(Hierarchical) Device Placement Optimization with RL

How RL doubled the speed of running TensorFlow graph on multiple GPUs.

Two papers

Device Placement Optimization with Reinforcement Learning

Azalia Mirhoseini [*12] Hieu Pham [*12] Quoc V. Le [1] Benoit Steiner [1] Rasmus Larsen [1] Yuefeng Zhou [1]
Naveen Kumar [3] Mohammad Norouzi [1] Samy Bengio [1] Jeff Dean [1]

Abstract

The past few years have witnessed a growth in size and computational requirements for training and inference with neural networks. Currently, a common approach to address these requirements is to use a heterogeneous distributed environment with a mixture of hardware devices such as CPUs and GPUs. Importantly, the decision of placing parts of the neural models on devices is often made by human experts based on simple heuristics and intuitions. In this paper, we propose a method which learns to optimize device placement for TensorFlow computational graphs. Key to our method is the use of a sequence-to-sequence model to predict which subsets of operations in a TensorFlow graph should run on which of the available devices. The execution time of the predicted placements is then used as the reward signal to optimize the parameters of the sequence-to-sequence model. Our main result is that on Inception-V3 for ImageNet classification, and on RNN LSTM, for language modeling and neural machine translation, our model finds non-trivial device placements that outperform hand-crafted heuristics and traditional algorithmic methods.

Submitted as a conference paper to ICLR 2018

A HIERARCHICAL MODEL FOR DEVICE PLACEMENT

Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean
{azalia,agoldie,hyhieu,bsteiner,qvl,jeff}@google.com

ABSTRACT

We introduce a hierarchical model for efficient placement of computational graphs onto hardware devices, especially in heterogeneous environments with a mixture of CPUs, GPUs, and other computational devices. Our method learns to assign graph operations to groups and to allocate those groups to available devices. The grouping and device allocations are learned jointly. The proposed method is trained with policy gradient and requires no human intervention. Experiments with widely-used computer vision and natural language models show that our algorithm can find optimized, non-trivial placements for TensorFlow computational graphs with over 80,000 operations. In addition, our approach outperforms placements by human experts as well as a previous state-of-the-art placement method based on deep reinforcement learning. Our method achieves runtime reductions of up to 60.6% per training step when applied to models such as Neural Machine Translation.

Plan

1. A high-level overview
2. Initial RNN-based method (ColocRL)
3. Improved, hierarchical method
4. Experimental data
5. Experimental setup
6. Results

1. A high-level overview

- Multi-device NN experiments. Need to split computation graphs across CPUs/GPUs. Past: humans/heuristics. Now: RL.
- Initial method involves the RNN-Attention model
- Improvement – hierarchical model
- Optimization based on measurements of real-life runs
- Both models more successful than most prior approaches

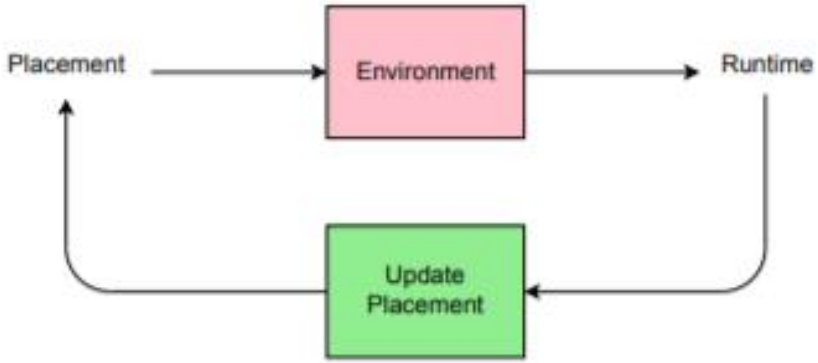


Figure 1. An overview of the RL based device placement model.

