Device Placement Optimization with Reinforcement Learning

Google Brain

ICML 2017

Background

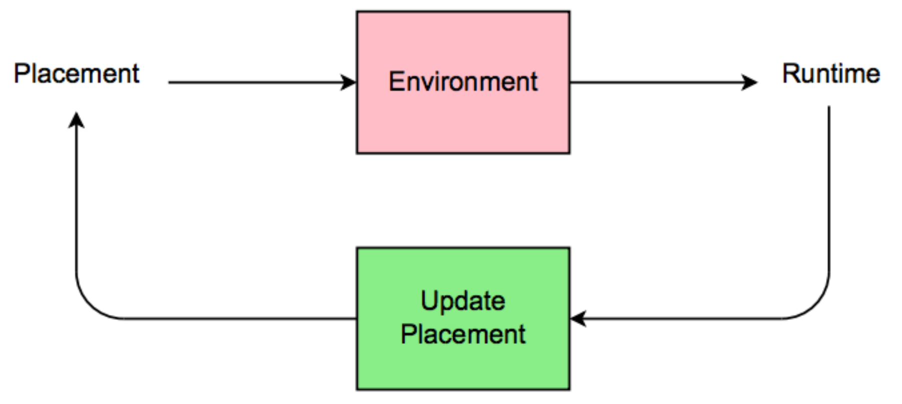
- Neural networks
  - Image classification, speech recognition, machine translation
- Training and inference with neural networks
  - A growth in size and computational requirements
- Heterogeneous distributed environment
  - A mixture of hardware devices, CPUs and GPUs
- Placing certain operations of the neural models on devices

Challenges

- When the network has many branches
- When the mini batches get larger
- Dynamic environment with many interferences

Overview

- Which parts of the model should be placed on which device
- How to arrange the computations to optimize the communication
- Reward signal: execution time



Method

- A TensorFlow computational graph G
- M operations $\{o_1, o_2, \ldots, o_M\}$
- D available devices
- Placement $P = \{p_1, p_2, \ldots, p_M\}$ is an assignment of an operation o_i to a device p_i
- $r(P)$ is the time to perform a complete execution of G under the placement P
- Goal: minimize the execution time $r(P)$

Method

- Noisy measurements of $r(P)$ in the beginning of the training process
  - The bad placements sampled
  - Inappropriate learning signals
- RL model gradually converges
  - The sampled placements become more similar to each other
  - Less distinguishable training signals
- Empirically use the square root of running time $R(p) = \sqrt{r(P)}$

