Evaluation of different Language Models for movie review Sentiment Classification

Gerard Planella
University of Amsterdam /
Science Park, Amsterdam
gerard.planella.fontanillas
@student.uva.nl

Chrysoula Pozrikidou
University of Amsterdam /
Science Park, Amsterdam
chrysa.pozrikidou
@student.uva.nl

Abstract

Sentiment analysis is a field of natural language processing (NLP) that has increased in popularity the past few years. With the rising popularity of social media platforms and the need to analyse large amounts of text data for various applications, finding efficient techniques for sentiment analysis is of utmost importance. In this article we discuss various techniques that have been proposed for this task and compare their performance on a given dataset. We rose questions about the importance of word order and context in sentiment classification and provided reasoning about our conclusions. We additionally investigated the impact that the text length and the inherent tree structure of a sentence, had on the implementation of different models. In orderto construct systems that can process text and accurately perform sentiment classification, further research must be conducted based on the grounds of the progress that has been already made.

1 Introduction

In this article, we analyse the performance of different models on the task of sentiment classification for movie reviews and how these different models can benefit from differences in sentence structure like sentence length and word order. Sentiment classification can prove to be very complex due to the difficulties present in our natural languages such as dealing with negations, sarcasm, and data biases among many other issues. However, several techniques have been proposed in order to attack these problems, some of which are discussed and implemented in this project.

To properly evaluate the different models which will be described in section 3 we define a series of experiments that will aid us in better understanding the strengths and weaknesses of the different models.

We start by performing a comparison between sparse and dense word embeddings, where the latter revolutionised the field of NLP (Collobert et al., 2011), and how a more generalised vector representation of our word tokens can affect the performance of our models for our specific task. Different research has been done for fields like Neural Machine Translation (NMT) (Qi et al., 2018) but we will solely focus on sentiment classification. We will use two types of dense word embeddings throughout the different experiments, namely GloVe (Pennington et al., 2014) and Word2Vec (Le and Mikolov, 2013).

With our different word representations defined, we will look into the effect of how word order can affect the different implemented models [(Abdou et al., 2022), (Kraus and Feuerriegel, 2019)] and evaluate whether simple count-based models which do not take word order into account such as the traditional Bag-of-Words model (Harris, 1954) could outperform more modern models that do take word order into consideration (Le and Mikolov, 2013). Other sentence structure variables will also be considered such as sentence length [(Gezici et al., 2012), (Pambudi and Suprapto, 2021)] and taking advantage of the implicit tree structure in our language, which motivated the development of more specialised language models like the ones implemented in (Tai et al., 2015), (Le and Zuidema, 2015) and (Zhu et al., 2017). Nevertheless, for our sentiment classification task, we will evaluate whether a more granulated analysis of our sentences using node-level sentiment scores can improve the performance of some of our models.

In addition, we will also perform a comparison of the different language models implemented by Tai et al. (Tai et al., 2015) and Le et al. (Le and Zuidema, 2015) which has not been previously addressed for our specific task.

Before starting these experiments, we had very concrete expectations about obtaining the best performance with more complex neural models on

short sentences that take into account word order and tree structures, while also assuming that pre-trained word embeddings would render the best results. Nevertheless, it was useful to analyse the results of the different implementations to enforce and disprove some of our expectations.

2 Background

This section describes the main techniques for sentiment classification used in this article, covering the basic notions required to understand each of them while linking these to the proposed questions stated in the introductory section 1.

Word Embeddings: We used three different types of word embeddings: trained sparse word embeddings, Word2Vec dense word embeddings and GloVe word embeddings. To obtain our sparse representations we built a vocabulary from our training set, logging the individual words with their respective frequencies and reserving the first two indices for the unknown word token <unk> and pad token <pad>. With our vocabulary built, we trained our embeddings through the use of a trainable embedding layer as the input to our BoW, CBoW and DeepCBoW models.
As for the pre-trained word embeddings, we used GloVe (Pennington et al., 2014) and Word2Vec (Le and Mikolov, 2013). These both learn vector encodings from words leveraging from their co-occurrence information taken from large text corpora. The main difference between these two embeddings is that Word2Vec is a predictive skip-gram model whereas GloVe is a count-based model.

