Movie Reviews Sentiment Analysis using CNNs

Nisrine Zerouali
Al Akhawayn University
CSC 4399
Instructor: Dr. Asmae Mourhir

***Abstract*—Sentiment analysis, also named opinion mining, plays an important role in our daily lives. It enables companies to survey customer satisfaction, bots to moderate online forums and social scientists to study media bias. Natural language processing (NLP) is used in opinion mining tasks to be able to analyze and extract various sentiments through text sequences. On sequential data, Recurrent neural networks (RNNs) are usually the favored solution. However, researchers were able to adapt Convolutional neural networks (CNNs) to text, and achieve state of art results. In this paper, I will present the use of CNNs on movie reviews text data to perform sentiment analysis. Using the rotten tomatoes Movie Review dataset.I have trained two different convolutional neural networks on this dataset: shallow CNN and deeper CNN.**

I. Introduction

RNNs and LSTM have long been the state-of-the-art when it came to processing sequences. Their architecture takes into account the temporal nature of language. In parallel, convolutional neural networks were also having a revolution of their own in the field of computer vision. Their ability to pick up patterns in images (matrices) being their strongest point. Recent developments [1] have shown that CNNs can perform particularly well in NLP tasks such as sentiment analysis. These results are particularly interesting because they demonstrate CNNs ability to extract patterns beyond visual data. Futhermore, CNNs parallelizing abilities make them fast learners. This paper attempts to reproduce [1].

II. State Of Art

In the last couple of decades, many researchers have achieved great results in sentiment analysis tasks. The Universal language model fine-tuning (ULNFiT) by fastAI [2] is a transfer learning method applicable to any NLP task, and specifically to text classification tasks. Unlike in computer vision, many tasks in NLP still necessitate manual adjustments to train models. This state-of-art ULNFiT has made transfer learning easier in classification tasks. The authors have developed what's called "AWD LSTM language model" [3] which made a huge success in language modeling area. They have made an improvement on what has been already developed before [4].

Unlike regular RNNs, or LSTMs for sequence data, CNNs have been known to achieve state-of-art results in their first use on many NLP tasks such as sentiment analysis and tasks such semantic parsing [5], sentence modeling [6], text classification [7]. In sentence classification tasks using CNNs, Kim et al. have achieved remarkable results, and this achieved only with little hyperparameter tuning.

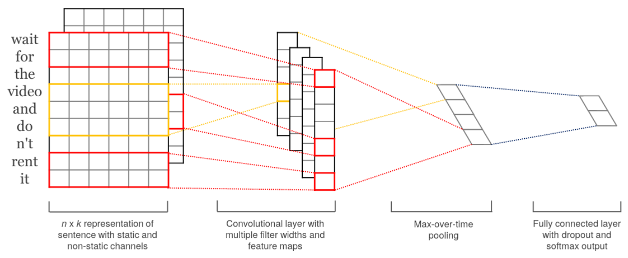

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Fig. 1: Deep CNN model architecture.

III. Methodology

Text is fed into the CNN as a matrix. Each row represents a word's embedding. This matrix is the equivalent of images fed to CNNs in tasks like image classification.

The dataset used in this work is from rotten tomatoes movie reviews. [8] For a total of 10662 reviews, with two labels representing the critical judgment: 5331 are labeled positive and 5331 negative. During training, I used two different architectures:

- Shallow CNN with one convolutional layer.
- Deep CNNs with multiple convolutional layers [7]

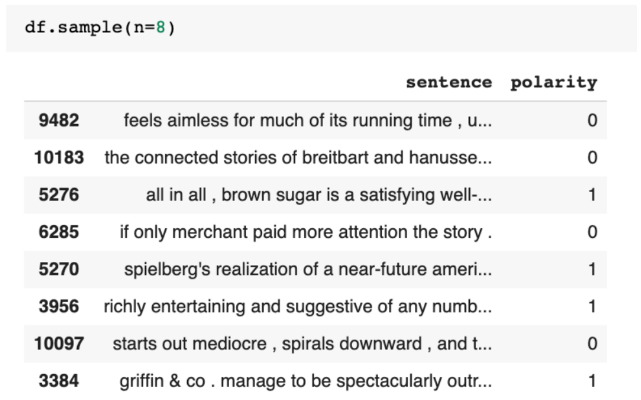
The deep CNN is the approach used in [1]. Figure 1 shows a diagram of the model.

A. Data Pre-processing

This section will describe the steps we have taken to clean and prepare the data to be fed to the model. These steps can be summarized to:

1) Tokenizing the sentences into words.
2) Removing punctuation and non-word tokens (e.g. smileys).
3) Build a look-up that maps an index to each word and itsa word embedding.
4) Convert sentences into vectors of indices.

