# NLP1 2022 Practical 2

**Andrei-Eusebiu Blahovici**
Student number 14532921

**Erencan Tatar**
Student number 12681598

## 1 Introduction

The purpose of this report is to evaluate different neural models for the task of understanding the sentiment expressed in text. We will use the Stanford Sentiment Treebank (SST) dataset, which provides sentences, their binary tree structure, and a 1 out of 5 sentiment score. The following models will be tested: Bag-of-words (BOW), Continuous bag-of-words (CBOW), Deep continuous bag-of-words (Deep CBOW), LSTM, and Tree-LSTM (using the provided tree structure). Understanding the optimal use case for each model can be beneficial in terms of the accuracy and complexity trade-off.

Sentiment analysis can be used to analyze social media or customer behavior, and thus can be very useful. To analyze the strong and weak points of certain models, we look into different aspects of text classification.

The report will first discuss the importance of word order for classifying sentiment, as the arrangement of words can alter the meaning and sentiment of a sentence. To investigate the importance of word order on the classification task, we trained the above models on ordered sentences and evaluated them on both an ordered and unordered dataset.

Another aspect of the model's performance is the sentence length. Longer sentences are often more complex, but at the same time contain more words that convey meaning, so understanding how well the models handle long or short sentences can be valuable information. All the BOW models are expected to be invariant for the sentence length, however LSTM models are expected to perform worse on longer sentences, agreeing with figure 4 in (Zulqarnain et al., 2020).

Using the SST dataset it is possible to explore if the annotated sentence structures on top of an LSTM improves the accuracy. By looking at bigger structures in the sentence, it is expected that the model can tackle negations, expressions, and ambiguities better. This has been confirmed by (Li et al., 2020), where they showed that the Tree-LSTM performed better compared to the regular LSTM approach for sentiment classification. This however comes at the expense of using a parse tree structure, expensive phrase-level annotations, and more computations.

Finally, building upon the tree-structure, we discuss a different model making use of subtrees. The dataset is augmented regarding shuffling, sentence length, and tree structure for each sub-question, in order to find the optimal use case for each model.

Our report shows that *PT DeepCBOW* works best across all experiments, even though we expected that LSTM or Tree-LSTM would have worked better.

However, word order did not have as much as an impact, for the LSTM as was initially expected. Also against expectations is that subtree's did not improve the models accuracy.

## 2 Background

In this project, we come across multiple core techniques in NLP such as BOW, n-grams, word embeddings, LSTM, Tree-LSTM. A bag-of-words (BOW) model is a good starting point for a simple NLP model. Here we count the word frequencies for a given context or a given classification class. Here the context can be the surrounding words, using for example an n-gram, where we look back at the n previous words.

### 2.1 Word embeddings

The DeepCBOW and LSTM models make use of the pre-trained Word2Vec or GloVe (Pennington et al., 2014) word embeddings.

Word embeddings are used for encoding the meaning of a word in a high-dimensional vector

space, where the distance between vectors reflects the similarity between the words they represent. In natural language processing, it is often necessary to convert words to numerical vectors in order to input them into a model. However, representing words as simple one-hot vectors (where each word is represented by a vector with a single 1 in the position corresponding to that word, and 0s everywhere else) is not very effective, because it does not capture the semantic relationships between words. Word embeddings, on the other hand, are able to capture these relationships, which makes them more useful for many NLP tasks.

Using these pre-trained word embeddings, we can initialize our word embedding lookup table and start from a point where similar words are already close to one another in the distributional semantic space.

### 2.2 LSTM

LSTM's is a type of RNN (recurrent neural network) that handles the flow of information through the network with a more complex structure, resulting in that LSTMs are able to better retain long-term information and prevent it from being forgotten by the network. Both RNNs and LSTMs are types of neural networks that can process sequential data, such as time series or natural language.

An LSTM cell, is a function that takes in an input *x* and a previous hidden state *h*, and combines all the information from the previous LSTM cells to produce an output *c'* and a next hidden state *h'*.

$$
\begin{aligned}
i &= \sigma(W_{ii}x + b_{ii} + W_{hi}h + b_{hi}) \\
f &= \sigma(W_{if}x + b_{if} + W_{hf}h + b_{hf}) \\
g &= \tanh(W_{ig}x + b_{ig} + W_{hg}h + b_{hg}) \\
o &= \sigma(W_{io}x + b_{io} + W_{ho}h + b_{ho}) \\
c' &= f * c + i * g \\
h' &= o \tanh(c')
\end{aligned}
$$

According to (Greff et al., 2016), *i* represents the input gate, $f$ the forget gate, and $o$ the output of that LSTM cell, with $\sigma$ being the sigmoid function.

