DEVICE PLACEMENT OPTIMIZATION WITH REINFORCEMENT LEARNING

11/12/21

Authors: Mirhoseini et al. [PMLR 2017]
Presenter: Joel Rorseth, MMath (thesis) in Computer Science

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

Roadmap

1. Introduction and Background
2. Device Placement Optimization with RL
3. Evaluation
4. Conclusion

INTRODUCTION AND BACKGROUND

Distributed DNN Training

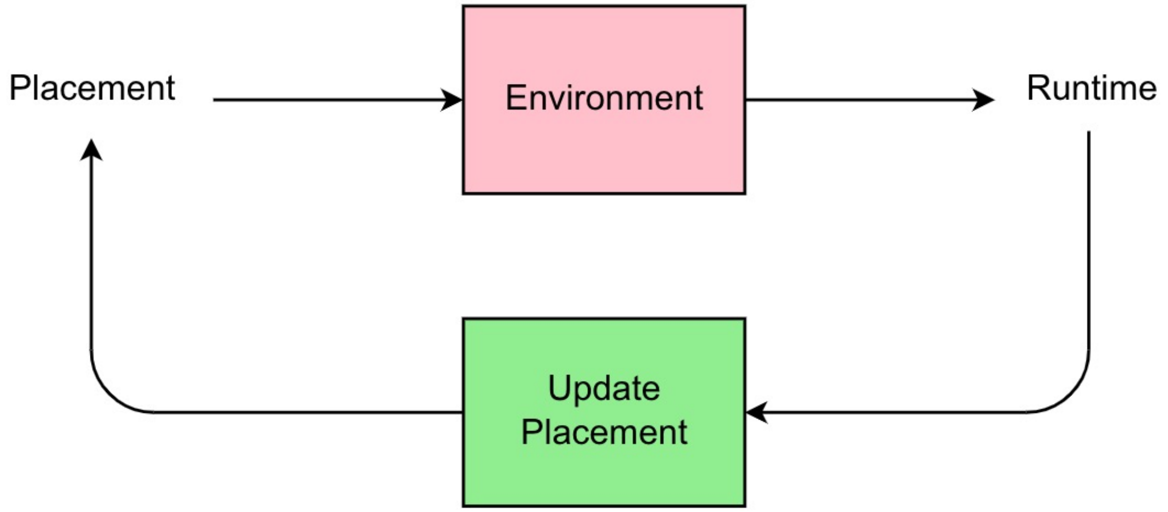
- The success of DNNs has come at a cost of increasing size and computational requirements for both *training* and *inference*
- Two predominant categories for parallel DNN training methods…
- **Data Parallelism (DP):** DNN replicated to n workers, each trained using $\frac{1}{n}$ of the data
- **Model Parallelism (MP):** DNN operators are partitioned across n workers, which operate sequentially to train all data

Device Placement

- Model Parallelism requires partitioning DNN operators to devices within a heterogeneous distributed environment
- Typically, practitioners **manually** specify device placement using simple heuristics
- Ideally, an algorithm should mathematically determine an optimal device placement
  - Eliminate the need for experts, and improve upon their suggested placements

Reinforcement Learning to Explore Placements

- Reinforcement learning (RL) is well suited for navigating the placement space
- The high-level solution:
  - Policy can yield placements
  - Reward / cost can be quantified via execution time of training + inference
  - Represent policy using neural network, calculate policy gradients w.r.t. this execution time



Comparison to Related Work

- RL has been applied to similar problems:
  - Job scheduling and resource management (Mao et al., 2016)
  - Combinatorial optimization (Vinyals et al., 2015; Bello et al., 2016)
- The proposed method is the first to apply RL to device placement optimization
  - Practical, large-scale models and data
  - An objective function that will minimize placement runtime
- Graph partitioning can be applied to device optimization
  - Scotch optimizer (Pellegrini, 2009) will be used as a baseline

DEVICE PLACEMENT OPTIMIZATION WITH RL

Defining the Problem Mathematically

- A TensorFlow computation graph $\mathcal{G}$ consists of:
  - A list of M operations $\{o_1, o_2, \ldots, o_M\}$
  - A list of D available devices $\{1, \ldots, D\}$
- A placement $\mathcal{P} = \{p_1, p_2, \ldots, p_M\}$ is the assignment of operations to devices
- $r(\mathcal{P})$ denotes execution time for $\mathcal{G}$ under placement $\mathcal{P}$
  - For more robust learning, authors instead use $\mathrm{R}(\mathcal{P}) = \sqrt{r(\mathcal{P})}$
- **Problem:** Find the optimal device placement $\mathcal{P}$ that minimizes $\mathrm{R}(\mathcal{P})$

