Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs

Phong Le and **Willem Zuidema**
Institute for Logic, Language and Computation
University of Amsterdam, the Netherlands
{p.le,zuidema}@uva.nl

Abstract

Recursive neural networks (RNN) and their recently proposed extension recursive long short term memory networks (RLSTM) are models that compute representations for sentences, by recursively combining word embeddings according to an externally provided parse tree. Both models thus, unlike recurrent networks, explicitly make use of the hierarchical structure of a sentence. In this paper, we demonstrate that RNNs nevertheless suffer from the vanishing gradient and long distance dependency problem, and that RLSTMs greatly improve over RNN's on these problems. We present an artificial learning task that allows us to quantify the severity of these problems for both models. We further show that a ratio of gradients (at the root node and a focal leaf node) is highly indicative of the success of backpropagation at optimizing the relevant weights low in the tree. This paper thus provides an explanation for existing, superior results of RLSTMs on tasks such as sentiment analysis, and suggests that the benefits of including hierarchical structure and of including LSTM-style gating are complementary.

1 Introduction

The recursive neural network (RNN) model became popular since the work of Socher et al. (2010). It has been employed to tackle several NLP tasks, such as syntactic parsing (Socher et al., 2013a), machine translation (Liu et al., 2014), and word embedding learning (Luong et al., 2013). However, like traditional *recurrent* neural networks, the RNN seems to suffer from the vanishing gradient problem, in which error signals propagating from the root in a parse tree to the child nodes shrink very quickly. Moreover, it encounters difficulties in capturing long range dependencies: information propagating from child nodes deep in a parse tree can be obscured before reaching the root node.

In the recurrent neural network world, the long short term memory (LSTM) architecture (Hochreiter and Schmidhuber, 1997) is often used as a solution to these two problems. A natural extension of the LSTM can be defined for tree structures, which we call Recursive LSTM (RLSTM), as proposed independently by Tai et al. (2015), Zhu et al. (2015), and Le and Zuidema (2015). However, while there is intensive research showing how the LSTM architecture can overcome those two problems compared to traditional recurrent models (e.g., Gers and Schmidhuber (2001)), such research is, to our knowledge, still absent for the comparison between RNNs and RLSTMs. Therefore, in the current paper we investigate the following two questions:

1. Is the RLSTM more capable of capturing long range dependencies than the RNN?
2. Does the RLSTM overcome the vanishing gradient problem more effectively than the RNN?

Supervised learning requires annotated data, which is often expensive to collect. As a result, examining a model on natural data on many different aspects can be difficult because the portion of data that fits a specific aspect could not be sufficient. Moreover, studying individual aspects separately is hard since many aspects are often correlated with each other. This, unfortunately, is true in our case: answering those two questions requires us to evaluate the examined models on datasets of dif-

Proceedings of the 1st Workshop on Representation Learning for NLP, pages 87–93,
Berlin, Germany, August 11th, 2016. ©2016 Association for Computational Linguistics

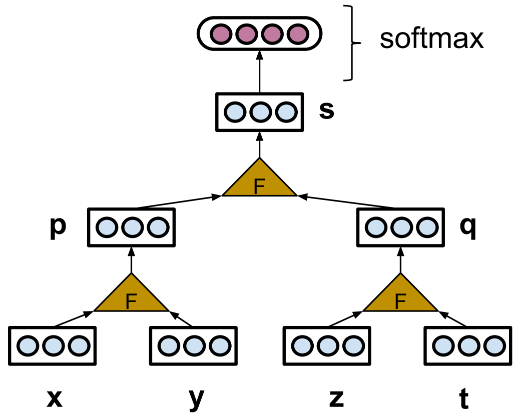


Figure 1: A recursive model (such as RNN and RLSTM) employ a composition function F in a bottom-up manner to compute a vector representation for each internal node in a tree. If the model is used for classification on the sentence level, a softmax layer is put on the top of the root node to compute a distribution over all possible classes.

ferent tree depths, in which the key nodes which contain decisive information in a parse tree must be identified. Using available annotated corpora such as the Stanford Sentiment Treebank (Socher et al., 2013b) and the Penn Treebank is thus inappropriate, as they are too small for this purpose (10k, 40k trees, respectively, compared to 240k trees in our experiments), and key nodes are not marked. Our solution is an artificial task where sentences and parse trees can be randomly generated under any arbitrary constraints on tree depth and key node's position.

