NLP1 Practical 2 - Representing Sentences with Neural Models

Ippokratis Pantelidis (16124006) and Nikos Toganidis (16306066)
MSc Artificial Intelligence, University of Amsterdam
{ippokratis.pantelidis, nikos.toganidis}@student.uva.nl

1 Introduction

Neural models are a core component of modern Natural Language Processing, as they learn representations of words, phrases, and sentences that capture both semantic and syntactic information. In this work, we study *sentence-level sentiment classification*, where the goal is to predict the sentiment expressed by a sentence. We conduct our experiments on the Stanford Sentiment Treebank (SST; Socher et al., 2013), a benchmark dataset that provides fine-grained sentiment annotations not only for complete sentences but also for every node in a binary parse tree.

The main objective of this practical is to examine how different modeling choices affect sentence representation quality and sentiment classification performance. We compare several families of neural architectures, starting with bag-of-words approaches that ignore word order, followed by sequential models such as Long Short-Term Memory networks (LSTMs; Hochreiter and Schmidhuber, 1997), and tree-structured LSTMs Tai et al., 2015, which explicitly incorporate syntactic structure.

Our study is guided by several research questions. First, we ask whether modeling word order improves performance compared to bag-of-words models. Second, we investigate whether syntactic structure provides additional benefits by comparing LSTMs and Tree-LSTMs. Third, we analyze how model performance varies with sentence length and whether some architectures are more robust to longer sentences. Fourth, we examine whether supervising sentiment at each internal node of the parse tree improves Tree-LSTM performance. Finally, we examine how the choice of pretrained word embeddings affects performance by comparing GloVe and Word2Vec representations. These questions are motivated by prior work on compositional models for sentiment analysis, but the empirical benefits of structure, supervision, and embedding choice remain an open question.

To answer these questions, we train all models under matched conditions and report average test accuracy across multiple random seeds. Our results show that incorporating word order and syntactic structure leads to consistent performance gains, with the Tree-LSTM achieving the best overall accuracy. We also find that Tree-LSTMs are more robust to increasing sentence length, and that node-level supervision yields a substantial improvement for the Tree-LSTM, while the choice of pretrained embeddings results in smaller but consistent gains. In particular, GloVe embeddings slightly outperform Word2Vec across all evaluated models.

2 Background

In this section, we describe the main building blocks used in our project focusing on the intuition behind each method and on how these techniques relate to one another.

Bag-of-Words Models represent a sentence by combining the vector representations of its words, typically through summation. These models ignore word order and treat the sentence as an unordered set of tokens. In this project, we consider several bag-of-words variants, including a simple BOW model, a Continuous Bag-of-Words (CBOW) model, and a Deep CBOW model. While these variants differ in architectural complexity, they all share the same core assumption: sentence meaning is derived from the aggregate contribution of individual words rather than from their order or syntactic relations. As a result, bag-of-words models are computationally efficient and serve as strong lexical baselines, but they are inherently limited in their ability to capture compositional phenomena that depend on word order or hierarchical structure.

Long Short-Term Memory Networks (LSTMs; Hochreiter and Schmidhuber, 1997) extend recurrent neural networks by introducing an explicit

memory cell together with gating mechanisms that control information flow. As an LSTM processes a sentence sequentially from left to right, it maintains a hidden state that summarizes past information and a memory cell that stores longer-term content. At each time step, the model learns to selectively incorporate new input, retain relevant past information, or discard irrelevant content. This gating structure mitigates the vanishing gradient problem of standard RNNs and allows LSTMs to capture dependencies that span multiple time steps. As a result, LSTMs are well suited for tasks where word order and sequential composition influence meaning, such as sentiment classification involving negation, intensification, or contrast. For completeness, the formal LSTM update equations are provided in Appendix A.

Tree-Structured LSTMs (Tai et al., 2015) extend the LSTM architecture from linear sequences to hierarchical tree structures. Rather than composing word representations strictly from left to right, a Tree-LSTM computes hidden and memory states bottom-up by recursively combining the states of a node's children. Tai et al. propose several variants, including the *N-ary Tree-LSTM*, which allows a fixed number of ordered children per node. In this practical, we use the binary Tree-LSTM variant, corresponding to the binary parse trees provided by the Stanford Sentiment Treebank. This allows the model to follow the syntactic organization of the sentence and to compose representations in accordance with linguistic structure. Such hierarchical composition is particularly relevant for sentiment analysis, where meaning often arises from interactions between phrases rather than from individual words alone. For completeness, the formal Tree-LSTM update equations used in our experiments are provided in Appendix B.