BoW: The BoW model, first introduced in (Harris, 1954) is a simple model that only considers word frequency. This model does not take word order into account which allowed us to analyse what effect, avoiding word order has on performance.

Continuous BoW (CBoW): This model builds up on the BoW model by allowing us to define embeddings of any arbitrary dimension, which in turn allows us to choose the dimension needed to represent the words. We should not confuse this with the Word2Vec CBOW model. Again, this model does not take word order into account but allows us to obtain better-trained embeddings.

Deep Continuous BoW (DeepCBoW): DeepCBoW tries to improve the performance of the CBOW model by making it deeper and non-linear, adding linear layers and non-linear activation functions. This model enables learning more complex functions that can improve performance compared to the CBoW model. Similar work has been performed by (Liu, 2020) with CBoW on sentiment analysis. Nevertheless, their models focused on a CNN-based implementation of this model.

Long Short-Term Memory Network (LSTM): This model is the first one we used that takes word order into account. We studied two implementations of the LSTM, one proposed in (Tai et al., 2015) and another one proposed by (Le and Zuidema, 2015). The main difference between these two implementations is that Le et al. included the previous cell state in the calculations of the input and forget gates, while their output gate includes the calculated current cell state.

Tree-LSTM: Similar to LSTMs, this model also takes word order into account. Moreover, it allows us to use the tree-structure of our data. We compare two implementations proposed by (Tai et al., 2015) and (Le and Zuidema, 2015) which also differ in the same way as their standard LSTM implementations. We should note that a third proposal for this model was introduced by (Zhu and Guo, 2017) but it will not be covered in our experiments.

3 Models

This section describes the implementation-specific details of the utilised models mentioned in section 2.

For consistency, we chose to use the cross entropy loss between the actual sentiment of the sentences and the predicted ones as a loss function for all models.We used an Adam optimizer for the LSTM and Tree-LSTM implementations, and a Stochastic Gradient Descent (SGD) optimizer for the BoW, CBoW and DeepCBoW implementations. This configuration remained constant for all the experiments. As for the non-linearities, all models used the hyoerbolic tangeant activation function when needed. No learning rate scheduling was used.

In order to prepare the data for training, we converted the sequences of word tokens in our sentences to sequences of IDs. This was performed by using the Vocabulary we generated either with the dense word embeddings or the sparse word embeddings as a look-up table. To prepare the data for the

LSTM model we implemented an additional mini-batch functionality, which instead of feeding the data sequentially to the model, allowed us to feed it mini-batches of data (the default mini-batch size used is 25). Finally, to prepare the data for the Tree LSTM models, we used their transition sequences. To adress the problems of transition sequences with different lengths and of unseen words in the test set, we introduced the <pad> and <unk> tokens.

4 Experiments

This section provides the defined setup, the hyper-parameters used for the experiments as well as a brief explanation of all the experiments performed. Each of the experiment results and explanations are discussed in section 5.

Dataset The dataset we used for our experiments is the Stanford Sentiment Treebank (Socher et al., 2013), which consists of 11,855 sentences and their overall sentiment score. Each sentence corresponds to a binary tree structure and includes a fine-grained sentiment score for each node in the binary tree. Each score belongs to one of the 5 categories: very negative, negative, neutral, positive, very positive, and is represented by a number in the range of 0 to 5. For this project, we used 8,544 sentences for training, 1,101 for development and 2,210 for testing. The average length of each sentence is 19.1 words.

Learning Rates and Number of Epochs We used a learning rate of 3e-4 for the LSTM implementations and 2e-4 for the mini-batched versions of LSTMs and for the Tree-LSTM. A learning rate of 5e-4 was used for the BoW, CBoW and Deep-CBoW implementations. All models were trained for 30,000 epochs.

