Twitter Sentiment Analysis using combined LSTM-CNN Models

Pedro M. Sosa

June 7, 2017

Abstract

In this paper we propose 2 neural network models: CNN-LSTM and LSTM-CNN, which aim to combine CNN and LSTM networks to do sentiment analysis on Twitter data. We provide detailed explanations of both network architecture and perform comparisons against regular CNN, LSTM, and Feed-Forward networks. Finally, we find that both our models achieve better results, with our best model (LSTM-CNN) achieving around 2.7%-8.5% better than regular models.

1 Motivation

This past decade has seen the exponential growth of social media services such as Facebook, Twitter, LinkedIn, and many others. Companies and organizations have realized that these platforms can be an invaluable source of information that allow them to interact and learn more about their customers. However, due to the sheer amount of users, posts, comments, messages, and other forms of interaction, tracking a user's overall appreciation of a brand can be incredibly complex. For example, Twitter, one of the biggest social media platforms today, has roughly 350 million active users posting around 500 million *tweets* per day. Nonetheless, Deep Learning and other Machine Learning (ML) techniques can provide an automated method to help process and extract useful data from these large amounts of information.

In this paper we will focus on performing Sentiment Analysis with Deep Learning on Twitter data. Our goal is to use Neural Networks to classify any given *tweet* into a "positive" or "negative" one. This could be very helpful for anyone who wants to quickly gauge the overall sentiment of *tweets* targeting a specific product, company, organization, or other entity.

We propose 2 different Neural Network models that aim to combine the popular Long-Short Term Memory Neural Networks (LSTMs) with Convolutional Neural Networks (CNNs), and compare their accuracy against regular CNN and LSTM networks. Furthermore, we also compare our models against our previous work [1], in which we hand-built a feed-forward network.

2 Background

In this section we will briefly elaborate on the basic model constructs of Convolutional Neural Networks and Long-Short Term Memory Recurrent Neural Networks.

2.1 Convolutional Neural Networks

Initially designed for image recognition, Convolutional Neural Networks (CNN) have become an incredibly versatile model used for a wide array of tasks. CNNs have the ability of recognizing local features inside a multi-dimensional field. For example, on an image, CNNs will be able to spot particular features such as a wheel or a smile, regardless of where these might be located.

Basic CNNs (as outlined in Figure 1) work by feeding multidimensional data (e.g. images, word embeddings, etc.) to a Convolutional layer which will be composed of multiple filters that will learning different features. Notice that these filters are sequentially applied to different sections of the input. The output is usually pooled or sub-sampled to smaller dimensions and later fed into a connected layer. [2]

The intuition behind using CNNs on text relies on the fact that text is structured and organized; As such, we can expect a CNN model to discover and learn patterns that would otherwise be lost in a feed-foward network. For example, it might be able to distinguish that using “down” in the context of “down to earth” is actually of positive sentiment as opposed to other phrases such as “feeling down”. Furthermore, it will be able to extract these features regardless of where they occur in the sentence.

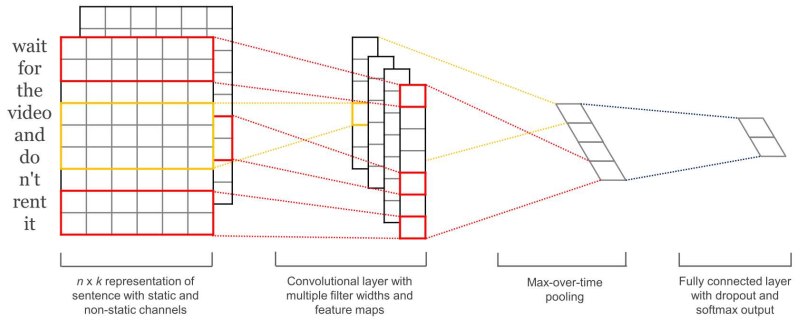
wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification

2.2 Long-Short Term Memory Neural Networks

Long-Short Term Memory (LSTM) networks are a type of Recurrent Neural Network architecture that is designed to “remember” previously read values for any given period of time. LSTMs usually contain three gates that control the flow to and from their memories. The “input gate” controls the input of new information to the memory. The “forget gate” controls how long certain values are held in memory. Finally,the “output gate” controls how much the value stored in memory affects the output activation of the block. [3]

Intuitively, the benefit of using LSTMs when doing any type of text analysis is that the network will remember what it has read previously, and thus is can have a better understanding of the input. In our case, it might be able to handle sentences with changing sentiment such as “I hated reading books, until I read Asimov”.

