Supplementary Materials: On Tree-Based Neural Sentence Modeling

Haoyue Shi †,‡ **Hao Zhou**‡ **Jiaze Chen**‡ **Lei Li**‡
†: School of EECS, Peking University, Beijing, China
hyshi@pku.edu.cn
‡: ByteDance AI Lab, Beijing, China
{zhouhao.nlp, chenjiaze, lileilab}@bytedance.com

1 Implementation Details

Our codebase is built on PyTorch 0.3.0.[1] All the sentences was tokenized with SpaCy.[2]

1.1 Sentence Encoding

We use LSTM based sentence encodings as the extracted features of sentences for downstream classification or generation tasks. We use typical long short term memory (LSTM; Hochreiter and Schmidhuber, 1997) units for linear structures, which can be summarized as:

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \tanh(\boldsymbol{c}_t)
\end{aligned}$$

where t indicates the time step of a state; $\boldsymbol{h}_t$ is the hidden state and $\boldsymbol{x}_t$ is the input vector. We apply binary tree LSTM units adapted from Zhu et al. (2015) for binary tree LSTMs, which can be summarized as:

$$\begin{aligned}
\boldsymbol{f}_l &= \sigma(\boldsymbol{W}_l \cdot [\boldsymbol{h}_l, \boldsymbol{h}_r] + \boldsymbol{b}_l) \\
\boldsymbol{f}_r &= \sigma(\boldsymbol{W}_r \cdot [\boldsymbol{h}_l, \boldsymbol{h}_r] + \boldsymbol{b}_r) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \cdot [\boldsymbol{h}_l, \boldsymbol{h}_r] + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \cdot [\boldsymbol{h}_l, \boldsymbol{h}_r] + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_l \boldsymbol{c}_l + \boldsymbol{f}_r \boldsymbol{c}_r + \boldsymbol{i}_t \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \tanh(\boldsymbol{c}_t)
\end{aligned}$$

where the subscript t denotes the current state, and l, r denote the left and right child states respectively. We also apply LSTM (Hochreiter and Schmidhuber, 1997) as leaf-node RNN when necessary.

It is worth noting that left-branching tree LSTM without leaf-node RNN is structurally equivalent to unidirectional LSTM. The only difference between them, which may cause the slight difference on performance, comes from the implementation of LSTM units.

The candidate set of dropout ratio we explore for the task of word-level semantic relation (WSR) is $\{0, 0.1, 0.15, 0.2, 0.3, 0.5\}$.

1.2 Sentence Relation Classification

In the task of sentence relation classification, the feature vector consists of the concatenation of two sentence vectors, their difference, and their element-wise product (Mou et al., 2016):

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \boldsymbol{s}_1 - \boldsymbol{s}_2 \\ \boldsymbol{s}_1 \odot \boldsymbol{s}_2 \end{pmatrix}$$

1.3 Pooling Mechanism

Following (Socher et al., 2011), we apply pooling mechanism to all leaf states (of tree LSTMs) and hidden states. The detailed pooling methods are described as follows.

Max Pooling. Max pooling takes the max value for each dimension

$$\begin{aligned}
\boldsymbol{H} &= (\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_m) \\
\boldsymbol{s}_i &= \max_{j=1}^{m} h_{j,i} \quad i = 1, 2, \cdots, d \\
\boldsymbol{s} &= (\boldsymbol{s}_1, \boldsymbol{s}_2, \cdots, \boldsymbol{s}_d)^T
\end{aligned}$$

where $\boldsymbol{h}_i$ denotes a leaf state in tree LSTMs or a hidden state; $m = 2n-1$ for tree LSTMs and $m = n$ for linear LSTMs; $\boldsymbol{s}$ denotes the final sentence encoding.

[1] https://pytorch.org/docs/0.3.0
[2] https://spacy.io

Mean Pooling. Mean pooling (average pooling) takes the average of all hidden states as the sentence representation, which can be summarized as:

$$\boldsymbol{H} = (\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_m)$$
$$\boldsymbol{s} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{h}_i$$

Self-Attention. We follow Conneau et al. (2017) and Lin et al. (2017) to build a self-attentive mechanism, which can be summarized as:

$$\boldsymbol{H} = (\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_m)$$
$$\boldsymbol{a} = \mathrm{softmax}(\mathbf{w}_\beta^T \tanh(\mathbf{W}_\alpha \boldsymbol{H}))$$
$$\boldsymbol{s} = \boldsymbol{H}\boldsymbol{a}^T$$

where $\boldsymbol{a}$ denotes attention weights computed by learned parameters $\mathbf{W}_\alpha$ and $\mathbf{w}_\beta$. In all experiments, $\mathbf{w}_\beta$ is a 128-d vector.

References

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proc. of EMNLP*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A Structured Self-Attentive Sentence Embedding. In *Proc. of ICLR*.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. In *Proc. of ACL*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Christopher D. Manning, and Andrew Y. Ng. 2011. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Proc. of NIPS*.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long Short-Term Memory Over Recursive Structures. In *Proc. of ICML*.