Word Embeddings map each word to a dense vector that encodes semantic and syntactic information. Instead of treating words as independent symbols, embeddings place them in a continuous vector space where words that occur in similar linguistic contexts receive similar representations. This distributional structure enables neural models to generalize across lexical variation and to exploit regularities in language use. In this project, we primarily use pre-trained *GloVe* embeddings (Pennington et al., 2014). GloVe embeddings are learned by factorizing global word co-occurrence statistics collected over large corpora, allowing them to capture both local contextual information and broader corpus-level patterns. Their rich semantic structure makes them a strong choice for sentiment classification, particularly when the amount of labeled training data is limited, and they therefore form the main embedding representation used in our experiments. For completeness, we also consider *word2vec* embeddings (Mikolov et al., 2013) in an additional analysis. Word2vec embeddings are learned by predicting local context relationships and primarily encode short-range distributional information. This comparison allows us to assess whether different embedding training objectives influence downstream performance beyond architectural choices.

3 Models

This section describes the architectures used in our experiments. All models take a sentence as input and predict one of the five sentiment labels of the Stanford Sentiment Treebank. In all models, words are mapped to embedding vectors, which are either randomly initialized or initialized with pre-trained GloVe embeddings. Across architectures, models differ in how they construct a sentence representation, ranging from simple aggregation of word embeddings to sequential and tree-structured composition. All models are trained using cross-entropy loss and optimized with the Adam optimizer.

3.1 Bag-of-Words Models

The BOW model learns an embedding vector for each word and represents a sentence by summing these embeddings and adding a trainable bias. The resulting vector is directly interpreted as class logits, with no hidden layers or non-linearities.

The CBOW model extends BOW by applying a linear projection to the summed embedding vector, allowing the model to learn weighted combinations of embedding dimensions relevant for sentiment classification.

The Deep CBOW model further increases model capacity by replacing the single projection layer with a multi-layer feed-forward network with Tanh activations. This enables non-linear transformations of the aggregated embeddings while still ignoring word order.

3.2 LSTM Model

The LSTM model represents a sentence as an ordered sequence of word embeddings. A unidirectional LSTM processes the sentence from left to

right, updating a hidden state at each time step using input, forget, and output gates. The final hidden state is used as the sentence representation. This representation is regularized with dropout and mapped to sentiment logits using a linear classifier, allowing the model to capture order-sensitive sequential composition.

3.3 Tree-LSTM Model

The Tree-LSTM model operates over binary parse trees provided by the Stanford Sentiment Treebank. Each word is mapped to an initial hidden and cell state, and tree structure is encoded as a sequence of SHIFT and REDUCE transitions. SHIFT operations push word representations onto a stack, while REDUCE operations combine the top two stack elements using a binary Tree-LSTM composition function. After all transitions are applied, the remaining stack element represents the root of the tree and serves as the sentence representation. This representation is regularized with dropout and passed to a linear classifier to obtain the final prediction.

4 Experiments

This section describes our experimental setup including data, hyper-parameters and evaluation metrics.

Each example in SST is represented as a sequence of tokens, a binary parse tree, and a sentiment label in the range 0–4. For models that operate on the surface form of the sentence (BOW, CBOW, Deep CBOW, and LSTM), we use only the token sequence. For the Tree-LSTM, we additionally use the sequence of SHIFT/REDUCE transitions derived from the parse tree. We experiment with both randomly initialized word embeddings and pre-trained GloVe embeddings of 300 dimensions.

All models are implemented in PyTorch and trained using the Adam optimizer with a learning rate of 5×10^{-4}. Unless stated otherwise, training proceeds by sampling a single example per iteration, resulting in an effective batch size of one; for the LSTM, we additionally experiment with a mini-batched variant. Each model is trained for 30,000 iterations using cross-entropy loss over the five sentiment classes. When pre-trained GloVe embeddings are used, they are kept fixed during training, while all remaining parameters are learned. To reduce overfitting, dropout is applied to the sentence representations in the LSTM and Tree-LSTM models.

