Model-Parallelization

FID3024 Systems for Scalable Machine Learning

Jacob Lindbäck, Tianze Wang, Lennart Van der Goten

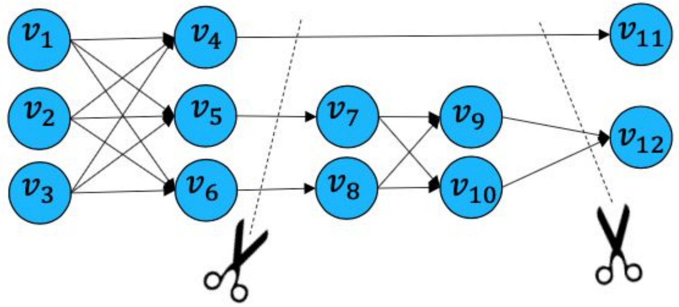
Nov. 2, 2020

Papers

1. The TensorFlow Partitioning and Scheduling Problem
2. Device Placement Optimization with Reinforcement Learning
3. A Hierarchical Model for Device Placement
4. Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning
5. A Single-Shot Generalized Device Placement for Large Dataflow Graphs
6. Spotlight: Optimizing Device Placement for Training Deep Neural Networks

The Tensorflow Partitioning and Scheduling problem: *It's the critical path!*

(Mayer et al 2017)



Some background

Distributed graph processing algorithms via partitioning:

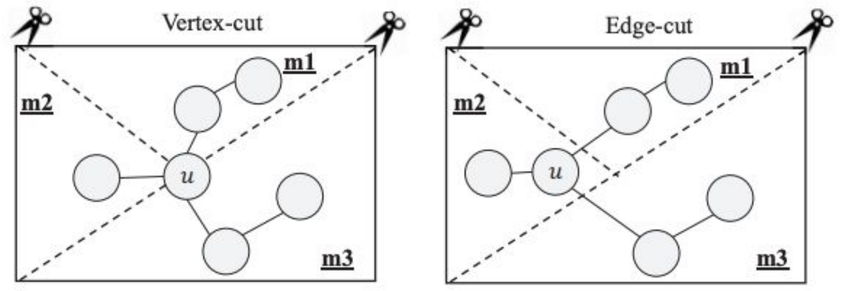


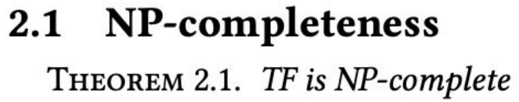
Fig. 1. Vertex-cut and Edge-cut.

Main objective: Minimize traffic between devices while preserving load balance. This is in fact NP-hard, and often approach with heuristics, e.g GraphH, GraphX etc.

Some background

- The computations associated with Tensorflow is also a Graph. Yet different than those of graph partitioning. It has some additional structure with data flows between vertex sources and sinks.

- Mayer et al. suggested a formal description of the tensorflow partitioning and scheduling problem, and used that to establish its NP-completeness.



2.1 NP-completeness

THEOREM 2.1. *TF is NP-complete*

Tensorflow = Partitioning + Scheduling

- Tensorflow involves both partitioning of the DAG, and local scheduling on the devices.

- Since solving the problem exactly is intractable, heuristics are preferred. The authors suggested solving the partitioning problem and the scheduling problem sequentially, and proposed a few heuristics for these subproblems.

The *TF* Problem

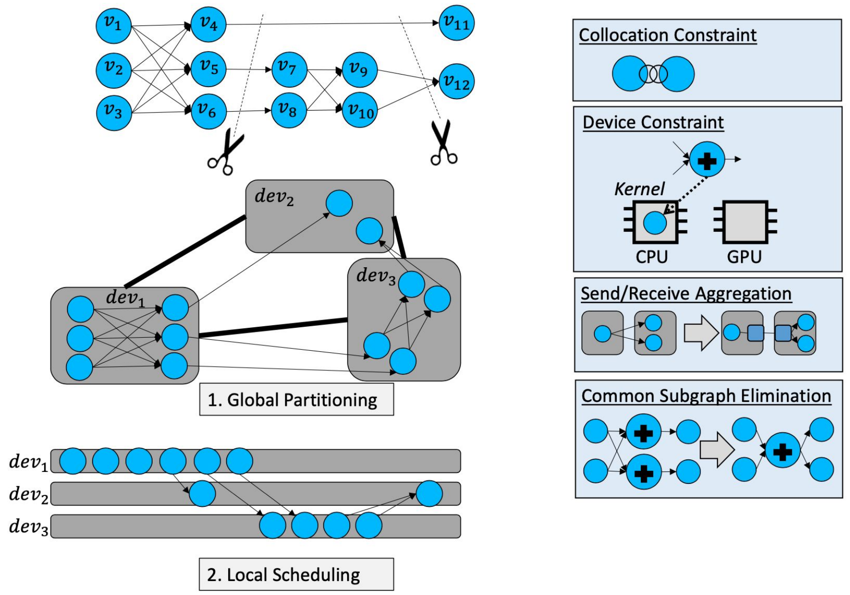


Figure 1: Problem Formulation.

A selection of partitioning heuristics (single objective)

- **CP (Critical Path):** Finds the longest path in the DAG from the source to the sink and assigns its vertices to fastest working unit if possible. If it is not possible, it’s split across the fastest devices.

  The remaining vertices are assigned based on available resources.

A selection of partitioning heuristics (multi-objective)

- **MITE (Memory, Importance, Traffic, Execution Time)**:

  Objective: $\mathrm{mite}(v_i, dev_j) = \mathrm{mem}(dev_j) \times \mathrm{imp}(v_i, dev_j) \times \mathrm{traffic}(v_i) \times \mathrm{exec\_time}(v_i, dev_j)$

  Randomly traverse all the vertices and assign to device that minimize mite.

- **DFS (Depth-First search):**

  Objective: $\mathrm{dfs\_score}(v_i, dev_j) = \mathrm{traffic}(v_i) \times \mathrm{exec\_time}(v_i, dev_j)$

  Using a depth-first search to traverse the vertices. Vertices are assigned to the device that minimizes the dfs_score.

Partitioning heuristics

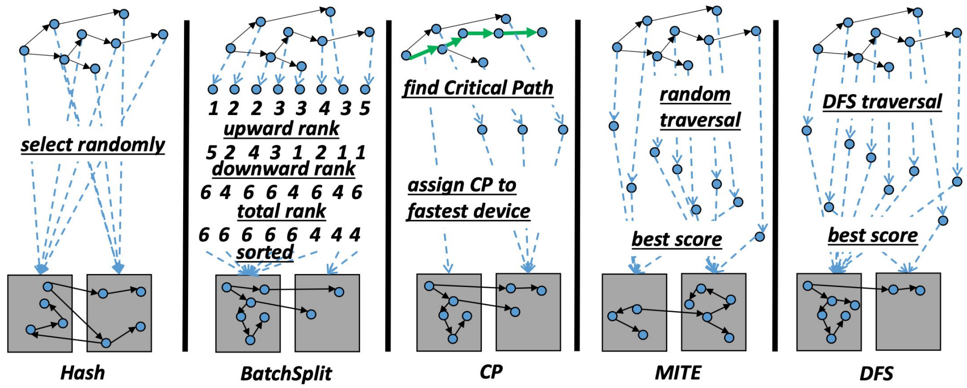


Figure 2: Overview: partitioning strategies.

Scheduling heuristics

- **Highest path computation time first (PCT) -scheduling**: If the computation times of the successors (direct or indirect) of a vertex is high, that vertex is prioritized.

- **Maximum successor rank first (MSR) -scheduling:** Prioritize vertices that are blocking the computations in other devices, to avoid idleness of other working nodes.

The Tensorflow Partitioning and Scheduling problem

Numerical experiments

- Simulated 50 devices with different computational speed and transfer times.
- Benchmarked the heuristics on three Tensorflow networks.
- Baselines: HEFT-algorithm (adjusted for Tensorflow), Hash partitioning & FIFO-scheduling.

Results

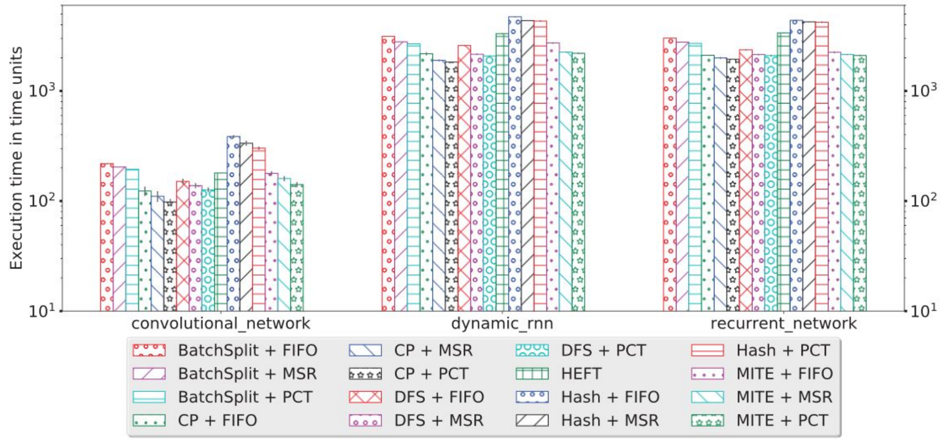


Figure 3: Execution time of 1 iteration at different partitioning and scheduling strategies on 50 devices.

Results

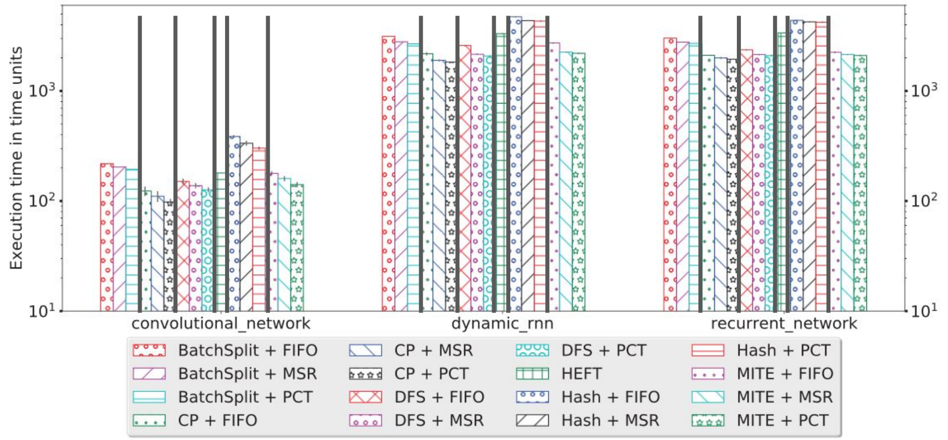


Figure 3: Execution time of 1 iteration at different partitioning and scheduling strategies on 50 devices.