2 Background

Both the RNN and the RLSTM model are instances of a general framework which takes a sentence, syntactic tree, and vector representations for the words in the sentence as input, and applies a composition function to recursively compute vector representations for all the phrases in the tree and the complete sentence. Technically speaking, given a production $p \rightarrow x\ y$, and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ representing x, y, we compute $\mathbf{p} \in \mathbb{R}^n$ for p by

$$\mathbf{p} = F(\mathbf{x}, \mathbf{y})$$

where F is a composition function (Figure 1).

In the RNN, F is a one-layer feed-forward neural network:

$$\mathbf{p} = f(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{y} + \mathbf{b})$$

where $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$ are weight matrices and $\mathbf{b} \in \mathbb{R}^n$ is a bias vector. f is an activation function.

In the RLSTM, a node p is represented by the vector $[\mathbf{p}; \mathbf{c}_p]$ resulting from concatenating a vector representing the phrase that the node covers and a memory vector. F could be any LSTM that can compute two such concatenation vectors, such as Structure-LSTM (Zhu et al., 2015), Tree-LSTM (Tai et al., 2015), and LSTM-RNN (Le and Zuidema, 2015). In the current paper, we use the implementation[1] of Le and Zuidema (2015) where an LSTM (for binary trees) has two input gates i_1, i_2, two forget gates f_1, f_2, an output gate o, and a memory cell c. The vector representation and memory vector for node p are computed as follows:

$$\begin{aligned}
\mathbf{i}_1 &= \sigma\big(\mathbf{W}_{i1}\mathbf{x} + \mathbf{W}_{i2}\mathbf{y} + \mathbf{W}_{ci1}\mathbf{c}_x + \mathbf{W}_{ci2}\mathbf{c}_y + \mathbf{b}_i\big) \\
\mathbf{i}_2 &= \sigma\big(\mathbf{W}_{i1}\mathbf{y} + \mathbf{W}_{i2}\mathbf{x} + \mathbf{W}_{ci1}\mathbf{c}_y + \mathbf{W}_{ci2}\mathbf{c}_x + \mathbf{b}_i\big) \\
\mathbf{f}_1 &= \sigma\big(\mathbf{W}_{f1}\mathbf{x} + \mathbf{W}_{f2}\mathbf{y} + \mathbf{W}_{cf1}\mathbf{c}_x + \mathbf{W}_{cf2}\mathbf{c}_y + \mathbf{b}_f\big) \\
\mathbf{f}_2 &= \sigma\big(\mathbf{W}_{f1}\mathbf{y} + \mathbf{W}_{f2}\mathbf{x} + \mathbf{W}_{cf1}\mathbf{c}_y + \mathbf{W}_{cf2}\mathbf{c}_x + \mathbf{b}_f\big) \\
\mathbf{c}_p &= \mathbf{f}_1 \odot \mathbf{c}_x + \mathbf{f}_2 \odot \mathbf{c}_y + \\
&\quad g\big(\mathbf{W}_{c1}\mathbf{x} \odot \mathbf{i}_1 + \mathbf{W}_{c2}\mathbf{y} \odot \mathbf{i}_2 + \mathbf{b}_c\big) \\
\mathbf{o} &= \sigma\big(\mathbf{W}_{o1}\mathbf{x} + \mathbf{W}_{o2}\mathbf{y} + \mathbf{W}_{co}\mathbf{c} + \mathbf{b}_o\big) \\
\mathbf{p} &= \mathbf{o} \odot g(\mathbf{c}_p)
\end{aligned}$$

where $\mathbf{u}$ and $\mathbf{c}_u$ are the output and the state of the memory cell at node u; $\mathbf{i}_1$, $\mathbf{i}_2$, $\mathbf{f}_1$, $\mathbf{f}_2$, $\mathbf{o}$ are the activations of the corresponding gates; $\mathbf{W}$'s and $\mathbf{b}$'s are weight matrices and bias vectors; and g is an activation function.

