Efficient Distributed RL: Actor-Learner Distillation

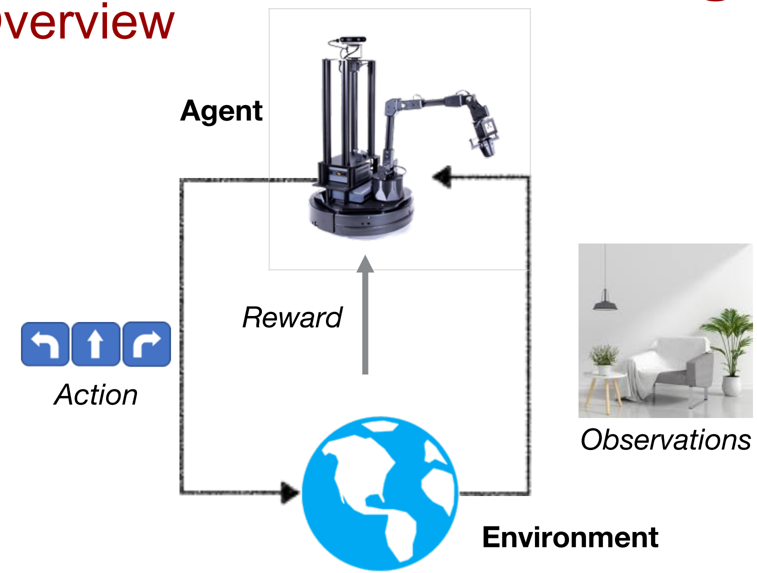
Deep Reinforcement Learning and Control

Fall 2021, CMU 10-703

Russ Salakhutdinov
(work with Emilio Parisotto)

Machine Learning Department
Carnegie Mellon University

Reinforcement Learning
Overview



Agent needs to move in the world physically.

Current actions affect future observations.

Require Spatial and Semantic Understanding.

- Agent interacts with environment.
- Predicts actions given observations **(policy)**.
- Receives *scalar feedback* **(reward)** from the environment.
- Interaction terminates given **H** actions:
  - Referred to as an **episode**.

Reinforcement Learning:
- Learn policy that maximizes episodic reward.

(On-Policy) Reinforcement Learning

- Agent formalized as policy π_θ.
  - Maps states to a distribution over actions.
  - Parameterized by θ.
- Sample from π_θ to collect episodes $(s_1, a_1, r_1, \ldots, s_H, a_H, r_H)$.

Want to maximize:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} r_t\right]$$

(On-Policy) Reinforcement Learning

- Agent formalized as policy π_θ.
  - Maps states to a distribution over actions.
  - Parameterized by θ.
- Sample from π_θ to collect episodes $(s_1, a_1, r_1, \ldots, s_H, a_H, r_H)$.

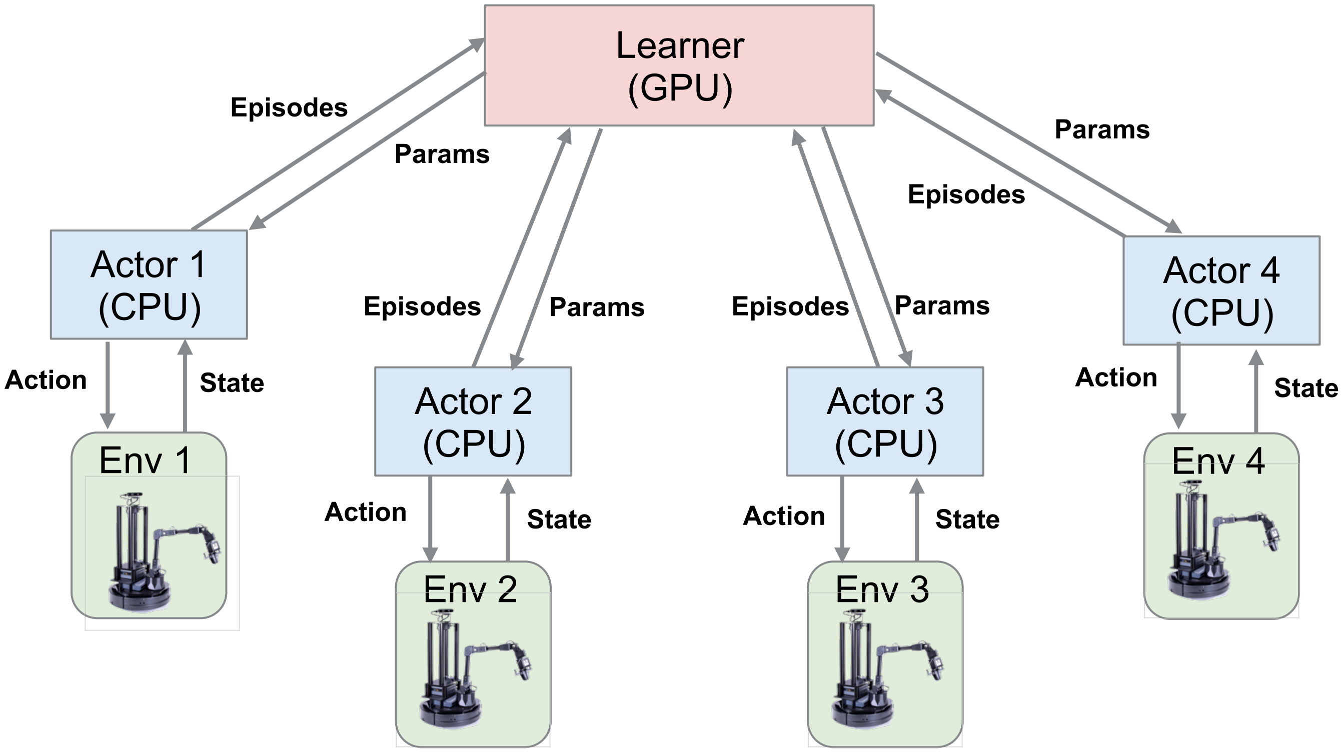
Want to maximize:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H} r_t\right]$$

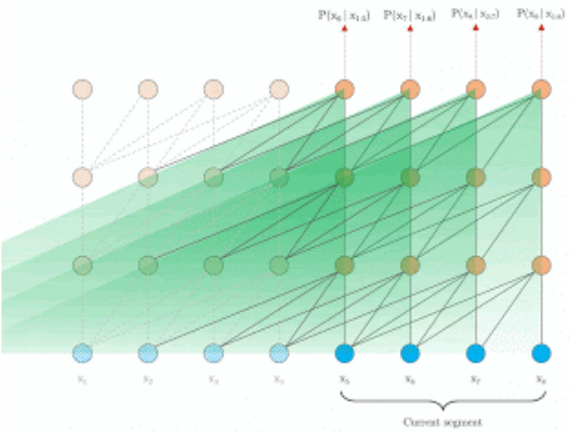
Update θ using Policy Gradient:

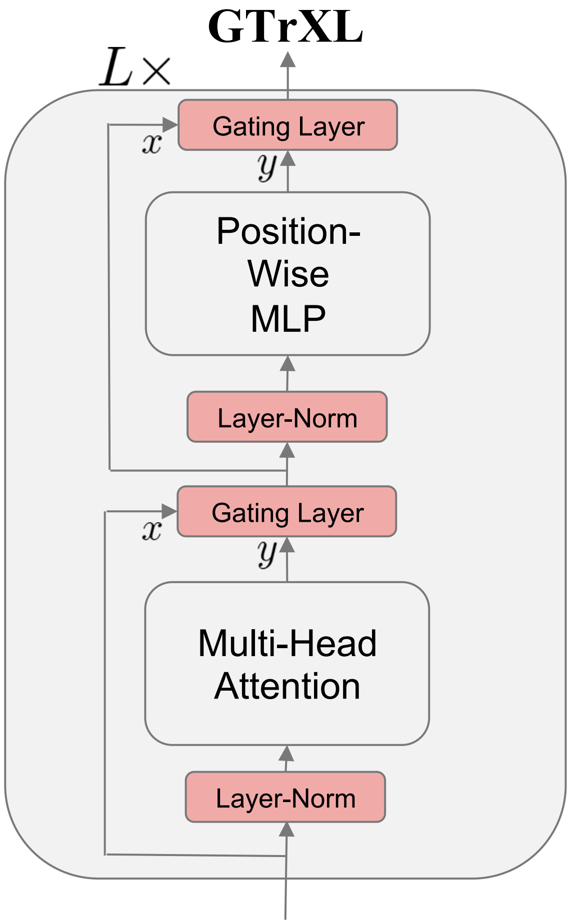
$$\frac{\partial J(\theta)}{\partial \theta} \approx \sum_{t=1}^{H} \log \pi_\theta(a_t|s_t)\mathcal{A}\left(\sum\nolimits_{k=t}^{H} r_t\right)$$

Distributed (On-Policy) RL



Larger Models Work Better
Gated Transformers

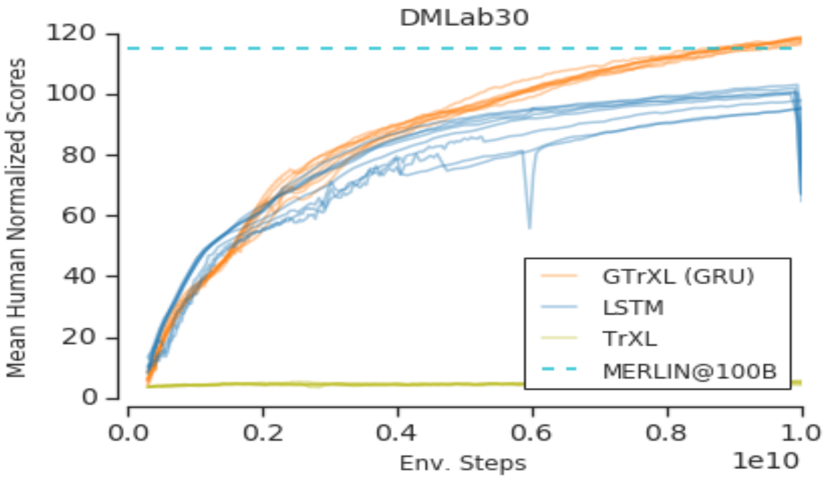


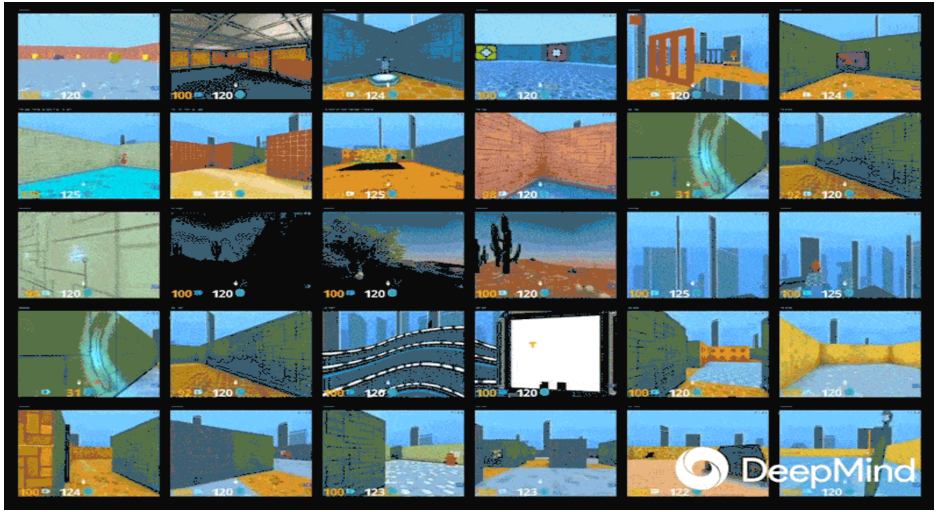


Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context (Dai et al., 2019)
Stabilizing Transformers for Reinforcement Learning (Parisotto et al., 2019)

Gated Transformers

State-of-the-Art in DMLab30

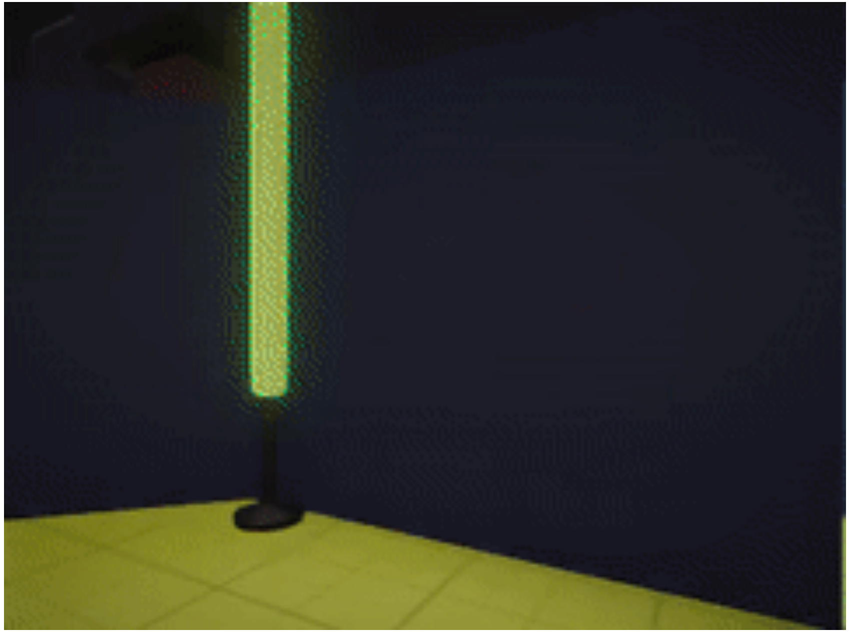


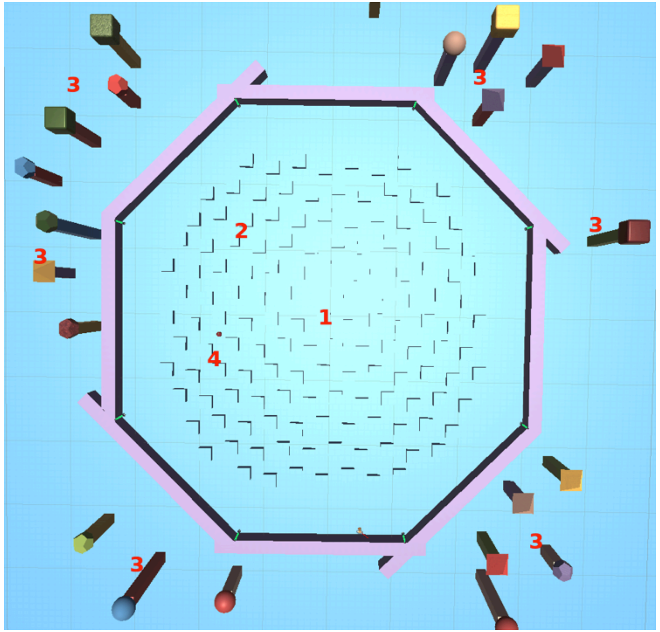


Stabilizing Transformers for Reinforcement Learning (Parisotto et al., 2019)
DeepMind Lab (Beattie et al., 2016)

