A Linguistically Informed Convolutional Neural Network

Sebastian Ebert and **Ngoc Thang Vu** and **Hinrich Schütze**
Center for Information and Language Processing
University of Munich, Germany
{ebert|thangvu}@cis.lmu.de, inquiries@cislmu.org

Abstract

Sentiment lexicons and other linguistic knowledge proved to be beneficial in polarity classification. This paper introduces a linguistically informed Convolutional Neural Network (lingCNN), which incorporates this valuable kind of information into the model. We present two intuitive and simple methods: The first one integrates word-level features, the second sentence-level features. By combining both types of features our model achieves results that are comparable to state-of-the-art systems.

1 Introduction

This paper explores the use of Convolutional Neural Networks (CNN) for sentiment analysis. CNNs reach state-of-the-art results in several polarity classification tasks (Mohammad et al., 2013; Tang et al., 2014a; Kim, 2014; Severyn and Moschitti, 2015; Kalchbrenner et al., 2014). Reasons are their ability to deal with arbitrary input sentence lengths and to preserve word order. Moreover, they learn to find the most important polarity indicators and ignore the rest of the sentence. That is beneficial, since most of the words in a text do not convey sentiment information. Finally, CNNs can make use of powerful pretrained word representations (e.g., Mikolov et al. (2013)).

However, training such a model requires a large amount of labeled training data. One approach to address this issue is to enlarge training data in a semi-supervised fashion (Severyn and Moschitti, 2015). Instead, we propose to make use of already available linguistically motivated resources. Especially sentiment lexicons are important cues for polarity classification (cf. Mohammad et al. (2013)).

We introduce two intuitive and simple methods of incorporating linguistic features into a CNN.

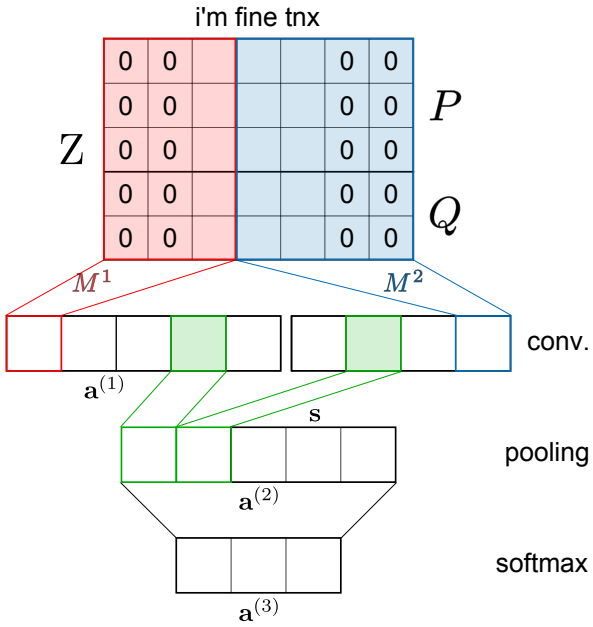

Figure 1: LingCNN architecture

The resulting architecture is called *linguistically informed CNN* (*lingCNN*). The first method is to add features to every word in a sentence. That enables the model to learn interactions between words and between individual word embeddings and linguistic features. The second method is to add feature vectors that are computed based on the entire sentence. The results show that word-level features can improve the classification and are more beneficial than sentence-level features. However, the combination of both methods reaches the best performance and yields results that are comparable to state-of-the-art on the SemEval Twitter polarity data set.

2 LingCNN Architecture

Figure 1 depicts the lingCNN architecture. We use the following terminology. $LT \in \mathbb{R}^{d \times |V|}$ denotes a lookup table that assigns each word in the vocabulary V a d-dimensional vector. Given a sequence of n tokens t_1 to t_n the model concatenates all n

word representations to the input of the lingCNN:

$$Z = \begin{bmatrix} | & | & | \\ LT_{\cdot,t_1} & \cdots & LT_{\cdot,t_n} \\ | & | & | \end{bmatrix}$$

The lingCNN consists of three types of layers (in the following indicated by a superscript index): a convolution layer, a max pooling layer, and a fully connected softmax layer.

