Twitter Sentiment Analysis with Deep Convolutional Neural Networks

Aliaksei Severyn*
Google Inc.
aseveryn@gmail.com

Alessandro Moschitti†
Qatar Computing Research Institute
amoschitti@qf.org.qa

ABSTRACT

This paper describes our deep learning system for sentiment analysis of tweets. The main contribution of this work is a new model for initializing the parameter weights of the convolutional neural network, which is crucial to train an accurate model while avoiding the need to inject any additional features. Briefly, we use an unsupervised neural language model to train initial word embeddings that are further tuned by our deep learning model on a distant supervised corpus. At a final stage, the pre-trained parameters of the network are used to initialize the model. We train the latter on the supervised training data recently made available by the official system evaluation campaign on Twitter Sentiment Analysis organized by Semeval-2015. A comparison between the results of our approach and the systems participating in the challenge on the official test sets, suggests that our model could be ranked in the first two positions in both the phrase-level subtask A (among 11 teams) and on the message-level subtask B (among 40 teams). This is an important evidence on the practical value of our solution.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models—*Neural nets*

Keywords

Convolutional neural networks; twitter sentiment analysis

1. INTRODUCTION

In this work we describe our deep convolutional neural network for sentiment analysis of tweets. Its architecture is most similar to the deep learning systems presented in [2, 3] that have recently established new state-of-the-art results on various NLP sentence classification tasks also including sentiment analysis. Convolutional neural networks have been also successfully applied in various IR applications, e.g., [8, 9]. While already demonstrating excellent results, training a convolutional neural network that would beat hand-engineered approaches that also rely on multiple manual and automatically constructed lexicons, e.g. [5, 11], requires careful attention. This becomes an even harder problem especially in cases when the amount of labelled data is relatively small, e.g., thousands of examples.

It turns out that providing the network with good initialisation parameters can have a significant impact on the accuracy of the trained model. To address this issue, we propose a three-step process to train our deep learning model for sentiment classification. Our approach can be summarized as follows: (i) word embeddings are initialized using a neural language model [4, 7], which is trained on a large unsupervised collection of tweets; (ii) we use a convolutional neural network to further refine the embeddings on a large distant supervised corpus [1]; (iii) the word embeddings and other parameters of the network obtained at the previous stage are used to initialize the network with the same architecture, which is then trained on a supervised corpus from Semeval-2015.

We apply our deep learning model on two subtasks of Semeval-2015 Twitter Sentiment Analysis (Task 10) challenge: phrase-level (subtask A) and message-level (subtask B). Our system achieves high results on the official tests sets of the phrase-level and on the message-level subtasks. In addition to the above test sets, we also used the so-called progress test set, which consists of five test sets, where our system again outperforms most of the systems participated in the challenge. In particular, if we ranked all systems (including ours) according to their accuracy on each of the six test sets and compute their average ranks, our model would be ranked first in both subtasks, A and B.

2. OUR DEEP LEARNING MODEL FOR SENTIMENT CLASSIFICATION

The architecture of our convolutional neural network for sentiment classification is shown on Fig. 1. It is mainly inspired by the architectures used in [2, 3] for performing various sentence classification tasks. Given that our training process (described in Sec. 3.3) requires to run the network on a rather large corpus, our design choices are mainly driven by the computational efficiency of our network. Hence, different from [2], which presents an architecture with several layers of convolutional feature maps, we adopt a single level architecture. Nevertheless, single-layer architectures have been shown in [3] to perform equally well.

Our network is composed of a single convolutional layer followed by a non-linearity, *max* pooling and a soft-max classification layer.

In the following, we give a brief explanation of the main components of our network: sentence matrix, activations, convolutional, pooling and softmax layers. We also describe how to adapt the network for predicting sentiment of phrases inside the tweets.

*This work was carried out at University of Trento.
†Professor at University of Trento, DISI.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.
© 2015 ACM. ISBN 978-1-4503-3621-5/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767830.

