Device Placement Optimization with Reinforcement Learning

Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean
What is device placement

- Consider a TensorFlow computational graph $G$, which consists of $M$ operations $\{o_1, o_2, \ldots, o_M\}$, and a list of $D$ available devices.

- A placement $P = \{p_1, p_2, \ldots, p_M\}$ is an assignment of an operation $o_i$ to a device $p_i$. 
Why device placement

- Trend toward many-device training, bigger models, larger batch sizes
- Growth in size and computational requirements of training and inference
Typical approaches

- Use a heterogeneous distributed environment with a mixture of many CPUs and GPUs
- Often based on greedy heuristics
- Require deep understanding of devices: bandwidth, latency behavior
- Are not flexible enough and does not generalize well
ML for device placement

- ML is repeatedly replacing rule based heuristics

- RL can be applied to device placement
  - Effective search across large state and action spaces to find optimal solution
  - Automatic learning from underlying environment only based on reward function
RL based device placement

Input

Neural Model

Available Devs

CPU

GPU

RL model

Policy

Output

Assignment of ops in Neural model to devices

Evaluate runtime
Problem formulation

\[ J(\theta) = \mathbb{E}_{P \sim \pi(P|G; \theta)} \left[ R(P) \mid G \right] \]

- \( J(\theta) \) : expected runtime
- \( \theta \) : trainable parameters of policy
- \( R \) : runtime
- \( \pi(P|G; \theta) \) : policy
- \( P \) : output placements
Training with REINFORCE

- Learn the network parameters using Adam optimizer based on policy gradients computed via the REINFORCE equation:

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G};\theta)} \left[ R(\mathcal{P}) \cdot \nabla_\theta \log p(\mathcal{P}|\mathcal{G};\theta) \right] \]

- Use K placement samples to estimate policy gradients & use a baseline term B to reduce variance:

\[ \nabla_\theta J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} (R(\mathcal{P}_i) - B) \cdot \nabla_\theta \log p(\mathcal{P}_i|\mathcal{G};\theta) \]
Model architecture
Challenges

- Vanishing
- Exploding gradient issue
- Large memory footprints

<table>
<thead>
<tr>
<th>Model</th>
<th>#operations</th>
<th>#groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNNLM</td>
<td>8943</td>
<td>188</td>
</tr>
<tr>
<td>NMT</td>
<td>22097</td>
<td>280</td>
</tr>
<tr>
<td>Inception-V3</td>
<td>31180</td>
<td>83</td>
</tr>
</tbody>
</table>
Distributed training
Experiments

- Recurrent Neural Language Model (RNNLM)
- Neural Machine Translation with attention mechanism (NMT)
- Inception-V3
Learned placement on NMT
NMT end-to-end runtime

Neural MT Training Curves with 1CPU, 4GPUs

- RL-based Placement
- Human Expert (One layer per device)
Learned placement on Inception-V3
Inception-V3 end-to-end runtime

Inception Training Curves with 1CPU, 4GPUs

- RL-based Placement
- Synchronous Towers
- Asynchronous Towers

cumulative avg train loss vs. hours
Profiling on NMT

RL-based placement

Expert-designed placement

- **encoder_lstm(\text{grad})**
- **attention(\text{grad})**
- **decoder_lstm(\text{grad})**
- **softmax(\text{grad})**
Profiling on Inception-V3

![Graphs showing operation runtime (s) for RL-based placement and Synchronous towers across GPUs with different operations like conv2d(gradient), avgpool(gradient), concat(gradient), and dropout(gradient).]
Profling on Inception-V3

RL-based placement

Synchronous towers

operation runtime (s)

<table>
<thead>
<tr>
<th>GPU</th>
<th>RL-based placement</th>
<th>Synchronous towers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU0</td>
<td>0.2</td>
<td>GPU0</td>
</tr>
<tr>
<td>GPU1</td>
<td>0.6</td>
<td>GPU1</td>
</tr>
<tr>
<td>GPU2</td>
<td>0.8</td>
<td>GPU2</td>
</tr>
<tr>
<td>GPU3</td>
<td>1.0</td>
<td>GPU3</td>
</tr>
</tbody>
</table>

memcpy
## Running times (in seconds)

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Single-CPU</th>
<th>Single-GPU</th>
<th>#GPUs</th>
<th>Scotch</th>
<th>MinCut</th>
<th>Expert</th>
<th>RL-based</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNNLM (batch 64)</td>
<td>6.89</td>
<td>1.57</td>
<td>2</td>
<td>13.43</td>
<td>11.94</td>
<td>3.81</td>
<td>1.57</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td>11.52</td>
<td>10.44</td>
<td>4.46</td>
<td>1.57</td>
<td>0.0%</td>
</tr>
<tr>
<td>NMT (batch 64)</td>
<td>10.72</td>
<td>OOM</td>
<td>2</td>
<td>14.19</td>
<td>11.54</td>
<td>4.99</td>
<td>4.04</td>
<td>23.5%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td>11.23</td>
<td>11.78</td>
<td>4.73</td>
<td>3.92</td>
<td>20.6%</td>
</tr>
<tr>
<td>Inception-V3 (batch 32)</td>
<td>26.21</td>
<td>4.60</td>
<td>2</td>
<td>25.24</td>
<td>22.88</td>
<td>11.22</td>
<td>4.60</td>
<td>0.0%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td></td>
<td>23.41</td>
<td>24.52</td>
<td>10.65</td>
<td>3.85</td>
<td>19.0%</td>
</tr>
</tbody>
</table>
Summary

- Propose a RL model to optimize device placements for neural networks
- Use policy gradient to learn parameters
- Policy finds non-trivial assignment of operations to devices that outperform heuristic approaches
- Profiling of results show policy learns implicit trade-offs between computation and communication in hardware