Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts

Cícero Nogueira dos Santos
Brazilian Research Lab
IBM Research
`cicerons@br.ibm.com`

Maíra Gatti
Brazilian Research Lab
IBM Research
`mairacg@br.ibm.com`

Abstract

Sentiment analysis of short texts such as single sentences and Twitter messages is challenging because of the limited contextual information that they normally contain. Effectively solving this task requires strategies that combine the small text content with prior knowledge and use more than just bag-of-words. In this work we propose a new deep convolutional neural network that exploits from character- to sentence-level information to perform sentiment analysis of short texts. We apply our approach for two corpora of two different domains: the Stanford Sentiment Treebank (SSTb), which contains sentences from movie reviews; and the Stanford Twitter Sentiment corpus (STS), which contains Twitter messages. For the SSTb corpus, our approach achieves state-of-the-art results for single sentence sentiment prediction in both binary positive/negative classification, with 85.7% accuracy, and fine-grained classification, with 48.3% accuracy. For the STS corpus, our approach achieves a sentiment prediction accuracy of 86.4%.

1 Introduction

The advent of online social networks has produced a crescent interest on the task of sentiment analysis for short text messages (Go et al., 2009; Barbosa and Feng, 2010; Nakov et al., 2013). However, sentiment analysis of short texts such as single sentences and and microblogging posts, like Twitter messages, is challenging because of the limited amount of contextual data in this type of text. Effectively solving this task requires strategies that go beyond bag-of-words and extract information from the sentence/message in a more disciplined way. Additionally, to fill the gap of contextual information in a scalable manner, it is more suitable to use methods that can exploit prior knowledge from large sets of unlabeled texts.

In this work we propose a deep convolutional neural network that exploits from character- to sentence-level information to perform sentiment analysis of short texts. The proposed network, named Character to Sentence Convolutional Neural Network (CharSCNN), uses two convolutional layers to extract relevant features from words and sentences of any size. The proposed network can easily explore the richness of word embeddings produced by unsupervised pre-training (Mikolov et al., 2013). We perform experiments that show the effectiveness of CharSCNN for sentiment analysis of texts from two domains: movie review sentences; and Twitter messages (tweets). CharSCNN achieves state-of-the-art results for the two domains. Additionally, in our experiments we provide information about the usefulness of unsupervised pre-training; the contribution of character-level features; and the effectiveness of sentence-level features to detect negation.

This work is organized as follows. In Section 2, we describe the proposed the Neural Network architecture. In Section 3, we discuss some related work. Section 4 details our experimental setup and results. Finally, in Section 5 we present our final remarks.

2 Neural Network Architecture

Given a sentence, CharSCNN computes a score for each sentiment label $\tau \in T$. In order to score a sentence, the network takes as input the sequence of words in the sentence, and passes it through a sequence of layers where features with increasing levels of complexity are extracted. The network extracts features from the character-level up to the sentence-level. The main novelty in our network architecture is the inclusion of two convolutional layers, which allows it to handle words and sentences of any size.

2.1 Initial Representation Levels

The first layer of the network transforms words into real-valued feature vectors (embeddings) that capture morphological, syntactic and semantic information about the words. We use a fixed-sized word vocabulary V^{wrd}, and we consider that words are composed of characters from a fixed-sized character vocabulary V^{chr}. Given a sentence consisting of N words $\{w_1, w_2, ..., w_N\}$, every word w_n is converted into a vector $u_n = [r^{wrd}; r^{wch}]$, which is composed of two sub-vectors: the *word-level embedding* $r^{wrd} \in \mathbb{R}^{d^{wrd}}$ and the *character-level embedding* $r^{wch} \in \mathbb{R}^{cl^0_u}$ of w_n. While word-level embeddings are meant to capture syntactic and semantic information, character-level embeddings capture morphological and shape information.

