NLP 1 - Practical II

Erik Jenner
13237896

Tom Lieberum
13253042

1 Introduction

A core competency in text comprehension is to gauge the overall sentiment expressed by the text at hand. The goal of sentence sentiment analysis is, given a sentence, to predict the sentiment score of this sentence. Due the ambiguity of language, and problems like negation, irony and sarcasm, this is a difficult enough problem to warrant considerable investigation.

One popular approach to this problem has been to first build a representation of a sentence and then use this representation for classification. A simple but surprisingly effective method are so-called bag-of-word (BOW) models (Pang et al., 2002) and more complex versions such as the continuous BOW (CBOW) (Mikolov et al., 2013a) and Deep CBOW models.

In recent years, recurrent neural models like the long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) have resurfaced, gaining increased popularity as available computation and datasets have become sufficiently large to train them.

Several modifications to the standard LSTM have been proposed. Of particular interest to the language processing community is the Tree-LSTM (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015), which operates on a tree structure instead of a linear sequence, which makes it very suitable to be applied on parse trees of sentences.

These models use different amounts of structure present in the data: BOW models treat sentences as unordered sets, LSTMs use their sequential structure, and Tree-LSTMs use their full tree structure. They also have very different capacities, for example the difference between (one-layer) CBOWs and Deep CBOWs is depth plus added non-linearities. Additionally, there are options for improving performance such as pretrained embeddings and node-level supervision. The goal of this report is to establish the relative importance of these various factors for the sentiment classification task, using the Stanford Sentiment Treebank (SST) (Socher et al., 2013).

We begin by benchmarking a simple BOW model and then test gradually more advanced methods, determining at each step how large the improvements from specific sources are. We increase the model capacity by moving to CBOWs, showing that this yields a slight performance gain but that the further increase in depth from deep CBOWs does not help. We get the most significant performance boost by using pretrained vector embeddings rather than learning them from scratch. Then we test an LSTM model, which leads to another very small improvement, but we show that almost the same improvement can be achieved on randomly shuffled sentences, which means that the word-order is not the important factor in our case. Finally, we analyze a Tree-LSTM on this task, with a similar performance as the LSTM. Therefore, knowledge of the syntax tree apparently also does not help much. Using node-level supervision may have some beneficial effects on performance but not a large one. In addition, we compare binary Tree-LSTMs to childsum Tree-LSTMs and find that the performance is almost identical.

These models are all well-known individually, and it is also no surprise that the more advanced models such as Tree-LSTMs tend to perform best. However, an investigation of what the important factors behind this success are is to the best of our knowledge missing from the literature.

2 Background

2.1 BOW models and embeddings

Bag-of-word models learn a representation of each word in the vocabulary. In the simplest version,

the embedding dimension is equal to the number of sentiments. To classify a sentence, each word in the sentence is mapped to its embedding vector and the predicted score is computed as the argmax over the sum of all embeddings in a sentence plus a bias vector representing the prior belief over sentiments. If there are V words in the vocabulary and c sentiment classes, then the model learns a matrix $\mathbf{W} \in \mathbb{R}^{V \times c}$ and a bias $\mathbf{b}$ to compute the predicted sentiment as

$$\hat{c} = \text{argmax}_{i \in [c]} \, \mathbf{W}^T \mathbf{v} + \mathbf{b},$$

where $\mathbf{v}$ is the one-hot word vector and $\mathbf{b}$ encodes prior beliefs over the sentiment distribution.

This model can be extended by introducing an intermediate hidden linear mapping between the initial embedding and the output embedding. These models are referred to as continuous BOW (CBOW) models and can be interpreted as an embedding followed by a logistic regression classifier. Further improvements can be gained by using a multi-layer perceptron as the classifier instead, which we refer to as Deep CBOW models.

Once a BOW model (or a different embedding model) is trained, the first linear mapping, i.e. $\mathbf{W}$, can be saved and re-used for other applications. This has the benefit of not having to retrain an embedding for every new task, as well as being able to use an embedding which has been trained on a very large corpus. Two popular of such pretrained embeddings are Word2Vec (Mikolov et al., 2013b), which we use in this report, and GloVe (Pennington et al., 2014).

2.2 Long short-term memory (LSTM)

Long short-term memory networks are a type of recurrent neural network which have been originally proposed in Hochreiter and Schmidhuber (1997). They improve upon vanilla recurrent neural networks (RNNs) by introducing several gating mechanisms, allowing more complex updates of their internal states, as well as being less prone to vanishing gradients. In each iteration the LSTM computes masking scores to decide which information from its old state it should keep, which information from the input to add to its state and which information will form the output at that time step. LSTMs process a sentence sequentially from left to right, taking the current word embedding vector as input at the current time step to update their internal states. After processing a sentence, their hidden state at the last time step can be interpreted as a sentence representation and used for classification.

2.3 Tree-LSTM

Conventional LSTMs traverse the input sequentially. Since sentences exhibit syntactical structure which can be represented by parse trees, several formulations of so-called Tree-LSTMs have been proposed to exploit this tree structure (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015). The focus of this report lies on Binary Tree-LSTMs, a special case of N-ary Tree-LSTMs, as presented in (Tai et al., 2015).

