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 (Google Brain)

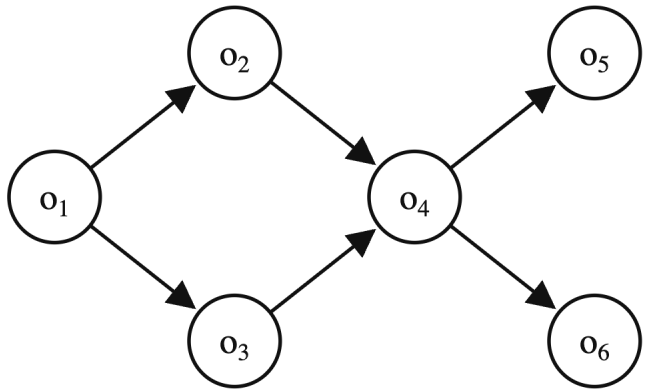
ICML 2017

Moritz Ladenburger

The problem: device placement

Models too large for one device, machines with a mix of CPUs and GPUs: which operation should run where?

Input: computational graph G

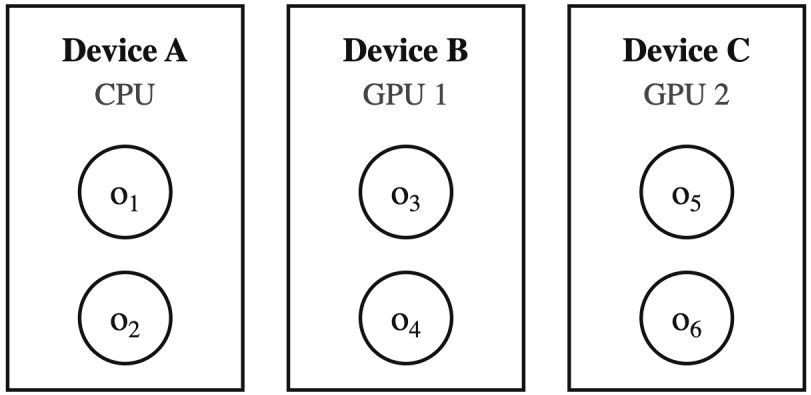


M operations $o_1, \ldots, o_M$; edges = tensors

placement

$P = \{p_1, \ldots, p_M\}$

Output: device for each operation



$p_i \in \{1, \ldots, D\}$, here D = 3 (1 CPU + 2 GPUs)

Goal: minimise total training time. Proxy: find the placement P that minimises r(P), the runtime of one training step

Existing approaches and where they fail

Scotch

Graph partitioning on a cost model

✕ Cost model is often inaccurate

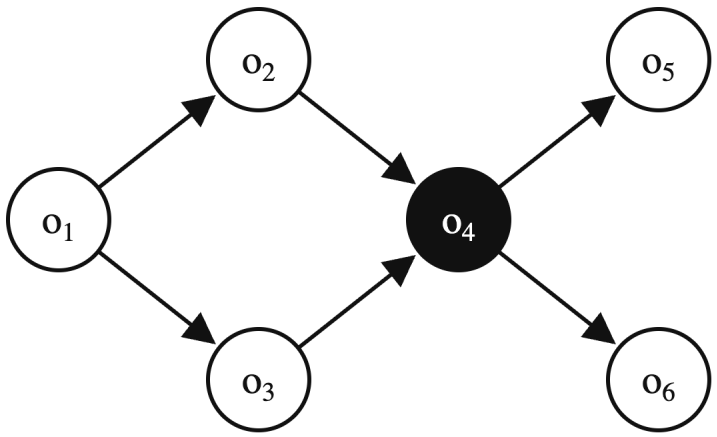
Human experts

Hand-crafted heuristics

✕ Needs deep knowledge of model and hardware

New idea: run one training step with the placement and use the measured runtime as the training signal for RL