2.1.1 Word-Level Embeddings

Word-level embeddings are encoded by column vectors in an embedding matrix $W^{wrd} \in \mathbb{R}^{d^{wrd} \times |V^{wrd}|}$. Each column $W^{wrd}_i \in \mathbb{R}^{d^{wrd}}$ corresponds to the word-level embedding of the i-th word in the vocabulary. We transform a word w into its word-level embedding r^{wrd} by using the matrix-vector product:

$$r^{wrd} = W^{wrd} v^w \tag{1}$$

where v^w is a vector of size $\left|V^{wrd}\right|$ which has value 1 at index w and zero in all other positions. The matrix W^{wrd} is a parameter to be learned, and the size of the word-level embedding d^{wrd} is a hyper-parameter to be chosen by the user.

2.1.2 Character-Level Embeddings

Robust methods to extract morphological and shape information from words must take into consideration all characters of the word and select which features are more important for the task at hand. For instance, in the task of sentiment analysis of Twitter data, important information can appear in different parts of a hash tag (e.g., "*#SoSad*", "*#ILikeIt*") and many informative adverbs end with the suffix "*ly*" (e.g. "*beautifully*", "*perfectly*" and "*badly*"). We tackle this problem using the same strategy proposed in (dos Santos and Zadrozny, 2014), which is based on a convolutional approach (Waibel et al., 1989). As depicted in Fig. 1, the convolutional approach produces local features around each character of the word and then combines them using a max operation to create a fixed-sized character-level embedding of the word.

Given a word w composed of M characters $\{c_1, c_2, ..., c_M\}$, we first transform each character c_m into a character embedding r^{chr}_m. Character embeddings are encoded by column vectors in the embedding matrix $W^{chr} \in \mathbb{R}^{d^{chr} \times |V^{chr}|}$. Given a character c, its embedding r^{chr} is obtained by the matrix-vector product:

$$r^{chr} = W^{chr} v^c \tag{2}$$

where v^c is a vector of size $\left|V^{chr}\right|$ which has value 1 at index c and zero in all other positions. The input for the convolutional layer is the sequence of character embeddings $\{r^{chr}_1, r^{chr}_2, ..., r^{chr}_M\}$.

The convolutional layer applies a matrix-vector operation to each window of size k^{chr} of successive windows in the sequence $\{r^{chr}_1, r^{chr}_2, ..., r^{chr}_M\}$. Let us define the vector $z_m \in \mathbb{R}^{d^{chr}k^{chr}}$ as the concatenation of the character embedding m, its $(k^{chr} - 1)/2$ left neighbors, and its $(k^{chr} - 1)/2$ right neighbors[1]:

$$z_m = \left(r^{chr}_{m-(k^{chr}-1)/2}, ..., r^{chr}_{m+(k^{chr}-1)/2}\right)^T$$

[1]We use a special *padding character* for the characters with indices outside of the word boundaries.

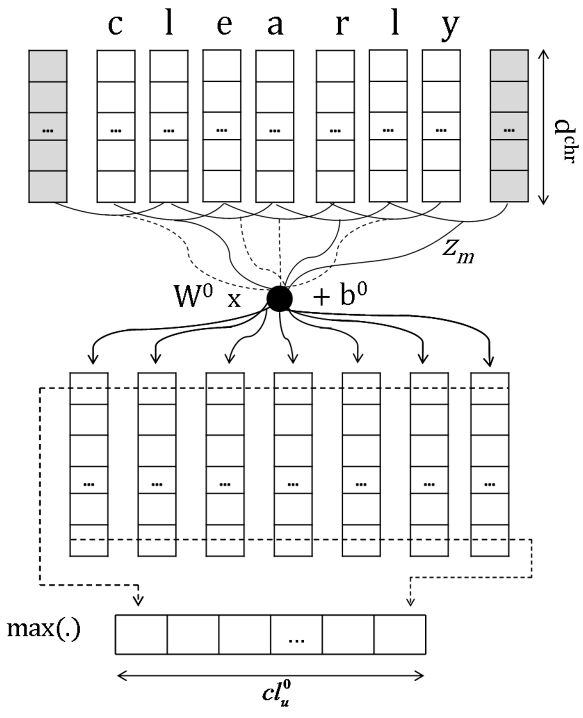

Figure 1: Convolutional approach to character-level feature extraction.

