Efficient bottom-up learning of tree-structured neural networks for sentiment classification

Phillip Lippe
University of Amsterdam
phillip.lippe@googlemail.com

Rick Halm
University of Amsterdam
rickhalm@me.com

Abstract

In this paper, we present a novel approach for efficient loss weighting in a tree-structured neural network. Current methods consider either only the top-node prediction for loss calculation, or weight all nodes equally yielding to a strongly imbalanced class loss. Our method progresses through the tree, starting at word level, to focus the loss on misclassified nodes. We propose three different heuristics for determining such misclassifications and investigate their effect and performance on the Stanford Sentiment Treebank based on a binary Tree-LSTM model. The results show a considerable improvement compared to previous models concerning accuracy and overfitting.[1]

1 Introduction

Recurrent Neural Networks (RNNs), especially those with gating mechanisms like LSTMs, constitute start-of-the-art models for various tasks in Natural Language Processing including machine translation (Cho et al., 2014b), speech recognition (Bahdanau et al., 2016) and sentiment classification (Tai et al., 2015; Semeniuta et al., 2016).

Mostly, RNNs are applied in a linear chain topology which only allows strictly sequential input data. However, the syntactic interpretation of sentences primarily relies on a tree-like structure which can not be captured by standard RNNs (Everaert et al., 2015). Therefore, various network architectures have recently been proposed that model the syntactic structure of a sentence by applying LSTM cells in a tree topology (Tai et al., 2015; Le and Zuidema, 2015; Zhu et al., 2015). Such networks yield superior results on several problems such as sentiment classification where representing sentences has been a crucial task (Li et al., 2015; Andreas et al., 2016).

Nevertheless, gradient backflow through RNNs has often shown to be a problem (Hochreiter and Schmidhuber, 1997; Cho et al., 2014a). Thus, to enrich the loss information, a classifier can be applied on every node within the tree model and compared to the annotated label in the dataset (if available). However, if a node is misclassified in the tree, all consecutive parent nodes would still rely on this poor embedding and therefore produce false predictions. The loss at those nodes are without information as the mistake was performed somewhere else in the tree, and probably leads to gradients in the wrong direction and unstable training. Besides, sentences are usually significantly biased towards neutral words and phrases such as *the* and *a*, so that the loss will be likewise focused on a neutral sentiment.

In this paper, we introduce a bottom-up learning technique to overcome class imbalances and efficiently learn from mistakes within a tree neural network. Starting from the lowest tree level, we determine at which nodes the network predicted a wrong class, and exclude the loss of all its parent nodes. Therefore, we focus the feedback signal to those nodes and subtrees that cause the misclassification of the sentence and prevent training on predictions that rely on poorly embedded input.

In experiments, we apply our proposed method on a binary Tree-LSTM network introduced by Tai et al. (2015) for the task of sentiment classification on the Stanford Sentiment Treebank (Socher et al., 2013). We compare our model against established methods for sentiment classification including the neural bag of words, sequential LSTMs and the Tree-LSTM network with standard training. Furthermore, the effect of the bottom-up

[1] Implementations of our models and experiments are available at https://github.com/phlippe/NLP_Project.git

learning technique is investigated by analyzing the influence of sentence length and word structures, as well as different heuristics to determine when to stop at a node. Overall, the new training method yields significant improvements in the performance of the tree network showing promising results for future work on other tasks in natural language processing.

2 Background

One of the classic models for sentiment analysis in NLP is the Bag-of-Words (BOW) model that analyzes the words in a sentence independently (Iyyer et al., 2015; Sheikh et al., 2016). In a BOW, a word is embedded by a weight for every class, and the maximum value of the sum of all words represents the prediction of the sentence. An extension of this model is the continuous BOW (CBOW) where a layer is added between the word embedding, that now can have arbitrary size, and the class weights. A CBOW with more layers is called Deep CBOW.

However, the word order can be crucial for certain tasks. One method to take this into account are Recurrent Neural Networks which process sequential data by sharing features over time. Still, standard RNNs miss long-distance dependencies as a result of the issue of vanishing gradients (Pascanu et al., 2013a). This problem is overcome by Long Short Term Memory (LSTM) cells that, next to using the output of the preceding cell, also keep a cell state that is passed over time (Hochreiter and Schmidhuber, 1997). For every word, the LSTM uses gates to determine what information to keep from previous inputs, and what new features to add based on the current input. This allows the LSTM to preserve gradients over longer sequences and leads to an efficient network training.

