and deeper NN. We trained these models on GloVe word embeddings, pre-trained on 2.5 billion tweets. Two baselines were implemented as a reference for the prediction accuracy of our final model. Our first baseline (random forest) achieved an accuracy of 72% based on the initial first half of the test set. Our second baseline (RNN/GRU) achieved an accuracy of 85.2%. Our best final model achieved an accuracy of 87.4% with this test set. After the submission deadline, an accuracy of 86.6% was revealed for our model based on the second half of the test set that was secret during development.**

I. Introduction

The global prevalence of social media has enabled a massive amount of users to publicly voice their opinions and interact with each other. Microblogging services like Twitter[1] make it very easy for users to comment on current events, tell other users about their lives, and discuss the products and services they use. This data can be analysed for sentiments and opinions and can be used for various purposes, such as consumer insight [1] or stock market prediction [2].

The goal of this project is to build a sentiment classifier that predicts whether a Twitter tweet used to include a positive smiley :) or a negative smiley :(, based on the remaining text.

II. Related Work

Recurrent neural networks (RNN) with their applicability to sequence-to-sequence (seq2seq) tasks have been used across many tasks, especially in the natural language processing (NLP) domain, such as generating text [3]. Although RNNs seem to be primarily used for problems more complex than binary classification, we were interested in exploring the application of short-term memory to our task. Because the sentiment of a sentence often depends on more than one word we assumed that this will lead to a better sentiment prediction of our model. This can be achieved via the Long short-term memory (LSTM) or Gated recurrent unit (GRU) variants of RNNs. Both have found extensive use in practical applications, such as predictive chat replies [4], question answering [5], and video to text [6].

III. Models

A. First Baseline (B1)

Our first baseline uses a random forest model with unlimited max_depth and 20 estimators. We used the random forest implementation provided with the scikit-learn library [7].

Tweets from the datasets are run through a pre-processing script, provided by the GloVe project. The pre-processor normalizes tweets, for example by replacing user names with a single token indicating a mention. More importantly, symbols used commonly in tweets, such as the heart "$<$3", are converted into tokens that are contained in the vocabulary.

Input tweets are then split into words. Each word is looked up in the embedding dictionary. We use 200 dimensional word embedding vectors. Words that are not in the vocabulary are ignored. All word embeddings for a given tweet are then aggregated into a single vector, by computing their mean. Word embeddings are provided by the GloVe [8] project from Stanford, pre-trained on two billion tweets.

The classifier is then trained on these tweet embeddings. We expected this relatively simple approach to already perform quite well, because a small number of features of single words would seem to have an impact on overall tweet sentiment in practice. Still, aggregating word embeddings destroys a lot of information contained in the order of words. In particular, this approach does not differentiate between permutations of similar words at all. It also assigns equal weight to every word when computing the mean, whereas we would commonly expect a small subset of words to be disproportionally strong indicators of sentiment ("love", "hate", "good", "bad", etc.).

This model achieved an accuracy of 72% with the initial half of the test set.

B. Second Baseline (B2)

For the second baseline, we trained a stacked, recurrent neural network. Two GRU cells were stacked; a hidden state size of 384 was used for each of them. In contrast to the random forest baseline, tweet embeddings were not aggregated in any way from their words. Rather, the network was trained on matrices of dimension $word\_count_{tweet} \times 200$. Words not

[1] www.twitter.com

known in the embedding vocabulary were ignored. Statistical analysis of the provided data in Figure 1 shows that the majority (>98%) of tweets have less or equal than 30 words. Tweets longer than that were ignored during training and shortened for prediction, whereas shorter tweets were padded with 0 vectors to achieve proper tensor dimensions. A dynamic RNN was used, and the word count was provided for each tweet.

All output of the RNN was used as input of a single layer fully connected NN to 2 nodes with a sigmoid activation function. Argmax was used to determine the sentiment. Loss was calculated with sparse softmax cross entropy with logits and an Adam optimizer was used for learning with clipped gradient at 10 and a learning rate of 10^{-4}. In total, the model was trained for 4 epochs on the Leonhard cluster. 98% of the training data was used for training; 2% was used for evaluation.