The convolutional layer computes the j-th element of the vector $r^{wch} \in \mathbb{R}^{cl^0_u}$, which is the character-level embedding of w, as follows:

$$[r^{wch}]_j = \max_{1<m<M} \left[W^0 z_m + b^0 \right]_j \tag{3}$$

where $W^0 \in \mathbb{R}^{cl^0_u \times d^{chr} k^{chr}}$ is the weight matrix of the convolutional layer. The same matrix is used to extract local features around each character window of the given word. Using the max over all character windows of the word, we extract a "global" fixed-sized feature vector for the word.

Matrices W^{chr} and W^0, and vector b^0 are parameters to be learned. The size of the character vector d^{chr}, the number of convolutional units cl^0_u (which corresponds to the size of the character-level embedding of a word), and the size of the character context window k^{chr} are hyper-parameters.

2.2 Sentence-Level Representation and Scoring

Given a sentence x with N words $\{w_1, w_2, ..., w_N\}$, which have been converted to joint word-level and character-level embedding $\{u_1, u_2, ..., u_N\}$, the next step in CharSCNN consists in extracting a sentence-level representation r^{sent}_x. Methods to extract a sentence-wide feature set most deal with two main problems: sentences have different sizes; and important information can appear at any position in the sentence. We tackle these problems by using a convolutional layer to compute the sentence-wide feature vector r^{sent}. This second convolutional layer in our neural network architecture works in a very similar way to the one used to extract character-level features for words. This layer produces local features around each word in the sentence and then combines them using a max operation to create a fixed-sized feature vector for the sentence.

The second convolutional layer applies a matrix-vector operation to each window of size k^{wrd} of successive windows in the sequence $\{u_1, u_2, ..., u_N\}$. Let us define the vector $z_n \in \mathbb{R}^{(d^{wrd}+cl^0_u)k^{wrd}}$ as the concatenation of a sequence of k^{wrd} embeddings, centralized in the n-th word[2]:

$$z_n = \left(u_{n-(k^{wrd}-1)/2}, ..., u_{n+(k^{wrd}-1)/2} \right)^T$$

[2]We use a special *padding token* for the words with indices outside of the sentence boundaries.

The convolutional layer computes the j-th element of the vector $r^{sent} \in \mathbb{R}^{cl_u^1}$ as follows:

$$[r^{sent}]_j = \max_{1<n<N} \left[W^1 z_n + b^1\right]_j \tag{4}$$

where $W^1 \in \mathbb{R}^{cl_u^1 \times (d^{wrd} + cl_u^0) k^{wrd}}$ is the weight matrix of the convolutional layer. The same matrix is used to extract local features around each word window of the given sentence. Using the max over all word windows of the sentence, we extract a "global" fixed-sized feature vector for the sentence. Matrix W^1 and vector b^1 are parameters to be learned. The number of convolutional units cl_u^1 (which corresponds to the size of the sentence-level feature vector), and the size of the word context window k^{wrd} are hyper-parameters to be chosen by the user.

Finally, the vector r_x^{sent}, the "global' feature vector of sentence x, is processed by two usual neural network layers, which extract one more level of representation and compute a score for each sentiment label $\tau \in T$:

$$s(x) = W^3 h(W^2 r_x^{sent} + b^2) + b^3 \tag{5}$$

where matrices $W^2 \in \mathbb{R}^{hl_u \times cl_u^1}$ and $W^3 \in \mathbb{R}^{|T| \times hl_u}$, and vectors $b^2 \in \mathbb{R}^{hl_u}$ and $b^3 \in \mathbb{R}^{|T|}$ are parameters to be learned. The transfer function $h(.)$ is the hyperbolic tangent. The number of hidden units hl_u is a hyper-parameter to be chosen by the user.

