A Comparative Review of Language Models for Text-Classification

Elias Dubbeldam
University of Amsterdam
e.f.dubbeldam@student.uva.nl

Luc Vermeer
University of Amsterdam
luc.vermeer@student.uva.nl

1 Introduction

With the recent release of ChatGPT, it is clear that large language models will have a large impact on society. One of the tasks of such models is to classify the sentiment of textual data. The performance of text classification is dependent on its ability and flexibility to incorporate syntaxes, such as word order and syntactic hierarchy, and additional information, such as pre-trained word embeddings (Qi et al., 2018). If models can incorporate this information, their performance can be increased with the same data or computational expenses. Designing data- and computation-efficient models is useful, as the size of these large language models is increasing (Brants et al., 2007).

To measure the impact of incorporating syntax and pre-trained word embeddings, we analyze multiple established text-classification models. These models fall into two categories: bag-of-words (BOW) models (Qader et al., 2019) and Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). By analyzing the accuracies of these trained models, we assess the importance of the ability to incorporate syntax and pre-trained word embeddings. We will do this by analyzing four aspects in detail.

First, acquiring train data for sentiment classification can be a tedious task, as it requires fine-grained manual annotation. To improve the data efficiency and performance over small datasets, pre-trained word embeddings can be used (Mikolov et al., 2013b). The quality, size and overlap compared with the words of the dataset influence its usefulness. To assess this, we will compare the established word embeddings GloVe (Pennington et al., 2014) and Word2Vec (Mikolov et al., 2013a).

Second, we assess the importance of word order. BOW models are order-insensitive (Qader et al., 2019), while recurrent neural networks (RNNs) such as LSTM networks are order-sensitive. They are thus able to capture long-term dependencies in the text data and make them more effective for sentiment analysis tasks (Hochreiter and Schmidhuber, 1997). We will investigate the impact of word order on the performance of these models and compare the results. Barry (2017) found that models that are order-sensitive do indeed perform better on the sentiment analysis task.

Third, we will explore how the performance of these models depends on the length of the sentences being classified. This has been found to have an effect in the sentiment analysis task by Pambudi and Suprapto (2021), although we will be exploring its effect on different models, as some models may perform better on longer sentences because of their ability to capture more context and dependencies in the text data. We will compare the performances on a variety of sentence lengths and determine whether there are any clear trends.

Finally, Tree-LSTM networks have been proposed as a way to incorporate a syntactic hierarchy into the LSTM network (Tai et al., 2015). We will investigate the impact of the tree structure on the performance of the sentiment classification task and the potential benefits of supervising the sentiment at each node in the tree for sentiment classification.

As our main contributions, we found that word order-sensitive models can capture long-term dependencies in the text data and outperform traditional models on sentiment classification such as BOW and CBOW. We also found that incorporating the tree structure of constituency trees can improve the performance, and that sentiment supervision at each node in the tree can provide additional benefits. Our results suggest that LSTMs and Tree-LSTMs are effective for sentiment analysis, and incorporating additional context and structure into the models can improve their performance.

2 Background

One of the earliest and simplest models used for text classification is the BOW model, which represents text data as a bag of individual words, ignoring the order and structure of the input sequences (Qader et al., 2019). The continuous bag-of-words (CBOW) model is an extension that allows for embeddings of arbitrary size to learn more aspects of each word (Fenn and Veren, 1991). The deep continuous bag-of-words (Deep CBOW) model uses deep neural networks to learn even more aspects of each word to improve the performance of the CBOW model.

As the experiments in this paper were conducted on a small dataset, pre-trained word embeddings GloVe and Word2vec were used (Pennington et al., 2014; Mikolov et al., 2013a). These are commonly used as a starting point for learning word embeddings for many natural language processing tasks. GloVe (Global Vectors for Word Representation) is a word-embedding model in which the vector representation of each word is learned by solving a least-squares optimization problem that minimizes the differences between the dot product of the vectors and the logarithm of the co-occurrence counts (Pennington et al., 2014). Word2vec is a word-embedding model that uses a shallow neural network with a single hidden layer to learn the vector representation of words. The model is trained on a large corpus of text data by predicting the surrounding context words (Mikolov et al., 2013a).