Until now, LSTMs are assumed to be a linear chain of individual cells which only allows for strictly sequential flow of information. However, with Tree-LSTMs, tree-structured data can be processed where the dependencies in the input are explicitly represented in the model architecture. An example for such a network is a N-ary Tree-LSTM (Tai et al., 2015) that extends LSTMs by taking, next to an (optional) input, the cell and hidden states of its N children into account. Thereby, a cell distinguishes between the order of its children. Next to N-ary, there are also different tree architectures such as the Child-Sum Tree-LSTM (Tai et al., 2015) or S-LSTM (Zhu et al., 2015).

3 Models

In this section, we describe the details of our models applied for the task of sentiment classification.

3.1 Model architectures

We use as baseline five standard architectures for sentiment analysis. The first model is the simple bag-of-words (BOW) with 5 weights for each word. We extend this approach by a CBOW and a three-layers deep CBOW (all 300-dim., tanh).

As a more complex approach, we implement a single-layer LSTM network where we enter the word embeddings sequentially into the LSTM cell, and apply a classifier on only the last cell state. The classifier consists of a single fully-connected layer and softmax activation, from which we retrieve our predictions for the five classes. Next to the standard LSTM, we investigate the performance of a binary Tree-LSTM model as introduced in Tai et al. (2015). The network is structured according to the sentence parse tree provided by the dataset. We use the same classifier as for the LSTM on the final embedding. For all models, we apply the sparse cross-entropy loss function.

3.2 Bottom-up learning

By training a tree-structured network on all labels within a tree, we can enrich the loss information, but the question arises how to combine the losses efficiently. If we weight all nodes equally, the loss would significantly be biased on low levels of the tree like single words, as low levels contain much more nodes. Still, the evaluation of the model is only based on the final node and should therefore be the focus. If we use a decay factor based on the height of a node, the weights will either vanish for deep trees, or the first case happens again.

To overcome this problem, we propose to focus our loss signal on those nodes/subtrees, where the model firstly made a mistake in the tree. Starting from the word level, we compare the classifiers' prediction with the given label by a heuristic (see later for discussion of heuristics). If a node's prediction is considered as wrong, the losses of all Tree-LSTM cells that are ancestors of this node and thereby rely on its embedding, are excluded from the update as the parent's prediction is based on incorrect input. This will lead to a richer, carefully selected loss signal on the important parts of the training example, and prevents that we train a

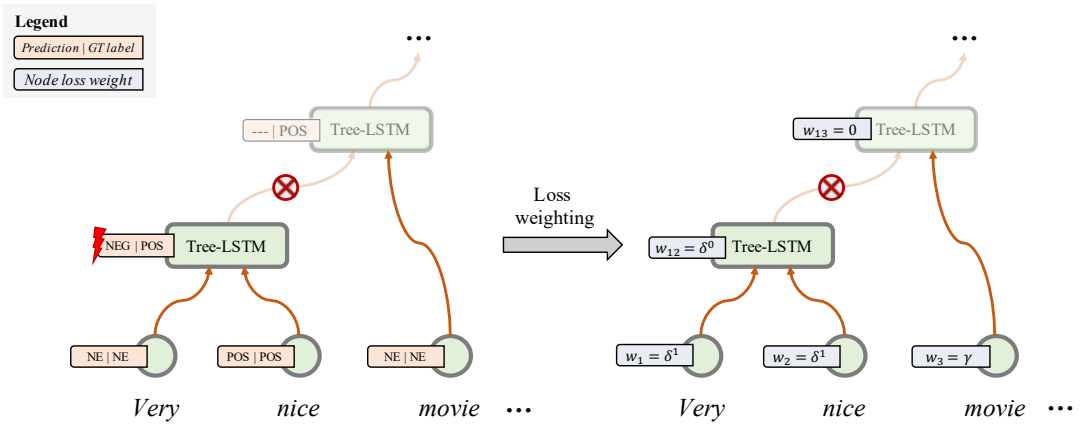

Figure 1: During training, we predict a class at every node and compare it to the label. Based on a heuristic like 1-distance, we decide at which nodes to stop. In the shown example, this is the case for the first Tree-LSTM cell where the classifier predicts *negative* although the correct label is *positive*. All subsequent Tree-LSTM cells using this embedding, are ignored and will not be punished in the loss (corresponding parts are shaded in the figure). The loss weights are set to 1 at each node where the heuristic stopped, and δ^1 for its children, δ^2 for its children's children, and so on. Nodes that are neither parent or child of a stopped node, are assigned with the default weight γ as for instance "movie". Note, that if multiple independent nodes are misclassified, we sum all concerning losses together.