Figure from Mirhoseini et al. 2017

1.1. Goal

Consider a TensorFlow computational graph $\mathcal{G}$, which consists of M operations $\{o_1, o_2, ..., o_M\}$, and a list of D available devices. A placement $\mathcal{P} = \{p_1, p_2, ..., p_M\}$ is an assignment of an operation $o_i \in \mathcal{G}$ to a device p_i, where $p_i \in \{1, ..., D\}$. Let $r(\mathcal{P})$ denote the time that it takes to perform a complete execution of $\mathcal{G}$ under the placement $\mathcal{P}$. The goal of *device placement optimization* is to find $\mathcal{P}$ such that the execution time $r(\mathcal{P})$ is minimized.

Definition from Mirhoseini et al. 2017

2. RNN-based method (ColocRL)

- Input: list of graph operations, e.g. matmul, conv2d, pool2d. Output: matching of graph operations to devices.
- Encoder-Decoder architecture.
- Encoding of a graph operation type and output shapes is fed to the attentional LSTM.
- Decoder: Attentional LSTM selects a device for each operation. Encoding of a selected device then fed to LSTM.

2.1 Architecture of the ColocRL

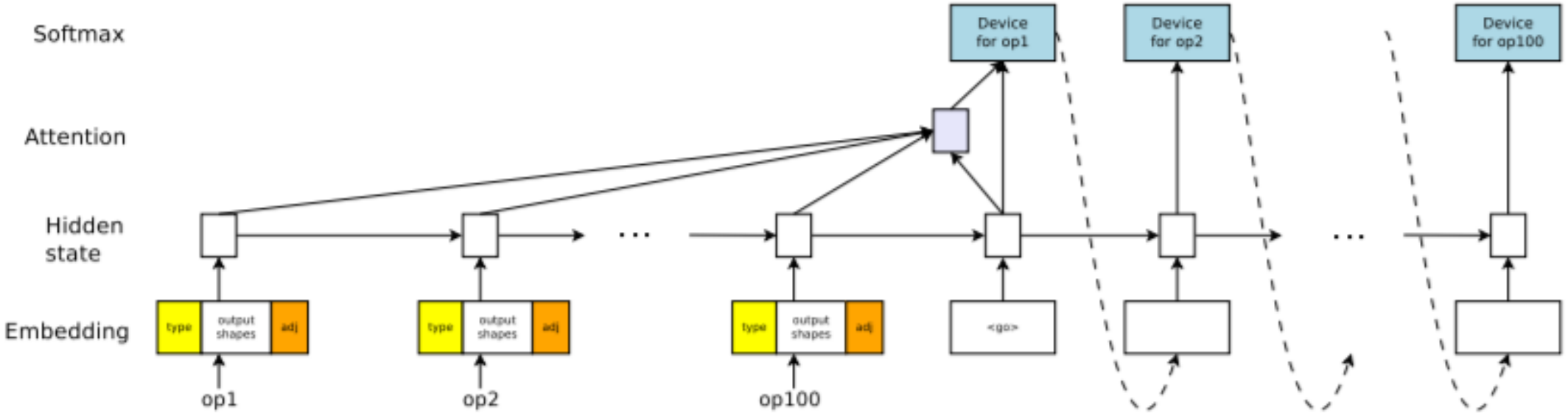


Figure 2. Device placement model architecture.

Figure from Mirhoseini et al. 2017

2.2 Co-location groups

- Co-locating operations: some ops batched together
  - Outputs with gradients
  - Only consumer of some layer's outputs, with the producer
  - Convolutions with pooling
  - LSTM group elements
- Aim: reduce dimensionality of search space.
  - RNNLM: from ~9k to 188
  - NMT: from ~22k to 280
  - Inception-v3: from ~30k to 83
- Co-location is a hard assumption and significant constraint!