LSTMs are a type of RNN that are able to solve the *vanishing gradient problem* (Hochreiter, 1991), i.e. the gradient signal that is propagated back through the network during training decays exponentially as it moves through the time steps, impairing the ability to learn long-term dependencies in data sequences. It solves this by introducing a number of gating mechanisms allowing the cell to remember information and preserve hidden states over time. This allows LSTMs to effectively model complex, long-term dependencies in sequential data, such as text and speech (Hochreiter and Schmidhuber, 1997). Tree-LSTM networks are a variant of LSTM networks that incorporate a tree structure into the LSTM architecture, allowing Tree-LSTMs to capture both the sequential and hierarchical relationships in data such as constituency trees, potentially improving the performance on sentiment analysis (Tai et al., 2015).

3 Models

3.1 BOW

The BOW model associates each word with a multi-dimensional vector expressing the sentiment a word conveys, each column representing one of the output classes. Using this vector the model classifies sentiment by analyzing the words in a given piece of text. The overall sentiment of the text is determined by summing the individual sentiment scores of each word, losing word-order information in the process, and then finding the maximal value to determine the most probable class.

The CBOW model works in a similar fashion but the word embeddings can be of arbitrary size, allowing it to convey more information about each word. The vector is still summed but later projected with a learned parameter matrix to get a vector with the desired number of classes. To learn the parameter the output is then assessed using the cross-entropy loss to calculate the error of the linear layer for backpropagation.

The Deep CBOW model adds two fully connected layers to add complexity and non-linearity in an attempt to convey more aspects of each word.

3.2 LSTM

There are three main gates in the LSTM cell: the input gate, the forget gate, and the output gate. The input gate determines which parts of the input are relevant to the current state of the cell and allows only those parts to pass through to the cell state. The forget gate determines which parts of the previous cell state are relevant to the current state and allows only those parts to pass through to the current cell state. The output gate determines which parts of the cell state are relevant to the current output and allows only those parts to pass through to the output (Hochreiter and Schmidhuber, 1997). Together, these gates enable the LSTM cell to selectively incorporate new information, which allows the LSTM network to learn long-term dependencies in the data. The workings of these gates are discussed in more depth in Appendix B.

When a sequence passes through these gates it produces a hidden state which, after the whole sequence is passed through, is projected to the output dimension using a linear output layer. The cross-entropy loss is then used to calculate the gradients to update the parameter matrices.

We used a Binary-Tree-LSTM to incorporate the tree structures in the Stanford Sentiment Treebank

(Tai et al., 2015; Socher et al., 2013). The general mechanism of the gates is similar to the regular LSTM except we have a separate forget gate for each child node. For any node j, we represent the hidden state and the memory cell of its k^{th} child as h_{jk} and c_{jk} respectively. The separate parameter matrices for each child k allow the Tree-LSTM to learn more fine-grained conditioning on the separate states of a unit's children. The differences between the regular LSTM and the Tree-LSTM are discussed in more depth in Appendix B

4 Experiments

We trained all models discussed above, on the Stanford Sentiment Treebank dataset (Socher et al., 2013) with a predefined training and test set. We differentiate between five classes, ranging from very negative to very positive sentiments. As described above, we use pre-trained word-embeddings Word2Vec and GloVe for the Deep CBOW and LSTM variants.

We used an embedding size of 300 for all models (except the traditional BOW which only accepts the same dimensionality as the number of classes). The number of hidden layers for Deep CBOW, LSTM and Tree-LSTM have been set to, 100, 168 and 150, respectively. We used the Adam optimizer with an initial learning rate and a number of epochs of 5×10^{-4} and 4×10^{4} for BOW models, respectively, and 5×10^{-4} and 3×10^{4} for LSTM models, which led to convergence for all runs. We run each model with three different seeds to get consistent results for reproducibility.

We evaluate and report the test accuracy for all models. We use accuracy over other evaluation metrics because the datasets are reasonably balanced, and accuracy is an insightful metric.

After training, we evaluate the models in some extra ways to give insightful results to answer the results questions. First, we shuffle the word order of the test set. Second, we calculate the accuracy per sentence length to analyze syntax in more depth. Third, we enrich the dataset by adding all the subtrees and its sentiments. For each evaluated subtree, we keep track if this subtree was part of a sentence that was predicted correctly, and its normalized height. The normalized height is defined as the height of the subtree divided by the height of the tree of the full sentence. By doing this, we can do a qualitative analysis of what node in the tree the model fails when making a wrong prediction.

5 Results and Analysis

The test accuracy of the models is shown in Table 1. From these results, we can already highlight some key differences between the performance of the models.

First, more complex BOW models increase the performance on the classification task. Deep CBOW performs better than CBOW, which performs better than BOW. This shows that a more complex model is desired over a simple BOW model.