Seeds We ran each experiment three times with seeds 42, 112, and 16. The results shown in section 5 will show the mean and standard deviations of these three runs.

Software Repository For reproducibility you can refer to the repository used under the folder Assignment2, the Jupyter notebook found was run in a Google Colaboratory environment.

Experiments We will now describe the different tests performed:

1. Effect of word order on sentiment classification: For this experiment, we will compare models that take word order into account and models that don't.
2. Sparse vs Dense embeddings: We compare the performance of models trained on our sparse, task-specific embeddings and the pre-trained word embeddings.
3. Pre-Trained word embedding initialisation: For all experiments the <pad> and <unk> tokens were initialised to 0s. For this experiment, we evaluated how would random initialisation of these affect our DeepCBoW model.
4. GloVe vs Word2Vec: A performance comparison between these two pre-trained embedding strategies will be performed for each model.
5. Comparison between different implementations of the LSTM and Tree-LSTM models to evaluate performance.
6. Node Level Sentiment: The Stanford sentiment treebank shows us node-level sentiment values. We investigated whether the Tree-LSTM models would benefit from this implementation. For this, we generated all possible coherent subtrees (setting a minimum tree height of 3) of our training data and evaluated our Tree-LSTM with this extended dataset.
7. Sentence Length: We split the sentences into short (less than 19 words) sentences and long sentences (more than 19 words). We then compared the performance of the different models on this data.

5 Results and Analysis

In this section, we present and discuss our findings.

As a general trend that can be observed by looking at the tables 1,2 and 3, the models that incorporated take word order into account outperformed the models that operate in a bag of words manner. This was expected since usually in natural languages the words used in a sentence are not independent of each other and a lot of context is captured in the order they are used. Thus, the models that follow the LSTM architecture were able to capture more meaning in the data and outperform the models that follow a BoW architecture. However, we must note that among these models the continuous BoW model performed best. This behavior was expected, since the bigger dimensionality of the embeddings allows the model to learn

more aspects of each word. Finally, our attempt to increase the performance of the CBoW model by making it deeper and non-linear did not yield any increase in performance.

Since the Stanford Sentiment Treebank is a small dataset, we boosted the learning ability of our models by incorporating pre-trained embeddings. This increase in performance can be observed by looking at the tables 2 and 3, by comparing the performance of the Deep CBoW model when only sparse embeddings are used and when we also included the pre-trained ones.

In order to incorporate the pre-trained word embeddings, we introduced the <unk> and <pad> tokens. By initialising any of the two tokens to the 0-vector will cause the model to ignore the given token when making predictions, treating it as neutral or unimportant. On the contrary, by initialising any of the tokens to a random vector, will cause the model to treat it as a unique word. In our experiments, initialising both tokens to the 0-vector and thus treating both tokens as neutral, yielded the best results as it is observed in table 8. This might be the case because the models do not know the sentiment of the unknown words and the padding and thus, it is better to treat them as neutral.

Comparing the Word2Vec and GloVe pre-trained embeddings, we see that using Word2Vec performed better except for the Le et al implementation of LSTM and mini-batched LSTM and for both implementations of tree LSTM, as can be seen in tables 2 and 3. In the case of standard LSTMs, GloVe might have yielded better performance because it is based on global word-to-word co-occurrence whereas Word2Vec leverages co-occurrence within local contexts. The Le et al implementation incorporates the previous cell state to the input and output gates of the LSTM cells and thus it could benefit from a more generalised representation of word embeddings.

Comparing the suggested LSTM implementations by Tai et al and Le et al. and observing the tables 2 and 3, we notice that the standard implementation of LSTM by Le et al yields better results. However, that is not the case with mini-batched LSTM and Tree LSTM where the Tai et al implementation gave the best performance. This might be the result of the incorporation of the previous cell state suggested by the Le et al that might be beneficial in capturing long-term dependencies between words and result in better control of the flow of information through the LSTM.