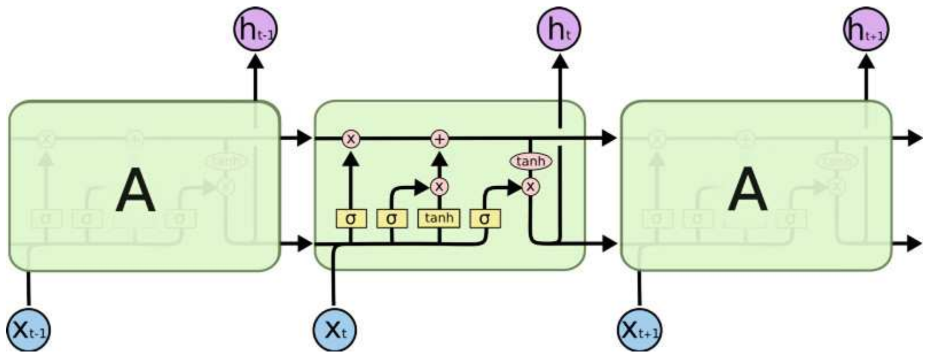

Figure 2: Unrolled LSTM network. (Source: Suriyadeepan Ramamoorth)

2.3 Twitter Dataset

The Twitter dataset used to train and validate our models is a combination of the University of Michigan Kaggle competition dataset [4] and the “Twitter Sentiment corpus” created by Neik Sanders [5]. In total, these datasets contain 1,578,627 tweets labeled as either *positive* or *negative*.

3 Models

In this section we will propose 2 models that aim combine CNN and LSTM networks.

3.1 CNN-LSTM Model

Our CNN-LSTM model (Figure 3) combination consists of an initial convolution layer which will receive word embeddings as input. Its output will then be pooled to a smaller dimension which is then fed into an LSTM layer. The intuition behind this model is that the convolution layer will extract local features and the LSTM layer will then be able to use the ordering of said features to learn about the input's text ordering. In practice, this model is not as powerful as our other LSTM-CNN model proposed.

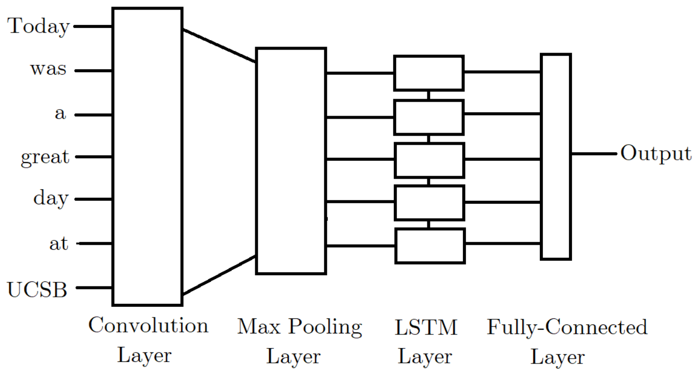

Figure 3: CNN-LSTM Model

3.2 LSTM-CNN Model

Our CNN-LSTM model (Figure 4) consists of an initial LSTM layer which will receive word embeddings for each token in the *tweet* as inputs. The intuition is that its output tokens will store information not only of the initial token, but also any previous tokens; In other words, the LSTM layer is generating a new encoding for the original input. The output of the LSTM layer is then fed into a convolution layer which we expect will extract local features. Finally the convolution layer's output will be pooled to a smaller dimension and ultimately outputted as either a positive or negative label.