2D Convolution Using a convolution matrix $M \in \mathbb{R}^{d \times m}$ (also called filter matrix) the lingCNN performs a 2d convolution:

$$\mathbf{a}_o^{(1)} = \sum_{i=1}^{d} \sum_{j=1}^{m} M_{i,j} Z_{i,o+j}$$

, where $\mathbf{a}^{(1)}$ is the layer's activation and $o \in [0, n-m]$ is the current position of the convolution.

The width of the filter m specifies how many words the filter spans ($m \in \{3, 4\}$ in Figure 1). The model uses multiple filter sizes at this level and several filters per filter size. Furthermore, we make use of *wide convolution* (Kalchbrenner et al., 2014), which pads the input Z with $m-1$ zero columns at the left and right side (i.e., the sentence length becomes $n + 2 * (m-1)$). This makes sure that every column of the filter reaches every column of the input. The model uses 2d convolution, because a filter that span all d dimensions in height has the advantage that it can find features that interact with multiple dimensions.

Max Pooling To keep only the most salient features, the lingCNN selects the largest value of each convolution output. This way we hope to find the most important polarity indicators, independently of their position. To the remaining values we add a bias $b^{(2)}$ and apply a rectified linear unit nonlinearity: $\mathbf{a}^{(2)} = \max(0, \mathbf{a}^{(1)} + b^{(2)})$ (Nair and Hinton, 2010).

Softmax Next, the output values of the pooling layer are concatenated with a sentence-level feature vector $\mathbf{s}$: $\mathbf{a}^{(2)'} = [\mathbf{a}^{(2)}\ \mathbf{s}]$, which is the input to a fully connected layer: $\mathbf{z} = W\mathbf{a}^{(2)'} + b^{(3)}$. This layer converts its input into a probability distribution over the sentiment labels using the softmax function: $\mathbf{a}_i^{(3)} = \frac{\exp(\mathbf{z}_i)}{\sum_j \exp(\mathbf{z}_j)}$.

3 Word-level Features

To incorporate linguistic features at word-level into the learning process we create the lookup table by concatenating two matrices: $LT = \begin{bmatrix} P \\ Q \end{bmatrix}$. $P \in \mathbb{R}^{d_P \times |V|}$ denotes a matrix of low-dimensional word representations, so called *word embeddings*. d_P, the size of the embeddings, is usually set to 50 – 300, depending on the task. We train skip-gram word embeddings (Mikolov et al., 2013) with the word2vec toolkit[1] on a large amount of Twitter text data. Previous work showed that pretrained word embeddings are helpful in various tasks (e.g., Kim (2014)). We first downloaded about 60 million tweets from the unlabeled Twitter Events data set (McMinn et al., 2013). The vocabulary is built out of all the words of the SemEval training data and the 50k most frequent words of the Twitter Events data set. Additionally, an *unknown* word is added to the vocabulary to learn a word embedding for out-of-vocabulary words. Finally, a skip-gram model with 60-dimensional vectors is trained on the unlabeled data and used to initialize the word embeddings matrix P. The matrix P is further fine-tuned during model training.

In addition to P, we introduce another matrix $Q \in \mathbb{R}^{d_Q \times |V|}$, which contains external word features. In this case d_Q is the number of features for a word. The features in Q are precomputed and not embedded into any embeddings space, i.e., Q is fixed during training. We use the following feature types:

Binary Sentiment Indicators Binary features that indicate a word's prior polarity. We create two such features per word per lexicon. The first feature indicates positive and the second negative polarity of that word in the lexicon. The lexicons for this feature type are MPQA (Wilson et al., 2005), Opinion lexicon (Hu and Liu, 2004), and NRCC Emotion lexicon (Mohammad and Turney, 2013).