Second, it is important which pre-trained word embeddings are used. Table 1 shows that GloVe consistently outperforms Word2Vec. This could be because the GloVe vocabulary has a larger overlap with the Stanford Sentiment Treebank dataset, compared to Word2Vec. This is discussed in more detail in Appendix A.

Third, taking word order into account improves performance. This is clear from two patterns. All LSTM variants outperform BOW variants. Furthermore, the performance of BOW variants does not decrease when the words are shuffled, but it does for LSTM models. This is expected, as LSTM variants do take word order into account, while BOW variants do not.

Finally, taking more syntactic structure into account improves the performance. Tree-LSTM models perform better than traditional LSTM models, although not significant. Furthermore, increasing the number of data samples by adding all subtrees as roots to the train set improves further the final performance.

Table 1: Test accuracy of various models on the Stanford Sentiment Treebank dataset, averaged over three seeds. Note that the Node Tree-LSTM is trained on all subtrees of the train data, but the test set only consists of the full sentences.

Word order:	**Normal**		**Shuffling**	
Model	mean	std	mean	std
BOW	0.285	0.00107	0.287	0.00128
CBOW	0.359	0.00996	0.344	0.01605
Deep CBOW	0.374	0.01180	0.361	0.00795
Pretrained Deep CBOW (Word2Vec)	0.355	0.04844	0.351	0.04567
Pretrained Deep CBOW (GloVe)	0.436	0.00021	0.416	0.00589
LSTM (Word2Vec)	0.449	0.00331	0.423	0.00289
LSTM (GloVe)	0.472	0.00806	0.438	0.00225
Tree-LSTM (Word2Vec)	0.467	0.00279	0.428	0.00372
Tree-LSTM (GloVe)	0.477	0.00608	0.415	0.01585
Node Tree-LSTM (Glove)	0.485	0.00289	0.418	0.00992

The test accuracies of Table 1 are split out per sentence length in Figure 1. The results are in line with the discussed results above. The accura-

cies surprisingly oscillate over the sentence lengths, with a period of around 10 words. To investigate whether this oscillation is due to an imbalanced test set, we plot the histogram of the number of sentences per length. However, this histogram does not have a clear relation with the accuracy distributions, indicating that the oscillation is not due to the number of sentences per length in the test set. Furthermore, we note that the overall accuracy decreases slightly with increasing sentence length. Finally, we note that the high peak of the Node Tree-LSTM is expected, as it presented shorter sentences (the subtrees) during training.

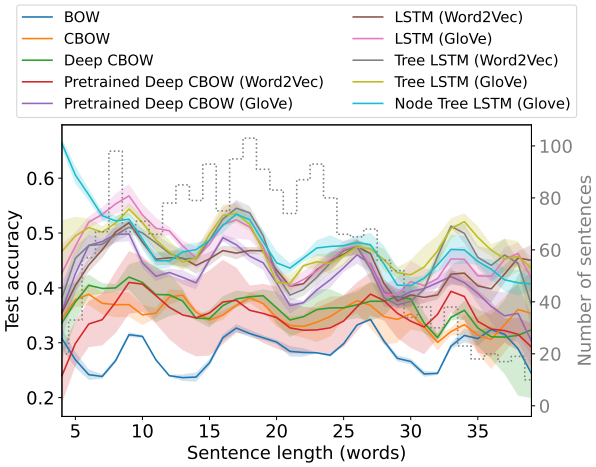


Figure 1: Test accuracy per sentence length, across various models, indicated by the left axis. The number of sentences per length is indicated by the grey histogram, with its values indicated on the right axis. Only sentence lengths that have sufficient data in the test set are taken into account. That is, we do not consider sentences with lengths below 4 or above 40. Additionally, the accuracies are smoothed by a Gaussian filter with $\sigma = 1$. The shaded area indicated the standard deviation across the seeds.

We have seen in Table 1 that the accuracies of the models keep increasing when we consider more complex models that can incorporate more aspects. This indicates that we have not reached an upper limit of what sentiment analysis models can process. To further investigate how to improve Node Tree-LSTM models, we analyze at what point in the tree the sentence fails. We apply the trained Node Tree-LSTM on the test set, but now also include all of its subtrees.