By our experimentation with the use of sub-trees in the Tree LSTM models using the pre-trained word embeddings from Word2Vec, we observe in table 7 that the use of sub-trees did not improve performance as the models did not generalise well. This might be the case because the sentiment of the sentence reflected in a sub-tree might have a different sentiment of the initial tree and thus, this might induce confusion to our model on how to perform sentiment classification.

Finally, by comparing the performance of the models when tested only on small-length reviews and long-length reviews we observe in tables 5, 4 and 6, that all models have performed better when tested only on small reviews with the sole exception of the simple BoW model. In larger sentences, there are more long-term dependencies in between words and also more context to be captured. Thus, it is easier for our models to capture sentiment in smaller sentences. However, the BoW model does not incorporate the word order of the sentences and thus, it might perform better on longer sentences in which there will be probably more words that are non-neutral and are helpful in sentiment extraction.

6 Conclusion

In this section, we provide a conclusion for this article and recommend possibilities for further research.

It is safe to conclude that in the task of sentiment classification, taking word order into account increases performance, as we expected. However, when experimenting with the BoW approach based models, it is surprising that introducing more layers and non-linearity to the CBoW model made the model perform worse. Additionally, the fact that the models perform better on shorter sentences matched our expectations and previous findings concerning the captioning of meaning in text. On the contrary, we expected that the use of sub-trees in the tree LSTM models will boost performance since it is a form of data augmentation, but it proved to do the opposite as described in section (sec). Finally, the different implementations of the LSTM and tree-LSTM models did not produce any significant differences in our results.

Future research should consider the potential effects of incorporating sub-trees in Tree LSTMs by developing a different implementation of how to extract them. Additionally, further experimenta-

tion with different hyperparameter settings might prove important. Finally, implementation of the same models presented in this article using different datasets could help us to further explore their strengths and limitations for the pre-defined task of sentiment classification.

References

Mostafa Abdou, Vinit Ravishankar, Artur Kulmizev, and Anders Søgaard. 2022. Word order does matter (and shuffled language models know it).

Ronan Collobert, Jason Weston, Jweston@google Com, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch.

Gizem Gezici, Berrin Yanikoglu, Dilek Tapucu, and Yücel Saygın. 2012. New features for sentiment analysis: Do sentences matter?

Zellig S. Harris. 1954. Distributional structure. 10:146–162.

Mathias Kraus and Stefan Feuerriegel. 2019. Sentiment analysis based on rhetorical structure theory:learning deep neural networks from discourse trees. *Expert Systems with Applications*, 118:65–79.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory.

Quoc Le and Tomas Mikolov. 2013. Distributed representations of sentences and documents.

Bing Liu. 2020. Text sentiment analysis based on cbow model and deep learning in big data environment. *Journal of Ambient Intelligence and Humanized Computing*, 11:451–458.

Agung Pambudi and Suprapto Suprapto. 2021. Effect of sentence length in sentiment analysis using support vector machine and convolutional neural network method. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, 15:21.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation.

Ye Qi, Devendra Singh Sachan, Matthieu Felix, Sarguna Janani Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation?

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks.

Qile Zhu, Yanjun Li, and Xiaolin Li. 2017. Character sequence-to-sequence model with global attention for universal morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 85–89, Vancouver. Association for Computational Linguistics.

Xiaodan Zhu and Hongyu Guo. 2017. Long short-term memory over recursive structures.

A Appendix

	training		**dev**		**test**	
model	mean	std	mean	std	mean	std
BOW	0.323	0.004	0.279	0.015	0.251	0.023
CBOW	0.523	0.165	0.347	0.012	0.347	0.044
DeepCBOW	0.540	0.096	0.354	0.041	0.343	0.044

Table 1: This table contains tests performed on various models using sparse word embeddings.