Gated Transformers

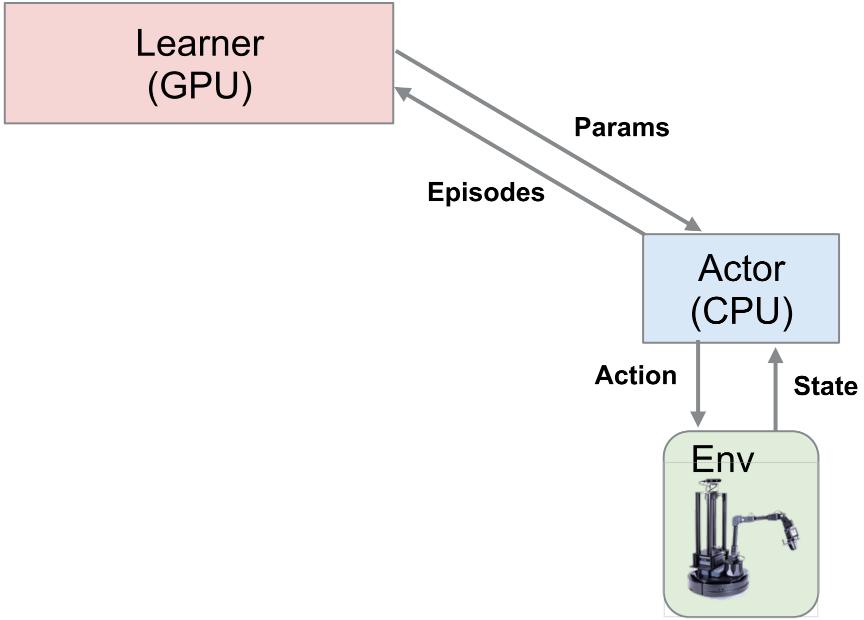
Rapid Memorization of Partially-Observed Environments





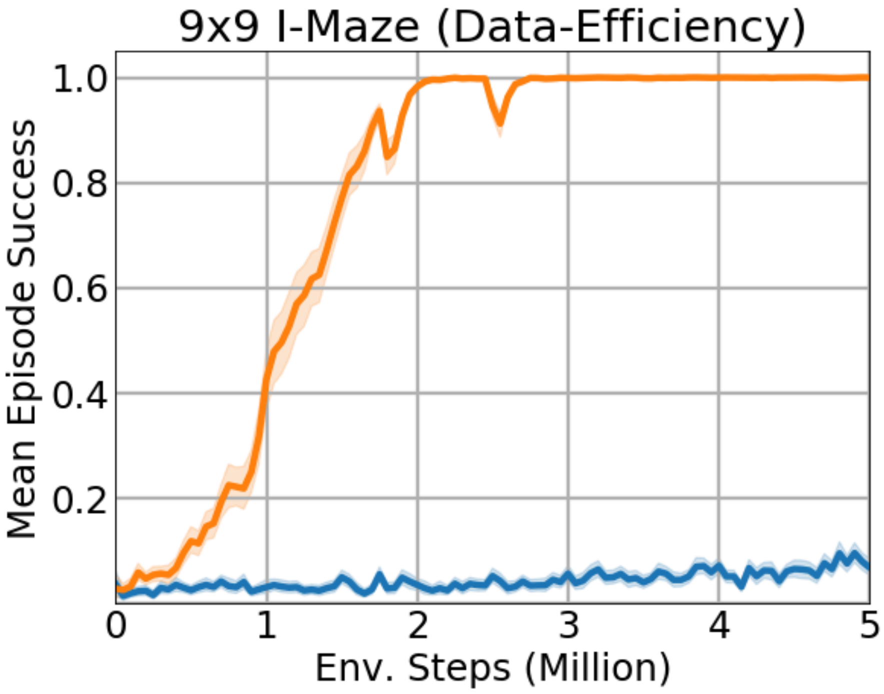
Stabilizing Transformers for Reinforcement Learning (Parisotto et al., 2019)

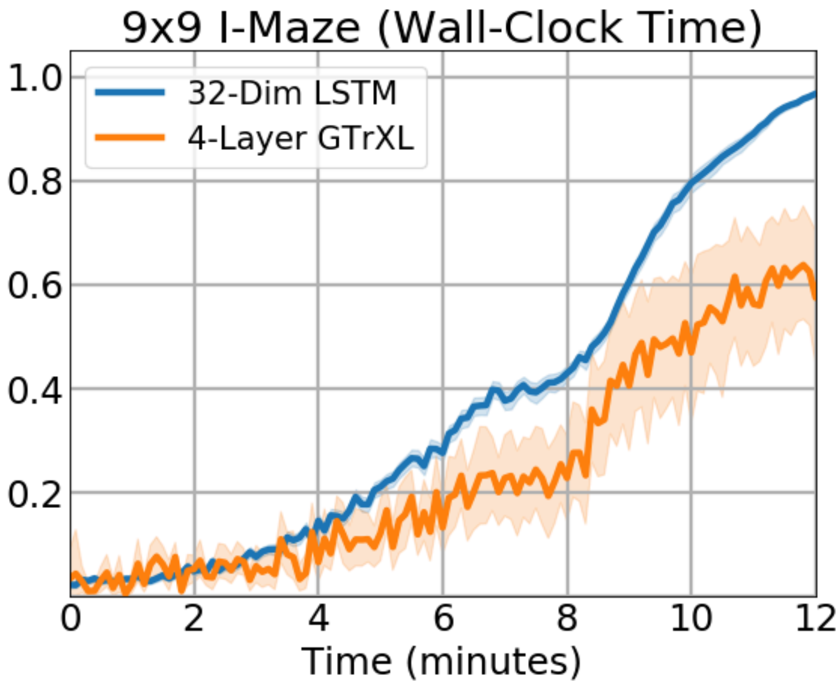
Actor-Latency is a Major Bottleneck



Model	% Idle for Traj.	Actor SPS
LSTM	34%	1053
GTrXL	84%	70

Motivation
Smaller Models More Time-Efficient?





Why Not Model Compression?

In supervised learning (with dataset $\mathcal{D}$):

$$J(\theta) = \mathbb{E}_{x \sim \mathcal{D}}[L(x; \theta)]$$

Do Deep Nets Really Need to be Deep? (Ba et al., 2013)
Distilling the Knowledge in a Neural Network (Hinton et al., 2015)

Why Not Model Compression?

In supervised learning (with dataset $\mathcal{D}$):

$$J(\theta) = \mathbb{E}_{x \sim \mathcal{D}}[L(x; \theta)]$$

In reinforcement learning:

$$J(\theta) = \mathbb{E}_{x \sim \pi_\theta}[L(x; \theta)]$$

Do Deep Nets Really Need to be Deep? (Ba et al., 2013)
Distilling the Knowledge in a Neural Network (Hinton et al., 2015)

Why Not Model Compression?

In supervised learning (with dataset $\mathcal{D}$):

$$J(\theta) = \mathbb{E}_{x \sim \mathcal{D}}[L(x;\theta)]$$

In reinforcement learning:

$$J(\theta) = \mathbb{E}_{x \sim \pi_\theta}[L(x;\theta)]$$

Depends on Current Parameters → Can't Learn Offline

Do Deep Nets Really Need to be Deep? (Ba et al., 2013)
Distilling the Knowledge in a Neural Network (Hinton et al., 2015)

Actor-Latency Constraints

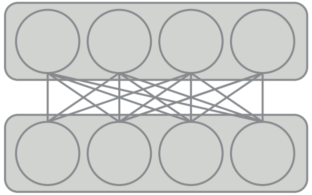
- Inference running on un-accelerated hardware:
  - CPUs, robotic platforms, mobile phones, etc.
  - Potential hard constraint on latency (robot acting)

- Learning running on accelerators.
  - GPU, TPU, large-scale CPU cluster, etc.

Goal: Leverage large model capacity while minimizing inference costs

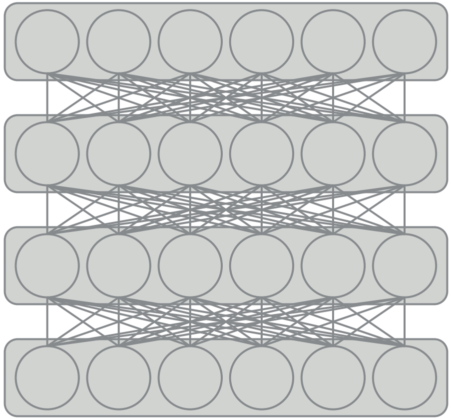
Idea: Separate Actor and Learner Models

Actor Model $\mathcal{M}_A$



- Low-Capacity
- Optimized for Inference Speed
- Sequential Execution (CPU)
- Generates Environment Episodes.

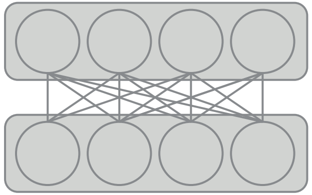
Learner Model $\mathcal{M}_L$



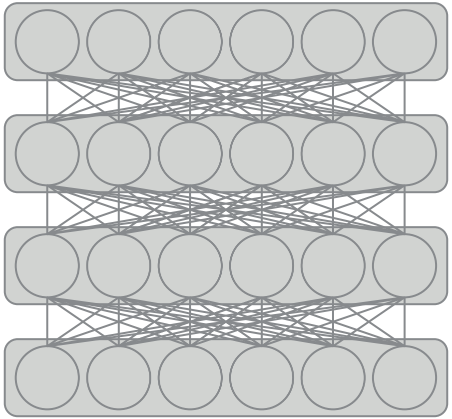
- High-Capacity
- Optimized for Learning Speed
- Easily Parallelizable (GPU)

Actor-Learner Distillation (ALD)

Actor Model $\mathcal{M}_A$

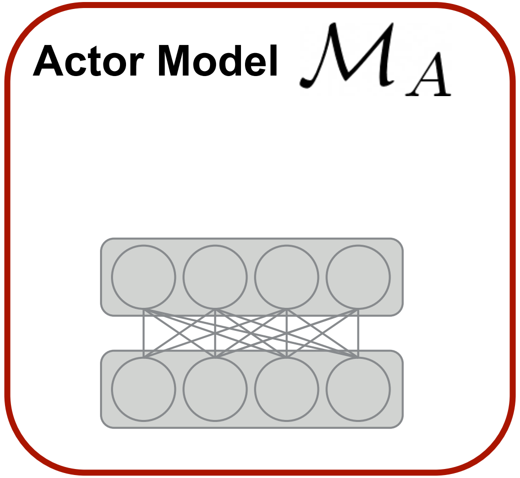


Learner Model $\mathcal{M}_L$

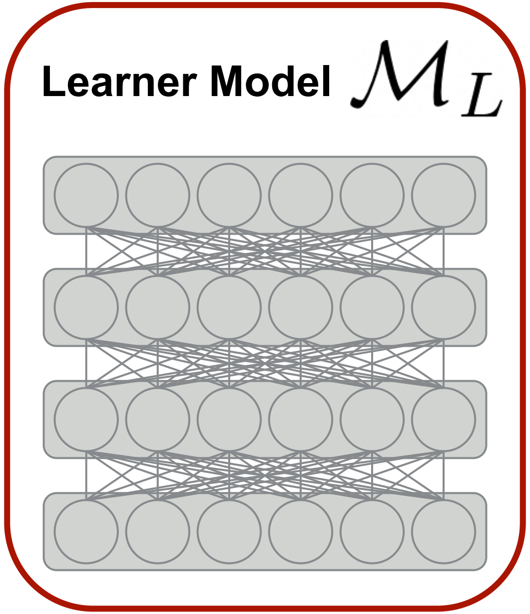


Train with RL

Actor-Learner Distillation (ALD)



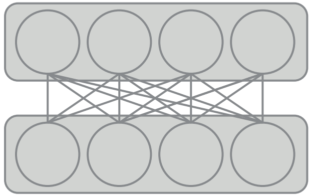
Distill from Learner



Train with RL

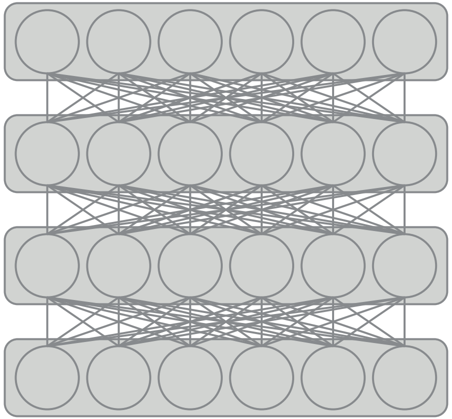
Actor-Learner Distillation (ALD)

Actor Model $\mathcal{M}_A$



Distill from Learner

Learner Model $\mathcal{M}_L$



Train with RL

Regularize towards Actor

Actor-Learner Distillation (ALD)

Objectives

Learner Objective:

$$J(\theta_L) = \mathbb{E}_{(s_t,a_t,r_t)_{t=1}^{H} \sim \pi_{\theta_A}} \left[\overbrace{L_{RL}((s_t, a_t, r_t)_{t=1}^{H}; \theta_L)}^{\text{Reinforcement Learning}} - \overbrace{\sum_{t=1}^{H} \mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t) || \pi_{\theta_L}(\cdot|s_t))}^{\text{Policy Distillation}} \right]$$

Actor-Learner Distillation (ALD)

Objectives

Learner Objective:

$$J(\theta_L) = \mathbb{E}_{(s_t, a_t, r_t)_{t=1}^{H} \sim \pi_{\theta_A}} \left[\overbrace{L_{RL}((s_t, a_t, r_t)_{t=1}^{H}; \theta_L)}^{\text{Reinforcement Learning}} - \overbrace{\sum_{t=1}^{H} \mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t)||\pi_{\theta_L}(\cdot|s_t))}^{\text{Policy Distillation}} \right]$$

Actor Objective:

$$J(\theta_A) = \mathbb{E}_{(s_t, a_t, r_t)_{t=1}^{H} \sim \pi_{\theta_A}} \left[\overbrace{\sum_{t=1}^{H} \mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t)||\pi_{\theta_L}(\cdot|s_t))}^{\text{Policy Distillation}} \right]$$

Actor-Learner Distillation (ALD)

Value Distillation Loss

Actor Objective w/ Value Distillation:

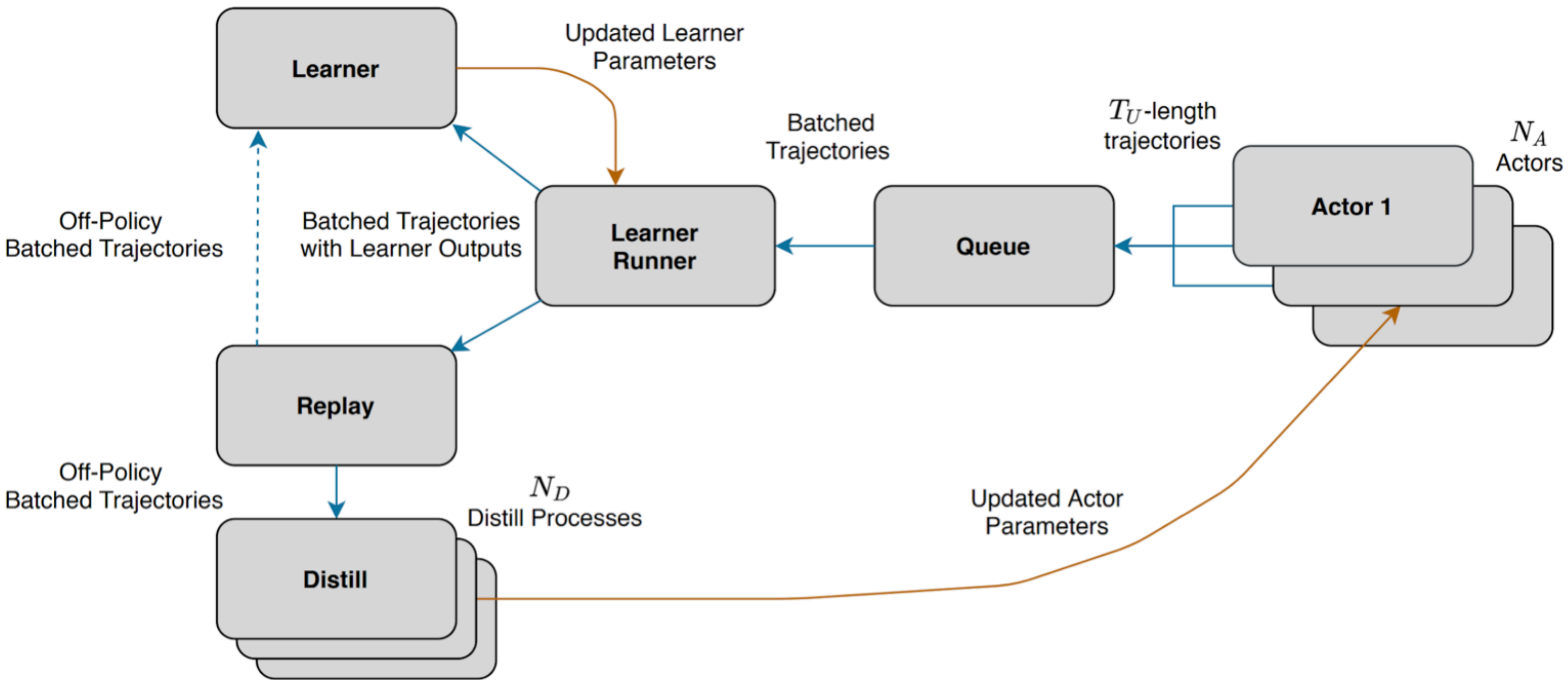
- Actor predicts Learner's value predictions.
- Improves Representation Learning at the feature level.

$$J(\theta_A) = \mathbb{E}_{(s_t, a_t, r_t)_{t=1}^{H} \sim \pi_{\theta_A}} \left[\sum_{t=1}^{H} \overbrace{\mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t) || \pi_{\theta_L}(\cdot|s_t))}^{\text{Policy Distillation}} + \overbrace{\frac{1}{2}(V_{\theta_L}(s_t) - V_{\theta_A}(s_t))^2}^{\text{Value Distillation}} \right]$$

Phasic Policy Optimization (Cobbe et al., 2020)

Actor-Learner Distillation (ALD)

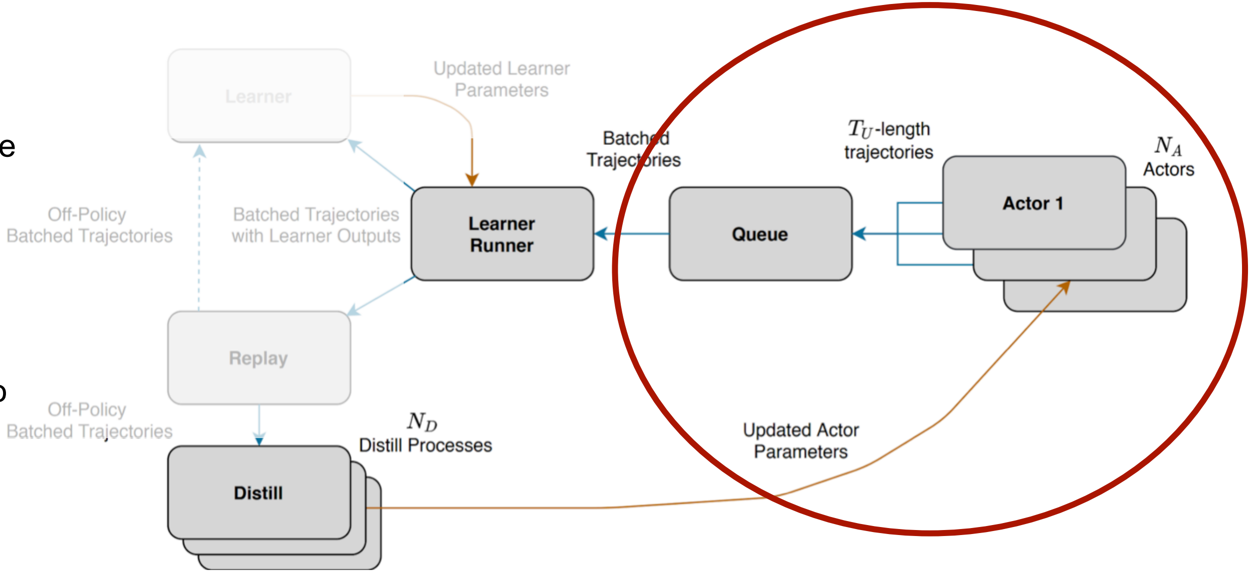
Distributed Structure



Actor-Learner Distillation (ALD)

Actors + Queue

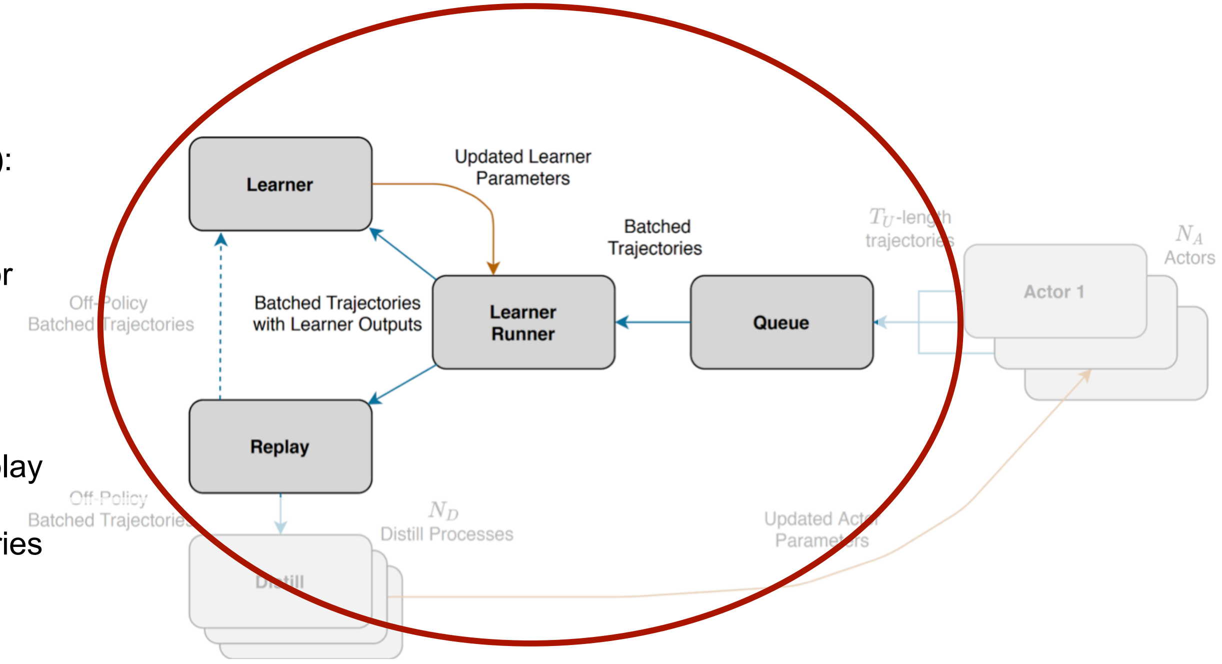
- There are N_A parallel Actors (CPU).
  - Executes actor model.
  - Submits episodes to the Queue process.
- The Queue receives episodes asynchronously.
  - Episodes are accumulated into batches.
  - The batched trajectories are then passed to the Learner Runner.



Actor-Learner Distillation (ALD)

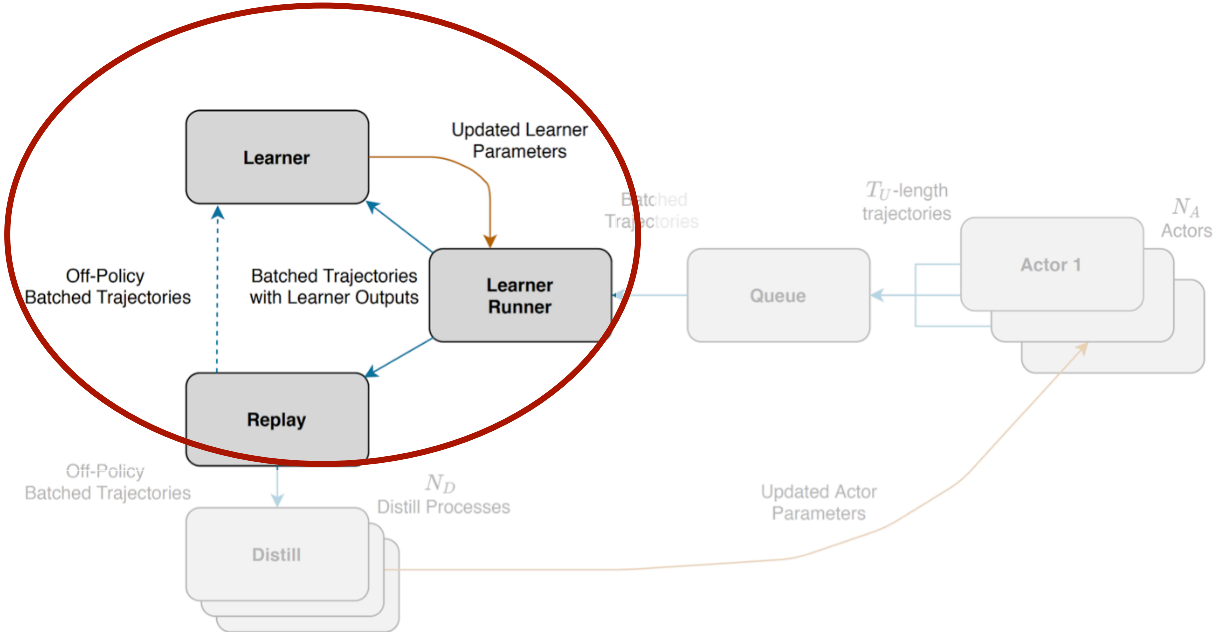
Learner Runner

- The Learner Runner process (GPU):
  - Runs Learner model on incoming batches of data.
  - Computes learning targets for Distillation to actor.
  - Pass outputs to Learner and Replay process.
- The Replay process manages a replay buffer:
  - Incoming batches of trajectories are archived.
  - Increases data diversity for distillation.



Actor-Learner Distillation (ALD)
Learner

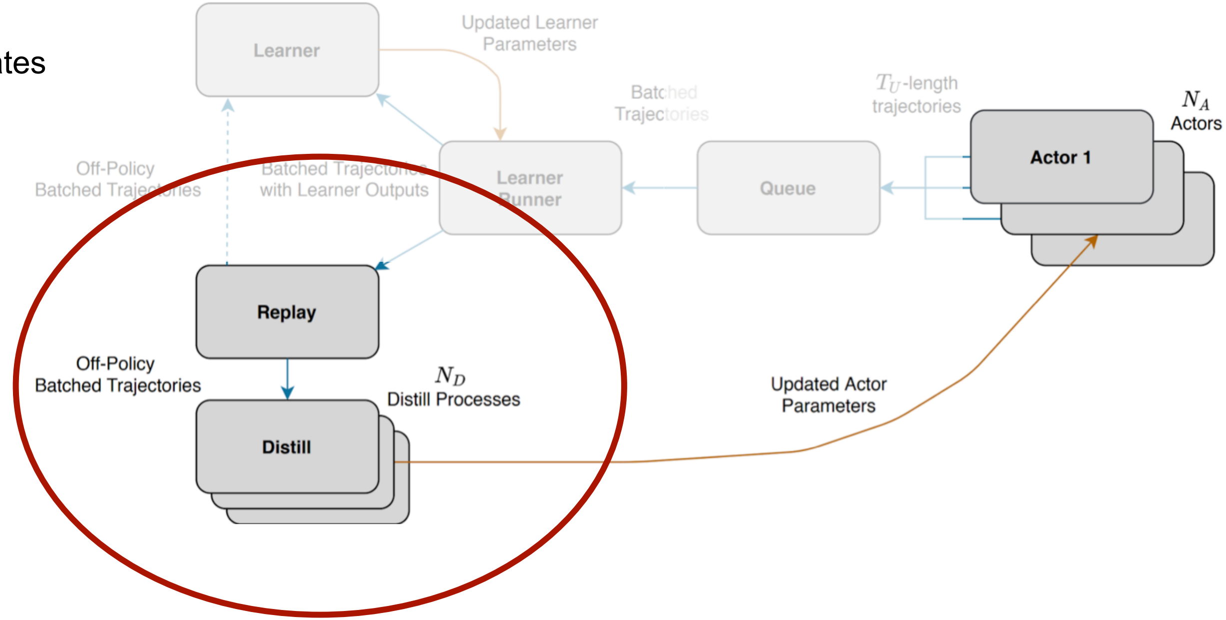
- The Learner process:
  - Computes Learner model updates based on trajectories from the Learner Runner and Replay.
  - Sends updated learner model parameters to learner runner



Actor-Learner Distillation (ALD)

Distill

- Distill process:
  - Computes Actor model updates on data from the Replay process.
  - Sends updated actor model parameters to the Actor processes.