Tree-LSTM on poorly embedded inputs. The described method is visualized in Figure 1.

Solely training on the misclassified nodes will again lead to a sparse loss signal as most nodes in the tree are ignored. We therefore add a decay factor δ between 0 and 1 for its children so that we can define our loss for a single misclassified node c as a recursive function:

$$\mathcal{L}_{tree}(c) = \mathcal{L}_{\text{CE}}(c) + \sum_{s \in T(c)} \delta \cdot \mathcal{L}_{tree}(s)$$

where $\mathcal{L}_{\text{CE}}$ represents the cross-entropy error function, and $T(c)$ the set of all children of c (in case of binary trees always two). Additionally, nodes that are unrelated to any stopped node, so neither a child nor parent, are assigned with a loss factor γ. In experiments, we experienced that low values of γ up to 0 perform best.

3.3 Heuristics

The proposed method requires a heuristic that determines where the network makes a mistake. In this paper, we focus on three heuristics that are based on predictions at each node of the tree.We consider the fine-grained classification of the Stanford Sentiment Treebank (Socher et al., 2013) with five classes from *very positive* to *very negative*.

Binary: The simplest heuristic for determining when to stop in a tree is comparing the prediction with the label of every node. If both are equal, we continue. Otherwise, we stop at the node where the prediction does not fit to the label.

n-distance: The binary heuristic does not consider the fact that labels like *positive* and *very positive* are more similar than *very positive* and *very negative*. Thus, we propose the n-distance heuristic which only stops at a node if its prediction is more than n class steps away from the actual label. For example, a 1-distance heuristic allows a prediction of *positive* when the label was actually *very positive*, but not a prediction of *negative*.

2nd-highest: Both previous heuristics solely consider the predicted class with the highest probability. Nevertheless, a prediction of 49% for the annotated class and 51% for a different class should be intuitively treated as "less wrong" than a predicted probability of 99% for a wrong class. Thus, the 2nd-highest heuristics assigns a stop probability to a node based on the difference to the correct class. We experienced that preferably stopping on high levels further improves the model. Therefore, the stop probability of a node is computed by $\left(1 - \frac{p_c}{\max_k p_k} \cdot \left(1 - \alpha^d\right)\right)$ where d is the node depth (top node has depth $d = 0$), $0 < \alpha < 1$ is a decay factor over depth, and p represents the predictions with p_c as predicted probability of the labeled class. Note that if the node predicts correctly, the stop probability equals zero.

4 Experiments

We evaluate our approach on the Stanford Sentiment Treebank (Socher et al., 2013) where the model's task is to classify a sentence into one

of the five fine-grained classes ranging from *very positive* to *very negative*. The train/val/test dataset contains 8544/1101/2210 sentences each provided with a full parse tree and labels at every node.

Next to all baseline models described in Section 3.1, we test three heuristics for our approach: *binary*, *1-distance* and *2nd-highest* ($\alpha = 0.25$), all with $\delta = 0.5$ and $\gamma = 0$. Similarly to Tai et al. (2015), we initialize the word representations by using the 300-dimensional Glove vectors (Pennington et al., 2014) for (Tree-) LSTM models. We also experienced that fine-tuning the embeddings improves the accuracy. In contrast to Tai et al. (2015), we implement the fine-tuning by an additional layer applied on word embeddings before entering the LSTM cell (300-dim., ReLU activation, dropout rate 0.25). This allows us to share the fine-tuning with unseen words.

For training, we apply the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3} for BoW models and $2 \cdot 10^{-4}$ for all others. On the input to the sentiment classifier in the LSTM and Tree-LSTM based models, we apply dropout (Srivastava et al., 2014) with a rate of 0.5. However, we experienced that a lower dropout rate of 0.25 for models trained with the proposed methods of Section 3.2 work better as it leads to less random stops. We compare our method to a Tree-LSTM trained on all nodes with equally weighted node loss as in (Tai et al., 2015). The same training settings are used as for the three heuristics. Our quantitative results are summarized in Table 1.