3 Experiments

We now examine how the two problems, the vanishing gradient problem and the problem of how to capture long range dependencies, affect the RLSTM model and the RNN model. To do so, we propose the following artificial task, which requires a model to distinguish useful signals from noise. We define:

- a sentence is a sequence of tokens which are integer numbers in the range $[0, 10000]$;
- a sentence contains one and only one *keyword* token which is an integer number smaller than 1000;
- a sentence is labeled with the integer resulting from dividing the keyword by 100. For

[1] https://github.com/lephong/lstm-rnn

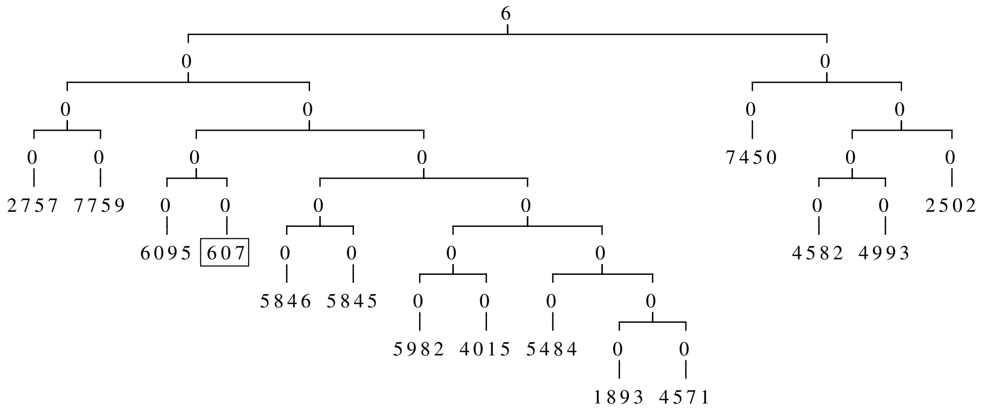


Figure 2: Example binary tree for the artificial task. The number enclosed in the box is the *keyword* of the sentence.

instance, if the keyword is 607, the label is 6. In this way, there are 10 classes, ranging from 0 to 9.

The task is to predict the class of a sentence, given its binary parse tree (Figure 2). Because the label of a sentence is determined solely by the keyword, the two models need to identify the keyword in the parse tree and allow only the information from the leaf node of the keyword to affect the root node. It is worth noting that this task resembles sentiment analysis with simple cases in which the sentiment of a whole sentence is determined by one keyword (e.g. "I like the movie"). Simulating complex cases involving negation, composition, etc. is straightforward and for future work. But here we believe that the current task is adequate to answer our two questions raised in Section 1.

The two models, RLSTM and RNN, were implemented with the dimension of vector representations and vector memories 50. Following Socher et al. (2013b), we used $tanh$ as the activation function, and initialized word vectors by randomly sampling each value from a uniform distribution $U(-0.0001, 0.0001)$. We trained the two models using the AdaGrad method (Duchi et al., 2011) with a learning rate of 0.05 and a mini-batch size of 20 for the RNN and of 5 for the RLSTM. Development sets were employed for early stopping (training is halted when the accuracy on the development set is not improved after 5 consecutive epochs). It is worth noting that we also tried other values for the hyper-parameters but did not gain significantly better results on development sets.

3.1 Experiment 1

We randomly generated 10 datasets. To generate a sentence of length l, we shuffle a list of randomly chosen $l-1$ non-keywords and one keyword. The i-th dataset contains 12k sentences of lengths from $10i-9$ tokens to $10i$ tokens, and is split into train, dev, test sets with sizes of 10k, 1k, 1k sentences. We parsed each sentence by randomly generating a binary tree whose number of leaf nodes equals to the sentence length.