2.3 Network Training

Our network is trained by minimizing a negative likelihood over the training set D. Given a sentence x, the network with parameter set θ computes a score $s_\theta(x)_\tau$ for each sentiment label $\tau \in T$. In order to transform these scores into a conditional probability distribution of labels given the sentence and the set of network parameters θ, we apply a softmax operation over the scores of all tags $\tau \in T$:

$$p\left(\tau|x,\theta\right) = \frac{e^{s_\theta(x)_\tau}}{\sum\limits_{\forall i \in T} e^{s_\theta(x)_i}} \tag{6}$$

Taking the log, we arrive at the following conditional log-probability:

$$\log p\left(\tau|x,\theta\right) = s_\theta(x)_\tau - \log\left(\sum_{\forall i \in T} e^{s_\theta(x)_i}\right) \tag{7}$$

We use stochastic gradient descent (SGD) to minimize the negative log-likelihood with respect to θ:

$$\theta \mapsto \sum_{(x,y)\in D} -\log p(y|x,\theta) \tag{8}$$

where (x, y) corresponds to a sentence in the training corpus D and y represents its respective label.

The backpropagation algorithm is a natural choice to efficiently compute gradients of network architectures such as the one proposed in this work (Lecun et al., 1998; Collobert, 2011). In order to perform our experiments, we implement the proposed CharSCNN architecture using the *Theano* library (Bergstra et al., 2010). *Theano* is a versatile Python library that allows the efficient definition, optimization, and evaluation of mathematical expressions involving multi-dimensional arrays. We use *Theano*'s automatic differentiation capabilities in order to implement the backpropagation algorithm.

3 Related Work

There are a few works on neural network architectures for sentiment analysis. In (Socher et al., 2011), the authors proposed a semi-supervised approach based on recursive autoencoders for predicting sentiment distributions. The method learns vector space representation for multi-word phrases and exploits the recursive nature of sentences. In (Socher et al., 2012), it is proposed a matrix-vector recursive neural network model for semantic compositionality, which has the ability to learn compositional vector representations for phrases and sentences of arbitrary length. The vector captures the inherent meaning of the constituent, while the matrix captures how the meaning of neighboring words and phrases are changed. In (Socher et al., 2013b) the authors propose the Recursive Neural Tensor Network (RNTN) architecture, which represents a phrase through word vectors and a parse tree and then compute vectors for higher nodes in the tree using the same tensor-based composition function. Our approach differ from these previous works because it uses a feed-forward neural network instead of a recursive one. Moreover, it does not need any input about the syntactic structure of the sentence.

Regarding convolutional networks for NLP tasks, in (Collobert et al., 2011), the authors use a convolutional network for the semantic role labeling task with the goal avoiding excessive task-specific feature engineering. In (Collobert, 2011), the authors use a similar network architecture for syntactic parsing. CharSCNN is related to these works because they also apply convolutional layers to extract sentence-level features. The main difference in our neural network architecture is the addition of one convolutional layer to extract character features.

In terms of using intra-word information in neural network architectures for NLP tasks, Alexandrescu et al. (2006) present a factored neural language model where each word is represented as a vector of features such as stems, morphological tags and cases and a single embedding matrix is used to look up all of these features. In (Luong et al., 2013), the authors use a recursive neural network (RNN) to explicitly model the morphological structures of words and learn morphologically-aware embeddings. Lazaridou et al. (Lazaridou et al., 2013) use compositional distributional semantic models, originally designed to learn meanings of phrases, to derive representations for complex words, in which the base unit is the morpheme. In (Chrupala, 2013), the author proposes a simple recurrent network (SRN) to learn continuous vector representations for sequences of characters, and use them as features in a conditional random field classifier to solve a character level text segmentation and labeling task. The main advantage of our approach to extract character-level features is it flexibility. The convolutional layer allows the extraction of relevant features from any part of the word and do not need handcrafted inputs like stems and morpheme lists (dos Santos and Zadrozny, 2014).