Method	**Test accuracy**	σ
BOW	36.1%	±1.9%
CBOW	36.2%	±1.2%
Deep CBOW	36.0%	±1.5%
Deep CBOW (Glove init)	44.0%	±1.0%
LSTM	46.7%	±0.8%
Tree-LSTM	47.6%	±0.9%
Tree-LSTM (fine-tuned)	48.3%	±0.6%
Tree-LSTM (all nodes)	49.1%	±0.6%
Our method		
- Binary	43.5%	±0.2%
- 1-distance	**50.5%**	±0.2%
- 2nd-highest	50.1%	±0.1%

Table 1: Test accuracies on the Stanford Sentiment Treebank. We report the mean accuracy over 10 runs for the baseline models and over 3 for the proposed model (limited due to longer runtime). The standard deviation is given on the right.

5 Results and Analysis

The initial results of the experiments show that for BoW models higher levels of complexity do not improve the accuracy, as all achieve about 36%. However, by using Glove initialized embeddings, the accuracy of Deep CBOW can be significantly increased to 44% which is consistent with the findings of Pennington et al. (2014)

By capturing dependencies between words, LSTMs achieve an accuracy of 46.7%. In contrast, Tree-LSTMs explicitly consider the sentence structure by the parse-tree and show superior results between 47.6% and 48.3% respectively for the fine-tuned Tree-LSTM. Contrary to our expectations, the LSTM that trained equally on all nodes, reaches 49.1% accuracy. This indicates that this model deals considerably well with the class imbalance. Still, the performance of our proposed method outperforms the benchmark. Furthermore, the significant difference in the accuracy of the three heuristics in Table 1 demonstrates that heuristics are a crucial element of the algorithm.

To analyze the model performances, we first compare the results and effects of the three proposed heuristics. We then look at how the models encode word structure and deal with sentences of different length. We finish our analysis by examining how our method helps to prevent overfitting.

5.1 Effect of different heuristics

The binary heuristic performs considerably worse (43.5%) compared to the other heuristics. By strictly discriminating correct from incorrect predictions, we stop too often at shallow levels which hinders the model to learn from the whole tree (see Figure 2). However, we experience that binary heuristic leads to a more balanced accuracy over classes as we train the nodes' classifier depending on the class accuracy instead of class count.

The 1-distance heuristic achieves the highest accuracy (50.5%) of all tested models on the Stanford Sentiment Treebank. In Figure 2, we see that this heuristic stops less often during training as the heuristic does not discriminate between the neighbouring classes. Furthermore, as the dataset is biased towards the classes *positive* and *negative*, our model also prefers one of these two classes. This converts the problem to a binary-like classifier within the tree. One of its effect is that especially the performance on neutral, long sentences of the 1-distance heuristic is considerably worse

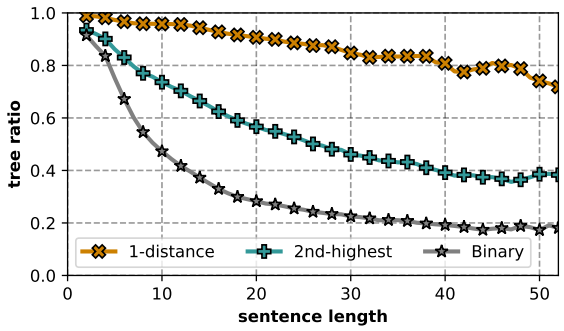

Figure 2: Tree exploration ratio over sentence length of training dataset after 50,000 iterations. Ratio is measured by the minimum height of a stopped node divided by tree height.

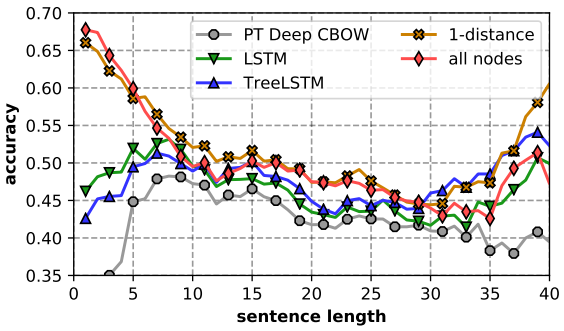

Figure 3: Comparing different model performances by accuracy over sentence length on test dataset. For each data point x, we plot the mean accuracy of test sentences of length $x \pm 2$ for noise reduction. We only plot the 1-distance heuristic as the other heuristics have shown a very similar behaviour.

compared to other sentence types.

