Learning Hierarchical Structures On-The-Fly with a Recurrent-Recursive Model for Sequences

Athul Paul Jacob[*†]
MILA
University of Waterloo

Zhouhan Lin[*†]
MILA
University of Montréal
AdeptMind Scholar

Alessandro Sordoni
Microsoft Research
Montréal, Canada

Yoshua Bengio
MILA
University of Montréal, CIFAR

Abstract

We propose a hierarchical model for sequential data that learns a tree on-the-fly, i.e. while reading the sequence. In the model, a *recurrent* network adapts its structure and reuses recurrent weights in a *recursive* manner. This creates adaptive skip-connections that ease the learning of long-term dependencies. The tree structure can either be inferred without supervision through reinforcement learning, or learned in a supervised manner. We provide preliminary experiments in a novel Math Expression Evaluation (MEE) task, which is explicitly crafted to have a hierarchical tree structure that can be used to study the effectiveness of our model. Additionally, we test our model in a well-known propositional logic and language modelling tasks. Experimental results show the potential of our approach.

1 Introduction

Many kinds of sequential data such as language or math expressions naturally come with a hierarchical structure. Sometimes the structure is hidden deep in the semantics of the sequence, like the syntax tree in natural language; Other times the structure is more explicit, as in math expressions, where the tree is determined by the parentheses.

Recurrent neural networks (RNNs) have shown tremendous success in modeling sequential data, such as natural language (Mikolov et al., 2010; Merity et al., 2017). However, RNNs process the observed data as a linear sequence of observations: the length of the computational path between any two words is a function of their position in the observed sequence, instead of their semantic or syntactic roles, leading to the appearance of difficult-to-learn long-term dependencies and stimulating research on strategies to deal with that (Bengio et al., 1994; El Hihi and Bengio, 1996; Hochreiter and Schmidhuber, 1997). Hierarchical, tree-like structures may alleviate this problem by creating shortcuts between distant inputs and by simulating compositionality of the sequence, compressing the sequence into higher-level abstractions. Models that use tree as prior knowledge (e.g. (Socher et al., 2013; Tai et al., 2015; Bowman et al., 2016)) have shown improved performances over sequential models, validating the value of tree structure. For example, TreeLSTM (Tai et al., 2015) learns a bottom-up encoder, but requires the model to have access to the entire sentence as well as its parse tree before encoding it, which limits its application in some cases, e.g. language modeling. There has been various efforts to learn the tree structure as a supervised training target (Dyer et al., 2016; Socher et al., 2010; Zhou et al., 2017; Zhang et al., 2015), which free the model from relying on an external parser.

More recent efforts learn the best tree structure without supervision, by minimizing the log likelihood of the observed corpus, or by optimizing over a downstream task (Williams et al., 2017). These models usually take advantage of a binary tree assumption on the inferred tree, which imposes restrictions on the flexibility of inferred tree structure, for example, Gumbel TreeLSTM (Choi et al., 2017; Yogatama et al., 2016).

We propose a model that reads sequences using a hierarchical, tree-structured process (Fig. 1): it creates a tree on-the-fly, in a top-down fashion. Our model sits in between fully recursive models that have access to the whole sequence, such as

[*] Equal contribution (Ordering determined by coin flip). Corresponding authors: zhouhan.lin@umontreal.ca, apjacob@edu.uwaterloo.ca

[†] Work done while at Microsoft Research, Montreal.

Proceedings of the 3rd Workshop on Representation Learning for NLP, pages 154–158
Melbourne, Australia, July 20, 2018.