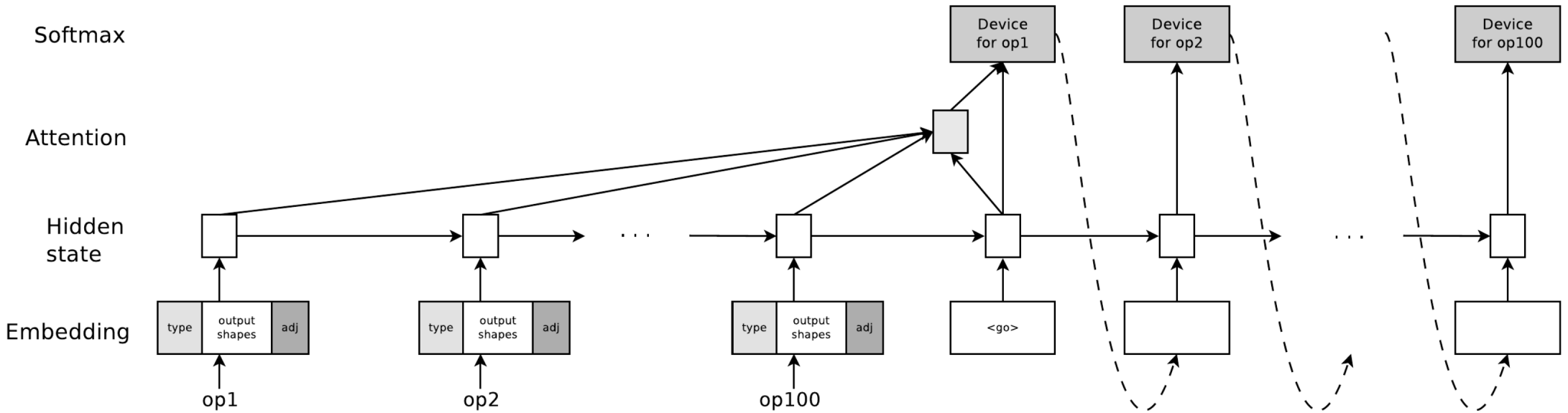
Encoding an operation



embedding of o_4 = concatenation

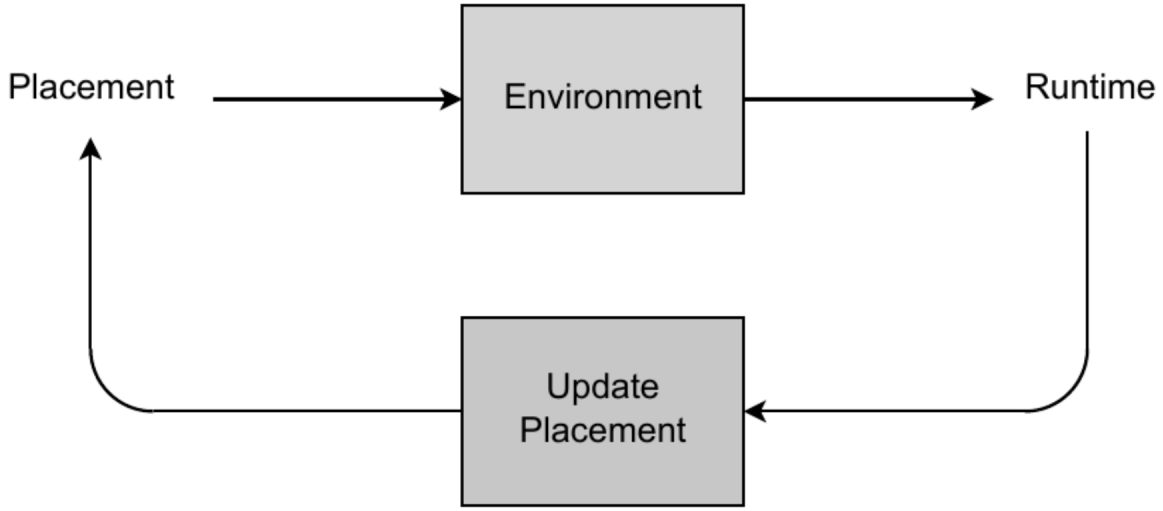
type	**output shape**	**adjacency**
e.g. MatMul	e.g. [32, 1024, 0, 0]	[0, 1, 1, 0, 1, 1]
learned vector per op type	output tensor sizes, zero-padded	1 = direct input or output (o2, o3, o5, o6)

Architecture



- **Seq2seq:** the encoder reads the operations, the decoder outputs one device per operation
- **LSTM** in both encoder and decoder
- **Content-based attention:** each decoder step looks back at all encoder states

Training with policy gradients



- **Policy** $\pi(P \mid G; \theta)$: the seq2seq model's probability of each placement
- **Objective:** minimise $J(\theta) = E[R(P)]$, with $R(P) = \sqrt{r(P)}$ (square root makes noisy runtimes more robust)
- **REINFORCE:** sample K placements, weight each by $R(Pi) - B$, where B is a moving-average baseline
- **Failed placements:** R set to a large constant, the "failing signal"; after 5,000 steps, failed placements no longer update the policy