Sentiment Scores The Sentiment140 lexicon and the Hashtag lexicon (Mohammad et al., 2013) both provide a score for each word instead of just a label. We directly incorporate these scores into the feature matrix. Both lexicons also contain scores for bigrams and skip ngrams. In such a case all words of the ngram receive the same ngram score.

[1] https://code.google.com/p/word2vec/

Binary Negation Following Christopher Potts,[2] we mark each word between a negation word and the next punctuation as negated.

In total each word receives 13 additional features ($3 * 2$ binary, $2 * 3$ scores, 1 negation). Since lingCNN performs a 2d convolution, it allows the detection of features that interact with word embeddings and linguistic features.

4 Sentence-level Features

An alternative to adding word-level features into the training process is to add sentence-level features. These features are concatenated with the pooling layer's output to serve as additional input for the softmax layer as described above. We use the following feature types:

Counts We use the following counts: number of terms that are all upper case; number of elongated words such as 'coooool'; number of emoticons;[3] number of contiguous sequences of punctuation; number of negated words.

Sentiment Scores The computed lexicon features are the number of sentiment words in a sentence, the sum of sentiment scores of these words as provided by the lexicons, the maximum sentiment score, and the sentiment score of the last word. These four numbers are calculated for all 5 previously mentioned sentiment lexicons. Moreover, they are computed separately for the entire tweet, for each POS tag, for all hashtag tokens in the tweet, and for all capitalized words in the tweet (Mohammad et al., 2013).

5 Experiments and Results

5.1 Data

To evaluate the lingCNN, we use the SemEval 2015 data set (Rosenthal et al., 2015). We train the model on the SemEval 2013 training and development set and use the SemEval 2013 test set as development set (Nakov et al., 2013; Rosenthal et al., 2015). The final evaluation is done on the SemEval 2015 test set. Table 1 lists all data set sizes in detail.

To test the generality of our findings we additionally report results on the manually labeled test set of the Sentiment140 corpus (Sent140) (Go et al., 2009). It contains about 500 tweets (cf. Table 1), which were collected by searching Twitter for specific categories, e.g., movies.

	total	pos	neg	neu
training set	9845	3636	1535	4674
development set	3813	1572	601	1640
SemEval test set	2390	1038	365	987
Sent140 test set	498	182	177	139

Table 1: Data set sizes.

The examples in all data sets are labeled with one of the three classes: *positive*, *negative*, or *neutral*.[4] Similar to the SemEval shared task we report the macro F_1 score of the positive and negative classes, i.e., $F_{1,macro} = (F_{1,positive} + F_{1,negative})/2$.

Prepocessing The SemEval and Sentiment140 data as well as the unlabeled Twitter Events data set, which is used for pretraining word embeddings, are preprocessed in the following way: Tweets are first tokenized with the CMU tokenizer (Owoputi et al., 2013). Afterwards, all user mentions are replaced by '<user>' and all urls by '<web>'. We keep hashtags, because they often contain valuable information such as topics or even sentiment.

Punctuation sequences like '!?!?' can act as exaggeration or other polarity modifiers. However, the sheer amount of possible sequences increases the out-of-vocabulary rate dramatically. Therefore, all sequences of punctuations are replaced by a list of distinct punctuations in this sequence (e.g., '!?!?' is replaced by '[!?]').

5.2 Model Settings

Baseline Systems We use the SemEval 2013 and SemEval 2014 winning system (Mohammad et al., 2013) as baseline. This system uses a Support Vector Machine (SVM) for classification. According to their analysis, bag-of-word features ($\{1, 2, 3\}$-grams for words and $\{3, 4, 5\}$-grams for characters) and linguistic features are the most important ones. Therefore, we implement both of them. There are three feature settings: (i) only bag-of-words features (for both, word and characters), (ii) only linguistic features, and (iii) the combination of bag-of-words and linguistic features. We use LIBLINEAR (Fan et al., 2008) to train the model and optimized the C parameter on the development set.

