CIL PROJECT: SENTIMENT ANALYSIS

Jovan Andonov, Frédéric Lafrance, Jovan Nikolic

Department of Computer Science
ETH Zürich
Zürich, Switzerland

ABSTRACT

The problem of understanding opinions and polarities from text data, known as sentiment analysis, has applications in domains such as market prediction and brand monitoring. Word embedding techniques have been successfully applied to many natural language problems. It therefore seems natural that the same could be done with sentiment analysis. In this work, we focus on binary classification of tweets in positive or negative sentiment. Starting from a simple baseline, we use recurrent neural networks to achieve a classification accuracy of nearly 90%.

1. INTRODUCTION

The increasing availability and popularity of social media enables Internet users of various backgrounds to publicly express their feelings and opinions, political or religious views, and discuss products and services. The posts they make usually express either positive / favorable or negative / unfavorable opinions toward the subject of discussion. Automatically mining these opinions, also known as sentiment analysis, has uses in various contexts: studying popularity of political leaders [1], making stock market predictions based on Twitter mood [2], brand monitoring [3] and even forecasting box-office revenues for movies [2]. This makes social media valuable sources of societys attitudes.

In this work, we focus on binary (positive/negative) classification of tweets. We start from a simple baseline and identify multiple issues related to word ordering and structure. These issues motivate us to move to a model based on recurrent neural networks (RNNs), which are known for being able to capture longer-term dependencies in text. We further improve this approach using state-of-the-art mechanisms such as attention, and obtain higher prediction accuracy.

2. MODELS

For this project, we implement three approaches: one simple baseline and two RNN-based extensions.

2.1. Baseline

The baseline model treats each tweet as unordered collection of words. As is common for tasks in natural language understanding, each word is represented as a dense vector called an embedding. To obtain a fixed-size representation for a tweet, we simply average the word embeddings of its contents, yielding a tweet embedding. Such an embedding can then be used with a standard classifier, where each dimension in the embedding is another feature for the classifier. We choose to use the random forest classifier available in Scikit-Learn [4].

2.2. Recurrent neural network

The baseline model suffers from several drawbacks. It considers every word of a tweet with the same weight, which means that words a human would consider more important to determine sentiment (adjectives such as “good”, “great”, “bad”, “poor”, and verbs such as “enjoy”, “dislike”, etc.) are not given extra weight. Further, word ordering is lost when averaging, which means that the classifier might be confused by more subtle or seemingly contradictory constructions. For instance, the sentence “The story started out great, but after that it went pretty terribly” has the same words as the sentence “The story started out pretty terribly, but after that it went great”, but seems to express an opposite polarity. By throwing away word order, the baseline cannot understand the difference between these two.

Recurrent neural networks (RNN) are a way of dealing with this problem. A RNN is a special kind of neural network that receives a sequence as an input, and maintains an internal state depending not only on the current element of the sequence but also the previous value of the state. This means that the network is able to learn the aforementioned complexities of sentence structure.

In practice, however, simple RNNs have difficulty learning longer-term dependencies between elements of a sequence [5]. Various models have been put forward to address this issue. Perhaps the most famous is the Long Short-Term Memory (LSTM) unit [6], which can be viewed as a “gated” unit, where the current sequence input and past

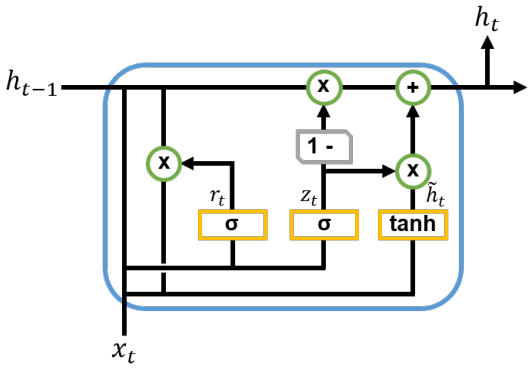

Fig. 1: Graphical interpretation of the GRU

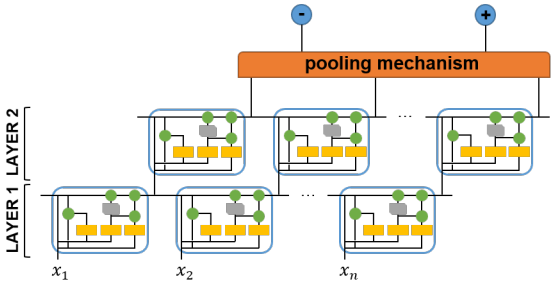

Fig. 2: Graphical visualization of the final model

state determine how much of the previous state to take into account, how much of the input to use when changing the state of the cell, and how much of the state to reveal to further layers in the network.