Actor-Learner Distillation (ALD)

Sampling Distribution Mismatch?

$$J_{RL}(\pi_{\theta_L}) = \mathbb{E}_{x \sim \pi_{\theta_L}}\left[L_{RL}(x; \theta_L)\right]$$

Actor-Learner Distillation (ALD)

Sampling Distribution Mismatch?

$$J_{RL}(\pi_{\theta_L}) = \mathbb{E}_{x\sim\pi_{\theta_L}}[L_{RL}(x;\theta_L)]$$

$$J(\theta_L) = \mathbb{E}_{(s_t,a_t,r_t)_{t=1}^{H}\sim\pi_{\theta_A}}\left[L_{RL}((s_t,a_t,r_t)_{t=1}^{H};\theta_L) - \sum_{t=1}^{H}\mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t)||\pi_{\theta_L}(\cdot|s_t))\right]$$

Actor-Learner Distillation (ALD)

Sampling Distribution Mismatch?

$$J_{RL}(\pi_{\theta_L}) = \mathbb{E}_{x \sim \pi_{\theta_L}}[L_{RL}(x; \theta_L)]$$

$$J(\theta_L) = \mathbb{E}_{(s_t, a_t, r_t)_{t=1}^{H} \sim \pi_{\theta_A}} \left[L_{RL}((s_t, a_t, r_t)_{t=1}^{H}; \theta_L) - \sum_{t=1}^{H} \mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t) || \pi_{\theta_L}(\cdot|s_t)) \right]$$

We're sampling from π_{θ_A} instead of π_{θ_L}

Actor-Learner Distillation (ALD)

Sampling Distribution Mismatch?

$$J_{RL}(\pi_{\theta_L}) = \mathbb{E}_{x \sim \pi_{\theta_L}}[L_{RL}(x; \theta_L)]$$

$$J(\theta_L) = \mathbb{E}_{(s_t,a_t,r_t)_{t=1}^{H} \sim \pi_{\theta_A}}\left[L_{RL}((s_t,a_t,r_t)_{t=1}^{H}; \theta_L) - \sum_{t=1}^{H} \mathcal{D}_{KL}(\pi_{\theta_A}(\cdot|s_t)||\pi_{\theta_L}(\cdot|s_t))\right]$$

We're sampling from π_{θ_A} instead of π_{θ_L}

Need to make sure $\pi_{\theta_A} \approx \pi_{\theta_L}$

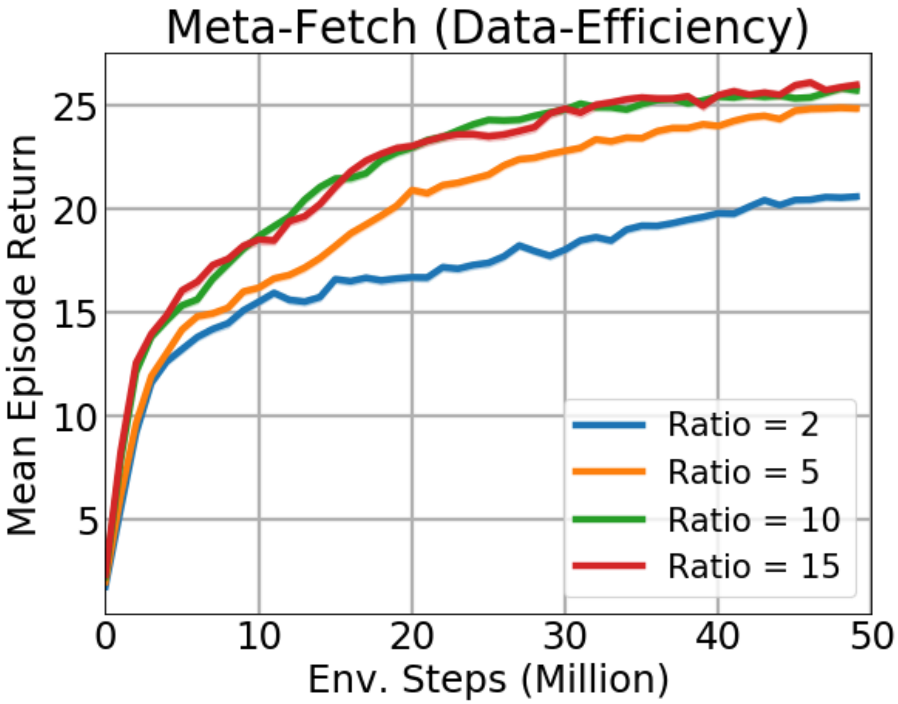
Actor-Learner Distillation (ALD)

Distillation steps per RL step (DpRL) Ratio

DpRL Ratio:

$$\frac{\texttt{\# Parameter Updates on Actor}}{\texttt{\# Parameter Updates on Learner}}$$

- **Better performance if we artificially constraint DpRL to be high.**



Actor-Learner Distillation (ALD)

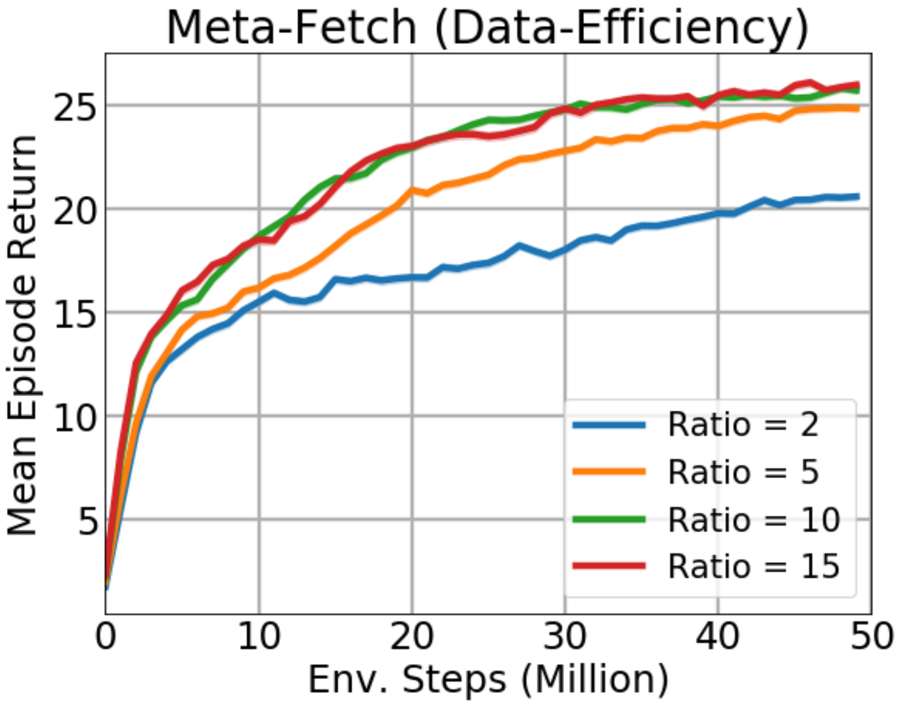
Distillation steps per RL step (DpRL) Ratio

DpRL Ratio:

$$\frac{\texttt{\# Parameter Updates on Actor}}{\texttt{\# Parameter Updates on Learner}}$$

- **Better performance if we artificially constraint DpRL to be high.**

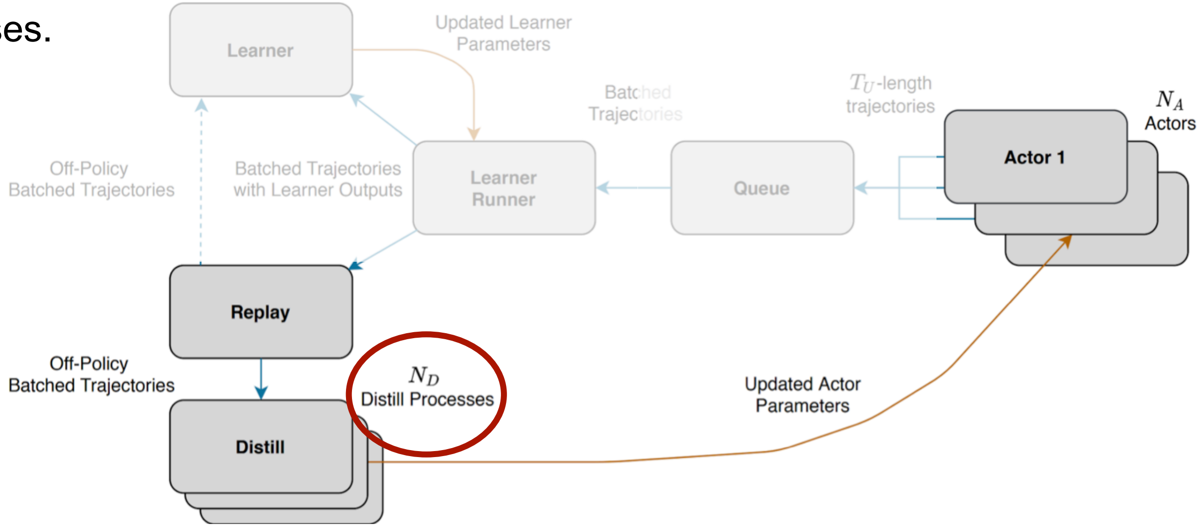
⇒ **Improve speed of distillation**



Actor-Learner Distillation (ALD)

Async Distillation using HOGWILD!

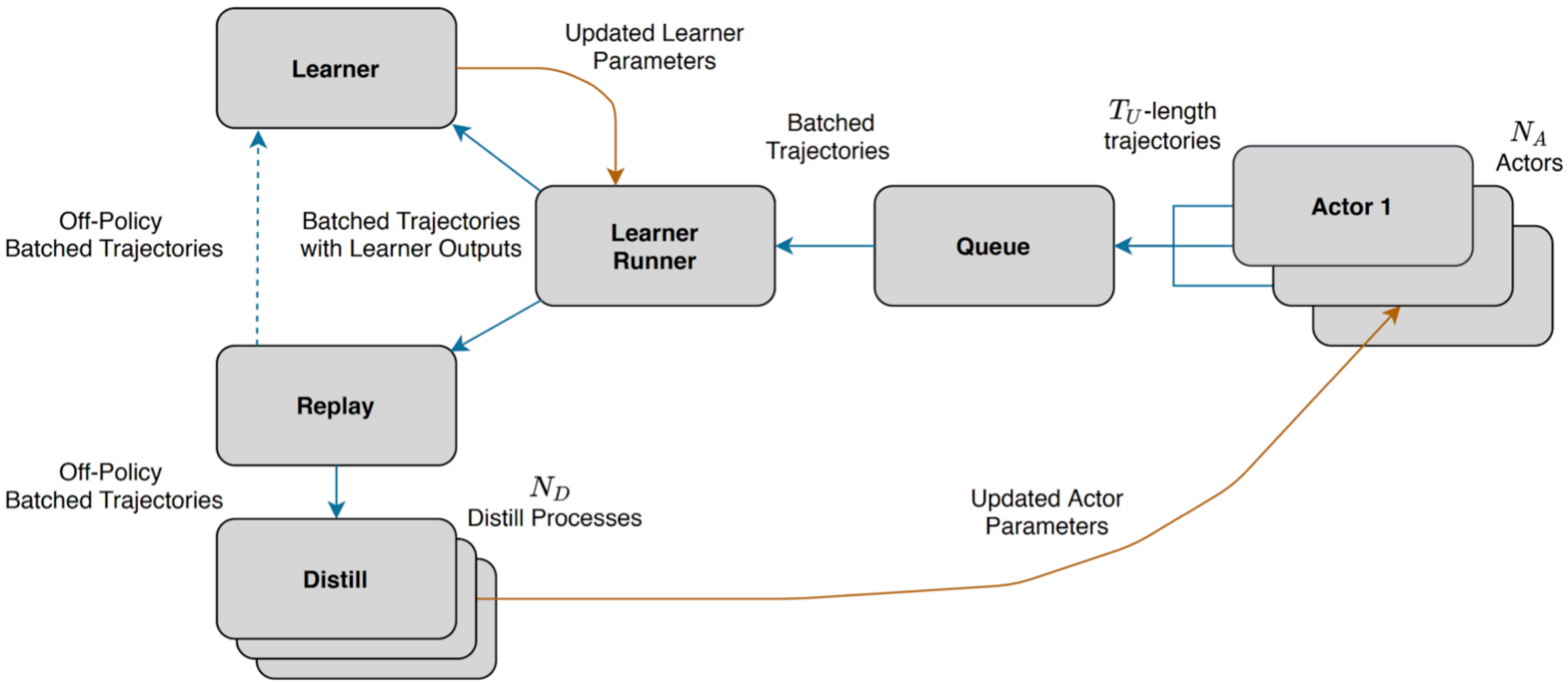
- Run N_D parallel Distill processes.
  - Each independently updates actor model parameters.
  - Asynchronously share parameters (i.e. HOGWILD!)



HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent (Niu et al., 2011)

Actor-Learner Distillation (ALD)

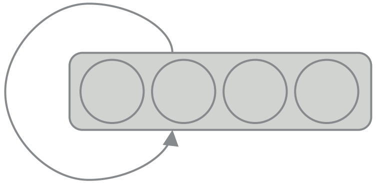
Complete System



Actor-Learner Distillation Experiments
LSTM Actor -- GTrXL Learner

Actor Model $\mathcal{M}_A$

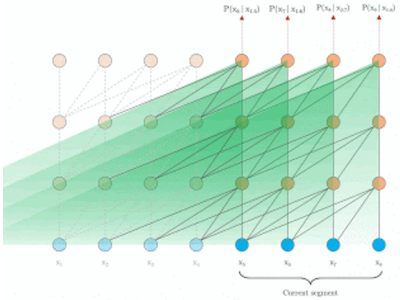
LSTM



- Low-Capacity
- Optimized for Inference Speed
- Sequential Execution (CPU)
- Generates Environment Episodes.

Learner Model $\mathcal{M}_L$

GTrXL



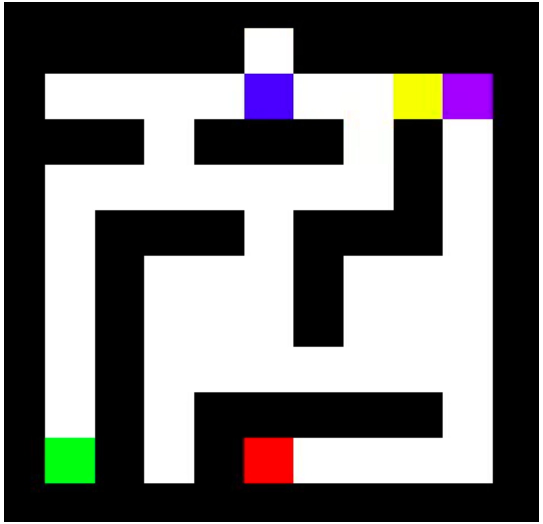
- High-Capacity
- Optimized for Learning Speed
- Easily Parallelizable (GPU)

I-Maze
Remembering Far into the Past

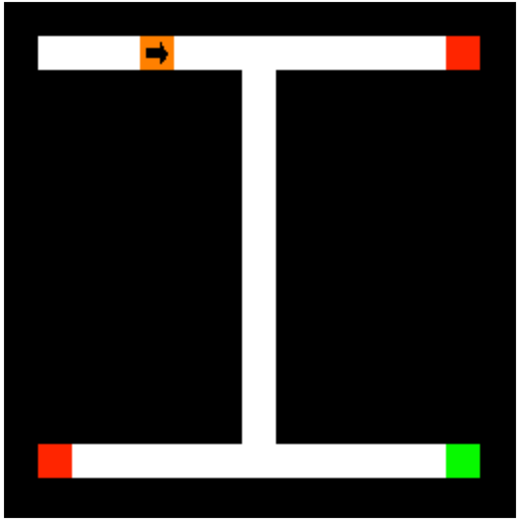
- Indicator: Either blue or pink

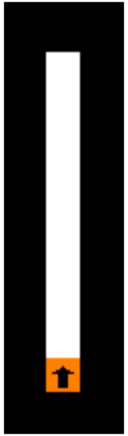
  - If blue, find the green block
  - If pink, find the red block

- Negative reward if agent does not find correct block in N steps or goes to wrong block.