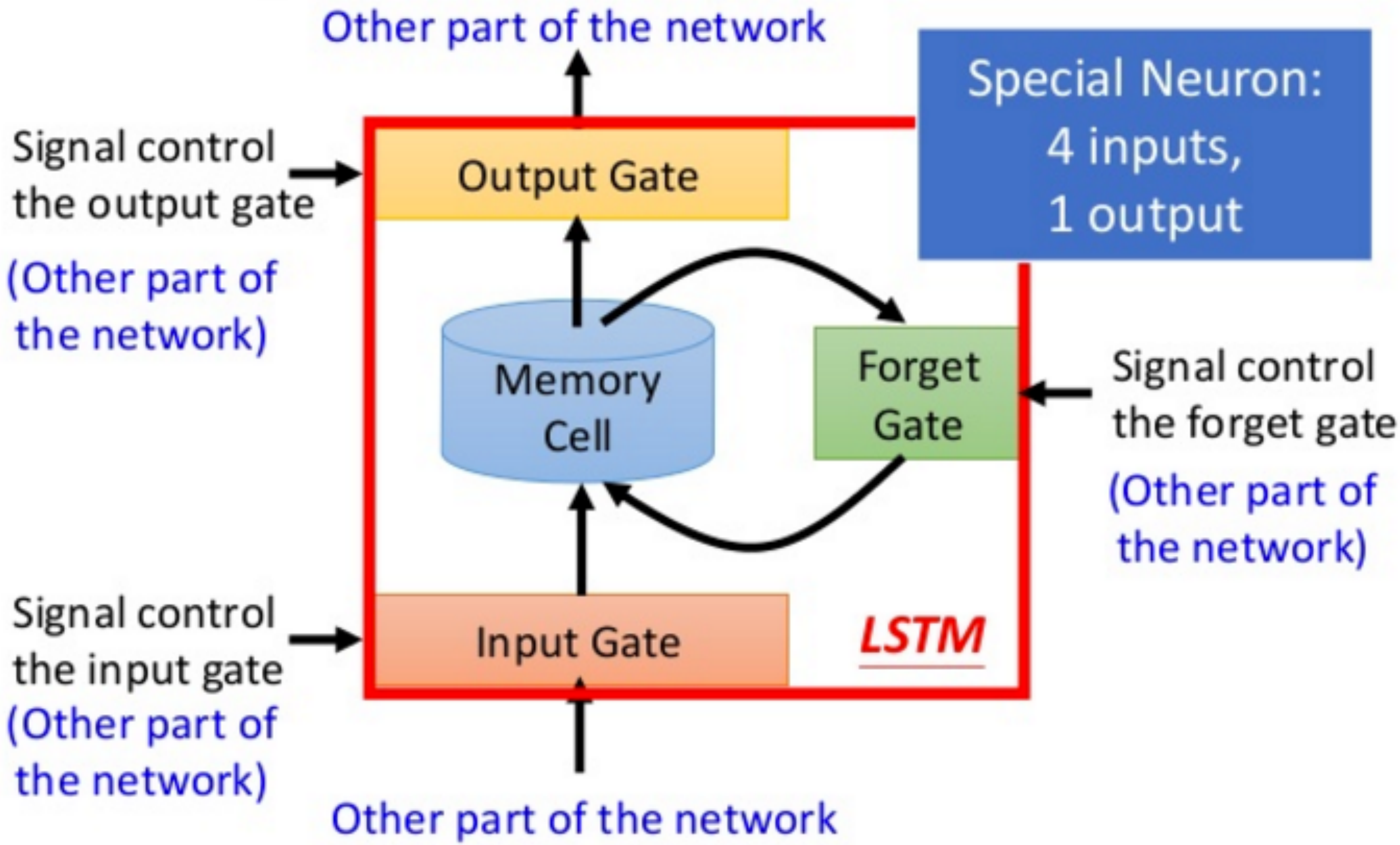
Method

- Train a stochastic policy $\pi(P|G;\theta)$ to minimize the objective $J(\theta) = E_{P\sim\pi(P|G;\theta)}[R(P)|G]$
- Policy gradient
  - Optimize parametrized policies with respect to the expected return by gradient descent
  - Learn network parameters
- $\nabla_\theta J(\theta) = E_{P\sim\pi(P|G;\theta)}[R(P) \cdot \nabla_\theta \log_p(P|G;\theta)]$
- Estimate $\nabla_\theta J(\theta)$ by drawing some placement samples
- $\nabla_\theta J(\theta) = \frac{1}{K}\sum_{i=1}^{K}(R(P_i) - B) \cdot \nabla_\theta \log_p(P_i|G;\theta)$

LSTM (Long Short Term Memory)

