Sentiment Analysis Using Convolutions: a Baseline Comparison

Jasper Adegeest 11650273
Simone van Bruggen 11869704
Alexander Keijser 11111585
Mark Romme 11415673

Laurens Samson 10448004

Abstract

In this paper, we discuss the use of recurrent and convolutional approaches for the task of sentiment analysis. A vanilla RNN, a LSTM and a convolutional architecture are implemented and tested on the Stanford Twitter Sentiment corpus (STS). On the binary classification task, our convolutional architecture reaches a accuracy of 75.2%, which outperforms the recurrent models. Additionally, qualitative analysis on both approaches is performed to gain further insight in the performance of the models.

1 Introduction

Sentiment analysis on text has become a subject of interest in a variety of applications such as user review understanding, tweet analysis and chat bots and effectively performing this task has been a focus areas in Natural Language Processing (NLP) research for the past decades.

The performance of algorithms in the main focus areas of NLP have increased significantly over the past couple of years due to recurrent neural networks (RNN) (Graves et al., 2013). These models are particularly good at handling sequential data, such as speech, audio and text. Mesnil et al. (2014) show that a simple RNN already performs very well on the sentiment analysis task and they achieved state-of-the-art performance combining an RNN with other approaches, such as n-grams and sentence embeddings.

Furthermore, within all major fields of AI, deep neural networks have shown to outperform the existing state-of-the-art methods. In particular the success of Convolutional Neural Networks (CNN) showed remarkable progress in computer vision tasks (Krizhevsky et al., 2012). CNNs haven also proven very successful for NLP applications (Kim, 2014), including sentiment analysis. Nogueira dos Santos and Gatti (2014) introduce a CNN capable of learning word representations at both character-level and word-level. These approaches accomplish state-of-the-art results for predicting the sentiment of short texts.

In this paper, CNN methods are compared to RNN methods for the task of sentiment analysis. Two different kind of RNNs (Vanilla RNN and long short-term memory (LSTM)) will be compared to the Character Sentence CNN (CharSCNN) model. First, a description of the different models will be provided. Secondly, the experiments will be described in detail and the results are discussed including a detailed comparison between the two methods. Finally, a conclusion is drawn, including some discussion on the current research and some ideas to further investigate and improve these models.

2 Models

2.1 RNN

Recurrent Neural Networks (RNNs) are popular for modeling sequences, as they are capable of learning sequences over time steps. This makes them suitable for the task of sentiment analysis. The most simple RNN (or *Vanilla* RNN) is a sequential model that at each time step t takes in an input x_t and its previous state h^{t-1}. RNNs greatly outperform traditional feed-forward architectures in handling sequential data. However, passing on the hidden state of the network can lead to vanishing or exploding gradients during back-propagation (Bengio et al., 1994). This becomes especially problematic for longer sequences. We implement a vanilla one layer RNN model with an embedding layer, with random initialized weights and a hidden state with zero initialization.

2.2 LSTM

Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) provide a solution to some of the issues of RNNs by introduc-

ing an internal memory cell with a forget-, input-, and output gate which enables it to learn long-term dependencies. The LSTM uses a similar implementation to the RNN model, with random weight initialization and zero initialization for the hidden layer.

2.3 Convolutional approach: CharSCNN

Instead of the sequential modeling of the recurrent approach, a convolution-based network can be used. The convolutional layers of such a network look at the input for a certain window size, and extract local features for each window. One of the benefits of this convolutional architecture is its robustness towards variable sequence length, whereas recurrent approaches require extra preprocessing such as padding.

Moreover, the basic recurrent approaches discussed above only use information on word- and sentence level. Often there is a lot of information about the sentence on the character-level which can be encoded in character embeddings. (Nogueira dos Santos and Gatti, 2014) propose a convolutional architecture, which uses both lower-level and higher-level features by using convolutional layers at the character-level and word-level. Our convolutional network uses a similar architecture, which combines the word-level and character-level features using convolutions. A high-level overview of the network can be found in Fig 1. The operations performed on the word and character embeddings are described in the following sections.

2.3.1 Word embeddings

The word embeddings encode the words of the input into a column vector W_i^{word} which is a column vector from the embedding matrix $W^{word} \in \mathbb{R}^{d_w \times V_w}$ where d_w denotes the embedding size and V_w the vocabulary size. The embedding matrix is randomly initialized and learned during training.

