Pre-Training a Graph Recurrent Network for Language Representation

Yile Wang¹, Linyi Yang¹, Zhiyang Teng¹, Ming Zhou², Yue Zhang¹
¹Westlake University
²Langboat Technology, Beijing, China
{wangyile, yanglinyi, tengzhiyang, zhangyue}@westlake.edu.cn
zhouming@chuangxin.com

Abstract

Transformer-based models have gained much advance in recent years, becoming one of the most important backbones in natural language processing. Recent work shows that the attention mechanism in Transformer may not be necessary, both convolutional neural networks and multi-layer perceptron based models have been investigated as Transformer alternatives. In this paper, we consider a graph recurrent network for language model pre-training, which builds a graph structure for each sequence with local token-level communications, together with a sentence-level representation decoupled from other tokens. We find such architecture can give comparable results against Transformer-based ones in both English and Chinese language benchmarks. Moreover, instead of the quadratic complexity, our model has linear complexity and performs more efficiently during inference.

1 Introduction

Pre-trained models (PTMs) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] have been widely used in natural language processing (NLP), benefiting a range of tasks including language understanding [11, 12], question answering [13, 14], and dialogue [15, 16]. The dominant methods take the Transformer [17] architecture, a heavily engineered model based on a self-attention network (SAN), it also showing competitive performance in computation vision [18, 19, 20], speech [21], and biological [22] tasks. Despite its success, Transformer typically suffers from quadratic time complexity [17], along with the requirement of large computational resources and associated financial and environmental costs [23]. In addition, recent studies show that the attention mechanism, which is the key ingredient of Transformer, may not be necessary [24, 25, 26]. For example, Tay et al. [26] find that models learning synthetic attention weights without token-token interactions also achieve competitive performance for certain tasks. Therefore, investigation of Transformer alternatives is of both theoretical and practical interest. To this end, various non-Transformer PTMs have recently been proposed [27, 28, 29, 30].

In this paper, we consider a graph neural network (GNN) [31] for language model pre-training. GNN and its variants have been widely used in NLP tasks, including machine translation [32], information extraction [33], and sentiment analysis [34]. For GNN language modeling, a key problem is how to represent a sentence in a graph structure. From this perspective, ConvSeq2seq [35] can be regarded as a graph convolutional network (GCN) [36] with node connections inside a local kernel. Transformer-based models can be regarded as a graph attention network (GAT) [37] with a full node connection. However, graph recurrent network (GRN) [38, 39] models have been relatively little considered.

We follow the structure of sentence-state LSTM (S-LSTM) [38], which represents a sentence using a graph structure by treating each word as a node, together with a sentence state node. State transitions

¹We release the code at https://github.com/ylwangy/slstm_pytorch

are performed recurrently to allow token nodes to exchange information with their neighbors and the sentence-level node. Such architecture has shown advantages over vanilla bidirectional LSTM in supervised text classification tasks. However, its potential for general-purpose language model pre-training has not been fully exploited. We optimize the model by exploring the suitable architecture design for pre-training, a comparison of our model and typical existing PTMs is shown in Table 1. Experimental results show that our model can give a comparable performance on general language understanding tasks for both English and Chinese languages. During inference, our model can gain 2~3 times speedup or more for extra long sentences against Transformer-based models. To our knowledge, we are the first to investigate a graph recurrent network for language model pre-training.

### 2 Model

The overall structure of our model is shown in Figure 1a. Following S-LSTM [38], we treat each sentence as a graph with token nodes and an external sentence state node. The node state is updated in parallel according to the information received in each layer (or recurrent step).

We first transform each token $w_i$ into token embedding using the trainable lookup table $E$ and the position embedding lookup table $P$, the model input $x_i$ is constructed by $x_i = E(w_i) + P(w_i)$. Then we initialize hidden states and hidden cells for each token node, the sentence-level node with $h_0, h_2, ..., h_n, g^0$ and $c_0, c_2, ..., c_j, c_g$, respectively. In each layer $t$ ($t = 1, 2, ..., L$), the token node states $h^t_i$ is calculated using gating mechanism similar with LSTM:

$$
\begin{align*}
\xi_t^{i-1} &= h_t^{i-1} \parallel h_t^{i+1} \\
\hat{g}_t &= \sigma(\text{LayerNorm}(W_\xi \xi_t^{i-1} + U_i x_i + V_g g_t^{i-1} + b_i)) \\
\hat{f}_t &= \sigma(\text{LayerNorm}(W_f \xi_t^{i-1} + U_i x_i + V_f g_t^{i-1} + b_f)) \\
\hat{i}_t &= \sigma(\text{LayerNorm}(W_i \xi_t^{i-1} + U_i x_i + V_i g_t^{i-1} + b_i)) \\
\hat{o}_t &= \sigma(\text{LayerNorm}(W_o \xi_t^{i-1} + U_i x_i + V_o g_t^{i-1} + b_o)) \\
\hat{h}_t &= \tanh(\text{LayerNorm}(W_h \xi_t^{i-1} + U_i x_i + V_h g_t^{i-1} + b_h)) \\
\hat{c}_t &= \hat{h}_t \odot \hat{c}_t^{i-1} + \hat{i}_t \odot \hat{c}_t^{i-1} + \hat{r}_t \odot \hat{c}_t^{i+1} + \hat{s}_t \odot \hat{c}_t^{g-1} + \hat{c}_t \odot u_i \\
\hat{h}_t &\parallel \odot \tanh(c_t^{i-1}) \\
\end{align*}
$$

where $\parallel$ is concatenation operation, $\xi_t^{i-1}, x_i, g_t^{i-1}$ represent the inputs from previous local states, token embedding and previous global states, respectively. In Eq. 1, we calculate multiple LSTM-style gates to control the corresponding information flow. $\hat{f}_t, \hat{i}_t, \hat{o}_t, \hat{h}_t$ are the forget gates with respect to the left token cell $c_t^{i-1}$, right token cell $c_t^{i+1}$, current token cell $c_t^{i-1}$, and sentence cell $c_t^{g-1}$. $\hat{f}_t, \hat{o}_t$ are
Inference Layer
Encoding Layer

Figure 1: Left: (a) Architecture of our model. We only show the update of token node $h_i$ and sentence-level node $g$ for brevity. Right: (b) Comparison with different model architectures.

the input gate and output gate, respectively. Layer normalization is used to control the distributions of neurons in each gate. $W_x, U_x, V_x,$ and $b_x$ ($x \in \{i, l, r, s, o, u\}$) are model parameters. The generated hidden states $h_{ti}$ and $g_{ti}$ are sent to the next layer, together with the memory state $c_{ti}$ and $c_{tg}$.

$$
\begin{align}
\tilde{h} &= \text{avg}(h_{t1}^{i-1}, h_{t2}^{i-1}, ..., h_{tn}^{i-1}) \\
\hat{f}^{i}_1 &= \sigma(\text{LayerNorm}(W_f g_{t1}^{i-1} + U_f h_{t1}^{i-1} + b_f)) \\
\hat{f}^{i}_g &= \sigma(\text{LayerNorm}(W_g g_{t1}^{i-1} + U_g \tilde{h} + b_g)) \\
o^{i} &= \sigma(\text{LayerNorm}(W_o g_{t1}^{i-1} + U_o \tilde{h} + b_o)) \\
f^{1}_1, ..., f^{1}_n, f^{g} &= \text{softmax}(\hat{f}^{1}_1, ..., \hat{f}^{1}_n, \hat{f}^{g}) \\
c^{t-1}_1, ..., c^{t-1}_n, c^{g} &= f^{1}_t \odot c^{t-1}_1 + ... + f^{1}_t \odot c^{t-1}_n \\
g^{t} &= o^{i} \odot \tanh(c^{t}) 
\end{align}
$$

Figure 1(b) shows the ways of hidden states generations of our model and other architectures. Different from CNN, we explicitly model sentence-level information as a feature for each token, which provides global information. Compared with Transformer, the sentence-level node representation is designed to be separated from other tokens. We make all the trainable parameters in Eq. 1 and Eq. 2 shared across GNN layers, which is similar to the parameters in LSTM along the sequence direction. In our model, we update each token using its fixed local context together with a sentence-level representation in each layer, making our model has $O(n)$ complexity.

3 Experiments

3.1 Pre-training

Dataset. English models are trained using the latest Wikipedia and BookCorpus [42]. Chinese models are trained using Wikipedia. The total amount of training data is on par with BERT [2] for both languages.

Baselines. Strictly comparing the PTMs is difficult because of the different dataset processing, training strategies, and environmental settings. As shown in Table 3, we consider the most related and popular models with similar training corpus for comparison. For English, we use the published RNN-based model (ELMo), compact version of BERT (DistilBERT), BERT, and recurrent version of BERT (ALBERT). For Chinese, we add some BERT variants which use Chinese word segmentor for whole word masking [44] or modify the masked token prediction as a correction target [46].