The test accuracies of the two models on the 10 datasets are shown in Figure 3; For each dataset we run each model 5 times and reported the highest accuracy for the RNN model, and the distribution of accuracies (via boxplot) for the RLSTM model. We can see that the RNN model performs reasonably well on very short sentences (less than 11 tokens). However, when the sentence length exceeds 10, the RNN's performance drops so quickly that the difference between it and the random guess' performance (10%) is negligible. Trying different learning rates, mini-batch sizes, and values for n (the dimension of vectors) did not give significant differences. On the other hand, the RLSTM model achieves more than 90% accuracy on sentences shorter than 31 tokens. Its performance drops when the sentence length increases, but is still substantially better than the random guess when the sentence length does not exceed 70. When the sentence length exceeds 70, both the RLSTM and RNN perform similarly.

3.2 Experiment 2

In Experiment 1, it is not clear whether the tree size or the keyword depth is the main factor of the rapid drop of the RNN's performance. In this ex-

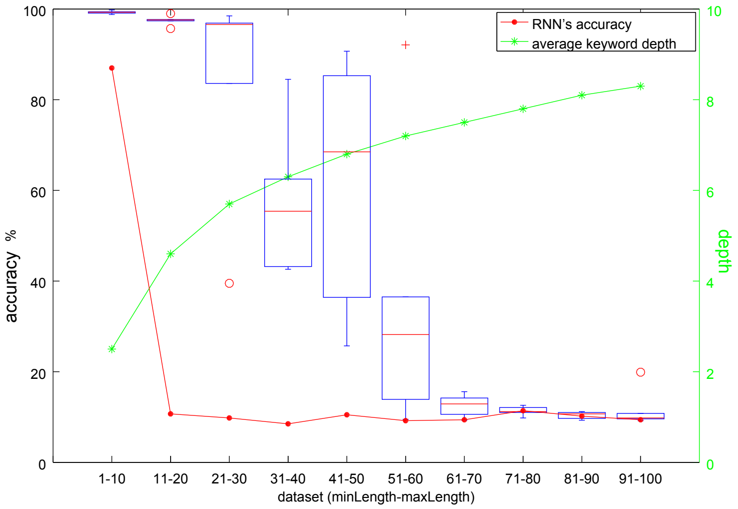


Figure 3: Test accuracies of the RNN (red solid curve, the best among 5 runs) and the RLSTM (boxplots) on datasets of different sentence lengths.

periment, we kept the tree size fixed and vary the keyword depth. We generated a pool of sentences of lengths from 21 to 30 tokens and parsed them by randomly generating binary trees. We then created 10 datasets each of which has 12k trees (10k for training, 1k for development, and 1k for testing). The i-th dataset consists of only trees in which distances from keywords to roots are i or $i+1$ (to stop the networks from exploiting keyword depths directly).

Figure 4 shows test accuracies of the two models on those 10 datasets. Similarly in Experiment 1, for each dataset we run each model 5 times and reported the highest accuracy for the RNN model, and the distribution of accuracies for the RLSTM model. As we can see, the RNN model achieves very high accuracies when the keyword depth does not exceed 3. Its performance then drops rapidly and gets close to the performance of the random guess. This is evidence that the RNN model has difficulty capturing long range dependencies. By contrast, the RLSTM model performs at above 90% accuracy until the depth of the keyword reaches 8. It has difficulty dealing with larger depths, but the performance is always better than the random guess.

3.3 Experiment 3

We now examine whether the two models can encounter the vanishing gradient problem. To do so, we looked at the the back-propagation phase of each model in Experiment 1 on the third dataset (the one containing sentences of lengths from 21 to 30 tokens). For each tree, we calculated the ratio

$$\frac{\left\|\frac{\partial J}{\partial \mathbf{x}_{keyword}}\right\|}{\left\|\frac{\partial J}{\partial \mathbf{x}_{root}}\right\|}$$

where the numerator is the norm of the error vector at the keyword node and the denominator is the norm of the error vector at the root node. This ratio gives us an intuition how the error signals develop when propagating backward to leaf nodes: if the ratio $\ll 1$, the vanishing gradient problem occurs; else if the ratio $\gg 1$, we observe the exploding gradient problem.

