NLP1 Practical 2 Report

Francijn Keur
13133152

Yunchong Huang
14775875

1 Introduction

This report focuses on the sentential-level sentiment analysis task implemented with deep learning models. We mainly discuss two model families of Bag-of-Words (BoW) and Long Short-Term Memory (LSTM), with a motivation to explore how different sentence encodings in various models influence their performance.

We start by inspecting *whether model performance improves if word order is encoded in sentence representations* (**RQ1**). A performance comparison between BoW-based models and LSTM-based models (Hochreiter and Schmidhuber, 1997; Greff et al., 2015; Tai et al., 2015) can reveal this, since the additive sentence encoding in the former is insensitive to word order, whereas the latter is designed to process sequential data. We expect that including word-order information would improve the performance as it helps to grasp an important aspect of natural language semantics.

The second question is *whether the model performance improves if the tree structure is encoded* (**RQ2**). A performance comparison between the classic LSTM models and the Tree-LSTM model can verify this. Several different tree-LSTM architectures were independently proposed (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015), and they were all reported to outperform the classic LSTM model in the sentiment analysis task, and we expect similar results for our experiments.

The third question is *finding out models that are more robust when the sentence length increases* (**RQ3**). Tai et al. (2015) showed that tree-LSTM models consistently outperform their sequential counterparts on the sentiment analysis task within the sentence length of 30. We expect LSTM models to outperform BoW-based models on longer sentences in general, and tree-LSTM models would outperform the classic LSTM model.

The fourth question is *whether fine-grained node-wise sentiment scores contribute to the model performance* (**RQ4**). This can be revealed by comparing a Tree-LSTM model including node sentiment scores and another trained only on sentence-level sentiment scores. We expect more fine-grained sentiment information would improve the performance, as specific sub-parts with rich emotional expression become “visible".

The last question is about *how important is the child node order information in a tree structure for the sentiment analysis task* (**RQ5**). This can be revealed by the comparison between two variants of tree LSTMs proposed by (Tai et al., 2015): n-ary tree LSTM and child-sum tree LSTM. Their results showed that n-ary tree LSTM slightly outperforms child-sum tree LSTM, revealing the benefits of encoding child node order. We expect to observe similar outcomes in our experiments.

We train a set of BoW models and LSTM models based on the Stanford Sentiment Treebank dataset (Socher et al., 2013) to conduct comparative studies on model performance measured by accuracy, with variables of specific implementations and sentence lengths. Results show that our predictions are mostly verified for **RQ1-RQ4**, but the potential importance of child node order (**RQ5**) is less than expected.

2 Background

We discuss two families of models that encode natural language sentences as inputs into neural networks for the sentence-level sentiment analysis task: Bag-of-Words (BoW) embedding models and Recurrent Neural Network - Long short-Term Memory (RNN-LSTM) models.

Tracing back to Harris (1954), the core idea of BoW is to represent sentences by summing component word embeddings. There are several BoW-based models. In most **basic BoW** models, word embeddings have a fixed dimensionality identical to the task-specific number of outputs. They are initialized randomly, trained by a feedforward neural network and summed to obtain sentence vectors. **Continuous BoW (CBoW)** addresses the low dimensionality limitation of the basic BoW model by allowing word embeddings with an arbitrary-size dimension, making it possible for them to represent more fine-grained information by learning, which also benefits the representation of sentences. A tradeoff of this improvement is that the interpretability of dimensionality in the basic BoW model is lost. **Deep CBoW** introduces more layers and non-linear activations based on CBoW to enhance the model's flexibility to fit complicated patterns, but this also brings the tradeoff such that predictions cannot be easily traced back anymore. Furthermore, Deep CBoW can also utilize pre-trained word embeddings such as Word2Vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014) for initial inputs, leading to **PTDeep CBoW**. The quality of word embeddings pre-trained on large-scale general corpora exceeds those learned from a smaller task-specific corpus, which results in better sentence representations and more satisfactory performance on downstream tasks.

An unavoidable drawback of the additive sentence representation in BoW models is the loss of word order and syntactic information. This is tackled by **Recurrent Neural Network (RNN)** for sequential data originating from Elman (1990). Its core idea is to share the same network weights across multiple time steps, allowing sequential data to be modeled at each step while maintaining an internal state capturing input history from previous steps. However, the training of classic RNNs faces the challenge of vanishing/exploding gradients (Cho, 2015), since gradients can become extremely small/large during backpropagation due to repeated multiplications.