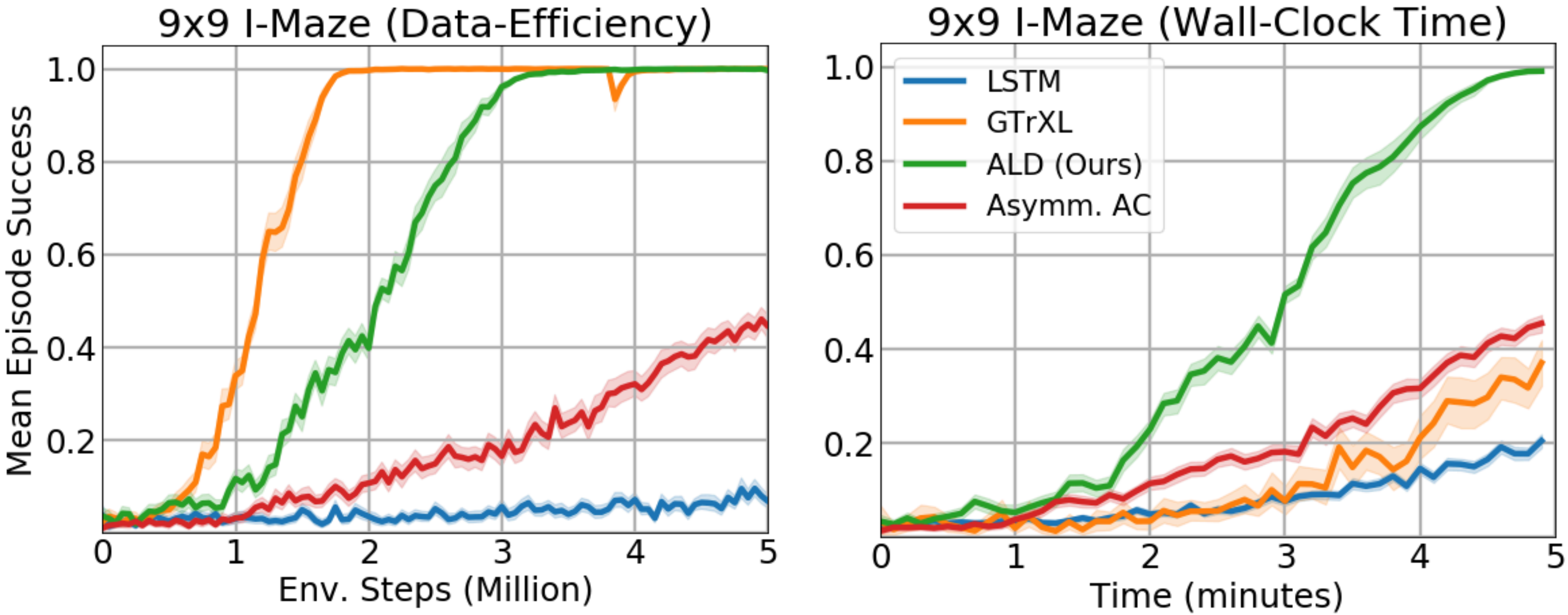


I-Maze
Remembering Far into the Past





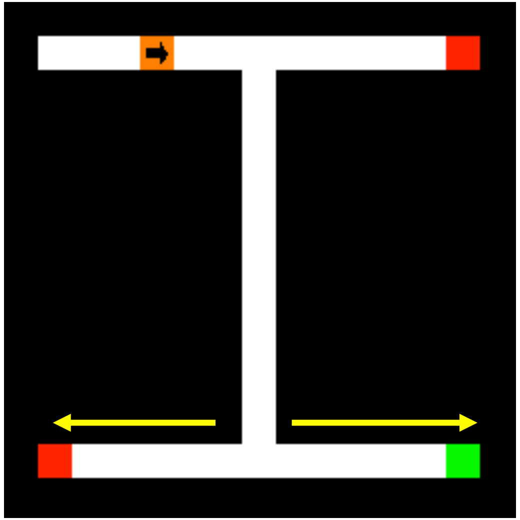
I-Maze Results



Transferring Learner's Inductive Bias

- Local optimum in I-Maze
  - Agent enters either goal without discernment.
  - Expected reward is 0.5 instead of 0 for not entering any goal.

- This behaviour does not require long-term memory:
  - Only the ability to navigate to a corner.

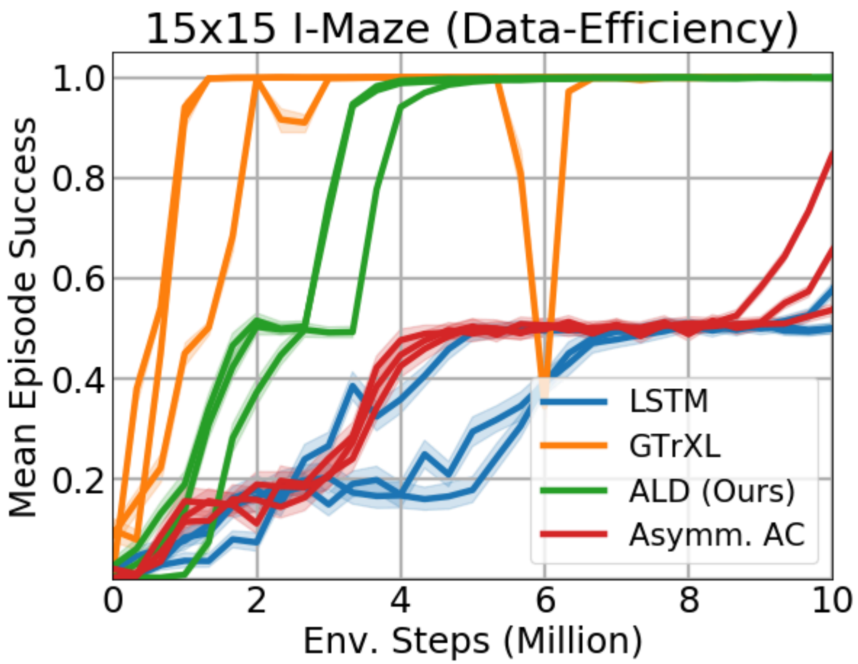


Transferring Learner's Inductive Bias

- Per-seed curves on 15x15 I-Maze

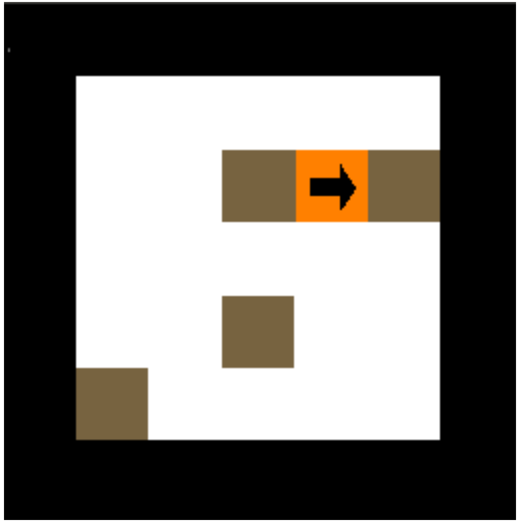
- Model stuck at 0.5 => local optimum.
  - Transformer avoids local optimum.
  - LSTM agents stuck there for significant amount of time.
  - ALD exits very quickly in comparison.

- ALD LSTM model has learned to rapidly compress the transformer memory.

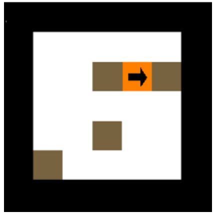


Meta-Fetch:

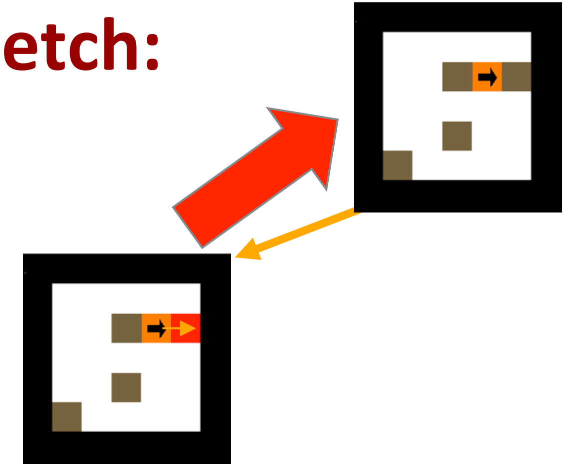
Partially-Observed, Combinatorial Search



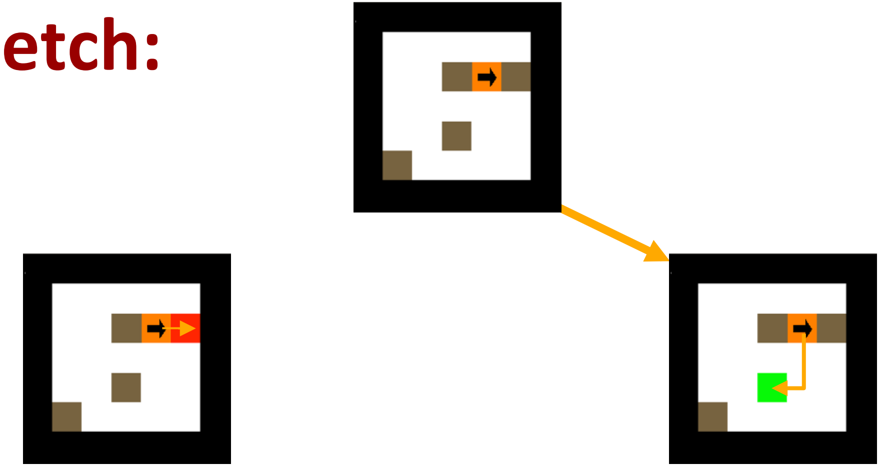
Meta-Fetch:



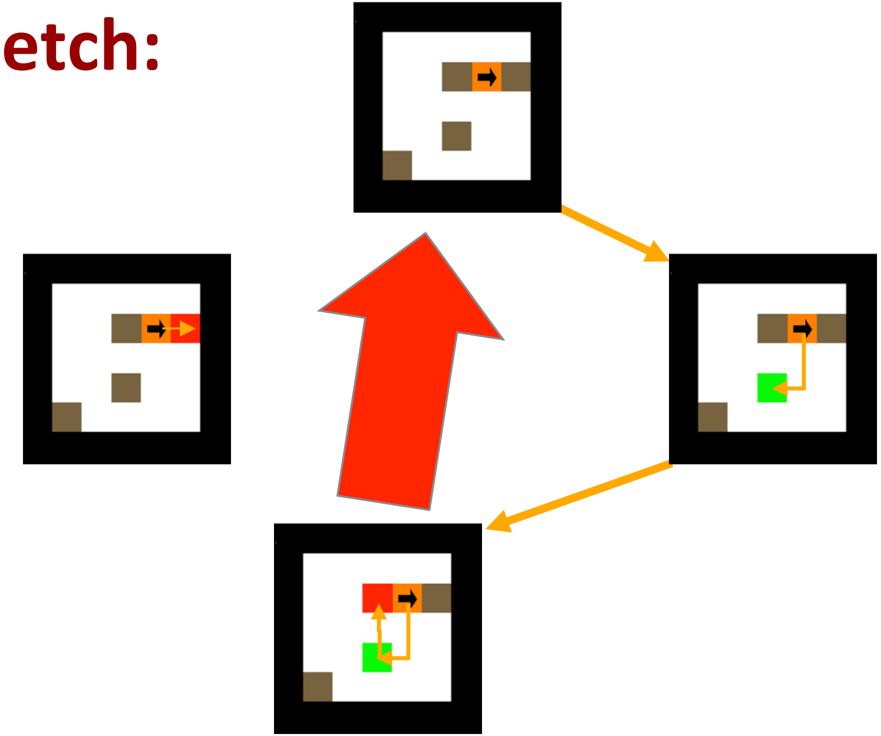
Meta-Fetch:



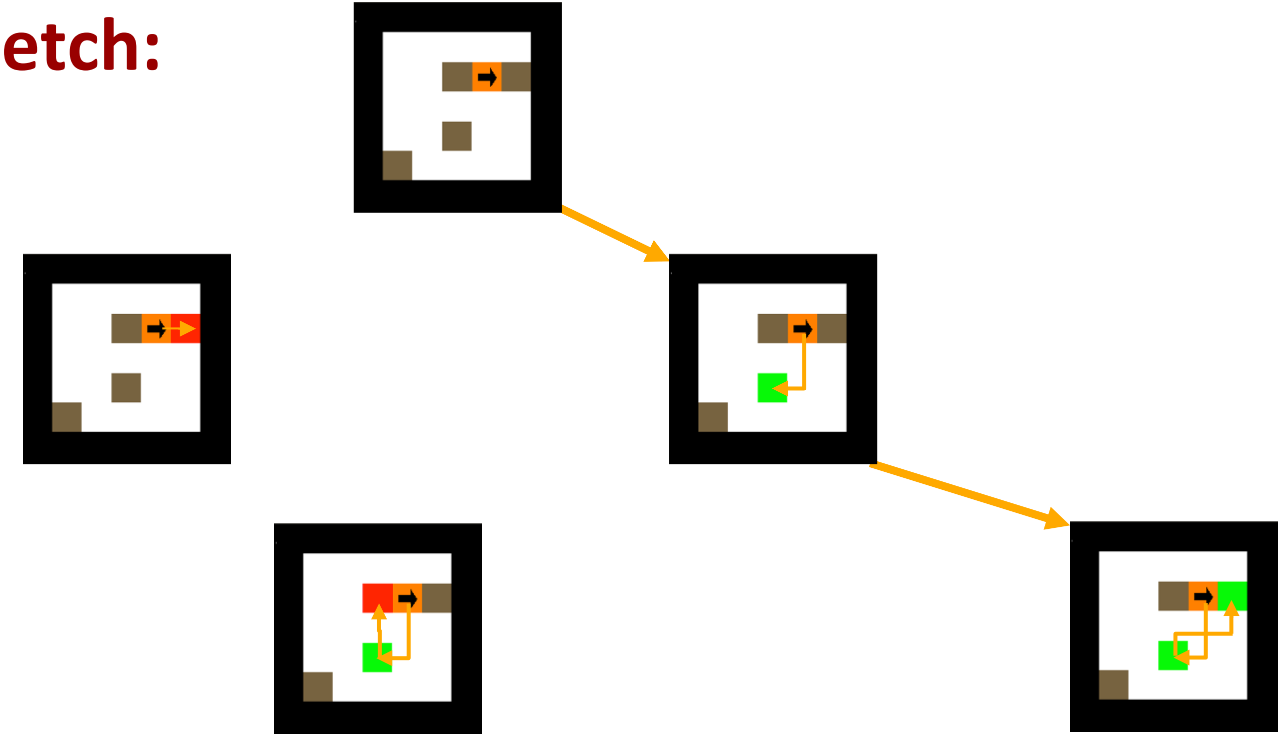
Meta-Fetch:



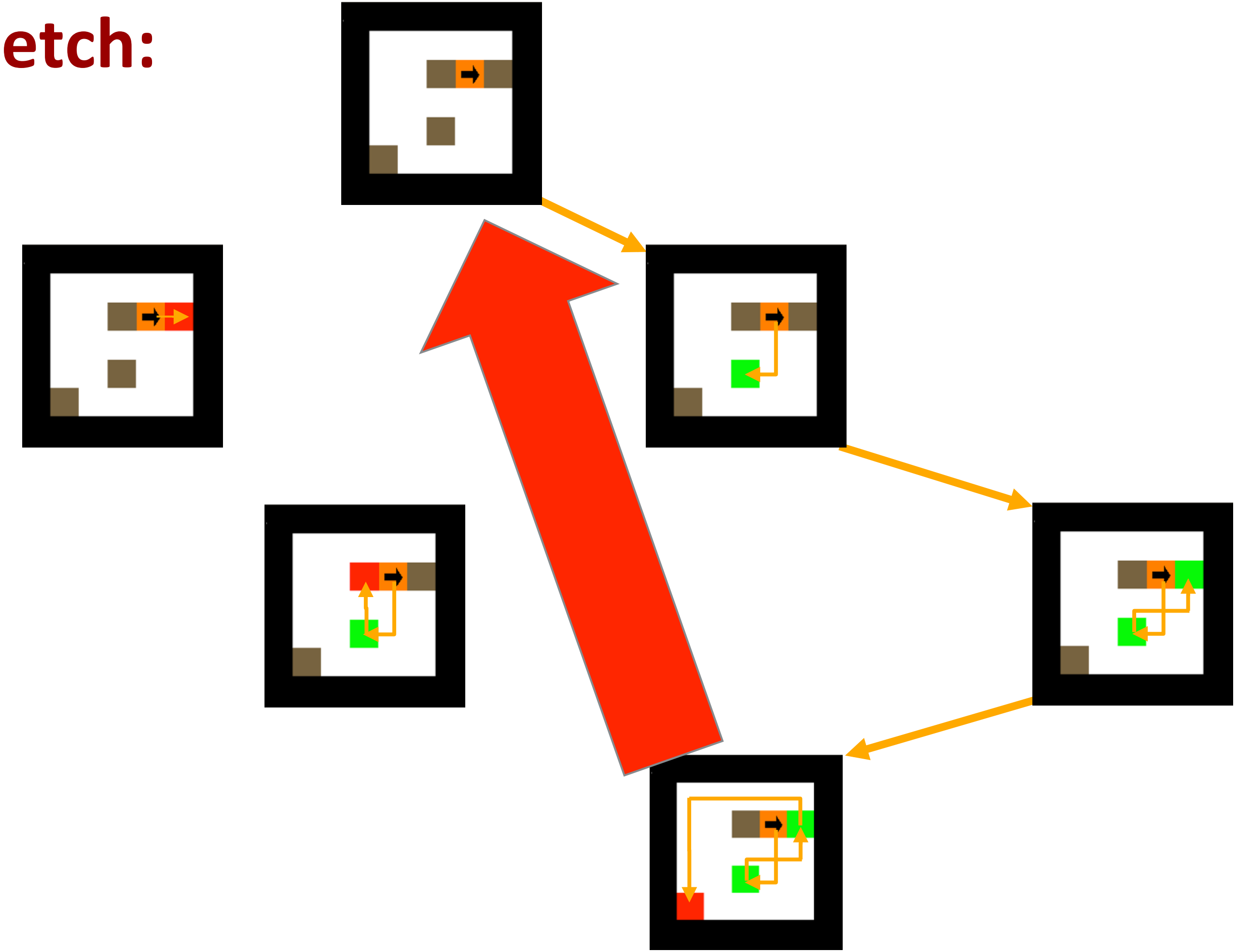
Meta-Fetch:



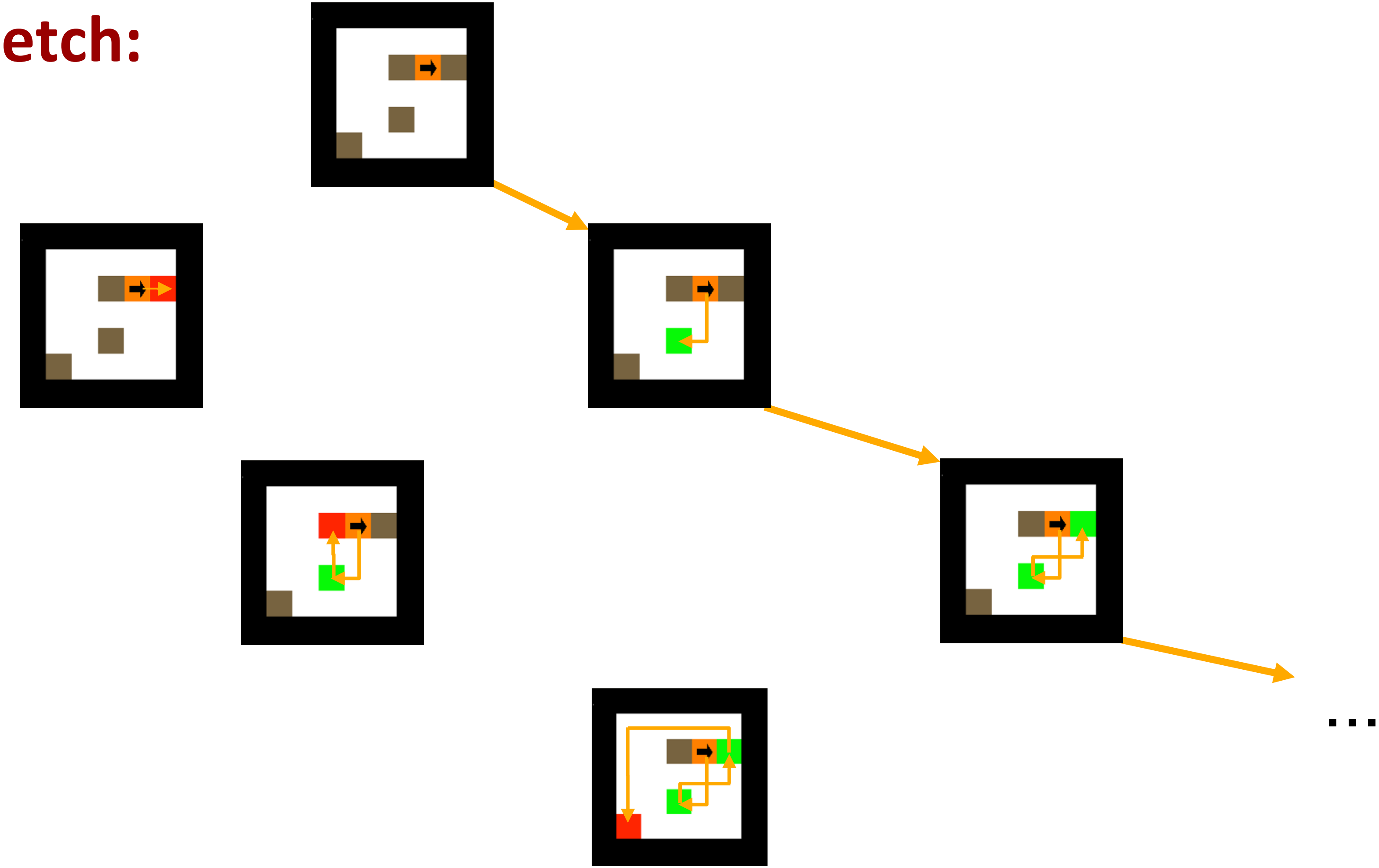
Meta-Fetch:



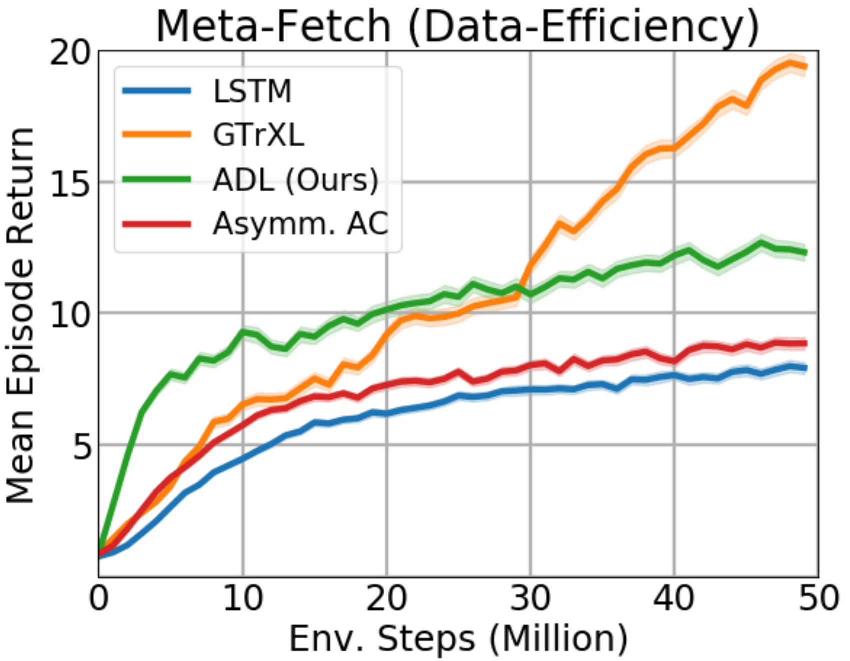
Meta-Fetch:



Meta-Fetch:



Meta-Fetch Results



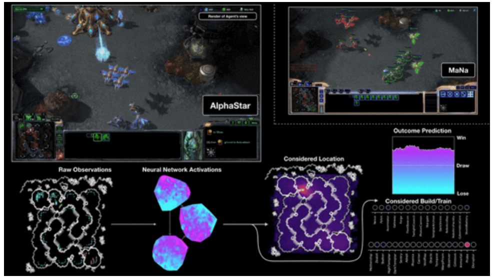
References

- Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context (Dai et al., 2019)
- Stabilizing Transformers for Reinforcement Learning (Parisotto et al., 2019)
- DeepMind Lab (Beattie et al., 2016)
- Do Deep Nets Really Need to be Deep? (Ba et al., 2013)
- Distilling the Knowledge in a Neural Network (Hinton et al., 2015)
- Phasic Policy Optimization (Cobbe et al., 2020)
- HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent (Niu et al., 2011)

Data Inefficiency Restricts Real-World Impact

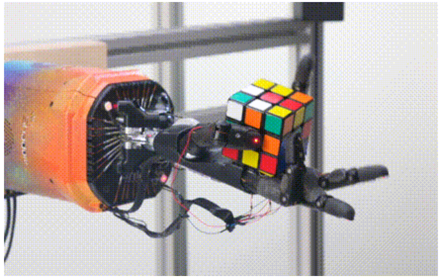
AlphaZero

140 million Go games



AlphaStar

>1 million SC2 games



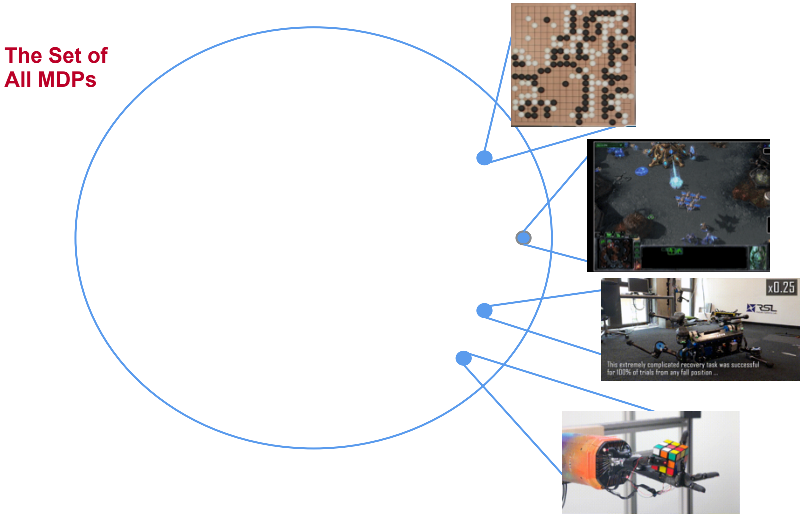
Rubik's Cube

13,000 simulation years

- Data inefficiency is a critical obstruction to Deep RL's widespread use:
  - Currently Deep RL is **constrained to environments with viable simulators.**
- Recent off-policy and model-based algorithms show improvements.
  - But still require extremely large amounts of data.

Addressing the Data Inefficiency of Deep RL

RL Algorithms work over any **Markov Decision Process (MDP)**.