2.3.2 Character embeddings

To incorporate more information about the morphology of the input, character-level features can be used to have more fine-grained representations of each words. Typical negative or positive prefixes (such as *un-* (e.g. *uneventful*) or *in-* (e.g. *inadequate*)) are indicators at the character-level for a specific sentiment. Social media data such as tweets can contain hash tags used to express an

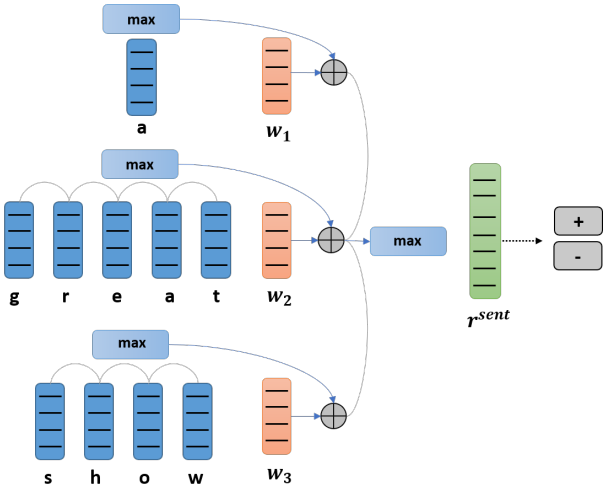

Figure 1: High-level overview of CharSCNN architecture. This example shows a short sentence '*a great show*' which is convoluted with a window size 2 and combined into a sentence embedding used for classification.

opinion (e.g. *#love*), and are useful to detect and take into account when classifying sentiments.

The character embeddings for each character $c \in \{c_1, ..., c_M\}$ are defined as column vectors $W^{char} \in \mathbb{R}^{d_{ch} \times V_{ch}}$ where d_{ch} denotes the character embedding size and V_{ch} denotes the number of characters. Next, a convolutional layer with window size k^{char} is applied to the character embeddings, where a special padding is used at the start and end (not shown in Fig 1). The character embeddings for each window are concatenated into a vector $z_m \in \mathbb{R}^{d_{char} \times k_{char}}$ and then a max operation over each concatenated dimension is performed:

$$[r_i^{wchar}] = \max_{1<m<M} [W_{z_m}^{conv_0} + b^{conv_0}]_j \quad (1)$$

where W^{conv_0} and b^{conv_0} are the weights and bias of the convolutional layer, and M the number of characters.

2.3.3 Sentence embedding

The word embedding r_i^{word} and character-level word embedding r_i^{wchar} are combined in a concatenated vector $u_i \in R^{(d^{word}+d^{char})}$. The embedding sizes d^{word}, d^{char} are hyper-parameters that can be optimized.

The concatenated vectors u are the input for sentence-level convolutional layer which performs a similar operation as (1) on its concatenated embeddings for each window $z_n \in \mathbb{R}^{d_u \times k_{sent}}$:

$$[r^{sent}] = \max_{1<n<N} [W_{z_n}^{conv_1} + b^{conv_1}]_j \quad (2)$$

where W^{conv_1} and b^{conv_1} are the weights and bias of the sentence-level convolutional layer and N the number of words. The final embedding r^{sent} is passed on to two fully connected linear layers which are left implicit in Fig 1 with the dashed line on the right.

2.4 Training

As we want to output the most likely label for each sample point, the softmax operation is performed over the output of the label to obtain a conditional probability distribution. To optimize the models, the negative log likelihood for each model is minimized using SGD with momentum:

$$\theta^{T+1} = \theta^{T} + \eta \sum_{j} - \log p(t_j | x_j, \theta^{T-1}) \quad (3)$$

where θ is the set of parameters we want to optimize, and (x_j, t_j) is an input pair of data and target label.

3 Experiments and results

3.1 Data

The experiments are conducted using the annotated Stanford Twitter Sentiment corpus data by (Go et al., 2009). This data set contains 1.6 million tweets, from which we used a subset of 32,000 tweets. These tweets consist of textual user input and a binary label indicating a positive or negative tweet. To make sure that the models are not biased towards a specific class due to a skewed training set, we balance the training data with an equal number of positive and negative examples. Furthermore, 20% of the data was used for validation and 20% for test purposes and the remaining 60% was used for training.

3.1.1 Preprocessing

The data set consists of a set of comments that contain an unknown number of sentences. It was therefore decided to first tokenize the comments on sentence level, whereafter these sentences where tokenized on word level, both using the Punkt Tokenizer (Kiss and Strunk, 2006). The input for the system was the list of word tokens concatenated for the different sentences. Finally, the comments where transformed to numerical inputs which used padding to the maximal number of words in a comment and the maximal characters in a word with the 0 token.