Plan

1. A high-level overview
2. Initial RNN-based method (ColocRL)
3. **Improved, hierarchical method**
4. Experimental data
5. Experimental setup
6. Results

3. An improved, hierarchical model

- Two sub-networks, trained jointly:
  - Grouper:
    - Splits operations into groups
    - Performs embedding on each group, pushes embeddings to Placer
    - Implemented as a simple feedforward network with a softmax layer
  - Placer
    - Assigns groups to devices
    - Implemented as a Seq2Seq model with LSTM and Attention.

3.1. Illustration of a hierarchical model

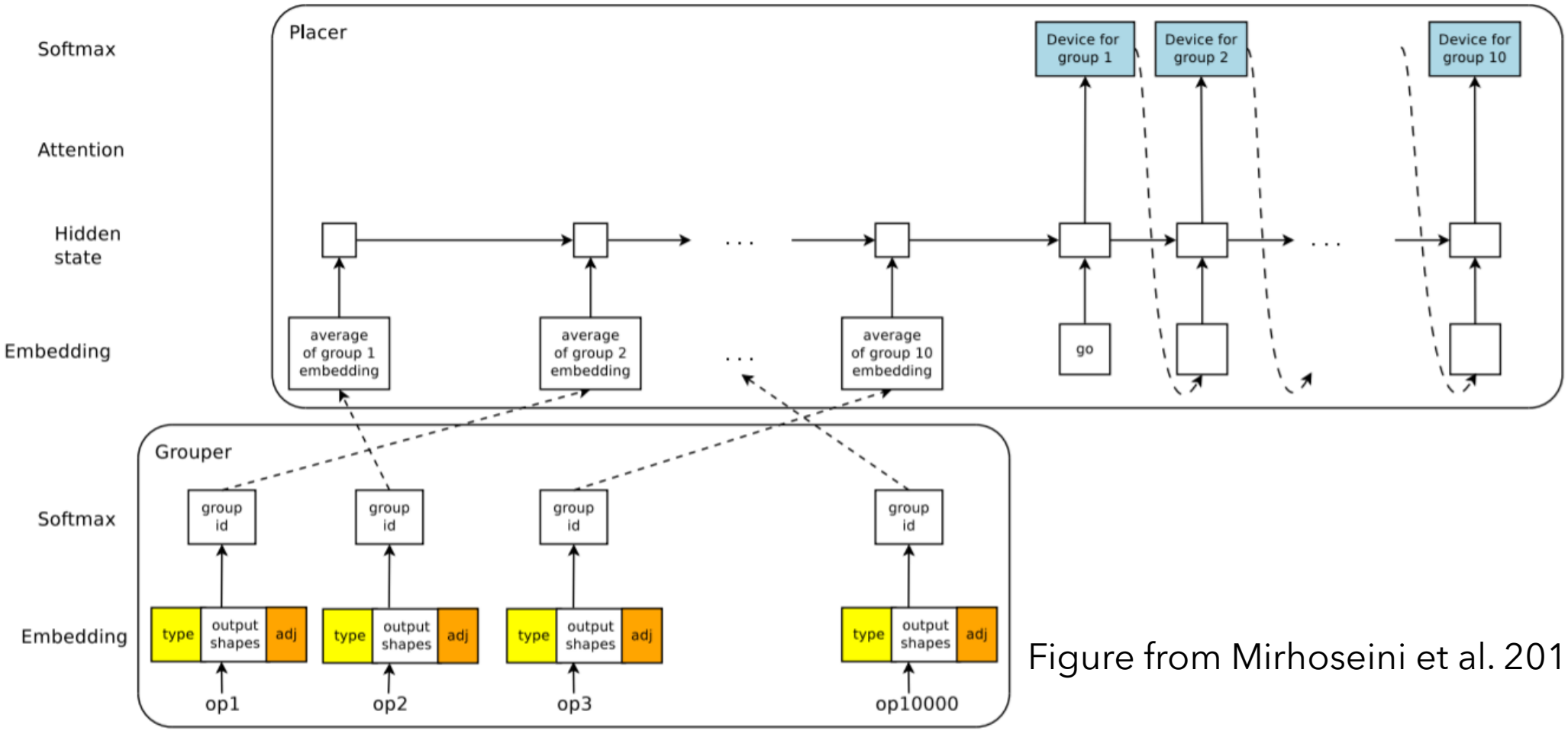


Figure from Mirhoseini et al. 2018

Plan

1. A high-level overview
2. Initial RNN-based method (ColocRL)
3. Improved, hierarchical method
4. **Experimental data**
5. Experimental setup
6. Results