Settings. We pre-train our model with a batch size of 128 and a maximum length of 512 for 300k steps, using Adam optimizer with learning rate $lr=0.003$, $\beta_1=0.9$, $\beta_2=0.98$, learning rate warmup
<table>
<thead>
<tr>
<th>Models (English)</th>
<th>Pre-training Data</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELMo [41]</td>
<td>1 Billion Word</td>
<td>CLM</td>
</tr>
<tr>
<td>DistilBERT [42]</td>
<td>Wiki+BooksCorpus</td>
<td>BERT+KD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Models (Chinese)</th>
<th>Data</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-wwm [44]</td>
<td>Wiki</td>
<td>MLM+NSP</td>
</tr>
<tr>
<td>RoBERTa-wwm [44]</td>
<td>Wiki+CLUECorpus</td>
<td>MLM</td>
</tr>
<tr>
<td>RoBERTa-wwm-ext [44]</td>
<td>Wiki+EXT</td>
<td>MLM</td>
</tr>
<tr>
<td>MacBERT [46]</td>
<td>Wiki+EXT</td>
<td>Mac+SOP</td>
</tr>
</tbody>
</table>

Table 2: Baseline models. CLM: casual language modeling. KD: knowledge distillation. SOP: sentence order prediction. Mac: MLM as correction. wwm: whole word masking. ext: external training data.

3.2 Fine-tuning

Evaluating Benchmarks. For English tasks, we evaluate our pre-trained models on tasks in GLUE [11], including linguistic acceptability (CoLA), sentiment analysis (SST), sentence pair similarity (MRPC, QQP), and natural language inference (MNLI, QNLI, RTE).

For Chinese tasks, we evaluate on short and long text classification (TNEWS, IFLYTEK), keywords matching (CSL) [48], question matching (LCQMC) [49], document classification (THUCNews) [50] and sentiment analysis (ChnSentiCorp) [51].

Settings. Although our model architecture is different from most baselines, the fine-tuning strategy for each task can be the same as BERT-style models. As shown in Figure 2, the output of the sentence node $g$ can be treated as the representation of [CLS] in BERT, which can be used for single sentence classification tasks directly. For sentence pair classification tasks, we concatenated two sentences and the target label is still predicted using the sentence node $g$ in the last layer.

We use the official code from Huggingface [52] and CLUE [48] for reproducing the baseline and our results without external data augmentation, we mainly tune the parameters with training epochs in $\{2, 3, 5, 10\}$, learning rate in $\{2e^{-5}, 3e^{-5}, 5e^{-5}\}$ and batch size in $\{16, 32, 64\}$.

3.3 Results

The main results on English and Chinese language understanding tasks are shown in Table 3. For English tasks, our model gives an average score of 78.67, which is higher than the ELMo (69.65) and
Table 3: Results on GLUE and CLUE benchmark dev sets. Results are reported by matthews correlation (for CoLA) or accuracy (for others).

<table>
<thead>
<tr>
<th>Model (English)</th>
<th>CoLA</th>
<th>SST2</th>
<th>MRPC</th>
<th>QQP</th>
<th>MNLI</th>
<th>QNLI</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELMo [41]</td>
<td>44.1</td>
<td>91.5</td>
<td>70.8</td>
<td>88.0</td>
<td>68.6</td>
<td>71.2</td>
<td>53.4</td>
<td>69.65</td>
</tr>
<tr>
<td>DistilBERT [43]</td>
<td>51.3</td>
<td>91.3</td>
<td>82.7</td>
<td>88.5</td>
<td>82.2</td>
<td>89.2</td>
<td>59.9</td>
<td>77.87</td>
</tr>
<tr>
<td>BERT-base [2]</td>
<td>56.3</td>
<td>91.7</td>
<td>83.5</td>
<td>89.6</td>
<td>84.0</td>
<td>90.9</td>
<td>65.3</td>
<td>80.18</td>
</tr>
<tr>
<td>ALBERT-base [5]</td>
<td>48.2</td>
<td>90.7</td>
<td>87.2</td>
<td>88.2</td>
<td>82.3</td>
<td>90.1</td>
<td>69.7</td>
<td>79.48</td>
</tr>
<tr>
<td>Ours</td>
<td>55.3</td>
<td>90.3</td>
<td>81.0</td>
<td>88.8</td>
<td>81.4</td>
<td>89.6</td>
<td>64.3</td>
<td>78.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model (Chinese)</th>
<th>TNEWS</th>
<th>IFLYTEK</th>
<th>CSL</th>
<th>LCQMC</th>
<th>THUCNews</th>
<th>ChnSentiCorp</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-base [2]</td>
<td>56.14</td>
<td>59.67</td>
<td>81.40</td>
<td>87.89</td>
<td>95.35</td>
<td>92.58</td>
<td>78.83</td>
</tr>
<tr>
<td>DistilBERT-wmm</td>
<td>56.47</td>
<td>59.71</td>
<td>81.23</td>
<td>87.93</td>
<td>95.28</td>
<td>93.00</td>
<td>79.09</td>
</tr>
<tr>
<td>RoBERTa-base-wwm† [44]</td>
<td>57.35</td>
<td>59.90</td>
<td>80.86</td>
<td>88.05</td>
<td>95.43</td>
<td>93.25</td>
<td>79.09</td>
</tr>
<tr>
<td>RoBERTa-wwm-ext† [44]</td>
<td>57.29</td>
<td>59.29</td>
<td>81.16</td>
<td>88.41</td>
<td>95.19</td>
<td>93.33</td>
<td>79.49</td>
</tr>
<tr>
<td>ALBERT-large [5]</td>
<td>55.69</td>
<td>58.36</td>
<td>80.46</td>
<td>88.27</td>
<td>93.52</td>
<td>91.25</td>
<td>77.92</td>
</tr>
<tr>
<td>MacBERT† [46]</td>
<td>57.50</td>
<td>59.36</td>
<td>81.83</td>
<td>89.18</td>
<td>95.74</td>
<td>93.33</td>
<td>79.49</td>
</tr>
<tr>
<td>Ours</td>
<td>57.56</td>
<td>60.10</td>
<td>80.73</td>
<td>86.06</td>
<td>95.17</td>
<td>93.08</td>
<td>78.78</td>
</tr>
</tbody>
</table>