Parameter	RNN	LSTM	CharSCNN
d^{word}	512	512	128
d^{char}	-	-	128
k^{char}	-	-	3
k^{word}	-	-	3
Hidden units	20	20	-
Non-linearities	ReLU	ReLU	Tanh
Optimizer	SGD	SGD	SGD
Learning rate	0.01	0.01	0.001

Table 1: Final hyper-parameters used for testing

Model	Accuracy
RNN	69.0%
LSTM	68.0%
CharSCNN	75.2%

Table 2: Accuracy of the different models on the test data.

3.2 Setup

The architectures discussed in Section 2 are implemented in Python using the PyTorch package. (Paszke et al., 2017)[1] To increase training speed, the data was handled in mini-batches of size 64.

The optimal hyper-parameters for each model were found by optimizing on the validation data and can be found in Table 1. Performance of the model was measured by calculating the accuracy on the test data on predicting the binary label.

3.3 Results

The accuracy results on the test data are shown in Table 2. The performances of each model are discussed in the subsequent sections.

3.3.1 RNN

The RNN model was trained for 40,000 iterations on the Twitter data set, and achieved a test accuracy of 69%. In order to see whether the model is better at predicting positive or negative comments, the recall and precision is calculated. The RNN achieves a precision of 68.8% for positive comments and 69.1% for negative comments, and a recall of 75% and 62.4% for positive and negative comments respectively. The difference in precision is very small, whereas the recall for positive comments is significantly higher. This means that the model is slightly better at detecting positive tweets.

[1]The code can be found at: https://goo.gl/XNdcM9

3.3.2 LSTM

Like the RNN model, the LSTM was trained for 40,000 iterations on the Twitter data set and had a similar learning curve. The model achieved a test accuracy of 68% which is slightly lower than the RNN accuracy. The LSTM achieves a precision of 70.8% for positive comments and 65.2% for negative comments, and a recall of 66.5 and 69.6% for positive and negative comments respectively. These numbers show LSTM does not specialize at classifying positive or negative comments.

3.3.3 CharSCNN

After training the network for 9,000 iterations on the training data, an accuracy of 75.2% was reached on the test set. The recall for positive comments is 73%, where on negative comments the recall is 77%. Also, the precision is slightly better for the negative predictions of the model (75.4% vs 74.8%). Thus, the model is better at classifying negative comments, which might be due to the nature of negative comments, where there might be a stronger use of adjectives. We also notice that negation is treated as an obvious negative comment for the model, e.g. the hashtag "#not" is labeled as negative.

Further, we can see that the model learns to deal with Twitter hashtags quite well. Out of the total of 223 hashtags in the validation set, 199 are classified correctly, which is probably learned by the combination of character-embeddings with the convolutions.

3.4 Qualitative analysis

3.4.1 RNN

To get a more complete image of what the RNN has learnt, it helps to look at examples of misclassifications. For example, the negative comment *"I am so happy that my family actually gives a shit #not"* is classified as positive, most likely because except for the final hashtag, the sentence seems positive. When the RNN gets to the final word, it is already too confident that this is a positive example, and the irony contained in the last word is not recognized. Another example of this is *"12 days til my birthday... yay #sarcasm"*. However, because 99% of all 1152 sentences in the data set with "sarcasm" as last word is labeled negative, recognizing the pattern should not be too difficult for the model. The RNN model seems to struggle with sentences defined by their last word, or sarcasm in general. Furthermore, there are some truly inexplicable examples. The obviously positive example *"Loving the weekend. Loving family. Loving showers and gifts. Love."* was classified as negative.

3.4.2 LSTM

The LSTM suffers from the same problem as the RNN, as can be seen from the misclassified negative example *"Same two teams in the finals again. NBA is so competitive! #not"*. The sentence *"Loving the weekend. Loving family. Loving showers and gifts. Love."* was also classified as negative by the LSTM. Inspection of LSTM misclassifications which were correctly classified by the RNN did not show clear patterns.