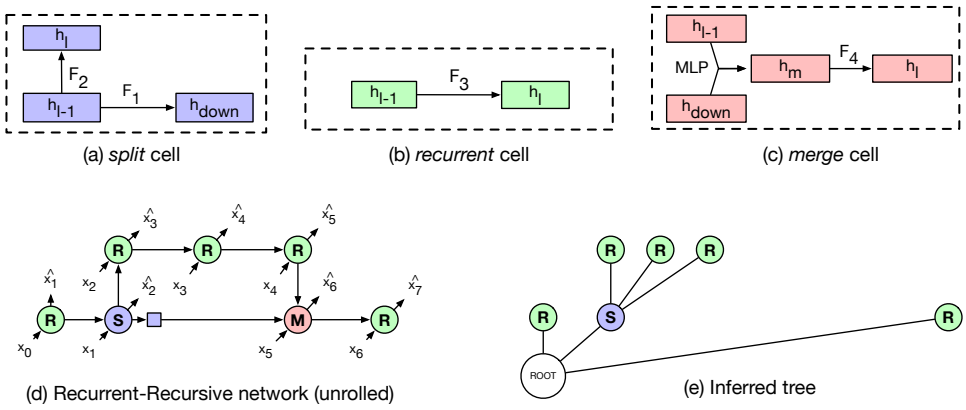

Figure 1: (a) - (c) are the 3 different cells. (d) is a sample model structure resulted from a sequence of decisions. "R", "S" and "M" stand for recurrent cell, split cell, and merge cell, respectively. Note that the "S" and "M" node can take inputs in datasets where splitting and merging signals are part of the sequence. For example, in math data the splitting signal are related to the brackets. (e) is the tree inferred from (d).

TreeLSTMs (Tai et al., 2015), and vanilla recurrent models that "flatten" input sequence, such as LSTMs (Schmidhuber, 1992). At each time-step in the sequence, the model chooses either to create a new sub-tree (split), to return and merge information into the parent node (merge), or to predict the next word in the sequence (recur). On split, a new sub-tree is created which takes control on which operation to perform. Merge operations end the current computation and return a representation of the current sub-tree to the parent node, which composes it with the previously available information on the same level. Recurrent operations use information from the siblings to perform predictions. Operations at every level in the tree use shared weights, thus sharing the recursive nature of TreeLSTMs. In contrast to TreeLSTMs however, the tree is created on-the-fly, which establishes skip-connections with previous tokens in the sequence and forms compositional representations of the past. The branching decisions can either be trained through supervised learning, by providing the true branching signals, or by policy gradients (Williams, 1992) which maximizes the log-likelihood of the observed sequence. As opposed to previous models, these three operations constantly change the structure of the model in an online manner.

Experimental evaluation aims to analyze various aspects of the model such as: how does the model generalize on sequences of different lengths than those seen during training? how hard is the tree learning problem? To answer those questions, we propose a novel multi-operation math expression evaluation (MEE) dataset, with a standard set of tasks with varying levels of difficulty, where the difficulty scales up with respect to the length of the sequence.

2 Model

Similar to a standard RNN, our model modifies a hidden state h_l for each step of the input sequence $x = \{x_1, \ldots, x_N\}$ by means of split, merge and recurrent operations. Denote the sequence of operations by $z = \{z_1, \ldots, z_L\}$, where L may be greater than the number of tokens N since only recurrent operations consume input tokens (see Fig. 1). Each operation is parametrized by a different "cell". A policy network controls which operation to perform during sequence generation.

split (S) The split cell creates a sub-tree by taking the previous state h_{l-1} as input and generating two outputs h_l and h_{down}. h_l is used for further computation, while h_{down} (the small blue rectangle in Fig. 1(d)) is pushed into a stack for future use. In our model, $h_{down} = F_1(h_{l-1}, x_t)$ and $h_l = F_2(h_{l-1}, x_t)$ where the F_1 and F_2 are LSTM units (Hochreiter and Schmidhuber, 1997), and x_t is the current input.

recurrent (R) This cell is a standard LSTM unit that takes as input the previous state h_{l-1} and the current token x_t, and outputs the hidden state h_l, which will be used to predict the next output $\hat{x}_{t+1}$. After application of this cell, the counter t is incremented and input x_t is consumed.

merge (M) The merge cell closes a sub-tree and returns control to its parent node. It does so by merging the previous hidden state h_{l-1} with the top of the stack h_{down} into a new hidden state $h_m = MLP(h_{l-1}, h_{down})$ (Fig. 1(c)). h_m is then used as input to another LSTM unit (F_4) to yield $h_l = F_4(x_t, h_m)$, the new hidden state of the overall network. Intuitively, h_{l-1} summarizes the contents within the sub-tree. This is merged with information obtained before the model entered the sub-tree h_{down} into the new state h_l.

Policy Network We consider the decision at each timestep $z_t \in \{S, M, R\}$ as a categorical variable sampled from a policy network p_π, conditioned on the hidden state h_t and the input embedding e_t of the current input x_t. In the supervised setting, $p_\pi(z_t|e_t, h_t)$ is trained by maximizing the likelihood of the true branching labels, while in the unsupervised setting, we resort to the REINFORCE algorithm using $-\log p(y_t|C)$ as a reward, where y_t is the task target (i.e. the next word in language modeling), and C is the representation learnt by the model up until time t.