This problem is addressed by a specialized RNN variant, the **Long Short-Term Memory Network** (LSTM) (Hochreiter and Schmidhuber, 1997; Greff et al., 2015). The LSTM introduces a separate cell state to store long-term dependencies and employs multiple "gates" to manipulate the flow of information. The process begins with the "forget gate", which controls how much long-term memory from the previous cell state to retain. Next, a candidate function extracts values potentially addable to the long-term memory, while the "input gate" scales the candidate values for the actual input. The sum of the retained long-term memory and the current scaled candidate values form the updated cell state. To generate the output, the updated cell state is filtered through a non-linear activation (tanh), and the "output gate" controls how much of this information contributes to the updated hidden state.
While the classic LSTM aims at processing linear sequential data, natural language sentences inherently exhibit hierarchical structures. Therefore, tree-LSTM variants are attractive alternatives for sentence modeling considering syntactic structures. They replace the single previous hidden state in classic LSTMs with multiple hidden states imitating branching nodes on trees, and by introducing independent "forget gates" for each child node, their individual contribution to the parent node is regulated. However, their specific methods of integrating branching nodes are different, to which we will turn in the Section 3.

3 Models

Basic BoW. Words are mapped to 5-dimensional trainable embeddings (corresponding to the number of target classes). Sentence embeddings are formed by summing word embeddings with trainable biases. The model consists of a single linear layer, using cross-entropy (Good, 1952) as the loss function optimized with the Adam algorithm (Kingma and Ba, 2014)[1]. The *argmax* function is applied to obtained sentence vectors to retrieve the output prediction. Both embedding weights and biases are updated during backpropagation.

CBOW. Words are mapped to 300-D trainable embeddings. Sentence embeddings are formed by summing word embeddings with trainable biases. The **basic CBOW** possesses a single projection layer with parameter matrices mapping 300-D sentence embeddings to 5-D vectors to make predictions with the *argmax* function. Both embedding weights and biases are updated during backpropagation. **Deep CBoW** adds two hidden layers with non-linear activation (tanh), while the linear projection layer becomes the final layer mapping sentence representations to predictions; **PTDeep CBoW** uses pre-trained Word2Vec word 300-D embeddings as initial inputs for the **Deep CBoW** structure, and they are frozen during training to retain pre-trained semantic information.

[1]The same loss function and optimizer are used across all models, thus they are not repeatedly reported in the following models.

LSTM. [2]The LSTM model encodes sentences using a custom cell where input and hidden state contributions are pre-computed together for efficiency and divided into gates (input, forget, candidate, output) using the chunk method. It uses 300-D Word2Vec word embeddings for input, processes sequences on each timestep, and leverages the final hidden state for the 5-label classification through a dropout layer (Srivastava et al., 2014) followed by a fully connected output layer. Padding is applied to support sentences of varying lengths. Additionally, the mini-batching technique is utilized to improve training efficiency and the embedding fine-tuning option is available to integrate task-specific semantic information.

Tree-LSTM. We have a n-ary tree-LSTM model and a child-sum tree-LSTM model. Both models initialize hidden and cell states using 300-D Word2Vec word embeddings. The n-ary tree-LSTM computes gates for each child with separate parameters based on sequential order and then sums their contributions to update the parent node cell state, whereas the child-sum tree-LSTM first sums the hidden states of all child nodes before passing the combined representation through shared gate parameters, discarding the child node order. Sentence embeddings are formed by applying **SHIFT** and **REDUCE** operations (Bradbury, 2017). The final hidden state is passed through a dropout layer and a fully connected output layer for the 5-label classification. Mini-batching and padding are also employed.

4 Experiments

To train and evaluate the models to perform a sentiment analysis task, the Stanford Sentiment Treebank dataset was used (Socher et al., 2013). This dataset includes around $12,000$ one-sentence reviews, their binary tree structures, sentence-level sentiment scores, and fine-grained sentiment scores annotated at each syntactic node in the tree. There are 5 different sentiment labels: very negative, negative, neutral, positive, and very positive. The train, validation, and test set were of size 8544, 1101 and 2210 respectively.

The models were trained for three seeds[3], using back-propagation to optimize the cross-entropy loss, using PyTorch (Paszke et al., 2019). Furthermore, to see whether performance increased when sentiment was supervised at each node in the tree, a new training set was created that extracted every subtree and its sentiment label as a new data point. The number of iterations of training was determined by a maximal number of iterations, which was set to 10^6 for the BoW and 10^5 for the LSTMs, with an early-stopping threshold that stopped the training when no improvement was made on the accuracy of the validation set for 10^5 and 5000 iterations, to avoid overfitting. Also, dropout layer was included to prevent overfitting for the LSTMs (Srivastava et al., 2014). Learning rate was set to $5e-4$ for the BoW models and $2e-4$ for the LSTMs, and these values were not tuned.