Some thoughts on their contributions

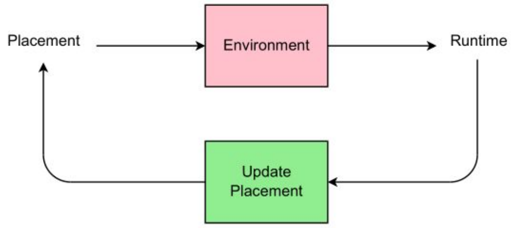
- Having a formal description for the Tensorflow problems is crucial to properly analyze algorithms.

- The proposed heuristics rely on that the computational speed, transfer speed between devices etc. are constant and known apriori. This is often not the case. However, one of their heuristic: CP + FIFO is both competitive and fairly independent of these parameters.

- Would have been interesting to compare their heuristics with the RL-based approaches.

Device Placement Optimisation with RL (*Google Brain*)

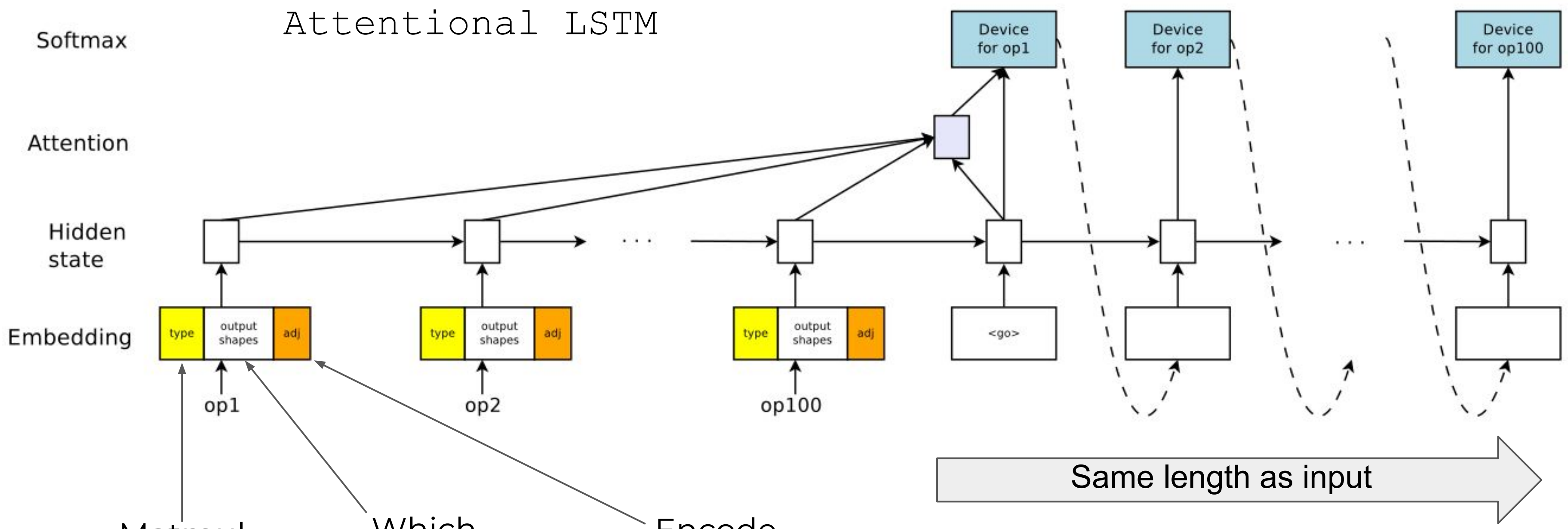
- Motivation: **Device placement** is typically done manually
  - Often difficult to make an informed decision
  - Tedious work
  - Speedup can be significant!
- Idea: Consider **device placement** as a RL problem in which the execution time determines the **reward**
  - Input: Computational graph (a DAG)
  - Output: One assigned device for each operation
- Challenges:
  - Modern neural networks typically comprise thousands of operations
  - Optimization should yield significant speedup to be useful
  - Should minimize amount of human assistance [next paper]



Device Placement Optimisation with RL (*Google Brain*)

- Formal Definition:
  - Let $\mathcal{G}$ be a computational graph with M operations $\{o_1, \ldots, o_M\}$
  - Assume we have D devices (GPUs/CPUs)
  - A placement $\mathcal{P} = \{p_1, \ldots, p_{\mathcal{M}}\}$ assigns exactly one device to each operation
  - Execution time: $r(\mathcal{P})$
- How to define the reward $R(\mathcal{P})$?
  - Straightforward: $R(\mathcal{P}) = r(\mathcal{P})$
    - Not robust enough at the beginning/end of training
  - Better choice: $R(\mathcal{P}) = \sqrt{r(\mathcal{P})}$ resp. large constant if placement is infeasible
- RL Objective:
  - Cost function: $J(\theta) = \mathbf{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G};\theta)}\left[R\left(\mathcal{P}\right)|\mathcal{G}\right]$
  - Gradient: $\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}\left(R\left(\mathcal{P}_i\right) - B\right) \cdot \nabla_\theta \log p\left(\mathcal{P}_i|\mathcal{G};\theta\right)$

Device Placement Optimisation with RL (*Google Brain*)



Matmul, Conv2d etc.

Which dimensions does the output have?

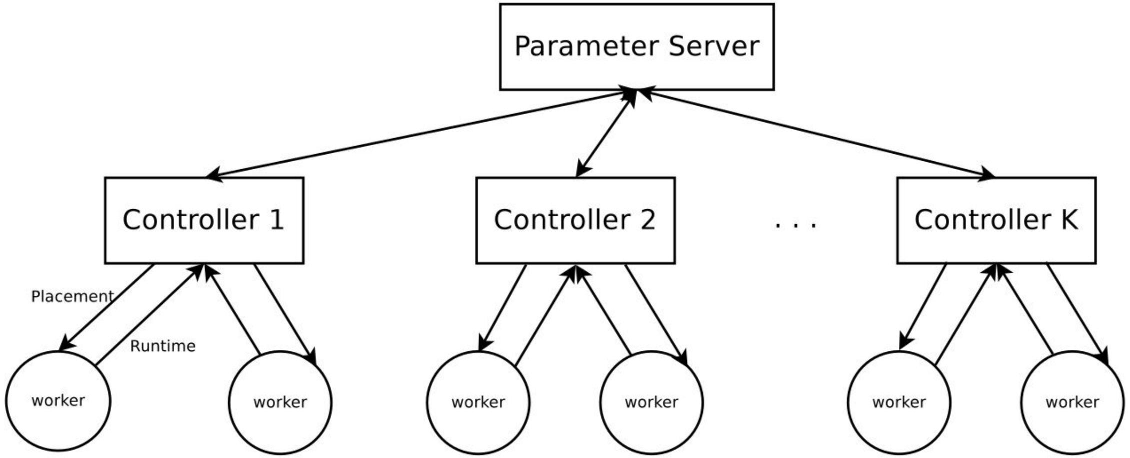
Encode preceding/subsequent operations

Device Placement Optimisation with RL (*Google Brain*)

- The need for **co-location**:
  - The approach does not scale to large networks comprising thousands of operations
  - A **heuristic** needs to be defined that **merges** (“co-locates”) operations
  - Instead of assigning one device to **one** operation, one device is assigned to **multiple** operations
  - Shrinks the sequence length as fewer operations need to be assigned
- Heuristic:
  - The output of each operations is co-located with its gradients
  - If the output of some operation X is solely used by some operation Y both operations are co-located
  - Recursively apply rules until they are no longer applicable
  - Add some manual corrections (e.g. each LSTM cell makes up one group)

Device Placement Optimisation with RL (*Google Brain*)

- Distributed Training:
    - Asynchronous training involving one parameter server and K controllers each having N workers
    - Controller sample placements $\mathcal{P}$ for each of their workers
    - Workers **measure** the elapsed time w.r.t. to some placement $\mathcal{P}$
- Measurements are repeated to get more reliable results

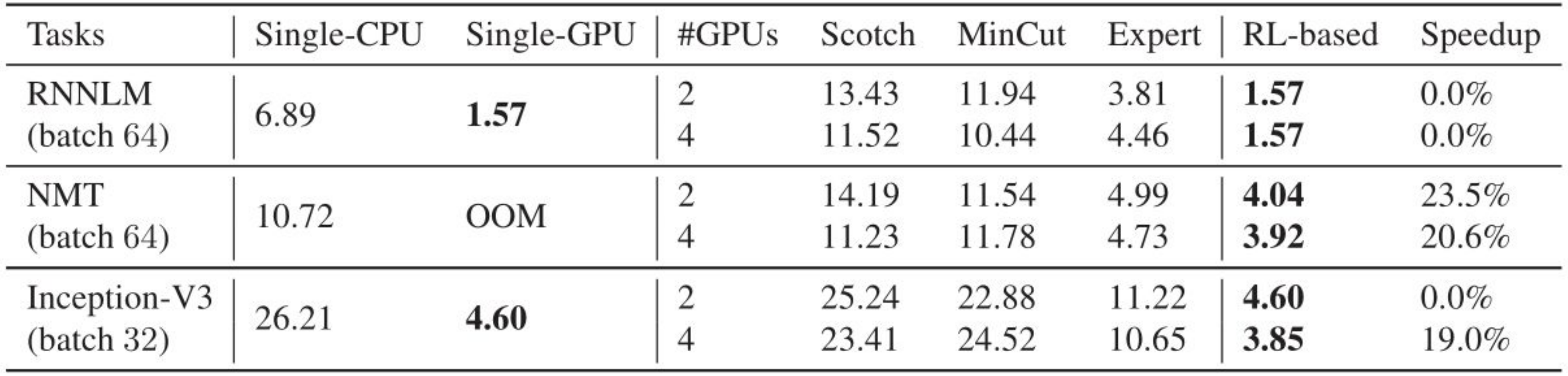


Device Placement Optimisation with RL (*Google Brain*)