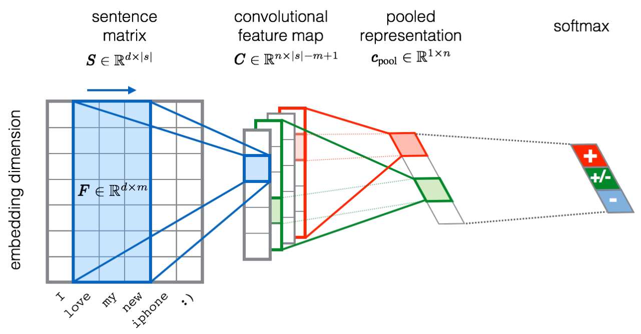
sentence matrix
$S \in \mathbb{R}^{d\times|s|}$
convolutional feature map
$C \in \mathbb{R}^{n\times|s|-m+1}$
pooled representation
$c_{\text{pool}} \in \mathbb{R}^{1\times n}$
softmax
embedding dimension
$F \in \mathbb{R}^{d\times m}$
I
love
my
new
iphone
:)
+
+/-
-

Figure 1: The architecture of our deep learning model for sentiment classification.

2.1 Sentence matrix

The input to our model are tweets each treated as a sequence of words: $[w_i, .., w_{|s|}]$, where each word is drawn from a vocabulary V. Words are represented by distributional vectors $\mathbf{w} \in \mathbb{R}^{1\times d}$ looked up in a word embeddings matrix $\mathbf{W} \in \mathbb{R}^{d\times|V|}$. This matrix is formed by simply concatenating embeddings of all words in V. For convenience and ease of lookup operations in $\mathbf{W}$, words are mapped into indices $1, \ldots, |V|$.

For each input tweet $\mathbf{s}$, we build a sentence matrix $\mathbf{S} \in \mathbb{R}^{d\times|s|}$, where each column i represents a word embedding $\mathbf{w}_i$ at the corresponding position i in a sentence (see Fig. 1). To learn to capture and compose features of individual words in a given sentence from low-level word embeddings into higher level semantic concepts, the neural network applies a series of transformations to the input sentence matrix $\mathbf{S}$ using convolution, non-linearity and pooling operations, which we describe next.

2.2 Convolutional feature maps

The aim of the convolutional layer is to extract patterns, i.e., discriminative word sequences found within the input tweets that are common throughout the training instances.

More formally, the convolution operation, $*$, between an input matrix $\mathbf{s} \in \mathbb{R}^{d\times|s|}$ and a filter $\mathbf{F} \in \mathbb{R}^{d\times m}$ of width m results in a vector $\mathbf{c} \in \mathbb{R}^{|s|+m-1}$, where each component is computed as follows:

$$\mathbf{c}_i = (\mathbf{S} * \mathbf{F})_i = \sum_{k,j} (\mathbf{S}_{[:,i-m+1:i]} \otimes \mathbf{F})_{kj}, \quad (1)$$

where $\otimes$ is the element-wise multiplication and $\mathbf{S}_{[:,i-m+1:i]}$ is a matrix slice of size m along the columns. Note that the convolution filter is of the same dimensionality d as the input sentence matrix. As shown in Fig. 1, it slides along the column dimension of $\mathbf{S}$ producing a vector $\mathbf{c} \in \mathbb{R}^{1\times(|s|-m+1)}$ in output. Each component c_i is the result of computing an element-wise product between a column slice of $\mathbf{S}$ and a filter matrix $\mathbf{F}$, which is then summed to a single value.

So far we have described a way to compute a convolution between the input sentence matrix and a single filter. To form a richer representation of the data, deep learning models apply a set of filters that work in parallel generating multiple feature maps (also shown on Fig. 1). A set of filters form a filter bank, $\mathbf{F} \in \mathbb{R}^{n\times d\times m}$, sequentially convolved with the sentence matrix $\mathbf{S}$ and producing a feature map matrix $\mathbf{C} \in \mathbb{R}^{n\times(|s|-m+1)}$.

In practice, we also need to add a bias vector $\mathbf{b} \in \mathbb{R}^n$ to the result of a convolution – a single b_i value for each feature map $\mathbf{c}_i$. This allows the network to learn an appropriate threshold.