Tree-LSTMs traverse a given tree from leaves to parents. In principle, they can take an input at every node but in our case, the only inputs are at the leaves. To compute the hidden and cell state of a given node, the hidden states of its children are combined, where each hidden state is multiplied with a different matrix. This enables the Binary Tree-LSTM to weight left and right children differently. The model also computes a different forget mask for each child cell state. The update equations are given in eqs. (1) to (4), where the indices l, r refer to the different matrices and vectors of the left and right child node respectively. We compare the binary model to the childsum Tree-LSTM which adds the hidden states of the children before processing them with a single parameter matrix. This means that $\mathbf{U}_l = \mathbf{U}_r$ in eqs. (1) and (2). The childsum LSTM does however still compute two different forget masks, so that for it eqs. (3) and (4) are the same as for the binary Tree-LSTM.

$$\mathbf{g}_i = \sigma\left(\mathbf{W}^{(i)}\mathbf{x} + \mathbf{U}_l^{(i)}\mathbf{h}_l + \mathbf{U}_r^{(i)}\mathbf{h}_r + \mathbf{b}^{(i)}\right) \quad (1)$$

$$\mathbf{u} = \tanh\left(\mathbf{W}^{(u)}\mathbf{x} + \mathbf{U}_l^{(u)}\mathbf{h}_l + \mathbf{U}_r^{(u)}\mathbf{h}_r + \mathbf{b}^{(u)}\right) \quad (2)$$

$$\mathbf{c} \leftarrow \mathbf{g}_{\text{in}} \odot \mathbf{u} + \mathbf{g}_{\text{forget},l} \odot \mathbf{c}_l + \mathbf{g}_{\text{forget},r} \odot \mathbf{c}_r \quad (3)$$

$$\mathbf{h} \leftarrow \mathbf{g}_{\text{out}} \odot \tanh(\mathbf{c}) \quad (4)$$

Here, $\mathbf{g}_i$ refers to the different gates, where the index i can be `in`, `out` or `forget` (with an additional l or r indexing the forget gates). A vanilla LSTM can be seen as a special case of these equations, by always setting the right hidden and cell state to zero and interpreting the input sequence as a "unary tree".

3 Models

All our models map a sentence to $\mathbb{R}^c$ in some way, with $c = 5$ the number of classes. In all cases, these outputs are interpreted as logits for the different sentiments, meaning we transform them into probabilities with the softmax function and then use the cross entropy loss.

For the Deep CBOW model, we use a 3-layer MLP for classification, with tanh non-linearities and no dropout.

Our LSTMs consist of only a single cell (i.e. layer). Before a word is passed to the LSTM, it is embedded with the pretrained Word2Vec embedding. After processing a sentence, the final hidden state $\mathbf{h}_T$ is projected via a linear layer to the output space to model class logits. We use a dropout probability of 0.5 before this final linear layer, in both LSTM and TreeLSTM, but do not use dropout inside the LSTM cells.

4 Experiments

During training, we evaluated the model every n steps on a dedicated validation set, where $n = 1000$ for BOW models and $n = 50$ for LSTM models (since one LSTM step consists of an entire minibatch). Training was stopped as soon as the validation accuracy did not improve for 20 such evaluations. Final performance was then evaluated on a test set, using the model that achieved the highest validation accuracy. We used a learning rate of 0.0003 and an embedding dimension of 300 for all models. All BOW models which have one or more hidden layers use a hidden dimension of 100 and all LSTM models use a hidden dimension of 150[1].

Each model was trained three times with different random seeds and all reported accuracies are averaged over these three runs. Reported errors are the standard deviations.

For the LSTM and Tree-LSTM model, we only used the pretrained Word2Vec embeddings and left them fixed during training. For the Deep CBOW model, we compare these same fixed embeddings with ones learned from scratch. The other BOW models all learn the embeddings from scratch.

In addition to the straightforward performance comparison of the models, we performed several other experiments. Firstly, we analyzed the importance of word order for the LSTM model by randomly shuffling words inside a sentence. We first evaluated the normally trained model on the shuffled test set, and then also retrained from scratch with shuffled sentences.[2]

Secondly, we attempted to improve the Tree-LSTM performance by adding all non-singleton subtrees for each sentence to the training set. This has a very similar effect to supervising on a node-level instead of only on the sentence-level[3].

Finally, we tested the performance of each final trained model for different sentence lengths.

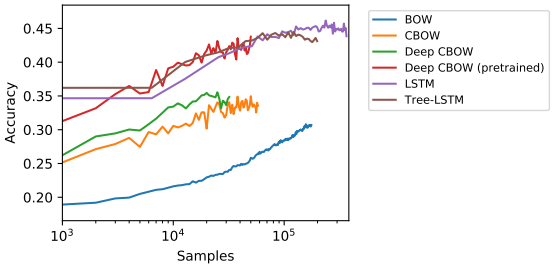

Figure 1: Mean validation accuracies for the different methods over training time. We use number of samples as a measure of training progress to make the comparison between BOW models (which use single samples for each step) and LSTM models (which use minibatches) more meaningful. The shaded areas represent $\pm 1\sigma$, calculated over the three seeds.

5 Results and Analysis

Table 1 contains the results of the different methods we compare. The development of validation accuracy during training is shown in fig. 1 and loss and accuracy curves of all runs are in the appendix.

Introducing the larger embedding space of CBOW yields a slight improvement over BOW but the further increase in model capacity of Deep CBOW models does not help. Presumably this is because the CBOW model is already overfitting to the training set and this effect is only exacerbated[4]. Using pretrained embeddings results in much better performance. The pretrained embeddings were trained on a significantly larger corpus and are thus probably better able to capture meaningful differences between the words. Additionally, the reduced

[1] This was prescribed by the notebook handed to us.

[2] For BOW-based model, word order does not have any effect by design, while for the Tree-LSTM, the structure of the syntactic tree is relevant rather than the word order, so we ran this experiment only on the LSTM.

[3] In the limiting case of full batch gradient descent the methods would be equivalent.

[4] We observed training accuracies upwards of 70% for the CBOW already

number of parameters makes overfitting less likely which might improve test performance.

The LSTM models are able to achieve a slightly better performance than the BOW models. The LSTM performance drops significantly if the word order is shuffled during test time but the drop is much less severe if it already encounters shuffled sentences in its training set. The respective deviations are roughly 4.37σ and 2.03σ.[5]

That shuffling degrades performance so little could imply that word order does not play a large role for this task. The small difference between the Deep CBOW and the LSTM with training-shuffling could be connected to the different sizes of the models or might simply be chance.

However also note that our LSTM has only a single layer and that the training set is not particularly large. So it is very plausible that word order is relevant for the task of sentiment classification, but that the LSTM is simply unable to make much use of it in our case.

The Tree-LSTM performs similarly to the LSTM, with no significant difference in accuracy. We already saw that the importance of word order for the LSTM is relatively small, apparently using the full syntactic structure does not help any more.