### 2.3 Tree-LSTM

A TreeLSTM is a type of recurrent neural network (RNN) that is specifically designed to operate on tree-structured data. It processes Tree-structured data that has a hierarchical structure, where each item has one or more "child" items that are grouped under it.

This is done by adding additional connections to the standard RNN structure, which allows it to propagate information between the different levels of the tree. In this way, a TreeLSTM is able to "remember" not only the previous items in a sequence but also the hierarchical structure of the data. This makes them well-suited for tasks such as natural language processing, where the hierarchical structure of sentences is important.

## 3 Models used

We have experimented with multiple models for these tasks and their architectures are described in this section.

**DeepCBOW**

The DeepCBOW used three linear with the tanh activation function in between. The embedding dimension for each word of 300 is first mapped to a hidden dimension of size 100 and then mapped back to the output dimensions for the 5 sentiment classes.

**LSTM**

Our LSTMClassifier model uses the 300-dimensional embeddings for all the 20727 words in our vocab. The LSTM maps this 300Dim vector to a 168 space using an MLP / dense layer, to ultimately project it to the 5 different sentence categories. Right before the last projection, a dropout layer has been placed to make the model generalize better,

**Tree-LSTM**

The Tree-LSTM makes use of the tree structure of the input which might unveil relevant syntactic information about the sentence. It does this by instead of only using the leaves of the tree, it computes a hidden state for each node in the tree base on its children. Additionally, using a Tree-LSTM on a chain graph is equivalent to using a simple LSTM.

## 4 Experiments

The task at hand is a multi-label classification problem, where the models predict between the five classes, *very negative* to *very positive*, and everything in between. Therefore during training the model loss is calculated using the cross-entropy function. For all of our experiments, three different models were trained using different seeds to obtain a final mean accuracy across all. To update the model weights during the training step the Adam optimizer was used. We are going to specify the

learning rate that we used for each model in the corresponding subsection. Also, we would like to mention that the metric we used for evaluating our models is the well-known accuracy. For most of our experiments we have used just the words in the tree bank as the features, while the labels are the top-level sentiment class. Of course, the exceptions from this rule are the experiments involving the Tree-LSTM, where we also used the binary tree structure of the data. The models we don't use the tree structure for are BOW, CBOW, Deep CBOW, Deep CBOW with pre-trained embeddings (which we keep fixed), LSTM, and Mini-batch LSTM. In addition to these 7 models, we also train two variations of Tree-LSTM, namely Child-Sum and N-arry for which we use the data as it is, disregarding the sentiment classes for each node. Table 1 shows the hyperparameters used for training each model. After establishing these benchmarks, we further run some more experiments to see if we can improve the baseline accuracy of our models and to check the robustness of our models to distributional shifts in the data. For the next experiments, we are going to talk about we are going to evaluate only the models DeepCBOW, LSTM (without mini-batches), and N-arry version of LSTM for reasons that we will talk about in the next section. In order to verify that our models can generalize fairly well to new datasets, we are going to evaluate them on the test data again, but this time we shuffle the order of the words to see if it has any effect on our overall accuracy. After this, we also check to see if certain models perform better or worse for sentences of a certain length. This is done to see how well our models can retain long-term dependencies in the data. The last experiment that we did was to augment the dataset using all the subtrees in the training data to see if it helps the Tree-LSTM perform better.

| **Model** | Learning rate | Epochs |
|---|---|---|
| *BOW* | 5e-4 | 60000 |
| *CBOW* | 5e-4 | 60000 |
| *DeepCBOW* | 5e-4 | 60000 |
| *PT DeepCBOW* | 5e-4 | 50000 |
| *LSTM* | 3e-4 | 7000 |
| *MB LSTM* | 2e-4 | 7000 |
| *Tree-LSTMs* | 2e-4 | 1000 |

Table 1: Hyperparameters used for training each model

## 5 Results and Analysis

The first results we have obtained are the benchmarks for our models which are presented in Table 2. After evaluating the first 3 BOW-based models, we have noticed that the models cannot learn meaningful embeddings because the dataset is too small, so we resorted to pre-trained embeddings, namely Glove embeddings, which boosts the overall accuracy by about 7%.

| **Model** | Accuracy |
|---|---|
| *BOW* | 28.42% |
| *CBOW* | 37.51% |
| *DeepCBOW* | 36.05% |
| *PT DeepCBOW* | 44.14% |
| *LSTM* | 40.5% |
| *MB LSTM* | 29.24% |
| *Tree-LSTMs* | 43.05% |

Table 2: Benchmark accuracies of the models

In the next experiments, we will be using only 3 models for comparison, namely DeepCBOW because it is the best CBOW-based model, simple LSTM with no mini-batches because it performs better than the one with mini-batches, and the N-arry Tree-LSTM.

