NLP 1- Practical II
Representing Sentences with Neural Model

Kamlesh Sahoo
12821721
kamlesh.sahoo@uva.nl

Manolis Rerres
13439022
manolis.reres@uva.nl

1 Introduction

Natural language sentences consist of words or phrases, follow grammatical rules, and convey complete semantic information. Compared with words and phrases, sentences have more complex structures, including both sequential and hierarchical structures, which are essential for understanding sentences. In NLP, representing sentences is critical for various downstream tasks like summarization and sentiment analysis, since they capture a lot of prior knowledge about the domain of interests and lead to improved performance.

Our goal here is to learn sentence representations from *Stanford Sentiment Treebank* (Socher et al., 2013). This is a benchmark dataset widely used for neural model evaluations with gold-standard sentiment labels for every parse tree constituent, from sentences to phrases to individual words. The learned representations can then be used to determine the sentiment contained in the text and categorise it into an n-point scale, e.g., very good, good, satisfactory, bad, very bad. The task can be interpreted as a classification task where each category represents a sentiment.

A simple yet effective approach is to use the so called bag-of-word (BOW) (Pang et al., 2002) class of models and the more complex continuous BOW (CBOW) (Mikolov et al., 2013a) and Deep CBOW models. These models sum or average the vectors of the words in a sentence to get sentence representations. Main problems with these models are that they cannot capture the sequential and structural information in sentences which can be crucial for the downstream task.

A more recent approach inspired by deep learning models, is to represent sentences with deep neural networks like LSTMs. They have become increasingly popular because of their ability to encode sequential and dependency information in sentences. Tai et al., 2015 proposed a tree-LSTM model to introduce useful structural information from sentence parse trees.

Because of the number of possible alterations in each model, it is important to do a comparative study and understand the potential limitations, advantages, and disadvantages of these models. This report aims to do the same by comparing the models in terms of architecture and performance on the sentiment classification task.

Our methodology is to work from simpler to more complex models. This helps to not only determine what caused the performance gain in a model but also reveal possible trade-offs involved. We started with the simple BOW model which serves as our baseline for rest of our experiments. Increasing the number of word vector dimensions in CBOW and adding more hidden layers in Deep CBOW resulted in sequential performance gain which ultimately plateaued, marking the limit of BOW models. Given the low computation time and interpretable dimensions, vector mixture models like BOW were surprisingly effective. We observed significant performance boost when using pretrained word embeddings in contrast to building one from scratch. We then changed gears to test the more state-of-the art LSTMs which led to a minor improvement. However, we found insignificant loss in performance when testing LSTMs with randomly shuffled words in sentences, indicating that word-order is not an important factor. We then leveraged the tree structure of our corpus, with the Tree-LSTM model which gave a minor performance gain compared to LSTMs. A node level supervision by creating additional subtrees, did improve performance, but not by much. Finally, we compared a binary tree LSTM with the ChildsumTree variant and found almost identical performances.

2 Background

2.1 BoW models and Word Embeddings

Before feeding sentences to ML algorithms we need to convert them to a numeric form using *word embeddings*. Word embeddings are mathematical vectors with values encoding semantic information. Probably the most intuitive representation of a sentence is to combine these word vectors using a symmetric operation like vector addition. To classify a sentence, the embedding vectors of the constituent words are added along with a bias term.[1] Finally, the predicted score is computed as the $argmax$ over the sum. 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 bias $\mathbf{b}$ to compute the predicted sentiment as

$$\hat{c} = argmax_{i \in [c]} \mathbf{W}^T \mathbf{v} + \mathbf{b} \tag{1}$$

where $\mathbf{v}$ is the one-hot word vector. This model can be extended by introducing intermediate hidden linear mappings between the initial embedding and the output embedding. These models are referred as continuous BOW (CBOW) models. Further improvements can be achieved by using a multi-layer perceptron as the classifier instead, which we refer to as Deep CBOW models. An important use case is that once such a model is trained, the learned embedding matrix $\mathbf{W}$ can be re-used for other applications. Thus embeddings trained on very large corpus can be stored and retrieved when needed without having to retrain an embedding for every new task. Two such popular pretrained embeddings are Word2Vec (Mikolov et al., 2013b), which we use in this report, and GloVe (Pennington et al., 2014).