- Benchmarks
  - **RNNLM**: Language model with multiple LSTM layers
  - **NMT:** Machine translation model with attention mechanism
  - **Inception-v3:** Image classification

- Baselines
  - **Single-CPU**
  - **Single-GPU**
  - **Scotch:** Off-the-shelf combinational optimizer
  - **MinCut:** Same as **Scotch** but only include GPUs
  - **Expert-designed:**
    - Put entire model on GPU (if it fits)
    - Co-locate attention and softmax

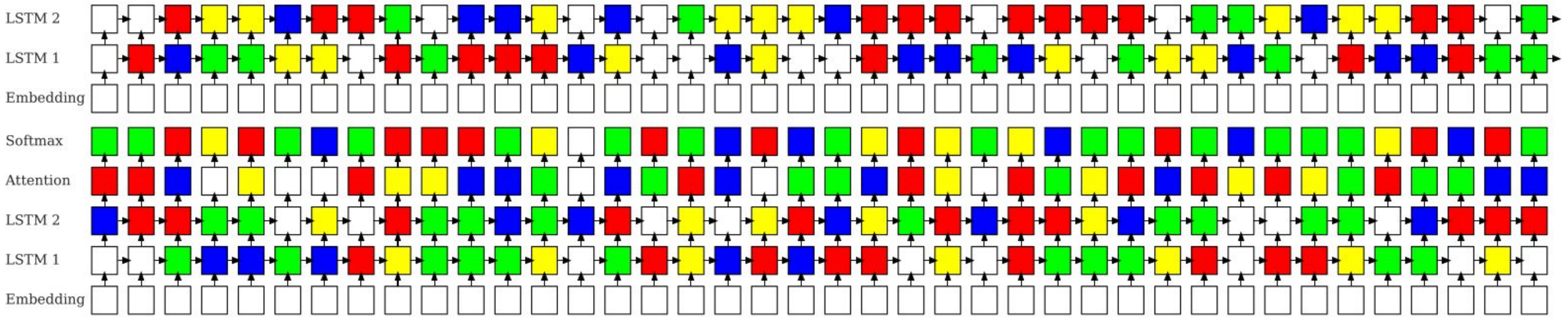
Device Placement Optimisation with RL (*Google Brain*)



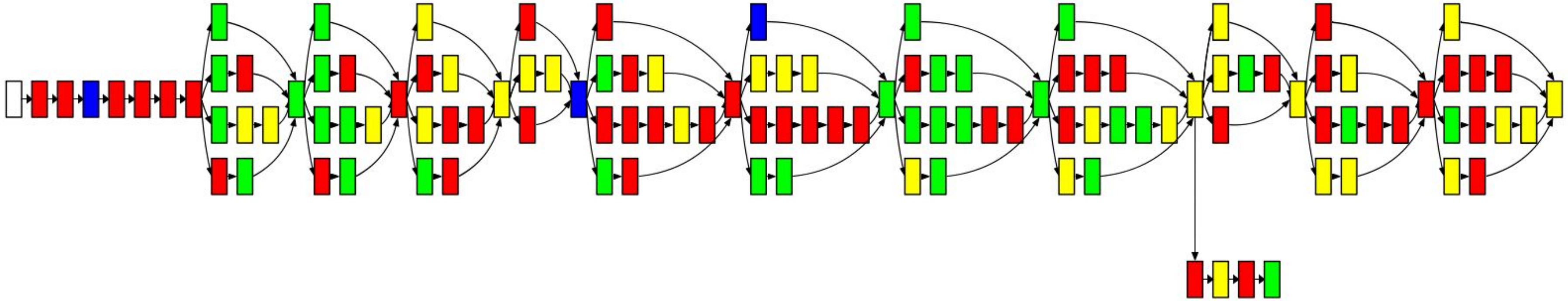
Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**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

- **RNNLM:** Model fits on one GPU and RL-based approach recognizes that
- **NMT:** RL-based approach recognizes that the embeddings can be placed on the CPU
- **Inception-v3:**
    - **2 GPUs:** Places all operations on a single GPU (second GPU idles)
    - **4 GPUs:** Leverages all four GPUs, yields significant speedup

Neural MT:



Inception-v3:



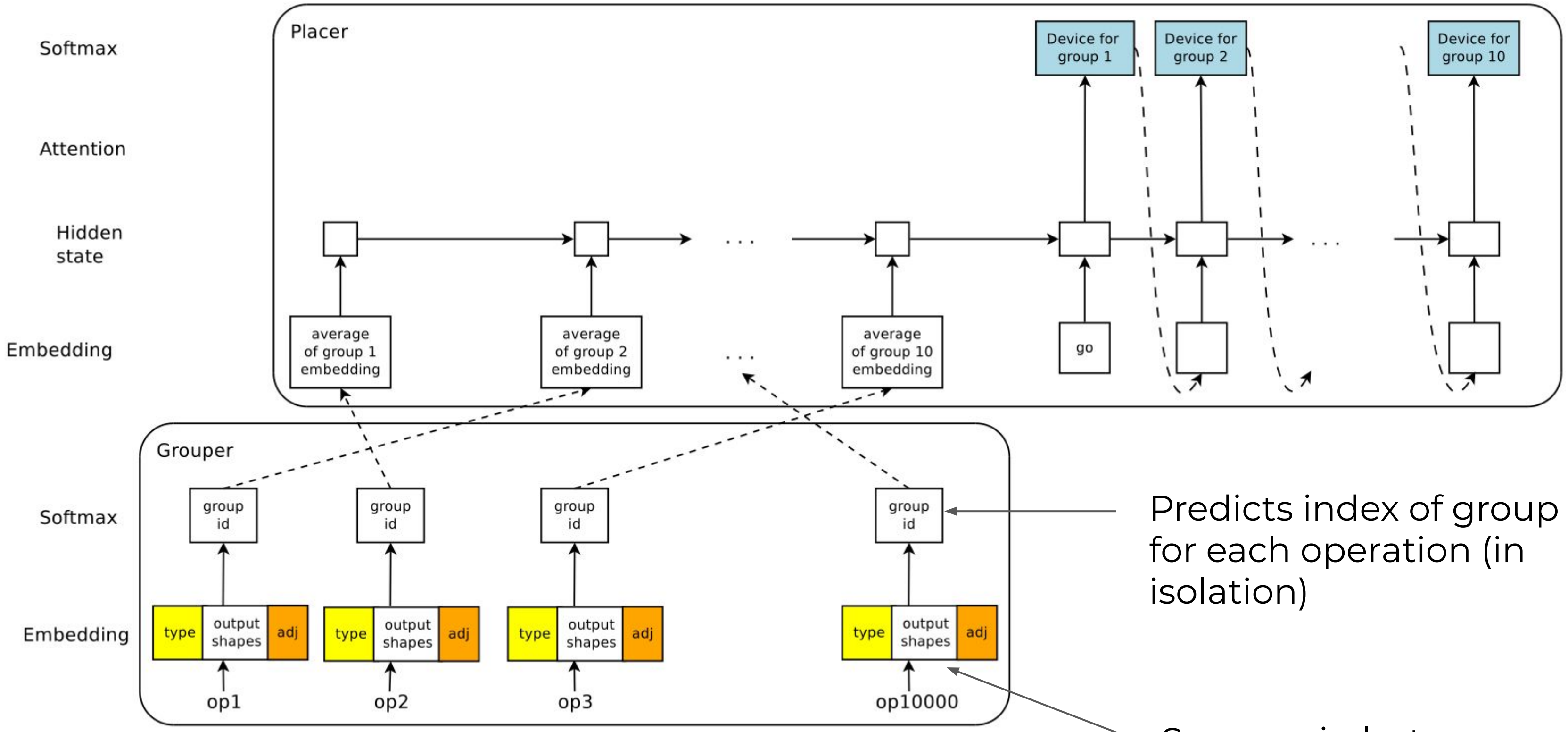
Device Placement Optimisation with RL (*Google Brain*)

- Conclusion:
  - First **RL-driven** approach to tackle the **device placement** task
  - Formulates **device placement** as sequence-to-sequence problem in which operations are associated to devices using LSTMs
  - Only the execution time and the number of devices are used as inputs
  - Identifies non-trivial placements

A Hierarchical Model for Device Placement (*Google Brain*)

- Essentially: An iterative update by the same authors
- Identified weaknesses of **first** paper:
  - Large networks can only be considered when operations are **co-located** *a priori*
  - Decreases granularity which possibly degrades quality of found placements
  - Not **end-to-end** trainable
- Solution: Define **two** networks
  - **Grouper**: Maps operations to groups
  - **Placer**: Maps groups to devices

A Hierarchical Model for Device Placement (*Google Brain*)

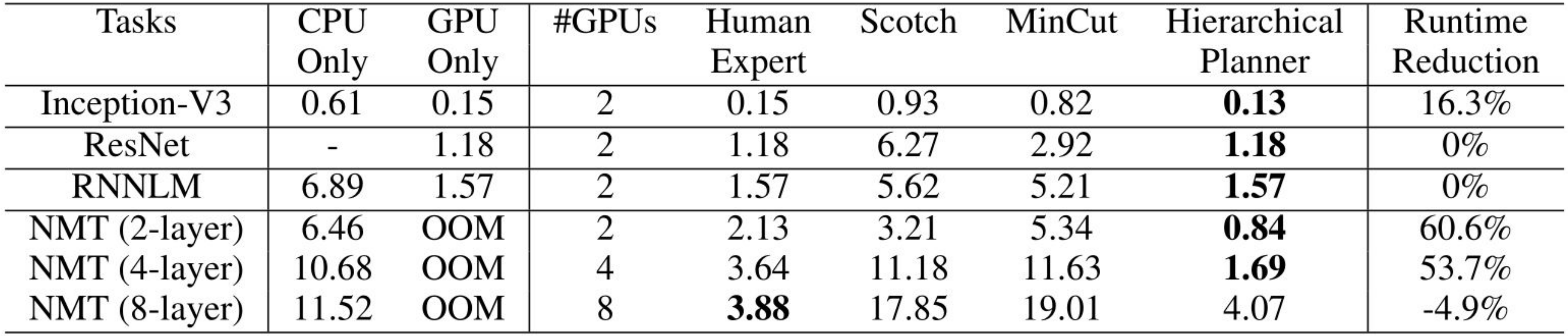


Predicts index of group for each operation (in isolation)

Same as in last paper

Time is not modeled!