Smaller and larger hidden state sizes were tested but resulted in lower accuracy. Fewer (one) and more (three) layers also resulted in lower accuracy. Additonally, we experimented with LSTM cells, which did not result in a higher score than with GRU cells. All outputs of the RNN were used as input of the NN to bring more state information into the final classifier step. Using only the final hidden state was tested and gave lower accuracy.

Typically, a matrix multiplication (and optionally addition of a bias vector) is used to transform the output state of each RNN time step into e.g. the vocabulary space for seq2seq predictions. Here we used the above-mentioned single-layer fully-connected NN for the reduction from the quite large RNN output state to the two nodes representing the sentiments to allow for additional non-linear transformations.

We expected this RNN model to perform better than baseline B1 because much more implicit information could be used for the model, such as specific word vectors and sequences of these vectors.

Indeed, this baseline model B2 achieved an accuracy of 85.2% with the initial half of the test set.

C. RNN + Convolution

With this large jump of accuracy from B1 to B2, we were very hopeful to achieve significant improvements when adding additional creative approaches to the problem. The most important ones are listed in section IV in more detail.

Unfortunately, most of them revealed to be unsuccessful. Thus, we settled on the improved model as described in detail in this section. It provided the best accuracy of all tested models.

Our final model is based on the promising RNN model used in baseline B2. We can consider the RNN output to represent learned tweet features, which bear significance for sentiment classification. Consequently, we wanted to recombine these features in a structure-preserving, context-sensitive manner. Convolutional neural networks (CNN) fit this bill and in addition possess the property of being invariant w.r.t translation [9]. Intuitively, this sounded useful to the task at hand, because we expect certain word sequences to indicate sentiment independent of where they appear inside a tweet.

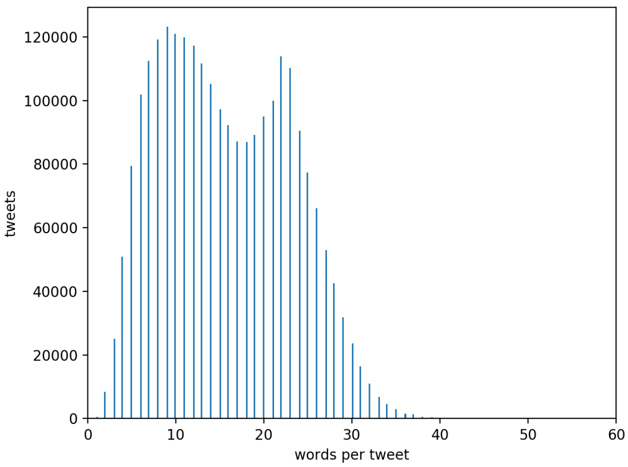

Figure 1. Word Count Distribution

We therefore introduced two one-dimensional convolution layers each followed by a one-dimensional max-pooling layer and a drop layer. We chose convolution window sizes of six and four respectively, with ReLu activation function. The max-pooling layers aggregate each pooling window into a single scalar. We chose a window size of two and strides of two for both layers.

Outputs for every window are concatenated and passed on to a fully connected rectifier. In order to lessen the impact of overfitting, we apply a dropout rate of 0.4 during training.

The final outputs are then fed into a two-unit layer, each unit indicating one of the two possible sentiments. Overall classification output is then obtained by applying the `argmax` function.

The architecture of this processing of the RNN output states follows the CNN example provided in the tensorflow documentation[2] with the difference that the tensorflow example uses a 2D convolution and we use a 1D (temporal) convolution. The architecture of our entire model is summarized in Figure 2.

We trained the model for four epochs. Loss was calculated with sparse softmax cross entropy with logits and an Adam optimizer was used for learning with clipped gradient at 10 (to avoid instabilities; see Exploding Gradients Problem[3]) and a learning rate of 10^{-4}. Training on the Leonhard cluster took around three hours.