Figure 5 reports the ratios w.r.t. the keyword node depth in each epoch of training the RNN model. The ratios in the first epoch are always very small. In each following epoch, the RNN model successfully lifts up the ratios steadily (see Figure 7a for a clear picture at the keyword depth 10), but a clear decrease when the depth becomes larger is observable. For the RLSTM model (see Figure 6 and 7b), the story is somewhat different. The ratios go up after two epochs so rapidly that there are even some exploding error signals sent back to leaf nodes. They subsequently go down and remain stable with substantially less exploding error signals. This is, interestingly, concurrent with the performance of the RLSTM model on the development set (see Figure 7b). It seems that the RLSTM model, after one epoch, quickly locates the keyword node in a tree and relates it to the root by building a strong bond between them via error signals. After the correlation between the keyword and the label at the root is found, it tries to stabilize the training by reducing the error signals sent back

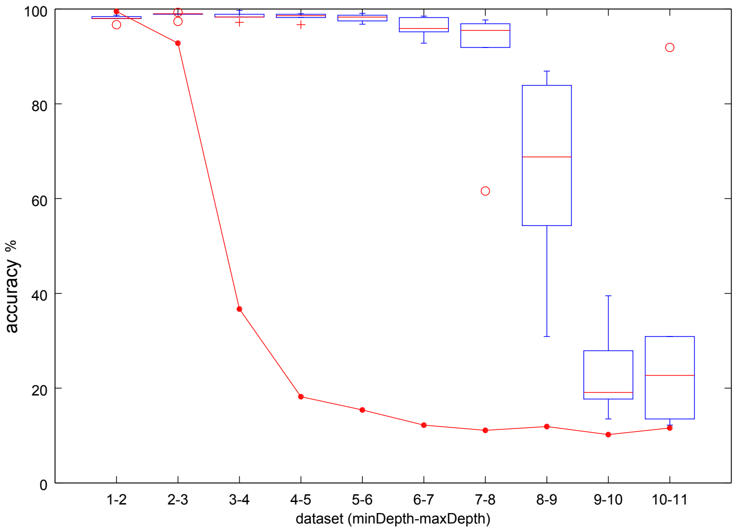


Figure 4: Test accuracies of the RNN (red solid curve, the best among 5 runs) and the RLSTM (boxplots) on datasets of different keyword depths.

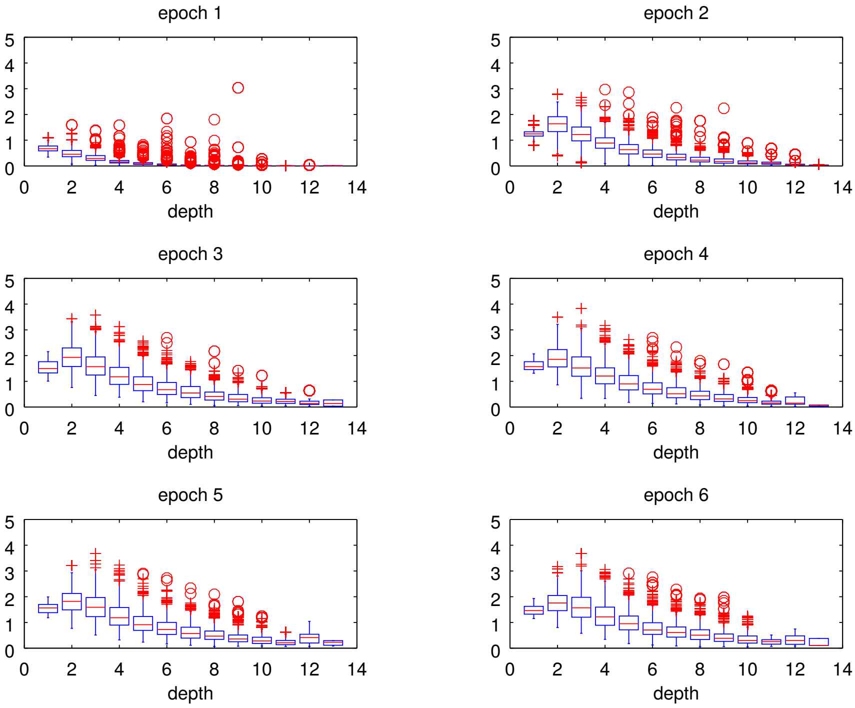


Figure 5: Ratios of norms of error vectors at keyword nodes to norms of error vectors at root nodes w.r.t. the keyword node depth in each epoch of training the RNN. Gradients gradually vanish with greater depth.

