

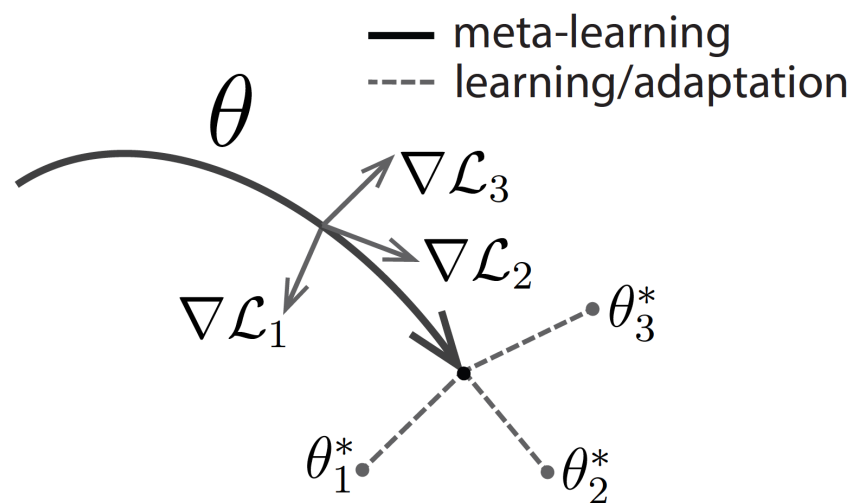


Reinforcement Learning

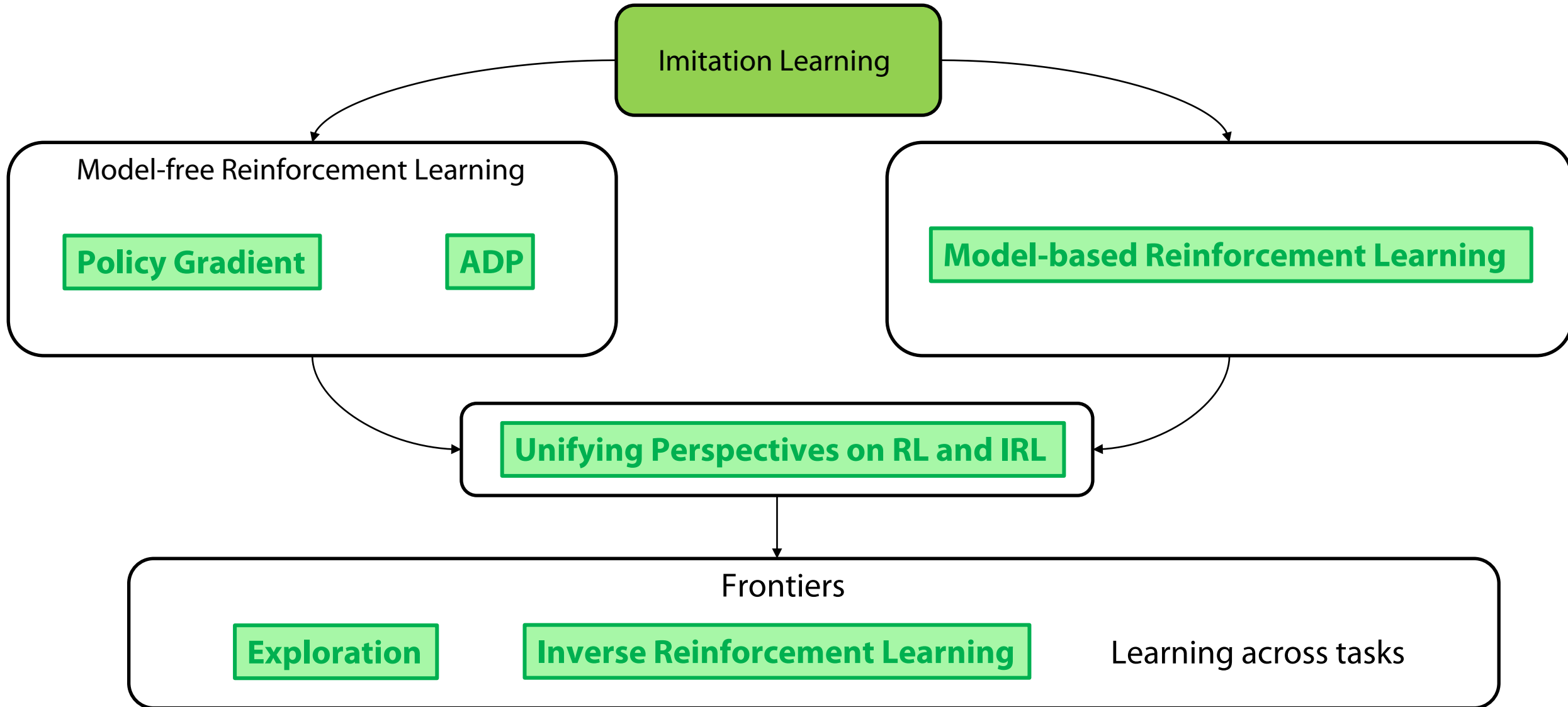
Spring 2026

Abhishek Gupta

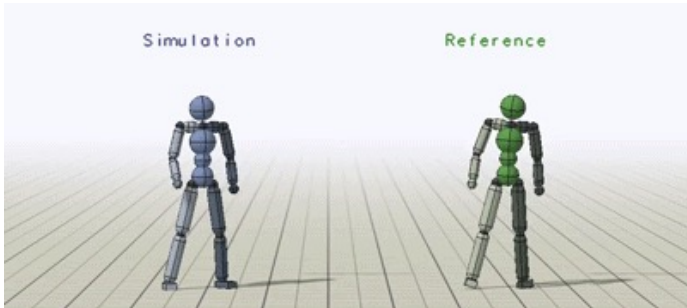
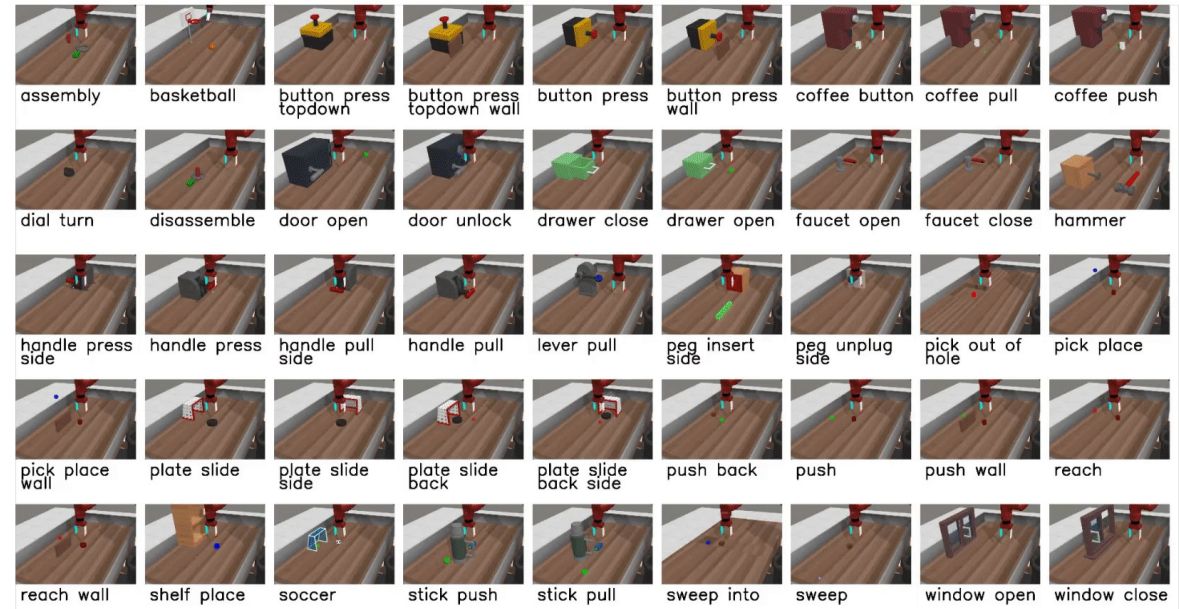
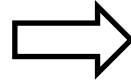
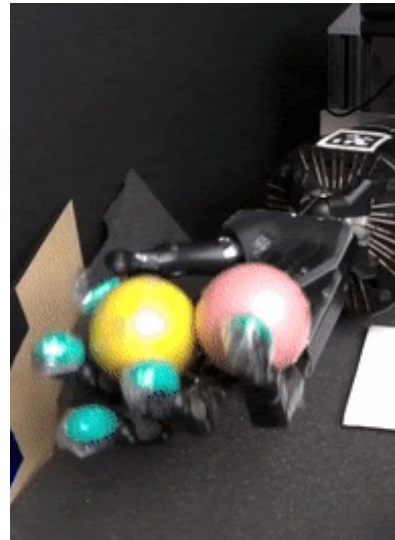
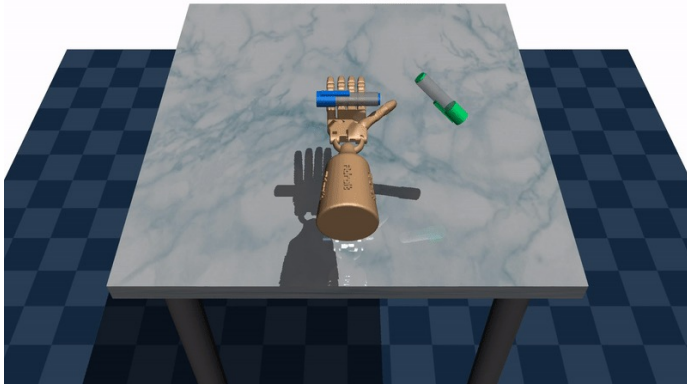
TA: Mateo Guaman Castro



Class Structure



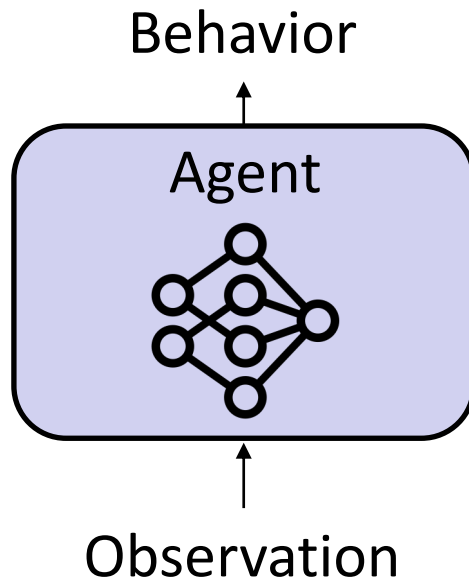
From Single Task to Multi-Task RL



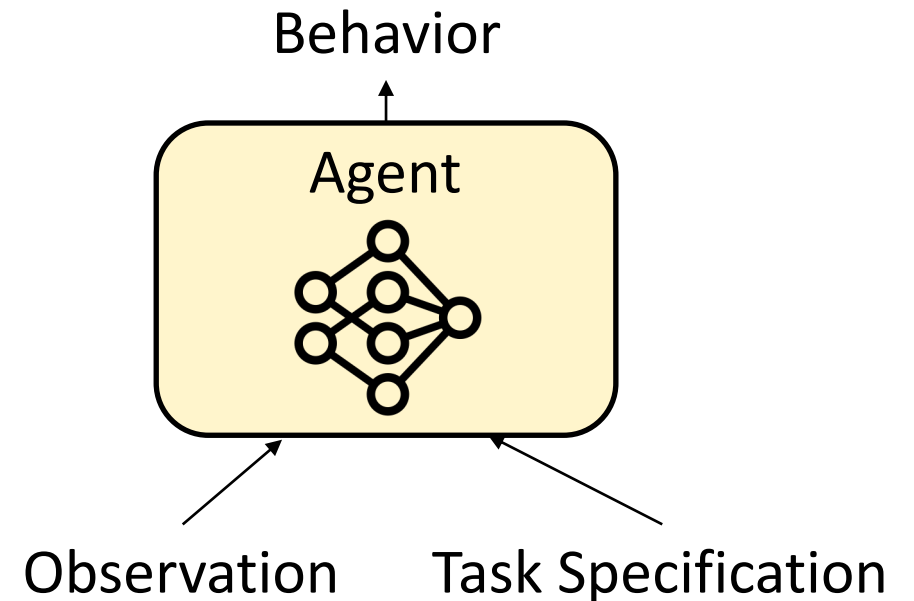
Can we make RL algorithms generalists?