After training, the models were evaluated on five test sets, consisting of sentences of length 1-13, length 15-20, length 21-30, and length > 31, and one including all sentences. This allowed us to look into whether sentence length impacted the performance of the models. Accuracy was measured by counting how many times the predicted label was the correct label for that sentence, divided by the total number of evaluations.

5 Results and Analysis

Figure 1 shows that the LSTM models, which use word order, perform better overall than the BoW models, not using word order. When aggregating the BoW models and the LSTM models, the former ended up with an average accuracy of 37.9% and the latter with 45.1%, suggesting that including word order in the sentence representation is helpful for better performance on a sentiment analysis task, answering **RQ1**. This result is not surprising since word order contains important information to judge the sentiment of a sentence, i.e. *"not nice"* is clearly of negative sentiment while for *"nice not"* it is less clear what the sentiment might be. We noticed that finetuning the pre-trained word embeddings for the N-ary LSTM decreased its performance, which dropped below the accuracy of the pt deepcBoW model. See the Appendix for an additional Figure showing this. Furthermore, note the big difference in performance for the not pre-trained deep BoW model compared to the pre-trained deep BoW model, which achieved accuracies of 37.6% and 43.1% respectively.

Additionally, Figure 1 shows that the Tree LSTM outperforms the regular LSTM: the average accuracy of the regular LSTM was 43.9% while for

[2]Please refer to Appendix A for specific formulas of LSTM and tree-LSTM models.

[3]7, 21 and 42

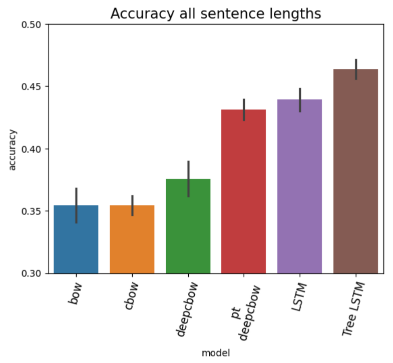

Figure 1: Accuracies for BoW models and two LSTM versions; the regular LSTM and the Tree LSTM. Accuracy is averaged over three seeds, error bar is std.

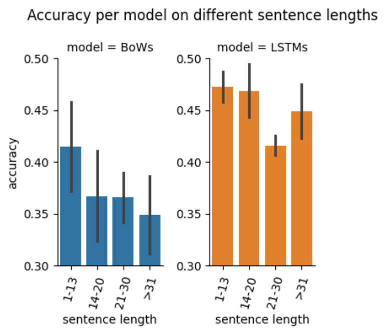

Figure 2: Accuracy of the BoW models and the (Tree) LSTM on different sentence lengths. The accuracies are averaged over three seeds and the different versions of the models, the error bar is std.

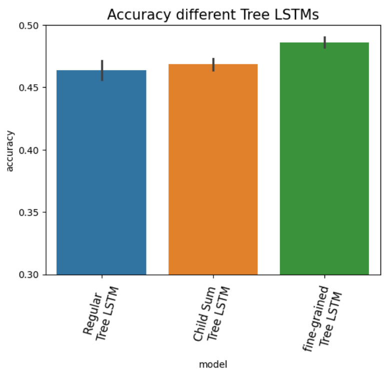

Figure 3: Accuracies for different Tree-LSTM models. Accuracies are averaged over three different seeds, the error bar is std.

the Tree LSTM it was 46.4%. This allows us to answer **RQ2**: performance increases by approximately 2% when tree structures are encoded during model training. Since hierarchical data can separate main clauses from subclauses, allowing one to obtain information on what part, and thus what information on sentiment in the sentence, is more important.

Figure 2 gives insight into model performance on different sentence lengths. Aligning with our expectations, the LSTM models perform better on longer sentences than the BoW models. For all the BoW models, the performance on sentences of lengths 1-13 was better than sentences >13[4]. These results equip us to answer **RQ3**: LSTM models are more robust for different lengths of sentences, while BoW models perform better on short sentences and worse on longer ones. This suggests that for longer sentences, word order is more important to classify sentiment than for short sentences. This can be investigated by looking into examples of shorter sentences and long sentences. An example sentence from the test data that shows this is *"if you're looking for comedy to be served up, better look elsewhere."*, where word order is needed for sentiment analysis (very negative), in comparison to a short sentence *"Extremely dumb"*, where word order is less useful. By investigating Figure 3, **RQ4** can be answered. Using more fine-grained sentiment data improved the model's performance slightly, resulting in an accuracy score of 46.4% for the regular Tree LSTM and 48.6% for the Tree LSTM trained on data with more finegrained sentiment information. This shows that adding lower-level information on sentiment in the training data is a fruitful manner to obtain better performance.

