SimplePCFG: Simple Hardware-Efficient PCFGs with Independent Left and Right Productions

Wei Liu*, Songlin Yang*, Yoon Kim, Kewei Tu

SIST, ShanghaiTech University
MIT CSAIL

Dec, 2023
Introduction

Simple PCFGs

A Hardware-efficient Inside Algorithm

Results

References

Wei Liu*, Songlin Yang*, Yoon Kim, Kewei Tu

SIST, ShanghaiTech University  MIT CSAIL

SimplePCFG: Simple Hardware-Efficient PCFGs with Independent Left and Right Productions
Introduction

Advantages Brought by Scaling
Language Modeling: The Achilles’ Heel of Low-rank PCFGs

Simple PCFGs

A Hardware-efficient Inside Algorithm

Results

References
1 Introduction

Advantages Brought by Scaling

Language Modeling: The Achilles’ Heel of Low-rank PCFGs

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm

4 Results

5 References
Scaling neural probabilistic context-free grammars (PCFGs) via a low-rank parameterization has demonstrated incredible improvements in unsupervised parsing [1]

- Low-rank parameterization enables a dramatic increase in the numbers of nonterminals (NT) preterminals (PT), from just over 30 and 60 to upwards to 5,000 and 10,000 respectively
- The Sentence-F1 score in unsupervised parsing sees an increase from 55.2 to 64.1, a significant improvement attributable to scaling
Introduction

Advantages Brought by Scaling

Language Modeling: The Achilles’ Heel of Low-rank PCFGs

Simple PCFGs

A Hardware-efficient Inside Algorithm

Results

References
Despite benefiting from scaling in unsupervised parsing, low-rank PCFGs perform poorly as a language model and underperform similarly-sized HMMs.

- On the Penn Treebank, PCFGs scaled via low-rank parameterization with thousands of states achieves $\approx 170$. 
- However, it lags behind a similarly-sized HMM which obtains $\approx 130$ perplexity, even though HMMs are subclass of PCFGs.
1 Introduction

2 Simple PCFGs
   Review Low-rank PCFGs
   Simple PCFGs with Independent Left and Right Productions

3 A Hardware-efficient Inside Algorithm

4 Results

5 References
1 Introduction

2 Simple PCFGs
   Review Low-rank PCFGs
   Simple PCFGs with Independent Left and Right Productions

3 A Hardware-efficient Inside Algorithm

4 Results

5 References
A PCFG can be defined by a 6-tuple \( G = (S, N, P, \Sigma, R, \pi) \), where

- \( S \) : start symbol
- \( N \) : nonterminals
- \( P \) : preterminals
- \( \Sigma \) : terminals
- \( R \) is a set of production rules of the form,
  
  - \( S \rightarrow A \), \( A \in N \)
  - \( A \rightarrow BC \), \( A \in N, B, C \in N \cup P \)
  - \( T \rightarrow w \), \( T \in P, w \in \Sigma \)

and \( \pi : R \rightarrow [0, 1] \) maps rules to their associated probabilities.
The previous approach to scaling HMMs and PCFGs to thousands of nonterminals is parameterizing the rule probability tensor $T \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}| \times |\mathcal{N}|}$ to be low-rank [1, 2, 3].

![Diagram](a)
Low-rank PCFGs can be viewed as introducing a new latent variable, namely a “rank variable” $R$, where $U$, $V$, $W$ are tensor/matrix representations of rule probabilities.

(a) (b) (c)
In fact, a low-rank PCFG can be parameterized as a PCFG with independent left/right productions by marginalizing nonterminal variables and viewing the rank variables as new nonterminal variables [1].

![Diagram of PCFG with independent left/right productions](image)
As such, low-rank PCFGs parameterize $L$, $R$ in a more restrictive manner: $L = VU^T$, $R = WU^T$. We speculate that the shared $U^T$ would restrict the expressiveness of low-rank PCFGs and thus hinder optimization, which motivates our simple PCFGs.
Introduction

Simple PCFGs

Review Low-rank PCFGs

Simple PCFGs with Independent Left and Right Productions

A Hardware-efficient Inside Algorithm

Results

References
In simple PCFGs, we simplify all these things.

- We decompose $\pi_{A \rightarrow BC}$ into $\pi_{B \rightarrow A} \cdot \pi_{A \rightarrow C}$, effectively assuming that left and right children are generated independently.
- In SimplePCFG, we parameterize $L, R$ directly instead of through the shared $U^T$, which in fact contributes to building a more flexible parameterization.
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm
   Inside Algorithm for Simple PCFGs
   FlashInside

4 Results

5 References
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm
   Inside Algorithm for Simple PCFGs
   FlashInside

4 Results

5 References
The recursive formula of the inside algorithm for simple PCFGs:

\[
\beta_{ij}^A = \sum_{B,C \in \mathcal{N}} \pi_{B \cap A} \cdot \pi_{A \cap C} \sum_{i<k<j} \beta_{ik}^B \cdot \beta_{kj}^C
\]

\[
= \sum_{i<k<j} \left( \sum_{B \in \mathcal{N}} \pi_{B \cap A} \cdot \beta_{ik}^B \right) \left( \sum_{C \in \mathcal{N}} \pi_{A \cap C} \cdot \beta_{kj}^C \right)
\]

where \( \beta_{ij}^A \) is the inside probability for span \((A, i, j)\).

