

See discussions, stats, and author profiles for this publication at: <http://www.researchgate.net/publication/275886528>

UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification

CONFERENCE PAPER · JANUARY 2015

CITATIONS

2

DOWNLOADS

18

VIEWS

28

2 AUTHORS:



[Aliaksei Severyn](#)

Università degli Studi di Trento

17 PUBLICATIONS 47 CITATIONS

[SEE PROFILE](#)



[Alessandro Moschitti](#)

Università degli Studi di Trento

177 PUBLICATIONS 2,100 CITATIONS

[SEE PROFILE](#)

UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification

Aliaksei Severyn

DISI, University of Trento
38123 Povo (TN), Italy
severyn@disi.unitn.it

Alessandro Moschitti

Qatar Computing Research Institute
5825 Doha, Qatar
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 process to initialize 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 initialize 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 which is then trained on the supervised training data from Semeval-2015. According to results on the official test sets, our model ranks 1st in the phrase-level subtask A (among 11 teams) and 2nd on the message-level subtask B (among 40 teams). Interestingly, computing an average rank over all six test sets (official and five progress test sets) puts our system 1st in both subtasks A and B.

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 (Kalchbrenner et al., 2014; Kim, 2014) that have recently established new state-of-the-art results on various NLP sentence classification tasks also including sentiment analysis. 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. (Mohammad et al., 2013; Xiaodan Zhu, 2014; Severyn and Moschitti, 2015), 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.

Turns out, providing the network with good initialisation parameters makes all the difference in training an accurate model. We propose a three-step process we follow to train our deep learning model for sentiment classification. It can be summarized as follows: (i) word embeddings are initialized using a neural language model (Ronan Collobert, 2008; Mikolov et al., 2013) which is trained on a large unsupervised collection of tweets; (ii) we use our convolutional neural network to further refine the embeddings on a large distant supervised corpus (Go et al., 2009); (iii) the word embeddings and other parameters of the network obtained at the previous stage are used to initialize the network that 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 (Rosenthal et al., 2015): phrase-level (subtask A) and message-level (subtask B). Our system ranks 1st on the official test set of the phrase-level and 2nd on the message-level subtask. In addition to the test set used to establish the final ranking in Semeval-2015, all systems were also evaluated on the progress test set which consists of five test sets, where our system also shows strong results. In particular, we rank all systems according to their performance on each test set and compute their average ranks. Interestingly, our model appears to be the most robust across all six test sets ranking 1st

according to the average rank in both subtasks A and B.

In the following, we describe the architecture of our convolutional neural network and the parameter initialization process we follow to train it.

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 (Kalchbrenner et al., 2014; Kim, 2014) 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 (Kalchbrenner et al., 2014) that presents an architecture with several layers of convolutional feature maps, we adopt a single level architecture, which has been shown in (Kim, 2014) 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 architecture: sentence matrix, activations, convolutional, pooling and softmax layers. We also describe how to adapt the network for predicting sentiment of phrases inside the tweets.

2.1 Sentence matrix

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

For each input tweet 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):

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \dots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

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 c_i . This allows the network to learn an appropriate threshold.

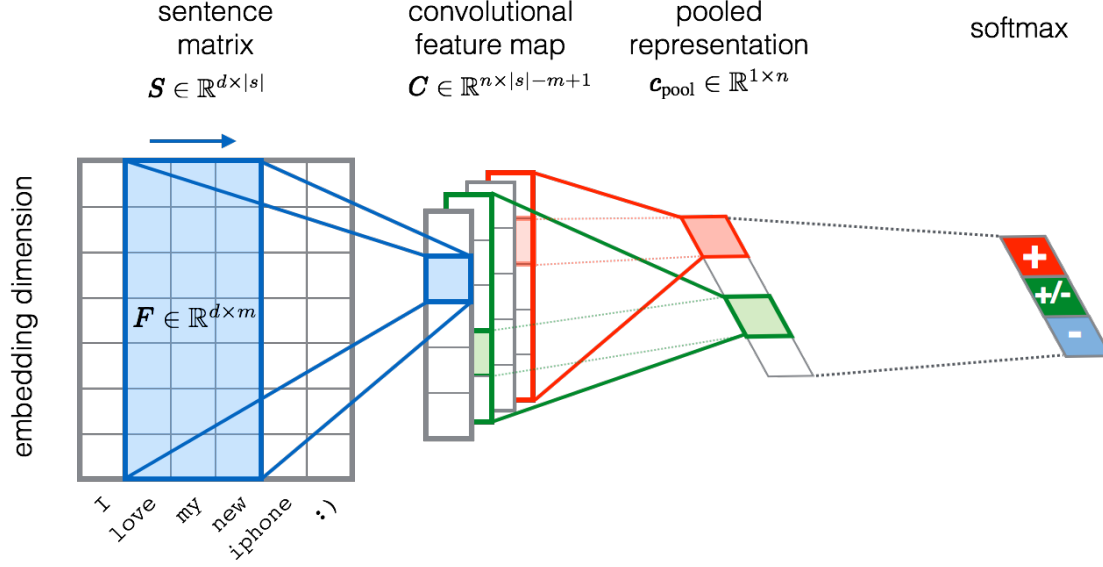


Figure 1: The architecture of our deep learning model for sentiment classification.