We need a single agent to be able to (quickly or directly) solve multiple different tasks

Specialist RL



Generalist RL

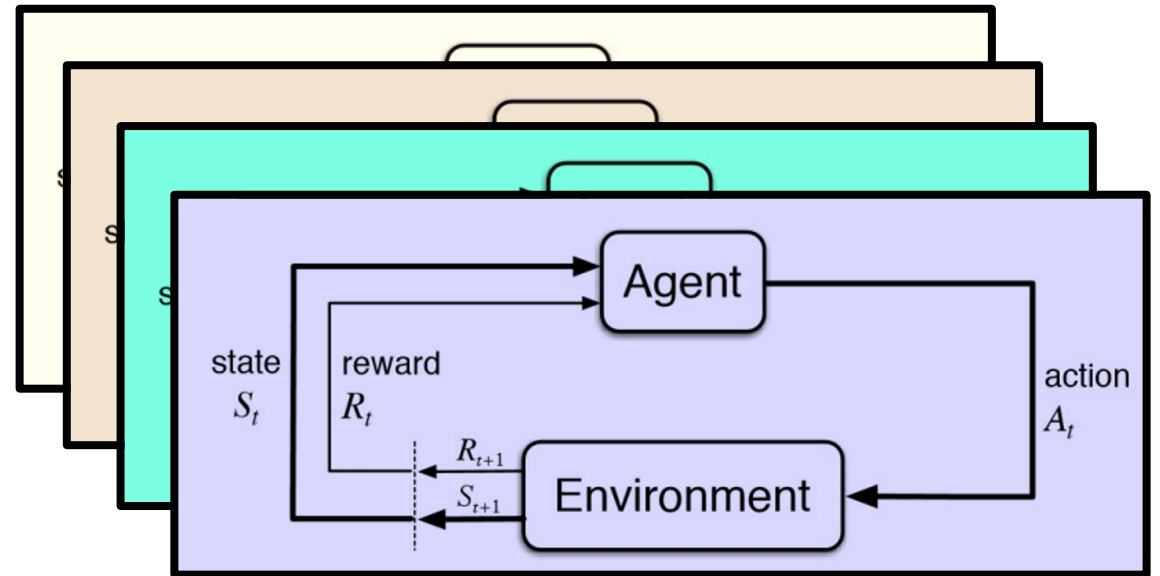
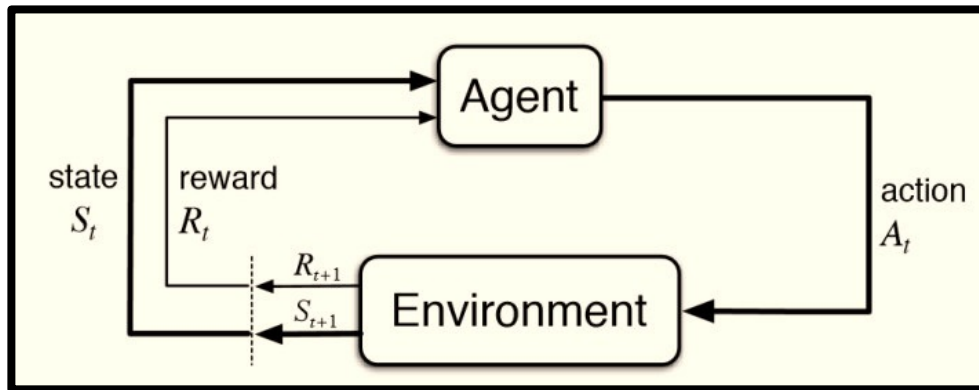


Multi-Task RL – Distribution over MDPs

Assumption: Same state/action space, varying dynamics and rewards

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

$$p(\mathcal{M}_i)$$
$$\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, \mathcal{T}_i, \mathcal{R}_i, \mu, \gamma)$$

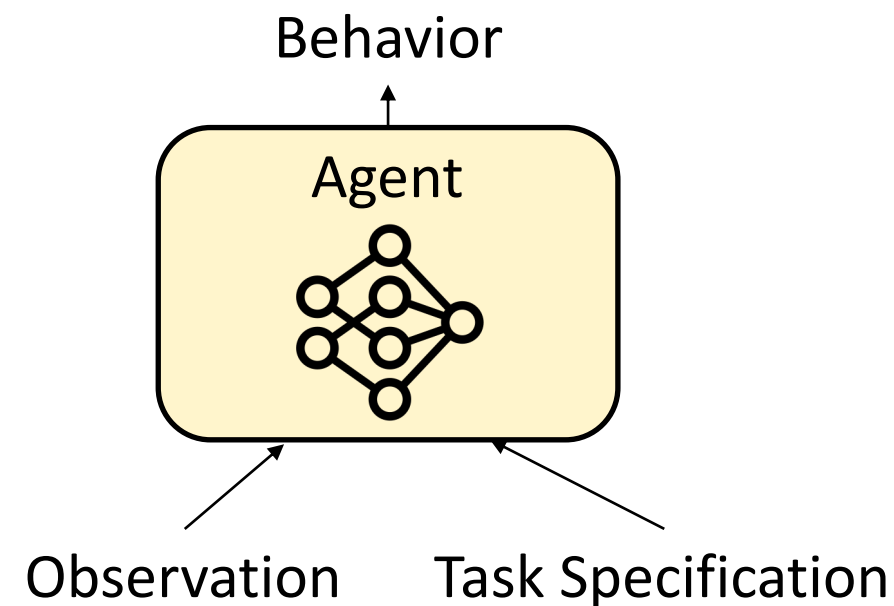
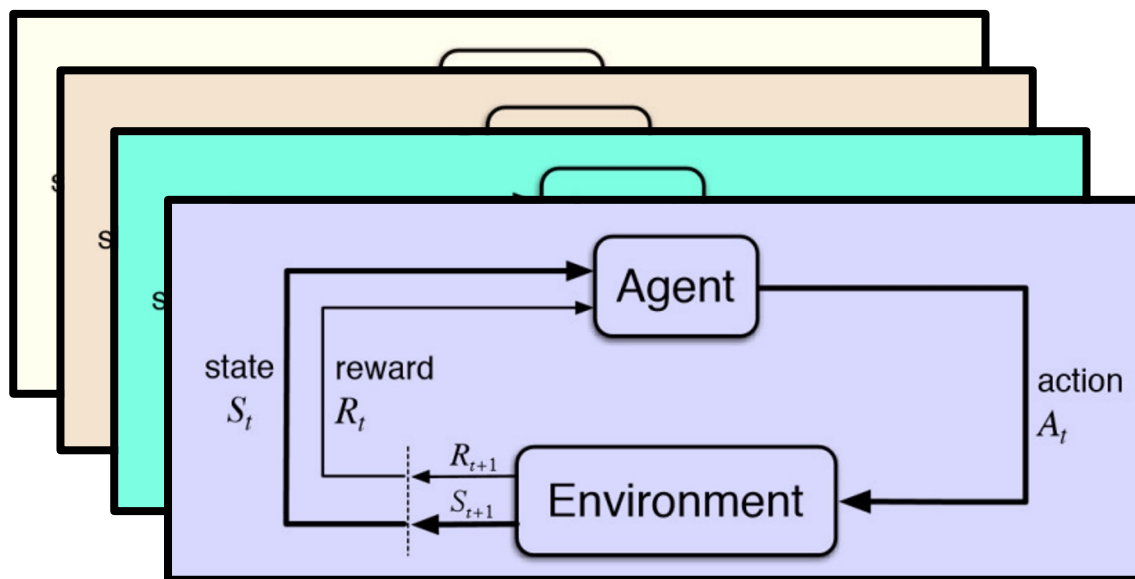


Goals for Today

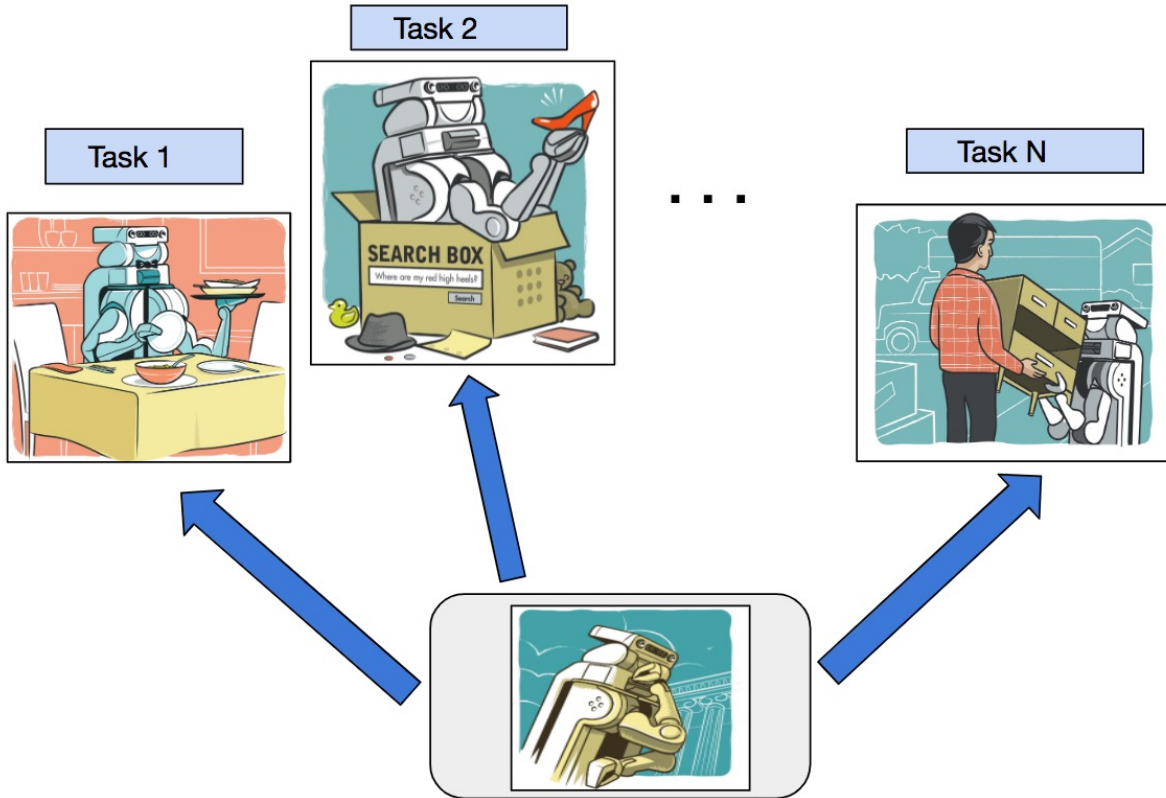
Our goal: understand different ways to solve meta-MDP/multi-task RL problem

$$p(\mathcal{M}_i)$$

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



Why should we do this?



- Learn faster by sharing data
- Generalize immediately (or quickly) to new, unseen tasks

Lecture Outline

From specialists to generalists



Multi-Task Reinforcement Learning



Meta-Reinforcement Learning



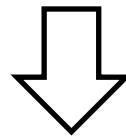
Frontiers

Multi-Task Meta-MDP

Let us assume the factor of variation across MDPs can be characterized by known ω_i
Eg: task ID, goal, video, language, ...