Co-location groups

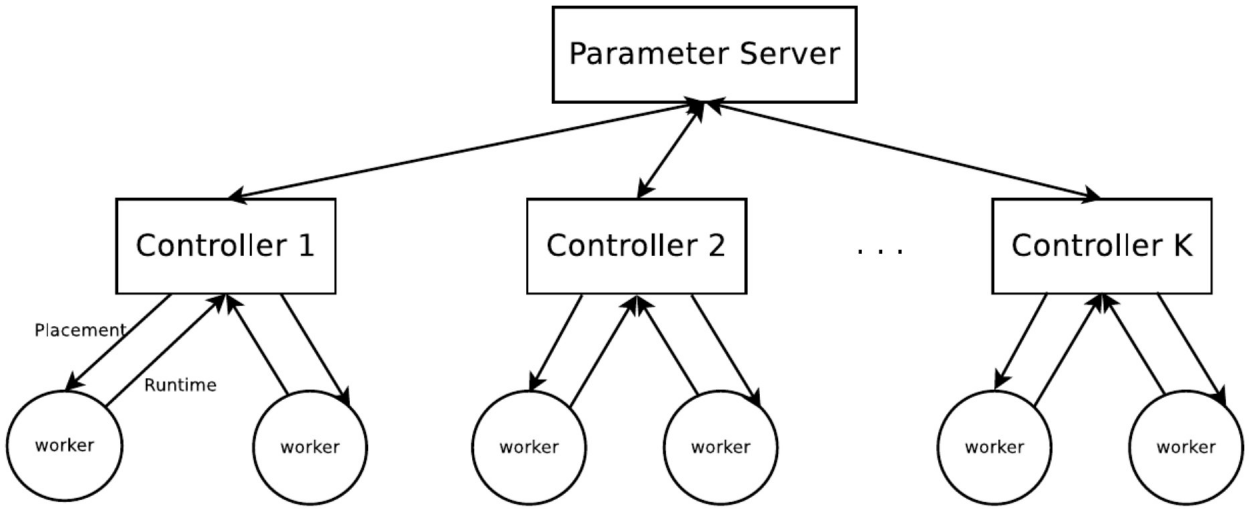
Problem: graphs have thousands of operations, too many for the seq2seq model

Solution: force groups of operations onto the same device, so the model places groups instead of single operations

How groups are formed:

- If an operation's output is consumed only by one other operation, co-locate the two (applied recursively)
- Architecture-specific rules, e.g. each LSTM cell, or several convolution and pooling layers, as one group

Distributed training



- **Parameter server** holds the shared controller parameters
- **Phase 1:** each controller samples placements, one per worker
- **Phase 2:** each worker executes its placement for 10 steps and reports the average runtime
- The controller scales the gradients by the runtimes and updates the parameter server

Search time: 12 to 27 hours per model, with up to 20 controllers × 4–8 workers

Experiments: three benchmarks

Name	Description	Purpose	Size	operations	groups
RNNLM	Stacked LSTM layers	Language modelling	2 LSTM layers of 2048, 40 steps, batch 64	8,943	188
NMT	LSTM encoder–decoder with attention	Machine translation	2 LSTM layers of 1024, sentences up to 40, batch 64	22,097	280
Inception-V3	ConvNet, blocks of parallel convolution and pooling branches	Image classification (ImageNet)	299 × 299 × 3 images, batch 32	31,180	83