2.3 Activation units

To enable the learning of non-linear decision boundaries, each convolutional layer is typically followed by a non-linear activation function, $\alpha()$, applied element-wise. Among the most common choices of activation functions are: sigmoid (or logistic), hyperbolic tangent *tanh*, and a rectified linear (ReLU) function defined as simply $max(0, \mathbf{x})$ to ensure that feature maps are always positive.

We use ReLU in our model since, as shown in [6], it speeds up the training and sometimes produces more accurate results.

2.4 Pooling

The output from the convolutional layer (passed through the activation function) is then passed to the pooling layer, whose goal is to aggregate the information and reduce the representation. The result of the pooling operation is:

$$\mathbf{c}_{\text{pooled}} = \begin{bmatrix} \text{pool}(\alpha(\mathbf{c}_1 + b_1 * \mathbf{e})) \\ \cdots \\ \text{pool}(\alpha(\mathbf{c}_n + b_n * \mathbf{e})) \end{bmatrix},$$

where $\mathbf{c}_i$ is the ith convolutional feature map with added bias (the bias is added to each element of $\mathbf{c}_i$ and $\mathbf{e}$ is a unit vector of the same size as $\mathbf{c}_i$) and passed through the activation function $\alpha()$.

The most popular choices for pooling operation are: max and average pooling. Recently, *max* pooling has been generalized to k-max pooling [2], where instead of a single max value, k values are extracted in their original order. We use *max* pooling in our model, which simply returns the maximum value. It operates on columns of the feature map matrix $\mathbf{C}$ returning the largest value: $\text{pool}(\mathbf{c}_i) : \mathbb{R}^{1\times(|s|+m-1)} \to \mathbb{R}$ (also shown schematically in Fig. 1).

The convolutional layer utilizing the activation function and the pooling layer acts as a non-linear feature extractor. Given that multiple feature maps are used in parallel to process the input, deep learning networks are able to build rich feature representations of the input.

2.5 Softmax

The output of the penultimate convolutional and pooling layers $\mathbf{x}$ is passed to a fully connected softmax layer. It computes the probability distribution over the labels:

$$\begin{aligned} P(y = j|\mathbf{x}, \mathbf{s}, \mathbf{b}) &= \text{softmax}_j(\mathbf{x}^T\mathbf{w} + \mathbf{b}) \\ &= \frac{e^{\mathbf{x}^T\mathbf{w}_j + b_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T\mathbf{w}_k + b_k}}, \end{aligned}$$

where $\mathbf{w}_k$ and b_k are the weight vector and bias of the k-th class.

2.6 Phrase-level sentiment analysis

To perform phrase-level sentiment analysis, we feed the network with an additional input sequence indicating the location of the target phrase in a tweet. The elements are encoded using only two word types: the tokens spanning the phrase to be predicted are encoded with 1s and all the others with 0s. Each word type is associated with its own embedding. So, when tackling the phrase-level sentiment classification, we form a sentence matrix $\mathbf{S}$ as follows: for each token in a tweet, we have to look up its corresponding word embedding in the word matrix $\mathbf{W}$, and the embedding for one of the two word types. Hence, the input sentence matrix is augmented with an additional set of rows from the word type embeddings. Other than that, the architecture of our network remains unchanged.

3. OUR APPROACH TO TRAIN THE NETWORK

Convolutional neural networks can be tricky to train as are often severely subject to overfitting when trained on small datasets. In the following, we describe our approach to train our deep learning model.

3.1 Network Parameters and Training

We use stochastic gradient descent (SGD) to train the network and use backpropogation algorithm to compute the gradients. We opt for the *Adadelta* [12] update rule to automatically tune the learning rate.

3.2 Regularization

While neural networks have a large capacity to learn complex decision functions they tend to easily overfit especially on small and medium sized datasets. To mitigate the overfitting issue, we augment the cost function with l_2-norm regularization terms for the parameters of the network.

We also use another popular and effective technique to improve regularization of the neural networks — dropout [10]. Dropout prevents feature co-adaptation by setting to zero (dropping out) a portion of hidden units during the forward phase when computing the activations at the softmax output layer.