3 Experimental Results

We conduct our experiments on a math induction task, a propositional logic inference task (Bowman et al., 2016) and language modelling. First of all, we aim to investigate whether a) our hierarchical model may help in tasks that explicitly exhibit hierarchical structure, and then b) whether the trees learned without supervision correspond to the ground-truth trees, c) how our model fare with respect to hierarchical models that have access to the whole sequence with a pre-determined tree structure and finally, d) are there any limitations for models that are not capable of learning hierarchical structures on-the-fly.

3.1 Math Induction

Our math expression evaluation dataset (MEE) consists of parenthesized mathematical expressions and their corresponding evaluations. The math expressions contain bracketing symbols ("()"), four different kinds of operations, "+-*%", where "%" is the modulo operation, and digits from 0 to 9. The "length" of an expression is the number of operations in the expression and its result is restricted to be a positive, two-digit integer (Table 1). We randomly generate expressions of different lengths and for each length the resulting sub-dataset is divided into 100,000, 10,000 and 10,000 expressions as training, valid and test sets. We make sure that there is no overlap between the splits and every expression is made unique across the whole dataset.

Length	Expression	Value
4	((9+(2+6))+(1*3))	20
5	(((7-2)%((3%1)+6))*9)	45
6	((((3-0)+(7-6))*(0+9))-7)	29
7	((4*(6+(7*(2*8))))%(9+(3+7)))	16

Table 1: Sample expressions from MEE dataset

We use an encoder-decoder approach where the encoder reads the characters in the expression and produces the encoding as input to the decoder, which in turn sequentially generates 2 digits as the predicted value. We experiment on various encoders, including our model, and compare their performances. We use the same decoder architecture to ensure a fair comparison. The output of the encoder is provided as the initial hidden state of the decoder LSTM. To test the generalization of our model, for all the experiments shown in this subsection, we train the model on expressions of length 4 and 5, and evaluate on expressions of length 4 to 7 in the test sets.

For this task, our baseline is a simple *LSTM* encoder (which corresponds to our model with only recur operations). We compare two versions of our *RRNet* encoder. In the supervised setting, we force the model to split and merge when it reads "(" and ")", respectively, and recur otherwise. This gives us an idea on how well the model would perform if it had access to the ground-truth tree. In the unsupervised setting, we learn the tree using policy gradient, where the reward is the accuracy of the math result prediction.

The results are in Table 2. The *supervised RRNet* yields the best performance showing that (a) it is important to exploit the hierarchical structure of the observed data, corroborating previous work (Williams et al., 2017), and (b) our model is effective at capturing that information. The *unsupervised RRNet* model also outperforms the baseline LSTM: the model learn to exploit branching operations to achieve better performance. We observe that the trees produced by the model do not correspond to the ground-truth trees. In order to assess whether the additional parameters of split and merge operations, rather than the learned tree

Model	Train	Test			
		L=4	L=5	L=6	L=7
LSTM	75.80	81.32	67.65	52.70	41.35
Uns RRNet	86.00	89.42	77.96	61.34	50.46
Sup RRNet	93.70	93.28	86.69	79.09	72.70

Table 2: Prediction accuracy on MEE dataset.

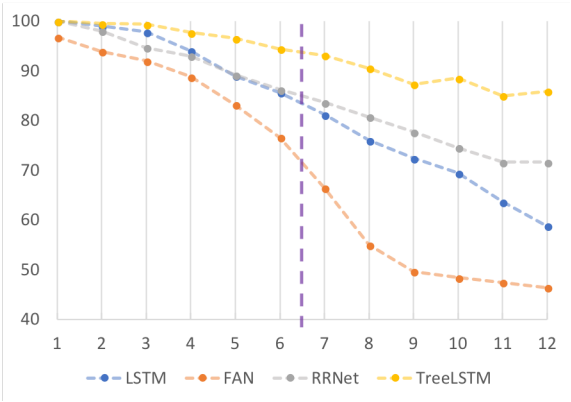

Figure 2: Test accuracy of the models, trained on sequences of length $\leq$ 6 in logic data. The horizontal axis indicates the length of the sequence, and the vertical axis indicates the accuracy of model's performance on the corresponding test set.

structure, produce better results, we measured the performance of our model trained with "random" deterministic policies (associating each of the input characters to either a split, merge or recur operation). We see that "random" policies perform worse than a "learnt" policy on the task, effectively similar to the baseline LSTM. In turn, the model with the learnt policy underperforms the model trained with ground-truth trees. Even in this seemingly easy task, it has appeared difficult for the model to learn the optimal branching policy.

