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

and

A Hierarchal Model for Device Placement

Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean

Large Scale Data Processing and Optimization

Background

- Large-scale neural networks needs training across a distributed environment of heterogenous devices
- Given TensorFlow graph, determine which device needs to execute which operations for best speed
  - ➢ This needs to include both speed of operation and linkages between devices and adjacent operations on the graph
- Currently done by human experts who manually determine which part of graph to schedule where
- Current automated systems like Scotch use optimal graph partitioning algorithms with a cost function
- Authors approach: assess placements using reinforcement learning by using runtime vs. baseline as the reward signal

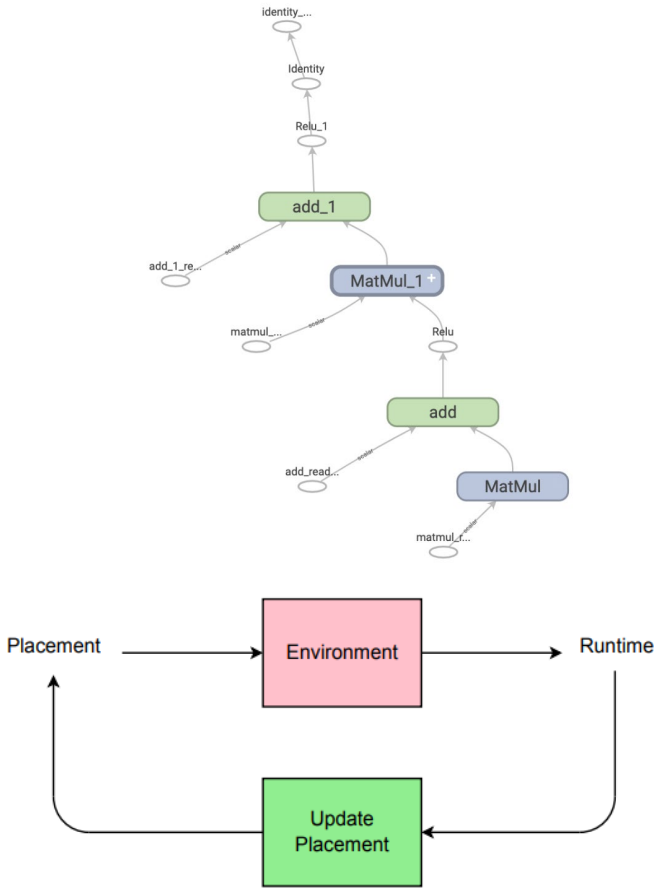


Figure 1. An overview of the RL based device placement model.

Approach

- Given a TensorFlow graph with M operations and D devices, compute a placement P (an M-tuple), consisting of an assignment of operation o_i to device d_j, where the runtime $r(P)$ is minimized
- Authors minimize: $J(\theta) = \mathbb{E}_{P\sim\pi(P|G;\theta)}[R(p)|G]$, usuing a sequence-to-sequence model (fig. 2) to define $\pi(P|G;\theta)$ and sample from it and use REINFORCE: $\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}(R(P_i) - B).\nabla_\theta p(P_i|G;\theta)$ to train
- Model input is the set of operations, defined as a 3-tuple with operation type, output shape, and a one-hot encoded vector representing other operations depending on current operation
- Model consists of an encoder RNN and a decoder RNN, which is an attentional LSTM

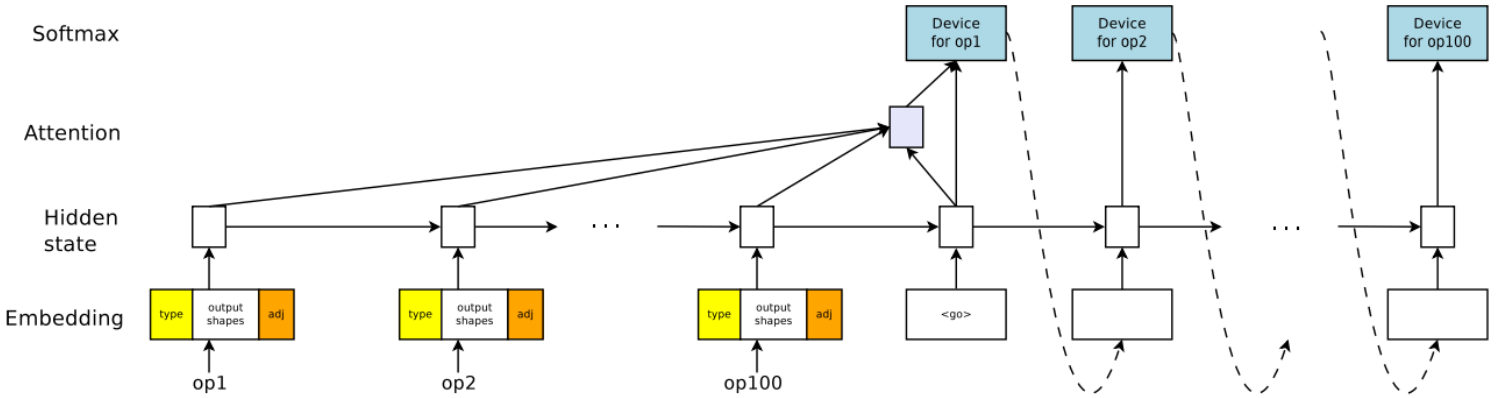


Figure 2. Device placement model architecture.