We also retrained with all non-singleton subtrees included in the training set, but got a result statistically indistinguishable from the original task. We note that this could be partially due to an outlier in the normal Tree-LSTM run, which achieved 48.3% test accuracy despite having only 44.7% validation accuracy. Running the experiments with more seeds would lead to smaller uncertainty and might make the difference significant.

Finally, the Childsum Tree-LSTM achieves virtually the same result as the Binary Tree-LSTM, suggesting that the ability to differentiate between left and right child does not confer a meaningful advantage in this task.

For all models, the performance was almost uniform across sentence lengths. Figure 2 shows this for the LSTM, the other plots are qualitatively similar and can be found in the appendix.

6 Conclusion

We have compared BOW-based models of various capacity and different LSTM-based architectures, also analyzing the effects of using pretrained word embeddings, of word order, and of augmenting the training set with subtrees. We found that the most significant improvement was achieved by using pretrained word embeddings, presumably because the size of our data set was an important limiting factor. Using LSTMs helped surprisingly little, which might be explained by the fact that word order does not seem to be very important for this task, or might simply mean that the benefits of LSTMs would only become apparent with larger capacities and larger training sets.

[5] Assuming Gaussian error propagation.

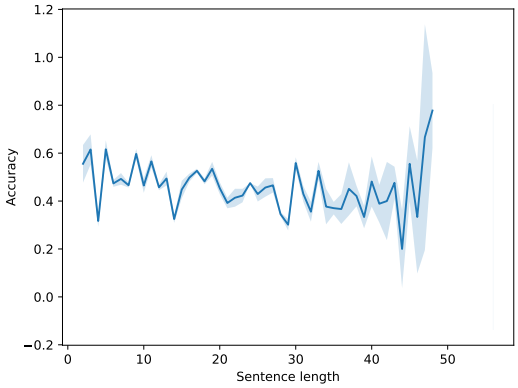

Figure 2: LSTM accuracy for sentences of different lengths. Shaded area shows the standard deviation across the three seeds

BOW	31.9 ± 1.6
CBOW	37.0 ± 1.1
Deep CBOW	36.6 ± 2.6
Pretrained Deep CBOW	43.7 ± 0.7
LSTM	45.9 ± 0.4
Tree-LSTM	46.7 ± 1.2
test-shuffle	43.1 ± 0.5
all-shuffle	44.6 ± 0.5
node supervision	47.6 ± 1.0
Childsum Tree-LSTM	46.8 ± 1.1

Table 1: Accuracies in % of the various models for sentiment classification

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*,

9(8):1735–1780.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 10–19, Denver, Colorado. Association for Computational Linguistics.

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

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, pages 3111–3119. Curran Associates, Inc.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

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.

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 Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1604–1612. JMLR.org.

A Results of all training runs

We now show the training loss and validation accuracy curves of all our training runs.