1) Tokenization: Tokenization is the task of breaking a string into smaller parts. Usually, these parts include words and punctuation. Tokenization is not always trivial. Tokenizing the string "Thank you, sir." should give the following tokens: ["Thank", "you", ",", "sir", "."]. Whereas the string "In the near-future" should result in ["In", "the", "near-future"]. Notice how the first example split based on some punctuation wheras the second kept the dash in near-future. Sophisticated tokenizers have some built-in rules that allow them to differentiate between the two



```
df.sample(n=8)
```

	sentence	polarity
9482	feels aimless for much of its running time , u...	0
10183	the connected stories of breitbart and hanusse...	0
5276	all in all , brown sugar is a satisfying well-...	1
6285	if only merchant paid more attention the story .	0
5270	spielberg's realization of a near-future ameri...	1
3956	richly entertaining and suggestive of any numb...	1
10097	starts out mediocre , spirals downward , and t...	0
3384	griffin & co . manage to be spectacularly outr...	1

Fig. 2: A sample from the rotten tomatoes movie reviews dataset.

cases. The dataset this paper uses is fortunately already pre-processed such that all tokens are space-separated. Therefore, tokenizing is as simple as splitting strings by space.

2) Clean-up: Nadja et al. suggest that punctuation and stopwords are not relevant to the task of sentiment analysis. As such, they remove them before proceeding any further. Removing punctuation can be efficiently done using regular expressions. Removing stopwords can be done using pre-defined dictionaries that contain some of the most frequent words in the English language. However, we notice that those dictionaries contain words like ”not” are ”never” which are clearly essential in the task of sentiment analysis as they reflect negation. Such words used with a positive verb will most likely indicate negative sentiments. Removing stopwords negatively impacts performance and is therefore skipped. [9]

3) Turning sentences into indices: A vocabulary consists of all the unique words that appear in the sentences. The dataset's vocabulary contains 18,398 unique words. In this phase, we map each word to an index ranging from 1 to 18,398. 0 is a special index reserved for unknown words that will be encountered during testing. All sentences should have the same size since we will stacking them into a matrix. The number of columns in our matrix will be the size of the largest sentence. The 0th index is used to pad sentences that are shorter than that. The maximum length of the sentences is 53.

Each index is also mapped to the word embedding that represents its word. For this, we use pre-trained GloVe word embeddings. [10] We opted for 300-dimensional vectors trained on Wikipedia corpus. 95% of the 18,398 words in our vocabulary had a representation in GloVe. The rest were initialized to random vectors.

Converting the sentences into indices is very memory-efficient. This becomes clearn when looking at sentences that contain many similar words such as. Repeating words in that sentence would all map to the same embedding from our look-up, thus saving us from storing the 300-dimension vector of the embedding for each repetition. This saving accross the whole dataset can make a lot of difference.

4) Architectures: The first model is a shallow CNN composed of an embedding layer, followed by a convolutional layer with one region filter size of 3 with 512 features. The filter size could be seen as the n-grams we will consider in pattern matching. If a filter size of 3 is used, it means that the model only detects patterns in 3-grams etc. Afterwards, a 1 max-pooling layer where the operation is applied on the feature map to get a fixed size vector. A softmax layer is then applied to get the final classification output. A dropout regularization technique is also used during training of this model with a probability of 0.5, and a rectified linear unit (ReLU) activation function, and finally a dense layer with 1 activation function sigmoid.

The second model is a deeper CNN proposed by [7]. It differs from the first model by using 3 filter sizes (3, 4, 5) for the same region instead of one with 128 features for the different filter sizes. This way the model is able to learn from 3, 4 and 5-grams. It also contains an embedding layer, a 1D convolution, a max-pooling layer, a dropout of 0.5, a rectified linear unit (ReLU) activation function, and a final dense layer with a sigmoid activation function. I used Adam optimizer in both models.

The embedding layer in both models takes sentences as a vector of indices and replaces the indices with the vector embedding of the corresponding words. It is this layer that transforms our sentences into a matrix to be fed into the CNN. At each training step, a batch of such matrices are constructed.

IV. Experiments

We have trained both models by varying the following variables:

- Freezing or un-freezing the embedding layer.
- Taking full vocabulary or only part of it.

A. Freezing the embedding layer

As mentioned above, the embedding layer is the layer that converts words in sentences to embedding vectors. Freezing this layer means that the embeddings will not be trained with the model. This means faster training. However, training this layer with a small learning rate can be beneficial. As discussed in [7], pre-trained word embeddings like GloVe have very general definitions of closeness for words. For example, the words ”good” and ”bad” in pre-trained embeddings are quite close. This is probably because they are grammatically similar (adjectives). However, for sentiment analysis, these words have opposite meanings.

B. Reducing vocabulary

A vocabulary includes all the unique words that appear in the dataset. However, some words such as movie and actor names appear only a few times. These can introduce some noise. Therefore discarding a number of least-frequent words from the vocabulary can help get rid of that noise. In our discussion, we try reducing the number of words to only the 14,000 most common words. The rest of the words are marked as unknown (assigned a 0 index).

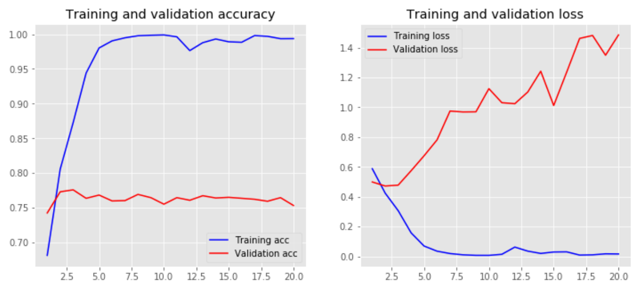


Fig. 3: Deep CNN model architecture.

