TopicThunder at SemEval-2017 Task 4: Sentiment Classification Using a Convolutional Neural Network with Distant Supervision

Simon Müller
Zurich University of Applied Sciences
Switzerland
muellsi3@students.zhaw.ch

Tobias Huonder
Zurich University of Applied Sciences
Switzerland
huondtob@students.zhaw.ch

Jan Deriu
Zurich University of Applied Sciences
Switzerland
deri@zhaw.ch

Mark Cieliebak
Zurich University of Applied Sciences
Switzerland
ciel@zhaw.ch

Abstract

In this paper, we propose a classifier for predicting topic-specific sentiments of English Twitter messages. Our method is based on a 2-layer CNN. With a distant supervised phase we leverage a large amount of weakly-labelled training data. Our system was evaluated on the data provided by the SemEval-2017 competition in the Topic-Based Message Polarity Classification subtask, where it ranked 4^{th} place.

1 Introduction

The goal of sentiment analysis is to teach the machine the ability to understand emotions and opinions expressed in text. There are numerous challenges which make this task very difficult, for instance the complexity of natural language which makes use of intricate concepts like irony, sarcasm, and metaphors to name a few. Usually we are not interested in the overall sentiment of a tweet but rather in the sentiment the tweet expresses towards a topic of interest. Subtask B in SemEval-2017 (Rosenthal et al., 2017) consists of predicting the sentiment of a tweet towards a given topic as either positive or negative.

Convolutional neural networks (CNN) have shown to be very efficient at tackling the task of sentiment analysis, as they are able to learn features themselves instead of relying on manually crafted ones (Kalchbrenner et al., 2014; Kim, 2014). Our work is based on the system proposed by (Deriu et al., 2016), which in turn is based on (Severyn and Moschitti, 2015). CNNs typically have a large number of parameters which need sufficient data to be trained efficiently. In this work, we leverage a large amount of data to train a multi-layer CNN. The training is based on the following 3-phase procedure (see Figure 1): (i) creation of word embeddings based on an unsupervised corpus of 200M English tweets, (ii) a distant supervised phase using an additional 100M tweets, where the labels are inferred by weak sentiment indicators (e.g. smileys), and (iii) a supervised phase, where the pre-trained network is trained on the provided data.

We evaluated the approach on the datasets of SemEval-2017 for the subtask B, where it ranked 4^{th} out of 23 submissions.

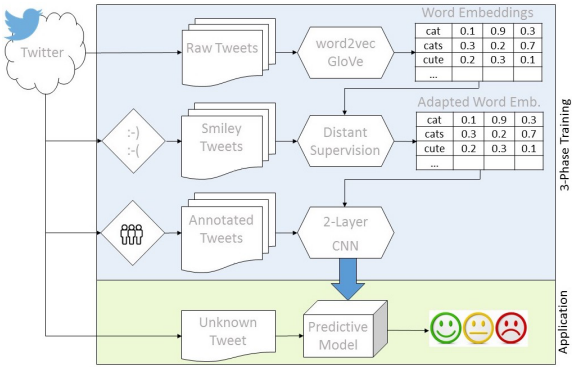

Figure 1: *Overview of the 3-Phase training procedure.*

2 Model

Our system is based on the 2-layer CNN proposed by (Deriu et al., 2016), as depicted in Figure 2 and described in details below. We refer to a layer as one convolutional and one pooling layer, thus, a 2-layer network consists of two consecutive pairs of convolutional-pooling layers.

Word embeddings. The word embeddings are initialized using word2vec (Mikolov et al., 2013) and then trained on an unlabelled corpus of 200M tweets. We apply a skipgram model of window size 5 and filter words that occur less than 15 times. The dimensionality of the vector representation is set to $d = 52$. The resulting vocabulary is stored as a mapping $V : t \rightarrow i$ which maps a token to an index, where unknown tokens are mapped to a default index. The word embeddings are stored in a matrix $\mathbf{E} \in \mathbb{R}^{v \times d}$ where v is the size of the vocabulary and the i-th row represents the embedding for the token with index $V(t) = i$.

Preprocessing The preprocessing is done by: (i) lower-case the tweet, (ii) replace URLs and usernames with a special token, and (iii) tokenize the tweet using the *NLTK-TwitterTokenizer*[1].

