Lightweight and Efficient End-to-End Speech Recognition Using Low-Rank Transformer

ICASSP 2020

Genta Indra Winata  Samuel Cahyawijaya  Zhaojiang Lin  Zihan Liu  Pascale Fung
Outline

- Background
- Preliminaries
- Low-Rank Transformer
- Experiment Setup
- Results and Analysis
- Conclusion
Background
Issues on Speech Recognition

- Speech recognition requires **large memory capacity**
- Large capacity is proportional to **high computational power and time** in training and inference, especially RNNs
- It is **ideal** to have **ASR run on low-end devices**, such as smartphone
Research Questions

- Can smaller models perform better than larger models?
- How to compress model without any performance loss? And speedup training and inference to save the computation cost?
Preliminaries

Low-Rank Matrix Factorization
Model Compression
End-to-End Speech Recognition
**Low-Rank Matrix Factorization**

A large matrix can be decomposed into two smaller matrices, where the rank of the matrices is smaller than the dimension of the original matrix.

\[
\begin{align*}
W_{m \times n} &= U_{m \times r} V_{r \times n}
\end{align*}
\]

**Computation advantages:**

- Produce compact and dense matrices
- Reducing flops from \( m \times n \rightarrow (m + n)r \)
- Compressing the model size \( m \times n \rightarrow (m + n)r \)
Non-negative Matrix Factorization

NMF algorithms aim at finding a rank \( r \) approximation of the form:

\[
W_{m \times n} = UV_{m \times r} V_{r \times n}^T,
\]

minimize \( \| W - UV \|_F^2 \).

where \( W \) and \( U \) are non-negative matrices of dimensions \( m \times r \) and \( r \times n \), respectively.
Preliminaries

Low-Rank Matrix Factorization

Model Compression

End-to-End Speech Recognition
Model Compression

**In-Training**
- Reduce the training time and memory cost
- The model is trained to learn compact representations

**Post-Training**
- Large model training may have bottlenecks in time and speed
- Useful for pre-trained models
- An approximation of the original model
In-Training Factorized LSTM (Kuchaiev and Ginsburg, 2017)

The model accelerates the training of LSTM. Apply matrix factorization by design.

The model improves the speed of training and inference with a small performance loss.

Post-Training Factorized LSTM (Winata, et al. 2019)

A comprehensive comparison of post-training methods on LSTM on language model and downstream NLP tasks. Low-Rank Matrix Factorization generally achieves better than pruning.

Preliminaries

Low-Rank Matrix Factorization
Model Compression

End-to-End Speech Recognition
End-to-End Speech Recognition

There are three main end-to-end sequence-to-sequence ASR architectures:

- RNN-based models with attention (Chan, et al 2016)
- Transformer-based model, a fully-attentional feed-forward architecture (Dong, et al 2018)

RNN with Attention Model (Chan, et al 2016)

The encoder processes the audio input and the decoder generates the transcription.

![Diagram of RNN with Attention Model](image)
**Transformer Model** (Dong, et al 2018)

Remove the recurrence and apply attention to speed up the training and inference.

Joint train with multiple objectives.

Low-Rank Transformer

A lightweight and efficient transformer
Low Rank Transformer (LRT)

- A factorized transformer-based model architecture
- Replacement large high-rank matrices with low-rank matrices to eliminate the computational bottlenecks.

Objective

Predict graphemes given audio inputs
Model Architecture

**Input Encoder:** VGG Encoder

**Components:**
- Low-Rank Multi-Head Attention (LRMHA)
- Low-Rank Feed Forward Network (LRFF)
Linear Encoder-Decoder (LED) Unit

Each $m \times n$ matrix is approximated by the multiplication of the linear encoder unit and a linear decoder unit.

If $r \ll \{m, n\}$:

- **Less parameters** compared to linear layer
- **Better generalization** due to the bottleneck layer
- **Faster training** with less flops
Low Rank Feed Forward (LRFF)

- Two LED units
- Residual connection
- Layer normalization

\[ g(x) = \text{LayerNorm}(\max(0, x E_1 D_1) E_2 D_2 + x), \]
Low Rank Multi Head Attention (LRMHA)

- Utilize LED units
  - Faster Q, K, V projection
  - Attention regularization

\[
\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}} V\right),
\]
\[
hd_i = \text{Attention}(Q E_i^Q D_i^Q, K E_i^K D_i^K, V E_i^V D_i^V),
\]
\[
f(Q, K, V) = \text{Concat}(h_1, \cdots, h_H) E^Q D^O + Q,
\]
Low Rank Multi Head Attention (LRMHA)

- Utilize LED units
  - Faster Q, K, V projection
  - Attention regularization
- Residual connection
  - To avoid gradient issues
- Layer Normalization
Experimental Setup
Datasets

AiShell-1

- A multi-accent Mandarin Chinese speech dataset.
- Consists of 150 hours, 10 hours, and 5 hours of training, validation, and testing, respectively.

HKUST

- A conversational telephone Chinese speech recognition dataset.
- Consists of 152 hours, 4.2 hours, and 5 hours of training, validation, and testing, respectively.
Baseline

- Transformer-based model

- 3 different model size
  - Small
  - Medium
  - Large

- Different model size per dataset due to different embedding size

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model Name</th>
<th># Param</th>
</tr>
</thead>
<tbody>
<tr>
<td>AiShell-1</td>
<td>Transformer (Small)</td>
<td>≈7.8M</td>
</tr>
<tr>
<td></td>
<td>Transformer (Medium)</td>
<td>≈11.5M</td>
</tr>
<tr>
<td></td>
<td>Transformer (Large)</td>
<td>≈22M</td>
</tr>
<tr>
<td>HKUST</td>
<td>Transformer (Small)</td>
<td>≈8.7M</td>
</tr>
<tr>
<td></td>
<td>Transformer (Medium)</td>
<td>≈12.7M</td>
</tr>
<tr>
<td></td>
<td>Transformer (Large)</td>
<td>≈25.1M</td>
</tr>
</tbody>
</table>
Training Phase

We train all characters in the corpus, including <PAD>, <SOS>, and <EOS>. The model consists of 2 encoder layers and 4 decoder layers.