Lastly, also observable in Figure 3, the Child-Sum tree and the N-ary tree differ barely in performance, with accuracy scores of 46.8% and 46.4% respectively. So, contrary to what was expected, child node order information is not important for the sentiment analysis task, answering **RQ5**. This might be explained by the fact that differences between the performance of the Child-Sum and the N-ary tree found by Tai et al. (2015) were most likely due to a smaller set of training data for the Child-Sum tree, since in our experiment the training sets were the same for both trees.

6 Conclusion

From these experiments, we conclude that word order is overall important for sentiment analysis, as LSTM models outperform BoW models on all lengths, especially on longer sentences. Furthermore, having hierarchical information also improves performance, especially when node-level sentiment information is used. These conclusions align with previous literature. A somehow surprising observation is that the child-sum tree LSTM performed similarly to the N-ary tree LSTM, thus the child node order is not as important as expected. However, the performance difference reported in Tai et al. (2015) is also insignificant.

For future work, it would be interesting to look into the performance of for example transformer models on a sentiment analysis task, or other more state-of-the-art models.

[4]An additional figure separating the BoW and LSTM models can be found in the Appendix.

References

James Bradbury. 2017. Recursive neural networks with pytorch.

Kyunghyun Cho. 2015. Natural language understanding with distributed representation. *ArXiv*, abs/1511.07916.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

I. J. Good. 1952. Rational decisions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 14(1):107–114.

Klaus Greff, Rupesh Srivastava, Jan Koutník, Bas Steunebrink, and Jürgen Schmidhuber. 2015. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28.

Zellig S. Harris. 1954. Distributional structure. *WORD*, 10(2-3):146–162.

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

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. *arXiv preprint arXiv:1503.02510*.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher 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, Doha, Qatar. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew 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, Seattle, Washington, USA. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

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 and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *International conference on machine learning*, pages 1604–1612. PMLR.

A Appendix

A.1 LSTM and tree-LSTM formulas

A.1.1 LSTM

$$
\begin{aligned}
\boldsymbol{i_t} &= \sigma(W_i \boldsymbol{x_t} + R_i \boldsymbol{h_{t-1}} + \boldsymbol{b_i}) && \textbf{input gate}\\
\boldsymbol{f_t} &= \sigma(W_f \boldsymbol{x_t} + R_f \boldsymbol{h_{t-1}} + \boldsymbol{b_f}) && \textbf{forget gate}\\
\boldsymbol{g_t} &= \tanh(W_g \boldsymbol{x_t} + R_g \boldsymbol{h_{t-1}} + \boldsymbol{b_g}) && \textbf{candidate}\\
\boldsymbol{o_t} &= \sigma(W_o \boldsymbol{x_t} + R_o \boldsymbol{h_{t-1}} + \boldsymbol{b_o}) && \textbf{output gate}\\
\boldsymbol{c_t} &= \boldsymbol{f_t} \odot \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \odot \boldsymbol{g_t} && \textbf{updated cell state}\\
\boldsymbol{h_t} &= \boldsymbol{o_t} \odot \tanh(\boldsymbol{c_t}) && \textbf{updated hidden state}
\end{aligned}
$$

A.1.2 n-ary tree LSTM

$$
\begin{aligned}
\boldsymbol{i_j} &= \sigma(W_i + \sum_{l=1}^{N} R_l^i \boldsymbol{h_{jl}} + \boldsymbol{b_i}) && \textbf{input gate}\\
\boldsymbol{f_{jk}} &= \sigma(W_f + \sum_{l=1}^{N} R_{kl}^i \boldsymbol{h_{jl}} + \boldsymbol{b_i}) && \textbf{forget gate}\\
\boldsymbol{u_j} &= \tanh(W_u + \sum_{l=1}^{N} R_l^u \boldsymbol{h_{jl}} + \boldsymbol{b_i}) && \textbf{candidate}\\
\boldsymbol{o_j} &= \sigma(W_o + \sum_{l=1}^{N} R_l^o \boldsymbol{h_{jl}} + \boldsymbol{b_i}) && \textbf{output gate}\\
\boldsymbol{c_j} &= \boldsymbol{i_j} \odot \boldsymbol{u_j} + \sum_{l=1}^{N} \boldsymbol{f_{jl}} \odot \boldsymbol{c_{jl}} && \textbf{updated cell state}\\
\boldsymbol{h_j} &= \boldsymbol{o_j} \odot \tanh(\boldsymbol{c_j}) && \textbf{updated hidden state}
\end{aligned}
$$