Input Layer. Each tweet is converted to a set of indices by applying the aforementioned preprocessing. The tokens are mapped to their respective indices in the vocabulary V.

Sentence model. Each word is associated to a vector representation, which consists in a d-dimensional vector. A tweet is represented by the concatenation of the representations of its n constituent words. This yields a matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$, which is used as input to the convolutional neural network.

Convolutional layer. In this layer, a set of m filters is applied to a sliding window of length h over each sentence. Let $\mathbf{X}_{[i:i+h]}$ denote the concatenation of word vectors $\mathbf{x}_i$ to $\mathbf{x}_{i+h}$. A feature c_i is generated for a given filter $\mathbf{F}$ by:

$$c_i := \sum_{k,j} (\mathbf{X}_{[i:i+h]})_{k,j} \cdot \mathbf{F}_{k,j} \qquad (1)$$

A concatenation of all vectors in a sentence produces a feature vector $\mathbf{c} \in \mathbb{R}^{n-h+1}$. The vectors $\mathbf{c}$ are then aggregated over all m filters into a feature map matrix $\mathbf{C} \in \mathbb{R}^{m \times (n-h+1)}$. The filters are learned during the training phase of the neural network.

Max pooling. The output of the convolutional layer is passed through a non-linear activation function before entering a pooling layer. The latter aggregates vector elements by taking the maximum over a fixed set of non-overlapping intervals. The resulting pooled feature map matrix has the form: $\mathbf{C_{pooled}} \in \mathbb{R}^{m \times \frac{n-h+1}{s}}$, where s is the length of each interval. In the case of overlapping intervals with a stride value s_t, the pooled feature map matrix has the form $\mathbf{C_{pooled}} \in \mathbb{R}^{m \times \frac{n-h+1-s}{s_t}}$. Depending on whether the borders are included or not, the result of the fraction is rounded up or down respectively.

Hidden layer. A fully connected hidden layer computes the transformation $\alpha(\mathbf{W} * \mathbf{x} + \mathbf{b})$, where $\mathbf{W} \in \mathbb{R}^{m \times m}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ the bias, and α the rectified linear ($relu$) activation function (Nair and Hinton, 2010). The output vector of this layer, $\mathbf{x} \in \mathbb{R}^m$, corresponds to the sentence embeddings for each tweet.

Softmax. Finally, the outputs of the final pooling layer $\mathbf{x} \in \mathbb{R}^m$ are fully connected to a softmax regression layer, which returns the class $\hat{y} \in [1, K]$ with largest probability. i.e.,

$$\begin{aligned} \hat{y} &:= \arg\max_j P(y = j | \mathbf{x}, \mathbf{w}, \mathbf{a}) \\ &= \arg\max_j \frac{e^{\mathbf{x}^\intercal \mathbf{w}_j + a_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\intercal \mathbf{w}_k + a_j}}, \end{aligned} \qquad (2)$$

where $\mathbf{w}_j$ denotes the weights vector of class j and a_j the bias of class j.

Network parameters. Training the neural network consists in learning the set of parameters $\mathbf{\Theta} = \{\mathbf{E}, \mathbf{F}_1, \mathbf{b}_1, \mathbf{F}_2, \mathbf{b}_2, \mathbf{W}, \mathbf{a}\}$, where $\mathbf{E}$ is the sentence matrix, with each row containing the d-dimensional embedding vector for a specific word; $\mathbf{F}_i, \mathbf{b}_i (i = \{1, 2\})$ the filter weights and biases of the first and second convolutional layers; $\mathbf{W}$ the concatenation of the weights $\mathbf{w}_j$ for every output class in the soft-max layer; and $\mathbf{a}$ the bias of the soft-max layer.