[2] http://sentiment.christopherpotts.net/lingstruc.html

[3] The list of emoticons was taken from SentiStrengh: http://sentistrength.wlv.ac.uk/

[4] objective instances were considered to be neutral

For reference we add the first and second best systems of the SemEval 2015 tweet level polarity task: Webis (Hagen et al., 2015) and UNITN (Severyn and Moschitti, 2015). Webis is an ensemble based on four systems, which participated in the same task of SemEval 2014. The UNITN system trains a CNN similar to ours. They rely on pretraining the entire model on a large distant supervised training corpus (10M labeled tweets). This approach is orthogonal to ours and can easily be combined with our idea if linguistic feature integration. This combination is likely to increase the performance further.

LingCNN To analyze the effect of the linguistic features and our extensions we train different CNN models with different combinations of features: (i) only pretrained word embeddings, (ii) integration of word-level features, and (iii) integration of sentence-level features. The model updates all parameters during training $\theta = \{P, M^*, W, b^{(*)}\}$. We set the embeddings size to $d_P = 60$. Our model uses filters of width 2 – 5 with 100 filters each. We train the models for a maximum of 30 epochs with mini-batch stochastic gradient descent (batch size: 100). The training was stopped when three consecutive epochs lead to worse results on the development set (early stopping). We use AdaGrad (Duchi et al., 2011) for dynamic learning rate adjustment with an initial learning rate of 0.01 and ℓ_2 regularization ($\lambda = 5e^{-5}$).

5.3 Results

Baselines Table 2 lists the SVM results. Similar to Mohammad et al. (2013)'s findings, the combination of ngram and linguistic features gives the best performance. Both SemEval participating systems beat the baseline by a large margin.

LingCNN The lower part of Table 2 shows the lingCNN results. With only word-level features the model yields similar performance to the SVM with only linguistic features. Adding sentence-level features improves the performance to the level of the SVM baseline system with bag-of-words and linguistic features. We see that using pretrained word embeddings to initialize the model yields large improvements. Sentence features on top of that can not improve the performance further. However, word-level features together with pretrained word embeddings yield higher performance. The best result is reached by the combination of word embeddings and both types of linguistic features. This performance is comparable with both state-of-the-art SemEval winner systems.

model	features			SemEval	Sent140
SVM	bow			50.51	67.34
	ling.			57.28	66.90
	bow + ling.			59.28	70.21
Webis				64.84	-
UNITN				64.59	-
	emb.	word	sent.		
lingCNN		+		57.83	72.58
		+	+	59.24	74.36
	+			62.72	77.59
	+		+	62.61	79.14
	+	+		63.43	80.21
	+	+	+	**64.46**	**80.75**

Table 2: Results of baselines (upper half) and lingCNN (lower half).

5.4 Analysis

Examples Here, we analyze examples on why the linguistic features help. Consider the example "saturday night in with toast , hot choc & <user> on e news #happydays". Only the hashtag '#happydays' indicates polarity. The hashtag exists in the sentiment lexicon, but does not exist in the training vocabulary. Therefore, there is no embedding for it. Here is another example: "shiiiiit my sats is on saturday . i'm going to fail". 'Fail' is strongly negative in all lexicons. However, it occurs only 10 times in the training set. That is likely not enough to learn a good sentiment-bearing embedding. As a result, the CNN without linguistic knowledge classifies the tweet as neutral. Having linguistic features enables the model to implicitly incorporate sentiment information into the word embeddings, helping to classify this example correctly.

Corpus Size In this section we analyze the benefit of linguistic features with respect to the size of the training corpus. Table 3 shows the performance of the CNN with and without linguistic features, where we only use the first 1000 and 3000 training samples. We clearly see that linguistic features are helpful in all cases. Especially, where only limited training data is available, the performance difference is large. Even with only 1000 training samples, the CNN with linguistic features yields a reasonable result of 60.89. The CNN that does not have access to this source of information reaches only 49.89. Although, the performance

	1000	3000	all
emb.	49.89	58.10	62.72
emb. + word + sent.	60.89	62.51	64.46

Table 3: Different training set sizes.

of the CNN without linguistic features increases much for 3000 training examples, this model is still more than 4 points behind the linguistically informed model.