3.4.3 CharSCNN

As stated before the model deals well with the hashtags, nevertheless the hashtag "#*irony*" is hard to correctly classify for the model (5 of the 23 misclassified hashtags). An example of this hashtag is *"Today I heard people saying what's wrong with society. Hmmm, society asking society what's wrong with society. smh #irony"*. The model predicted this comment as negative, where it was labeled positive. This is an example where it is hard to tell if a comment is positive or negative, which happens quite a lot with the hashtag *irony*. Once again, irony and sarcasm are probably hard concepts for the model to understand and are likely to lead to misclassification. Another example of a wrongly classified tweet is : *"the new green day cd sux *** i miss old green day."*[2]. The model incorrectly predicted this tweet as positive. This might be due to the occurrence of unseen or rare words in the test data, which is likely given the relatively small training set. Secondly, the model might spot some sort of double negation, because of the word *sucks* and the word *miss*. Nevertheless, for humans this sentence would be easy to classify as negative.

3.5 Comparison

The CharSCNN architecture generally performs better on this task, both for the quantitative as qualitative aspect. Compared to the RNN baseline, the CharSCNN architecture with character embeddings reaches a relative improvement of 9% on the accuracy. Furthermore, the CharSCNN manages to reach this accuracy in significantly

[2]The *** indicate explicit negative content

less iterations than the recurrent models. As described in the previous sections, the CharSCNN is more capable of learning features on the character-level. In particular, we found using the convolutional model with character embeddings highly improved the detection of hashtags. As a result, the model can make more fine-grained decisions on the sentiment of each tweet. The two recurrent models performed similar on this task, which is likely to be due to the short input strings used which do not fully exploit the capabilities of the LSTM model. Especially when performing quantitative analysis, the models seemed to struggle with similar examples.

4 Discussion

The CharSCNN architecture was originally proposed by (Nogueira dos Santos and Gatti, 2014). When comparing the performance of our model, we do not reach their state-of-the-art performance on accuracy. The two main reasons for this are their use of pretrained word embeddings, whereas our model learnt these embeddings during training. Using a large unlabeled data set helps to make the embeddings more general and therefore more likely to perform well on test data. Secondly, the data set used for both their pretraining and training was significantly larger than the Twitter data set used in our experiments.

To be able to correctly predict the sentiment of a given text, the training samples need to be correctly and non-ambiguously classified. Unfortunately, it is not always clear how to label Twitter data, since its sentiment sometimes is classified differently depending on how the sentence is interpreted. This could lead to wrongly classified tweets. Specifically for our data set we found some annotations that seemed questionable and this shows the subjectivity involved in this task, which might influence the performance of the models.

The network in its current form is able to do binary sentiment classification. However, it is not yet able to classify the text in more detail, such as "*extremely happy*", "*happy*", "*neutral*". This capability could result in a network being able to filter out hate tweets, which is something that could be very valuable in the age of social media.

4.1 Further research

To improve upon the results from our experiments, several adjustments to the architecture and training procedure could be made. As mentioned before, one can use pretrained embeddings that already have a general representation of word meanings. These embeddings can be fine-tuned during training, resulting in a potentially more general embedding which will perform better on the test data. Furthermore, the word embeddings could be trained to be sentiment specific, which should lead to a better initial understanding of sentiment (Erk and Smith, 2016).

An attention-based model, which are more suited to model dependencies between different parts of the sentence by 'paying attention' to relevant words or characters (Bahdanau et al., 2014), could also be applied to the problem of sentiment analysis. Such a model could learn which aspects of the sentence are important for classifying its sentiment and therefore make more accurate predictions.

Lastly, testing the models on different types of data could give more insight into the performance. For example, classifying movie reviews (Maas et al., 2011) or user reviews of products which contain more information require the model to handle longer sequences.

5 Conclusion

In this paper, we compared the performance of recurrent and convolutional neural networks on the sentiment analysis task for the Stanford Twitter Sentiment corpus. We found in our experiments that the convolutional model outperformed the recurrent models.

To get more insight in the results, we performed qualitative analysis. The main findings of this analysis were that in the domain of tweet classification, CharSCNN performs better partly due to its capabilities to use hashtag information while classifying tweets. Furthermore it was shown that for the Stanford Twitter Sentiment dataset, the LSTM did not perform better than the RNN, which is likely due to the fact that tweets have a maximum of 140 characters.

We recommend further research to address issues raised in this paper such as irony and sarcasm detection and classifying texts not in a binary manner but in more detail.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Erk, K. and Smith, N. A. (2016). Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1.

Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Mesnil, G., Mikolov, T., Ranzato, M., and Bengio, Y. (2014). Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews. *arXiv preprint arXiv:1412.5335*.

Nogueira dos Santos, C. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.