Memory Augmented Recursive Neural Networks

March 19, 2019

1 Introduction

Recursive neural networks such as Tree LSTMS [Tai et al., 2015] and Tree NNs [Socher et al., 2011] have shown promising results in modeling compositional data such as sentences that can be represented by parse trees, computer programs represented by their abstract syntax tree and mathematical equations [Zaremba et al., 2014, Allamanis et al., 2017, Arabshahi et al., 2018]. However, their performance degrades drastically when generalizing to expressions of higher depth. This suggests that information might be lost as it propagates from the leafs to the tree root. Therefore, we propose augmenting these networks with additional memory to overcome this issue.

Inspired by the recent work on augmenting neural networks with abstract queues and stacks [Grefenstette et al., 2015, Sun et al., 2017, Joulin and Mikolov, 2015, Jason Weston, 2015, Kumar et al., 2016, Sukhbaatar et al., 2015, Graves et al., 2014] we propose to augment recursive neural networks such as Tree-NNs and Tree-LSTMs [Tai et al., 2015] with external differentiable memory such as neural stacks and queues.

Works such as [Pham et al., 2018, Fernando et al., 2018] construct a memory with a specific structure and are different from our work.

2 Memory augmented recursive neural networks

2.1 Memory augmented Tree-NNs

We will start with simple Tree NNs. The memory augmented Tree NN is presented below. We will present equations for a given node j. For the sake of simplicity we consider only binary trees, but the equations hold for n-ary trees. Therefore, we denote the children of j with C_{j1} and C_{j2}. Ordinary Tree-NNs without the stack are given below

$$I_j = [O_{C_{j1}}; O_{C_{j2}}] \tag{1}$$
$$O_j = f(W_j I_j + b_j) \tag{2}$$

where $[\cdot\ ;\ \cdot]$ indicates concatenation and f is a nonlinear function such as Sigmoid. $O_j \in \mathbb{R}^n$ and $W_j \in \mathbb{R}^{n\times 2n}$. We will now augment each node with a stack and present the update equations accordingly.

Each node j is augmented with a stack $s_j \in \mathbb{R}^p$. We indicate the top of the stack with $s_j[0]$. The stack has two operations pop and push. We use the 2-dimentional vector a_j to indicate the probability of popping and pushing. These probabilities depend on the input vectors.

$$a_j = g(A_j I_j) \tag{3}$$

where $A_j \in \mathbb{R}^{2\times 2n}$ and g is the softmax function. We denote the probability of push with $a_j[\text{push}] = a_j[0] \in [0, 1]$ and the probability of pop with $a_j[\text{pop}] = a_j[1] \in [0, 1]$. Based on these probabilities we have the stack update equations given below.

$$s_j[0] = a_j[\text{push}]\ \sigma(D_j I_j) + a_j[\text{pop}]\ \sigma(D_{j\,\text{pop}}(s_{C_j}[1])) \tag{4}$$
$$s_j[i] = a_j[\text{push}]\ \sigma(D_{j,\text{push}} s_{C_j}[i-1]) + a_j[\text{pop}]\ (D_{j,\text{pop}}(s_{C_j}[i+1])) \tag{5}$$

where $D_j \in \mathbb{R}^{2n}$, $D_{j,\text{push}}, D_{j,\text{pop}} \in \mathbb{R}^2$ are trainable weight vectors for node j and is shared between all nodes of type j. To be more clear, if j is a node that represents addition, all node of type addition will use the same weights. $s_{C_j}[i] \in \mathbb{R}^2$ is the status of the i^{th} element of the children's stack and is given below:

$$s_{C_j}[i] = [s_{j1}[i]; s_{j2}[i]] \tag{6}$$

Given the stack the update equations for each node j will be modified as

$$O_j = f(W_j I_j + P_j s^k_{C_j}) \tag{7}$$

where $s^k_{C_j} = [s^k_{C_1}; s^k_{C_2}]$ indicates the top k elements of the children's stack and $P_j \in \mathbb{R}^{n \times 2k}$.

2.2 Memory augmented Tree LSTMS

In this section we present the stack equations for binary tree LSTMs, however these extend naturally to n-ary Tree LSTMS and child-sum tree-LSTMs. For the sake of simplicity we will assume that the tree-LSTM nodes do not have an external input. However, external inputs can be easily added to the equations. For an ordinary binary Tree-LSTM we have the following update equations for node j