Training. We pre-train the word embeddings on an unsupervised corpus of 200M tweets. During the distant-supervised phase, we use emoticons to infer the polarity of a set of additional 100M tweets (Read, 2005; Go et al., 2009). The resulting dataset contains positive 79M tweets and 21M

[1] http://www.nltk.org/api/nltk.tokenize.html

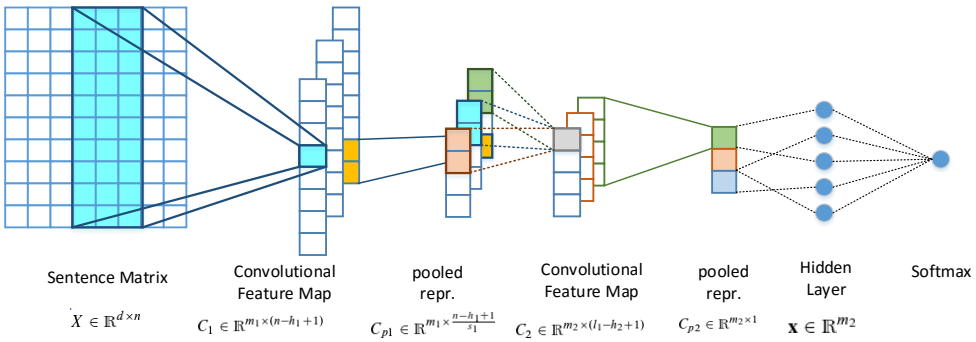

Figure 2: The architecture of the CNNs used in our approach.

negative tweets. The neural network is trained on these 100M tweets for one epoch. Then, the neural network is trained in a supervised phase on the labelled data provided by SemEval-2017. To avoid over-fitting we employed early-stopping, where we stop the training if the score on the validation set did not increase for 50 epochs. The word-embeddings $\mathbf{E} \in \mathbb{R}^{d \times n}$ are updated during both the distant and the supervised training phases, as back-propagation is applied through the entire network.

The network parameters are learned using *AdaDelta* (Zeiler, 2012), which adapts the learning rate for each dimension using only first order information. We used the hyper-parameters $\epsilon = 1\text{e}^{-6}$ and $\rho = 0.95$ as suggested by (Zeiler, 2012).

Computing Resources. The implementation of the system is written in *Python*. We used the *Keras* framework (Chollet, 2015) using the *Theano* (Theano Development Team, 2016) backend. *Theano* allows to accelerate the computations using a GPU. We used an *NVIDIA TITAN X* GPU to conduct our experiments. To generate the word embeddings we used the *Gensim* framework (Řehůřek and Sojka, 2010) which implements the word2vec model in *Python*. It took about 2 days to generate the word embeddings. The distant phase takes approximately 5 hours and the supervised phase up to 10 minutes.

3 Experiments & Results

3.1 Data

We evaluate our system on the datasets provided by SemEval-2017 for subtask B (see Table 3.1). The training-set is composed by *Train 2016*, *Dev 2016*, and *DevTest 2016*, whereas we use *Test 2016* as validation set. The training and the validation set have 100 topics each. Since our dataset of 200M tweets for the word embeddings consists primarily of tweets from 2013, some of the topics are not covered by the vocabulary. Thus, we download an additional 20M tweets where we use the topic-names as search key. This reduced the percentage of unknown words from 14% to 13%.

Dataset	Total	Posit.	Negat.	Topics
Train 2016	5355	2749	762	60
Dev 2016	1269	568	214	20
DevTest 2016	1779	883	276	20
Test: *Test 2016*	20632	7059	3231	100

Table 1: Overview of datasets and number of tweets (or sentences) provided in SemEval-2016. The data was divided into training, development and testing sets.

3.2 Setup

We use the following hyper parameters for the 2-layer CNN: In both layers we use 200 filters, a filter length of 6, and for the first layer we use a pooling length of 4 with striding 2.

In order to optimize our results, we varied the batch-sizes, we introduced class-weights to lessen the impact of the unbalanced training set, and we experimented with different schemes for shuffling the training set (see below). To determine the class-weight of class i we used the following formula: $\frac{n}{c*n_i}$, where n is the total number of samples in the training set, c is the number of classes, namely 2, and n_i is the number of samples that belong to class i.