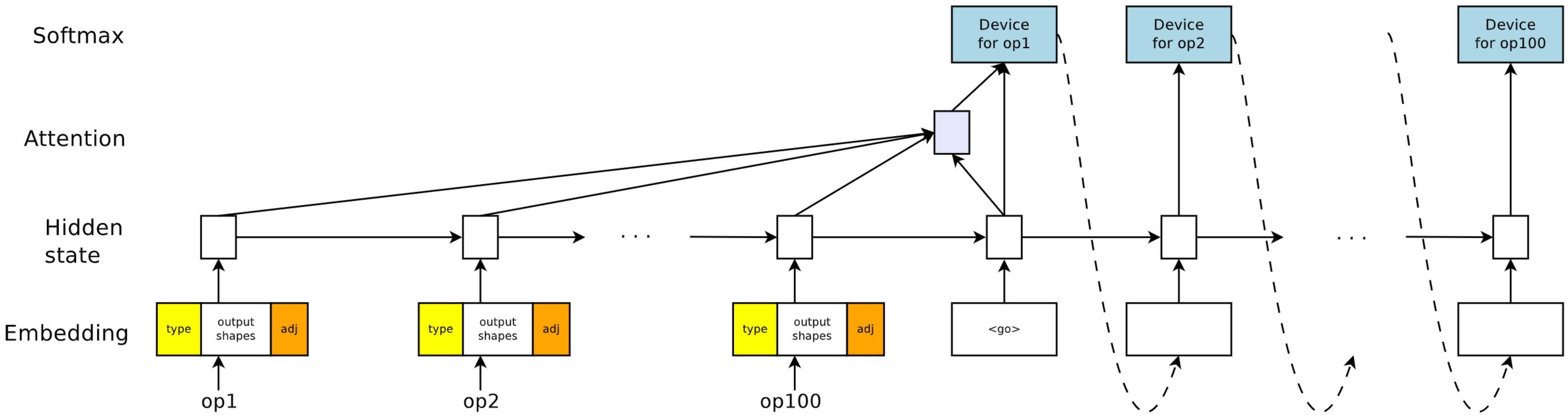
Formulating the Problem Using Reinforcement Learning

- Train a stochastic policy: $\pi(\mathcal{P}|\mathcal{G};\theta)$
- Minimize the objective function: $\mathcal{J}(\theta) = \mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}|\mathcal{G};\theta)}[\mathrm{R}(\mathcal{P})|\mathcal{G}]$
- Learn parameters using Adam (Kingma & Ba, 2014) based on policy gradients from REINFORCE (Williams, 1992) :
  - $\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\mathcal{P}\sim\pi(\mathcal{P}|\mathcal{G};\theta)}[\mathrm{R}(\mathcal{P}) \cdot \nabla_\theta \log p\,(\mathcal{P}|\mathcal{G};\theta)]$
- In practice, we estimate by drawing K samples
  - $\nabla_\theta \mathcal{J}(\theta) \approx \frac{1}{K}\sum_{i=1}^{K}(\mathrm{R}(\mathcal{P}_i) - B) \cdot \nabla_\theta \log p\,(\mathcal{P}_i|\mathcal{G};\theta)$
  - Reduce variance using baseline B (moving average works well)