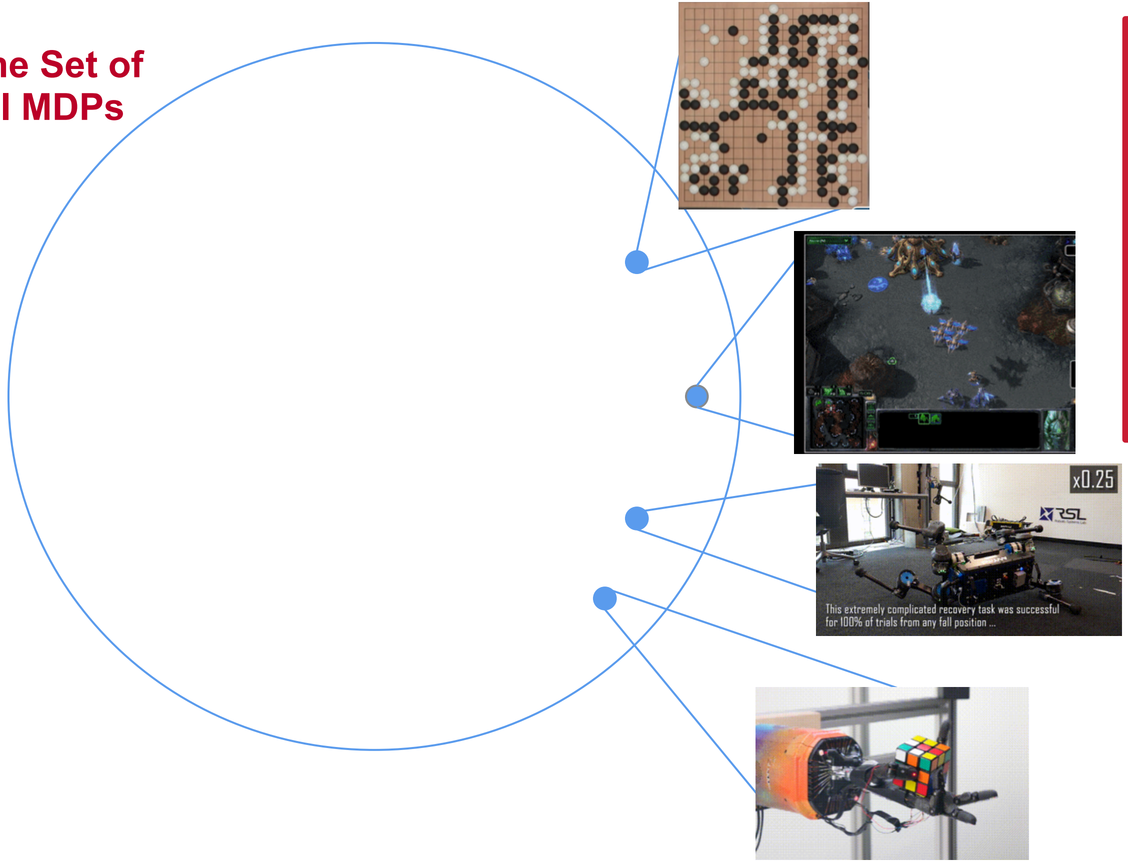


Key Insight:

- **<u>Specialize</u>** the learning algorithm.

Addressing the Data Inefficiency of Deep RL

The Set of All MDPs

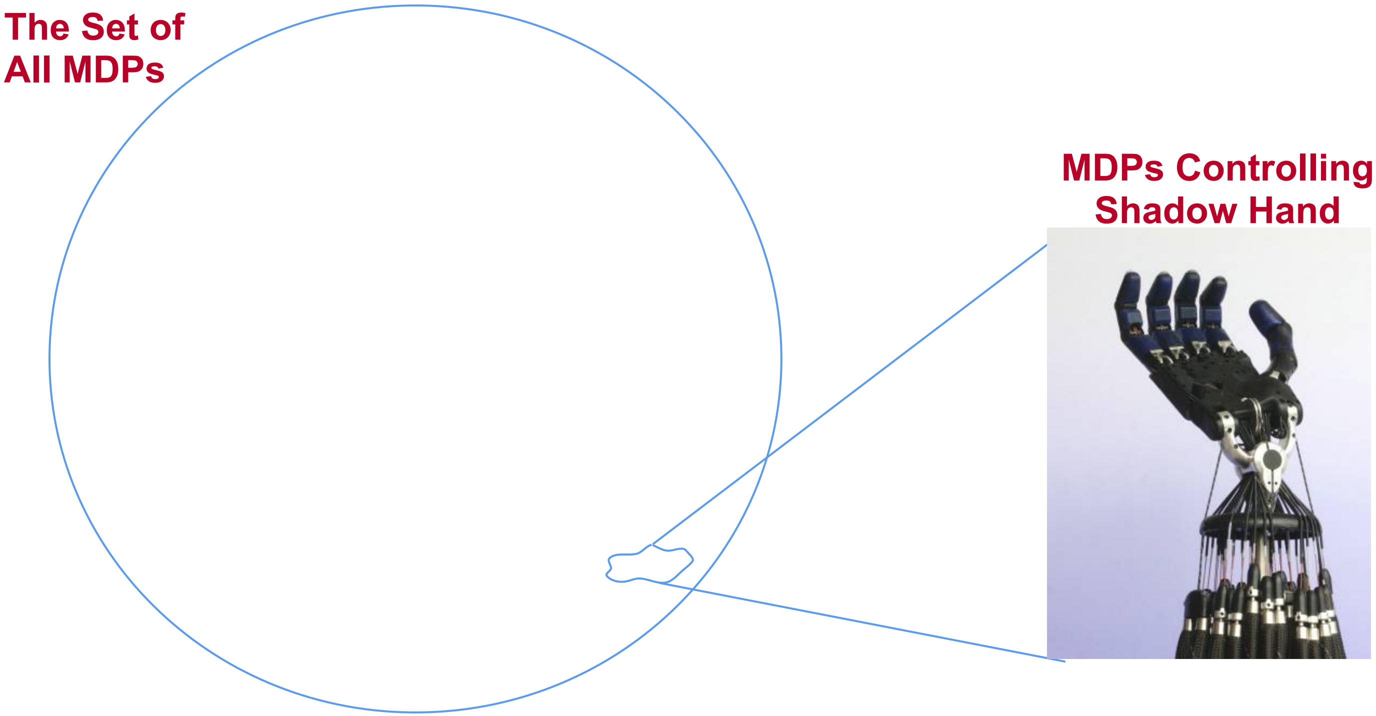


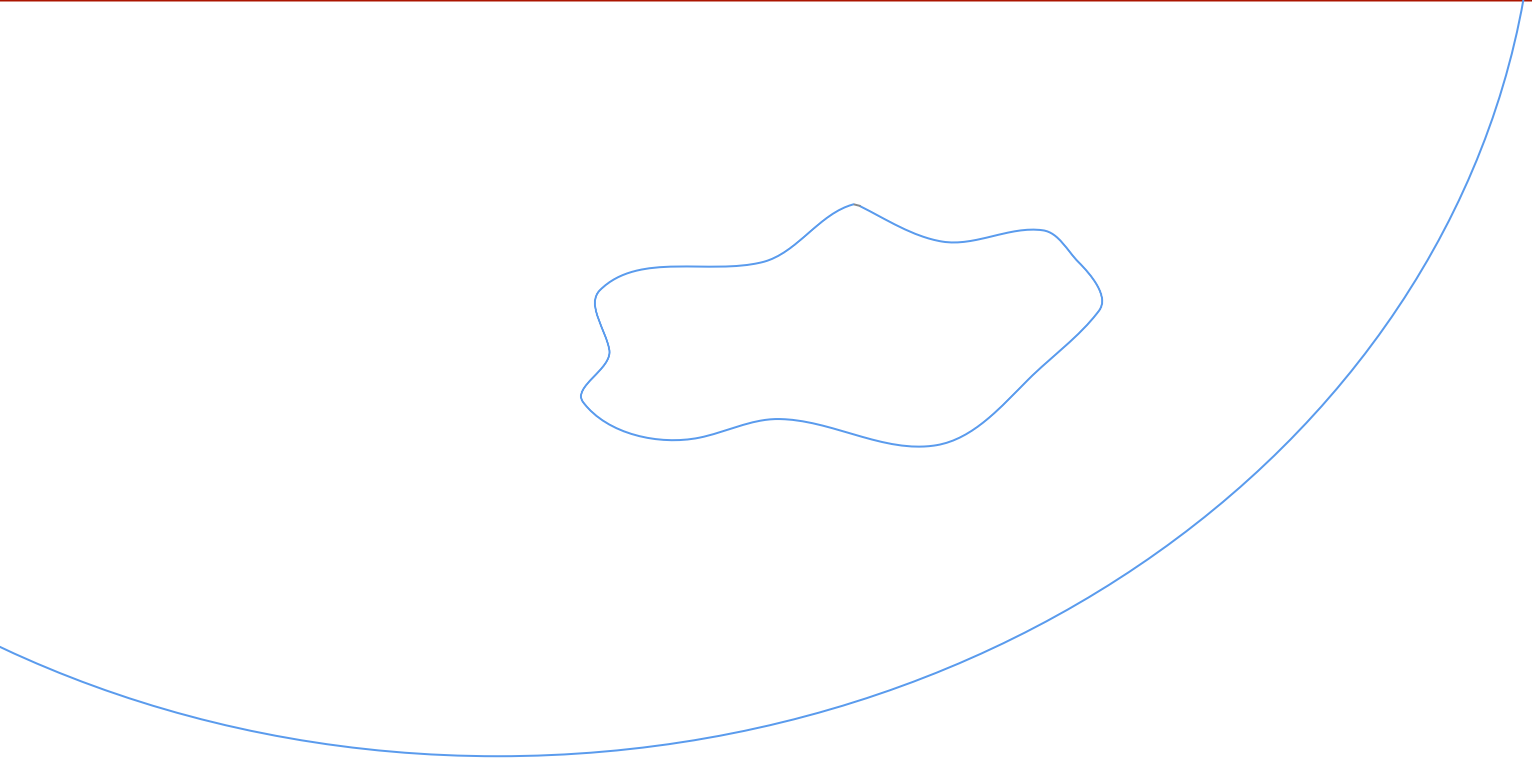
RL works over any MDP.

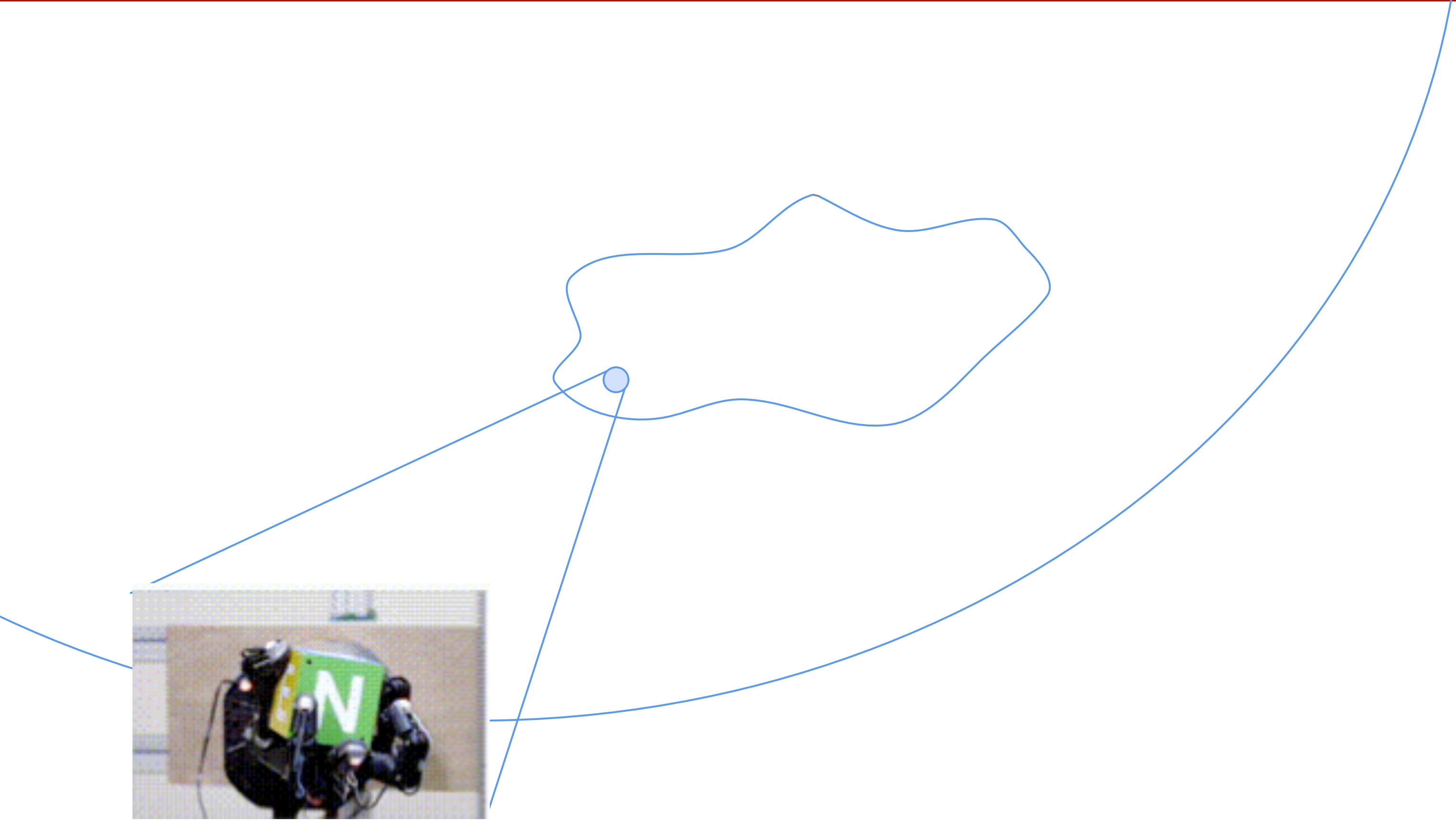
Key Insight:

- Develop specialized learning algorithms.