2.2 LSTM

Originally proposed by Hochreiter and Schmidhuber (1997), LSTMs have units or memory blocks in the recurrent hidden layer, which contains memory cells with self-connections storing the temporal state of the network. In addition to this the network has special multiplicative units called gates to control the flow of information in the network. Unlike conventional *RNNs*, the stored values aren't iteratively squashed over time which solves the vanishing gradient problem. Instead the model adaptively forgets or keeps the previous state information using the gating mechanisms. After embedding and several passes through the gates and activation functions, the data makes its way to the final output layer. The output layer consists of only an activation function for the prediction, and generally a softmax activation is used in most cases.

2.3 Tree LSTM

A limitation of the LSTM architectures is that they only allow for strictly sequential information propagation. Since sentences exhibit syntactical structure which can be represented by parse trees, several formulations of so-called Tree-LSTMs have been proposed that leverage this tree structure (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015). We specifically focus on Binary Tree-LSTMs, a special case of N-ary Tree-LSTMs, as presented in Tai et al., 2015. In a binary tree LSTM each node can have at most 2 children, with the forget gates parametrized in a way so that siblings affect each other. The update equations are given in Equation 2 to 5, where the indices l, r refer to the the left and right child node respectively. We compare the binary model to the childsum Tree-LSTM which does not take into account children order. Here the hidden states of the children are added before processing them with a single parameter matrix. Thus $\mathbf{U}_l = \mathbf{U}_r$ in Equation 2 and 3. The childsum LSTM does however still compute two different forget masks, so that Equation 4 and 5 remain the same as for the binary Tree-LSTM.

$$\mathbf{g}_i = \sigma\left(\mathbf{W}^{(i)}\mathbf{x} + \mathbf{U}_l^{(i)}\mathbf{h}_l + \mathbf{U}_r^{(i)}\mathbf{h}_r + \mathbf{b}^{(i)}\right) \tag{2}$$

$$\mathbf{u} = \tanh\left(\mathbf{W}^{(u)}\mathbf{x} + \mathbf{U}_l^{(u)}\mathbf{h}_l + \mathbf{U}_r^{(u)}\mathbf{h}_r + \mathbf{b}^{(u)}\right) \tag{3}$$

$$\mathbf{c} = \mathbf{g}_{in} \odot \mathbf{u} + \mathbf{g}_{forget,l} \odot \mathbf{c}_l + \mathbf{g}_{forget,r} \odot \mathbf{c}_r \tag{4}$$

$$\mathbf{h} = \mathbf{g}_{out} \odot \tanh(\mathbf{c}) \tag{5}$$

where $\mathbf{g}_i$ refers to the different gates and the index i denotes the different gates: *in*, *out* or *forget*.

3 Models

In all our models we map a sentence to $\mathbb{R}^c$ in some way, with $c = 5$ for the number of sentiment classes. The outputs are raw unnormalized prediction scores, often called *logits*, which are turned into probabilities using the softmax function. We use the cross entropy loss as our loss criterion and use the gradient-based optimization algorithm, Adam. For the Deep CBOW model, we

[1] a bias term accounts for any prior beliefs over the sentiment distribution

use a 3-layer MLP[2] 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 pre-trained Word2Vec embeddings. After processing a sentence, the final hidden state $\mathbf{h}_T$ is projected via a linear layer to the output space to get the sentiment class logits. We use a dropout probability of 0.5 before this final linear layer, in both LSTM and Tree-LSTM classifiers, and dropouts of 0.25 inside the Tree-LSTM cells.

4 Experiments

Before training our models we did some initial exploratory data analysis as shown in Figure 1 to check for imbalanced datasets. We found almost identical distributions across each sentiment class, which motivated our choice of using *accuracy* as evaluation metric. During training we evaluated the