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

$$\mathcal{M}_i = (\mathcal{S}, \mathcal{A}, \mathcal{T}_{\omega_i}, \mathcal{R}_{\omega_i}, \mu, \gamma)$$



Slight reformulation

$$s \rightarrow (s, \omega_i)$$

$$\mathcal{T} \rightarrow p(s' | s, a, \omega_i)$$

$$\mathcal{R} \rightarrow r(s, a, \omega_i)$$

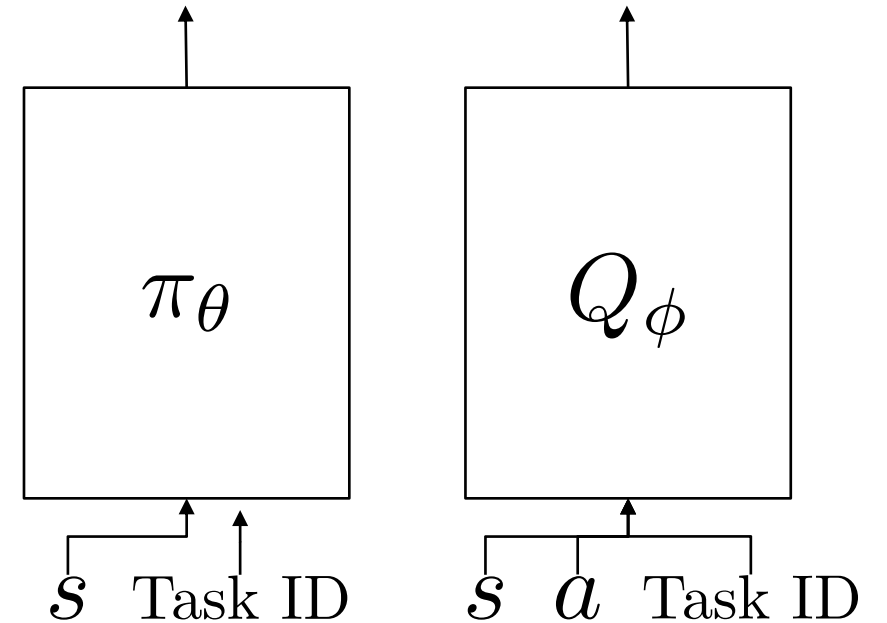
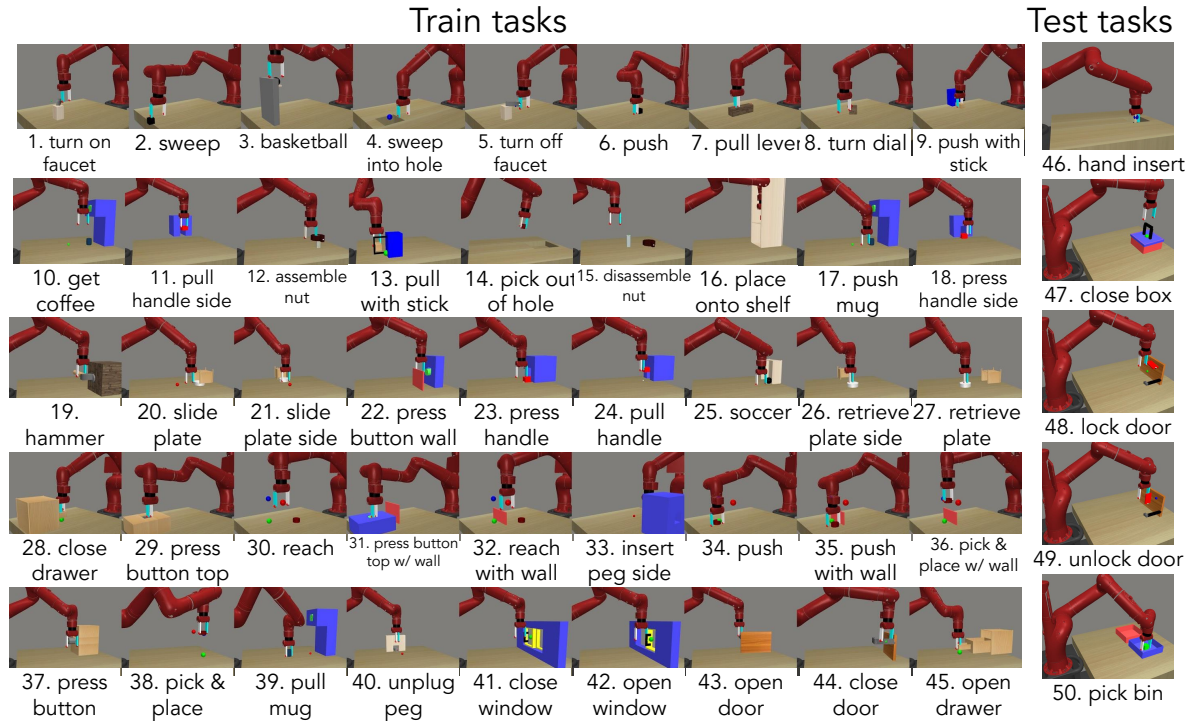
$$\mu \rightarrow \mu(s_0)p(\omega_i)$$

Key idea: Multi-task RL == Single task RL in modified MDP

Just include ω_i in state and run standard RL, solve new ω_i 0-shot

Multi-Task Actor-Critic

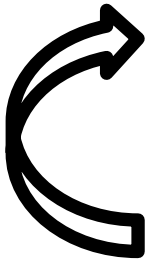
We often want to learn a single policy, Q function which can solve multiple tasks.

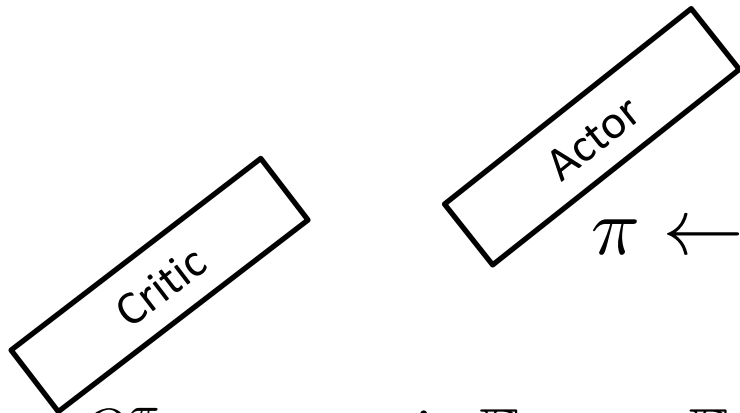


$$\max_{\phi} \mathbb{E}_{\omega \sim p(\omega)} \left[\mathbb{E}_{\pi_{\phi}(a|s, \omega)} \left[\sum_t \gamma^t r(s, a, \omega) \right] \right]$$

Template for Multi-Task RL

Canonical paradigm for doing multi-task RL via RL

- 
1. Sample data from all tasks using the same actor with different task ID
 2. Collect all data into a single batch with $(s, a, s', \text{task ID})$ pairs
 3. Perform actor and critic updates on the shared actor and critic with losses summed up across tasks



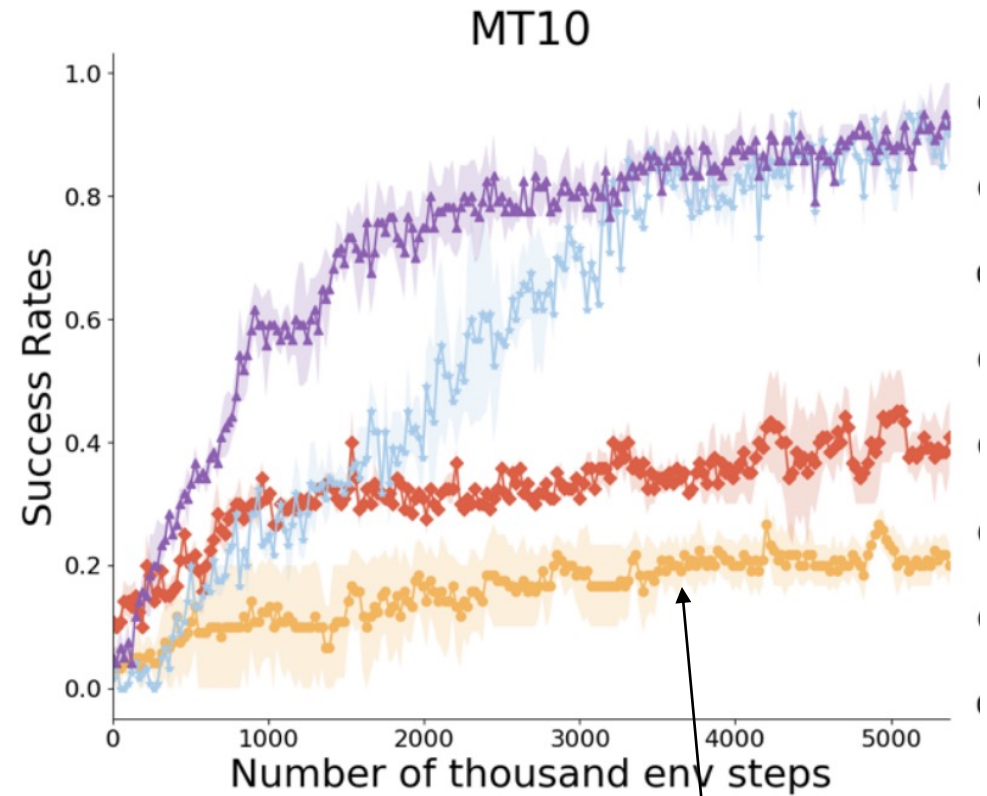
$$\pi \leftarrow \arg \max \mathbb{E}_{\tau \sim p(\tau)} \mathbb{E}_{a \sim \pi} [Q^\pi(s, a, \tau)]$$

$$Q^\pi \leftarrow \arg \min \mathbb{E}_{\tau \sim p(\tau)} \mathbb{E}_{(s, a, s') \sim p} [(Q(s, a, \tau) - (r(s, a, \tau) + \gamma \mathbb{E}_{a' \sim \pi(\cdot | s', \tau)} Q(s', a', \tau)))^2]$$

Does it work?