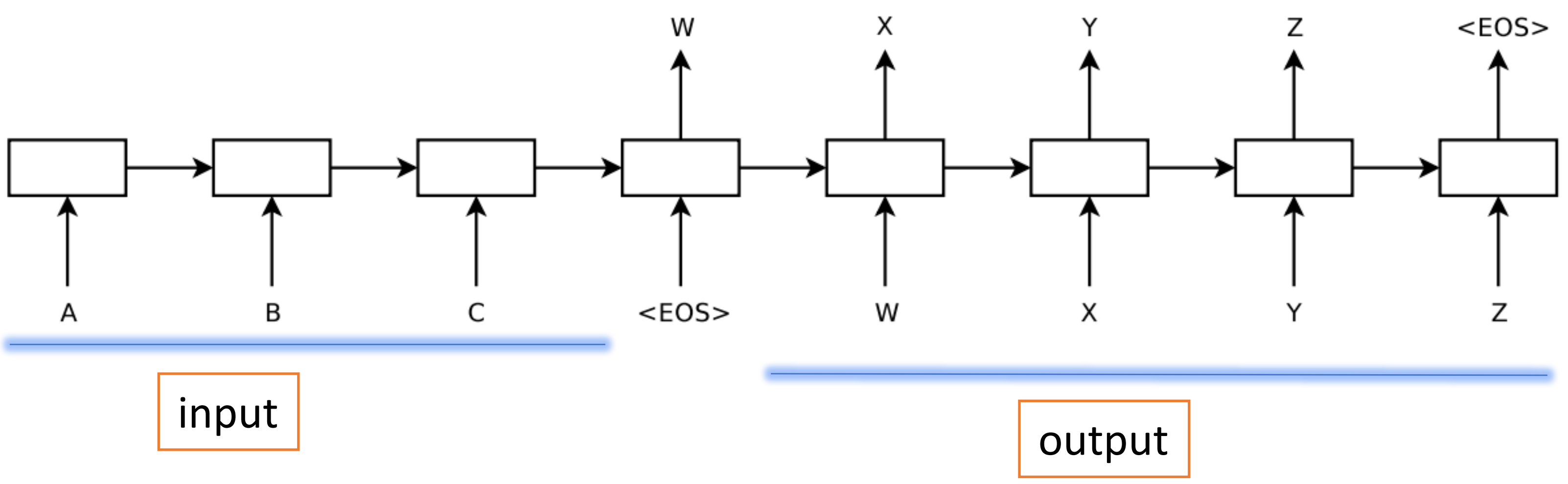
- Learn long-term dependencies
- What information to throw away from the cell state
- What information to store in the cell state
  - Which value to update
  - Create a vector of new candidates values
- Decide what information to output
  - A filtered version based on cell state

Long Short-term Memory (LSTM)



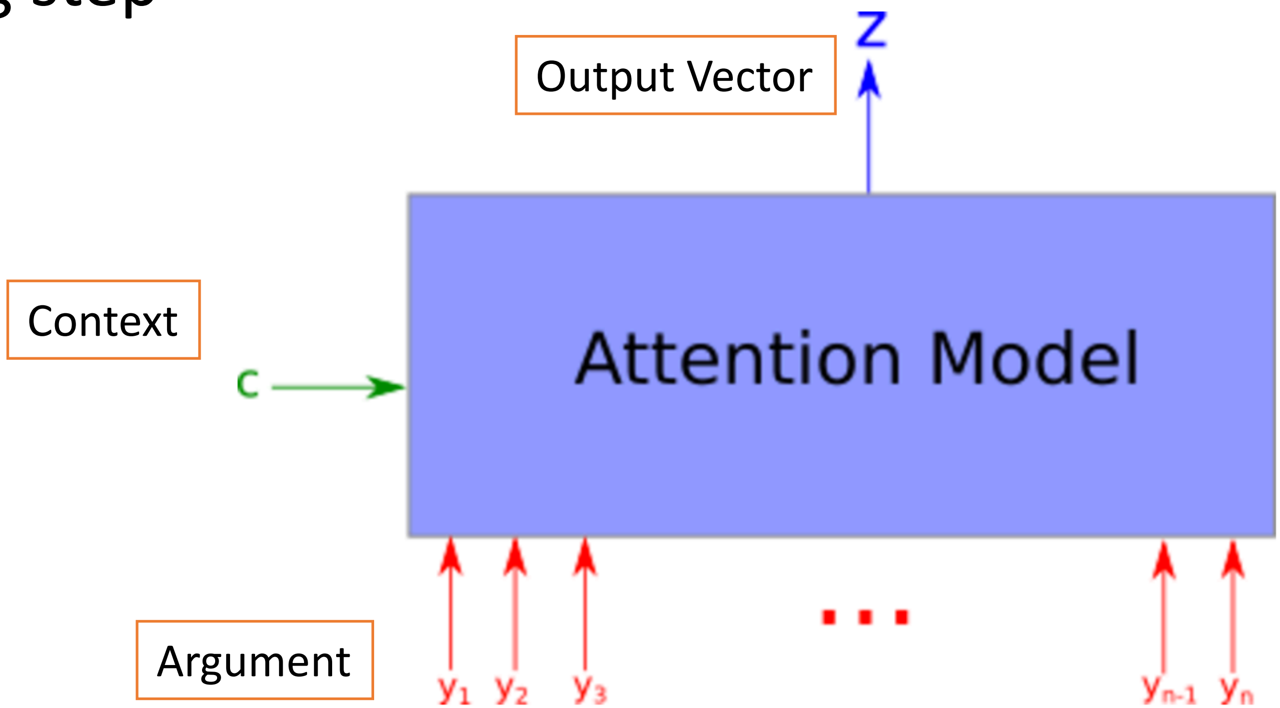
Sequence-to-sequence model

- Deep neural networks require the dimensionality of the inputs and outputs is known and fixed
- Can not map sequences to sequences
- Encoder: process the input
- Decoder: generate the output