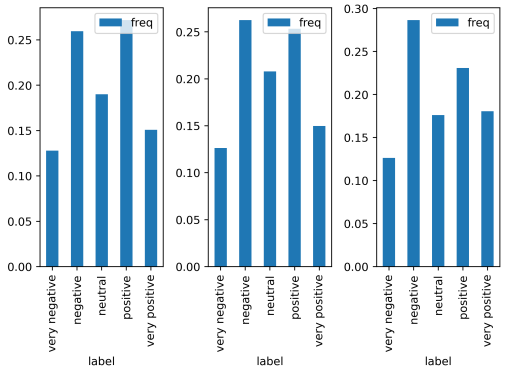

Figure 1: Relative frequency for each sentiment class in datasets **(Left)** Train **(Middle)** Dev(**Right)** Test

model every n steps on the dev set, with $n = 1000$ for BOW models and $n = 50$ for the LSTMs.[3]. As far as training convergence is concerned, we checked whether the validation accuracy improved for 25 such consecutive evaluations and stopped training if it did not. This had the benefit of not having a fixed number of maximum iterations after which we would be still unsure about the convergence. To make comparisons across different models meaningful, we used number of samples as the measure of training progress which is simply the number of sentences a model has seen between each evaluation step[4]. Final performance was then evaluated on a test set, using the model that achieved the highest validation accuracy. We used a learning rate of $3e-4$ and an embedding dimension of 300 for all models. All BOW models which have one or more hidden layers used a hidden dimension of 100 and all LSTM models used a hidden dimension of 150. We accounted for the stochasticity in the models by training each model 20 times with different random seeds and computed mean accuracies with a 95% confidence interval. For the LSTM and Tree-LSTM model, we used the pretrained Word2Vec embeddings and left them fixed during training. For the Deep CBOW model, we compared both GloVe and Word2Vec embeddings with ones learned from scratch. All other BOW models learnt the embeddings from scratch. In addition to the straightforward performance comparison of the models, we performed several other experiments. To quantify the affect of word order in the LSTM model, we tested two contrastive scenarios- shuffling both while training and testing; shuffling only while testing. Next, we tried to improve the Tree-LSTM performance by adding all non-singleton subtrees for each sentence to the training set. This worked like some sort of node-level supervision instead of only on the sentence-level. Finally, we tested the performance of each final trained model for different sentence lengths.

5 Results and Analysis

The performance of all models on the test set are visualised using violin-plots as shown in Figure 2. The numeric values are listed in Table 1. The development of validation accuracy during training is shown in Figure 3 while loss and validation accuracy curves of all runs are provided in Appendix A.

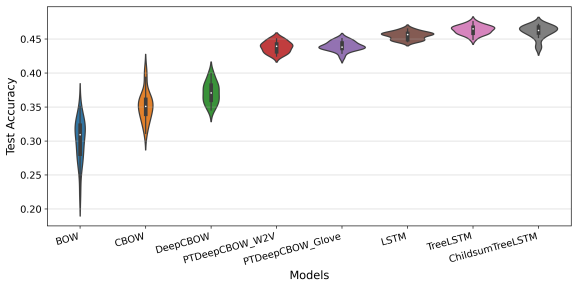

Figure 2: Performance on test set of the models

We observed that introducing larger embedding space in CBOW yields a slight improvement over BOW but further increase in the capacity of Deep CBOW models did not help. This is because the

[2]Multi Layer Perceptron
[3]at each step LSTMs were fed a batch of 25 sentences
[4]LSTMs, for example, see $50 \times 25 = 1250$ sentences

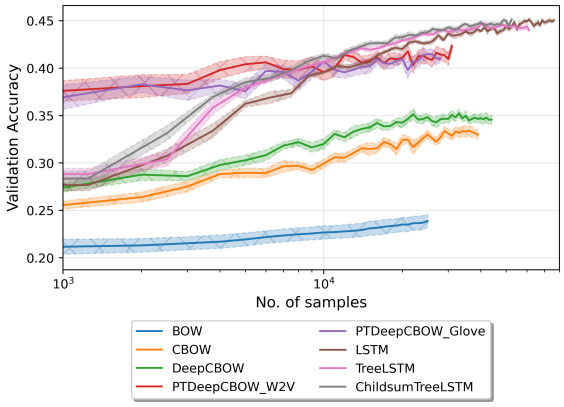

Figure 3: Mean validation accuracies for the different methods over training time. The hatched area shows 95% conf. interval