to the keyword node. Comparing the two models by aligning Figure 5 with Figure 6, and Figure 7a with Figure 7b, we can see that the RLSTM model is more capable of transmitting error signals to leaf nodes.

It is worth noting that we do see the vanishing gradient problem happening when training the RNN model in Figure 5; but Figure 7a suggests that the problem can become less serious after a long enough training time. This might be because depth 10 is still manageable for the RNN model. (Notice that in the Stanford Sentiment Treebank, more than three quarters of leaf nodes are at depths less than 10.) The fact the the RNN model still doesnot perform better than random guessing can be explained using the arguments given by Bengio et al. (1994), who show that there is a trade-off between avoiding the vanishing gradient problem

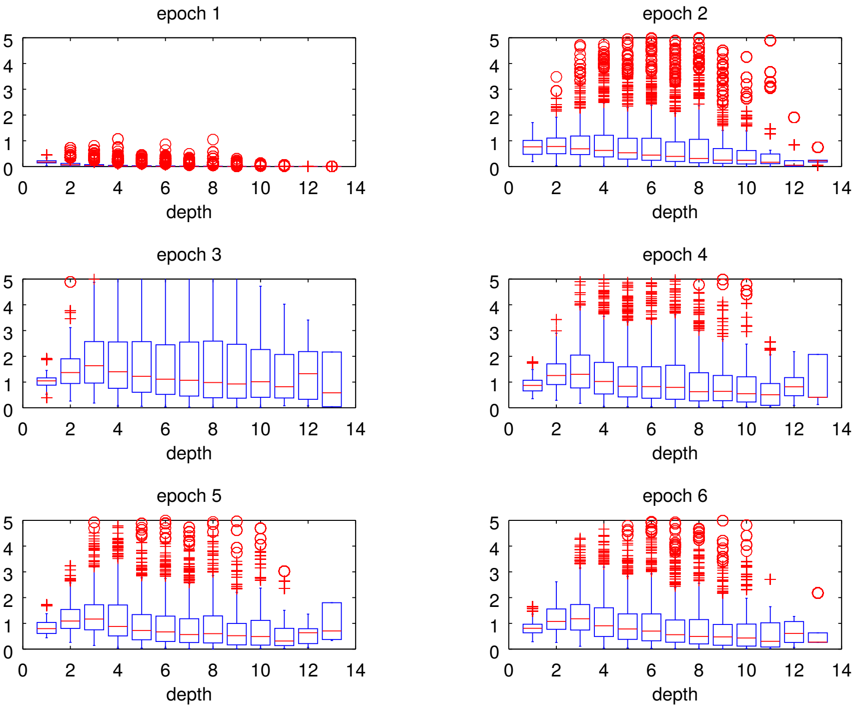


Figure 6: Ratios of norms of error vectors at keyword nodes (at different depths) to norms of error vectors at root nodes, in the RLSTM. Many gradients explode in epoch 2, but stabilize later. Gradients do not vanish, even at depth 12 and 13.

and capturing long term dependencies when training traditional recurrent networks.

4 Conclusion

Because long range dependencies and vanishing gradients are serious challenges in deep learning, evaluating how well a model overcome these challenges is necessary. In this current paper, we focus on two recursive models, RNN and RLSTM. Due to lack of natural data, we proposed a novel artificial task where the label of a sentence is solely determined by a key word it contains. The experimental results show that the RLSTM is superior to the RNN. This is in parallel with general conclusions about the power of the LSTM architecture compared to traditional Recurrent neural networks.