Specialization to a Distribution of Environments





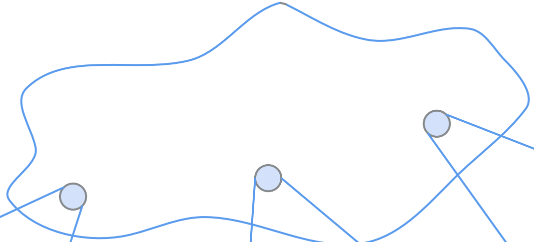


Dexterous
Manipulation

Dexterous Manipulation

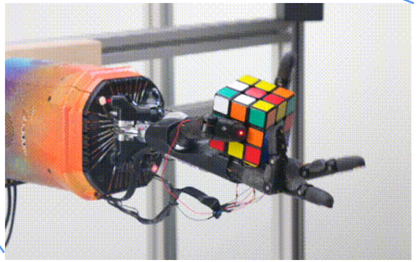
Safe Manipulation

A Learning Approach to Specialization



Dexterous Manipulation

Safe Manipulation



Solving Rubik's Cube

Standard Reinforcement Learning

Task T_1

RL Algorithm

Dexterous Manipulation

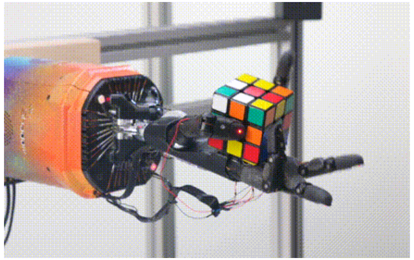
Task T_2

RL Algorithm

Safe Manipulation

Task T_3

RL Algorithm



Solving Rubik's Cube

Meta-Learning

Task T_2

Meta Algorithm

Task Distribution Parameters μ

Safe Manipulation

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta Reinforcement Learning

Dataset T_1

Meta Algorithm ← μ

Dexterous Manipulation

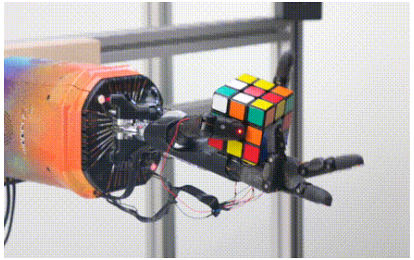
Dataset T_2

Meta Algorithm ← μ

Safe Manipulation

Dataset T_3

Meta Algorithm ← μ



Solving Rubik's Cube

Meta Reinforcement Learning

Dataset T_1

Meta Algorithm

μ

Dexterous Manipulation

Dataset T_2

Meta Algorithm

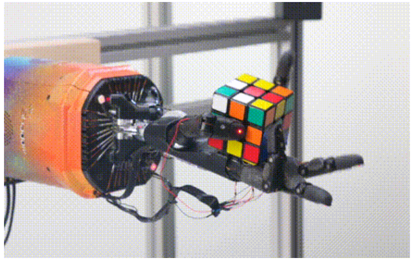
μ

Safe Manipulation

Dataset T_3

Meta Algorithm

μ



Solving Rubik's Cube

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta Reinforcement Learning

Dataset T_1

Meta Algorithm

μ

Backpropagate & Update

Dexterous Manipulation

Dataset T_2

Meta Algorithm

μ

Backpropagate & Update

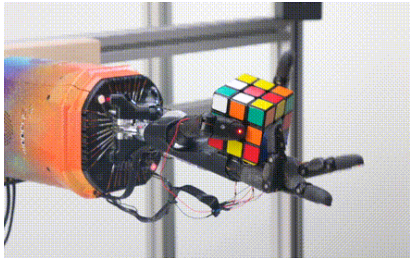
Safe Manipulation

Dataset T_3

Meta Algorithm

μ

Backpropagate & Update



Solving Rubik's Cube

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta-Learning as a formalism

Definitions:

Task: T

Experience: $x \in T$

Agent Parameters: θ

Task Distribution: $\mathcal{D}_T$

Performance Measure: $\phi(\theta, x)$

Expected Performance in Task T

$$\Phi(\theta) = \mathbb{E}_{x \in T}[\phi(\theta, x)]$$

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta-Learning as a formalism

Definitions:

Task: T

Experience: $x \in T$

Agent Parameters: θ

Task Distribution: $\mathcal{D}_T$

Performance Measure: $\phi(\theta, x)$

Learning Algorithm: $\mathbf{L}_\mu(\theta, T)$

Algorithm Parameters: μ

Expected Performance in Task T

$$\Phi(\theta) = \mathbb{E}_{x \in T}[\phi(\theta, x)]$$

Expected Performance Gain of $\mathbf{L}_\mu$ in Tasks $\mathcal{D}_T$

$$\delta(\mathbf{L}_\mu) = \mathbb{E}_{\theta \in \Theta, T \in \mathcal{D}_T}\left[\Phi(\mathbf{L}_\mu(\theta, T)) - \Phi(\theta)\right]$$

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta-Learning as a formalism

Definitions:

Task: T

Experience: $x \in T$

Agent Parameters: θ

Task Distribution: $\mathcal{D}_T$

Performance Measure: $\phi(\theta, x)$

Learning Algorithm: $\mathbf{L}_\mu(\theta, T)$

Algorithm Parameters: μ

Meta-Algorithm: $\mathbf{ML}(\mu, T)$

Expected Performance in Task T

$$\Phi(\theta) = \mathbb{E}_{x \in T}[\phi(\theta, x)]$$

Expected Performance Gain of $\mathbf{L}_\mu$ in Tasks $\mathcal{D}_T$

$$\delta(\mathbf{L}_\mu) = \mathbb{E}_{\theta \in \Theta, T \in \mathcal{D}_T}\left[\Phi(\mathbf{L}_\mu(\theta, T)) - \Phi(\theta)\right]$$

Maximize:

$$\mathbb{E}_{\mu \in M, T \in \mathcal{D}_T}\left[\delta(\mathbf{L}_{\mathbf{ML}(\mu, \mathbf{T})}) - \delta(\mathbf{L}_\mu)\right] > 0$$

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta-RL

Definitions:

Task: T

Experience: $x \in T$

Agent Parameters: θ

Task Distribution: $\mathcal{D}_T$

Performance Measure: $\phi(\theta, x)$

Learning Algorithm: $\mathbf{L}_\mu(\theta, T)$

Algorithm Parameters: μ

Meta-Algorithm: $\mathbf{ML}(\mu, T)$

$$T = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R})$$

$$x = (s_1, a_1, r_1, \ldots, s_H, a_H, r_H), x \sim \pi_\theta, T$$

$$\phi(x) = \sum_t r_t$$

Expected Performance in Task T

$$\Phi(\theta) = \mathbb{E}_{x \sim \pi_\theta, T}\left[\sum_t r_t\right]$$

Expected Performance Gain of $\mathbf{L}_\mu$ in Tasks $\mathcal{D}_T$

$$\delta(\mathbf{L}_\mu) = \mathbb{E}_{\theta \in \Theta, T \in \mathcal{D}_T}\left[\Phi(\mathbf{L}_\mu(\theta, T)) - \Phi(\theta)\right]$$

Maximize:

$$\mathbb{E}_{\mu \in M, T \in \mathcal{D}_T}\left[\delta(\mathbf{L}_{\mathbf{ML}(\mu, \mathbf{T})}) - \delta(\mathbf{L}_\mu)\right] > 0$$

Tom Schaul and Juergen Schmidhuber (2010), Scholarpedia, 5(6):4650.

Meta-Reinforcement Learning

Learning Algorithm: $\mathbf{L}_{\mu}(\theta, T)$

Algorithm Parameters: μ

Meta-Algorithm: $\mathbf{ML}(\mu, T)$

Meta-Reinforcement Learning

Amortizing RL's Data Inefficiency

Learning Algorithm: $\mathbf{L}_{\mu}(\theta, T)$

Algorithm Parameters: μ

Meta-Algorithm: $\mathbf{ML}(\mu, T)$

$$\mathbf{ML}(\mu, \{T_1, \ldots, T_N\}) = \begin{cases} \textbf{Policy Gradient} \\ \textbf{Q-learning} \\ \textbf{...} \end{cases}$$

This can be very expensive, but we gain the ability to later train any new task with much less data.

Meta-Reinforcement Learning
Parameterizing the Learning Algorithm

Learning Algorithm: $\mathbf{L}_\mu(\theta, T)$

Algorithm Parameters: μ

Meta-Reinforcement Learning
Parameterizing the Learning Algorithm

Learning Algorithm: $\mathbf{L}_\mu(\theta, T)$

Algorithm Parameters: μ

- Optimization-based:
  - MAML (Finn et al. 2017)
  - Reptile (Nichol et al. 2018)
- Episodic Control:
  - Model-Free Episodic Control (Blundell et al. 2016)
  - Neural Episodic Control (Pritzel et al. 2017)
- Memory / Sequence Models:
  - Learning to Learn Using Gradient Descent (Hochreiter 2001)
  - RL^2 (Duan et al. 2016)
  - L2RL (Wang et al. 2016)

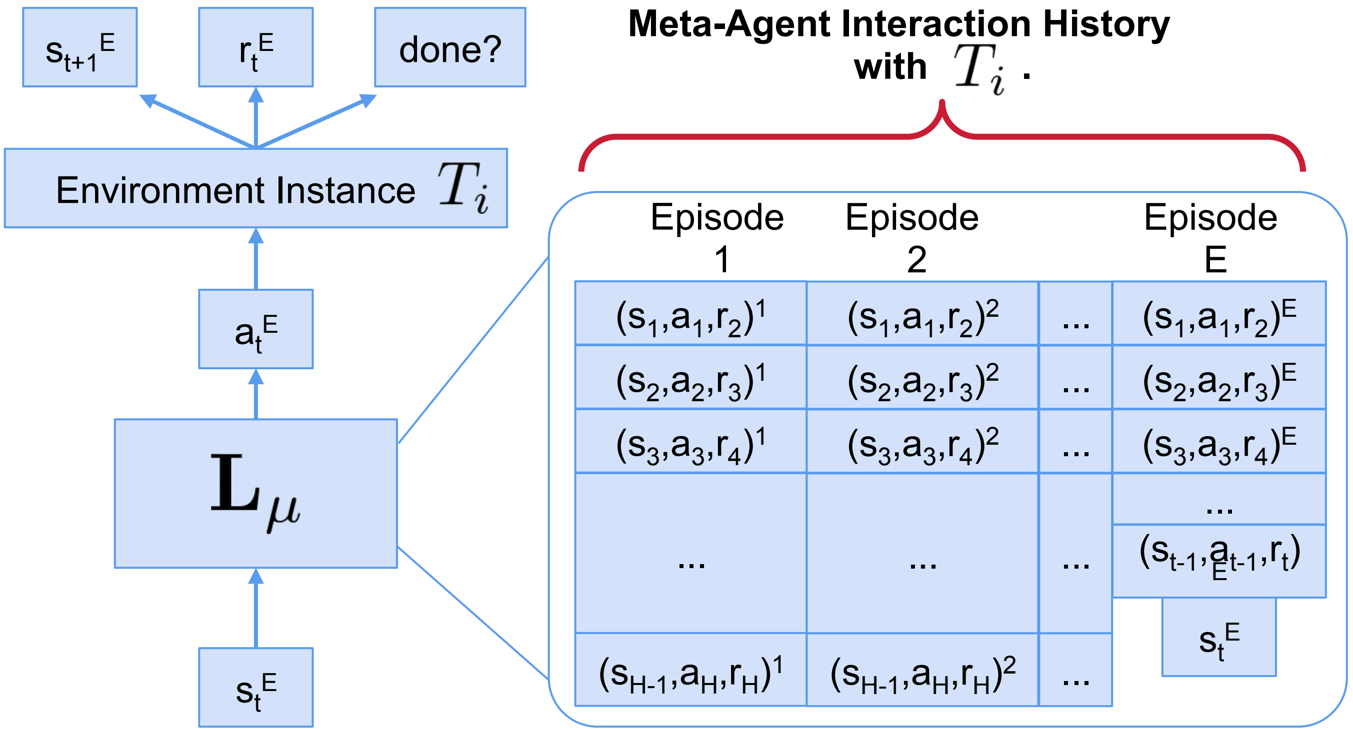
Meta-Reinforcement Learning

Learning Algorithm: $\mathbf{L}_{\mu}(\theta, T)$

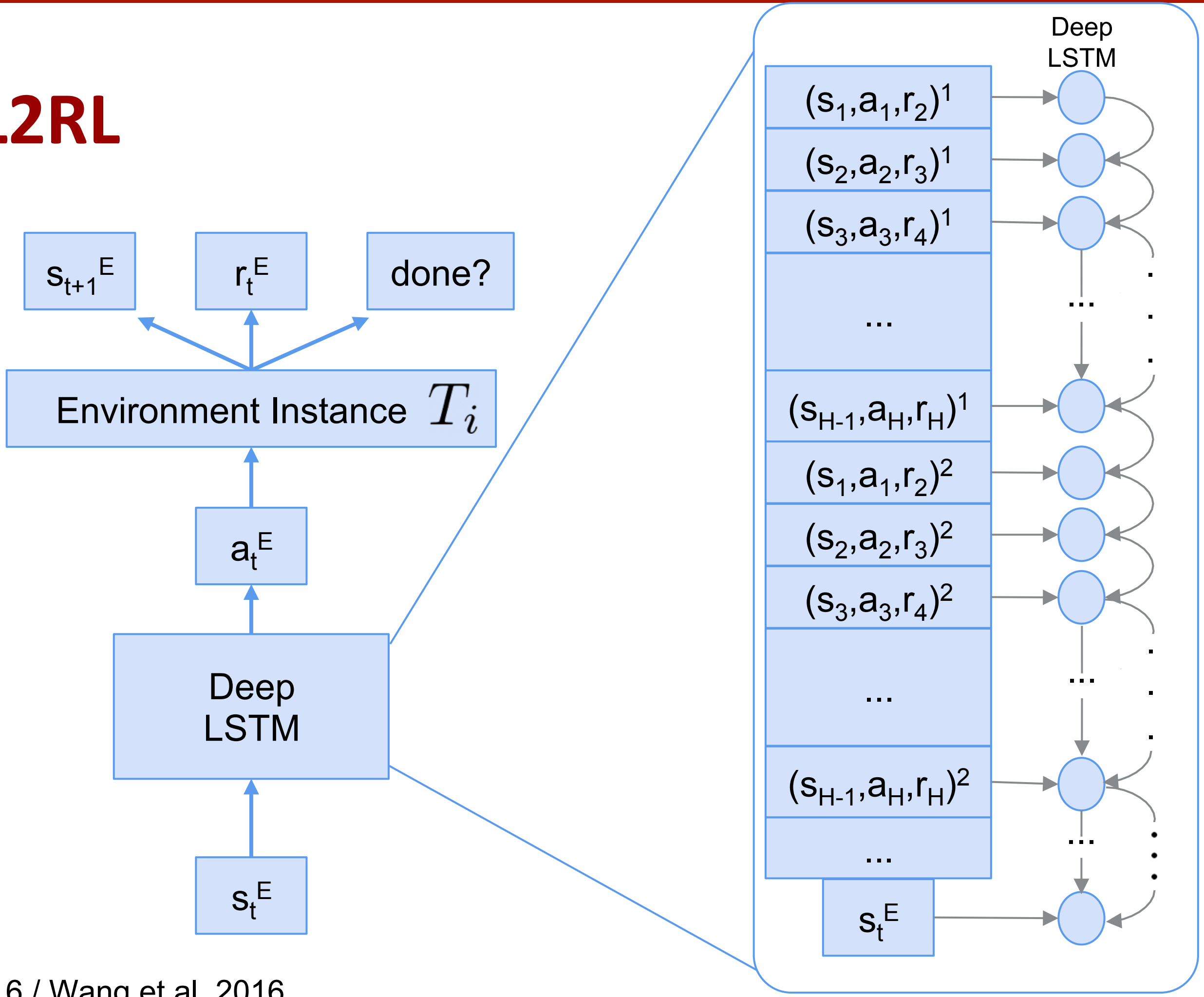
Algorithm Parameters: μ

- Optimization-based:
  - MAML (Finn et al. 2017)
  - Reptile (Nichol et al. 2018)
- Episodic Control:
  - Model-Free Episodic Control (Blundell et al. 2016)
  - Neural Episodic Control (Pritzel et al. 2017)
- Memory / Sequence Models:
  - Learning to Learn Using Gradient Descent (Hochreiter 2001)
  - RL^2 (Duan et al. 2016)
  - L2RL (Wang et al. 2016)

Meta-RL through Memory



RL² / L2RL



Duan et al. 2016 / Wang et al. 2016

Algorithm as Architecture