To analyze this, we plot the normalized height (see Section 4 for the definition) split per correctness of the subtrees and full sentence sentiment prediction. The histograms as shown in Figure 2, show that nodes with a small normalized height are often predicted correctly. This is in line with the peak of short sentences of the Node Tree-LSTM in Figure 1. Higher subtrees are more often predicted incorrectly. Furthermore, contrary to our expectations, there is no significant difference between subtrees that are part of correctly or incorrectly predicted full sentences. A qualitative analysis of two tree structures, and where the subtree node prediction fails, is discussed in Appendix C.

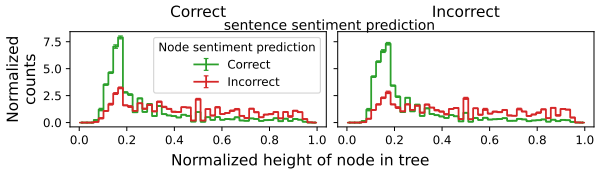


Figure 2: Histograms of node (the subtree) sentiment prediction across normalized height (see main text for definition), split out when the subtree was part of a full sentence was predicted correctly (left) or incorrectly (right). Error bars (although hardly visible) indicate the standard deviation across three different seeds.

6 Conclusion

We tested various BOW and LSTM models to assess the impact of word embedding, word order, supervising node sentiment and hierarchy.

We found that choosing the correct word embedding is important, which is in line with the findings of Mikolov et al. (2013b). The vocabulary of the word embeddings needs to have sufficient overlap with the dataset. We leave the degree of importance for future research.

As opposed to Pambudi and Suprapto (2021), we found no clear trend in the performance to predict sentences of certain lengths, besides a slight decline near the longest sentences.

We found that taking more syntactic structure (word order, hierarchy, supervising node sentiment) into account improves performance as found by Barry (2017); Tai et al. (2015). Taking all this syntactical information into account, e.g. with Node Tree-LSTMs, an upper limit on the performance has not yet been reached. We have investigated this in more detail by analyzing the distribution of incorrectly and correctly predicted nodes. We found that there was no difference in these distributions if the full sentence was predicted correctly or incorrectly. This leaves the question open of how models can be improved to incorporate more syntactical information.

References

James Barry. 2017. Sentiment analysis of online reviews using bag-of-words and lstm approaches. In *AICS*, pages 272–274.

Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. 2007. Large language models in machine translation.

J. Fenn and L. Veren. 1991. Expert system development methodologies in theory and practice. In *1991 IEEE/ACM International Conference on Developing and Managing Expert System Programs*, pages 262–266. IEEE Computer Society.

Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

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. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Wisam Qader, Musa M. Ameen, and Bilal Ahmed. 2019. An overview of bag of words;importance, implementation, applications, and challenges. pages 200–204.

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? *arXiv preprint arXiv:1804.06323*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075.

A Overlap pre-trained word embeddigns with dataset

The discrepancy between the performance of models train with Word2Vec and GloVe can because of (a combination of) two reasons: the values of the weights it achieved during pretrainig, or the overlap of words with the Stanford Sentiment Treebank dataset.

An analysis of the second reason is shown in Figure 3, where we compare the vocabulary of the pre-trained word embeddings with the union of the train and test dataset. We chose to compare it with the union, because both sets have an effect on the performance of the test accuracy. GloVe has both in a relative and absolute way, a larger overlap with the dataset, compared to Word2Vec. This could be the reason that GloVe consistently outperforms Word2Vec. It shows that choosing a suitable word-embedding is important.

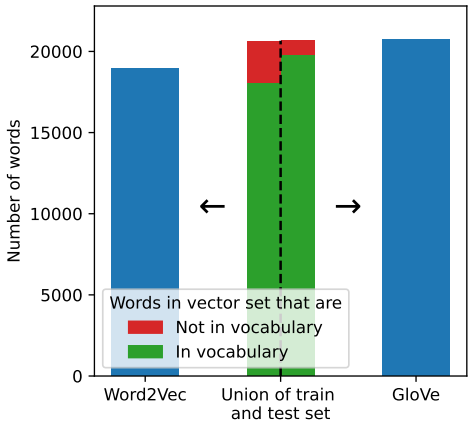


Figure 3: Overlapping (green) and missing (red) words in the train and test set with the vocabulary of Word2Vec and GloVe.