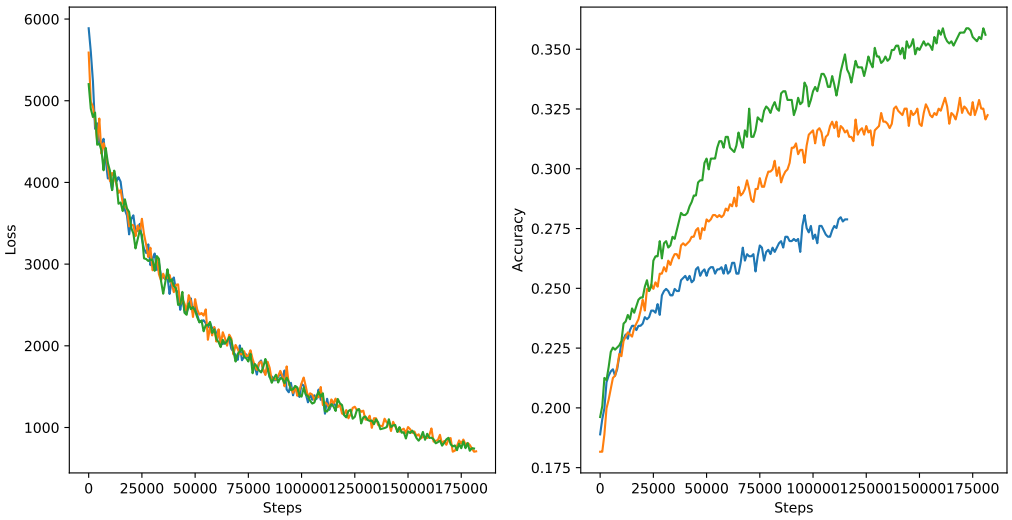

Figure 3: BOW

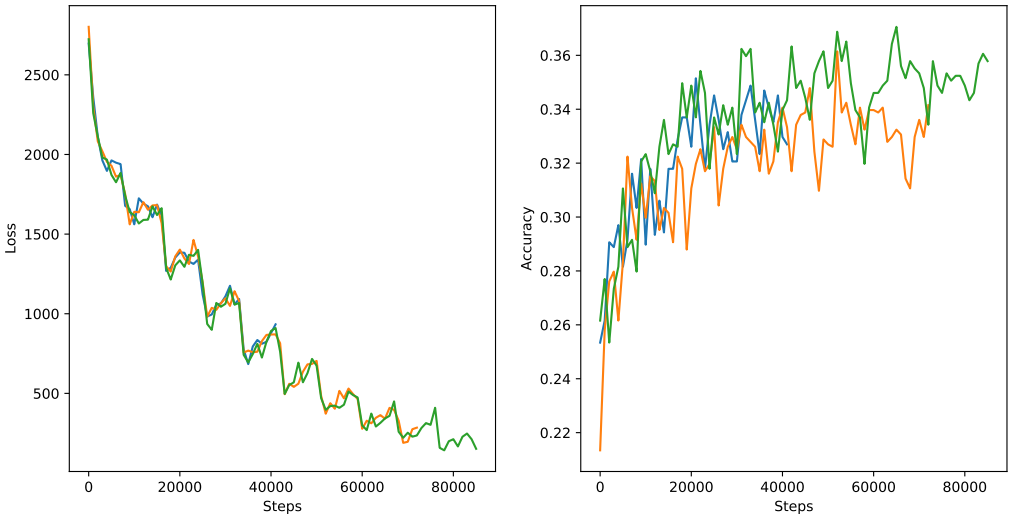

Figure 4: CBOW

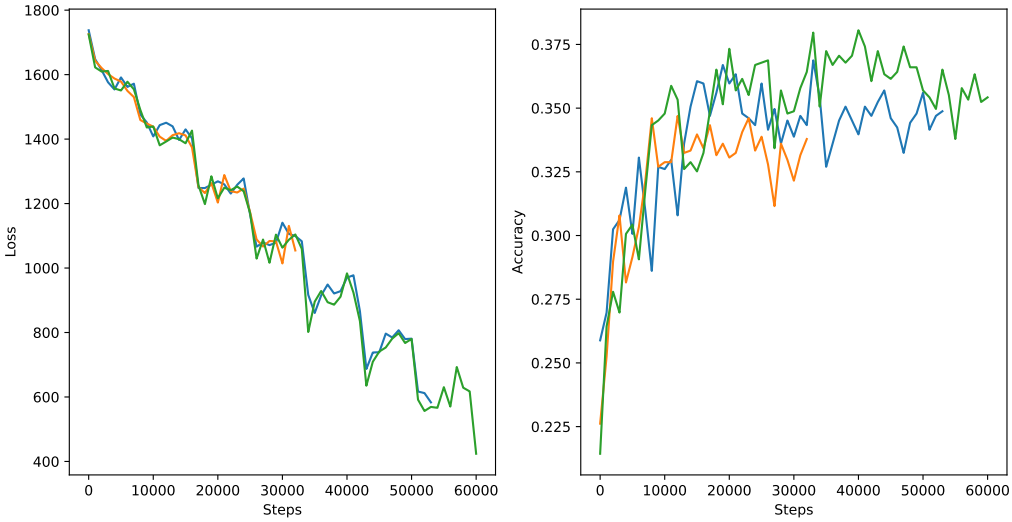

Figure 5: Deep CBOW

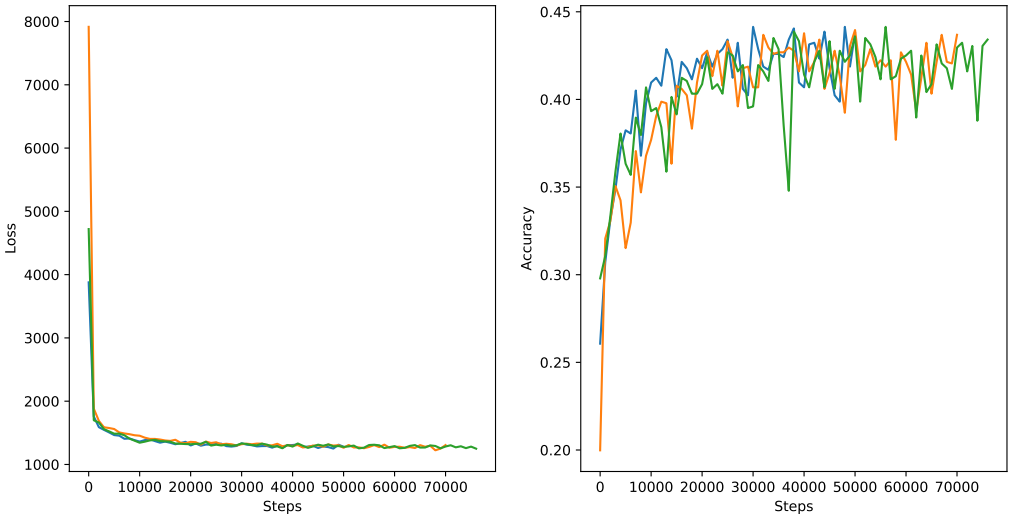

Figure 6: Deep CBOW (pretrained)