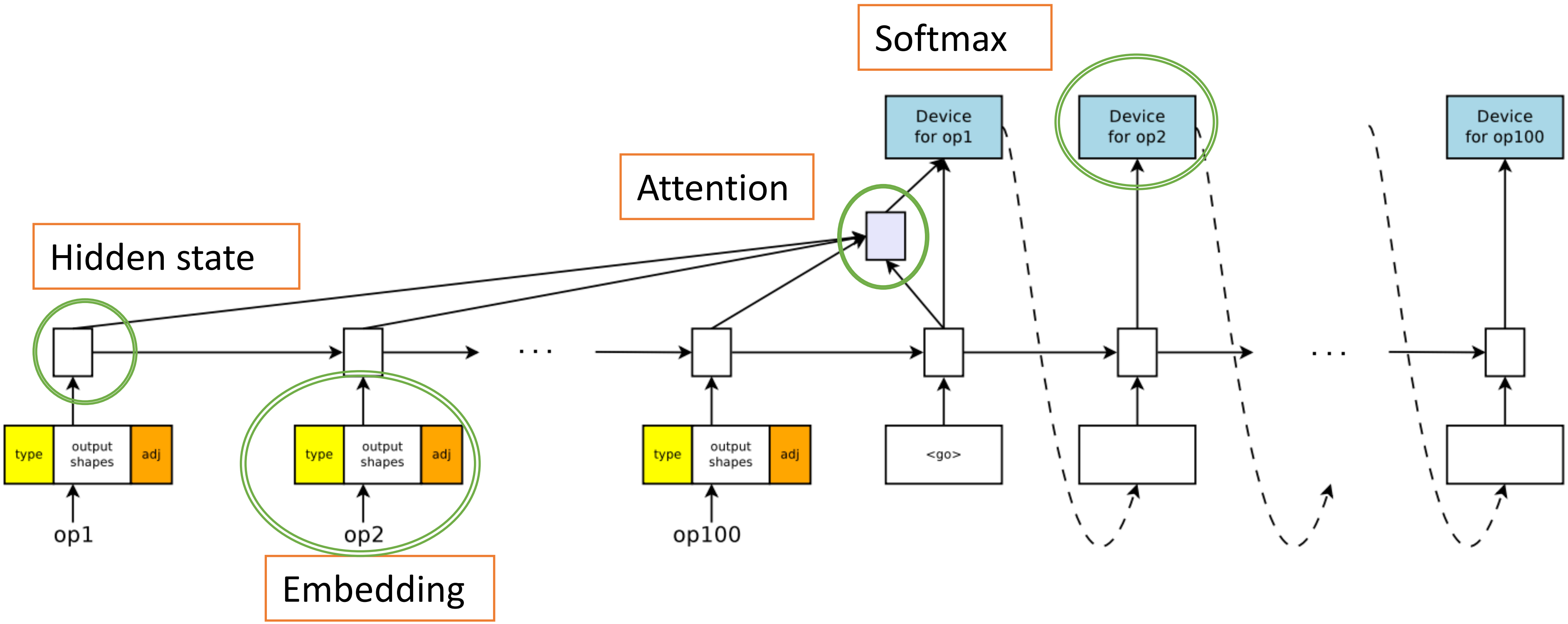


Content-based attention mechanism

- Allow the decoder more direct access to the input
- Allow the decoder to peek into the input at every decoding step



Architecture Details



Architecture Details

- Input:
- The sequence of operations of the input graph
- Encoder:
- Collect the types of its operations
- Concatenate the size of each operation's list into a fixed-sized, zero-padded list, i.e., output shape
- Take the one-hot encoding vector that represents the adjacency information