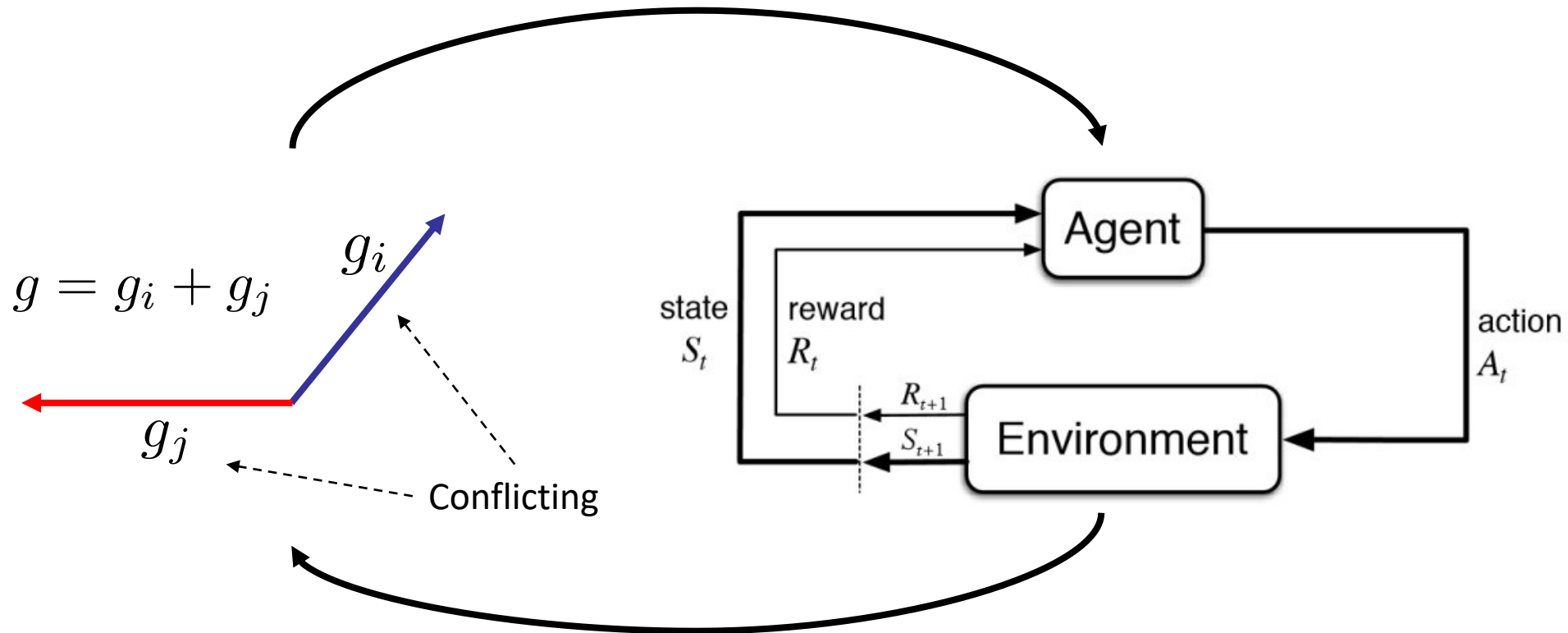
Let's not even study generalization, let's understand if this fits the train set

Methods	MT50
Multi-task PPO	8.98%
Multi-task TRPO	22.86%
Task embeddings	15.31%
Multi-task SAC	28.83%
Multi-task multi-head SAC	35.85%



Why is it hard to do Multi-Task RL?

Gradients from different tasks often conflict and hamper performance of all tasks, especially when coupled with exploration

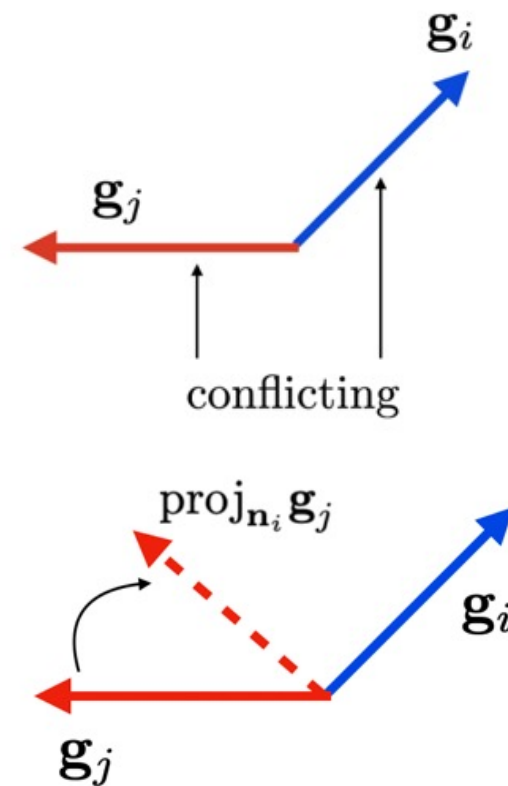
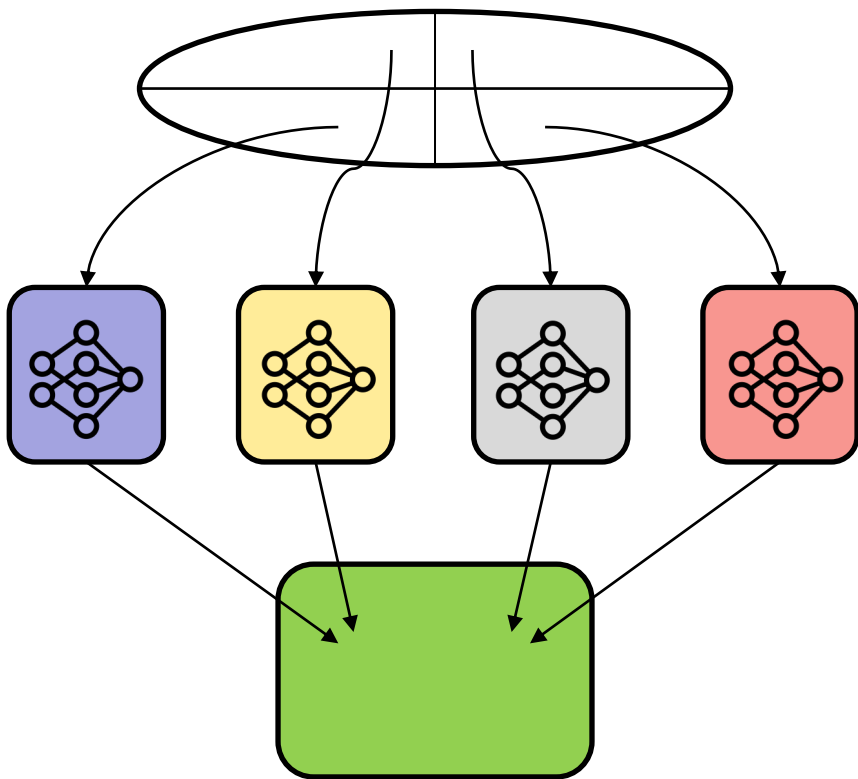


How can we deal with gradient interference in RL?

If issue is exploration + conflicting gradients is bad

Idea 1: Remove exploration from MTRL

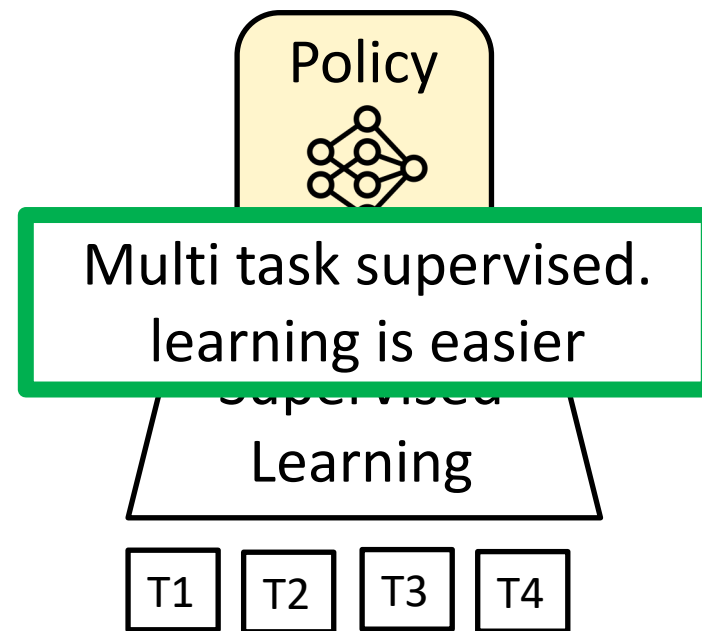
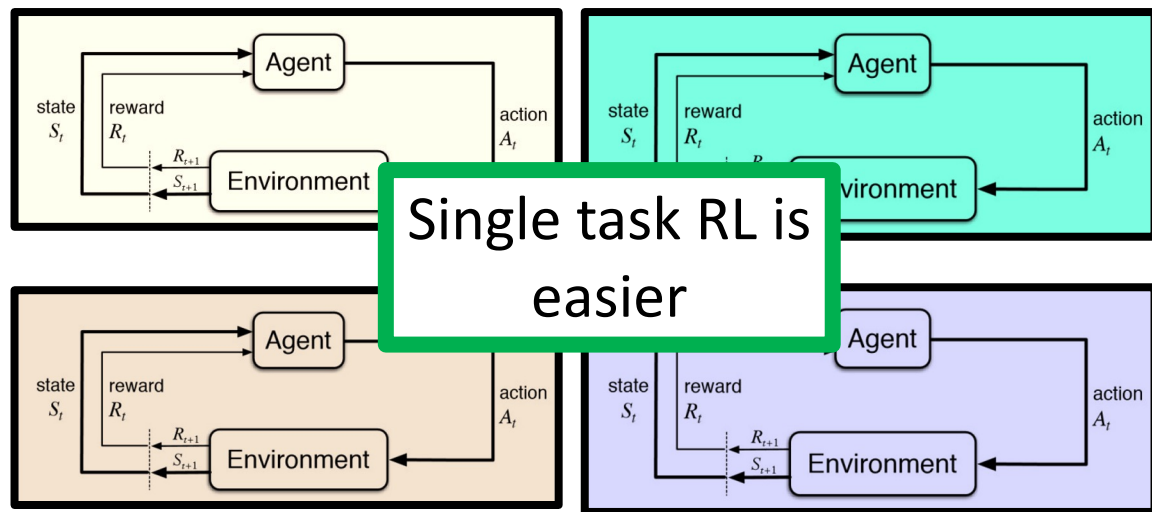
Idea 2: Modify gradients



Resolving Gradient Interference with Distillation

Empirical observation:

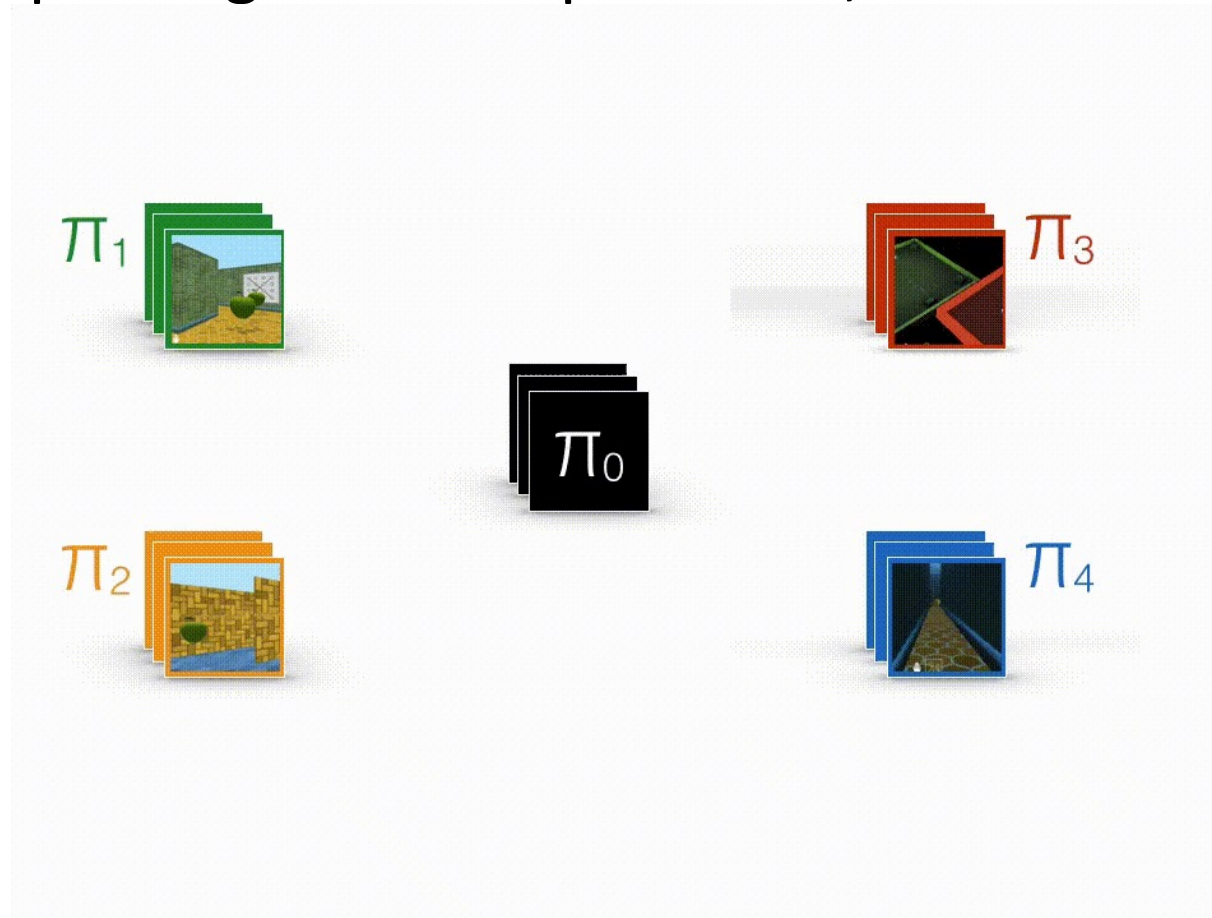
Multi-task SL (no exploration) is stable, multi-task RL (exploration) is unstable