4. Experimental data

- Recurrent Neural Network Language Model (RNNLT)
  - Many LSTM models
  - Grid-based architecture
- Neural Machine Translation with attention (NMT)
  - Many hidden states
  - Originally, LSTM, Attention and Softmax separate
- Inception-V3 (used for computer vision)
  - Block-based architecture
  - Block comprises convolutional and pooling layers
- ResNet (used for computer vision, only benchmarked with Hierarchical)

5. Experimental Setup

- RL algorithm: REINFORCE (a vanilla policy gradient method)
- Optimizer: Adam / RMSProp
- Devices:
  - ColocRL: Intel Haswell 2300 CPU + 1-8 Nvidia Tesla K80 GPU
  - Hierarchical: Intel Haswell 2300 CPU + 1-8 Nvidia Tesla K40 GPU
- Tensorflow versions different

5. Distributed & asynchronous training

- Up to 20 controllers
- 4-8 workers per controller
- Training takes 12-27h

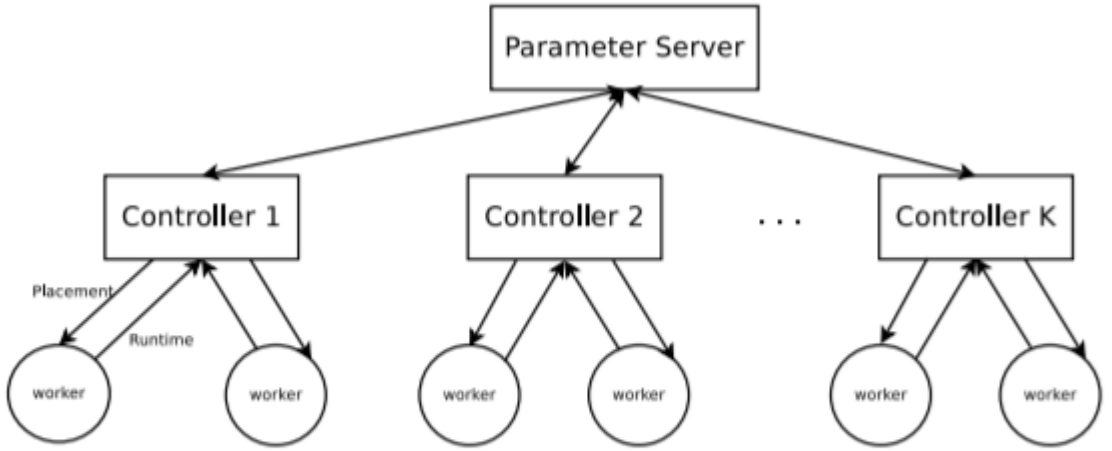


Figure 3. Distributed and asynchronous parameter update and reward evaluation.

Figure from Mirhoseini et al. 2017

6. Benchmarks

- Single-CPU
- Single-GPU.
  - If operation has no GPU implementation, run on CPU
- Scotch – a 2009 heuristic-based graph execution optimizer
- MinCut – Scotch without CPU.
  - No GPU-based op – use CPU.
- Expert design