A Hierarchical Model for Device Placement (*Google Brain*)



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%

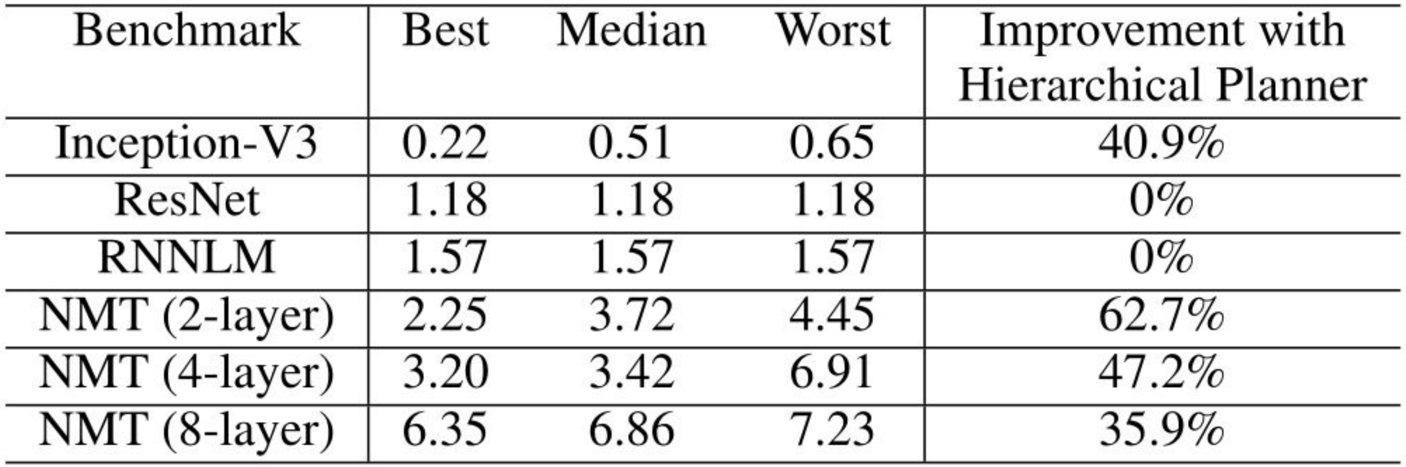
- Authors argue that the new model can't be compared with the previous one because the hardware is different.
- If we measure the speedup w.r.t the best heuristic (a relative comparison), the new method performs significantly better than the old one (60% on **NMT**)

Device Placement Optimisation with RL (*Google Brain*)

- Is it beneficial to optimize **device placements**?
    - Could be faster to just train a model naively and thereby saving the runtime optimization costs…
- **WMT ‘14:** Machine translation corpus for EN-DE
    - New model reduces runtime/step by 46.7%
    - Optimizing **device placement** before saves 265 GPU hours
- Takeaway: It makes sense to do the optimization (at least for **WMT ‘14)**

Device Placement Optimisation with RL (*Google Brain*)

- Is the **Grouper** necessary?
- **Ablation study:** Replace groupings with randomized groupings



Benchmark	Best	Median	Worst	Improvement with Hierarchical Planner
Inception-V3	0.22	0.51	0.65	40.9%
ResNet	1.18	1.18	1.18	0%
RNNLM	1.57	1.57	1.57	0%
NMT (2-layer)	2.25	3.72	4.45	62.7%
NMT (4-layer)	3.20	3.42	6.91	47.2%
NMT (8-layer)	6.35	6.86	7.23	35.9%

Device Placement Optimisation with RL (*Google Brain*)

- Conclusion:
  - Both papers tackle the **device placement** problem using **policy gradients**
  - In order to be able to optimize large networks, the operations need to be grouped first to address performance issues/vanishing gradients
    - First paper uses (recursive) heuristics + human input
    - Second paper uses a **Grouper** network
  - **Optimization reduces overall training time on WMT’14**

Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning

Addanki, Ravichandra, et al. "Placeto: Learning generalizable device placement algorithms for distributed machine learning." *arXiv preprint arXiv:1906.08879* (2019).

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Motivation

- Previous RL-based device placement methods are promising but require significant amount of retraining to find a good placement for each computation graph due to:
  - **Not generalizable** device placement policies (i.e., learn policy only for a single computation graph)
  - **Low training efficiency**

- **Placeto** incorporates two key ideas:
  - Find a sequence of **iterative placement improvements** (simpler to learn)
  - Use **graph embeddings**

Problem Formulation

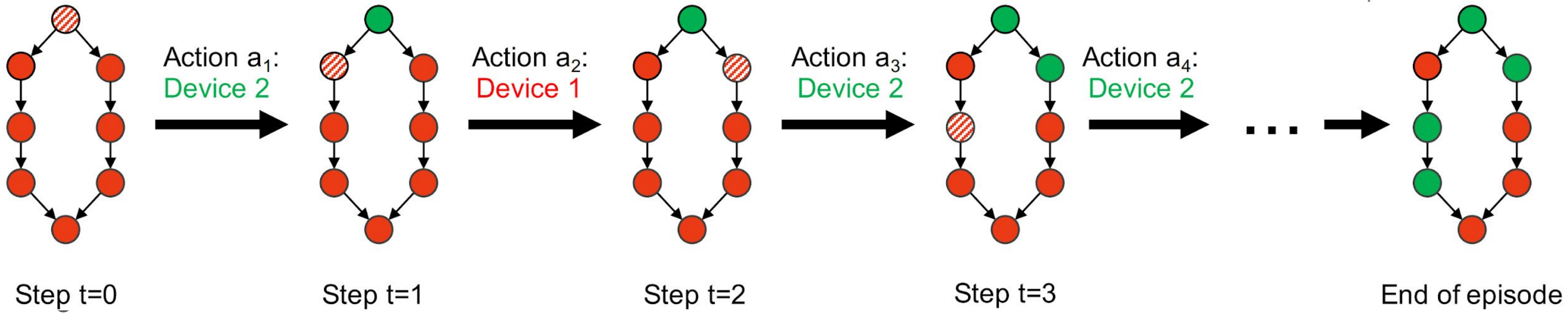
- V : set of atomic computational operations (neurons)
- E : set of communication edges (data dependencies)
- $G(V, E)$: computation graph (DNN)
- $D = \{d_1, d_2, \cdots, d_m\}$: set of $\boldsymbol{m}$ devices (CPU and/or GPU)
- $\pi : V \rightarrow D$: a mapping that assigns a device to each op, (a placement)
- **Goal**: minimize the **execution time** (one step training time) of the placement

Iterative Placement (MDP Formulation)

- Let G be a family of computation graphs, each node v in an observation state of the **Markov decision process (MDP)** has the following features:
  - Estimated run time of v
  - Total size of tensors output by v
  - The current device placement of v
  - A flag indicating whether v has been "visited" before
  - A flag indicating whether is the "current" node for which the placement to be updated.
- At the **initial state s_0**
  - Nodes are assigned to device arbitrarily
  - The visit flags are all 0
  - An arbitrary node is selected as the current node

Iterative Placement (MDP Formulation), Cont.

- At a step t in the MDP, the agent selects an **action** to update the placement for the current node v in **state** s_t. The MDP then transitions to a **new state** s_{t+1} in which v is marked as visited and an unvisited node is selected as the new current node.



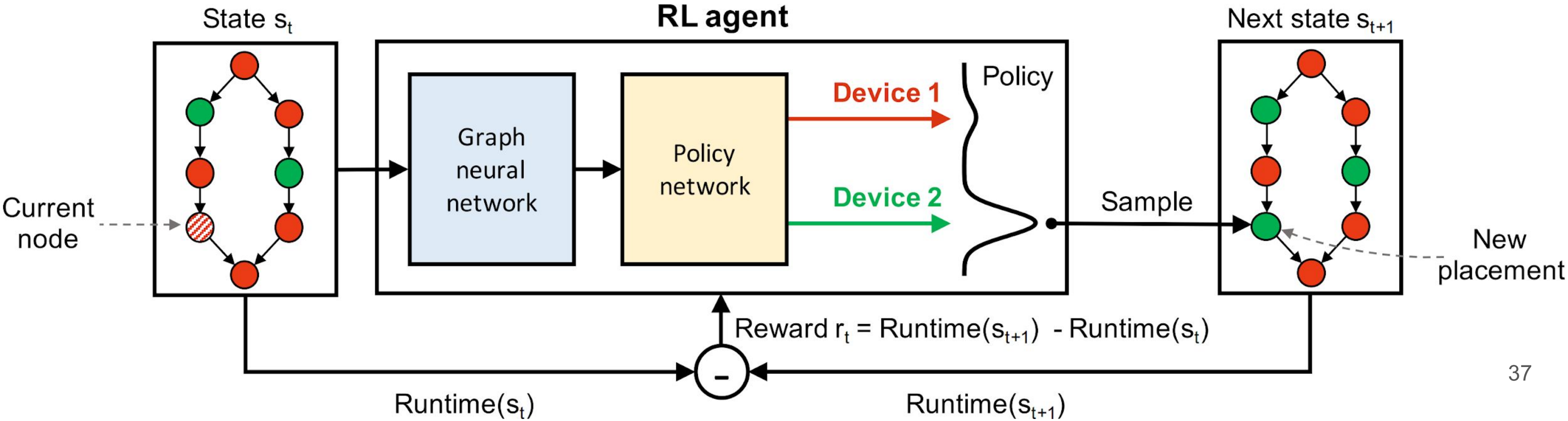
- The episode ends in $|V|$ steps when devices has been assigned to all nodes.

Iterative Placement (MDP Formulation), Cont.

- Two approaches for assigning rewards in the MDP:
  - 0 for intermediate step, the negative run time of the final placement at the final step
  - Assigning an intermediate reward $r_t = r(s_{t+1}) - r(s_t)$
- Intermediate rewards:
  - **Improve** credit assignment in long training episodes (e.g., large NN) and **reduce** variance of the policy gradient estimates
  - Training with intermediate rewards is mode **expensive**.
- To avoid generating placement that **exceeds the memory limit** on device, a penalty in the reward proportional to the peak memory utilization if it is above a certain **threshold** M.