Implementation Details

- $R(P) = \sqrt{r(P)}$ is used as reward signal to dampen large signals
- Some placements will fail (e.g. exceeding device's memory limitations), here runtime is set to a high value or ignored at the end
- As each operation is input to RNN, and TensorFlow graphs contain very large number of operations, co-located groups of operations that can be execute on a single device need to be manually created., including:
  - TensorFlow's default colocation groups (e.g. operation outputs with gradients)
  - Operations that only depend on one parent
  - Network specific groupings – e.g. convolution/pooling layers in CNNs and LSTM cells
- Model resides on parameter server and dispatches placements to controllers, which execute on workers and calculate runtime
  - Controller gathers gradient update and sends to server
  - Average of 10 steps except first is used

Model	#operations	#groups
RNNLM	8943	188
NMT	22097	280
Inception-V3	31180	83

Table 1. Model statistics.

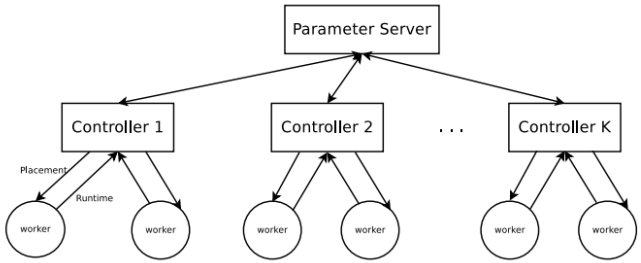
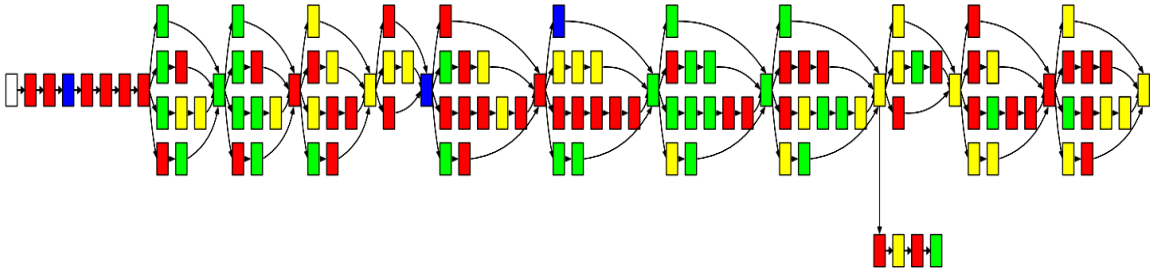


Figure 3. Distributed and asynchronous parameter update and reward evaluation.

Experiments (I)

- Benchmarked on 2 large RNNLM language/machine translation models, and Inception-V3
- Comparison vs. single CPU/GPU, Scotch, MinCut (i.e. Scotch but configured to exclude CPU unless necessary), and expert designed placements
- Results show RL-based approach matches or beats experts in all benchmarks
- Current automated approaches heavily underperform both expert-designed and RL based approach



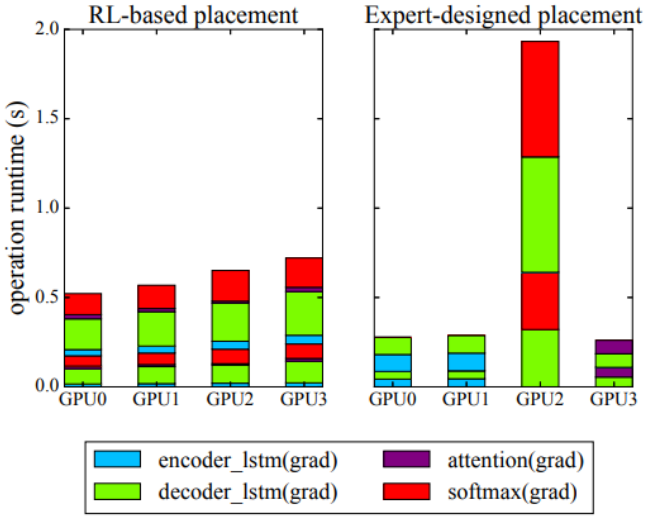


Figure 5. RL-based placement of Inception-V3. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. RL-based placement achieves the improvement of 19.7% in running time compared to expert-designed placement.