3.2 Logical inference

In the next task, we analyze performance on the artificial language as described in Bowman et al. (2015b). This language has six word types $\{a, b, c, d, e, f\}$ and three logical operations $\{or, and, not\}$. There are seven mutually exclusive logical relations that describe the relationship between two sentences: entailment ($\sqsubset$, $\sqsupset$), equivalence ($\equiv$), exhaustive and non-exhaustive contradiction ($^\wedge$, $|$), and two types of semantic independence ($\#$, $\smile$). The train/dev/test dataset ratios are set to 0.8/0.1/0.1 as described [1] with the number of logical operations ranging from 1 to 12.

From Figure 2, we report the performance of our model when trained with ground-truth trees as input. It is encouraging to see that our recurrent-recursive encoder improves performance over Transformer (FAN) (Tran et al., 2018) and LSTM, especially for long sequences. The best performance on this dataset is given by TreeLSTM, which has access to the whole sequence (Bowman et al., 2015b) and does not encode sequences on-the-fly.

3.3 Language Modelling

In language modeling, architectures such as TreeLSTM aren't directly applicable since their structure isn't computed on-the-fly, while reading the sentence. We perform preliminary experiments using the Penn Treebank Corpus dataset (Marcus et al., 1993), which has a vocabulary of 10,000 unique words and 929k, 73k and 82k words in training, validation and test set respectively. Our cells use one layer and the hidden dimensionality is 350. Our model yields test perplexity of 107.28 as compared to the LSTM baseline which gets 113.4 (Dyer et al., 2016). This preliminary result shows that the endeavor to exploit explicit hierarchical structures for language modeling, although challenging, may be promising.

4 Final Considerations

In this work, we began exploring properties of a recurrent-recursive neural network architecture that learns to encode the sequence on-the-fly, i.e. while reading. We argued this may be an important feature for tasks such as language modeling. We additionally proposed a new mathematical expression evaluation dataset (MEE) as a toy problem for validating the performance of sequential models to learn from hierarchical data. We empirically observed that, in this task, our model performs better than a standard LSTM architecture with no explicit structure and also outperforms the baseline LSTM and FAN architectures on the propositional logic task.

We hope to further study the properties of this model by either more thorough architecture search (recurrent dropouts, layer norm, hyper-parameter sweeps), different variation of RL algorithms such

[1] https://github.com/sleepinyourhat/vector-entailment

as deep Q-learning (Mnih et al., 2013) and employing this model on various other tasks such as SNLI (Bowman et al., 2015a) and semi-supervised parsing.

Acknowledgement

The authors would like to thank Compute Canada for providing the computational resources. Zhouhan Lin would like to thank AdeptMind for generously supporting his research via scholarship.

References

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

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015a. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding .

Samuel R. Bowman, Christopher D. Manning, and Christopher Potts. 2015b. Tree-structured composition in neural networks without tree-structured architectures. *CoRR* abs/1506.04834.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. 2017. Learning to compose task-specific tree structures. *arXiv preprint arXiv:1707.02786* .

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *Proceedings of ACL* .

Salah El Hihi and Yoshua Bengio. 1996. Hierarchical recurrent neural networks for long-term dependencies. In *Proceedings of NIPS*.

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

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182* .

Tomas Mikolov, Martin Karafit, Luks Burget, Jan Cernock, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*. ISCA, pages 1045–1048.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Jürgen Schmidhuber. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Computation* 4(2).

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*. pages 1–9.

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.

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* .

K. Tran, A. Bisazza, and C. Monz. 2018. The Importance of Being Recurrent for Modeling Hierarchical Structure. *ArXiv e-prints* .

Adina Williams, Andrew Drozdov, and Samuel R Bowman. 2017. Learning to parse from a semantic objective: It works. is it syntax? *arXiv preprint arXiv:1709.01121* .

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. 2016. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100* .

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060* .

Ganbin Zhou, Ping Luo, Rongyu Cao, Yijun Xiao, Fen Lin, Bo Chen, and Qing He. 2017. Generative neural machine for tree structures. *arXiv preprint arXiv:1705.00321* .