Table 4: Time cost (seconds) during inference for different architectures. Numbers in the parentheses denote the speedup compared to RoBERT-base.

We compared the inference speed of our model with different architectures in Table 4. ELMo gives the lowest results as the sequential nature of RNN structure. For Transformer-based models, DistilBERT shows the minimum time cost because of the lightweight architecture, BART is slower than RoBERTa due to the nonparallel computation in the decoder. All the models take much more time when the sequence becomes much longer. For example, sequences with a length of 512 need about 3 times more computational time than sequences with a length of 64. For our model, adding the recurrent layer and the hidden size will both leads to more inference time. However, by increasing the sequence length, the inference cost grows much slower than the baselines when the sequence length reaches 256 or more, our model can give a 2~3 times speedup than DistilBERT or RoBERTa, even for large model settings.

3.4 Analysis of Efficiency

We compare the inference speed of our model with different architectures in Table 4. ELMo gives the lowest results as the sequential nature of RNN structure. For Transformer-based models, DistilBERT shows the minimum time cost because of the lightweight architecture, BART is slower than RoBERTa due to the nonparallel computation in the decoder. All the models take much more time when the sequence becomes much longer. For example, sequences with a length of 512 need about 3 times more computational time than sequences with a length of 64. For our model, adding the recurrent layer and the hidden size will both leads to more inference time. However, by increasing the sequence length, the inference cost grows much slower than the baselines when the sequence length reaches 256 or more, our model can give a 2~3 times speedup than DistilBERT or RoBERTa, even for large model settings.

We compared our model with Transformer variants for an extra long sequence in Figure 3. Longformer [53] and Reformer [54] give almost linear growth of runtime w.r.t sequence length. Our model
is the fastest when the sequence length is below 8.5k. Linformer and Performer give slightly faster speed when the size reaches 10k. However, the models are particularly designed for long sequences. For example, Linformer project the full self-attention and find the low-rank representation, reducing the complexity from $O(n^2)$ to $O(nk)$, thus the projection dimension $k$ should be pre-defined and less than the sequence length $n$. Similarly, Performer pre-defined kernel feature numbers $m$ and reduce the complexity from $O(n^2)$ to $O(nm)$, the most computational efficiency is achieved only when $n$ is relatively large. Overall, our model can handle both short and long sequences friendly.

4 Conclusion

We investigated a graph recurrent network for large-scale language model pre-training. Our model does not rely on the self-attention mechanism and retaining linear computational complexity with respect to the sequence length. Results show that the inference cost can be largely reduced while without much accuracy loss. For future work, we will study our model for seq2seq-style pre-training as in BART or T5, exploring the applications to generation tasks such as machine translation.

Acknowledgments

We would like to thank reviewers for their valuable insights and helpful advice. The work is funded by the Zhejiang Province Key Project 2022SDXHDX0003. Yue Zhang is the corresponding author.

References


[48] Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Ye Chen Xu, Kai Sun, Dian Yu, Cong Yu, Yin Tian, Qianqian Dong, Weitang Liu, Bo Shi, Yiming Cui, Junyi Li, Jun Zeng, Rongzhao Wang, Weijian Xie, Yanting Li, Yina Patterson, Zuoyu Tian, Yiwen Zhang, He Zhou, Shaowei Liu, Zhe Zhao, Qipeng Zhao, Cong Yue, Xinrui Zhang, Zhengliang Yang, Kyle Richardson, and Zhenzhong Lan. CLUE: A Chinese language understanding evaluation benchmark. In Proc. of COLING, 2020.