Idea: convert multi-task RL into single task RL + multi task SL

Divide and Conquer Approach to RL

Divide into multiple single task RL problems, “distill” into a single solution



Single task RL \rightarrow standard RL

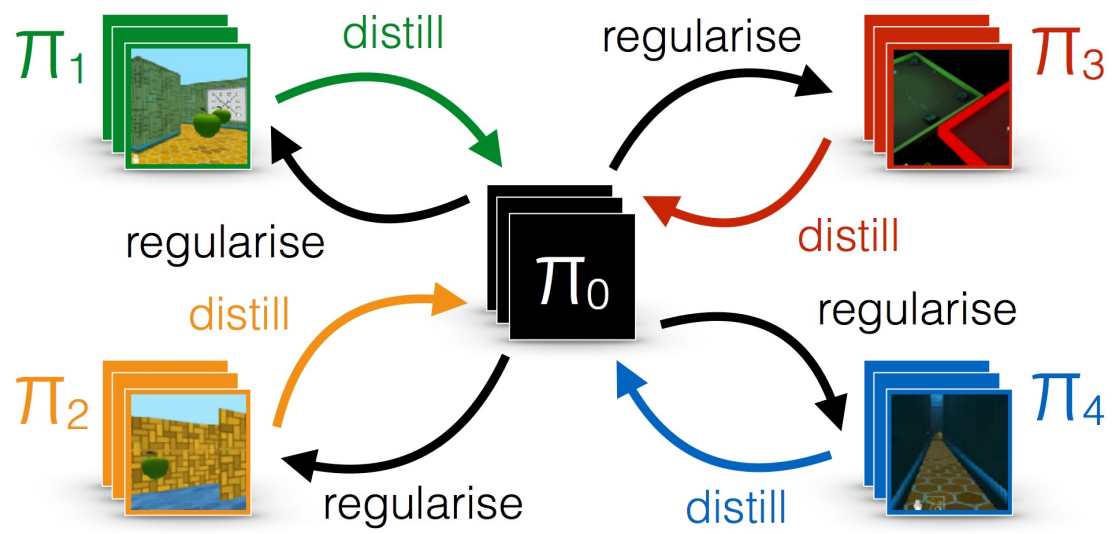
Distillation \rightarrow supervised learning

Divide and Conquer RL: Mathematical Formulation

Shared policy Per-task policy

$$J(\pi_0, \{\pi_i\}_{i=1}^n) = \sum_i \mathbb{E}_{\pi_i} \left[\sum_{t \geq 0} \gamma^t R_i(a_t, s_t) - c_{\text{KL}} \gamma^t \log \frac{\pi_i(a_t | s_t)}{\pi_0(a_t | s_t)} - c_{\text{Ent}} \gamma^t \log \pi_i(a_t | s_t) \right]$$

Per-task reward Match shared/individual policy Entropy



Single task RL \rightarrow standard RL

$$\max_{\pi_i} J(\pi_0, \{\pi_i\}_{i=1}^n)$$

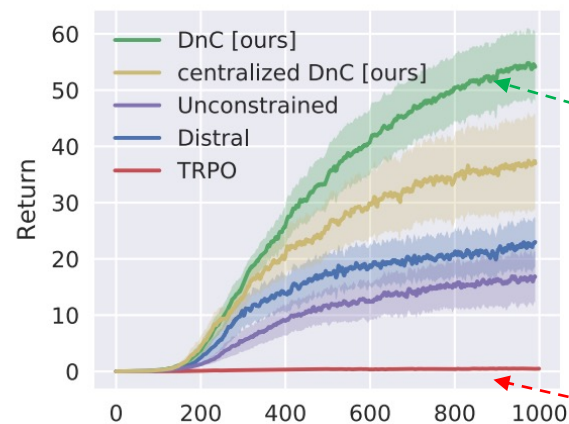
Distillation \rightarrow supervised learning

$$\max_{\pi_0} J(\pi_0, \{\pi_i\}_{i=1}^n)$$

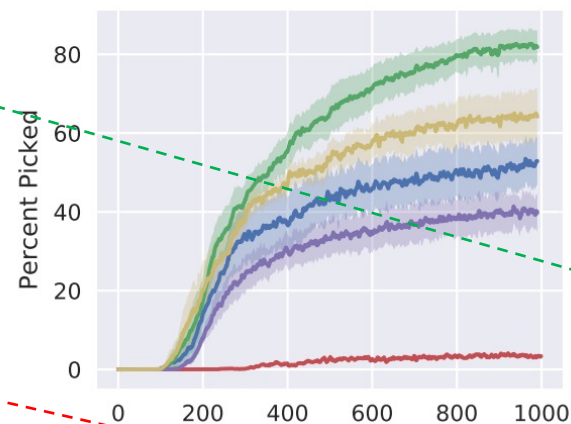
Experimental Validation



(a) *Picking task*

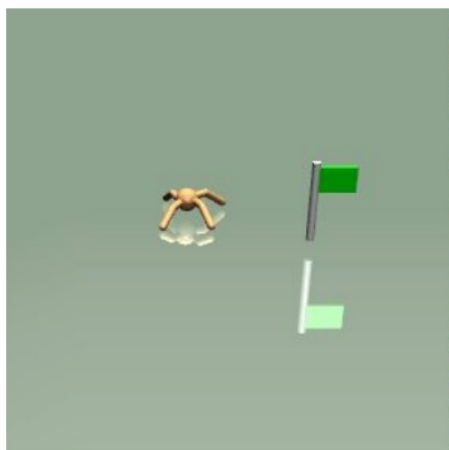


(b) Average Return on *Picking*

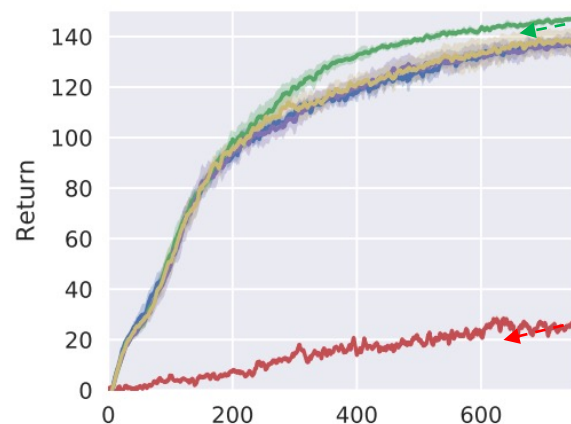


(c) Success rate on *Picking*

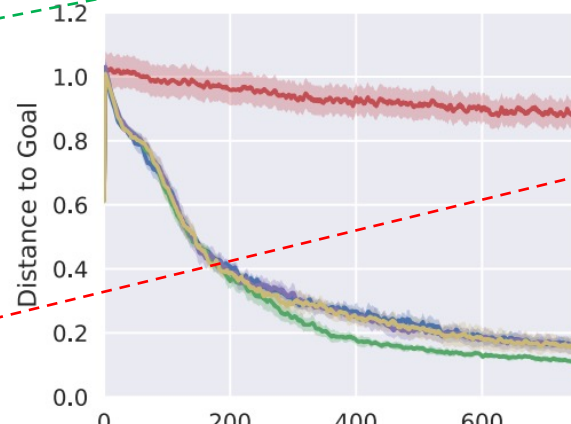
Divide and Conquer



(j) *Ant*



(k) Average Return on *Ant*



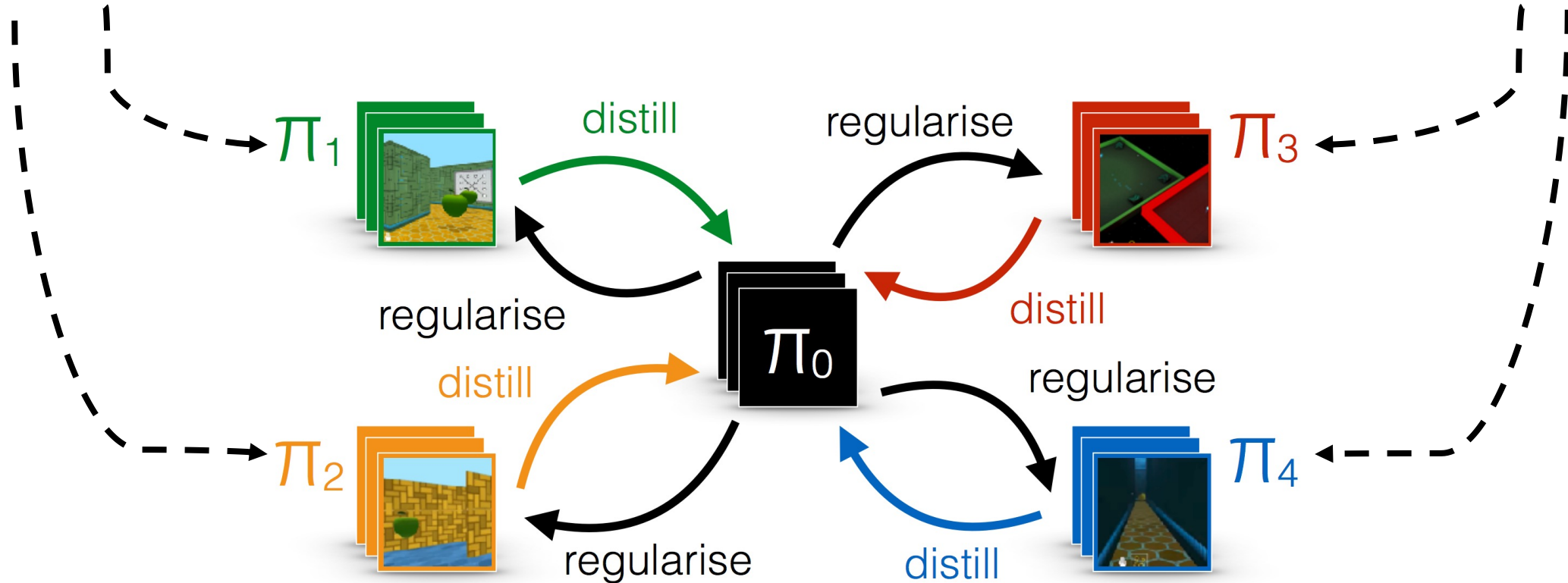
(l) Distance from goal on *Ant*

Monolithic RL

Divide and Conquer Reinforcement Learning

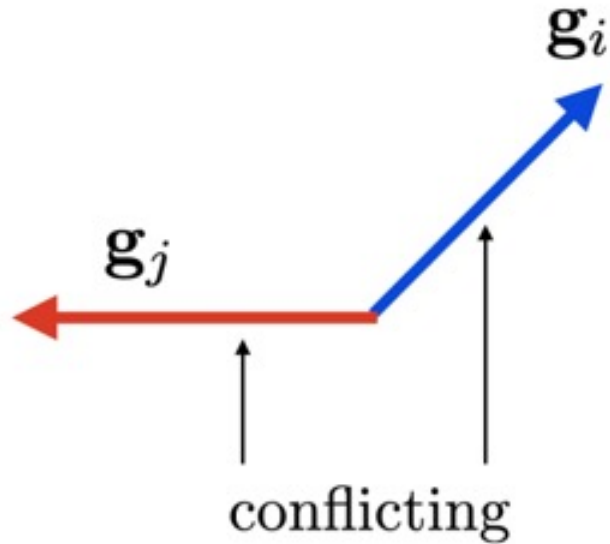
Is this enough?