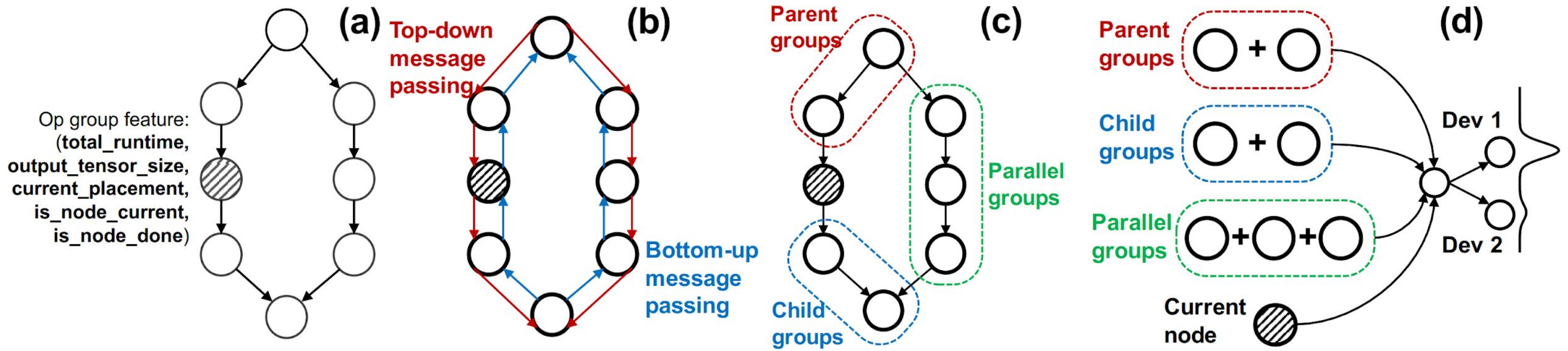
Placeto Architecture

- **Placeto** learns placement policies by directly parameterizing the MDP policy using a neural network.
- At each step t of the MDP, the policy network takes the graph configuration in state s_t as input, and outputs an updated placement for the t-th node.

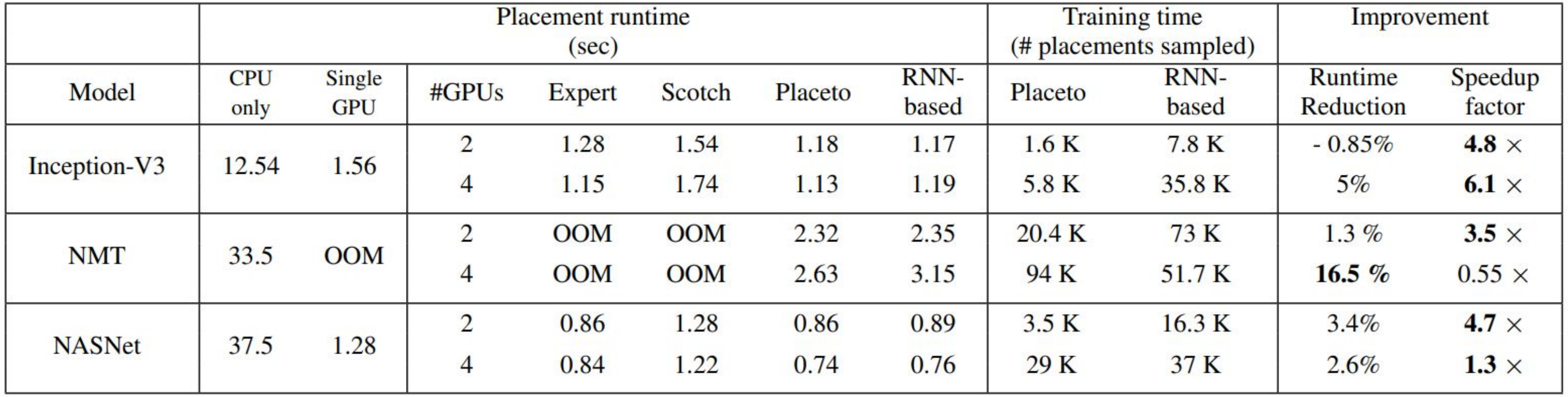


Placeto Architecture, Cont.

- Need to encode the graph-structured information of the state as a real-valued vector.
- Placeto achieves this vectorization via a graph embedding procedure.



Results



	Placement runtime (sec)							Training time (# placements sampled)		Improvement	
Model	CPU only	Single GPU	#GPUs	Expert	Scotch	Placeto	RNN-based	Placeto	RNN-based	Runtime Reduction	Speedup factor
Inception-V3	12.54	1.56	2	1.28	1.54	1.18	1.17	1.6 K	7.8 K	- 0.85%	**4.8** ×
			4	1.15	1.74	1.13	1.19	5.8 K	35.8 K	5%	**6.1** ×
NMT	33.5	OOM	2	OOM	OOM	2.32	2.35	20.4 K	73 K	1.3 %	**3.5** ×
			4	OOM	OOM	2.63	3.15	94 K	51.7 K	**16.5 %**	0.55 ×
NASNet	37.5	1.28	2	0.86	1.28	0.86	0.89	3.5 K	16.3 K	3.4%	**4.7** ×
			4	0.84	1.22	0.74	0.76	29 K	37 K	2.6%	**1.3** ×

Table 1: Running times of placements found by Placeto compared with RNN-based approach [10], Scotch and human-expert baseline. The number of measurements needed to find the best placements for Placeto and the RNN-based are also shown (K stands for kilo). Reported runtimes are measured on real hardware. Runtime reductions and speedup factors are calculated with respect to the RNN-based approach. Lower runtimes and lower training times are better. OOM: Out of Memory. For NMT model, the number of LSTM layers is chosen based on the number of GPUs.

Results, Cont.

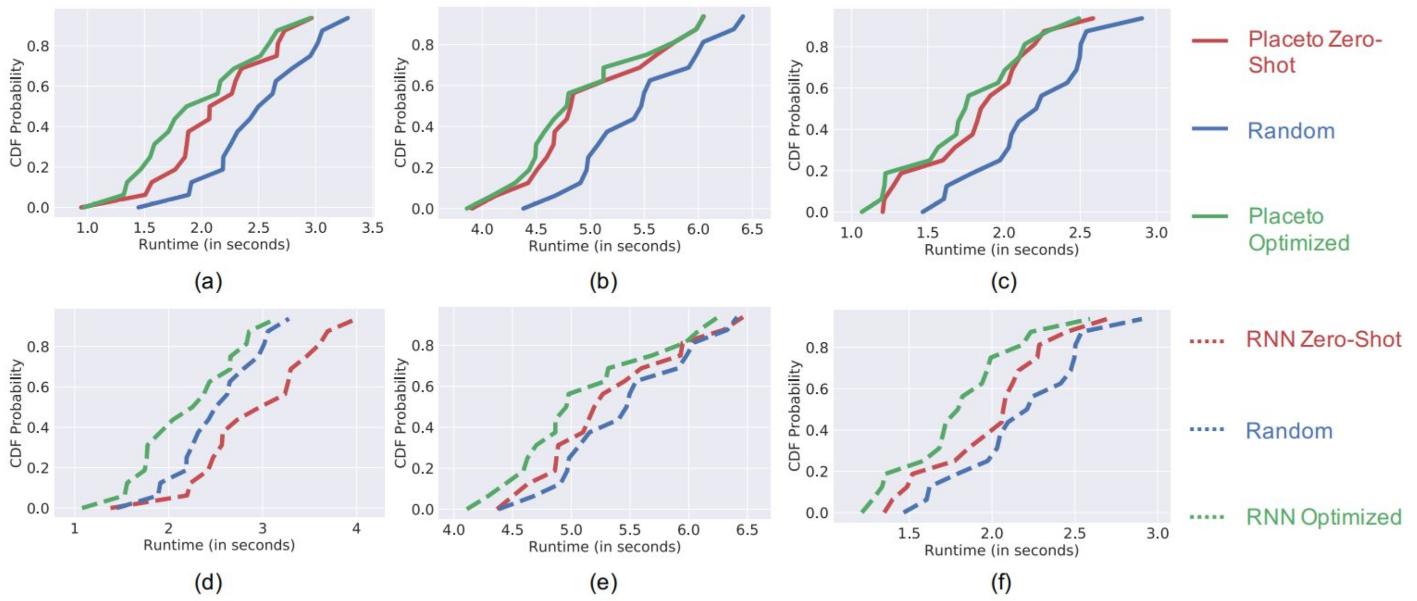


Figure 5: CDFs of runtime of placements found by the different schemes for test graphs from (a), (d) *nmt* (b), (e) *ptb* and (c), (f) *cifar10* datasets. Top row of figures ((a), (b), (c)) correspond to Placeto and bottom row ((d), (e), (f)) to RNN-based approach. *Placeto Zero-Shot* performs almost on par with fully optimized schemes like *Placeto Optimized* and *RNN Optimized* even without any re-training. In contrast, *RNN Zero-Shot* performs much worse and only slightly better than a randomly initialized policy used in *Random* scheme.

Placeto Summary

Pros

- Iterative placement improvement policy
- Generalizability
- More efficient training and can find better placement compared to the chosen baseline
- Simulator to improve training speed
- Code available online

Cons

- Generalizability within the same family of computation graphs
- Policy gradient training
- Still need grouping, otherwise can not scale to large computation graphs
- **Spotlight** is mentioned but not included in the baseline

A Single-Shot Generalized Device Placement for Large Dataflow Graphs (**SGDP**)

Zhou, Yanqi, et al. "A Single-Shot Generalized Device Placement for Large Dataflow Graphs." *IEEE Micro* 40.5 (2020): 26-36

Motivation

- Previous RL-based device placement methods are promising but are:
    - Computationally **expensive**
    - **Unable** to handle large graphs (more than 50,000 nodes)
    - **Do not generalize well** to unseen computation graphs
- **SGDP**: an efficient single-shot, generalized deep RL method based on a scalable sequential attention mechanism over a graph neural network that is transferable to new graphs.