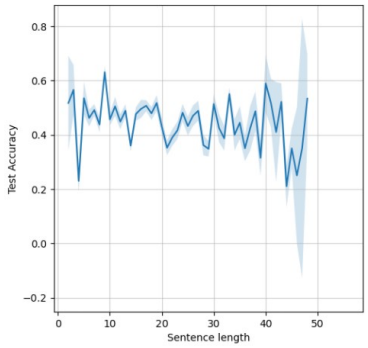

Figure 4: Performance of LSTM for different sentence length. The hatched area shows $\pm 1\sigma$

Model	Test Accuracy
BOW	30.1 ± 3.0
CBOW	35.2 ± 2.2
Deep CBOW	37.1 ± 1.5
Pt(W2V) Deep CBOW	43.8 ± 0.8
Pt(GloVe) Deep CBOW	43.9 ± 0.7
LSTM	45.5 ± 0.5
LSTM shuffle train	45.4 ± 0.5
LSTM shuffle all	45.2 ± 0.5
TreeLSTM	46.4± 0.6
TreeLSTM with subtrees	45.6± 0.9
Childsum TreeLSTM	46.1± 0.9

Table 1: Mean test accuracies in % of different models.

CBOW model might already be overfitting to the training set as we observed training accuracy values of more than 70%. Using pretrained embeddings resulted in much better performance. This is also what we expected because the embeddings were trained on a significantly larger corpus and are better at capturing meaningful differences between words. Additionally, the reduced number of parameters made overfitting less likely which explains the improvement in test performance. LSTMs achieve a slightly better performance than the pretrained deep CBOW model. Their performance did not change significantly with word order shuffling indicating that word order did not had a large role in the task. It is important to note that our LSTM has only a single layer and the train set is not particularly large. It's plausible that word order is relevant for the task of sentiment classification, but the LSTM is simply unable to make much use of it. Since the importance of word order for the LSTM is relatively small, using the full syntactic structure did not give significant performance boost with Tree-LSTM. Including all non-singleton subtrees in the training set, resulted in a drop of about 1.2% in test accuracy compared to the normal Tree-LSTMs. This can be explained by the fact that with subtrees we trained on more redundant phrases which might cause overfitting. The Childsum Tree-LSTM achieved virtually the same result as Binary Tree-LSTM, suggesting that the ability to differentiate between child nodes did not give a significant advantage in this task. Finally, for all models we found the performance was almost uniform across sentence lengths. We show this for LSTM in Figure 4. The other plots are qualitatively similar and can be found in Appendix A.

6 Conclusion

We compared different architectures of BOW and LSTM-based models. We analyzed the effects of using pretrained word embeddings, word order, and node-level supervision with subtrees. A possible extension would be to pick specific meaningful subtrees which are important to reconstruct their parent node vector. The most significant improvement was achieved by using pretrained word embeddings, with little subsequent improvements from LSTMs. Presumably this is because the size of our data set was an important limiting factor and the benefits from expressive models like LSTMs would only become apparent with larger datasets. We also observed that these methods progressively improved classification at the cost of increased computation and reduced transparency. As Box, 1976 said: *All models are wrong, some are useful.*

References

George EP Box. 1976. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799.

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

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

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*, pages 3111–3119.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

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.

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. *arXiv preprint arXiv:1503.00075*.