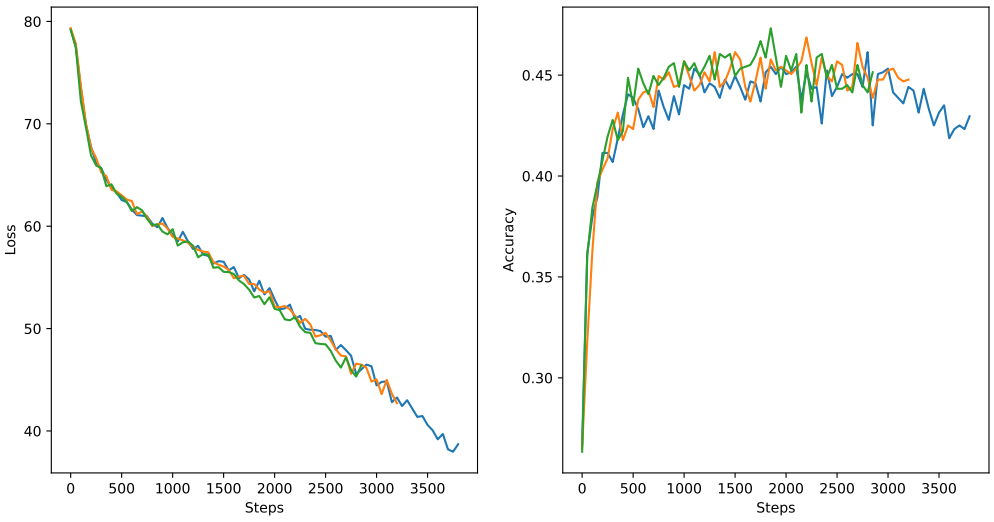

Figure 7: LSTM

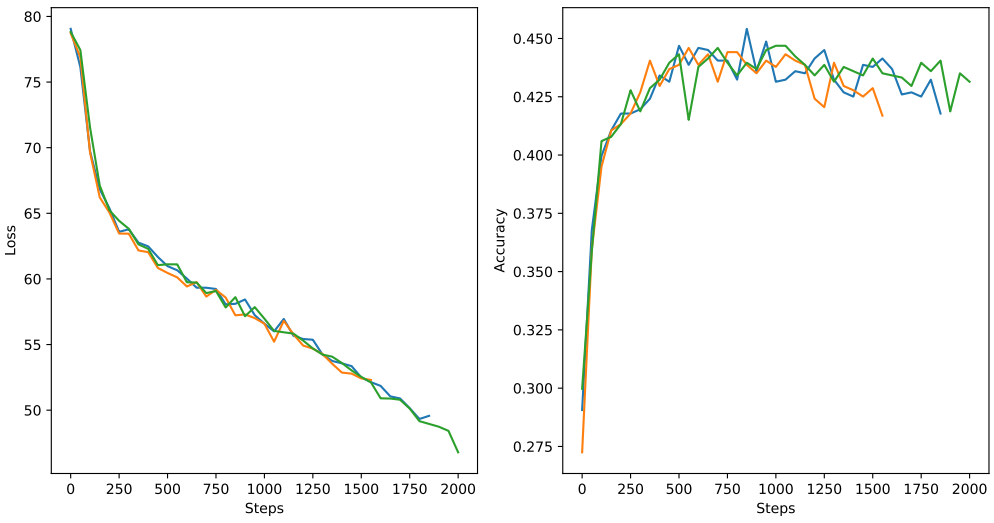

Figure 8: Binary Tree-LSTM

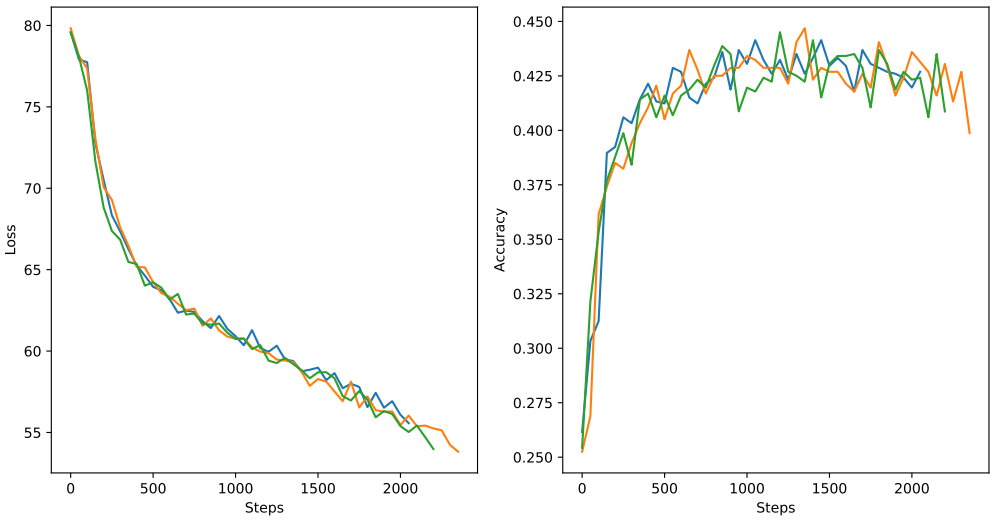

Figure 9: LSTM with shuffled sentences

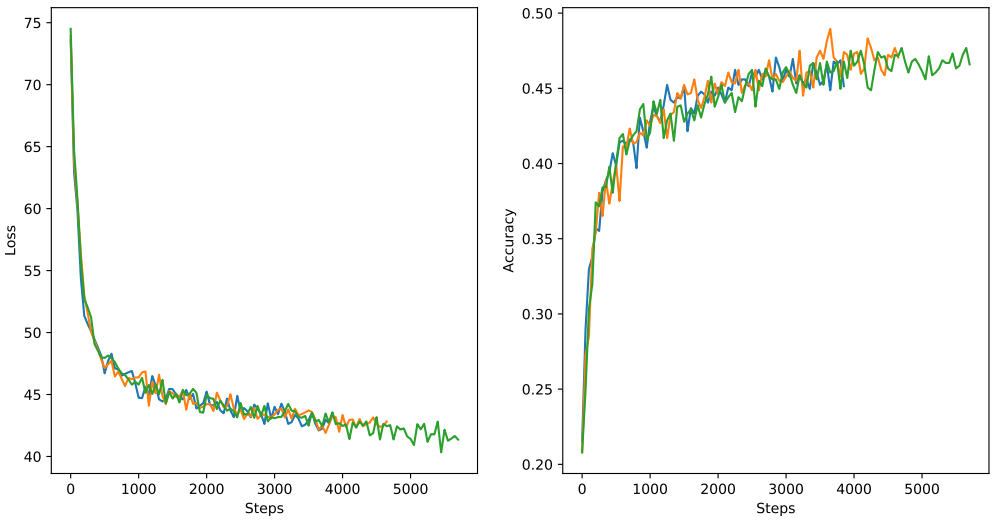

Figure 10: Binary TreeLSTM with subtrees in training set.