A.1.3 Child-sum tree LSTM

$$
\begin{aligned}
\boldsymbol{\tilde{h}_j} &= \sum_{k \in C(j)} \boldsymbol{h_k} && \textbf{previous hidden states}\\
\boldsymbol{i_j} &= \sigma(W_i \boldsymbol{x_j} + R_i \boldsymbol{\tilde{h}_j} + \boldsymbol{b_i}) && \textbf{input gate}\\
\boldsymbol{f_{jk}} &= \sigma(W_f \boldsymbol{x_j} + R_f \boldsymbol{h_k} + \boldsymbol{b_f}) && \textbf{forget gate}\\
\boldsymbol{u_j} &= \tanh(W_u \boldsymbol{x_j} + R_u \boldsymbol{\tilde{h}_j} + \boldsymbol{b_u}) && \textbf{candidate}\\
\boldsymbol{o_j} &= \sigma(W_o \boldsymbol{x_j} + R_o \boldsymbol{\tilde{h}_j} + \boldsymbol{b_o}) && \textbf{output gate}\\
\boldsymbol{c_j} &= \boldsymbol{i_j} \odot \boldsymbol{u_j} + \sum_{k \in C(j)} \boldsymbol{f_{jk}} \odot \boldsymbol{c_k} && \textbf{updated cell state}\\
\boldsymbol{h_j} &= \boldsymbol{o_j} \odot \tanh(\boldsymbol{c_j}) && \textbf{updated hidden state}
\end{aligned}
$$

A.2 Figures

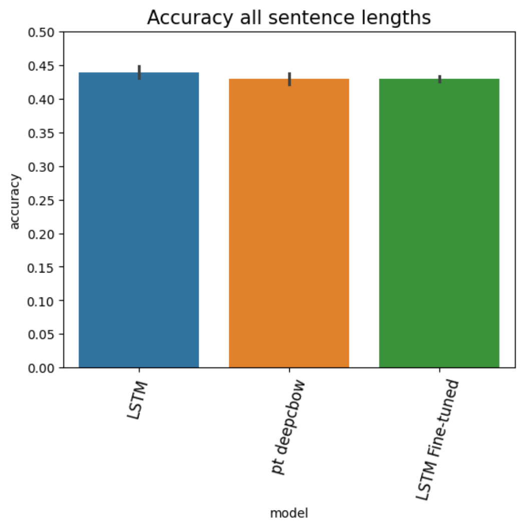

Figure 4: Accuracy of the LSTM, pre-trained DeepCBoW, LSTM finetuned. The accuracies are averaged over three seeds, the error bar shows the standard deviation from the mean. We can see that the accuracy of the fine-tuned LSTM is lower than the LSTM that was simply using the pre-trained word embeddings.

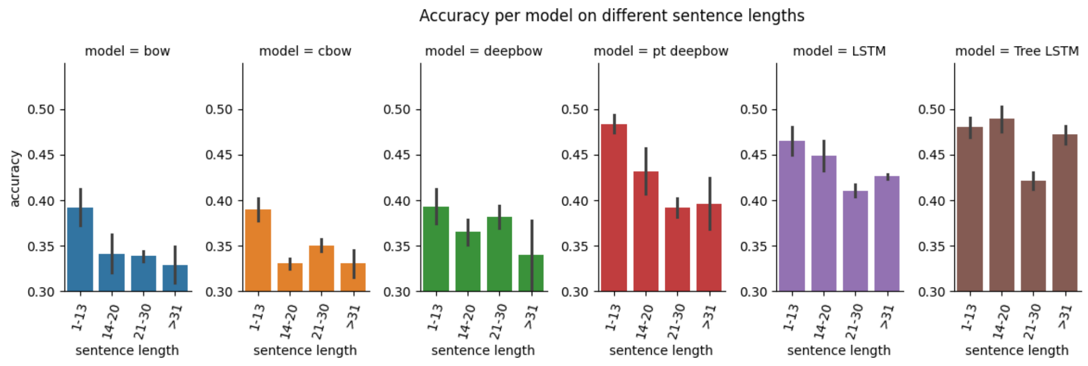

Figure 5: Accuracy of the BoW, CBoW, DeepCBoW, pre trained DeepCBoW, LSTM, and Tree LSTM on different sentence lengths. The accuracies are averaged over three seeds, the error bar shows the standard deviation from the mean.