As already mentioned, after evaluating these benchmarks, we try to evaluate the robustness of our model to distributional shifts so we evaluate the models on shuffled data. The results of this experiment are shown in Table 3.

| **Model** | Accuracy | Std |
|---|---|---|
| *PT DeepCBOW* | 44.62% | 0.002 |
| *LSTM* | 43.11% | 0.011 |
| *Tree-LSTM* | 42.65% | 0.013 |

Table 3: Mean accuracies and standard deviations for the word shuffling experiment

Furthermore, we are interested in seeing if different models perform better or worse depending on the length of the sentence. To do this, we sorted the test data by the length of the sentence and split it in 4 separate datasets on which we will evaluate the models. Table 4 shows the results we obtained on the 4 datasets. Each column corresponds to a percentage of the dataset, so the column 25% refers to the first 25% of samples in the sorted dataset, the column 50% refers to the next 25% of words, and so on.

| **Model** | 25% | 50% | 75% | 100% |
|---|---|---|---|---|
| *DeepCBOW* | 51.39% | 44.72% | 47.27% | 44.2% |
| *LSTM* | 49.45% | 42.42% | 47.03% | 42.14% |
| *Tree-LSTM* | 47.63% | 40.00% | 44.96% | 41.42% |

Table 4: Mean accuracies for the varying sequence length experiments.

| **Model** | 25% | 50% | 75% | 100% |
|---|---|---|---|---|
| *DeepCBOW* | 0.017 | 0.007 | 0.007 | 0.01 |
| *LSTM* | 0.011 | 0.014 | 0.014 | 0.004 |
| *Tree-LSTM* | 0.018 | 0.010 | 0.015 | 0.010 |

Table 5: Mean standard deviation for the varying sequence length experiments across the experimental seeds.

So far, we haven't used the intermediary node sentiments from our data samples. To do so, we are going to augment our training dataset by taking all the subtrees from the training tree bank and add them as training data samples. This way we are going to have both more samples and more diversity in our training samples which might lead to better generalization since data augmentation is a known regularisation technique used in deep learning. The results of this experiment are shown in Table 6.

| **Model** | Accuracy | Std |
|---|---|---|
| *PT DeepCBOW* | 44.59% | 0.006 |
| *LSTM* | 44.05% | 0.016 |
| *Tree-LSTM* | 43.80% | 0.01 |

Table 6: Mean accuracies and standard deviations for the subtrees data augmentation experiment

Moreover, we experimented to see if the two types of Tree-LSTMS, N-arry and Child-Sum, differ in any way in terms of the accuracy on the test data. After training the Child-sum Tree-LSTM on the data with no additional pre-processing than for the benchmarks we obtain an accuracy of 42.96% which is a bit lower than the N-arry LSTM.

## 6 Conclusion

From the first benchmarks in table 2, we can see that the DeepCBOW with pre-trained embeddings performs best out of the box, although the LSTM approaches are not too far from it. Even when shuffling the sentence tokens, PT DeepCBOW outperforms the other models.

From table 2 it can be seen that Tree-LSTM outperforms vanilla LSTM, which is in line with the literature. So we conclude that the tree structure helps the model get a higher accuracy? However, in the shuffling and sub-tree data augmentation experiments we see that models are comparable.

As was expected, for an increase in sentence length, we see a decrease in mean accuracies across all models even for the DeepCBOW model. From this we can conclude that having longer sentences, and thus more sentimental words, results in a decrease in accuracy.

Future work can experiment with transformer architectures like BERT(Devlin et al., 2018) or XLM-RoBERTa(Conneau et al., 2019) to extract features. The transformer architecture is a powerful tool for NLP tasks because of its ability to capture long-range dependencies, its parallelization, and its attention mechanism, which allows it to focus on the most relevant parts of the input when making predictions.

## References


Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

Weijiang Li, Fang Qi, Ming Tang, and Zhengtao Yu. 2020. Bidirectional lstm with self-attention mechanism and multi-channel features for sentiment classification. *Neurocomputing*, 387:63–77.

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.

Muhammad Zulqarnain, Rozaida Ghazali, Yana Mazwin Mohmad Hassim, and Muhammad Rehan. 2020. A comparative review on deep learning models for text classification. *Indones. J. Electr. Eng. Comput. Sci*, 19(1):325–335.