Lot of the learning is done independently, limited data/parameter sharing



Can we do better?

What if we directly modified the gradients?



Replace g_i by g_i'

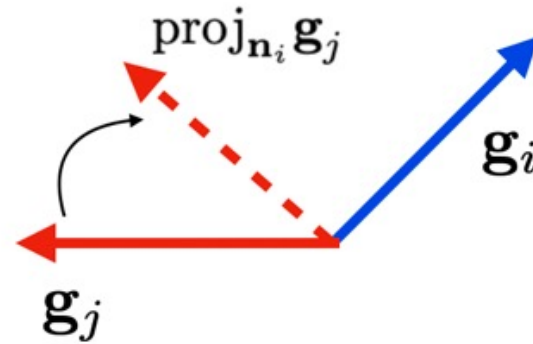
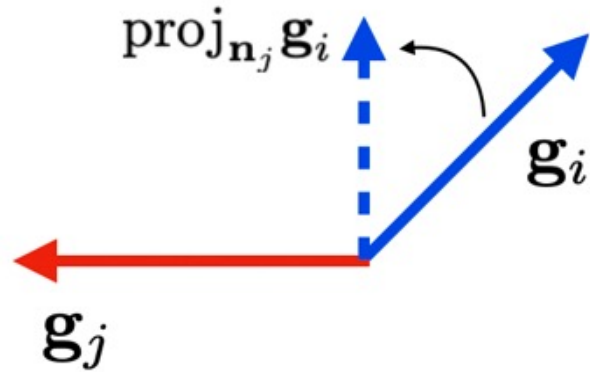
Replace g_j by g_j'

What should g_i' and g_j' be?

Idea: When gradients conflict, project them to deconflict

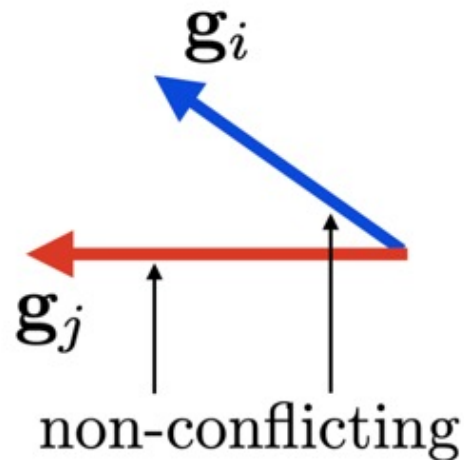
Deconflicting gradients with PCGrad

If gradients conflict: project them onto the normal plane



$$g_i = g_i - \frac{g_i \cdot g_j}{\|g_j\|^2} \cdot g_j$$

Otherwise: leave them alone

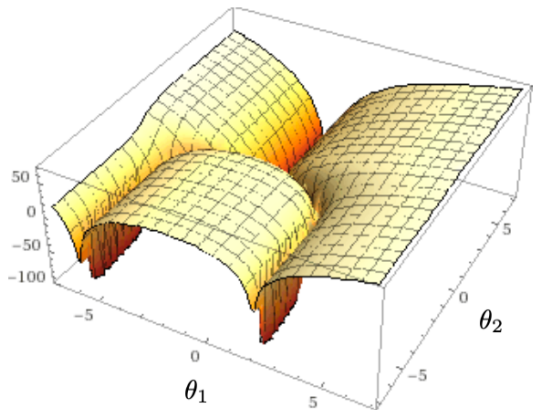


Does this empirically help?

$$\mathcal{L}_1(\theta) = 20 \log(\max(|.5\theta_1 + \tanh(\theta_2)|, 0.000005))$$

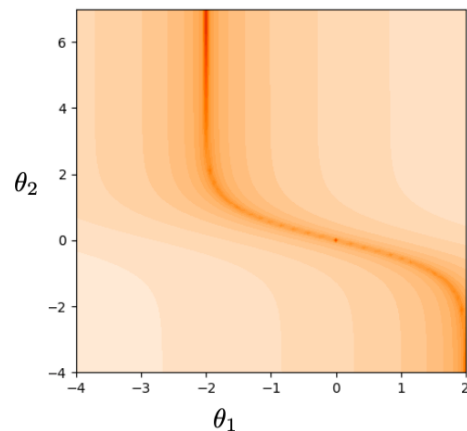
$$\mathcal{L}_2(\theta) = 25 \log(\max(|.5\theta_1 - \tanh(\theta_2) + 2|, 0.000005))$$

Multi-Task Objective



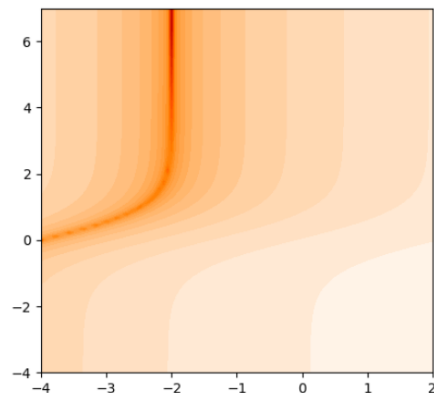
(a)

Task 1 Objective



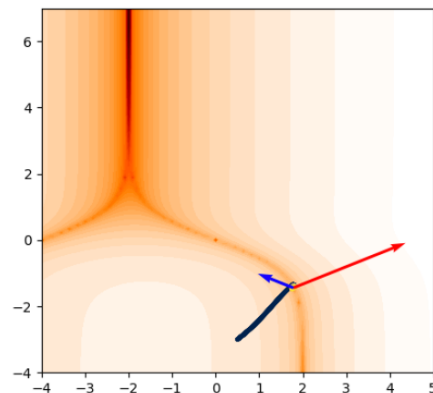
(b)

Task 2 Objective



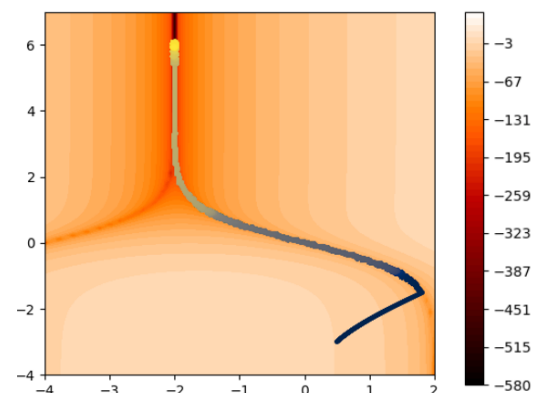
(c)

Adam



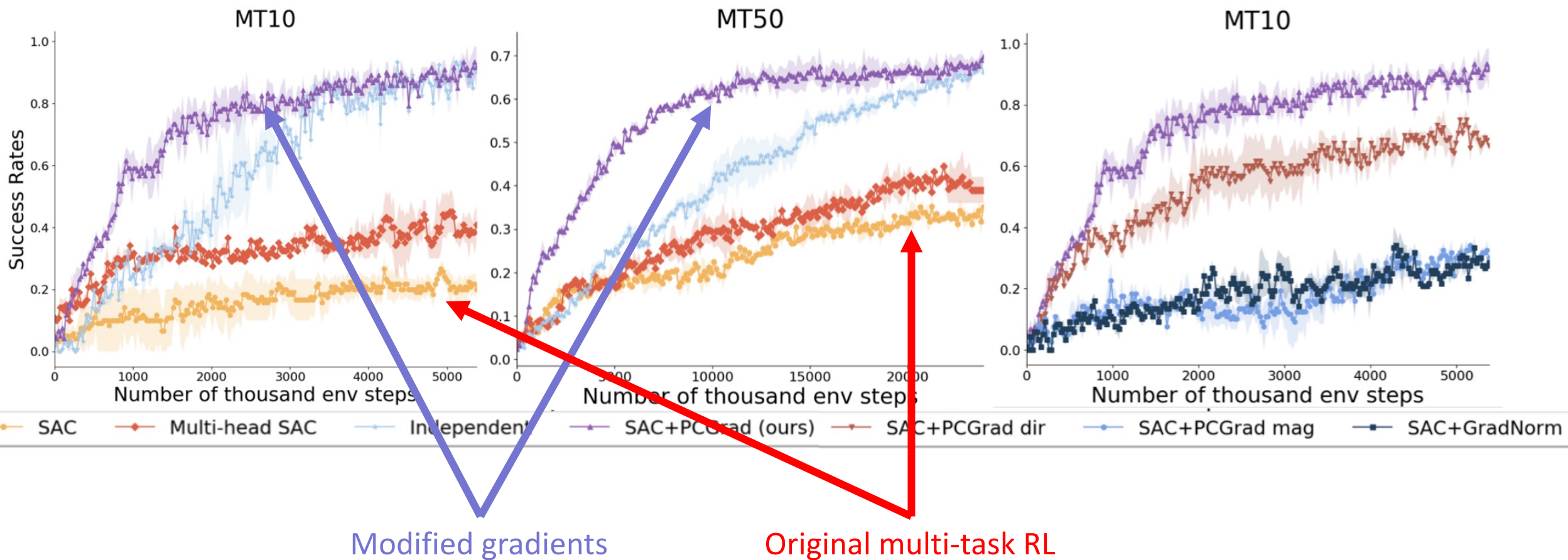
(d)

Adam + PCGrad



(e)

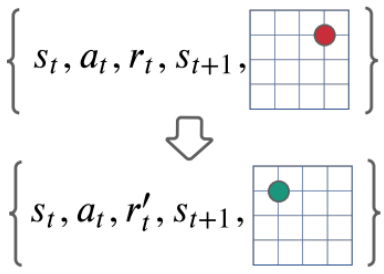
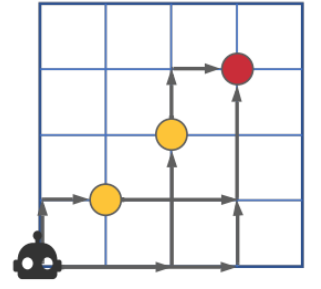
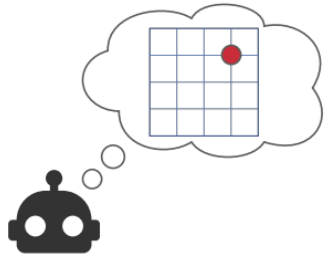
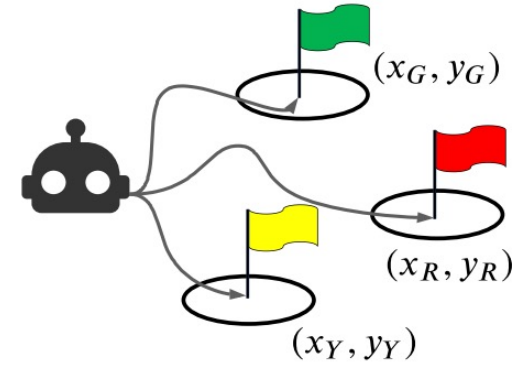
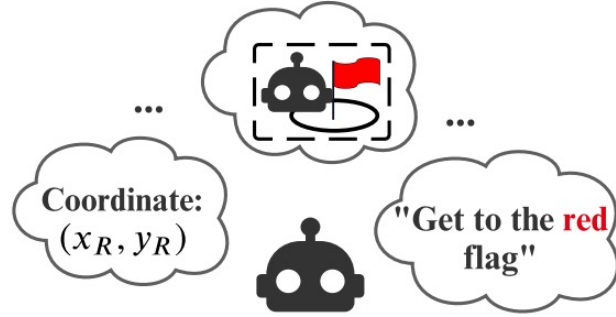
Does this empirically help?