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 Appendices

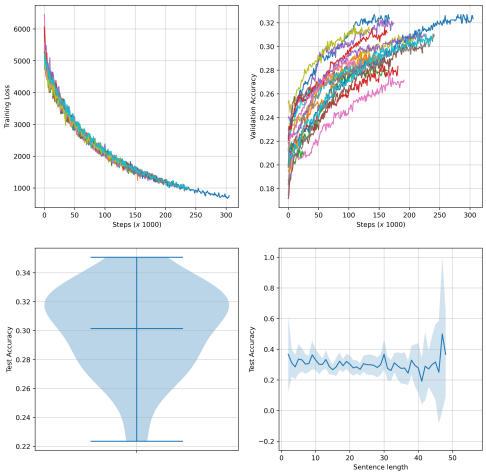

Figure 5: BOW

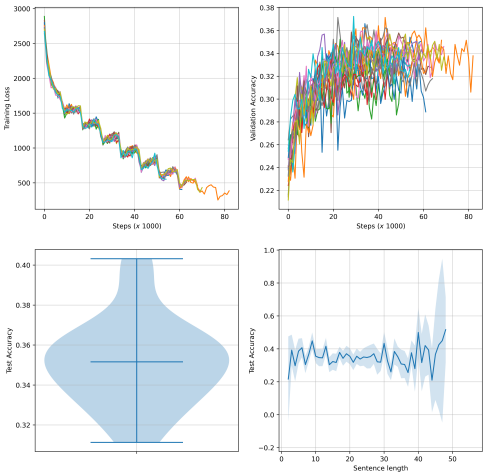

Figure 6: CBOW

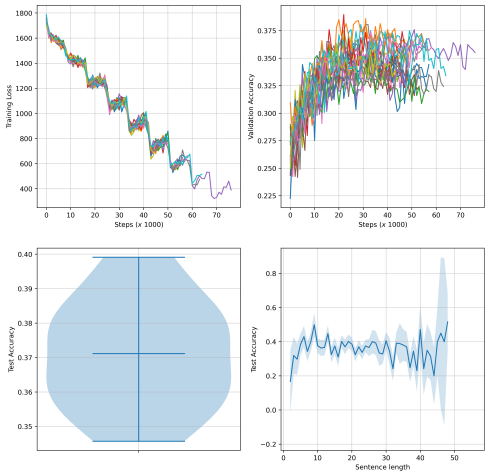

Figure 7: Deep CBOW

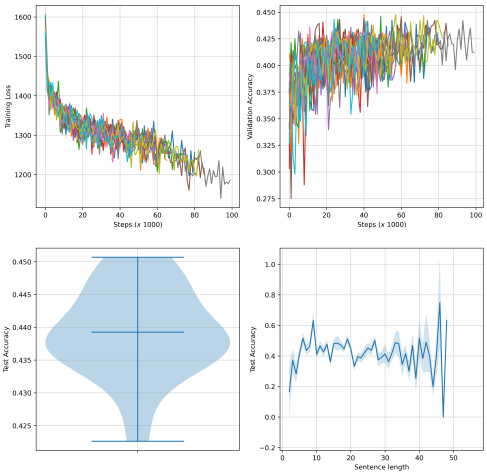

Figure 9: GloVe Pretrained DeepCBow

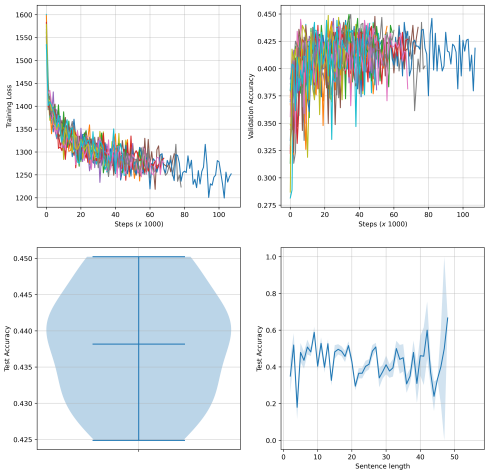

Figure 8: Word2Vec Pretrained Deep CBow

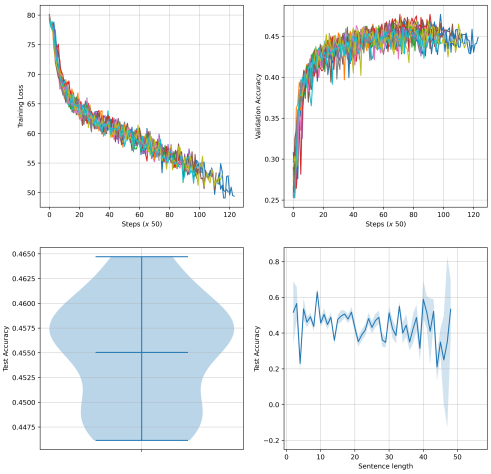

Figure 10: LSTM

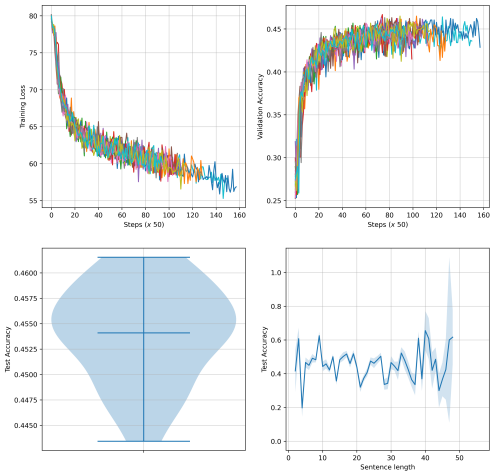