Problem Formulation

- V : set of atomic computational operations (neurons)
- E : set of communication edges (data dependencies)
- $G(V,E)$: computation graph (DNN)
- $D = \{d_1, d_2, \cdots, d_m\}$: set of $\boldsymbol{m}$ devices (CPU and/or GPU)
- $\pi : V \rightarrow D$: a mapping that assigns a device to each op, (a placement)
- **Goal**: minimize the **execution time** (one step training time) of the placement

SGDP: End-to-End Placement Policy

- The policy network of SGDP consists:
  - A **graph embedding** network (learns the graphical representation of any dataflow graph)
  - A **placement** network (learns a placement strategy over the given graph embeddings)
- The two components in the policy network are jointly trained in an end-to-end fashion.
- The RL objective in SGDP is defined to simultaneously reduce the expected runtime of the placements over a set of N computation graph.
- SGDP use **Proximal Policy Optimization (PPO)** to optimize the objective to improve efficiency.

SGDP: End-to-End Placement Policy, Cont.

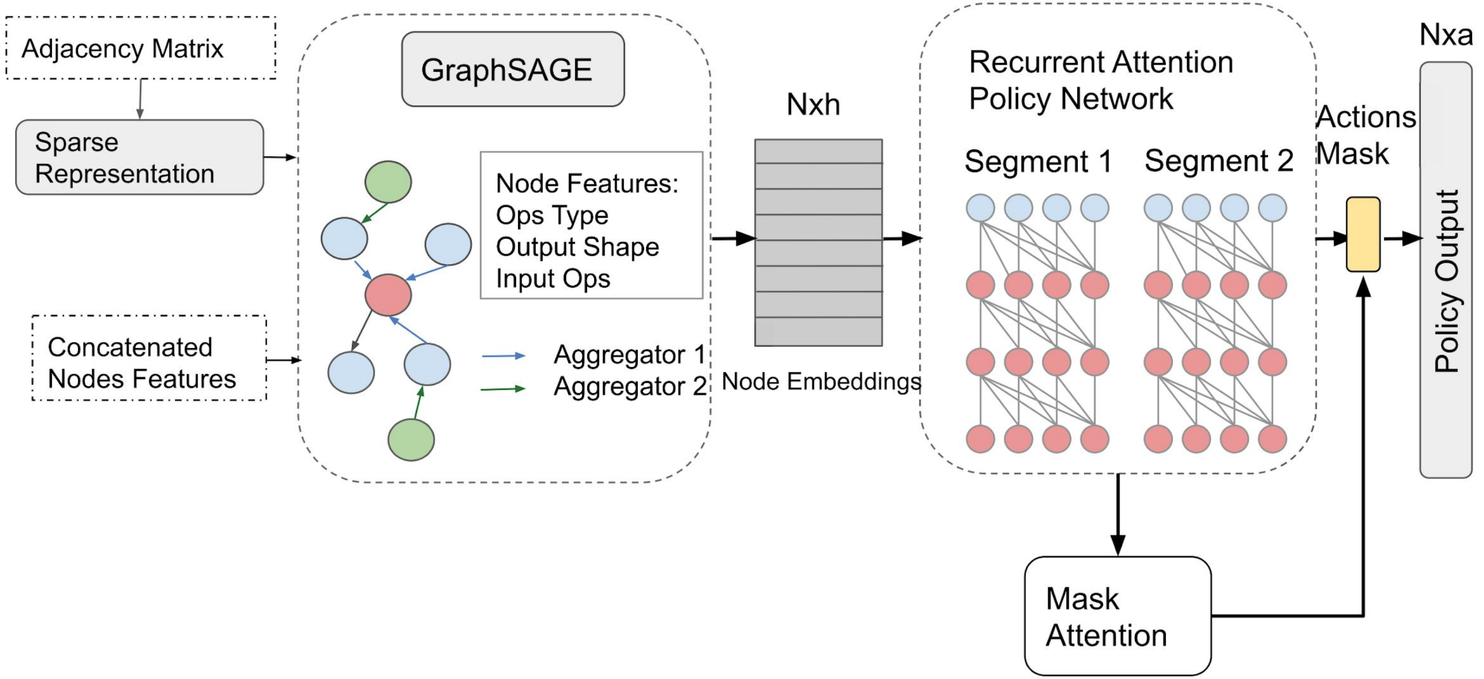


Figure 1. Overview of SGDP: An end-to-end placement network that combines graph embedding and sequential attention. N: Number of nodes. h: Hidden size. d: Number of devices.

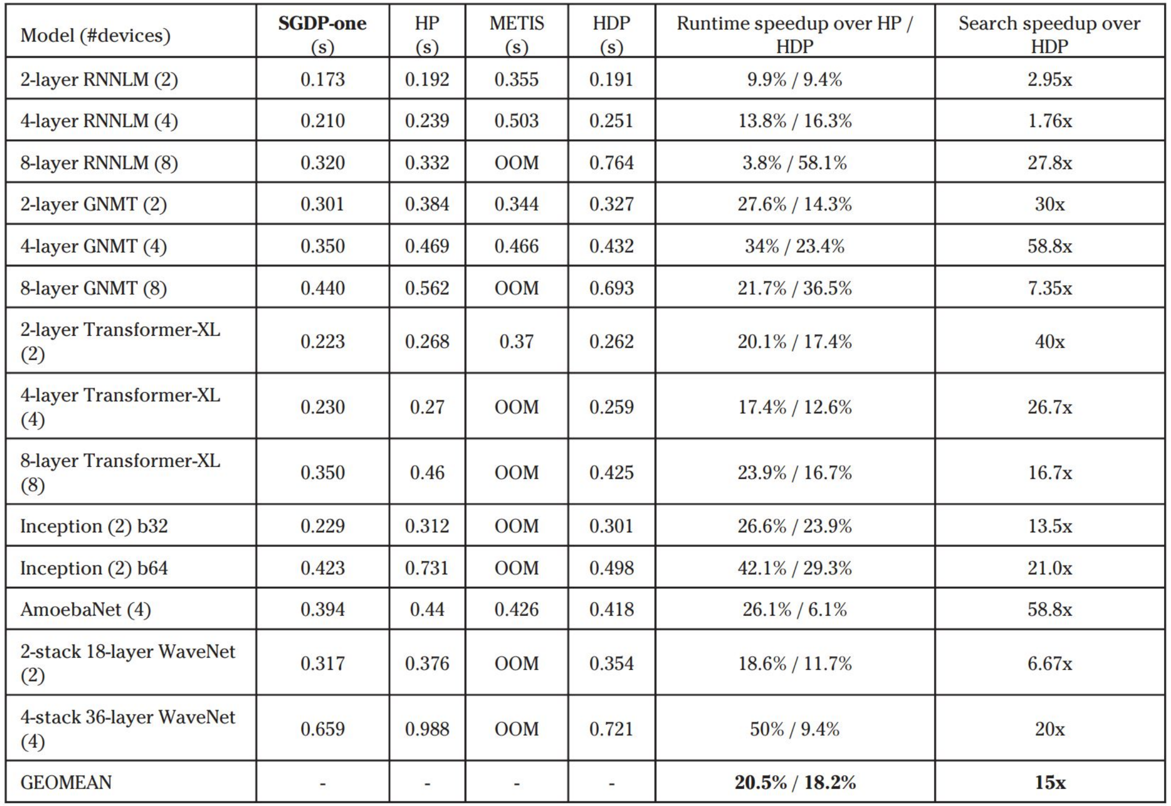
Graph Embedding Network

- SGDP uses **Graph Neural Networks (GNNs)** to capture the topological information in the dataflow graph
- **GraphSAGE** is an inductive framework that leverages node attribute information to efficiently generate representations on previously unseen data.
- SGDP adopts the feature aggregation scheme in GraphSAGE to **model the dependencies** between the operations and build a general, end-to-end device placement method for a wide set of dataflow graphs.
- Nodes and edges in the dataflow graph are represented as the **concatenation** of their meta features (e.g., operation type, output shape, adjacent node ids) and are further encoded by the graph embedding network into a trainable representation.

Placement Networks

- **SGDP** uses a transformer-based attentive network to generate operation placements in an end-to-end fashion.
    - Remove **the positional embedding** in the original transformer (graph embedding already has spatial information, prevent overfitting on node identifications.
    - Use **segment-level recurrence** (capture long-term dependencies efficiently)

Results



Model (#devices)	**SGDP-one** (s)	HP (s)	METIS (s)	HDP (s)	Runtime speedup over HP / HDP	Search speedup over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
8-layer GNMT (8)	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
Inception (2) b32	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Table 1. Runtime comparison between SGDP-one, human expert, TensorFlow METIS, and HDP on six graphs (RNNLM, GNMT, Transformer-XL, Inception, AmoebaNet, and WaveNet). Search speedup is the policy network training time speedup compared to HDP (reported values are averages of six runs).

Results, Cont.

Scalability Analysis

- **SGDP** demonstrate superhuman performance on **large graphs** such as 8-layer GNMT (21.7%/36.5% better than HP/HDP) and 8-layer RNNLM (3.8%/58.1% better than HP/HDP).
- For all of the related SoTA work, **Placeto** and **REGAL** do not provide any results on 8-layer RNNLM or 8-layer GNMT (more than 50 000 nodes). **HDP** reports inferior performance on 8-layer RNNLM and 8-layer GNMT than human placement.

Generalization

- **SGDP-zeroshot**: directly runs inference on the pre-trained SGDP model.
- **SGDP-finetune**: further trains the pre-trained SGDP model for an additional 50 training steps.
- We find that:
  - **SGDP-finetune** almost matches the performance of **SGDP-one**, degrading the placement runtime on average by only 1.2% compared to **SGDP-one** and outperforms both **human placement** and **HDP** significantly.
  - **SGDP-zeroshot** completely eliminates the training for the target unseen graphs, while being only 3.7% worse on average than **SGDP-one** and being over 10% better than **human placement**.