Policy Network

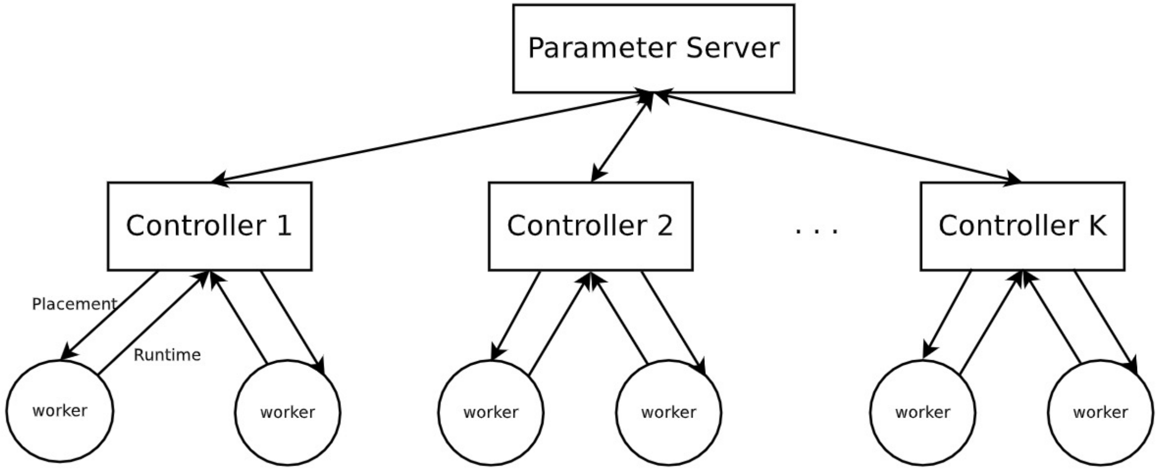
- Policy is defined using a sequence-to-sequence model which uses LSTM with Attention mechanism
  - In this paper, the policy yields placements
- **Encoder** embeds sequence of operations in $\mathcal{G}$
  - Embedding consists of operation type, output shape, and I/O adjacency info
- **Decoder** outputs device assignments for each operation
  - Outputs have tunable embeddings that are fed to subsequent decoder time steps

Policy Network Architecture



Training the Policy Model

- Asynchronous distributed training creates controllers to execute the policy
- Each controller samples K placements, then assigns each to one of its K workers
- Workers report running time to controller, which calculates gradients and sends them to parameter server



Implementation Details

- The authors use 20 controllers, each with 4 or 8 workers
  - Finding optimal placements took anywhere from **12 to 27 hours**
- Each controller maintains a separate baseline B
- To scale better, operations are manually co-located
  - Default TensorFlow co-location groups are used
  - Additional heuristic co-locates operations whose output is sent to only one other operation
  - Co-location rules are manually specified for different types of models

EVALUATION

Evaluation Models

- Evaluate placements yielded for 3 well-known deep learning models:
  - **Recurrent Neural Network Language Model with LSTM layers (RNNLM)**
    - (Zarembaet al., 2014; Jozefowicz et al., 2016)
  - **Neural Machine Translation with attention (NMT)**
    - (Bahdanau et al., 2015; Wu et al., 2016)
  - **Inception-V3**
    - (Szegedy et al., 2016)
- Compare placements by their resulting training execution time, as follows:
  - 1 forward pass + 1 backward pass + 1 parameter update

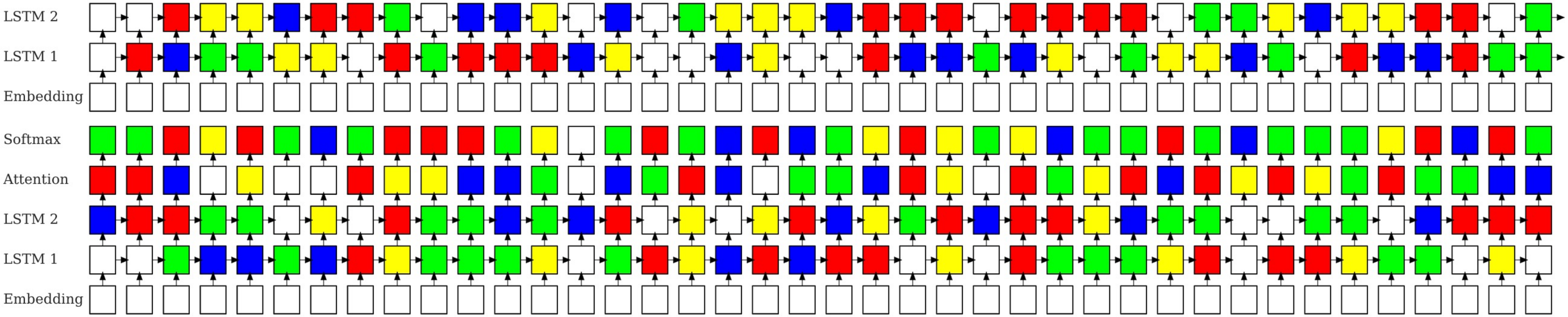
Evaluation Placements

- Compare RL-based placements against placements from other baseline methods:
  - **Single-CPU:** Execute NN on 1 CPU
  - **Single-GPU:** Execute NN on 1 GPU, or CPU if no GPU is available
  - **Scotch** (Pellegrini, 2009)
  - **MinCut:** Same as Scotch, but only use GPUs when possible
  - **Expert-designed:** Use model-specific placements suggested as optimal in the literature