4 Experimental Setup and Results

4.1 Sentiment Analysis Datasets

We apply CharSCNN for two different corpora from two different domains: movie reviews and Twitter posts. The movie review dataset used is the recently proposed Stanford Sentiment Treebank (SSTb) (Socher et al., 2013b), which includes fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences. In our experiments we focus in sentiment prediction of complete sentences. However, we show the impact of training with sentences and phrases instead of only sentences.

The second labeled corpus we use is the Stanford Twitter Sentiment corpus (STS) introduced by (2009). The original training set contains 1.6 million tweets that were automatically labeled as positive/negative using emoticons as noisy labels. The test set was manually annotated by Go et al. (2009). In our experiments, to speedup the training process we use only a sample of the training data consisting of 80K (5%) randomly selected tweets. We also construct a development set by randomly selecting 16K tweets from Go et al.'s training set. In Table 1, we present additional details about the two corpora.

Dataset	Set	# sentences / tweets	# classes
SSTb	Train	8544	5
	Dev	1101	5
	Test	2210	5
STS	Train	80K	2
	Dev	16K	2
	Test	498	3

Table 1: Sentiment Analysis datasets.

4.2 Unsupervised Learning of Word-Level Embeddings

Word-level embeddings play a very important role in the CharSCNN architecture. They are meant to capture syntactic and semantic information, which are very important to sentiment analysis. Recent work has shown that large improvements in terms of model accuracy can be obtained by performing unsupervised pre-training of word embeddings (Collobert et al., 2011; Luong et al., 2013; Zheng et al., 2013; Socher et al., 2013a). In our experiments, we perform unsupervised learning of word-level embeddings using the *word2vec* tool[3], which implements the *continuous bag-of-words* and *skip-gram* architectures for computing vector representations of words (Mikolov et al., 2013).

We use the December 2013 snapshot of the English Wikipedia corpus as a source of unlabeled data. The Wikipedia corpus has been processed using the following steps: (1) removal of paragraphs that are not in English; (2) substitution of non-western characters for a special character; (3) tokenization of the text using the tokenizer available with the Stanford POS Tagger (Manning, 2011); (4) and removal of sentences that are less than 20 characters long (including white spaces) or have less than 5 tokens. Like in (Collobert et al., 2011) and (Luong et al., 2013), we lowercase all words and substitute each numerical digit by a 0 (e.g., *1967* becomes *0000*). The resulting clean corpus contains about 1.75 billion tokens.

When running the *word2vec* tool, we set that a word must occur at least 10 times in order to be included in the vocabulary, which resulted in a vocabulary of 870,214 entries. To train our word-level embeddings we use *word2vec*'s skip-gram method with a context window of size 9. The training time for the English corpus is around 1h10min using 12 threads in a Intel® Xeon® E5-2643 3.30GHz machine.

In our experiments, we do not perform unsupervised pre-training of character-level embeddings, which are initialized by randomly sampling each value from an uniform distribution: $\mathcal{U}(-r, r)$, where $r = \sqrt{\frac{6}{|V^{chr}| + d^{chr}}}$. There are 94 different characters in the SSTb corpus and 453 different characters in the STS corpus. Since the two character vocabularies are relatively small, it has been possible to learn reliable character-level embeddings using the labeled training corpora. The raw (not lowercased) words are used to construct the character vocabularies, which allows the network to capture relevant information about capitalization.

4.3 Model Setup

We use the development sets to tune the neural network hyper-parameters. Many different combinations of hyper-parameters can give similarly good results. We spent more time tuning the learning rate than tuning other parameters, since it is the hyper-parameter that has the largest impact in the prediction performance. The only two parameters with different values for the two datasets are the learning rate and the number of units in the convolutional layer that extract sentence features. This provides some indication on the robustness of our approach to multiple domains. For both datasets, the number of training epochs varies between five and ten. In Table 2, we show the selected hyper-parameter values for the two labeled datasets.