This model achieved an accuracy of 87.4% with the initial half of the test set, which is a small improvement over the

[2] https://www.tensorflow.org/tutorials/layers
[3] https://machinelearningmastery.com/exploding-gradients-in-neural-networks/

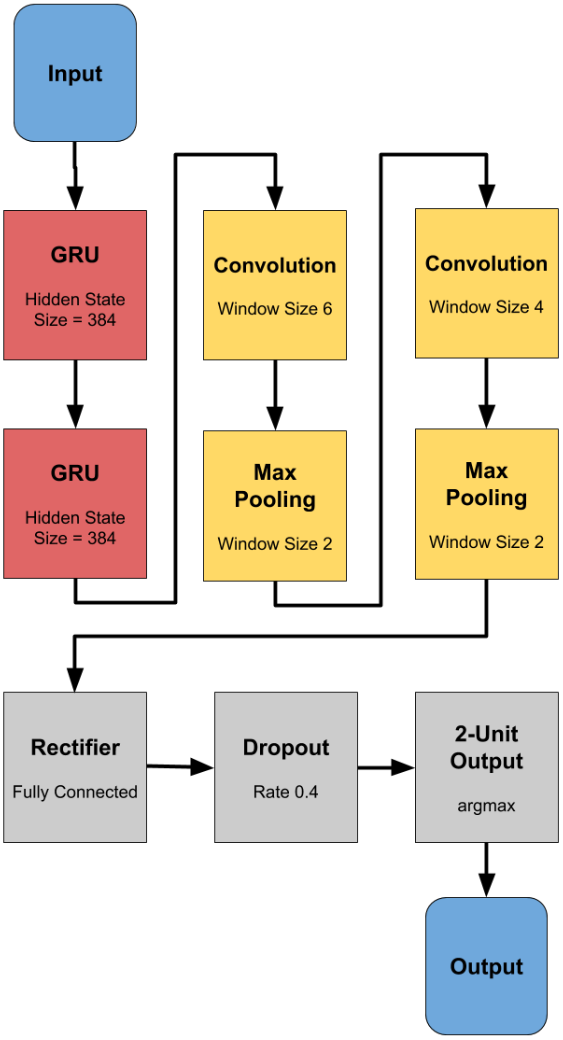

Figure 2. Our Model

baseline B2. The model achieved an accuracy of 86.6% with the second/final half of the test set.

IV. Other Approaches

The model described in subsection III-C significantly outperforms the random forest classifier described in subsection III-A and improves slightly upon the plain RNN approach in subsection III-B. In addition to these, we experimented with various other approaches. However, no additional improvements could be achieved with any of these creative methods.

Effect of word order

In particular, we were interested in exploring the tradeoff between increasing the amount of information encoded in each input (e.g. bag of words vs embedding matrix) and the additional noise invited by doing so. For example, even though it would seem that a sophisticated model like the RNN used in subsection III-B and subsection III-C should benefit from more information (word order, context) this is only true in practice assuming sufficiently large datasets and computational resources.

It was our suspicion that weighing words independently had a much stronger effect on classification performance than respecting different word orders. Therefore we re-trained the RNN model presented in subsection III-B again, but modified the pre-processing steps. Instead of providing each tweet's embedding matrix as is, we sorted the columns by vector norm. The idea was to normalize word order, while still preserving the ability to assign individual weights to words. This corresponds to a restriction of the usual sequence-to-sequence formulation for RNNs to a bag-of-words model.

This model achieved an accuracy of 85.2%, indicating that word order was not the primary contributor to accuracy, but also showing that the RNN did indeed benefit from the additional input information given by the word sequence in the tweet.

Detecting negation

We also tried to identify words within the scope of a negation using the `nltk.sentiment.util.mark_negation()` method from NLTK[4]. Embedding vectors of negated words would then be multiplied by -1.0 before feeding them into the RNN. Although detecting negation scopes and even simple cases of double negation seemed to work very robustly, the resulting model had a bit a lower accuracy than chosen final model presented in subsection III-C: 86.7%. On the other hand, this result underlines the power of the LSTM/GRU cells inside the RNN, which seem to be able to keep track of negation scopes automatically.

Additional training of the GloVe vectors

For most models, we used the pre-trained 200 dimensional GloVe embeddings as constants that were looked up within the tensorflow model. We tested additional training on these pre-trained embeddings. Due to memory limitations of the GPU on Leonhard, we could only use 100 dimensional GloVe embeddings for that test, which did not improve the accuracy with the first half of the test set.

Additional embeddings

Finally, we also tested additional embeddings. We considered the remaining emojis in the tweets of particular interest. Thus, we created an additional embedding vector that encoded the counts of the known emojis. We could not

[4] https://www.nltk.org/

find precedence for this sort of processing in the literature, but deemed it worthwile to attempt.