Architecture Details

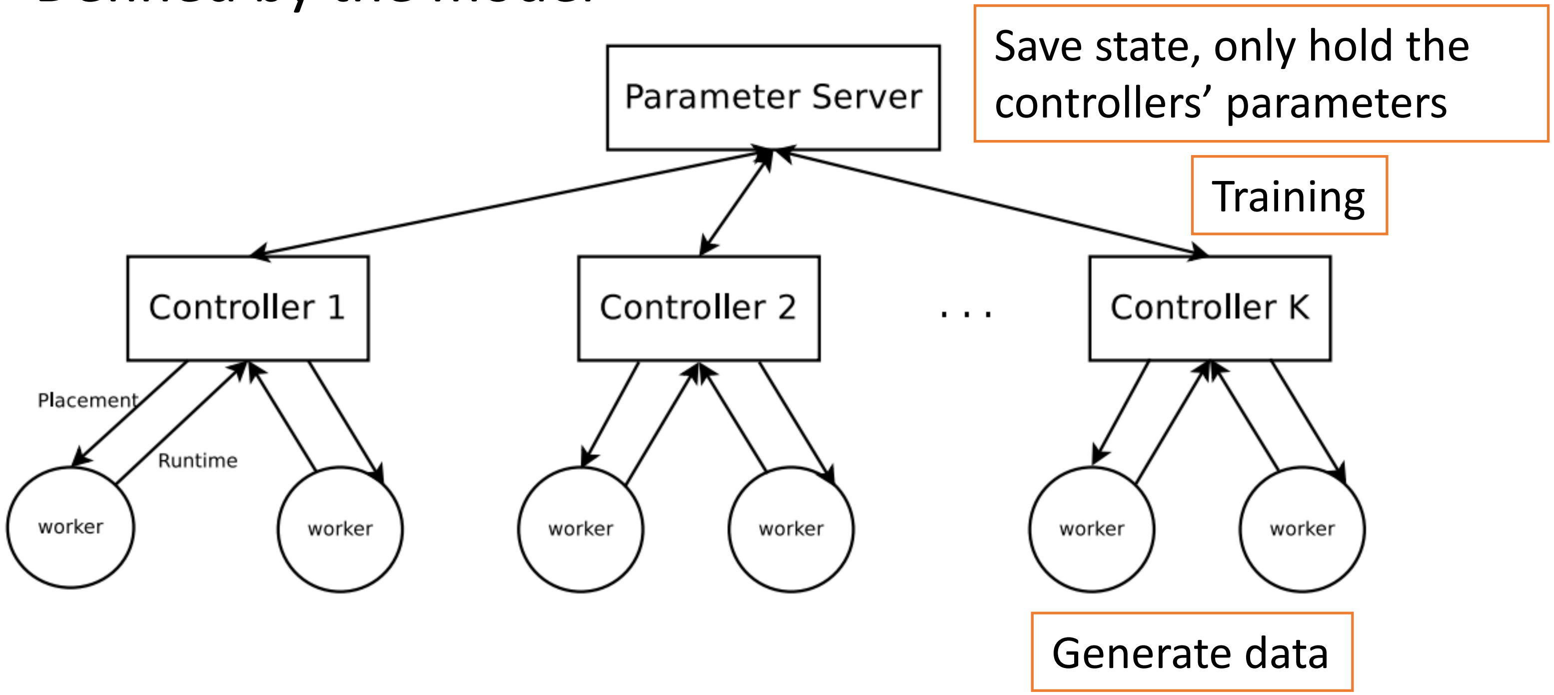
- Decoder:
- An attentional LSTM
- A fixed number of steps
  - Equal to the number of operations
- Output the device of the operator at the same encoder time step

Co-locating Operations

- TensorFlow computational graphs have thousands of operations
- Reduce the number of objects to place on different devices
  - Manually forcing several operations to be located on the same device
- E.g., the output of an operation X is consumed only by another operation Y $\rightarrow$ operations X and Y are co-located
- E.g., initialization operations

Distributed Training

- Asynchronous distributed training
- Each controller execute the current policy
  - Defined by the model



First Phase

- Each worker receives a signal
  - Indicate that it should wait for placements from its controller
- Each controller receives a signal
  - Indicate it should sample K placements
- Each controller then independently sends the placements to their workers

Second Phase

- Each worker executes the placement and measures the running time
- Each controller waits for all of its workers
  - Finish executing their assigned placements
  - Return their running times
- The controller scales the gradients to asynchronously update the controller parameters