	training		**dev**		**test**	
	mean	std	mean	std	mean	std
DeepCBOW	0.4247	0.0454	0.4124	0.0079	0.4472	0.0198
LSTM	0.4943	0.0697	0.4375	0.0375	0.4589	0.0433
Mini-Batched LSTM	0.5562	0.0189	0.4708	0.0091	0.5057	0.0169
Le et al. LSTM	0.5275	0.0475	0.4396	0.0464	0.4773	0.0409
Mini-Batched Le et al. LSTM	0.6042	0.0733	0.4605	0.0068	0.4768	0.0360
Tree LSTM	0.5811	0.0104	0.4668	0.0051	0.4999	0.0046
Le et al. Tree LSTM	0.5669	0.0333	0.4732	0.0057	0.4973	0.0141

Table 2: This table includes the runs for our different models using pre-trained GloVe embeddings.

	training		**dev**		**test**	
	mean	std	mean	std	mean	std
DeepCBOW	0.4677	0.0101	0.4344	0.0037	0.4709	0.0056
LSTM	0.4842	0.0048	0.4535	0.0052	0.4748	0.0056
Mini-Batched LSTM	0.4939	0.0112	0.4638	0.0066	0.4832	0.0063
Le et al. LSTM	0.5324	0.0535	0.4638	0.0010	0.4759	0.0123
Mini-Batched Le et al. LSTM	0.5199	0.0228	0.4608	0.0019	0.4756	0.0100
Tree LSTM	0.5118	0.0475	0.4559	0.0010	0.4857	0.0131
Le et al. Tree LSTM	0.5089	0.0291	0.4475	0.0014	0.4834	0.0063

Table 3: This table includes the runs for our different models using pre-trained Word2Vec embeddings.

	Short Sentences		**Long Sentences**	
	mean	std	mean	std
BOW	0.2509	0.0227	0.2652	0.0460
CBOW	0.3473	0.0441	0.3182	0.0400
DeepCBOW	0.3428	0.0441	0.3211	0.0661

Table 4: Accuracy results using sparse word embeddings with short and long sentences.

	Short Sentences		**Long Sentences**	
	mean	std	mean	std
DeepCBOW	0.4472	0.0056	0.3847	0.0089
LSTM	0.4589	0.0056	0.3949	0.0466
Mini-Batched LSTM	0.4796	0.0035	0.3866	0.0577
Le et al. LSTM	0.5035	0.0090	0.4452	0.0111
Mini-Batched Le et al. LSTM	0.4965	0.0051	0.4389	0.0153
Tree LSTM	0.4996	0.0123	0.4357	0.0142
Le et al. Tree LSTM	0.4987	0.0063	0.4383	0.0043

Table 5: Accuracy results using GloVe word embeddings with short and long sentences.

	Short Sentences		**Long Sentences**	
	mean	std	mean	std
DeepCBOW	0.4709	0.0055	0.4014	0.0086
LSTM	0.4748	0.0055	0.4241	0.0126
Mini-Batched LSTM	0.4815	0.0034	0.4231	0.0108
Le et al. LSTM	0.4684	0.009	0.4159	0.0312
Mini-Batched Le et al. LSTM	0.4848	0.005	0.429	0.0122
Tree LSTM	0.497	0.0122	0.4211	0.0055
Le et al. Tree LSTM	0.4834	0.0063	0.4376	0.0088

Table 6: Accuracy results using Word2Vec word embeddings with short and long sentences.

	training		**dev**		**test**	
	mean	std	mean	std	mean	std
TreeLSTM	0.5629	0.0024	0.4575	0.0005	0.4752	0.0113
Le et al. Tree LSTM	0.5660	0.0048	0.4659	0.0042	0.4684	0.0130

Table 7: This table includes the tests using subtrees of minimum height 3 on the training set. The used pretrained word embeddings are from Word2Vec.

<unk>, <pad>	0, 0		rand, rand		rand, 0		0, rand	
values	mean	std	mean	std	mean	std	mean	std
Word2Vec	0.4528	0.0257	0.4366	0.0164	0.3816	0.0828	0.4369	0.0119
GloVe	0.4653	0.0079	0.4424	0.004	0.3202	0.0968	0.3868	0.087

Table 8: Accuracy values with different pad and unknown word token initialisations for DeepCBoW using GloVe and Word2Vec embeddings.