We evaluate models using classification accuracy. During training, we compute development-set accuracy every 1,000 iterations. We save the model checkpoint that achieves the highest development accuracy and use this checkpoint to compute the final test accuracy. To account for randomness in parameter initialization and data order, each model is run with three different random seeds (42, 0, 123). We report the mean and standard deviation of test accuracy across these runs in the following section.

This setup ensures that all models are trained under comparable conditions and that our results are reproducible.

5 Results and Analysis

This section presents and analyses the experimental results. All results are evaluated on the test set using the best checkpoint selected on the development data, and we focus on how model architectures differ with respect to the research questions introduced earlier. Table 1 reports the test accuracy for all models. Results are shown as mean accuracy and standard deviation across the three random seeds mentioned above.

Model	Test Acc. (mean $\pm$ std) (%)
BOW	0.2600 ± 0.0174
CBOW	0.3416 ± 0.0300
Deep CBOW	0.3555 ± 0.0136
Deep CBOW+GloVe	0.4314 ± 0.0097
LSTM	0.4603 ± 0.0068
LSTM (mini-batch)	0.4597 ± 0.0072
Tree-LSTM	0.4756 ± 0.0035

Table 1: Mean test accuracy and standard deviation across three random seeds.

5.1 Effect of Word Order

We examine the effect of word order by comparing bag-of-words models (BOW, CBOW, Deep CBOW) with order-sensitive models (LSTM and Tree-LSTM). As shown in Table 1, models that use word order consistently achieve higher accuracy. While richer bag-of-words models such as Deep CBOW+GloVe perform reasonably well, LSTM and Tree-LSTM further improve performance indicating that modeling word order provides a clear but moderate benefit for this task.

5.2 Effect of Tree Structure

To assess the impact of syntactic structure, we compare the Tree-LSTM with the standard LSTM. As shown in Table 1, the Tree-LSTM slightly but consistently outperforms the LSTM, achieving the highest overall accuracy. The improvement is modest but stable, suggesting that incorporating tree structure provides additional benefits beyond sequential modeling alone.

5.3 Performance Across Sentence Length

To analyze how sentence length affects model performance, we group test sentences into three bins based on the number of tokens: short (1–10), medium (11–20), and long (21+). For each bin, we compute classification accuracy separately for all models.

Figure 1 shows accuracy by sentence length bin. Overall, accuracy decreases as sentence length increases, indicating that longer sentences are more challenging for all models. Bag-of-words models exhibit the largest drop in performance for long sentences, while order-sensitive models degrade more gradually. In particular, the Tree-LSTM maintains strong performance on medium and long sentences, suggesting that explicit syntactic structure helps mitigate the impact of increasing sentence length.

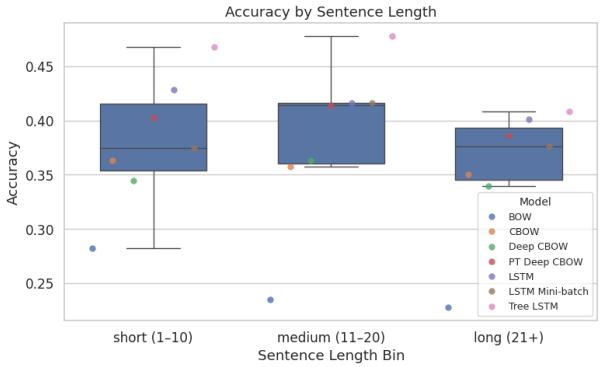

Figure 1: Accuracy by sentence length bin for all models.

5.4 Effect of Node-Level Supervision

To investigate whether supervising sentiment at each node improves model performance, we apply node-level supervision to the Tree-LSTM model only. We expand the training data by extracting all valid subtrees from each sentence and treating each subtree as an independent training example, labeled with the sentiment at its root. This increases the size of the training set from **8,544** full-sentence examples to approximately **163,563** subtree-level examples, corresponding to a **19.1**× increase.

Evaluation is performed only on full sentences in the original development and test sets. With node-level supervision, the Tree-LSTM achieves a test accuracy of **51.09%**, compared to **47.96%** when trained using only root-level supervision, an absolute improvement of **3.13** percentage points. Although this gain is modest relative to the increase in training data, it is consistent and suggests that intermediate node supervision provides useful additional learning signals for the Tree-LSTM.