Parameter	**Parameter Name**	**SSTb**	**STS**
d^{wrd}	Word-Level Embeddings dimension	30	30
k^{wrd}	Word Context window	5	5
d^{chr}	Char. Embeddings dimension	5	5
k^{chr}	Char. Context window	3	3
cl_u^0	Char. Convolution Units	10	50
cl_u^1	Word Convolution Units	300	300
hl_u	Hidden Units	300	300
λ	Learning Rate	0.02	0.01

Table 2: Neural Network Hyper-Parameters

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

In order to assess the effectiveness of the proposed character-level representation of words, we compare the proposed architecture CharSCNN with an architecture that uses only word embeddings. In our experiments, SCNN represents a network which is fed with word representations only, i.e, for each word w_n its embedding is $u_n = r^{wrd}$. For SCNN, we use the same NN hyper-parameters values (when applicable) shown in Table 2.

4.4 Results for SSTb Corpus

In Table 3, we present the result of CharSCNN and SCNN for different versions of the SSTb corpus. Note that SSTb corpus is a sentiment treebank, hence it contains sentiment annotations for all phrases in all sentences in the corpus. In our experiments, we check whether using examples that are single phrases, in addition to complete sentences, can provide useful information for training the proposed NN. However, in our experiments the test set always includes only complete sentences. In Table 3, the column *Phrases* indicates whether all phrases (*yes*) or only complete sentences (*no*) in the corpus are used for training. The *Fine-Grained* column contains prediction results for the case where 5 sentiment classes (labels) are used (*very negative*, *negative*, *neutral*, *positive*, *very positive*). The *Positive/Negative* column presents prediction results for the case of binary classification of sentences, i.e, the neutral class is removed, the two negative classes are merged as well as the two positive classes.

Model	Phrases	Fine-Grained	Positive/Negative
CharSCNN	yes	**48.3**	**85.7**
SCNN	yes	**48.3**	85.5
CharSCNN	no	43.5	82.3
SCNN	no	43.5	82.0
RNTN (Socher et al., 2013b)	yes	45.7	85.4
MV-RNN (Socher et al., 2013b)	yes	44.4	82.9
RNN (Socher et al., 2013b)	yes	43.2	82.4
NB (Socher et al., 2013b)	yes	41.0	81.8
SVM (Socher et al., 2013b)	yes	40.7	79.4

Table 3: Accuracy of different models for fine grained (5-class) and binary predictions using SSTb.

In Table 3, we can note that CharSCN and SCNN have very similar results in both fine-grained and binary sentiment prediction. These results suggest that the character-level information is not much helpful for sentiment prediction in the SSTb corpus. Regarding the use of phrases in the training set, we can note that, even not explicitly using the syntactic tree information when performing prediction, CharSCNN and SCNN benefit from the presence of phrases as training examples. This result is aligned with Socher et al.'s (2013b) suggestion that information of sentiment labeled phrases improves the accuracy of other classification algorithms such as support vector machines (SVM) and naive Bayes (NB). We believe that using phrases as training examples allows the classifier to learn more complex phenomena, since sentiment labeled phrases give the information of how words (phrases) combine to form the sentiment of phrases (sentences). However, it is necessary to perform more detailed experiments to confirm this conjecture.

Regarding the fine-grained sentiment prediction, our approach provides an absolute accuracy improvement of 2.6 over the RNTN approach proposed by (Socher et al., 2013b), which is the previous best reported result for SSTb. CharSCN, SCNN and Socher et al.'s RNTN have similar accuracy performance for binary sentiment prediction. Compared to RNTN, our method has the advantage of not needing the output of a syntactic parser when performing sentiment prediction. For comparison reasons, in Table 3 we also report Socher et al.'s (2013b) results for sentiment classifiers trained with recursive neural networks (RNN), matrix-vector RNN (MV-RNN), NB, and SVM algorithms.

Initializing word-embeddings using unsupervised pre-training gives an absolute accuracy increase of around 1.5 when compared to randomly initializing the vectors. The *Theano* based implementation of CharSCNN takes around 10 min. to complete one training epoch for the SSTb corpus with all phrases and five classes. In our experiments, we use 4 threads in a Intel® Xeon® E5-2643 3.30GHz machine.