The Vector Form: \( \beta_{ij} = \sum_{i<k<j} \eta_{ik} \odot \zeta_{kj} \), \( \eta_{ij} = L \beta_{ij} \), \( \zeta_{ij} = R \beta_{ij} \)
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm
   Inside Algorithm for Simple PCFGs
   FlashInside

4 Results

5 References
To facilitate scaling of simple PCFGs, we introduce *FlashInside*, a hardware-efficient IO-aware implementation of the inside algorithm. It consists of four techniques:

- Span-level Parallelism
- The log-einsum-exp trick
- Kernel Fusion
- Recomputation
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm

4 Results
   Speed Comparison
   Language Modeling
   Unsupervised Parsing

5 References
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm

4 Results
   Speed Comparison
   Language Modeling
   Unsupervised Parsing

5 References
### Results

| Algorithm      | $|\mathcal{N}|$ | $|I|$ | Speed | Memory |
|----------------|--------------|------|-------|--------|
| log-sum-exp    | 512          | 20   | 1x    | 100x   |
| log-einsum-exp | 512          | 20   | 4.8x  | 3x     |
| FlashInside    | 512          | 20   | 9.5x  | 1x     |
| log-einsum-exp | 8192         | 20   | 1x    | 2x     |
| FlashInside    | 8192         | 20   | 6x    | 1x     |
| log-sum-exp    | 512          | 40   | 1x    | 50x    |
| log-einsum-exp | 512          | 40   | 16x   | 3x     |
| FlashInside    | 512          | 40   | 44x   | 1x     |
| log-einsum-exp | 8192         | 40   | 1x    | 2.4x   |
| FlashInside    | 8192         | 40   | 39x   | 1x     |

The log-einsum-exp technique significantly enhances computational efficiency, and FlashInside further improves its performance superiority.
1 Introduction

2 Simple PCFGs

3 A Hardware-efficient Inside Algorithm

4 Results
   Speed Comparison
   Language Modeling
   Unsupervised Parsing

5 References
### Table 1: Results on the PTB language modeling split from [4]. **NT** denotes the number of nonterminals and **ppl** denotes perplexity. Top results are from previous papers [3, 1], while the bottom results are from the current work. Our runs are averaged over 4 seeds. SN-PCFG is our model, simple neural PCFG.

<table>
<thead>
<tr>
<th>Model</th>
<th>NT</th>
<th>ppl (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHMM</td>
<td>4096</td>
<td>147</td>
</tr>
<tr>
<td>LHMM</td>
<td>16384</td>
<td>131.8</td>
</tr>
<tr>
<td>Rank HMM</td>
<td>16384</td>
<td>127.0</td>
</tr>
<tr>
<td>Rank HMM</td>
<td>32768</td>
<td>126.4</td>
</tr>
<tr>
<td>Rank PCFG†</td>
<td>4096</td>
<td>174.5±11.1</td>
</tr>
<tr>
<td>Rank PCFG†</td>
<td>8192</td>
<td>161.2±8.9</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>4096</td>
<td>125.4±4.1</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>8192</td>
<td><strong>119.0±5.3</strong></td>
</tr>
</tbody>
</table>
Introduction