Additionally, we tested whether sentiment classification using lexicons of known positive and negative words could be used to improve our classifier. Basically a difference value was calculated from the count of positive words in a tweet minus the count of negative words. Defined words like ”not” were used to invert this value.[5] This concept follows the ideas described in the paper of Chaturvedi et al. 2017 [10] and the available source code [11].

We tested both additional embeddings in various variants whether they could improve the accuracy of the model. First, we tested them as additional inputs of a final NN that took the results of baseline 2 model and these additional small embedding vectors to get the final 2 sentiment nodes. `argmax` for prediction, loss, and optimizing functions for training were used identically as defined above.

Second, we tested these two additional embeddings in a small one layer NN to generate a non-zero initial hidden state for the RNN. The rest of the model was identical to baseline 2. Here, the idea was tested whether such key information could be used to train relevant biases to the afterwards running RNN similar to the human brain that may take such key words as a first bias/primer when reading the entire tweet.

Despite theoretical considerations and despite the added complexity, both variants did not improve the accuracy of the models over the baseline B2.

V. Conclusions

Overall our additions do improve on the simple RNN baseline established in subsection III-B, but not by much. Both RNN models do however improve significantly on the random forest classifier. This could hint at the strength and relative ease of an approach based on taking short-term memory into account.

Despite testing various additional published and creative methods in addition, we could not improve accuracy over the chosen final model.

A drawback of the more complex model (with respect to the relatively small increase in accuracy) is the increased amount of available hyper-parameters that can be tuned in the model, such as stack size, convolution and pooling window sizes, stride lengths, dropout rate, activation functions, etc. These hyper-parameters would have to be empirically determined (via a grid search, for example). In combination with the computational resources required to train the model, this makes it harder to further improve the model in a structured way.

Acknowledgements

The authors wish to express their gratitude to the Euler and Leonhard clusters at ETH, without whose unwavering computational effort this project could not have been done.

References

[1] W. Chamlertwat, P. Bhattarakosol, T. Rungkasiri, and C. Haruechaiyasak, “Discovering consumer insight from twitter via sentiment analysis,” *J. UCS*, vol. 18, pp. 973–992, 2012.

[2] T. Rao and S. Srivastava, “Analyzing stock market movements using twitter sentiment analysis,” in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ser. ASONAM ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 119–123. [Online]. Available: http://dx.doi.org/10.1109/ASONAM.2012.30

[3] I. Sutskever, J. Martens, and G. Hinton, “Generating text with recurrent neural networks,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML’11. USA: Omnipress, 2011, pp. 1017–1024. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104482.3104610

[4] P. Khaitan, “Chat smarter with allo,” https://ai.googleblog.com/2016/05/chat-smarter-with-allo.html, (Accessed on 3/07/2018).

[5] D. Wang and E. Nyberg, “A long short-term memory model for answer sentence selection in question answering,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, 2015, pp. 707–712. [Online]. Available: http://www.aclweb.org/anthology/P15-2116

[6] S. Venugopalan, M. Rohrbach, J. Donahue, R. J. Mooney, T. Darrell, and K. Saenko, “Sequence to sequence - video to text,” *CoRR*, vol. abs/1505.00487, 2015. [Online]. Available: http://arxiv.org/abs/1505.00487

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation.” in *EMNLP*, vol. 14, 2014, pp. 1532–43.

[9] E. Kauderer-Abrams, “Quantifying translation-invariance in convolutional neural networks,” *CoRR*, vol. abs/1801.01450, 2018. [Online]. Available: http://arxiv.org/abs/1801.01450

[10] S. Chaturvedi, H. Peng, and D. Roth, “Story Comprehension for Predicting What Happens Next,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1603–1614.

[11] S. C. et al, “StoryComprehension EMNLP,” https://github.com/snigdhac/StoryComprehension_EMNLP.

[5] A more extended version of this additional Python code written by the same author (PS) had been used in his NLU group project 2 as described there. For this project here, no n-grams were built for multiple sentences, of course. Only the plain score for each tweet was used.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

CIL 2018: Text Sentiment Classification

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Fasciati	Aryaman
Göbel	Nikolas
Junker	Philip
Schmid	Pirmin

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge“ beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum

Zürich, 6. Juli 2018

Unterschrift(en)

P. Schmid

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.