Benchmarks

- Recurrent Neural Network Language Model
  - Grid structure
  - Parallel executions
- Neural Machine Translation with attention mechanism
  - large number of hidden states due to the source and target sentences
  - Require model parallelism
- Inception-V3
  - Image recognition and visual feature extraction
  - Sequential processing

Baselines

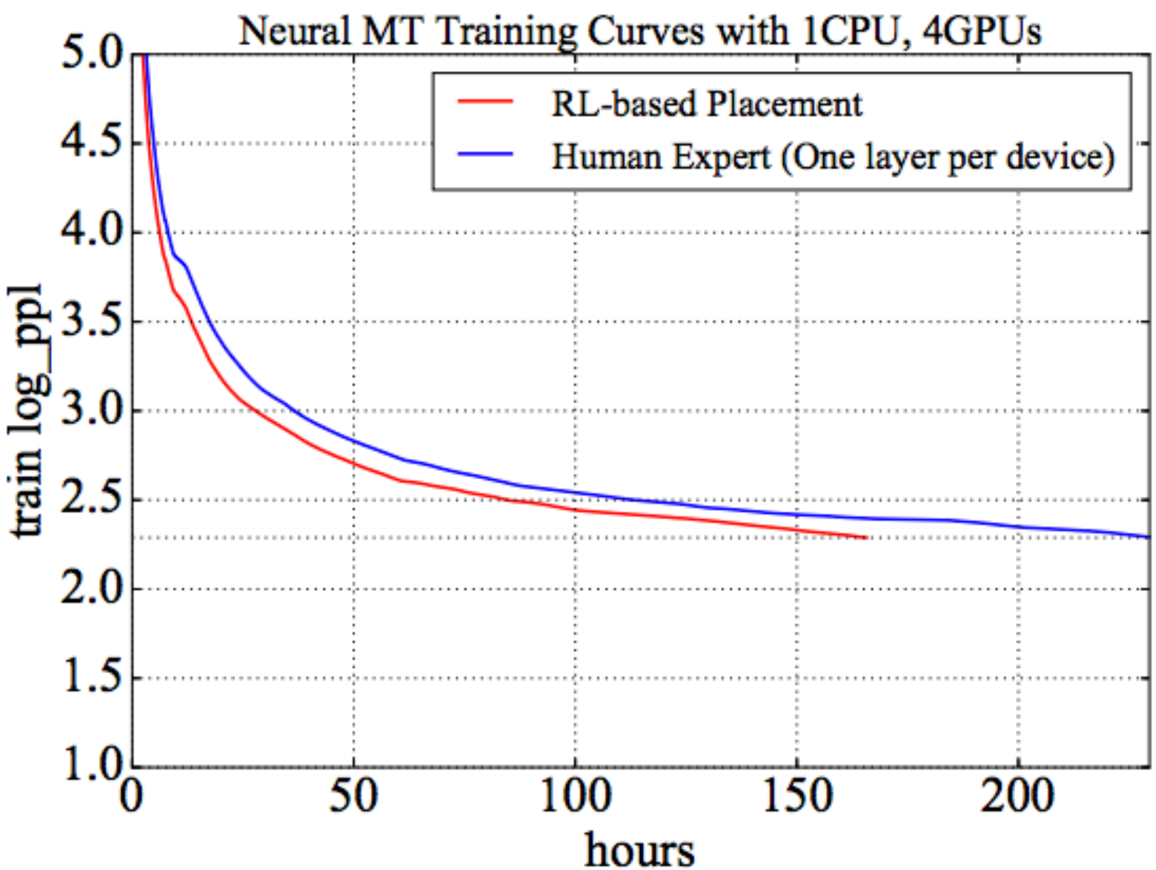
- Single-CPU
- Single-GPU
- Scotch
    - Estimate the computational costs, amount of flow data
- MinCut
    - Run on GPU if no CPU implementation
- Expert-designed
    - Put each layer on a device
    - Partition the model into contiguous parts with roughly the same number of layers

Single-Step Runtime Efficiency

- Only information
  - Running times of the placement
  - The number of available devices
- Learn subtle tradeoffs
  - Performance gain by parallelism
  - The costs induced by inter-device communications