SGDP Summary

Pros

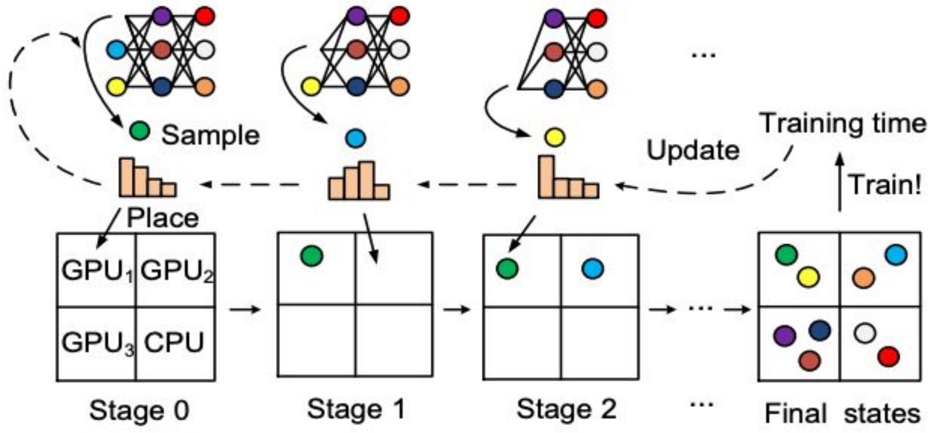
- Generalizability to any unseen graphs
- More efficient training with PPO
- End-to-end placement policy

Cons

- Writing can be better? e.g., using term with defining, GDP and SGDP are used interchangeably.
- Not comparing to better baselines (**Spotlight**, **Placeto**).
- What does single-shot mean?

Spotlight: Optimizing Device Placement for Training Deep Neural Networks

(Gao et al. 2018)



The main idea

- They viewed the partition problem as a Markov Decision Process:

- That is, each operation (i.e. a vertex of the graph) is assigned to the working units sequentially, based on current state.

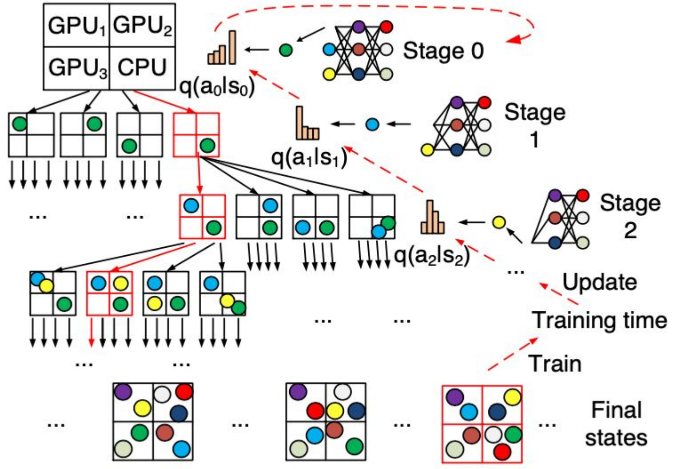


Figure 2. The state tree of a device placement MDP.

Some preliminaries

- Reward function: $r(s_n) = \begin{cases} 0, & n < N \\ \overline{R} - R, & n = N, \end{cases}$
- State-action value function: $Q_\pi(s_n, a_n) = E_{\{a_{n+1},\ldots,a_{N-1}\}\sim\pi}[r(s_N)]$
- Expected reward: $\eta(\pi) = E_{\{a_0,a_1,\ldots,a_{N-1}\}\sim\pi}[r(s_N)]$

$$\eta(\pi') = E_{\{a_0,\ldots,a_{n-1}\}\sim\pi'}[\sum_{a_n} q'(a_n|s_n)Q_{\pi'}(s_n, a_n)]$$

In classical policy ascent, the first representation of the objective is used, which gives the updates:

$$\theta' = \theta + \frac{1}{N}\sum_{a_n\sim\pi} \nabla_\theta \log q(a_n|s_n)\cdot(\overline{R} - R)$$

Proximal Policy Optimization

- Rather than using policy gradient algorithms to update the parameters, they suggested a proximal policy optimization, which is uses the 2nd representation of the expected reward.**:**

$$\eta(\pi') = E_{\{a_0,\dots,a_{n-1}\}\sim\pi'}[\sum_{a_n} q'(a_n|s_n)Q_{\pi'}(s_n,a_n)]$$
$$F_\pi(\pi') = E_{\{a_0,\dots,a_{n-1}\}\sim\pi}[\sum_{a_n} q'(a_n|s_n)Q_{\pi}(s_n,a_n)]$$

- The latter constitutes a lower bound of the expected reward via:

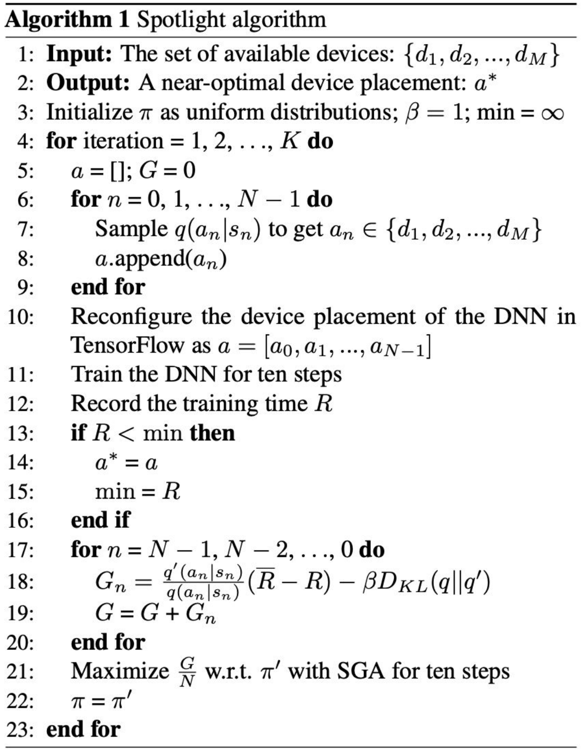
$$\eta(\pi') \geq F_\pi(\pi') - \epsilon_1 - 2\epsilon_2 n D_{KL}^{\max}(\pi||\pi').$$

Proximal Policy Optimization

Rather than maximizing the expected reward, the lower bound is iteratively maximized. The lower bound is approximated as:

$$\max_{\pi'} \frac{1}{N} \sum_{\substack{n=0 \\ a_n \sim \pi}}^{N-1} [\frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')]$$

The Spotlight algorithm

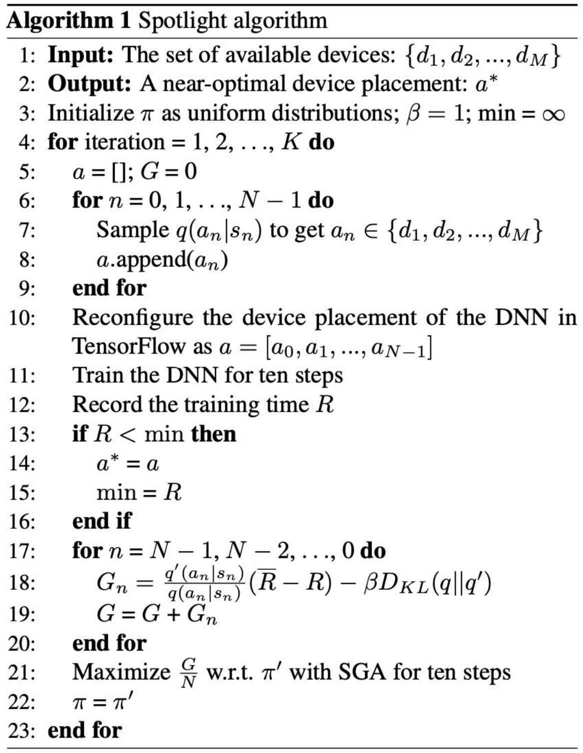


Algorithm 1 Spotlight algorithm

```
**Input:** The set of available devices: $\{d_1, d_2, ..., d_M\}$
**Output:** A near-optimal device placement: $a^*$
Initialize $\pi$ as uniform distributions; $\beta = 1$; min = $\infty$
**for** iteration = 1, 2, . . ., $K$ **do**
    $a$ = []; $G$ = 0
    **for** $n$ = 0, 1, . . ., $N - 1$ **do**
        Sample $q(a_n|s_n)$ to get $a_n \in \{d_1, d_2, ..., d_M\}$
        $a$.append($a_n$)
    **end for**
    Reconfigure the device placement of the DNN in
    TensorFlow as $a = [a_0, a_1, ..., a_{N-1}]$
    Train the DNN for ten steps
    Record the training time $R$
    **if** $R <$ min **then**
        $a^*$ = $a$
        min = $R$
    **end if**
    **for** $n$ = $N - 1$, $N - 2$, . . ., 0 **do**
        $G_n = \frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')$
        $G$ = $G$ + $G_n$
    **end for**
    Maximize $\frac{G}{N}$ w.r.t. $\pi'$ with SGA for ten steps
    $\pi$ = $\pi'$
**end for**
```

The Spotlight algorithm

Simulate actions from current policy

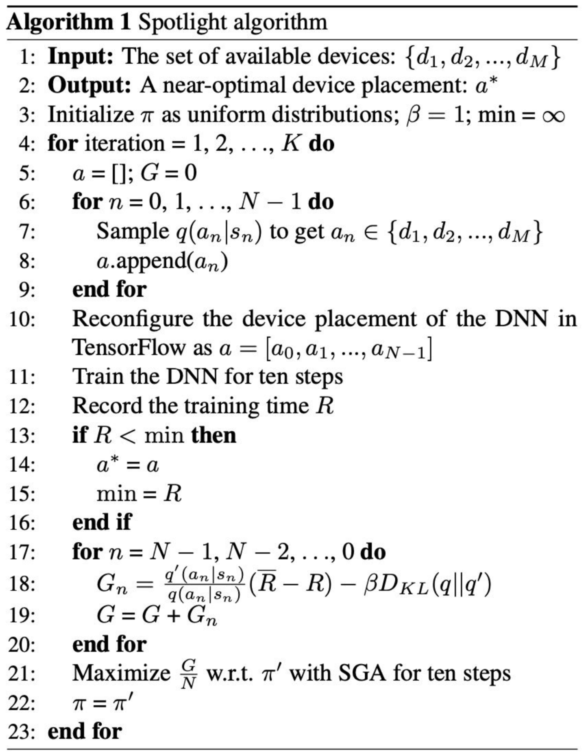


Algorithm 1 Spotlight algorithm

```
Input: The set of available devices: ${d_1, d_2, ..., d_M}$
Output: A near-optimal device placement: $a^*$
Initialize $\pi$ as uniform distributions; $\beta = 1$; min = $\infty$
for iteration = 1, 2, . . ., $K$ do
    $a$ = []; $G$ = 0
    for $n$ = 0, 1, . . ., $N - 1$ do
        Sample $q(a_n|s_n)$ to get $a_n \in \{d_1, d_2, ..., d_M\}$
        $a$.append($a_n$)
    end for
    Reconfigure the device placement of the DNN in TensorFlow as $a = [a_0, a_1, ..., a_{N-1}]$
    Train the DNN for ten steps
    Record the training time $R$
    if $R <$ min then
        $a^*$ = $a$
        min = $R$
    end if
    for $n$ = $N - 1$, $N - 2$, . . ., 0 do
        $G_n = \frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')$
        $G$ = $G$ + $G_n$
    end for
    Maximize $\frac{G}{N}$ w.r.t. $\pi'$ with SGA for ten steps
    $\pi$ = $\pi'$
end for
```