3.3 Initializing the model parameters

Convolutional neural networks are trained with non-convex function optimization algorithms, which typically lead to locally optima. Hence, starting the optimization from *a good point* can be crucial to train an accurate model. We propose the following 3-step process to initialize the parameter weights of the network:

1. Given that the largest parameter of the network is the word matrix $\mathbf{W}$, it is crucial to feed the network with the high quality embeddings. We use a popular `word2vec` neural language model [4] to learn the word embeddings on an unsupervised tweet corpus. For this purpose, we collect 50M tweets over a two-month period. We perform a minimal preprocessing, tokenizing the tweets, normalizing the URLs and author ids. To train the embeddings, we use a skipgram model with window size 5 and filter words with frequency less than 5.

2. When dealing with small amounts of labelled data, starting from pre-trained word embeddings is a large step towards successfully training an accurate deep learning system. However, while the word embeddings obtained at the previous step should already capture important syntactic and semantic aspects of the words they represent, they are completely clueless about their sentiment behaviour. Hence, we use a distant supervision approach [1] using our convolutional neural network to further refine the embeddings. More specifically, (i) we collected 10M tweets treating tweets containing positive emoticions, which can be used as distantly supervised labels; and (ii) we used such noisy label to train our embeddings. This step required a few days.

3. Finally, we take the the parameters θ of the network obtained at the previous step and use it to initialize the network which is trained on the supervised training corpus from Semeval-2015.

Table 1: Semeval-2015 data.

Dataset	Subtask A	Subtask B
Twitter'13-train	5,895	9,728
Twitter'13-dev	648	1,654
Twitter'13-test	2,734	3,813
LiveJournal'14	660	1,142
SMS'13	1,071	2,093
Twitter'14	1,807	1,853
Sarcasm'14	82	86
Twitter'15	3,092	2,390
# Teams	11	40

4. EXPERIMENTS AND EVALUATION

4.1 Data and setup

We test our model on two subtasks from Semeval-2015 Task 10: phrase-level (subtask A) and message-level (subtask B)[1]. The datasets used in Semeval-2015 are summarized in Table 1. We use train and dev from Twitter'13 for training and Twitter'13-test as a validation set. The other datasets are used for testing, whereas Twitter'15 is used to establish the official ranking of the systems. For evaluation we use the official scorers from Semeval 2015, which compute the average between F-measures for the positive and negative classes.

To pre-train the weights of our network, we use a large unsupervised corpus containing 50M tweets for training the word embeddings and a 10M tweet corpus for distant supervision. The latter corpus was built similarly to [1], where tweets with positive emoticons, like '`:)`', are assumed to be positive, and tweets with negative emoticons, like '`:(`', are labeled as negative. The dataset contains equal number of positive and negative tweets.

The parameters of our model were (chosen on the validation set) as follows: the width m of the convolution filters is set to 5 and the number of convolutional feature maps is 300. We use ReLU activation function and a simple max-pooling. The dimensionality of the word embeddings d is set to 100. For the phrase-level subtask the size of the word type embeddings, which encode tokens that span the target phrase or not, is set to 10.

4.2 Pre-training the network

To train our deep learning model, we follow our 3-step process as described in Sec. 3.3. We report the results for training the network on the official supervised dataset from Semeval'15 using parameters that were initialized: (i) completely at random (`Random`); (ii) using word embeddings from the neural language model trained on a large unsupervised dataset (`Unsup`) with the `word2vec` tool and (iii) initializing all the parameters of our model with the parameters of the network that uses the word embeddings from the previous step and are further tuned on a distant supervised dataset (`Distant`).

Table 2 summarizes the performance of our model on five test sets using three parameter initialization schemas.

We note that: first, training the network with all parameters initialized completely at random results in a rather mediocre accuracy. This is due to a small size of the training set.

[1]the test datasets of SemEval'15 subsume the test sets from previous editions of Semeval, i.e., Semeval'13 and Semeval'14, so our results directly apply to those of the previous years.