4.5 Results for STS Corpus

In Table 4, we present the results of CharSCNN and SCNN for sentiment prediction using the STS corpus. As expected, character-level information has a greater impact for Twitter data. Using unsupervised pre-training, CharSCNN provides an absolute accuracy improvement of 1.2 over SCNN. Additionally, initializing word-embeddings using unsupervised pre-training gives an absolute accuracy increase of around 4.5 when compared to randomly initializing the word-embeddings.

In Table 4, we also compare CharSCNN performance with other approaches proposed in the literature. In (Speriosu et al., 2011), a label propagation (LProp) approach is proposed, while Go et al. (2009) use maximum entropy (MaxEnt), NB and SVM-based classifiers. CharSCNN outperforms the previous approaches in terms of prediction accuracy. As far as we know, 86.4 is the best prediction accuracy reported so far for the STS corpus.

Model	**Accuracy (unsup. pre-training)**	**Accuracy (random word embeddings)**
CharSCNN	**86.4**	81.9
SCNN	85.2	82.2
LProp (Speriosu et al., 2011)	84.7	
MaxEnt (Go et al., 2009)	83.0	
NB (Go et al., 2009)	82.7	
SVM (Go et al., 2009)	82.2	

Table 4: Accuracy of different models for binary predictions (positive/negative) using STS Corpus.

4.6 Sentence-level features

In figures 2 and 3 we present the behavior of CharSCNN regarding the sentence-level features extracted for two cases of negation, which are correctly predicted by CharSCNN. We choose these cases because negation is an important issue in sentiment analysis. Moreover, the same sentences are also used as illustrative examples in (Socher et al., 2013b). Note that in the convolutional layer, 300 features are first extracted for each word. Then the $\max$ operator selects the 300 features which have the largest values among the words to construct the sentence-level feature set r^{sent}. Figure 2 shows a positive sentence (left) and its negation. We can observe that in both versions of the sentence, the extracted features concentrate mainly around the main topic, *"film"*, and the part of the phrase that indicates sentiment (*"liked"* and *"did 'nt like"*). Note in the left chart that the word *"liked"* has a big impact in the set of extracted features. On the other hand, in the right chart, we can see that the impact of the word *"like"* is reduced because of the negation *"did 'nt"*, which is responsible for a large part of the extracted features.

In Figure 3 a similar behavior can be observed. While the very negative expression *"incredibly dull"* is responsible for 69% of the features extracted from the sentence in the left, its negation *"definitely not dull"*, which is somewhat more positive, is responsible for 77% of the features extracted from the sentence in the chart at right . These examples indicate CharSCNN's robustness to handle negation, as well as its ability to capture information that is important to sentiment prediction.

5 Conclusions

In this work we present a new deep neural network architecture that jointly uses character-level, word-level and sentence-level representations to perform sentiment analysis. The main contributions of the paper are: (1) the idea of using convolutional neural networks to extract from character- to sentence-level features; (2) the demonstration that a feed-forward neural network architecture can be as effective as RNTN (Socher et al., 2013a) for sentiment analysis of sentences; (3) the definition of new state-of-the-art results for SSTb and STS corpora.