The Spotlight algorithm

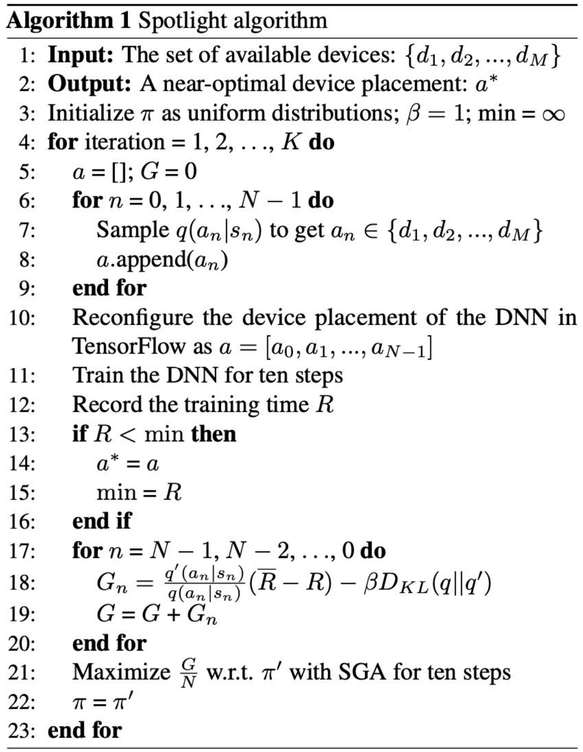
Estimate the training time of the current actions. If it’s better that previous iterates, then update the currently best action.



Algorithm 1 Spotlight algorithm

```
1:  Input: The set of available devices: $\{d_1, d_2, ..., d_M\}$
2:  Output: A near-optimal device placement: $a^*$
3:  Initialize $\pi$ as uniform distributions; $\beta = 1$; min = $\infty$
4:  for iteration = 1, 2, . . ., $K$ do
5:      $a$ = []; $G$ = 0
6:      for $n$ = 0, 1, . . ., $N - 1$ do
7:          Sample $q(a_n|s_n)$ to get $a_n \in \{d_1, d_2, ..., d_M\}$
8:          $a$.append($a_n$)
9:      end for
10:     Reconfigure the device placement of the DNN in
        TensorFlow as $a = [a_0, a_1, ..., a_{N-1}]$
11:     Train the DNN for ten steps
12:     Record the training time $R$
13:     if $R <$ min then
14:         $a^*$ = $a$
15:         min = $R$
16:     end if
17:     for $n$ = $N - 1, N - 2$, . . ., 0 do
18:         $G_n = \frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')$
19:         $G$ = $G$ + $G_n$
20:     end for
21:     Maximize $\frac{G}{N}$ w.r.t. $\pi'$ with SGA for ten steps
22:     $\pi$ = $\pi'$
23: end for
```

The Spotlight algorithm



Algorithm 1 Spotlight algorithm

```
1: **Input:** The set of available devices: ${d_1, d_2, ..., d_M}$
2: **Output:** A near-optimal device placement: $a^*$
3: Initialize $\pi$ as uniform distributions; $\beta = 1$; min = $\infty$
4: **for** iteration = 1, 2, . . ., $K$ **do**
5:     $a$ = []; $G$ = 0
6:     **for** $n$ = 0, 1, . . ., $N - 1$ **do**
7:         Sample $q(a_n|s_n)$ to get $a_n \in {d_1, d_2, ..., d_M}$
8:         $a$.append($a_n$)
9:     **end for**
10:    Reconfigure the device placement of the DNN in
       TensorFlow as $a = [a_0, a_1, ..., a_{N-1}]$
11:    Train the DNN for ten steps
12:    Record the training time $R$
13:    **if** $R <$ min **then**
14:        $a^*$ = $a$
15:        min = $R$
16:    **end if**
17:    **for** $n$ = $N - 1$, $N - 2$, . . ., 0 **do**
18:        $G_n = \frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')$
19:        $G$ = $G$ + $G_n$
20:    **end for**
21:    Maximize $\frac{G}{N}$ w.r.t. $\pi'$ with SGA for ten steps
22:    $\pi$ = $\pi'$
23: **end for**
```

Compute the objective function and maximize w.r.t the new policies.

Model of the placement policy: RNN with LSTM-cells

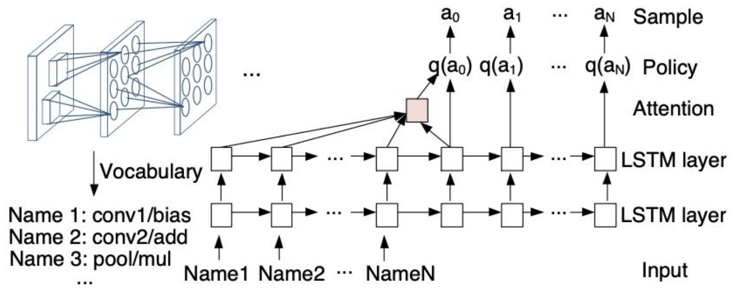
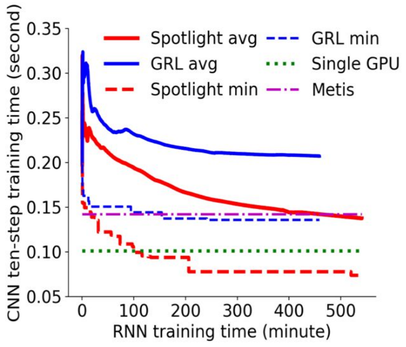


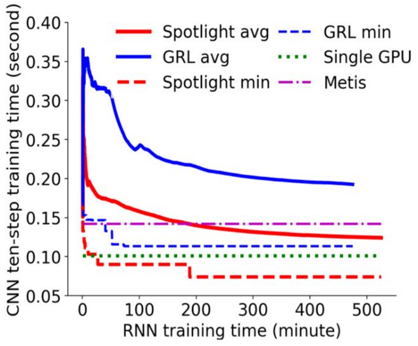
Figure 3. Using a sequence-to-sequence recurrent neural network to represent the placement policy.

Rather than feeding all operations into the net, they use a operation vocabulary to group them into larger ops. This ensures that related operations are co-located.

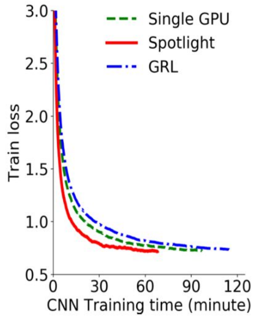
Results



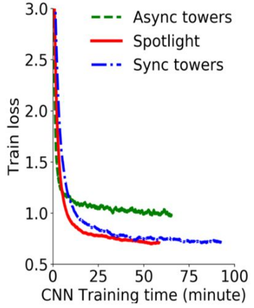
(a) 4 GPUs and 1 CPU



(b) 2 GPUs and 1 CPU



(a) 2 GPUs and 1 CPU

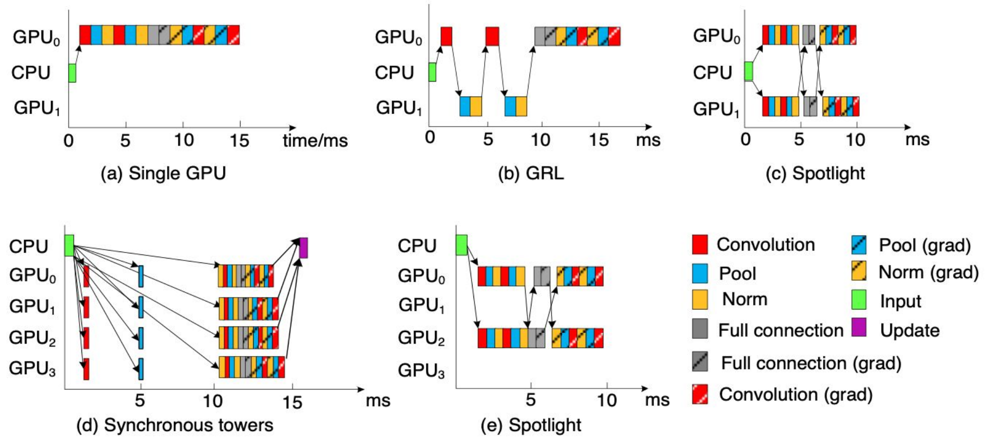


(b) 4 GPUs and 1 CPU

Table 1. Per-step training time (in seconds) of placements given by the baselines for the environment with 4 GPUs. Experts place each LSTM layer on one GPU.

Models	Experts	Metis	GRL	*Spotlight*
RNNLM	3.86	6.12	3.15	2.27
NMT	5.54	10.50	4.74	3.62

Clever partitions that Spotlight finds



Some thoughts

- Applying proximal policy optimization to the partitioning problem is both very innovative, and a seemingly powerful technique.

- The derived properties are based the theoretical lower bound of the expected reward:

$$\max_{\pi'} F_\pi(\pi') - \epsilon_1 - 2\epsilon_2 n D_{KL}^{\max}(\pi||\pi').$$

  While in their implementation, the approximation is considered:

$$\max_{\pi'} \frac{1}{N} \sum_{\substack{n=0 \\ a_n \sim \pi}}^{N-1} [\frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')]$$

  Beta is treated as a hyperparameter (set to 1). Are there tighter lower bounds to exploit?

… yet another comment

- Their grouping strategy is both intuitive and simple, but might be suboptimal. Would have been interesting to see the effect of augmenting Spotlight with a model-based grouper (e.g. one similar to that of the Hierarchical model considered in this module).