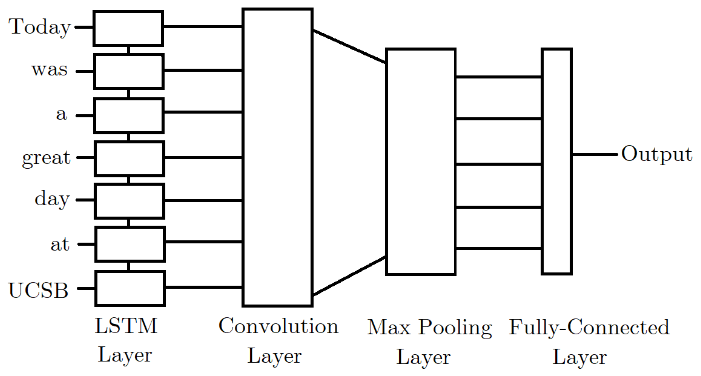

Figure 4: LSTM-CNN Model

4 Results

4.1 Experiment Setup

The results on Table 2 show the average accuracy of each network taken after 5 tests. For these test we generated training sets of 10,000 tweets and testing sets of 2,500 labeled tweets from our original dataset. These training and testing sets contained equal amount of positive and negative tweets. We trained each dataset using the parameters presented on Table 1 , and we recorded the model's accuracy when trying to label the testing set.

4.2 Parameters

The best parameters for our models were initially selected based on previous implementations of CNNs and LSTMs, and further fine-tunned through manual testing. We analyze the effects of these parameters on Section 4.4.

Embedding Dimension	32
Epoch	10
Batch Size	128
Filters	32
Kernel Size	3
Pool Size	2
Dropout	0.5
Word Embeddings	Not Pre-trained

Table 1: Parameters selected for our CNN-LSTM and LSTM-CNN models

4.3 Model Comparison

In this section we show the results of our sentiment analysis tests using different network models. Aside from testing our CNN-LSTM and LSTM-CNN models, we also compare against standard CNN and LSTM networks. Furthermore, we include the results of our previous work [1] using a hand-built feed-forward network with word embeddings or engineered feature vectors.

Neural Network Model	**Avg. Accuracy**
Feed-Forward (Word Embeddings) [1]	58.4%
Feed-Forward (Feature Vectors) [1]	66.8%
CNN	66.7%
LSTM	72.5%
CNN-LSTM	69.7%
LSTM-CNN	75.2%

Table 2: Average accuracy of different neural network models.

Result Analysis Our CNN-LSTM model achieved an accuracy of 3% higher than the CNN model, but 3.2% worse than the LSTM model. Meanwhile, our LSTM-CNN model performed 8.5% better than a CNN model and 2.7% better than an LSTM model.

These results seem to indicate that our initial intuition was correct, and that by combining CNNs and LSTMs we are able to harness both the CNN's ability in recognizing local patterns, and the LSTM's ability to harness the text's ordering. However, the ordering of the layers in our models will play a crucial role on how well they perform.

We believe that the 5.5% difference between our models is not coincidental. It seems that the initial convolutional layer of our CNN-LSTM is loosing some of the text's order / sequence information. Thus, if the order of the convolutional layer does not really give us any information, the LSTM layer will act as nothing more than just a fully connected layer. This model seems to fail to harness the full capabilities of the LSTM layer and thus does not achieve its maximum potential. In fact, it even does worse than a regular LSTM model.

On the other hand, the LSTM-CNN model seems to be the best because its initial LSTM layer seems to act as an encoder such that for every token in the input there is an output token that contains information not only of the original token, but all other previous tokens. Afterwards, the CNN layer will find local patterns using this richer representation of the original input, allowing for better accuracy.

4.4 Further Results

In this section we discuss further observations we noticed when experimenting and fine-tuning our models.

4.4.1 Learning Rates

We found that the optimal number of epochs to train our models was 10. However during our search, we realized that each model had different "learning rates". In other words, some models would learn and start over-fitting quicker than others.

As we can see in Table 3, training the LSTM and LSTM-CNN models with just a few epochs was enough to get accuracy above 60%. However, too many epochs would affect both those models substantially. The CNN and CNN-LSTM models behaved the opposite way; With few epochs, these models could not reach the 60% accuracy rate, however doing too many epochs resulted in less overfitting when compared to the LSTM and LSTM-CNN models.

Epochs	CNN	LSTM	CNN-LSTM	LSTM-CNN
3	56.5%	62.1%	58.1%	64.5%
10	66.7%	72.5%	69.7%	75.2%
20	70.7%	51.9%	68.2%	70.1%

Table 3: Average accuracy of each model when trained with different epoch sizes. All other parameters where the same as those described in Table 1.