The uncompressed Transformer (Large) has a $\text{dim}_{\text{inner}}$ of 2048, $\text{dim}_{\text{model}}$ of 512, and $\text{dim}_{\text{emb}}$ of 512. We select the same parameters as the LRT model with $r=100$, $r=75$ and $r=50$. 
Inference Phase

We generate the predictions using a beam-search decoding, we take $\alpha = 1$, $\gamma = 0.1$, and a beam size of 8.

$$P(Y) = \alpha P_{trans}(Y|X) + \gamma \sqrt{wc(Y)},$$

We evaluate our model using a single GeForce GTX 1080Ti GPU and three Intel Xeon E5-2620 v4 CPU cores. $wc(Y)$ is the word count to avoid generating very short/long sentences.
Results and Analysis
Results on AiShell-1 Dataset

LRT model outperforms all baseline models with the same number of parameters.

LRT achieves better performance with around 60% compression rate compared to baseline Transformer (Large) model.
Results on HKUST Dataset

**LRT** model outperforms all baseline models with the same number of parameters.

**LRT** achieves better performance with around 60% compression rate compared to baseline **Transformer (Large)** model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>CER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN-hybrid [12]</td>
<td>-</td>
<td>35.9%</td>
</tr>
<tr>
<td>LSTM-hybrid (with perturb.) [12]</td>
<td>-</td>
<td>33.5%</td>
</tr>
<tr>
<td>TDNN-hybrid, lattice-free MMI (with perturb.) [12]</td>
<td>-</td>
<td>28.2%</td>
</tr>
</tbody>
</table>

**Hybrid approach**

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>CER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attention Model [12]</td>
<td>-</td>
<td>37.8%</td>
</tr>
<tr>
<td>CTC + LM [15]</td>
<td>≈12.7M</td>
<td>34.8%</td>
</tr>
<tr>
<td>MTL + joint dec. (one-pass) [12]</td>
<td>≈9.6M</td>
<td>33.9%</td>
</tr>
<tr>
<td>+ RNNLM (joint train) [12]</td>
<td>≈16.1M</td>
<td>32.1%</td>
</tr>
<tr>
<td>Transformer (large)</td>
<td>22M</td>
<td>29.21%</td>
</tr>
<tr>
<td>Transformer (medium)</td>
<td>11.5M</td>
<td>29.73%</td>
</tr>
<tr>
<td>Transformer (small)</td>
<td>7.8M</td>
<td>31.30%</td>
</tr>
<tr>
<td>LRT ($r = 100$)</td>
<td>11.5M</td>
<td><strong>28.95%</strong></td>
</tr>
<tr>
<td>LRT ($r = 75$)</td>
<td>9.7M</td>
<td>29.08%</td>
</tr>
<tr>
<td>LRT ($r = 50$)</td>
<td>7.8M</td>
<td>30.74%</td>
</tr>
</tbody>
</table>

**End-to-end approach**
Memory and Time Efficiency

Our LRT models gain inference time speed-up by up to 1.35x in the GPU and 1.23x in the CPU, compared to the uncompressed Transformer (large) baseline model.

<table>
<thead>
<tr>
<th>dataset</th>
<th>r</th>
<th>ΔCER</th>
<th>compress.</th>
<th>speed-up</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GPU</td>
<td>CPU only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AiShell-1</td>
<td>base</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>23.08</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.40%</td>
<td>49.40%</td>
<td>1.17x</td>
<td>1.15x</td>
<td>23.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>0.26%</td>
<td>57.37%</td>
<td>1.23x</td>
<td>1.16x</td>
<td>23.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>-1.10%</td>
<td>65.34%</td>
<td>1.30x</td>
<td>1.23x</td>
<td>23.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HKUST</td>
<td>base</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>22.43</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.26%</td>
<td>47.72%</td>
<td>1.21x</td>
<td>1.14x</td>
<td>22.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>0.13%</td>
<td>55.90%</td>
<td>1.26x</td>
<td>1.15x</td>
<td>22.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>-1.53%</td>
<td>64.54%</td>
<td>1.35x</td>
<td>1.22x</td>
<td>22.49</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LRT Training Convergence

**LRT model** is more stable to train and convergences faster in just around **15 epochs**.

**LRT model** achieves lower training & validation loss compared to the baseline model with the same number of parameters.
Conclusion
Let’s answer our questions

- Can smaller models perform better than larger models?

Yes, it is! With the better approach, smaller models can not only perform better but also faster than larger models!
Let’s answer our questions

- Can *smaller models* perform *better* than *larger models*?

  Yes, it is! *With the better approach, smaller models can not only performs better but also faster than larger models!*

- How to compress model *without any performance loss*? *And speedup training and inference* to save the computation cost?

  In-training compression, LRT!
Conclusion

• LRT is a memory-efficient with faster-computational neural architecture that eliminate the memory and time bottlenecks.

• LRT can generalize better on test set while also reducing the parameters by 50%.

• LRT is faster to converge compared to normal transformer model.
Thank you

All questions are welcome