Figure 11: LSTM with word order shuffling done only on train set

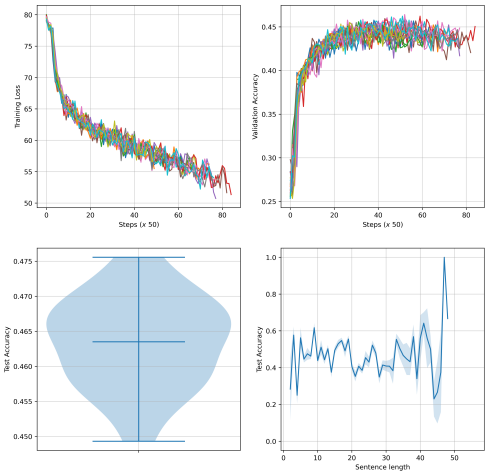

Figure 13: Tree-LSTM

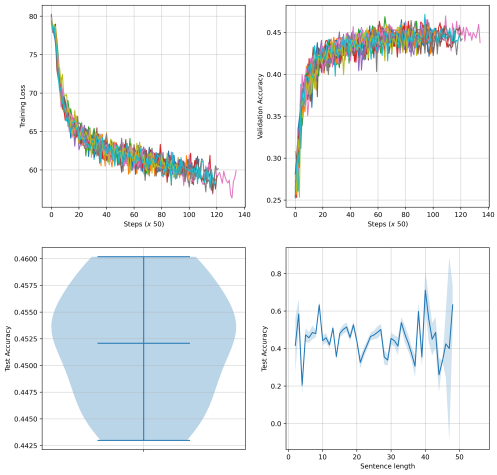

Figure 12: LSTM with word order shuffling done on both train test set

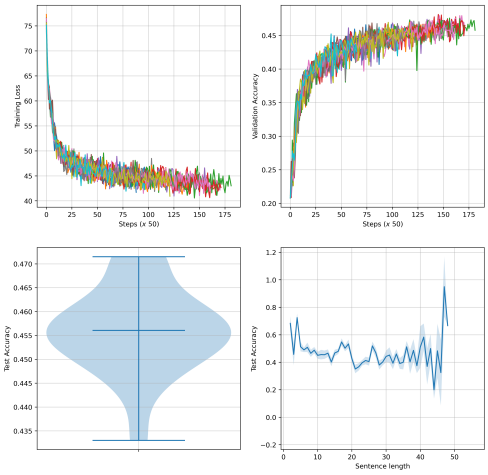

Figure 14: Tree-LSTM with subtrees

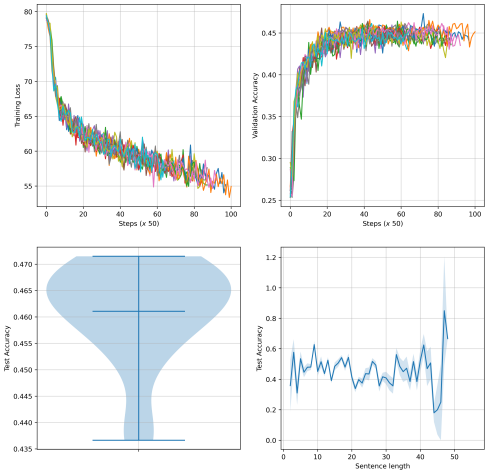

Figure 15: Childsum Tree-LSTM

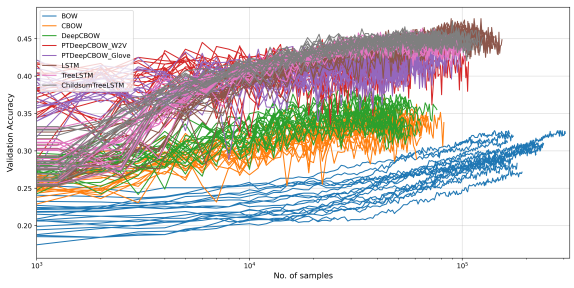

Figure 16: Validation accuracies for multiple runs of various models