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, which has been stripped of emoticons, used to include a positive smiley ":)" or a negative smiley ":(". We present three models, a random forest classifier, a simple stacked GRU/RNN model, and a more complex neural network combining GRU/RNN with convolutional layers. We train these models on GloVe word embeddings, pre-trained on 2.5 billion tweets. Our final model achieves a classification accuracy of 86.6%.**

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 brand monitoring or stock market prediction.

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.

Our first baseline uses random forests and achieved an accuracy of 72%. Our second baseline uses a Recurrent Neural Network (RNN) model with an accuracy of TODO: rnn-baseline-accuracy.

In a third model, we refined our second baseline, incorporating TODO: describe novel approach, in a RNN-based approach. This model achieved TODO: % accuracy.

II. Related Work

Recurrent neural networks with their applicability to sequence-to-sequence tasks have been used across many tasks, especially in the NLP domain, such as generating text (TODO: Sutskever et al.). 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. 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 TODO: cite https://ai.googleblog.com/2016/05/chat-smarter-with-allo.html.

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 [1].

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 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 [2] 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%.

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. 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 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 than 30 words. Tweets longer than that were ignored during training and shortened for prediction, whereas shorter tweets were padded.

[1] www.twitter.com

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 Leonhard. 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. Additonally, we experimented with LSTM cells, which did not give us 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 hidde state has been tested and gave lower accuracy. And finally, a small NN was used to handle the binary classification from the RNN state to allow also for non-linear transformation in contrast to the classic transformation with a matrix multiplication and potentially adding of a bias offset.

pisch: not yet sure about this discussion points here In contrast to B1, this model does take word order into account and allows for different weights to be assigned to certain, sentiment-implying words, during training. Intuitively, we would expect the latter to result in significantly higher classification accuracy, compared to the random forest model. It is not as clear, whether taking word order into account during training is important for the task of sentiment classification and would thus dilute inputs in a sense.

This model achieved an accuracy of 85%.

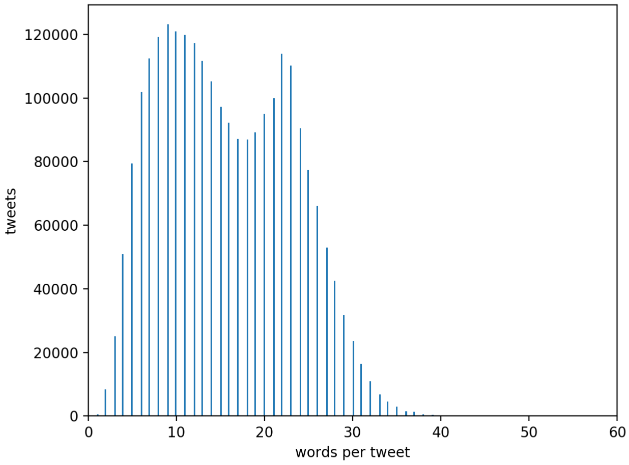

Figure 1. Word Count Distribution

C. Our Model

For our final model, we decided to expand on the promising RNN approach taken in baseline B2. We can consider the RNN output to represent learned tweet features, which bear significance for sentiment classification. Consequently, we wanted to re-combine these features in a structure-preserving, context-sensitive manner. Convolutional neural networks fit this bill and in addition possess the property of being invariant w.r.t translation. 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.

We therefore introduced two one-dimensional convolution layers each followed by a one-dimensional max-pooling layer. We chose convolution window sizes of six and four respectively, with no bias and ReLu activation. 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 our model is summarized in Figure 2.

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. Further investigation is neccessary to determine, whether the improvements are due to the GRU units or simply a result of the more powerful input model (sequence of word embeddings instead of a simple bag of words).

A drawback of the more complex model (with respect to the relatively small increase in accuracy) is the increase in parameters (stack size, convolution and pooling window sizes, stride lengths, dropout rate, activation functions, etc...) that 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 very hard 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] 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.

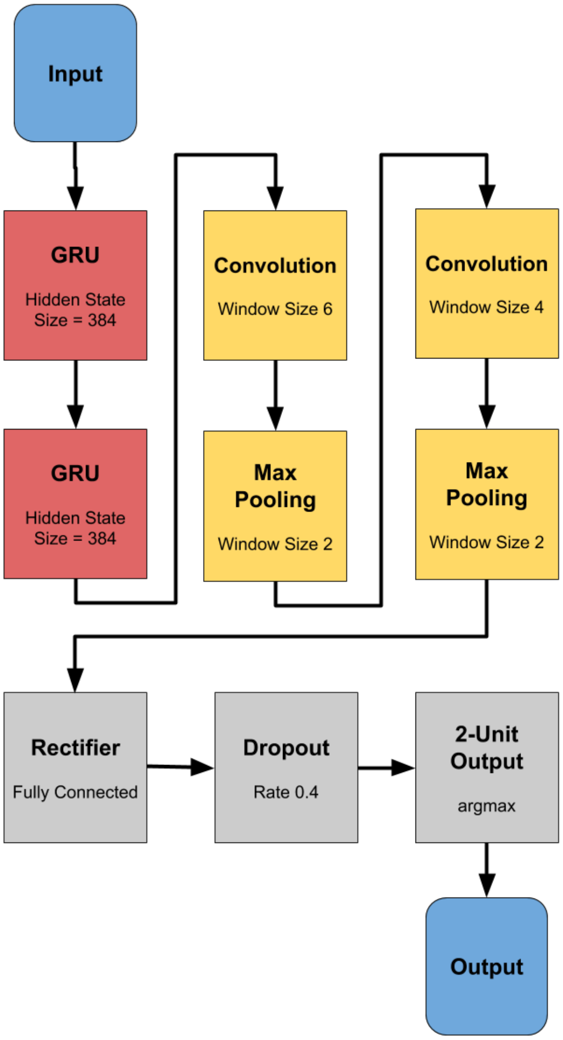

Figure 2. Our Model

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