B Transition equations of LSTM and Tree-LSTMs

The three main gates in an LSTM cell: the input gate, the forget gate, and the output gate, are defined as follows:

$$i_t = \sigma\Big(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\Big)$$

$$f_t = \sigma\Big(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\Big)$$

$$o_t = \sigma\Big(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\Big)$$
$$u_t = \sigma\Big(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\Big)$$
$$c_t = i_t \odot u_t + f_t \odot c_{t-1}$$
$$h_t = o_t \odot \tanh(c_t)$$

In which i_t is the input gate at timestep t, f_t is the forget gate, and o_t is the output gate as discussed in section 3.2. Each gate has its own parameter matrix W, U and a bias vector b. The cell state is represented by c_t and the hidden state is h_t.

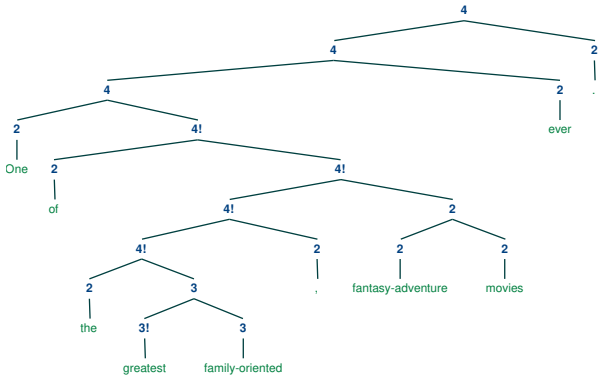
The Binary Tree-LSTM transition equations are the following:

$$i_j = \sigma\Big(W^{(i)}x_j + \sum_{l=1}^{N} U_l^{(i)}h_{jl} + b^{(i)}\Big)$$
$$f_{jk} = \sigma\Big(W^{(f)}x_j + \sum_{l=1}^{N} U_{kl}^{(f)}h_{jl} + b^{(f)}\Big)$$
$$o_j = \sigma\Big(W^{(o)}x_j + \sum_{l=1}^{N} U_l^{(o)}h_{jl} + b^{(o)}\Big)$$
$$u_j = \tanh\Big(W^{(u)}x_j + \sum_{l=1}^{N} U_l^{(u)}h_{jl} + b^{(u)}\Big)$$
$$c_j = i_j \odot u_j + \sum_{l=1}^{N} f_{jl} \odot c_{jl}$$
$$h_j = o_j \odot \tanh(c_j)$$

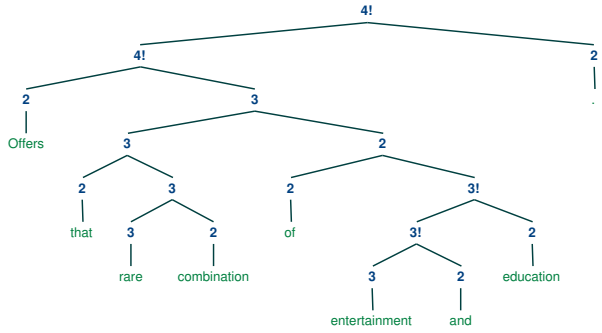
The main difference between the regular LSTM is the individual forget gate for each child in the tree. For any node j, we represent the hidden state and the memory cell of its k^{th} child as h_{jk} and c_{jk} respectively. This allows the tree to determine which nodes are of importance for the task at hand. Other adjustments follow from this, as in the calculation of the cell state c_j, where we now sum over the product of the forget gates and the cell states, as there are multiple for each child.

C Node-level prediction

Figure 4 shows two tree structures, also indicating whether the node sentiment is predicted correctly or incorrectly. All incorrect (correct) nodes in Figure 4a will contribute to the red (green) histogram in left plot of Figure 2, because the full sentence was predicted correctly. Similarly, all the nodes from the Figure 4b will contribute to the histograms in the right plot of Figure 2, because the full sentence was predicted incorrectly.



(a) correctly predicted sentence



(b) Incorrectly predicted sentence

Figure 4: Trees of test set with digits indicating the treue classes. The exclamation mark '!' behind a digit indicates if a node is predicted incorrectly by the Node Tree-LSTM.

As we have seen in Figure 2, there was no difference between the histograms of correctly and incorrectly predicted sentences. We see this pattern also when we examine Figure 4 in more detail. Mistakes deep in the tree (so with a low normalized height) are not necessarily propagated to the full sentence prediction. The incorrectly predicted node that makes the words *the greatest family-oriented* in Figure 4a propagates higher up in the tree, but the full sentence is still predicted correctly. Contrary, Figure 4b has most deeper and middle nodes predicted correctly, while the full sentence still gets predicted incorrectly. This is in line with our observations in Figure 2