However, LSTMs have the downside of being complicated to understand and having long training times due to a large number of parameters. For this reason, our model uses the Gated Recurrent Unit (GRU) [7], which is conceptually simpler. The GRU is illustrated in figure 1. The input vector of the cell at time t is designated as x_t, while the cell state from the previous step is designated as h_{t-1}. The GRU is based on "reset" and "update" gates. The "reset" gate indicates how much of the previous state should be brought in, and is defined by:

$$r_t = \sigma(W_r \left[h_{t-1}; x_t\right])$$

Where $[;]$ represents vector concatenation, σ an element-wise logistic sigmoid function and W_r a learned weight matrix. Similarly, the "update" gate is defined as:

$$z_t = \sigma(W_z \left[h_{t-1}; x_t\right])$$

Since the sigmoid produces output between 0 and 1, the gates can be viewed as producing "masks" to combine with other vectors. It is the actual combination of these masks that determine what operation the gate is performing. The GRU first computes a "proposed" new state $\tilde{h}_t$ as follows:

$$\tilde{h}_t = tanh(W \left[(r_t \odot h_{t-1}); x_t\right])$$

Where $tanh$ is applied element-wise and $\odot$ represents element-wise multiplication. We here see the effect of the "reset" gate which selectively preserves or resets dimensions of the previous hidden input.

The new cell state is the determined as:

$$h_t = ((1 - z_t) \odot h_{t-1}) + (z_t \odot \tilde{h})$$

In this case, the "update" gate discriminates between either taking the previous state or the proposed new state for each dimension.

2.3. Pooling and attention

One downside of the previously described model is that it does not take into account the entire sequence of hidden states when making a prediction. Rather, the entire sequence is encoded into one vector which is then used for prediction. We can instead take into account the sequence of hidden states (in our case, 40 states) to make the prediction. We experiment with two ways of going about this:

Average pooling. We can use the mean hidden vector over the sequence to compute a final vector that we use for prediction. However, this necessarily causes some information to be lost.

Attention. We can use a weighted average mechanism in which the weights are learned as part of the training. This has the advantage of giving the network more flexibility in deciding what is important and what is not when classifying a sequence. Attention models have been successfully used for machine translation [8], and we believe that they might also help in the setting of sentiment analysis, since not all words of a given sequence have an impact on the overall sentiment.

Our final recurrent model consists of a two-layer GRU followed by a pooling layer implementing either average or attention. A graphical visualization can be seen in figure 2. The TensorFlow graph for the attention model is depicted in figure 3 (the mean pooling model is similar, except without an "attention" node).

3. EXPERIMENTAL SETUP

3.1. Dataset

We use the provided dataset of 2.5 million tweets, of which half is labeled as positive and the other as negative. We reserve 10,000 of those to perform validation, and discard tweets containing more than 40 tokens (where a token can

be a word, a punctuation sign or an emoji). This leaves us with 99.955% of the original dataset.

For our recurrent models, we perform some additional processing. We restrict our vocabulary to the 20,000 most frequent tokens, and replace the other ones by a special $\langle UNK \rangle$ token. Despite there being 592,278 unique tokens, the 20,000 most frequent account for over 98% of the occurrences in the tweets. We additionally append a special $\langle PAD \rangle$ token to the end of tweets to ensure that all have the same length of 40 tokens. Contrary to the setting of language models, we do not add beginning and ending of sentence tokens, since our goal is not to predict language specifics. We also use the dataset to train our own 200-dimensional word embeddings using the word2vec algorithm [9] (specifically the continuous bag-of-words variant with negative sampling of size 5), an implementation of which is provided in the Python package `gensim` [10].

3.2. Baseline

For the baseline, we use pre-trained embeddings of dimensionality 300 distributed by Google [11], and trained on a news corpus. The random forest classifier consists of 200 decision trees with a maximum depth of 10 nodes. For training, we only use a sample of 10,000 tweets equally divided between positive and negative. If a word of a tweet is not in the vocabulary present in the pre-trained embeddings, we do not consider it when calculating the average vector for the tweet embedding.