V. Results and Discussion

The table below shows the accuracy of the different models we have trained.

Validation accuracy		
	Shallow CNN	Deep CNN
Frozen embeddings and full vocab	75%	77%
Trainable embeddings and full vocab	77.4%	78.1%
Trainable embeddings and reduced vocab	84.2%	84.62%

TABLE I: Validation accuracy)

We notice that making the embeddings trainable increases the performance. This goes in hand with what is discussed in [7] with regards to fine-tuning embeddings on the dataset used. This fine-tuning can make subtle but very important changes to the embeddings that fits the problem at hand e.g. making the words "good" and "bad" further apart in the vector space [7].
The big surprise comes with the effect of reducing the vocab. It seems that reducing the vocabulary to only the 14,000 most frequent words caused more than 6% increase in accuracy. This is something that was not used in the original paper but that we found yields comparable results. This can be explained by the fact that the least frequent words are those referring to specific actors or movies that have no impact on the sentiment of the review.
Surprisingly, models only required 3 epochs to converge as is shown in figure 3. In this example, we trained it on 20 epochs. After finding the ideal number of epoch, we used early-stopping to avoid overfitting.
In figure 3, we notice that although the validation accuracy stagnates after the third epoch, the validation loss increases drastically. This means that the model might be overfitting.

A. *Interpreting mistakes*

The table below shows sentences that the model got wrong. Meaning either sentences that were positive but were flagged negative or vice-versa.

Sentence	Predicted	Actual
although it lacks the detail of the book the film does pack some serious suspense	N	P
a depressing confirmation of everything those of us who don't object to the description unelected have suspected all along george w bush is an incurious uncharismatic overgrown frat boy with a mean streak a mile wide	N	P
only two words will tell you what you know when deciding to see it anthony. hopkins.	N	P
if you are into splatter movies then you will probably have a reasonably good time with the salton sea	P	N

TABLE II: Wrongly classified sentences. (P=positive, N=negative)

This is a selection of some of the interesting examples of mistakes.
The first sentence is an example of a nuanced opinion. Not all opinions fall perfectly into negative or positive. In that one, the user brings up a weakness of the movie and ends up on a good note. The model predicted it is a negative probably because it contains the 3-gram "lacks the detail" which is quite negative.
The second sentence is an example of a review that talks negatively about a character of the movie. In this case, it talks negatively about former US president George Bush which was the subject of the documentary. The overwhelming use of negative words biased the model into predicting it as negative.
The third sentence is an example of a reference that the model has missed. In here, the user is referring to Emmy Awards-winning actor Anthony Hopkins. Since we strip the sentences from least commonly used words, the model probably didn't even get the tokens "Anthony" and "Hopkins". Even if it did, it wouldn't relate to the common knowledge that Anthony Hopkins is a good actor.
The fourth sentence marks a difficult problem to solve: sarcasm detection. The reviewer is making use of sarcasm to depict a negative review. In the eyes of the model, the sentence uses many positive words such as "good time".

VI. Conclusions

This paper reproduces results from [1]. CNNs in NLP-related tasks is surprisingly high. This proves that there is no limit to creativity in the field of deep learning. We have found that the performance of the model can be significantly improved by removing some of the least-used words from the vocabulary. Surprisingly, the common practice of removing stopwords also turned out to be detrimental to the performance of the model. [9] Not so surprisingly, using filters of many sizes enabled the model to pick up on more patterns and therefore achieve higher accuracy. From the mistakes analyzed, it seems that the model still struggles with references to famous actors/movies, attacks on the topics/characters discussed as opposed to the quality of the movie and the use of sarcasm. The latter has been an area of heavy research [11, 12].

References

[1] N. Nedjah, I. Santos, and L. de Macedo Mourelle, “Sentiment analysis using convolutional neural network via word embeddings,” *Evolutionary Intelligence*, Apr 2019.

[2]

[3] Salesforce, “salesforce/awd-lstm-lm,” Jun 2018.

[4] Dai, A. M., and Q. V., “Semi-supervised sequence learning,” Nov 2015.

[5]

[6] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences.”

[7] Y. Kim, “Convolutional neural networks for sentence classification,” 2014.

[8] B. Pang and L. Lee, “Seeing stars,” *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL 05*, p. 115–124, 2005.

[9] H. Saif, M. Fernandez, and Y. He, “On stopwords, filtering and data sparsity for sentiment analysis of twitter,” *Knowledge Media Institute*.

[10] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *In EMNLP*, 2014.

[11] D. Hazarika, S. Poria, S. Gorantla, E. Cambria, R. Zimmermann, and R. Mihalcea, “Cascade: Contextual sarcasm detection in online discussion forums,” 2018.

[12] S. Castro, D. Hazarika, V. Pérez-Rosas, R. Zimmermann, R. Mihalcea, and S. Poria, “Towards multimodal sarcasm detection (an obviously perfect paper),” 2019.