Experiments (II)

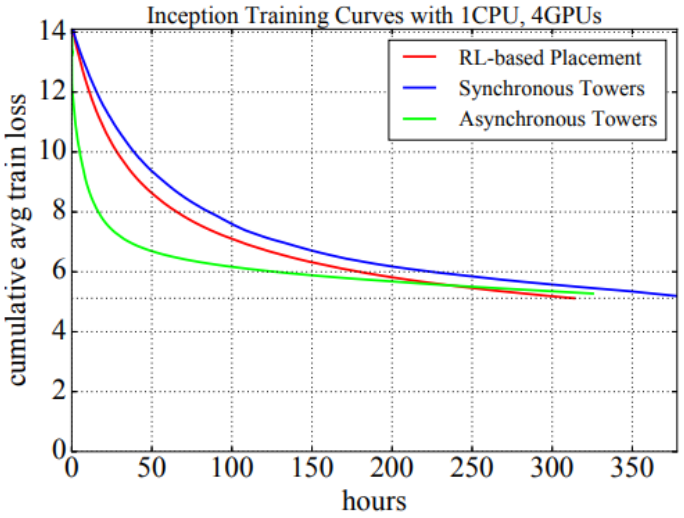
- RNNLM: RL-based approach discovers it's possible to fit model onto one GPU, so does that and beats expert which used 2 GPUs
- Neural MT: RL learns to place less computationally intensive operations e.g. embeddings lookup on CPUs to reduce overhead for GPUs resulting in 20% speedup
- Inception V3: With 2 GPUs, RL approach notices that communication costs > parallelism gains and uses just one CPU, whereas it finds a better placement with 4 GPUs

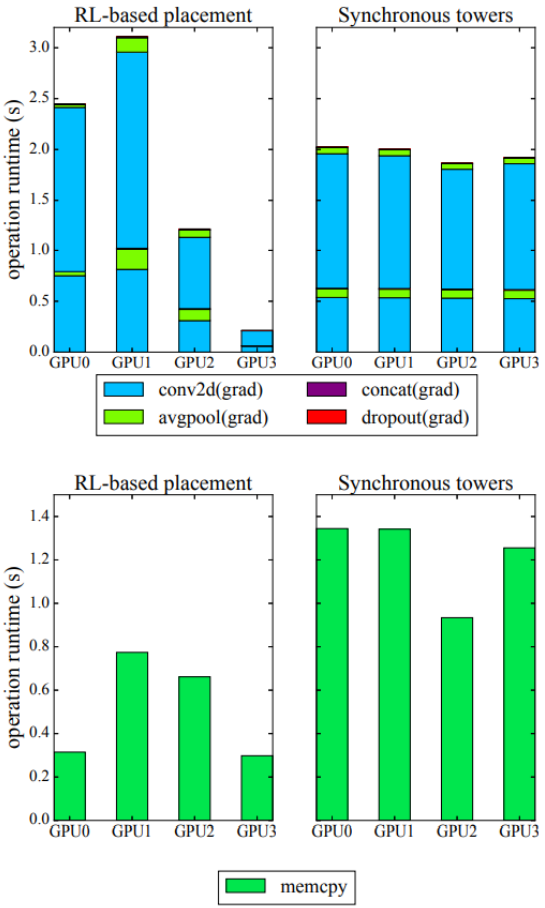
Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**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

Table 2. Running times (in seconds) of placements found by RL-based method and the baselines (lower is better). For each model, the first row shows the results with 1 CPU and 2 GPUs; the second row shows the results with 1 CPU and 4 GPUs. Last column shows improvements in running time achieved by RL-based placement over fastest baseline. To reduce variance, running times less than 10 seconds are measured 15 times and the averages are recorded. OOM is Out Of Memory.

Comparison vs. Data-Parallelism

- Inception V3 with data-parallelism with “synchronous and asynchronous towers” approaches (e.g. data-parallelism with asynchronous/synchronous gradient updates from batches) was compared vs. model parallelism from RL-based approach
- Results showed faster initial training time for asynchronous towers, but faster convergence for RL-based approach





Key Drawbacks

Limitation	Discussion
The process takes a long time to train, ~12-27 hours for models studied	Limitation on how much additional workers can help without more hardware. Approach designed for static hardware graph.
Optimal placement is hardware-dependent – approach cannot be scaled to the cloud	Inherent limitation of approach. However, this could potentially be overcome by including device features in training. Key solution by ML is non-stationary $R(P)$.
Timely training requires many more GPUs than model is trained to use	Author works at Google, framework likely aimed at creating specialized execution graphs for particular applications
Approach requires manual determination of groups, which can be architecture-specific	**?**
TensorFlow graphs can be huge, since operations are each an element of the input sequence, it limits how large a TF graph can go through this process	**?**