Simple PCFGs

A Hardware-efficient Inside Algorithm

Results

- Speed Comparison
- Language Modeling
- Unsupervised Parsing

References
<table>
<thead>
<tr>
<th>Model</th>
<th>NT</th>
<th>S-F1 (↑)</th>
<th>ppl (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-PCFG</td>
<td>30</td>
<td>50.8</td>
<td>252.6</td>
</tr>
<tr>
<td>C-PCFG</td>
<td>30</td>
<td>55.2</td>
<td>-</td>
</tr>
<tr>
<td>TN-PCFG</td>
<td>500</td>
<td>57.7</td>
<td>210.0</td>
</tr>
<tr>
<td>Rank PCFG</td>
<td>4500</td>
<td>64.1</td>
<td>168.0</td>
</tr>
<tr>
<td>Rank PCFG†</td>
<td>4096</td>
<td>60.1±7.6</td>
<td>165.1±7.7</td>
</tr>
<tr>
<td>Rank PCFG†</td>
<td>8192</td>
<td>61.1±5.9</td>
<td>171.2±11.7</td>
</tr>
<tr>
<td>N-PCFG†</td>
<td>128</td>
<td>56.7±3.7</td>
<td>181.1±15.3</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>128</td>
<td>51.1±4.1</td>
<td>231.7±8.1</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>4096</td>
<td>65.1±2.1</td>
<td>132.5±4.9</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>8192</td>
<td>62.9±2.8</td>
<td>134.6±9.1</td>
</tr>
<tr>
<td>SC-PCFG</td>
<td>512</td>
<td>54.3±4.8</td>
<td>-</td>
</tr>
<tr>
<td>SC-PCFG</td>
<td>2048</td>
<td>60.6±3.6</td>
<td>-</td>
</tr>
<tr>
<td>PRPN</td>
<td>-</td>
<td>37.4</td>
<td>-</td>
</tr>
<tr>
<td>ON</td>
<td>-</td>
<td>47.7</td>
<td>-</td>
</tr>
<tr>
<td>DIORA+span constraint</td>
<td>-</td>
<td>61.2</td>
<td>-</td>
</tr>
<tr>
<td>S-DIORA</td>
<td>-</td>
<td>57.6</td>
<td>-</td>
</tr>
<tr>
<td>Constituency test</td>
<td>-</td>
<td>62.8</td>
<td>-</td>
</tr>
<tr>
<td>StructFormer</td>
<td>-</td>
<td>54.0</td>
<td>-</td>
</tr>
<tr>
<td>Fast-R2D2</td>
<td>-</td>
<td>57.2</td>
<td>-</td>
</tr>
<tr>
<td>Right-Branching</td>
<td>-</td>
<td>39.5</td>
<td>-</td>
</tr>
<tr>
<td>Oracle Trees</td>
<td>-</td>
<td>84.3</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table 2:** Unsupervised parsing performance on the PTB test set, including comparison against prior work (bottom). SC-PCFG is the compound version of simple neural PCFG.
<table>
<thead>
<tr>
<th>Model</th>
<th>NT</th>
<th>Chinese</th>
<th></th>
<th>French</th>
<th></th>
<th>German</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>S-F1↑</td>
<td>ppl↓</td>
<td>S-F1↑</td>
<td>ppl↓</td>
<td>S-F1↑</td>
<td>ppl↓</td>
</tr>
<tr>
<td>Left-Branching</td>
<td>-</td>
<td>7.2</td>
<td></td>
<td>5.7</td>
<td></td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>Right-Branching</td>
<td>-</td>
<td>25.5</td>
<td></td>
<td>26.4</td>
<td></td>
<td>14.07</td>
<td></td>
</tr>
<tr>
<td>Random Trees</td>
<td>-</td>
<td>15.2</td>
<td></td>
<td>16.2</td>
<td></td>
<td>13.9</td>
<td></td>
</tr>
<tr>
<td>kim-2022-revisiting</td>
<td>-</td>
<td>-</td>
<td></td>
<td>41.9</td>
<td></td>
<td>47.3</td>
<td></td>
</tr>
<tr>
<td>li-lu-2023-contextual</td>
<td>-</td>
<td>-</td>
<td></td>
<td>48.7</td>
<td></td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td>N-PCFG</td>
<td>30</td>
<td>26.3±2.5</td>
<td>45.0±2.0</td>
<td>42.3±1.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-PCFG</td>
<td>30</td>
<td>38.7±6.6</td>
<td>-</td>
<td>45.0±1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TN-PCFG</td>
<td>250</td>
<td>39.2±5.0</td>
<td>39.1±4.1</td>
<td>47.1±1.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank PCFG</td>
<td>4096</td>
<td>31.00±8.9</td>
<td>409.4±29.5</td>
<td>31.2±9.3</td>
<td>355.8±13.7</td>
<td>35.6±9.1</td>
<td>215.3±57.1</td>
</tr>
<tr>
<td>Rank PCFG</td>
<td>8192</td>
<td>32.4±8.2</td>
<td>372.6±31.4</td>
<td>32.9±10.6</td>
<td>332.2±60.8</td>
<td>38.9±9.6</td>
<td>190.5±65.9</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>4096</td>
<td>39.9±6.3</td>
<td>328.3±62.1</td>
<td>38.0±3.1</td>
<td>379.7±5.2</td>
<td>46.7±4.9</td>
<td>157.8±65.6</td>
</tr>
<tr>
<td>SN-PCFG</td>
<td>8192</td>
<td>41.2±3.5</td>
<td>288.2±11.7</td>
<td>43.3±9.9</td>
<td>259.9±70.2</td>
<td>46.9±5.1</td>
<td>159.5±77.2</td>
</tr>
<tr>
<td>SC-PCFG</td>
<td>512</td>
<td>38.4±7.4</td>
<td>-</td>
<td>47.9±1.2</td>
<td>-</td>
<td>47.7±1.0</td>
<td>-</td>
</tr>
<tr>
<td>SC-PCFG</td>
<td>2048</td>
<td>42.9±2.9</td>
<td>-</td>
<td>49.9±1.7</td>
<td>-</td>
<td>49.1±1.0</td>
<td>-</td>
</tr>
</tbody>
</table>

Wei Liu*, Songlin Yang*, Yoon Kim, Kewei Tu

SIST, ShanghaiTech University  MIT CSAIL

SimplePCFG: Simple Hardware-Efficient PCFGs with Independent Left and Right Productions
Thank You
1. Introduction

2. Simple PCFGs

3. A Hardware-efficient Inside Algorithm

4. Results

5. References