Lastly, the 2nd-highest heuristic performs marginally worse than 1-distance (50.1%). Figure 2 indicates that the 2nd-highest heuristic on average reaches higher nodes than binary heuristic as we allow uncertainties in the prediction. However, compared to the 1-distance heuristic, we stop much earlier in a tree, especially for long sentences. This arises the question whether it is beneficial to ignore strong differences in the predicted class probabilities. In Section 5.3 we further examine this aspect and analyze its influence on regularization and overfitting within the model.

5.2 Word structure and sentence length

Word structure: To investigate how strongly a model considers the word structure for classification, we evaluate on the test dataset again but shuffle the words of a sentence before entering it into the network. Whereas, trivially, the BoW models are unaffected by any change in word order, the accuracy of the LSTM slightly drops by 3%. This demonstrates that LSTM only uses small elements of the sentence structure for its prediction. Our finding agrees with Everaert et al. (2015) arguing that sequential RNNs only slightly capture sentence structure as it is left implicit in the data.

On the other hand, Tree-LSTMs model sentence structures explicitly from the parse tree. As expected, shuffling a sentence lead to a 15% accuracy drop to 32% which is only 12% above a random guess. This demonstrates that Tree-LSTMs are able to retrieve semantically-useful information of the sentence structure. Our proposed training method does not influence the word structure, so that we retrieve similar results for those.

Sentence length: In most tasks, recurrent neural networks perform worse with increasing sequence length as more inputs must be processed correctly (Graves et al., 2014; Pascanu et al., 2013b). Thus, we would also expect that the accuracy of our models is negatively related to the sentence length, which is indeed the case. However, in Figure 3we see a slight increase in accuracy for sentences larger than 35 words due to higher noise by limited test sentences available. Most sentences in the training dataset have length between 10-30 words and thus all models have similar results and curvatures for this range. However, for larger sentences we see that the performance of Tree-LSTM reasonably diverges from LSTM and BoW supporting the hypothesis that Tree-LSTM deals better with complex sentence structures.

Tree-LSTM models with supervision at every node show significantly better performance on short sentences, because they were trained on much more short examples. Furthermore, our method also achieves superior performance on long sentence as while we ensure good classification on low level, we still don't lose focus on the top node. Nevertheless, the results of the benchmark models contrast with Tai et al. (2015) where their LSTMs have already stronger performance on short sentences. Thus we expect that they might have used all possible subtrees to train their LSTMs to ensure a fair comparison of the models.

5.3 Overfitting and generalization

Overfitting is a common problem in the context of Deep Learning where models adapt to noise in the training dataset, especially if it is relatively small (Srivastava et al., 2014). As indication of whether a model overfits, we look at the validation accuracy over training iterations visualized in Figure 4.

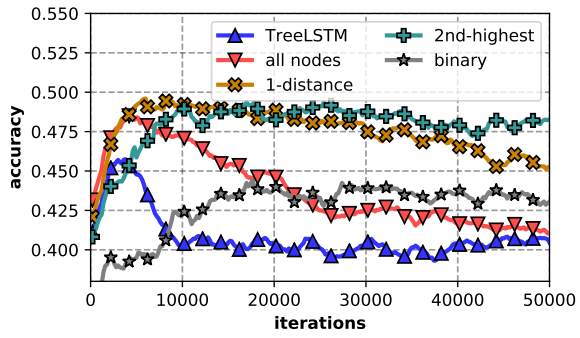

Figure 4: Validation accuracy over training iterations for a single training run. All models were trained with batch size 32 and evaluated every 200 iterations. For each data point x, we plot the average between $x \pm 10$ for noise reduction.

Both Tree-LSTM models, trained solely on top node and equally on all nodes, clearly overfit after 5,000 iterations leading to a low validation accuracy for later iterations. In contrast, models trained with our method seem to be less sensitive to overfitting. Especially the models trained with the binary and 2nd-highest heuristic keep a steady validation accuracy over time. This is also reflected by low standard deviations in the test accuracy.