Solution: Hierarchal Model for Automated Groupings

- Add a “grouper” model that calculates operation groupings automatically, and feed groupings to same model as previous paper, and train both models in tandem
- Projection of operations into a lower dimensional space – compresses TF graph and optimizes execution at the same time

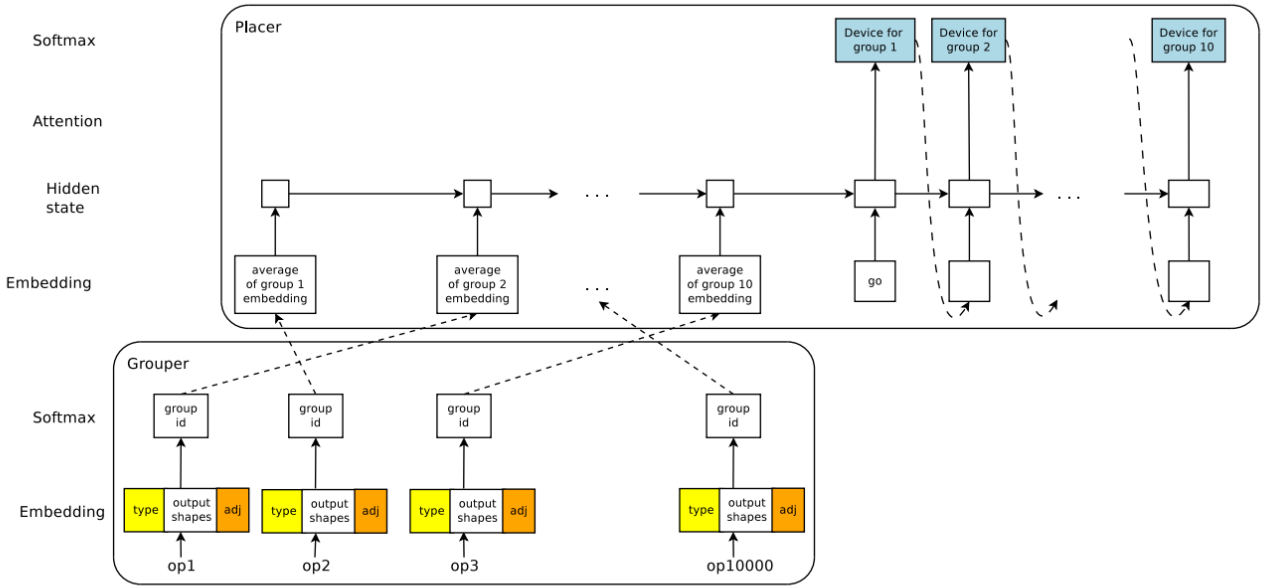


Figure 1: Hierarchical model for device placement (see text for more details).

Hierarchal Model Implementation and Benchmarks

- Previous approach limited to graphs with <1000 groupings
- Input to grouper is similar to previous work: operation type, output shape, and adjacency information
- Input to placer is now 1) vector with counts of operation types in group, 2) total amount of output shapes in group, 3) adjacency information
- Given most placements infeasible, only feasible placements considered in first 500 steps
- Benchmarks computed using Inception-V3, NMT and RNNLM as before, but also larger versions of NMT and ResNet
- Same baseline comparisons as before

Hierarchal Model Results

- Hierarchal planner shows as good or substantially better results than human experts in most of the benchmarks
- Results show non-trivial and human-impossible placements to find, including different parts of unrolled LSTM placed across different devices, which previous approach would not be able to do at all

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

Table 1: Model Runtimes (seconds) for different placements (lower is better). OOM: Out Of Memory.

Pros & Cons

- Pros:
  - Completely automated placement of compute graph across devices
  - Achieves faster-than-state of the art training results across benchmarks, both faster than other automated systems and humans
  - Can be used across arbitrary hardware at training time, no prior on hardware needed
- Cons
  - Training is slow and potentially requires much more hardware than the underlying TF graph is actually being run on
  - Optimal placement is hardware-depending, and does not scale to cloud compute
  - Discussion of why Scotch underperforms is limited
  - Approach still uses feed-forward network to determine placings, potentially limiting the size of the number of operations
  - Needs sensitivity analysis of placements to see how robust placement is

Citations

Mirhoseini, A., Goldie, A., Pham, H., Steiner, B., Le, Q.V., Dean, J., 2018. A Hierarchical Model for Device Placement, in: International Conference on Learning Representations.

Mirhoseini, A., Pham, H., Le, Q.V., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., Dean, J., 2017. Device Placement Optimization with Reinforcement Learning. CoRR abs/1706.04972.

Pellegrini, F., Roman, J., 1996. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs, in: Liddell, H., Colbrook, A., Hertzberger, B., Sloot, P. (Eds.), High-Performance Computing and Networking. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 493–498.