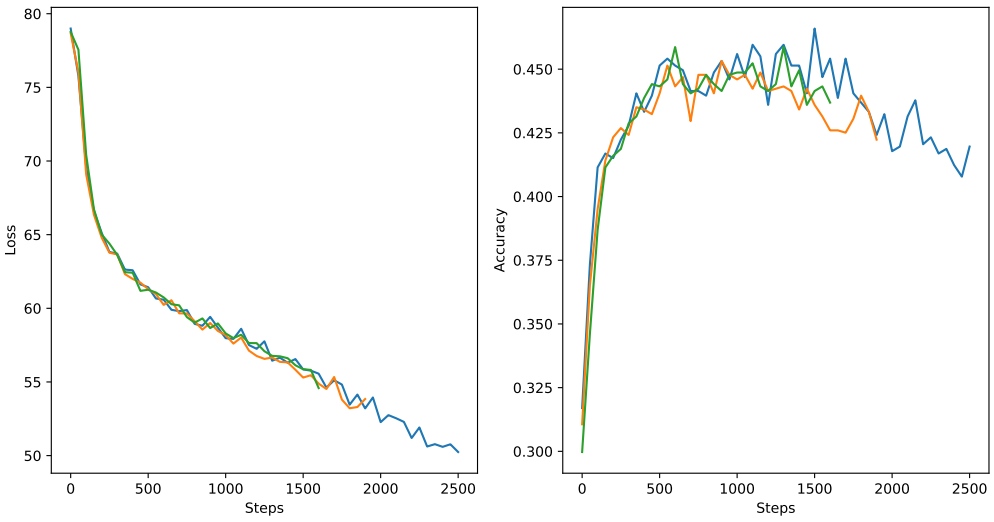

Figure 11: Childsum Tree-LSTM

B Sentence length experiments

Now we show all the accuracy plots for the sentence length experiments. We used the normally trained models (with training curves in the previous section), the plots here show the test accuracies of these models grouped by sentence lengths.

The shaded areas illustrate the standard deviation across the three different seeds. The fluctuations between sentence lengths are often much larger than these standard deviations – however that does not mean that sentence length itself has a significant effect. Rather, the test set may simply happen to contain more easy to classify sentences of one length than of another. Because it is not that large, such differences are to be expected.