6 Related Work

Collobert et al. (2011) published the first CNN architecture for a range of natural language processing tasks. We adopt their idea of using look-up tables to incorporate linguistic features at the word-level into the CNN.

Since then CNNs have been used for a variety of sentence classification tasks (e.g., Zeng et al. (2014)), including polarity classification (e.g., Kim (2014)). Kalchbrenner et al. (2014) showed that their DCNN for modeling sentences can achieve competitive results in this field. Our CNN architecture is simpler than theirs.

There are alternative approaches of integrating linguistic features into model training. By adding more labeled data, implicit knowledge is given to the model. This approach usually requires manual labeling effort. A different approach is to incorporate linguistic knowledge into the objective function to guide the model training. For instance Tang et al. (2014b) incorporate the polarity of an ngram into a hinge loss function.

Tang et al. (2014a) used a CNN to compute representations of input sentences. These representation together with linguistic features on sentence-level form the input to an SVM. In contrast, we use linguistic features at the word-level, which allows interaction between linguistic features and word embeddings. Furthermore, we use similar sentence-features and directly incorporate them into the CNN.

In addition to CNNs, researchers have been using different neural network architectures. However, each of these has its own disadvantages. A deep feed forward network cannot model easily that inserting many types of words into a string (e.g., "happy to drive my new car" vs "happy to drive my *red* new car") does not change sentiment. Recurrent Neural Networks (RNN) (Elman, 1990) and Long Short Term Memory networks (LSTM) (Hochreiter and Schmidhuber, 1997) are powerful for unbounded dependencies, but tweets are short; the sentiment of a tweet is usually determined by one part of it and unlike RNN/LSTM, convolution plus max pooling can learn to focus on that. Recursive architectures like the Recursive Neural Tensor Network (Socher et al., 2013). assume some kind of hierarchical sentence structure. This structure does not exist or is hard to recognize for many noisy tweets.

As mentioned before, we use the SemEval 2013 and SemEval 2014 winning system (Mohammad et al., 2013) as baseline. Moreover, we include several features of their system to improve the CNN.

7 Conclusion

In this paper we introduced an intuitive and simple way of incorporating linguistic word-level and sentence-level features into a CNN architecture. Using such features yields significant improvements on two polarity classification Twitter data sets. Using both feature types, our lingCNN performs comparable to state-of-the-art systems.

References

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (almost) from Scratch. *JMLR*, 12.

John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *JMLR*, 12.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2).

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *JMLR*, 9.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter Sentiment Classification using Distant Supervision.

Matthias Hagen, Martin Potthast, Michel Büchner, and Benno Stein. 2015. Webis: An Ensemble for Twitter Sentiment Detection. In *SemEval*.

Sepp Hochreiter and H. Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8).

Minqing Hu and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *KDD*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *ACL*.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*.

Andrew J. McMinn, Yashar Moshfeghi, and Joemon M. Jose. 2013. Building a large-scale corpus for evaluating event detection on twitter. In *CIKM*.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR*.

Saif M. Mohammad and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence*, 29(3).

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *SemEval*.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *SemEval*.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. 2013. Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters. In *NAACL HLT*.

Sara Rosenthal, Preslav Nakov, Svetlana Kiritchenko, Saif M. Mohammad, Alan Ritter, and Veselin Stoyanov. 2015. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In *SemEval*.

Aliaksei Severyn and Alessandro Moschitti. 2015. UNITN: Training Deep Convolutional Neural Network for Twitter Sentiment Classification. In *SemEval*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *EMNLP*.

Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. 2014a. Coooolll: A Deep Learning System for Twitter Sentiment Classification. In *SemEval*.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014b. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In *ACL*.

Theresa Ann Wilson, Janyce Wiebe, and Paul Hoffmann. 2005. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In *HLT/EMNLP*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. 2014. Relation Classification via Convolutional Deep Neural Network. In *COLING*.