3.3. Recurrent models

While training the mean pooling model we used a batch size of 128 samples, and for the attention model we used a batch size of 100 samples. A hidden state size of 512 and an unrolling factor of exactly 40 (corresponding to the size of our processed tweets) was used in both models. We train for 4 epochs, optimizing the cross-entropy using Adam [12]. In addition, we clip the norm of resulting gradients to 10 to prevent exploding gradients [13].

As training advances, we keep track of the cross-entropy loss as well as the classification accuracy on training and validation sets. This helps us determine when the model begins to overfit to avoid training for too long. We save model checkpoints every 1500 batches and generate a submission file after every epoch.

4. RESULTS AND DISCUSSION

We now report the performance of our models. In this section, the "test set" refers to the data publicly visible on Kaggle, and the corresponding results are those publicly visible on the leaderboard.

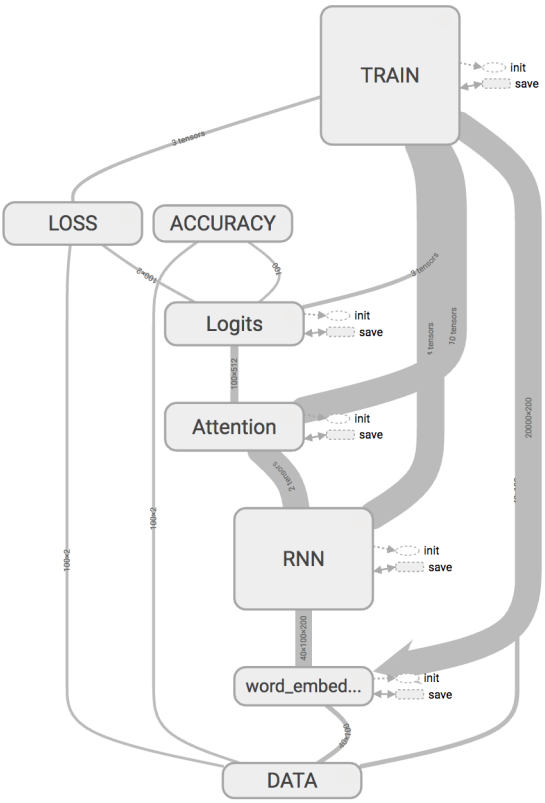

Fig. 3: Tensorflow graph for the GRU RNN with attention

Acc/Loss \ Dataset	Training	Validation	Test
Accuracy	0.906	0.918	0.889
Loss	0.217	0.209	-

Table 1: Accuracy and loss using mean pooling.

The random forest baseline model obtains a test accuracy of approximately 70%, which is significantly worse in comparison to the RNN models. Therefore, in this section, we focus on the performance and differences between the two RNN models (with mean pooling and attention).

4.1. GRU RNN with mean pooling

By looking at the accuracy (fig. 4) and loss (fig. 5) curves for the mean-pooling model, we see that it starts overfitting after approximately 80,000 batches, when the training accuracy dominates the validation accuracy. This implies that our model is learning the training data by heart and will therefore generalize poorly. Having this in mind, we decide to use the model obtained after training for 4 epochs, which in this case corresponds to roughly 80,000 batches. Table 1 details the results on all datasets.

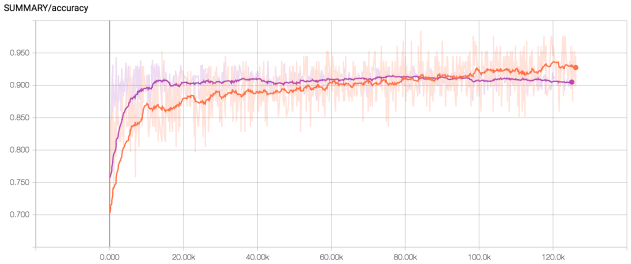

Fig. 4: Training (orange) and validation (purple) accuracy curves using mean pooling.

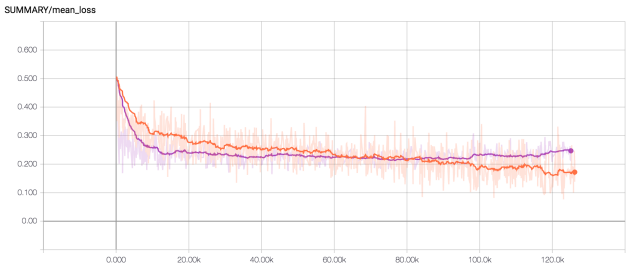

Fig. 5: Training (orange) and validation (purple) loss curves using mean pooling.

