Device Placement Optimization using Reinforcement Learning

By Mirhoseini et al.

Shyam Tailor

21/11/18

The Problem

- Neural Networks are getting bigger and require greater resources for training and inference.
- Want to schedule in a *heterogeneous distributed environment*.
  - CPUs and GPUs in the paper.
  - All benchmarks run on a single machine.
- Traditionally: **use heuristics**
  - Previous automated approaches e.g. Scotch [3] do not work too well.

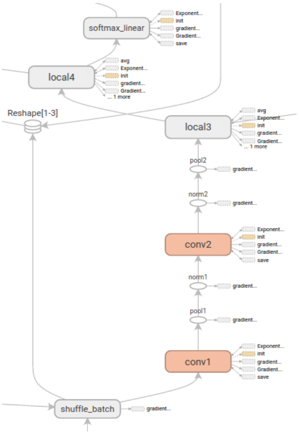
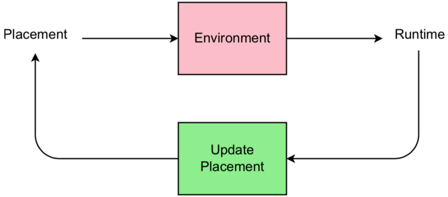


Figure from TensorFlow website.

This Paper's Approach



- Use Reinforcement Learning to create the placements.
- Run placements in the real environment and measure their execution time as a reward signal.
- Use the evaluated reward signals to improve placement policy.

Revision: Policy Gradients

- We have *parameterised* policies π_θ, where θ is the parameter
- We want to pick a policy π^* that maximises our reward $R(\tau)$.

- With policy gradients, we have an objective $J(\theta)$.

$$J(\theta) = E_{\tau \sim \pi_\theta(\cdot)}[R(\tau)]$$

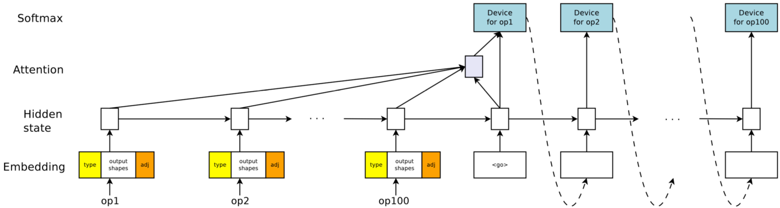
- Use gradient descent to optimise $J(\theta)$ to find π^*.
  - Details out of scope but can be done using Monte Carlo Sampling.

The Reward Signal

$R(\mathcal{P}) =$ *Square root* of total time for forward pass, backward pass, and parameter update.

- Sometimes placements just don't run — have a large constant representing a failed placement.
- Square root to make training more robust.
- **Variance reduction**: take ten runs and discard the first.

The Policy

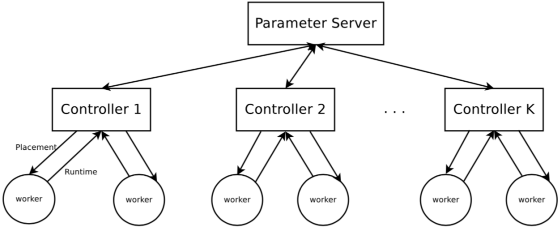


- Use an attentional sequence-to-sequence model which knows about devices that can be used for placements.
- **Input**: sequence of operations in the computation graph.
- **Output**: sequence of placements for the input operations.

Cutting Down the Search Space

- **Problem**: the computation graph can be very big.
- **Solution**: try to fuse portions of the graph as a pre-processing step where possible.
- Co-locate operations when it makes sense to.
  - e.g. if an operation's output only goes to one other operation, keep them together.
  - Can be architecture specific too e.g. keeping LSTM cells together or keeping convolution / pool layers together.
- On evaluated networks, fused graph is around **1%** the size of the original.

Training Setup



- To avoid bottleneck, distribute parameters to controllers.
- Controllers take samples, and instruct workers to run them.

Evaluation: Architectures and Machines

- Experiments involved 3 popular network architectures:
  1. Recurrent Neural Network Language Model [5, 2].
  2. Neural Machine Translation with Attention Mechanism [1].
  3. Inception-V3 [4].