The reason for the strong overfitting of previous Tree-LSTMs is that it can easily adapt to noise in the input and pass it through the tree to the final node without being punished (loss is solely calculated on the final prediction). However, when in our training method noise is passed through the tree, it will effect the predictions of all inner nodes as well and lead to misclassifications. Hence, we will prematurely stop at one of those nodes, train the model to correct its prediction (therefore getting rid of this noise) and thus, prevent the network from overfitting. This is in particular the case for the binary heuristic as it stops immediately at the first slight misclassification in the tree. The 1-distance heuristic, however, allows small amount of noise to pass as it just checks whether the prediction of a node is within the specified range of the label. For instance, if the final node of a sentence is *positive*, we would still allow the model to overfit on noise to predict all neutral nodes of this tree as *positive* as well and simplify the final classification. Nevertheless, the accuracy of the 1-distance model is much higher compared to the binary model as it does not require all inner nodes to be perfectly predicted. Hence, it still learns from more nodes even though it is slightly incorrect.

The benefits of binary and 1-distance are actually combined in the 2nd-highest heuristic. By considering the probability distribution over classes instead of just the class with the highest prediction, we prevent the problem of fully losing track of the correct class label. By that, the heuristic finds a trade-off between optimizing on the training examples and generalizing without adapting to noise. This is also one of the reasons for the higher accuracy on shorter phrases/sentences as discussed in the Section 5.2. Still, the performance of the 2nd-highest heuristic is slightly worse than of 1-distance. Nevertheless, this might change for other tasks and settings depending on properties like the test and training dataset distribution.

6 Conclusion

This paper discusses a bottom-up approach for efficient loss weighting in tree-structured neural networks. Our method focuses the loss on misclassified nodes. Additionally, we developed three heuristics that determine the extend of misclassification. This approach results in reasonable improvements in classification accuracy and overfitting performance compared to previous models. Although, Tai et al. (2015) achieved an accuracy of 51.0%, we were unable to replicate this result due to strong class imbalances in trees. We expect that they implemented a method to prevent this imbalance but this was left undiscussed in the paper.

One of the crucial parts of the algorithm has been proven to be the heuristic. Each has its own benefits and drawbacks, and hence should be carefully selected for the specific task. For instance, the 1-distance heuristic has achieved the best results in the given setting, but the 2nd-highest heuristic has favorable generalization properties.

Our paper represents a first step towards developing more effective heuristics as for example the hyperparameter space for parameters such as δ, γ and α has not been sufficiently explored yet. Furthermore, our method can easily be adapted to other domains as for instance the distance-based heuristics can be transformed to continuous output. Our method is however limited to problems where we can divide the task into sub-tasks on a node level, and the corresponding training dataset must provide labels for those nodes.

We hope that future work will further investigate this area of research as this paper shows that careful loss weighting is a crucial element for deep learning algorithms, especially in the context of natural language processing.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. pages 1545–1554. http://aclweb.org/anthology/N/N16/N16-1181.pdf.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. 2016. End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, pages 4945–4949.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014a. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 1724–1734. https://doi.org/10.3115/v1/D14-1179.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. pages 1724–1734. http://aclweb.org/anthology/D/D14/D14-1179.pdf.

Martin BH Everaert, Marinus AC Huybregts, Noam Chomsky, Robert C Berwick, and Johan J Bolhuis. 2015. Structures, not strings: linguistics as part of the cognitive sciences. *Trends in cognitive sciences* 19(12):729–743.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. 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)*. volume 1, pages 1681–1691.

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

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

Jiwei Li, Minh-Thang Luong, Dan Jurafsky, and Eudard Hovy. 2015. When are tree structures necessary for deep learning of representations? *arXiv preprint arXiv:1503.00185* .

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013a. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. pages 1310–1318.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013b. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. pages 1310–1318.

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.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss. *CoRR* abs/1603.05118. http://arxiv.org/abs/1603.05118.

Imran Sheikh, Irina Illina, Dominique Fohr, and Georges Linares. 2016. Learning word importance with the neural bag-of-words model. In *ACL, Representation Learning for NLP (Repl4NLP) workshop*.

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.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313.

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.