4.4.2 Effects of Dropout

In our original implementation of both models, we included dropout layers to test if that would increase performance. On the CNN-LSTM we added a dropout layer immediately following the CNN before feeding into the LSTM. On the LSTM-CNN layer we added the dropout layer after then CNN layer, before feeding into the fully connected layer. We found that Dropout was extremely important for these models to work properly. Table 4 shows the effects of using dropout.

Dropout Prob.	CNN-LSTM	LSTM-CNN
20%	58.4%	59.1%
50%	69.7%	75.2%
98%	51.7%	48.1%

Table 4: Average accuracy of each model when trained with different dropout probability. All other parameters where the same as those described in Table 1.

4.4.3 Using Pre-trained Word Embeddings

Aside from using an embedding layer to learn the word embeddings during training, we also experimented using GloVe pre-trained word embeddings [6]. However, using these pre-trained word embeddings resulted in worse accuracy. We believe this is because of the textual irregularities that tweets present.

Since each Tweet is limited to a maximum of 140 characters, it is not unlikely to find misspellings, unconventional word abbreviations, internet-related slang, and other tokens that are unknown to the GloVe word embeddings. In addition, there are internet-specific irregularities such as emojis (☺,☹), mentions (`@konukoii`), and hyper-links (`http://www.konukoii.com`) which further complicate the sentiment analysis.

Table 5 shows the results of using GloVe pre-trained embeddings as opposed to learning them during training.

Embeddings	**CNN**	**LSTM**	**CNN-LSTM**	**LSTM-CNN**
Pre-trained (GloVe)	62.0%	64.8%	66.8%	70.3%
Not Pre-trained	66.7%	72.5%	69.7%	75.2%

Table 5: Average accuracy of each model when using GloVe pre-trained embeddings, as opposed to learning these embeddings during training. All other parameters where the same as those described in Table 1.

5 Future Work

The models that we present seem to perform better than regular CNN and LSTMs, and might be suitable for other tasks aside from sentiment analysis. It would be interesting to further apply these models for more complex tasks such as image analysis.

Furthermore, it would be beneficial to test different types of RNNs aside from LSTMs for our models. For example, using bidirectional LSTMs on the LSTM-CNN model might yield an even better result as each token fed to the CNN layer would contain information from all the other tokens in the original input.

Lastly, in this paper we abstained from using hand-engineered features as we did on a previous work [1]. However, it would be interesting to see the results of integrating the use of these feature vectors along with our models.

6 Conclusion

In this paper we have presented two models that aim to combine CNN and LSTM neural networks to achieve better performance on sentiment analysis tasks by a substantial margin. Our CNN-LSTM model does 3% better than a regular CNN model, albeit doing 3.2% worse than an LSTM model. On the other hand the LSTM-CNN model does 2.7% better than a regular LSTM model and 8.5% better than a CNN model. Furthermore, we have done an in-depth exploration of the effects of epochs, dropout, and pre-train word embedding in our models to help future researchers further understand the nuances of our models. We hope that this research may aid future development of accurate sentiment analysis tools.

Code The Tensorflow code for both the CNN-LSTM and LSTM-CNN models is open-source and freely available at `https://github.com/pmsosa/CS291K`

References

[1] P. M. Sosa and S. Sadigh, "Twitter sentiment analysis with neural networks," *Academia.edu*, 2016.

[2] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[3] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling.," in *Interspeech*, pp. 194–197, 2012.

[4] University of Michigan, "Sentiment classification dataset." `https://inclass.kaggle.com/c/si650winter11`, 2011. [Online; accessed 18-May-2017].

[5] N. Sanders, "Twitter sentiment corpus." `http://www.sananalytics.com/lab/twitter-sentiment/`, 2011. [Online; accessed 18-May-2017].

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

[7] D. Britz, "Implementing a cnn for text classification in tensorflow." `github.com/dennybritz/cnn-text-classification-tf`, 2015. [Online; accessed 18-May-2017].

[8] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining.," in *LREc*, vol. 10, 2010.

[9] B. Pang, L. Lee, *et al.*, "Opinion mining and sentiment analysis," *Foundations and Trends® in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[10] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 142–150, Association for Computational Linguistics, 2011.