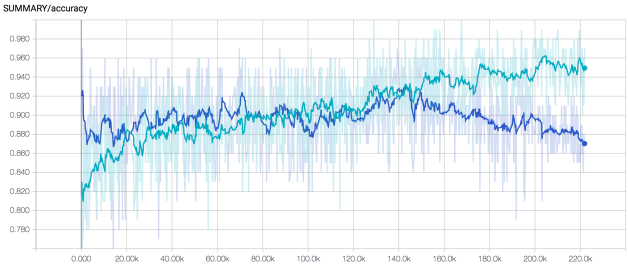

Fig. 6: Training (green) and validation (blue) accuracy curves using attention.

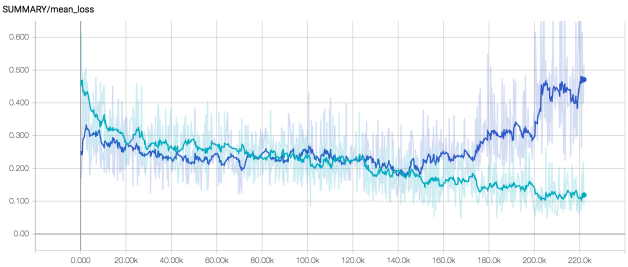

Fig. 7: Training (green) and validation (blue) loss curves using attention.

4.2. GRU RNN with attention

Analyzing the accuracy (fig. 6) and loss (fig. 7) curves, we note that the model starts overfitting either around 100,000 batches (which corresponds to epoch 4) or around 140,000 batches (which corresponds to halfway between epochs 5 and 6) . To be conservative, we select the model obtained by training for 4 epochs. The results are found in table 2.

4.3. Comparison

The results obtained on the training and validation sets are slightly better for the attention model than for the mean pooling model. However, the opposite is true for the test set. We believe that this is caused by the fact that the attention model might be less stable than the mean pooling model. Indeed, introducing more parameters makes the model more prone to overfitting. The effect of this can be seen by looking at the differences between training and validation for the attention model. Although the cross-entropy loss decreases between training and validation, the accuracy remains the same. This means that the model is "choosing" to be more confident in its own predictions rather than generalizing. This could explain the lower test score for the attention model.

Acc/Loss \ Dataset	Training	Validation	Test
Accuracy	0.930	0.930	0.887
Loss	0.212	0.184	-

Table 2: Accuracy and loss using attention.

Another explanation might be the fact that we are learning an attention model that has constant weights for all tweets. However, it is certainly the case that attention should be directed to different words in different tweets. This can be solved by making the attention weights dependent on a context vector[1]. In our case, obtaining this context can be a bit tricky since we are using the hidden outputs of the network directly for classification. One avenue to explore for this is the use of a paragraph vector model [14]. Paragraph vectors, also known as document embeddings, provide a fixed-length distributed representation of a variable-length sequence of text. In this case, we would infer paragraph vectors for every tweet and use them as the context for the attention model. We did not attempt this because we were unable to obtain useful results on the classification task using paragraph vectors[2]. This goes against the results obtained by Mikolov and Le [14], who successfully used paragraph vectors for sentiment analysis on IMDB reviews. The difference might come from the fact that tweets are shorter and can be more cryptic than IMDB reviews when expressing sentiment, making it harder for a document embedding to capture the required nuances. In future work, we would certainly revisit this paragraph vector idea and try to merge it with our attention model to achieve better results.

[1] In the original machine translation implementation, this context is the current word being generated.

[2] On vectors trained with `gensim`, our accuracy was consistently around 50%.

5. REFERENCES

[1] Andrea Ceron, Luigi Curini, Stefano M Iacus, and Giuseppe Porro, "Every tweet counts? how sentiment analysis of social media can improve our knowledge of citizens political preferences with an application to italy and france," *New Media & Society*, vol. 16, no. 2, pp. 340–358, 2014.

[2] Sitaram Asur and Bernardo A Huberman, "Predicting the future with social media," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*. IEEE, 2010, vol. 1, pp. 492–499.

[3] Wilas Chamlertwat, Pattarasinee Bhattarakosol, Tippakorn Rungkasiri, and Choochart Haruechaiyasak, "Discovering consumer insight from twitter via sentiment analysis.," *J. UCS*, vol. 18, no. 8, pp. 973–992, 2012.

[4] Leo Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[6] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[10] Radim Řehůřek and Petr Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, pp. 45–50, ELRA, http://is.muni.cz/publication/884893/en.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013.

[12] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[14] Quoc V. Le and Tomas Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.