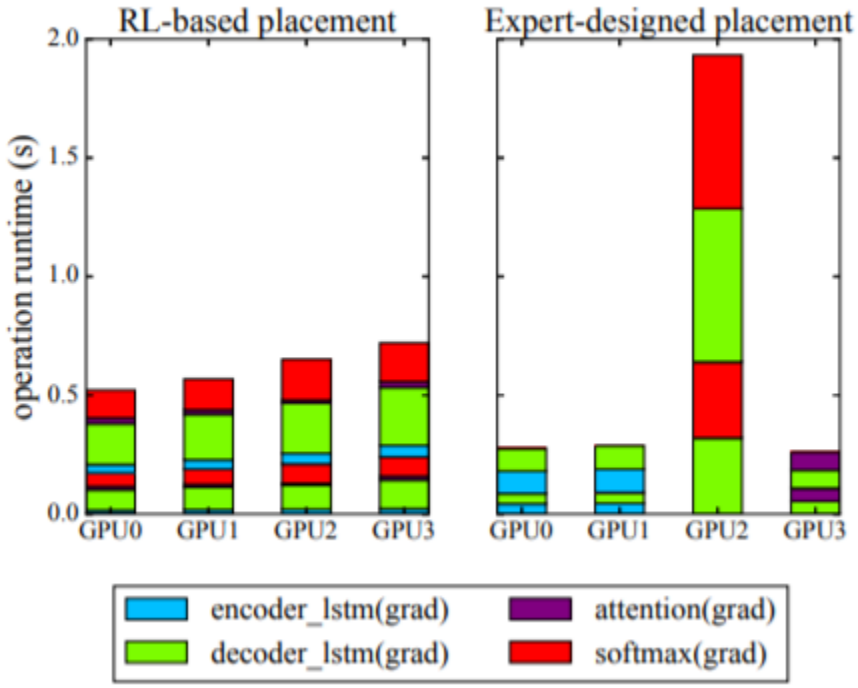
6. Results

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%

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%

Figures from Mirhoseini et al. 2017 (top), Mirhoseini et al. 2018 (bottom)

6. Placements in ColocRL



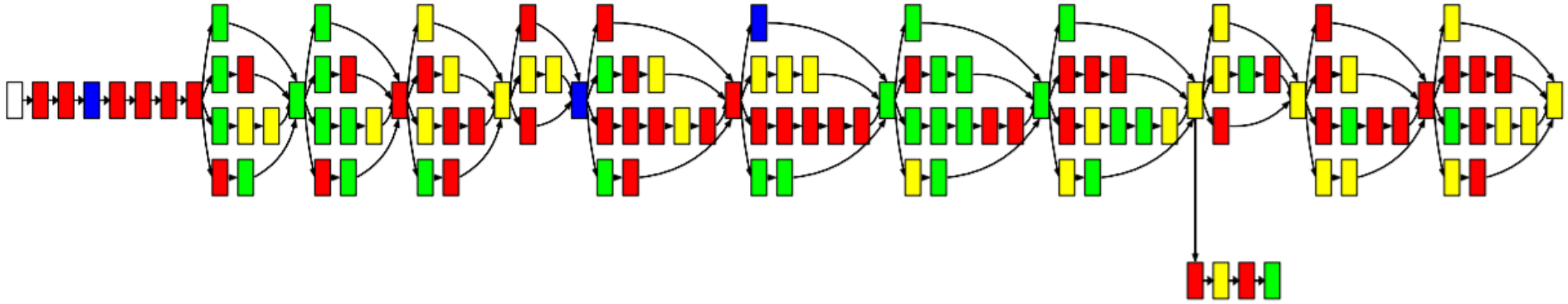


Figure 5. RL-based placement of Inception-V3. Devices are denoted by colors, where the transparent color represents an operation on a CPU and each other unique color represents a different GPU. RL-based placement achieves the improvement of 19.7% in running time compared to expert-designed placement.

