Device Placement Optimization with Reinforcement Learning
A Hierarchical Model for Device Placement

A. Mirhoseini, Hieu Pham, A. Goldie et al

November 2019
Problem Background

- Tensorflow allows user to place operators on different devices to take advantage of parallelism and heterogeneity
- Current solution: human experts use heuristics to place the operators as best they can
- Some simple graph-based automated approaches (e.g. Scotch) perform worse
Approach

- Use reinforcement learning and neural nets to find the best placement
RNNs model dependencies between data; they have persistence.

E.g. previous words or previous placements of operators.
Background: LSTM and the Vanishing Gradient Problem

- Too many multiplications means gradient quickly diminishes to 0
- Gated structure can model long term dependencies better
- Forget, input and output gates control a hidden state
Traditional use of NNs is in a supervised setting with labelled training data.

Need to learn from the environment.

Want to maximise the expected reward:
\[ J(\theta) = \sum_{\tau} P(\tau; \theta)R(\tau) \]

The derivative, \( \nabla_{\theta} J(\theta) \) is equivalent to
\[ \sum_{\tau} P(\tau; \theta)\nabla_{\theta} \log(P(\tau; \theta)R(\tau)) \]

This is actually an expected value, so can use monte-carlo sampling to approximate:
\[ \nabla_{\theta} J(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} R(x_i)\nabla_{\theta} \log(P(x_i|\theta)) \]
Implementation: Neural network architecture

- Sequence-to-sequence model; this is two RNNs that communicate via shared state
- Input: sequence of vectors representing the type of each operation, output sizes, encoding of links with other operators
- Output: placements for operations
Implementation: RL

- Uses monte-carlo sampling as discussed
- Reward function is the square-root of running time
- High fixed cost for OOM on e.g. single GPUs
- Subtract a moving average from reward to decrease variance
Grouping

- Dataflow graph huge: big search space and vanishing gradient
- Solution one: Co-locate operators manually into groups that should be executed on the same device
- Solution two: Add another (feed-forward) neural network, the grouper
- Hierarchical approach: grouper and placer
Evaluation: Experimental setup

- Measure time for single step of several different models: RNNLM, NMT, Inception-V3, ResNet
- Run on a single machine, using CPU and 2 - 8 GPUs
- Baselines are single CPU, single GPU, using the Scotch library, expert placement
Evaluation: Results

<table>
<thead>
<tr>
<th>Tasks</th>
<th>CPU Only</th>
<th>GPU Only</th>
<th>#GPUs</th>
<th>Human Expert</th>
<th>Scotch</th>
<th>MinCut</th>
<th>Hierarchical Planner</th>
<th>Runtime Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-V3</td>
<td>0.61</td>
<td>0.15</td>
<td>2</td>
<td>0.15</td>
<td>0.93</td>
<td>0.82</td>
<td>0.13</td>
<td>16.3%</td>
</tr>
<tr>
<td>ResNet</td>
<td>-</td>
<td>1.18</td>
<td>2</td>
<td>1.18</td>
<td>6.27</td>
<td>2.92</td>
<td>1.18</td>
<td>0%</td>
</tr>
<tr>
<td>RNNLM</td>
<td>6.89</td>
<td>1.57</td>
<td>2</td>
<td>1.57</td>
<td>5.62</td>
<td>5.21</td>
<td>1.57</td>
<td>0%</td>
</tr>
<tr>
<td>NMT (2-layer)</td>
<td>6.46</td>
<td>OOM</td>
<td>2</td>
<td>2.13</td>
<td>3.21</td>
<td>5.34</td>
<td>0.84</td>
<td>60.6%</td>
</tr>
<tr>
<td>NMT (4-layer)</td>
<td>10.68</td>
<td>OOM</td>
<td>4</td>
<td>3.64</td>
<td>11.18</td>
<td>11.63</td>
<td>1.69</td>
<td>53.7%</td>
</tr>
<tr>
<td>NMT (8-layer)</td>
<td>11.52</td>
<td>OOM</td>
<td>8</td>
<td><strong>3.88</strong></td>
<td>17.85</td>
<td>19.01</td>
<td>4.07</td>
<td>-4.9%</td>
</tr>
</tbody>
</table>

- Only 3 hours for hierarchical model
- Performance significantly better than the manually co-located version
Evaluation: Understanding the results

- Classic tradeoff: distributing more for more parallelism, want to minimise copying costs
- Different architectures have different amounts of parallelism available to exploit
Strengths

- Hierarchical planner completely end-to-end
- Overhead of three hours is small (original paper 13-27 hours)
- Capable of finding complex placements which are beyond a human
- Sometimes very substantial improvements
Weaknesses

- First paper not reproducible: don’t mention the version of Tensorflow, even original authors couldn’t reproduce results
- Results mixed; often no improvement if best placement is trivial. Can this be determined by looking at the amount of parallelism in the graph?
- Will it scale? NMT 8-layer has a decrease in performance compared to human expert. Why this sudden decline?
- How many times did they run the random RL process?
- Incorporate humans to improve placements even further
Questions