2.3 Activation units

To allow the network learn 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 (Nair and Hinton, 2010), 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) are then passed to the pooling layer, whose goal is to aggregate the information and reduce the representation. The result of the pooling operation is:

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

where \mathbf{c}_i is the i th 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 $\alpha()$.

Among the most popular choices for pooling operation are: max and average pooling. Recently, *max* pooling has been generalized to k -max pooling (Kalchbrenner et al., 2014), 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: $\text{pool}(\mathbf{c}_i) : \mathbb{R}^{1 \times (|s| + m - 1)} \rightarrow \mathbb{R}$ (also shown schematically in Fig. 1).

Convolutional layer passed through the activation function together with 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: tokens spanning the phrase to be predicted are encoded with 1s and all the other 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.

This ends the description of our convolutional neural network for sentiment classification of tweets.

3 Our approach to train the network

Convolutional neural networks can be tricky to train often severely overfitting 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 backpropagation algorithm to compute the gradients. We opt for the *Adadelta* (Zeiler, 2012) update rule to automatically tune the learning rate.

The following parameters are optimized by our network:

$$\theta = \{\mathbf{W}; \mathbf{F}; \mathbf{b}; \mathbf{w}_s; \mathbf{b}_s\},$$

namely the word embeddings matrix \mathbf{W} , filter weights and biases of the convolutional layer, the weight and bias of the softmax layers.

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 adopt another popular and effective technique to improve regularization of the NNs — dropout (Srivastava et al., 2014). 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. As suggested in (Goodfellow et al., 2013) dropout acts as an approximate model averaging.

3.3 Initializing the model parameters

Convolutional neural networks live in the world of non-convex function optimization leading to locally optimal solutions. 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 (Mikolov et al., 2013) to learn the word embeddings on an unsupervised tweet corpus. For this purpose, we collect 50M tweets over the two-month period. We perform 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 filtering 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 (Go et al., 2009) using our convolutional neural network to further refine the embeddings.
3. Finally, we take the the parameters θ of the network obtained at the previous step and use it to

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

initialize the network which is trained on a supervised training corpus from Semeval-2015.

4 Experiments and evaluation

Data and setup. We test our model on two subtasks from Semeval-2015 Task 10: phrase-level (subtask A) and message-level (subtask B). The datasets use 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.

Additionally, 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 (Go et al., 2009), 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 L2 regularization term is set to $1e - 4$, dropout is applied to the penultimate level with $p = 0.5$. 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.

Pre-training the network. To train our deep learn-

Table 2: Testing the model on the progress test sets from Semeval-2015 with different parameter initialization 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

ing 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 which 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. First, we observe that training the network with all parameters initialized completely at random results in a rather mediocre performance. This is due to a small size of the training set. 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 capture the sentiment aspect of the words they represent results in a further improvement across all test sets (except for a small drop on LiveJournal’14).

Official rankings. The results from the official rankings for both subtasks A and B are summarized in Table 3. As we can see our system performs particularly well on subtask A ranking 1st on the official Twitter’15 set, while also showing excellent performance on all other test sets.

On subtask B our system ranks 2nd also showing high rankings on the other test sets (apart from the LiveJournal’14). In fact, no single system at Semeval-2015 performed equally well across all test

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

sets. For example, a system that ranked 1st on the official Twitter’15 dataset performs much worse on the progress test sets ranking {14, 14, 11, 7, 12} on {LiveJournal’14, SMS’13, Twitter’13, Twitter’14, and Sarcasm’14} correspondingly. It has an AveRank of 9.8, which is only 6th best result if systems were ranked according to this metric. In contrast, our system shows robust results across all tests having the best AveRank of 4.3 among all 40 systems.

5 Conclusions

We described our deep learning approach to Twitter sentiment analysis on both message and phrase levels. We gave 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 demonstrates state-of-the-art performance on both the phrase-level and message-level subtasks. Considering the average rank across all test sets (including progress test sets) our system is 1st on both subtasks.

References

- Alex Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. In *CS224N Project Report, Stanford*.
- Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. 2013. Max-out networks. In *ICML*, pages 1319–1327.
- Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.
- Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, Doha, Qatar.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems* 26, pages 3111–3119.
- Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. In *7th International Workshop on Semantic Evaluation (Semeval’13)*.
- Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814.
- Jason Weston Ronan Collobert. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*.
- Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M Mohammad, Alan Ritter, and Veselin Stoyanov. 2015. Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval ’2015*, Denver, Colorado.
- Aliaksei Severyn and Alessandro Moschitti. 2015. On the automatic learning of sentiment lexicons. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2015)*.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.
- Saif M. Mohammad Xiaodan Zhu, Svetlana Kiritchenko. 2014. Nrc-canada-2014: Recent improvements in sentiment analysis of tweets, and the Voted Perceptron. In *Eighth International Workshop on Semantic Evaluation Exercises (SemEval-2014)*.
- Matthew D. Zeiler. 2012. Adadelta: An adaptive learning rate method. *CoRR*.