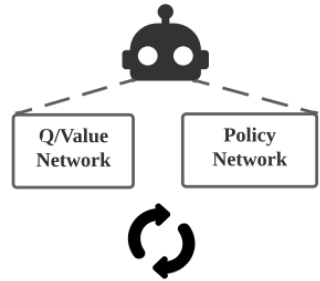
Special Class of Multi-Task RL: Goal-Conditioned

Tasks = reaching particular state g

Shortest path from s to g



Relabel



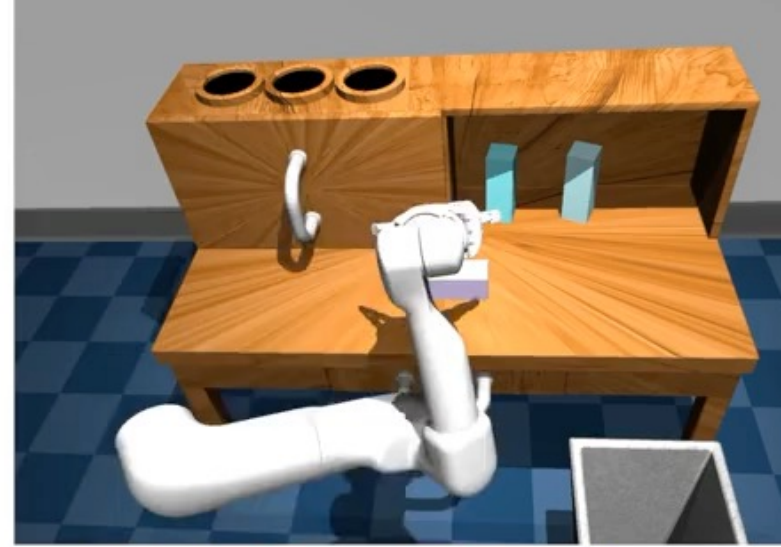
Suboptimal data is optimal for **some** task

→ Goal = reached state

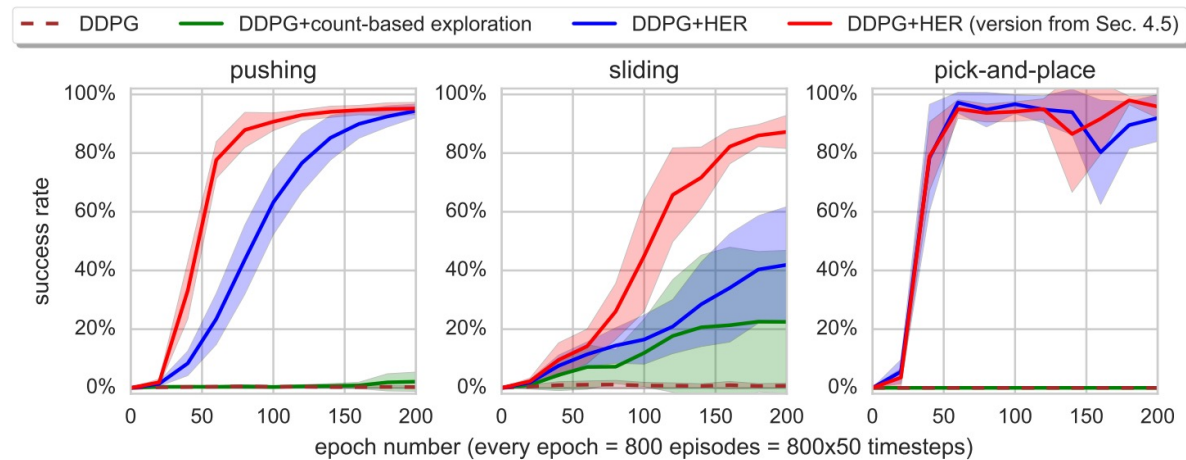
Goal-Conditioned RL via Hindsight



Goal

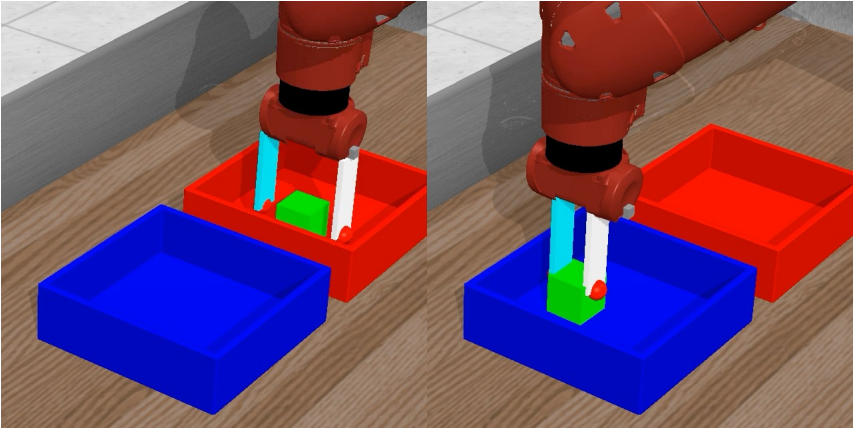
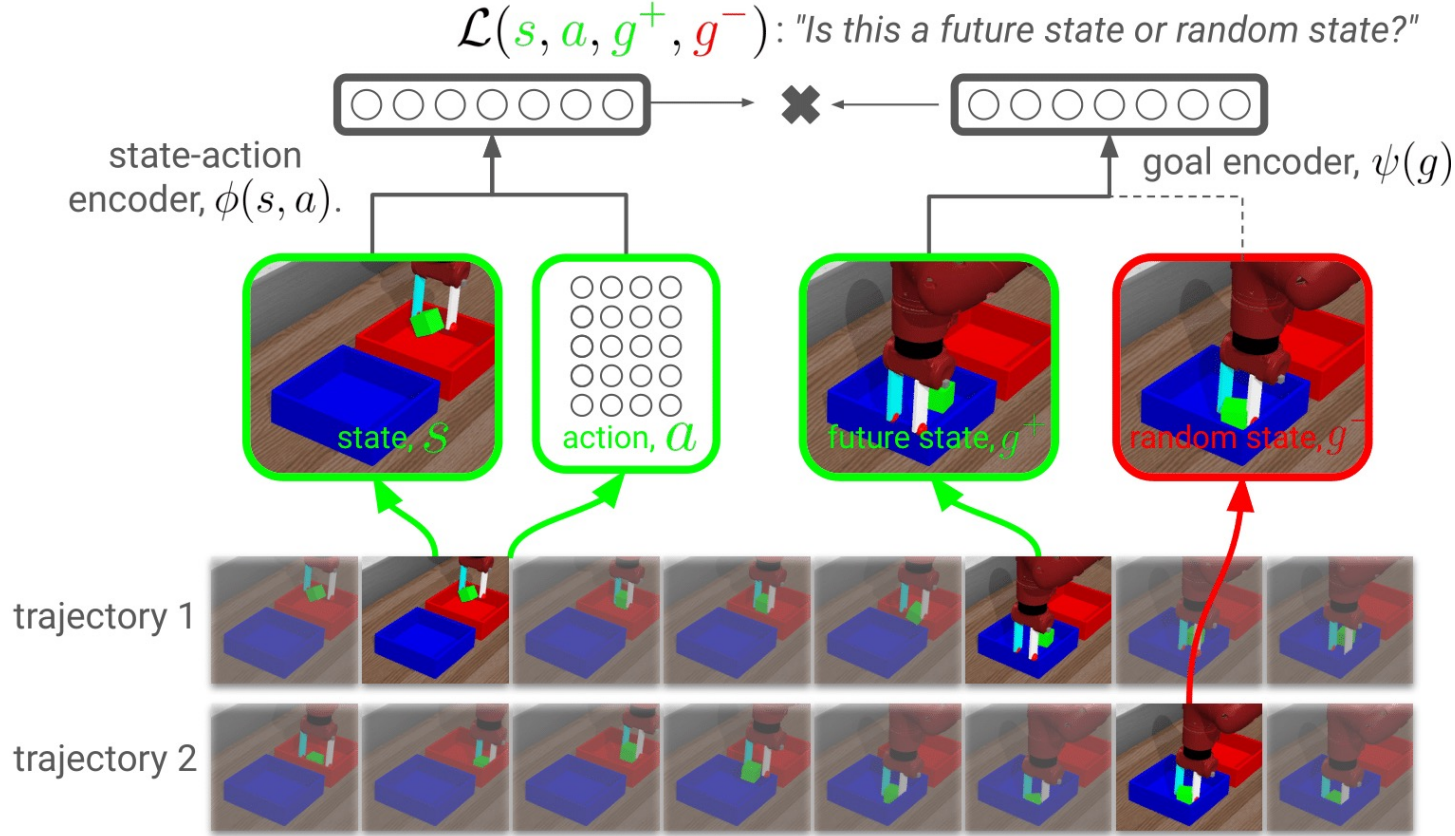


Single Play-LMP policy

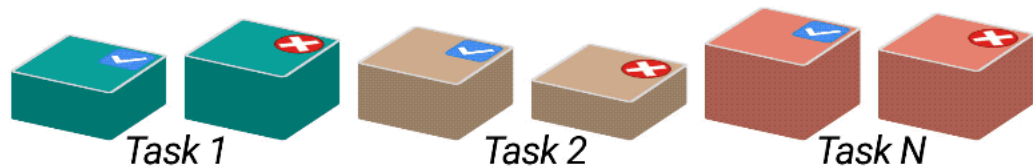
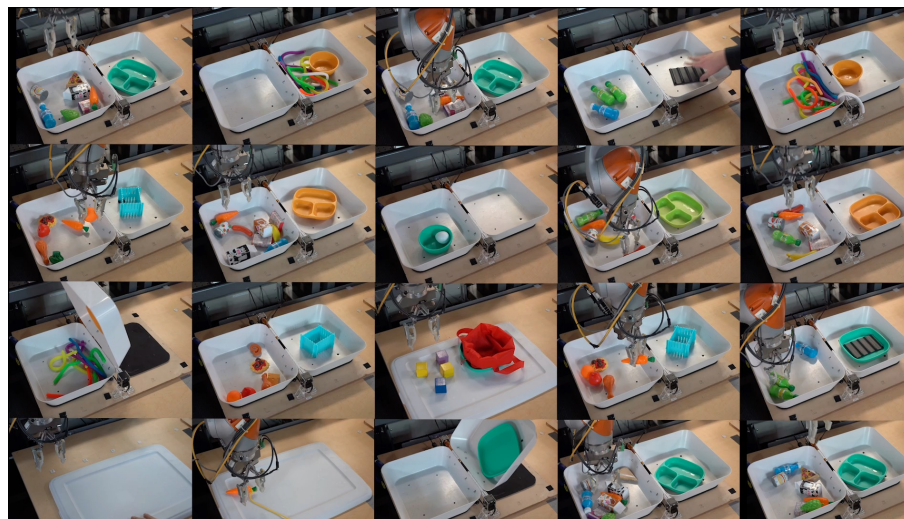
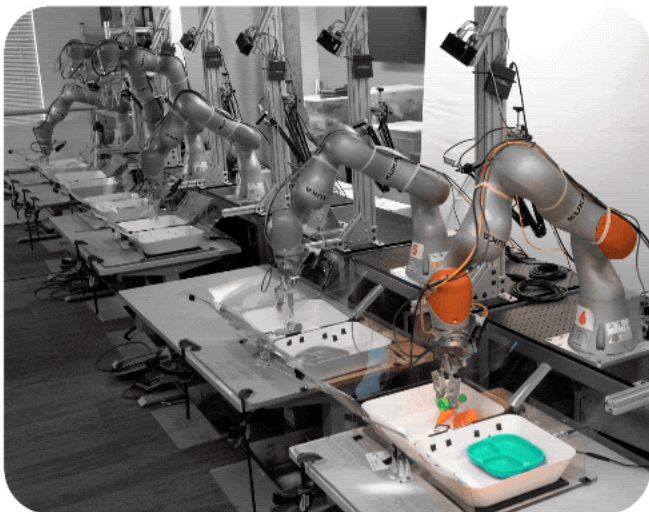