5.5 GloVe vs. Word2Vec Embeddings

We compare the effect of pretrained word embeddings by evaluating models using GloVe and Word2Vec embeddings under the same random seed (42). Across all models, GloVe embeddings consistently outperform Word2Vec, although the differences are generally modest. For Deep CBOW, GloVe achieves slightly higher accuracy than Word2Vec (43.4% vs. 42.3%). A similar trend is observed for LSTM-based models: the standard LSTM improves from 44.4% to 45.3%, while the mini-batched LSTM shows nearly identical performance with both embeddings. The largest gap appears for the Tree-LSTM, where GloVe provides a clearer advantage (48.0% vs. 45.9%). Overall, these results suggest that while both pretrained embeddings are effective, GloVe offers small but consistent gains, particularly for structure-aware models.

6 Conclusion

In this work, we compared several neural architectures for sentence-level sentiment classification on the Stanford Sentiment Treebank. Our experiments show that modeling word order improves performance over bag-of-words models, and that incorporating syntactic structure with Tree-LSTMs leads to the best overall results. Tree-LSTMs are also more robust to longer sentences.

We further find that supervising sentiment at internal tree nodes and the choice of pretrained embeddings provide smaller but consistent gains, with GloVe slightly outperforming Word2Vec. Overall, our results largely match expectations and support prior findings on the importance of compositional structure for sentiment analysis.

Use of AI Tools

The authors confirm that no AI-based tools or language models were used in the writing of this report, the implementation of the code, or the design and analysis of the experiments.

References

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. 2013. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, pages 1532–1543.

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. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 1556–1566.

A LSTM Formulation

An LSTM processes an input sequence $\{x_t\}_{t=1}^{T}$ by maintaining a hidden state $h_t \in R^d$ and a memory cell $c_t \in R^d$ at each time step t. Given the current input x_t and the previous hidden state h_{t-1}, the LSTM computes the following gating functions:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (2)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (3)$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$

Here, i_t, f_t, and o_t denote the input, forget, and output gates, respectively, and $\tilde{c}_t$ is the candidate memory content. The memory cell and hidden state are then updated as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (5)$$
$$h_t = o_t \odot \tanh(c_t) \quad (6)$$

where $\sigma(\cdot)$ denotes the sigmoid function and $\odot$ denotes element-wise multiplication. The gating mechanisms allow the LSTM to regulate how information is accumulated and propagated over time.

B Binary Tree-LSTM Formulation

A binary Tree-LSTM computes representations for tree-structured inputs by composing information from two child nodes at each internal node. Each node j maintains a hidden state $h_j \in R^d$ and a memory cell $c_j \in R^d$. Let l and r denote the left and right children of node j, with hidden states h_l, h_r and memory cells c_l, c_r.

Given an input vector x_j at node j (corresponding to a word embedding for leaf nodes and a zero vector for internal nodes), the Tree-LSTM computes the following gating functions:

$$i_j = \sigma(W_i x_j + U_i^{(l)} h_l + U_i^{(r)} h_r + b_i) \quad (7)$$
$$f_{jl} = \sigma(W_f x_j + U_f^{(l)} h_l + U_f^{(r)} h_r + b_f) \quad (8)$$
$$f_{jr} = \sigma(W_f x_j + U_f^{(r)} h_r + U_f^{(l)} h_l + b_f) \quad (9)$$
$$o_j = \sigma(W_o x_j + U_o^{(l)} h_l + U_o^{(r)} h_r + b_o) \quad (10)$$
$$\tilde{c}_j = \tanh(W_c x_j + U_c^{(l)} h_l + U_c^{(r)} h_r + b_c) \quad (11)$$

Here, i_j is the input gate, f_{jl} and f_{jr} are forget gates for the left and right child respectively, o_j is the output gate, and $\tilde{c}_j$ is the candidate memory content.

The memory cell and hidden state of node j are then updated as:

$$c_j = i_j \odot \tilde{c}_j + f_{jl} \odot c_l + f_{jr} \odot c_r \quad (12)$$
$$h_j = o_j \odot \tanh(c_j) \quad (13)$$

where $\sigma(\cdot)$ denotes the sigmoid function and $\odot$ denotes element-wise multiplication. These equations generalize the standard LSTM update by allowing separate forget gates for each child node, enabling selective composition of hierarchical information.