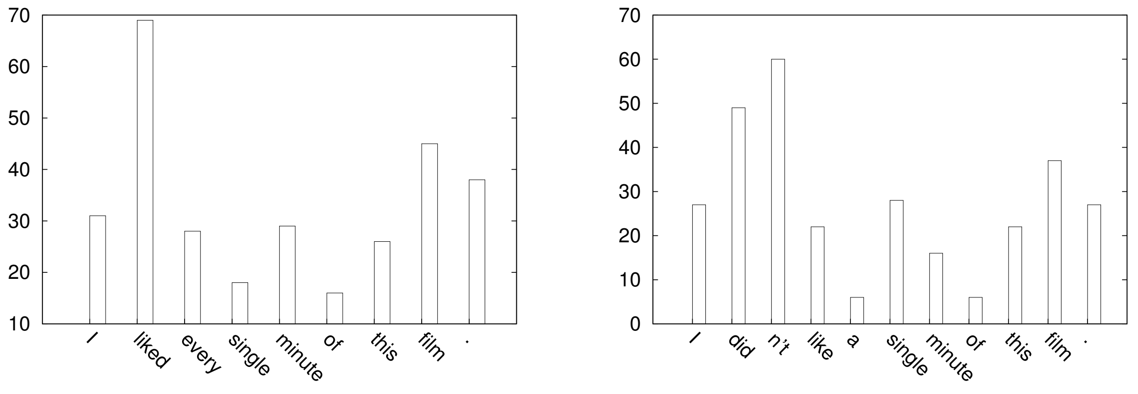

Figure 2: Number of local features selected at each word when forming the sentence-level representation. In this example, we have a positive sentence (left) and its negation (right).

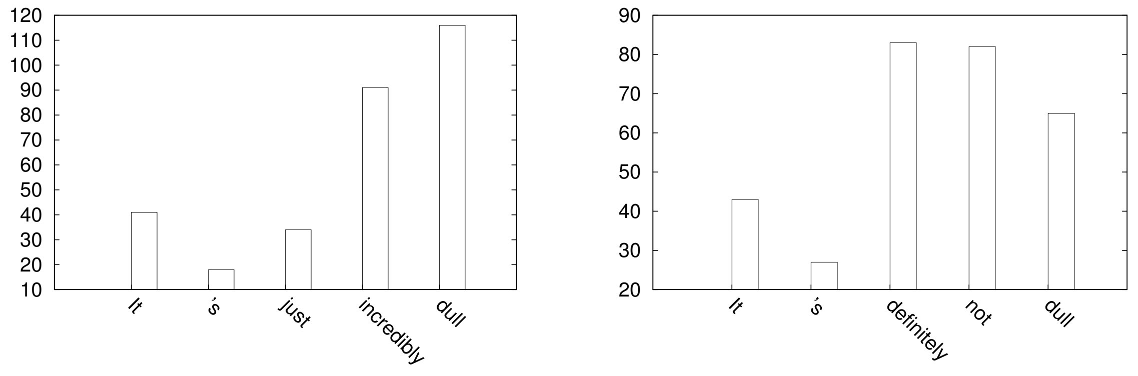

Figure 3: Number of local features selected at each word when forming the sentence-level representation. In this example, we have a negative sentence (left) and its negation (right).

As future work, we would like to analyze in more detail the role of character-level representations for sentiment analysis of tweets. Additionally, we would like to check the impact of performing the unsupervised pre-training step using texts from the specific domain at hand.

References

Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 1–4, New York City, USA, June.

Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 36–44.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Grzegorz Chrupala. 2013. Text segmentation with character-level text embeddings. In *Proceedings of the ICML workshop on Deep Learning for Audio, Speech and Language Processing*.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

R. Collobert. 2011. Deep learning for efficient discriminative parsing. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 224–232.

Cícero Nogueira dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning, JMLR: W&CP volume 32*, Beijing, China.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. Technical report, Stanford University.

Angeliki Lazaridou, Marco Marelli, Roberto Zamparelli, and Marco Baroni. 2013. Compositional–ly derived representations of morphologically complex words in distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1517–1526.

Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

Minh-Thang Luong, Richard Socher, and Christopher D. Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Conference on Computational Natural Language Learning*, Sofia, Bulgaria.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'11, pages 171–189.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at International Conference on Learning Representations*.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. Semeval-2013 task 2: Sentiment analysis in twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA, June. Association for Computational Linguistics.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 151–161.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of theConference on Empirical Methods in Natural Language Processing*, pages 1201–1211.

Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. 2013a. Parsing with compositional vector grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Michael Speriosu, Nikita Sudan, Sid Upadhyay, and Jason Baldridge. 2011. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First Workshop on Unsupervised Learning in NLP*, EMNLP, pages 53–63.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3):328–339.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for chinese word segmentation and pos tagging. In *Proceedings of the Conference on Empirical Methods in NLP*, pages 647–657.