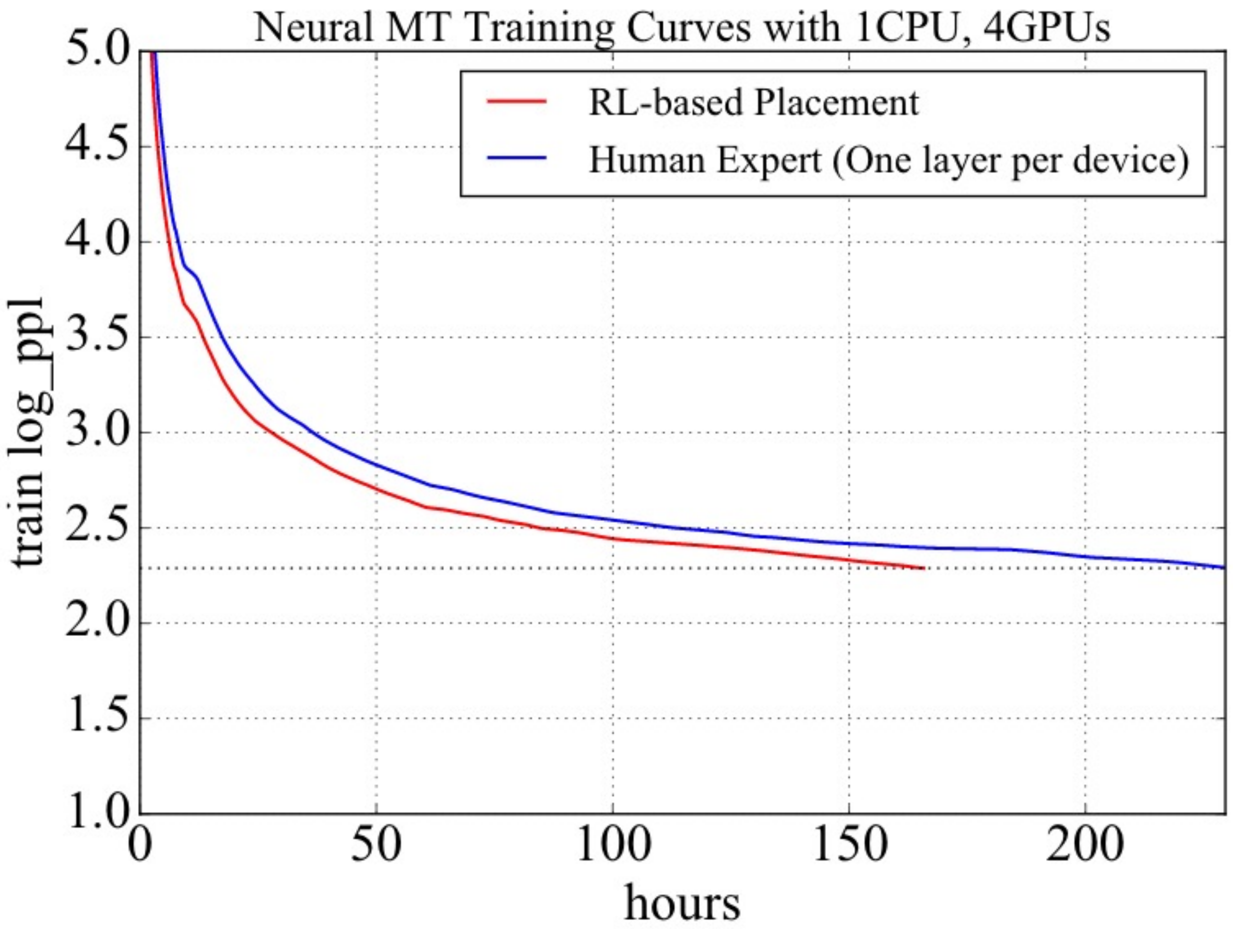
Observed Speedup

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%

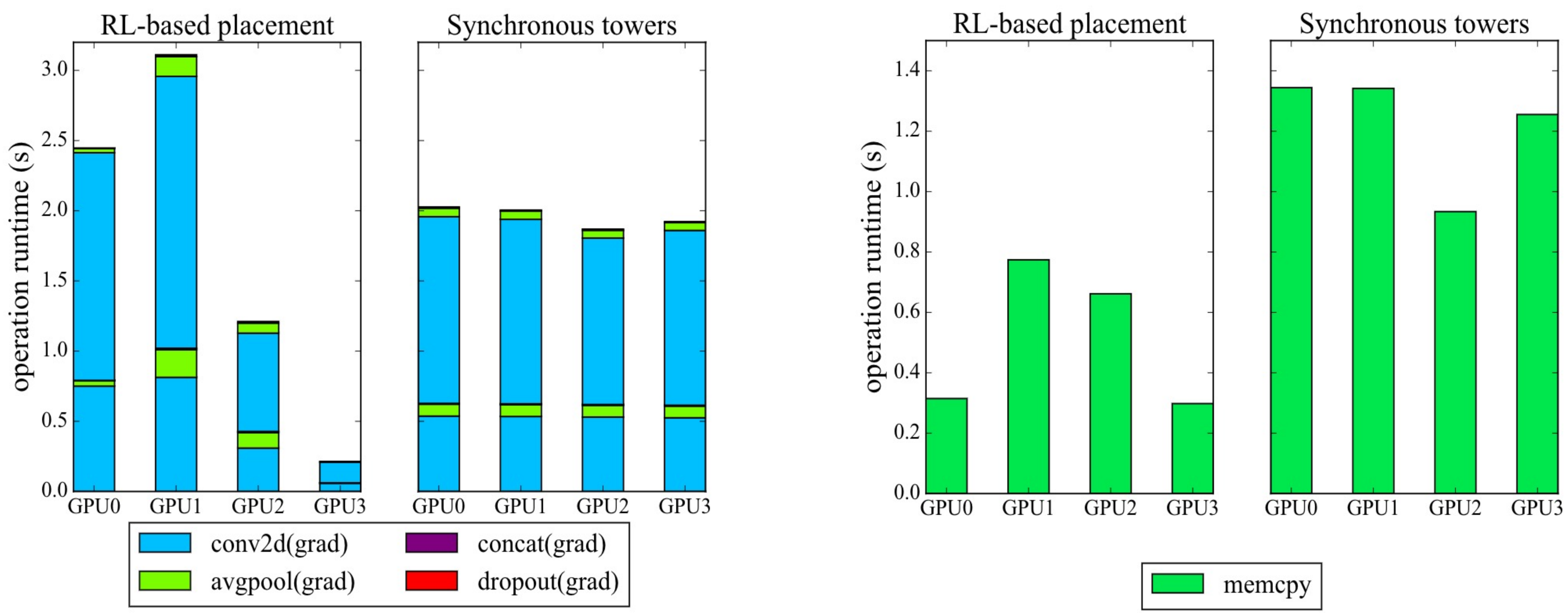
Example Placement for NMT



Example Training Curve for NMT



Profiling Placements



CONCLUSION

Contributions and Advancements

- A *new* RL-based model which discovers optimal device placements
  - Learns to balance the tradeoff between parallelism and inter-device communication
  - Learns non-trivial optimal device placements that are unlikely to be manually crafted
  - Finds optimal placement for minimal training and inference execution time in distributed environment
- Formalization of device placement optimization as a RL problem
  - Objective function with matching sample-based gradient update
  - Sequence-to-sequence model for flexible policy representation
- Empirical evaluation to illustrate optimality of discovered placements

Future Work

- Extend existing model to balance model parallelism *and* data parallelism, and explore newer hybrid parallelism approaches such as pipeline parallelism
- Generalize the model to work with ML frameworks other than TensorFlow
- Improved explanation for the policy network, particularly to expand upon the actual effect and capabilities gained by using Seq2Seq / LSTM / Attention
- Further mathematical discussion of RL portion of the model, and comparison with other existing deep-RL models

References

- Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, 2430-2439, 2017.
- Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR)*, 53(1):1-37, 2020.

UNIVERSITY OF WATERLOO

FACULTY OF MATHEMATICS

Thank you for listening!

Contact: joel.rorseth@uwaterloo.ca