Table 2: Testing the model on the progress test sets from Semeval-2015 with different parameter initializion schemes: `Random` (random word embeddings); `Unsup` (word2vec embeddings); `Distant` (all parameters from a network trained on a distant supervised dataset).

Dataset	Random	Unsup	Distant
LiveJournal'14	63.58	73.09	72.48
SMS'13	58.41	65.21	68.37
Twitter'13	64.51	72.35	72.79
Twitter'14	63.69	71.07	73.60
Sarcasm'14	46.10	52.56	55.44

Secondly, using embeddings pre-trained by a neural language model considerably boosts the performance.

Finally, using a large distant supervised corpus to further tune the word embeddings to also capturing the sentiment aspect of the words results in a further improvement across all test sets (except for a small drop on LiveJournal'14).

4.3 Official rankings

The comparison of our system performance with the official system rankings from Semeval'15 for both subtasks A and B are summarized in Table 3. As we can see our system performs particularly well on subtask A, it would be ranked 1st on the official Twitter'15 test set, while also showing excellent accuracy on all other test sets.

On subtask B our system would rank 2nd also showing high results on the other test sets (except for the LiveJournal'14). In fact, no single system at Semeval-2015 performed equally well across all test sets. For example, a system that ranked 1st on the official Twitter'15 dataset performed much worse on the progress test sets ranking, i.e., $\{14, 14, 11, 7, 12\}$ on {`LiveJournal'14`, `SMS'13`, `Twitter'13`, `Twitter'14`, and `Sarcasm'14`}, respectively. This results on an AveRank of 9.8, which is only at the 6th position if systems were ranked according to this average rank metric. In contrast, our system shows high robustness as its results, across all tests, would provide the AveRank of 4.3, which is the top-score according to this metric among all 40 submissions.

5. CONCLUSIONS

We described our deep learning approach to sentiment analysis of tweets for predicting polarities at both message and phrase levels. We give a detailed description of our 3-step process to train the parameters of the network that is the key to our success. The resulting model sets a new state-of-the-art on the phrase-level and is 2nd on the message-level subtask. Considering the average rank across all test sets our system is 1st on both subtasks.

Our network initialization process includes the use of distant supervised data (noisy labels are inferred using emoticons found in the tweets) to further refine the weights of the network passed from the completely unsupervised neural language model. Thus, our solution successfully combines together two traditionally important aspects of IR: unsupervised learning of text representations (word embeddings from neural language model) and learning on weakly supervised data. In the future we plan to apply deep learning approach to other IR applications, e.g., learning to rank for Microblog retrieval and answer reranking for Question Answering.

Acknowledgments. This work has been supported by the EC project CogNet, 671625 (H2020-ICT-2014-2). The first author was supported by the Google Europe Doctoral Fellowship Award 2013.

Table 3: Results on Semeval-2015 for phrase and tweet-level subtasks. Rank shows the absolute position of our system on each test set. AveRank is the averaged rank across all test sets.

Dataset	Score	Rank
Phrase-level subtask A		
LJournal'14	84.46	2
SMS'13	88.60	2
Twitter'13	90.10	1
Twitter'14	87.12	1
Sarcasm'14	73.65	5
Twitter'15	**84.79**	**1**
AveRank	2.0	1
Message-level subtask B		
LJournal'14	72.48	12
SMS'13	68.37	2
Twitter'13	72.79	3
Twitter'14	73.60	2
Sarcasm'14	55.44	5
Twitter'15	**64.59**	**2**
AveRank	4.3	1

REFERENCES

[1] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. In *CS224N Project Report, Stanford*, 2009.

[2] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

[3] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[5] S. M. Mohammad, S. Kiritchenko, and X. Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. In *Semeval*, 2013.

[6] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[7] J. W. Ronan Collobert. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, 2008.

[8] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. *CIKM*, 2014.

[9] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW*, 2014.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[11] S. M. M. Xiaodan Zhu, Svetlana Kiritchenko. Nrc-canada-2014: Recent improvements in sentiment analysis of tweets, and the Voted Perceptron. In *SemEval*, 2014.

[12] M. D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, 2012.