- *Single* machine used to run experiments.
  - Either 2 or 4 GPUs per machine for experiment purposes.

Evaluation: Baselines for Comparison

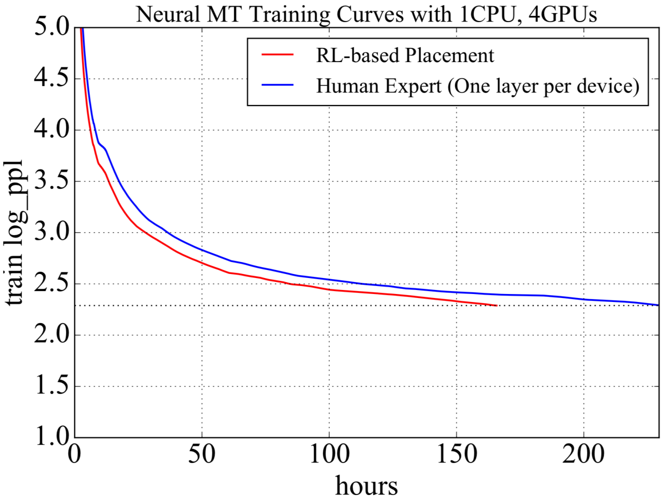
1. Run entire network on the CPU.
2. Run entire network on a *single* GPU.
3. Use Scotch to create a placement over the CPU and GPU.
   - Also run experiment without allowing the CPU.
4. Expert-designed placements from the literature.

Evaluation: How Fast are the RL Placements?

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

- Took between 12-27 hours to find placements.

Evaluation: How Fast are the RL Placements? continued



Analysis: Why are the Placements Chosen Faster?

- The RL placements generally do a better job of *distributing computation load and minimising copying costs*.
- **This is tricky** — and it's different for different architectures!
  - Inception — it's hard to exploit model parallelism due to dependencies restricting parallelism so try to *minimise copying*
  - NMT — the opposite applies, so balance computation load.

Authors' Conclusions

- It looks like RL can optimise around the tradeoff between computation and copying.
- The policy is learnt with nothing except the computation graph and the number of available devices.

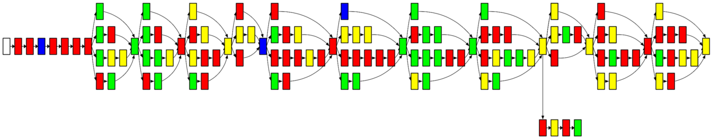
Opinion: Positives

- This method shows promise, as it learns simple baselines automatically, and can exceed human performance where more advanced setup is required.
    - At least on the networks they tested it on.
- The technique was applied to different architectures, and positive results were obtained for each one.
- The technique should be generalisable to other system optimisation problems, in principle.

Opinion: Flaws in Evaluation

- Policy gradients are *stochastic* — so why haven't multiple runs been reported?
- Is there a large variance between solutions found?
- Does the algorithm sometimes fail to converge to anything useful?

Opinion: Improvement — Post-Processing



- Is there low hanging fruit missed by the RL optimisation?
- The authors never attempt to interpret the placements beyond superficial comments about computation and copying.

Opinion: Improvement — Transfer Learning

- Each time the algorithm is run, it is learning about balancing copying and computation *from scratch*.
- These concepts are not inherently unique to each network though — the precise tradeoffs may change, but the general concepts remain.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: (Sept. 1, 2014). URL: https://arxiv.org/abs/1409.0473 (visited on 11/20/2018).

Rafal Jozefowicz et al. "Exploring the Limits of Language Modeling". In: *arXiv:1602.02410 [cs]* (Feb. 7, 2016). arXiv: 1602.02410. URL: http://arxiv.org/abs/1602.02410 (visited on 11/20/2018).

François Pellegrini. "A Parallelisable Multi-level Banded Diffusion Scheme for Computing Balanced Partitions with Smooth Boundaries". In: *Euro-Par 2007 Parallel Processing*. Ed. by Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 195–204. ISBN: 978-3-540-74466-5.

Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: (Dec. 2, 2015). URL: https://arxiv.org/abs/1512.00567 (visited on 11/20/2018).

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent Neural Network Regularization". In: (Sept. 8, 2014). URL: https://arxiv.org/abs/1409.2329 (visited on 11/20/2018).