3.3 Results

The following results are computed on *Test 2016*. We use the *macroaveraged recall* ρ as scoring mechanism for our experiments, which is more robust regarding class imbalances (Rosenthal et al.,

2017; Esuli and Sebastiani, 2015).
The results show that the usage of class weights increases the score by approximately 2 points from 0.8225 to 0.8403 points.
Figure 3 shows the score depending on the batch size used. The results show that higher batch sizes tend to give higher scores. This effect however subsides when batch sizes greater than 1200 are used.
Figure 4 shows the result when different schemes for shuffling the training set before training are applied. When we do not shuffle the training set, the *macroaveraged recall* score drops by 4-5 points. We compare two schemes for shuffling: (i) the *epoch based shuffling*, where the training set is shuffled entirely before each epoch, (ii) *batch based shuffling*, where the training set is split into batches and each batch is shuffled separately. The results in Figure 4 show that there is just a small difference of 1 point among the two strategies.

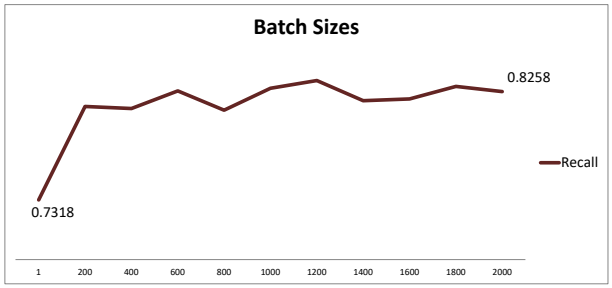

Figure 3: The architecture of the CNNs used in our approach.

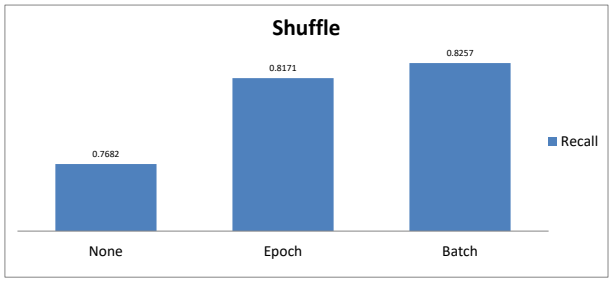

Figure 4: The architecture of the CNNs used in our approach.

Table 2 shows the results on the *Test 2017* set, which is the official test set of the competition. The system achieves a *macroaveraged recall* of 0.846, thus, ranking at 4^{th} place. The *BB twtr* system, ranked 1^{st}, outperforms our approach by 4 points, whereas the *DataStories* system and the *Tweester* system outperform our system by only 1 point.

System	ρ	F_1^{PN}	Acc
TopicThunder	0.846	0.847	0.854
BB twtr	0.882	0.890	0.897
DataStories	0.856	0.861	0.869
Tweester	0.854	0.854	0.863

Table 2: Official results on *Test 2017*. We compare our system to the systems that ranked 1^{st}, 2^{nd}, and 3^{rd}.

4 Conclusion

We described a deep learning approach to solve the problem of topic-specific sentiment analysis on twitter. The approach is based on a 2-layer Convolutional Neural Network which is trained using a large amount of data. Furthermore, we experimented with different parameters to fine tune our system. The system was evaluated at SemEval-2017 for the task of Topic-Based Message Polarity Classification, ranking at 4^{th} place.

References

Franois Chollet. 2015. keras. https://github.com/fchollet/keras.

Jan Deriu, Maurice Gonzenbach, Fatih Uzdilli, Aurelien Lucchi, Valeria De Luca, and Martin Jaggi. 2016. Swisscheese at semeval-2016 task 4: Sentiment classification using an ensemble of convolutional neural networks with distant supervision. *Proceedings of SemEval* pages 1124–1128.

Andrea Esuli and Fabrizio Sebastiani. 2015. Optimizing text quantifiers for multivariate loss functions. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9(4):27.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford* 1:12.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *ACL - Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, Maryland, USA, pages 655–665.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP 2014 - Empirical Methods in Natural Language Processing*. pages 1746–1751.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013. Exploiting Similarities among Languages for Machine Translation. *arXiv* .

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. pages 807–814.

Jonathon Read. 2005. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL student research workshop*. Association for Computational Linguistics, pages 43–48.

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

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. SemEval-2017 task 4: Sentiment analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation*. Association for Computational Linguistics, Vancouver, Canada, SemEval '17.

Aliaksei Severyn and Alessandro Moschitti. 2015. Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pages 959–962.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688. http://arxiv.org/abs/1605.02688.

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *arXiv* page 6. http://arxiv.org/abs/1212.5701.