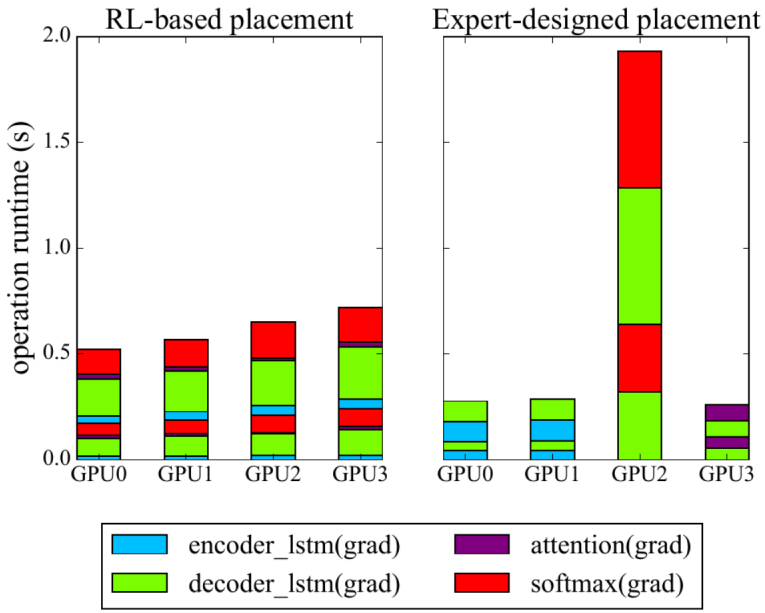
Results: per-step running time

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%

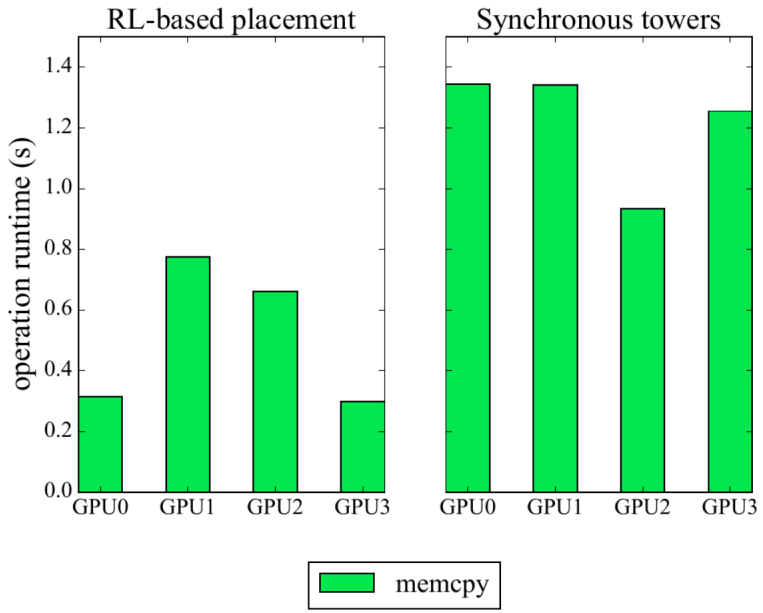
Table 2. Running times (in seconds) of placements found by RL-based method and the baselines (lower is better). For each model, the first row shows the results with 1 CPU and 2 GPUs; the second row shows the results with 1 CPU and 4 GPUs. Last column shows improvements in running time achieved by RL-based 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.

Other findings

NMT: embedding lookups on the CPU, compute balanced better than the expert (Fig. 8)



Inception-V3, 4 GPUs: gain from fewer data copies, not compute balance (Fig. 9)



Critique

Unclear what RL adds

No random search, simulated annealing or genetic baseline with the same compute budget

Weak baselines: Scotch and MinCut are slower than a single GPU, experts are simple heuristics

Mixed results

0% gain over the best baseline in 3 of 6 settings

Gains only for NMT and Inception-V3 on 4 GPUs; only 2 or 4 identical GPUs tested

Costly search, no transfer

New search for every model and hardware setup: 12 to 27 hours on up to 20 controllers × 4–8 workers

No GPU-hours, no break-even against the ~64 h of NMT training saved

Manual grouping

Hand-written rules shrink NMT from 22,097 operations to 280 groups

Shifts part of the problem to humans, and is never ablated

Comparison to X-RLflow

	Device placement (2017)	**X-RLflow (2023)**
Decision	Where each operation runs	Which subgraph rewrites to apply
Target	Training step, multiple GPUs	Inference, single GPU
Model	Seq2seq LSTM over the flattened graph	Graph neural network (graph attention)
RL	REINFORCE, whole placement in one shot	PPO, one rewrite per step

Shared: measured runtime instead of a cost model; trained per graph, no transfer to new models; training cost not reported

Further developments

Unclear what RL adds

FlexFlow (2019): execution simulator plus search, no RL; up to 3.8× higher throughput than prior approaches

Mixed results

Alpa (2022): compiler-based; matches or beats hand-tuned model-parallel systems

Costly search, no transfer

Placeto, GDP (2019): one policy transfers to unseen graphs

Manual grouping

Hierarchical Planner (2018): learns the grouping itself; graphs with over 80,000 operations