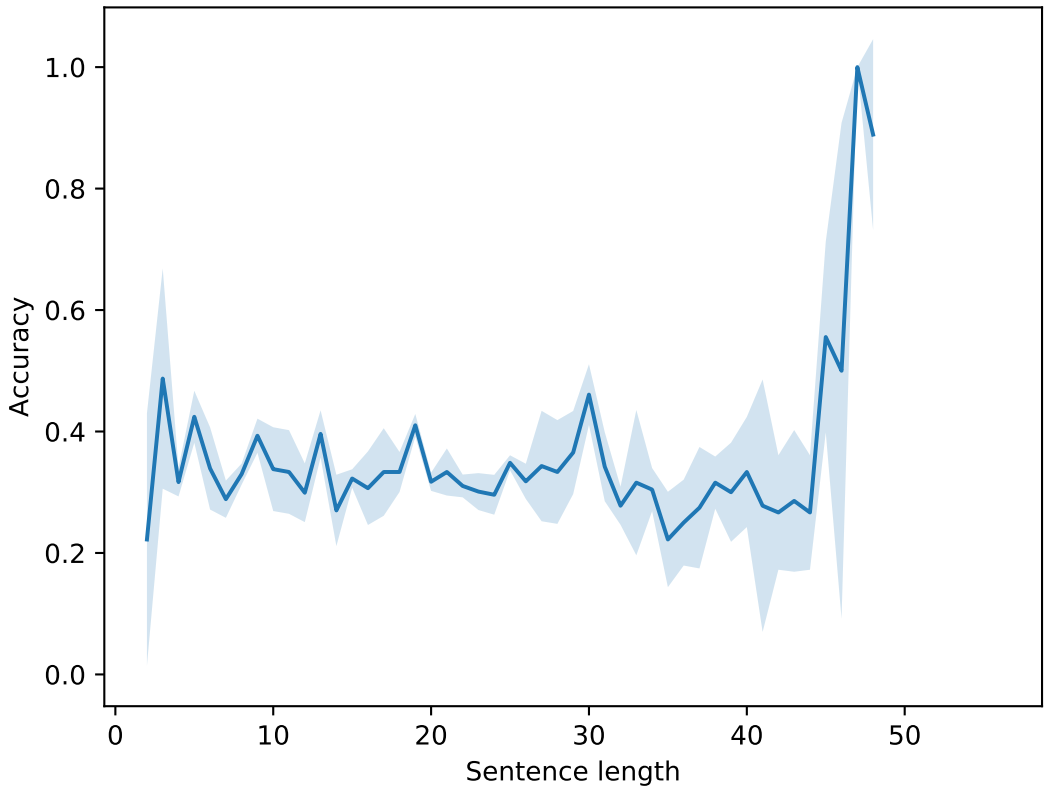

Figure 12: BOW

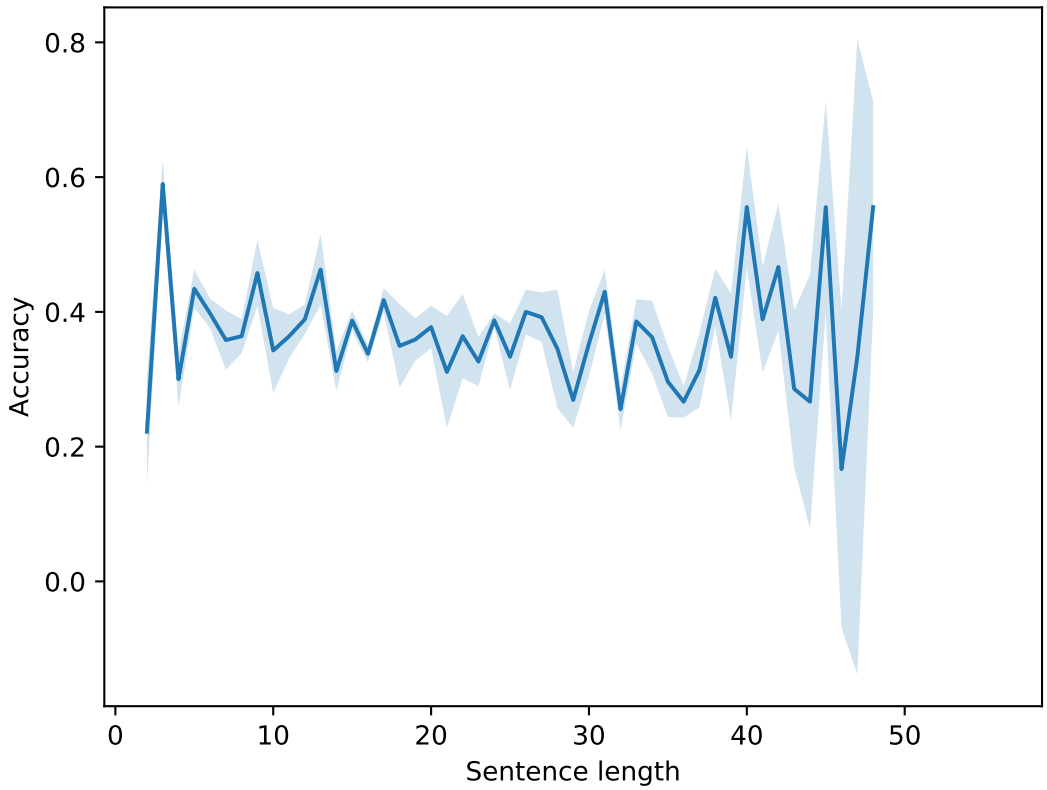

Figure 13: CBOW

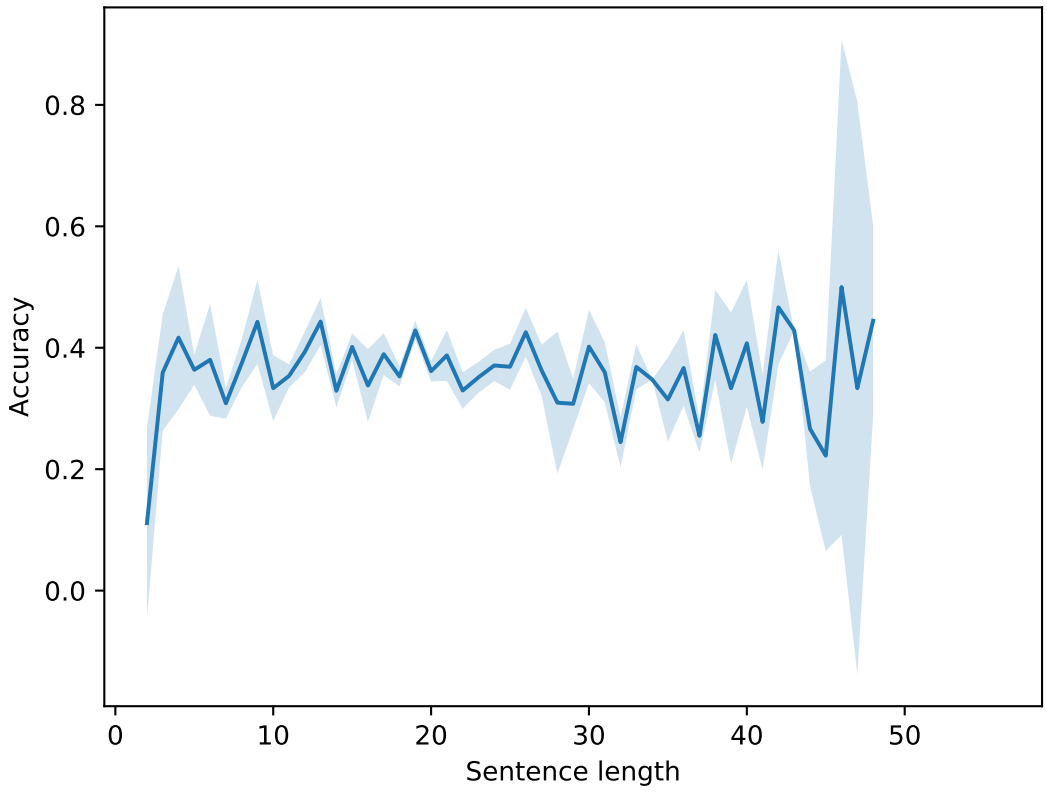

Figure 14: Deep CBOW

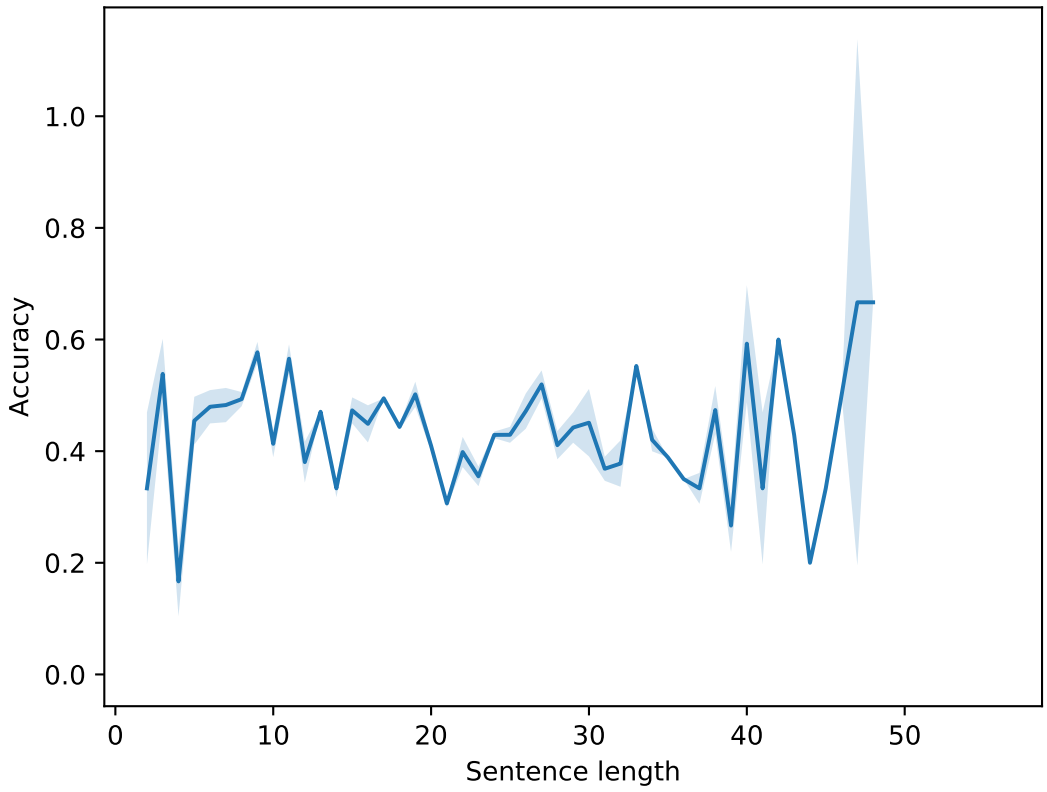

Figure 15: Deep CBOW (pretrained)