Figures from Mirhoseini et al. 2017

6. Placements in Hierarchical

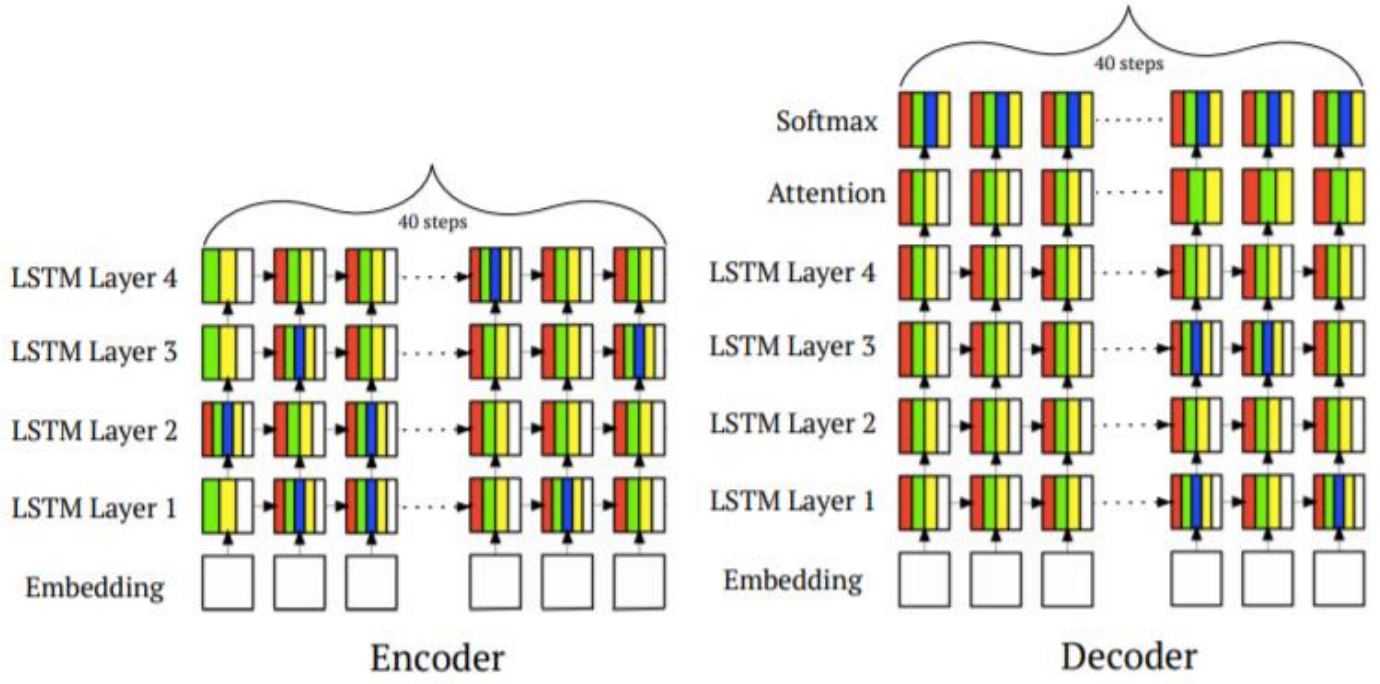


Figure 2: The Hierarchical Planner's placement of a NMT (4-layer) model. White denotes CPU and the four colors each represent one of the GPUs. Note that every step of every layer is allocated across multiple GPUs. This placement is 53.7% faster than that generated by a human expert.

Figure from Mirhoseini et al. 2018

Summary

- Both ColocRL and Hierarchical substantially improve on prior work in device placement
- Effective device placement requires prior grouping of operations
- In ColocRL grouping is handpicked, in Hierarchical automated
- Both ColocRL and Hierarchical introduce novel ideas
- Hierarchical seems to be faster, but there’s no direct benchmark

Credits + Q&A

- Azalia Mirhoseini*, Hieu Pham*, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean: Device Placement Optimization with Reinforcement Learning (2017)
- Azalia Mirhoseini*, Anna Goldie* , Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean: A Hierarchical Model for Device Placement (2018)
- Presented by F. Budrowski, for the 2020 R244 Cambridge class