Although our proposed task is simple, it is sufficient for testing recursive models since solving the task requires models to be capable of capturing long range dependencies and propagating errors to leaf nodes far from the root. It is, moreover, straightforward to extend the task such that more complex cases can be taken into account. For instance, for compositionality, a sentence can contain more than one keywords and the sentence label is determined by some kind of interaction between those keywords (such as addition).

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

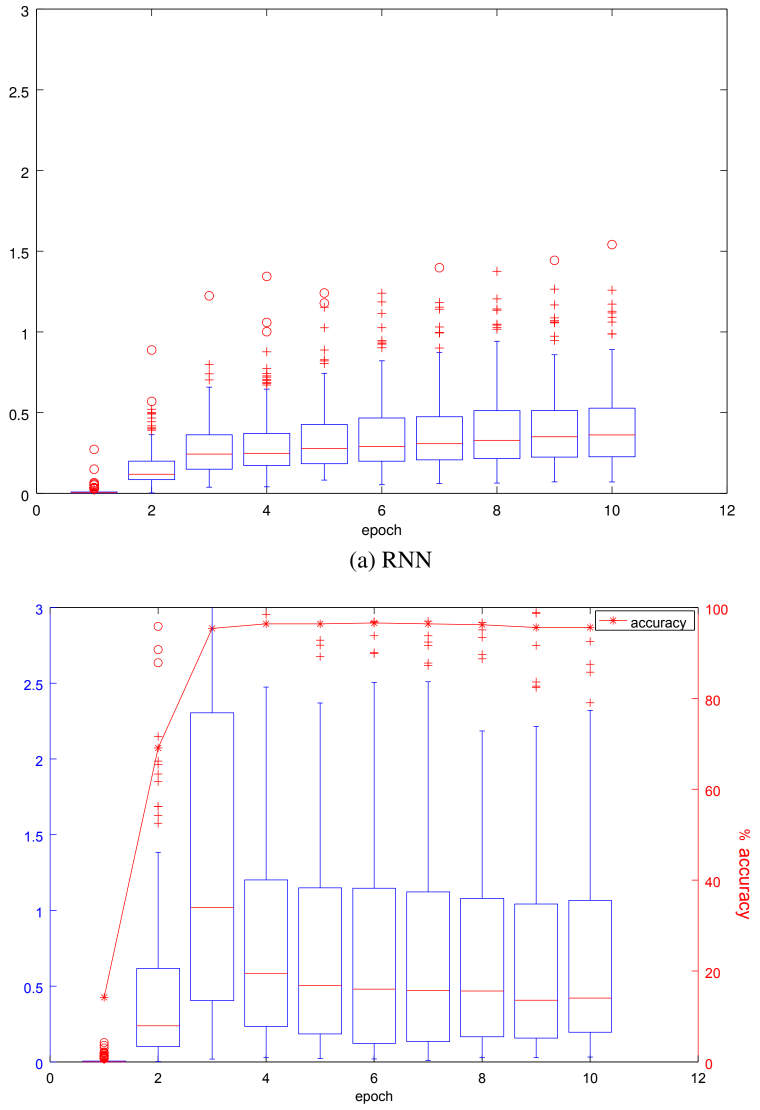
John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pages 2121–2159.

Felix A Gers and Jürgen Schmidhuber. 2001. Lstm recurrent networks learn simple context-free and context-sensitive languages. *Neural Networks, IEEE Transactions on*, 12(6):1333–1340.

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. In *Proceedings of the Joint Conference on Lexical and Computational Semantics (*SEM)*. Association for Computational Linguistics.

Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. 2014. A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational*



(a) RNN

(b) RLSTM (with development accuracies)

Figure 7: Ratios at depth 10 in each epoch of training the RNN (a) and the RLSTM (b).

Linguistics (Volume 1: Long Papers), pages 1491–1500, Baltimore, Maryland, June. Association for Computational Linguistics.

Minh-Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. *CoNLL-2013*, 104.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013a. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings EMNLP*.

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, July. Association for Computational Linguistics.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of International Conference on Machine Learning*, July.