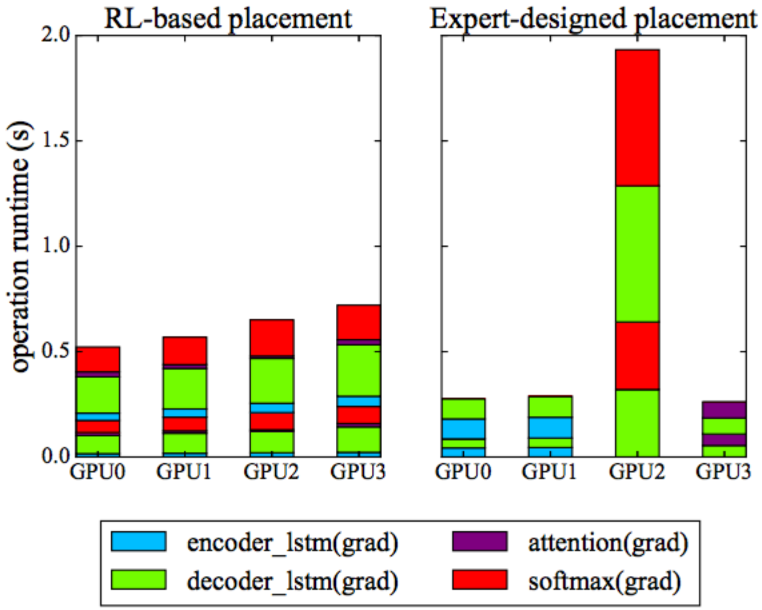
Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**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

End-to-End Runtime Efficiency



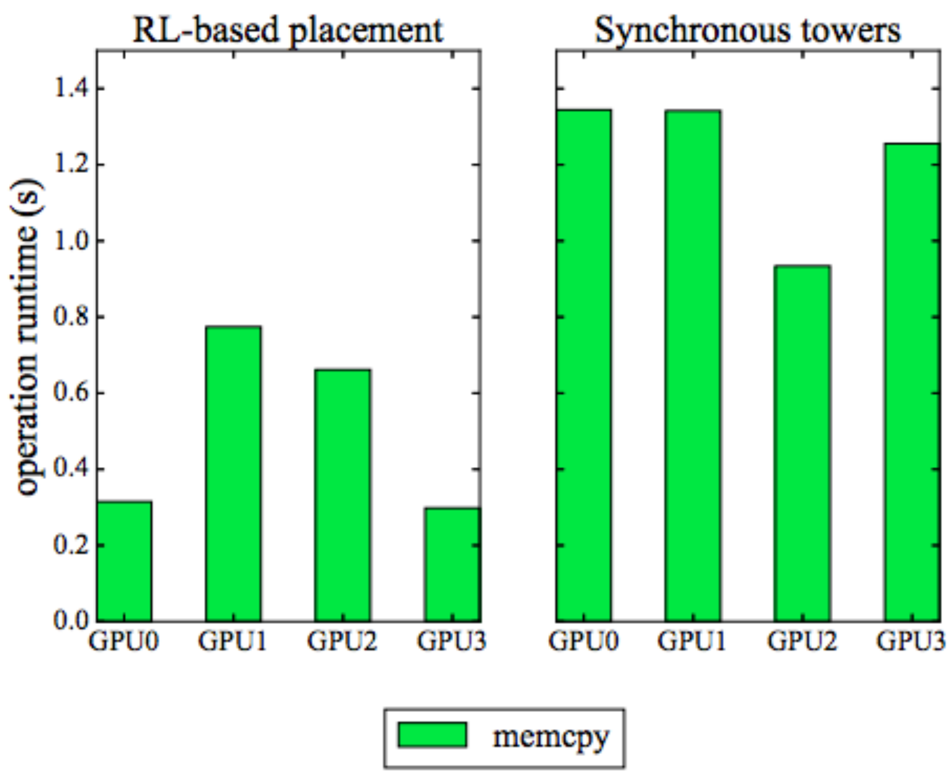
Analysis of Found Placements

- Per-device computational load
  - Load-balancing



Analysis of Found Placements

- Memory copy time



Summary

- Weakness
  - No data parallelism
  - Overhead (hundreds of machines, one day)
  - Not general
- Use reinforcement learning to optimize placement in the machine learning system
  - Combine with theory work (bandit)
- Model parallelism
  - Complementary to current work