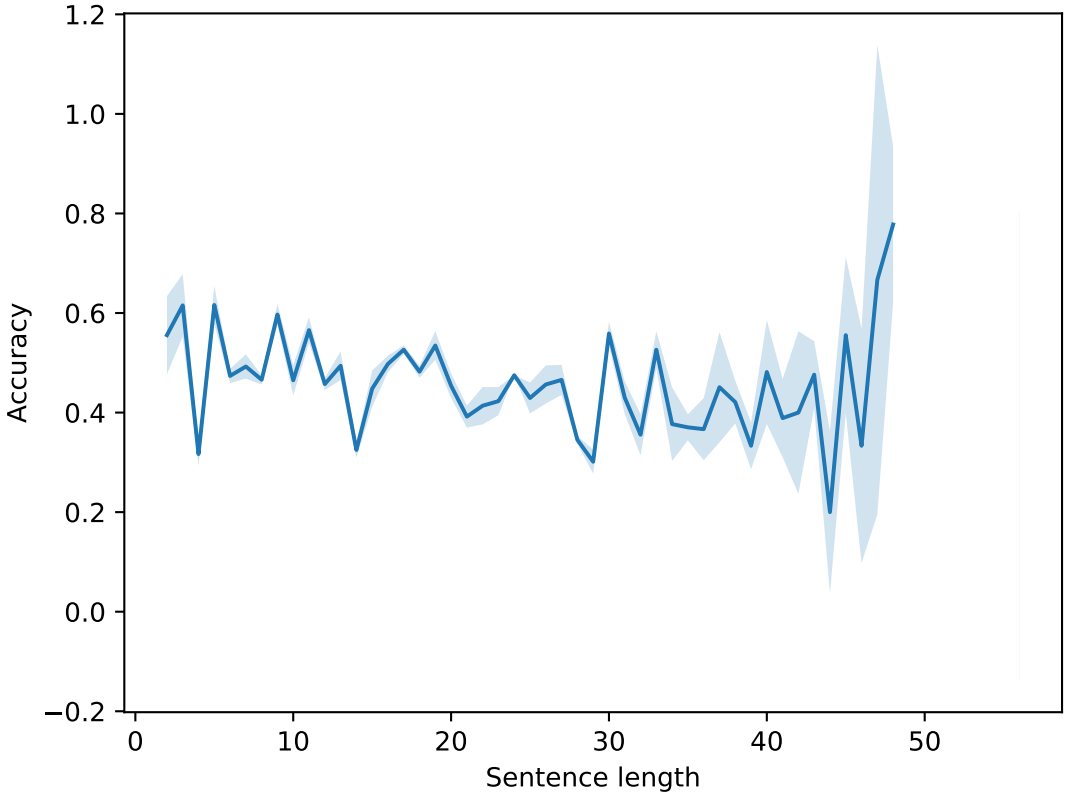

Figure 16: LSTM

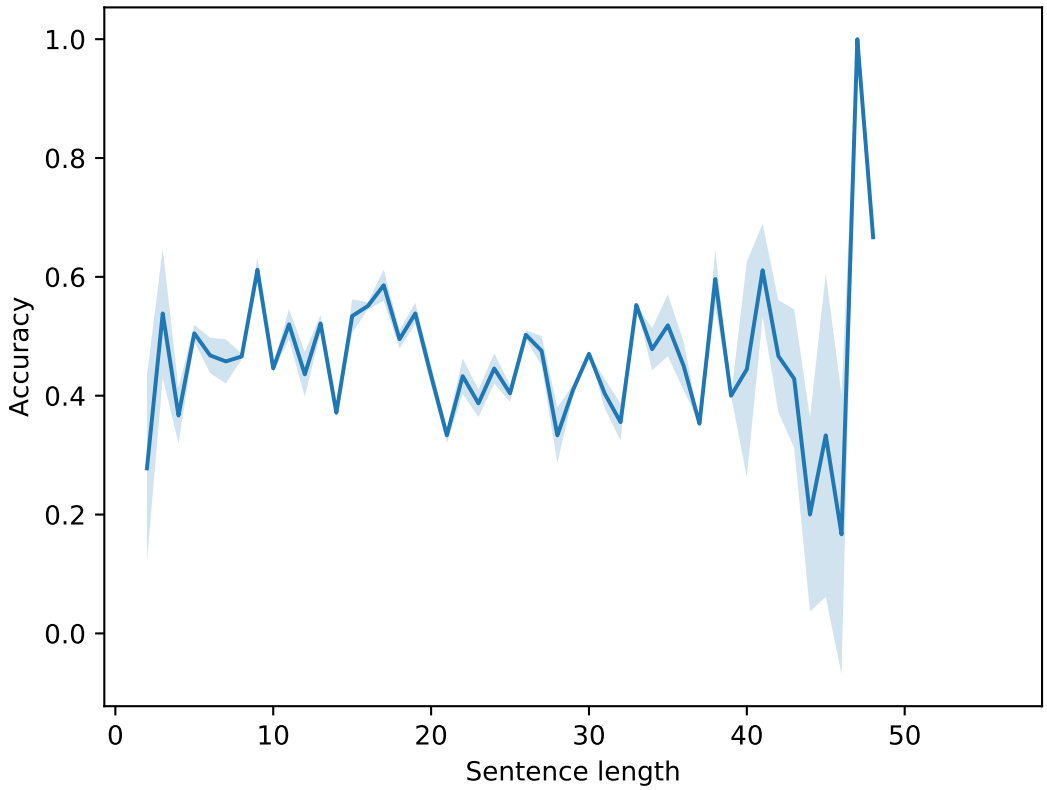

Figure 17: Tree-LSTM