$$i_j = \sigma(U_1^{(i)} h_{j1} + U_2^{(i)} h_{j2} + b^{(i)}) \tag{8}$$
$$f_{jk} = \sigma(U_{1k}^{(f)} h_{j1} + U_{2k}^{(f)} h_{j2} + b^{(f)}) \tag{9}$$
$$o_j = \sigma(U_1^{(o)} h_{j1} + U_2^{(o)} h_{j2} + b^{(o)}) \tag{10}$$
$$u_j = \tanh(U_1^{(u)} h_{j1} + U_2^{(u)} h_{j2} + b^{(u)}) \tag{11}$$
$$c_j = i_j \odot u_j + f_{j1} \odot c_{j1} + f_{j2} \odot c_{j2} \tag{12}$$
$$h_j = o_j \odot \tanh(c_j) \tag{13}$$

where $\odot$ indicates elementwise multiplication. In order to augment tree LSTMs with memory we keep a stack for the cell state and a stack for the hidden state. Similar to Tree-NN we will use the cell state and the hidden state to decide about the push and pop operations with a probability. We have

$$a_{cj} = g(A_{cj} c_j) \tag{14}$$
$$a_{hj} = g(A_{hj} h_j) \tag{15}$$

where g is the softmax function, $A_{cj}, A_{h_j} \in \mathbb{R}^{2 \times m}$. Similar to Tree-NNs $a_{cj}, a_{hj} \in [0,1]^2$ and the first and second elements in each vector represents the probability of pushing and popping, respectively.

The stack update equations for the cell memory and the hidden state for node j are given below.

$$s_{cj}[0] = a_{cj}[\text{push}]\ \sigma(D_{cj} c_j) + a_{cj}[\text{pop}]\ \sigma(D_{cj,\text{pop}}(s_{cCj}[1])) \tag{16}$$
$$s_{cj}[i] = a_{cj}[\text{push}]\ \sigma(D_{cj,\text{push}} s_{cC_j}[i-1]) + a_{cj}[\text{pop}]\ (D_{cj,\text{pop}}(s_{cC_j}[i+1])) \tag{17}$$
$$s_{hj}[0] = a_{hj}[\text{push}]\ \sigma(D_{hj} h_j) + a_{hj}[\text{pop}]\ \sigma(D_{hj,\text{pop}}(s_{hCj}[1])) \tag{18}$$
$$s_{hj}[i] = a_{hj}[\text{push}]\ \sigma(D_{hj,\text{push}} s_{hC_j}[i-1]) + a_{hj}[\text{pop}]\ (D_{hj,\text{pop}}(s_{hC_j}[i+1])) \tag{19}$$

where $D_{cj} \in >$ and $D_{cj,\text{push}}, D_{cj,\text{pop}}, D_{hj,\text{push}}, D_{hj,\text{pop}} \in \mathbb{R}^2$. $s_{cC_j}[i], s_{hC_j}[i] \in \mathbb{R}^2$ is the status of the i^{th} element of the children's cell memory and hidden state stack, respectively and is given below:

$$s_{cC_j}[i] = [s_{cj1}[i]; s_{cj2}[i]] \tag{20}$$
$$s_{hC_j}[i] = [s_{hj1}[i]; s_{hj2}[i]] \tag{21}$$

incorporating the stack will update the equation for h_i and c_j in Equations (13) and (12) as below

$$c_j = i_j \odot u_j + f_{j1} \odot c_{j1} + f_{j2} \odot c_{j2} + \sigma(P_{cj} s^k_{cC_j}) \tag{22}$$
$$h_j = o_j \odot \tanh(c_j) + \sigma(P_{hj} s^k_{hC_j}) \tag{23}$$

where $s^k_{cC_j} = [s^k_{cj1}; s^k_{Cj2}]$ and $s^k_{hC_j} = [s^k_{hj1}; s^k_{hj2}]$ indicates the top k elements of the children's cell memory and hidden state stacks, respectively and $P_{cj}, P_{hj} \in \mathbb{R}^{m \times 2k}$ that will be shared among all appearances of the type of node j.

References

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 80–88. JMLR. org, 2017.

Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Combining symbolic expressions and black-box function evaluations in neural programs. 2018.

Tharindu Fernando, Simon Denman, Aaron McFadyen, Sridha Sridharan, and Clinton Fookes. Tree memory networks for modelling long-term temporal dependencies. *Neurocomputing*, 304:64–81, 2018.

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

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

Antoine Bordes Jason Weston, Sumit Chopra. Memory networks. 2015.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in neural information processing systems*, pages 190–198, 2015.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

Trang Pham, Truyen Tran, and Svetha Venkatesh. Graph memory networks for molecular activity prediction. *arXiv preprint arXiv:1801.02622*, 2018.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

Guo-Zheng Sun, C Lee Giles, Hsing-Hen Chen, and Yee-Chun Lee. The neural network pushdown automaton: Model, stack and learning simulations. *arXiv preprint arXiv:1711.05738*, 2017.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Wojciech Zaremba, Karol Kurach, and Rob Fergus. Learning to discover efficient mathematical identities. In *Advances in Neural Information Processing Systems*, pages 1278–1286, 2014.