Goal-Conditioned RL affords Interesting Algorithms

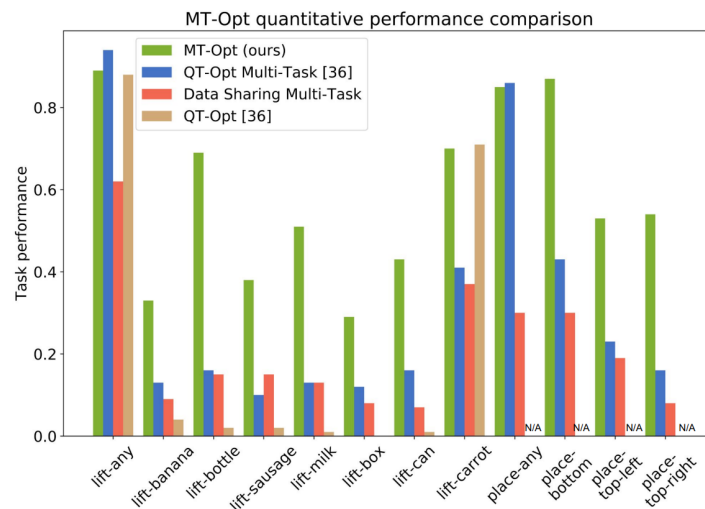
Can cast goal-conditioned RL as a contrastive learning problem!



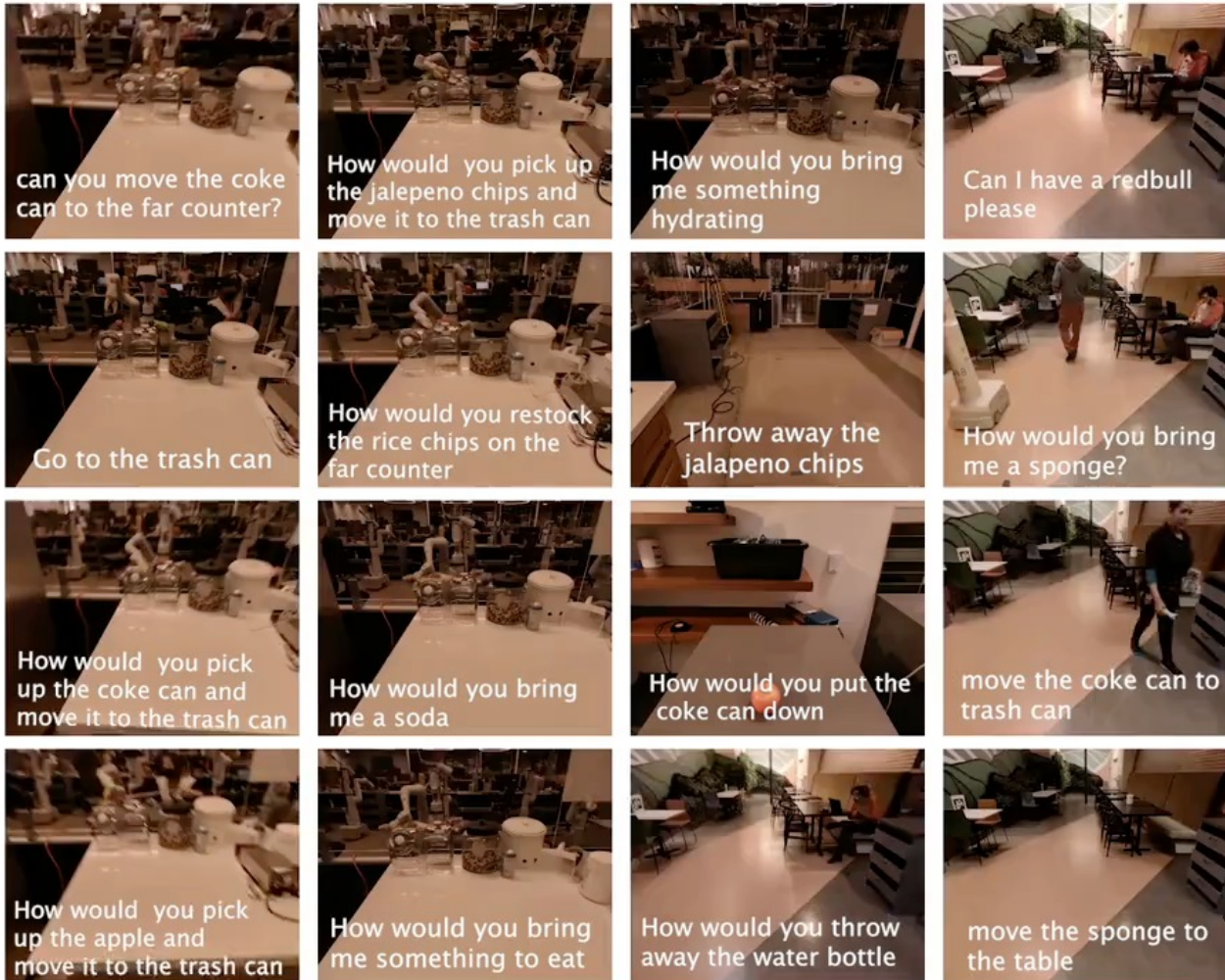
So multi-task RL is pretty cool, does it work?



 MT-Opt training



So multi-task RL is pretty cool, does it work?



ω_i can be language too!

Takeaways

1. Multi-task RL solves a contextual meta-MDP for 0-shot generalization
 - Can help with efficiency and generalization
2. Optimization in multi-task RL can be challenging:
 - Gradient interference during optimization
 - Winner take all during optimization
3. Solutions to multi-task optimization include:
 - Divide and conquer
 - Gradient projection
 - ...

Lecture Outline

From specialists to generalists



Multi-Task Reinforcement Learning



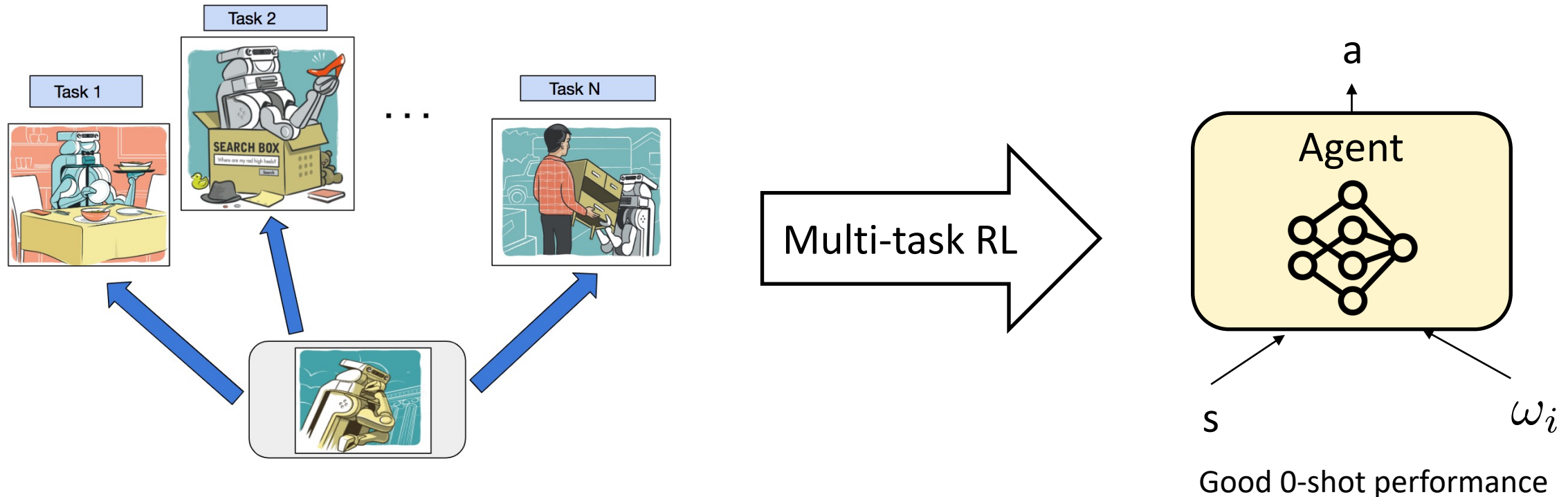
Meta-Reinforcement Learning



Frontiers

Recap: Multi-task RL Setup, 0-shot generalization

Factor of variation across MDPs can be characterized by ω_i , which is known
Eg: task ID, goal, video, language, ...

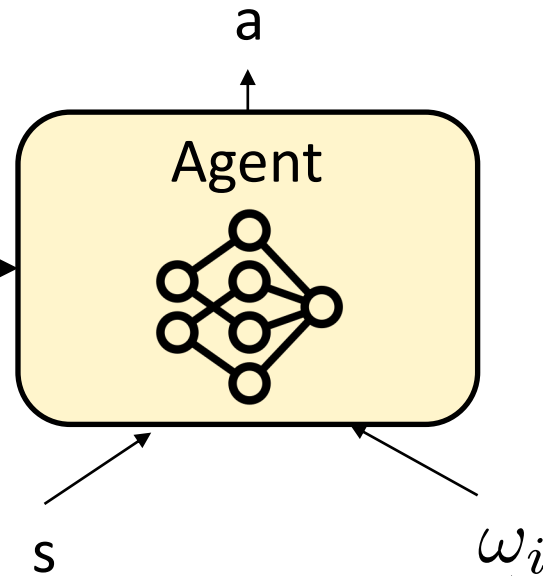


When is this not enough?

From 0-shot learning to few-shot learning

Factor of variation across MDPs can be characterized by ω_i , which is known
Eg: task ID, goal, video, language, ...

Policy is not good enough 0-shot, need to finetune

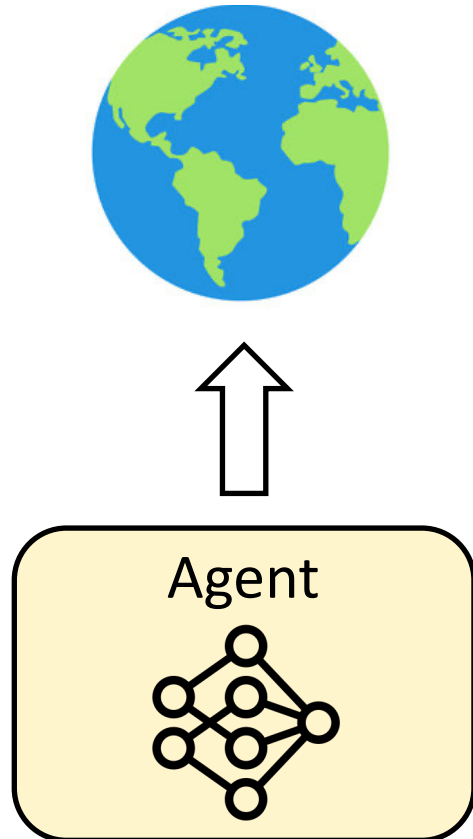


Context is unknown or hard to specify analytically

When is this not enough?

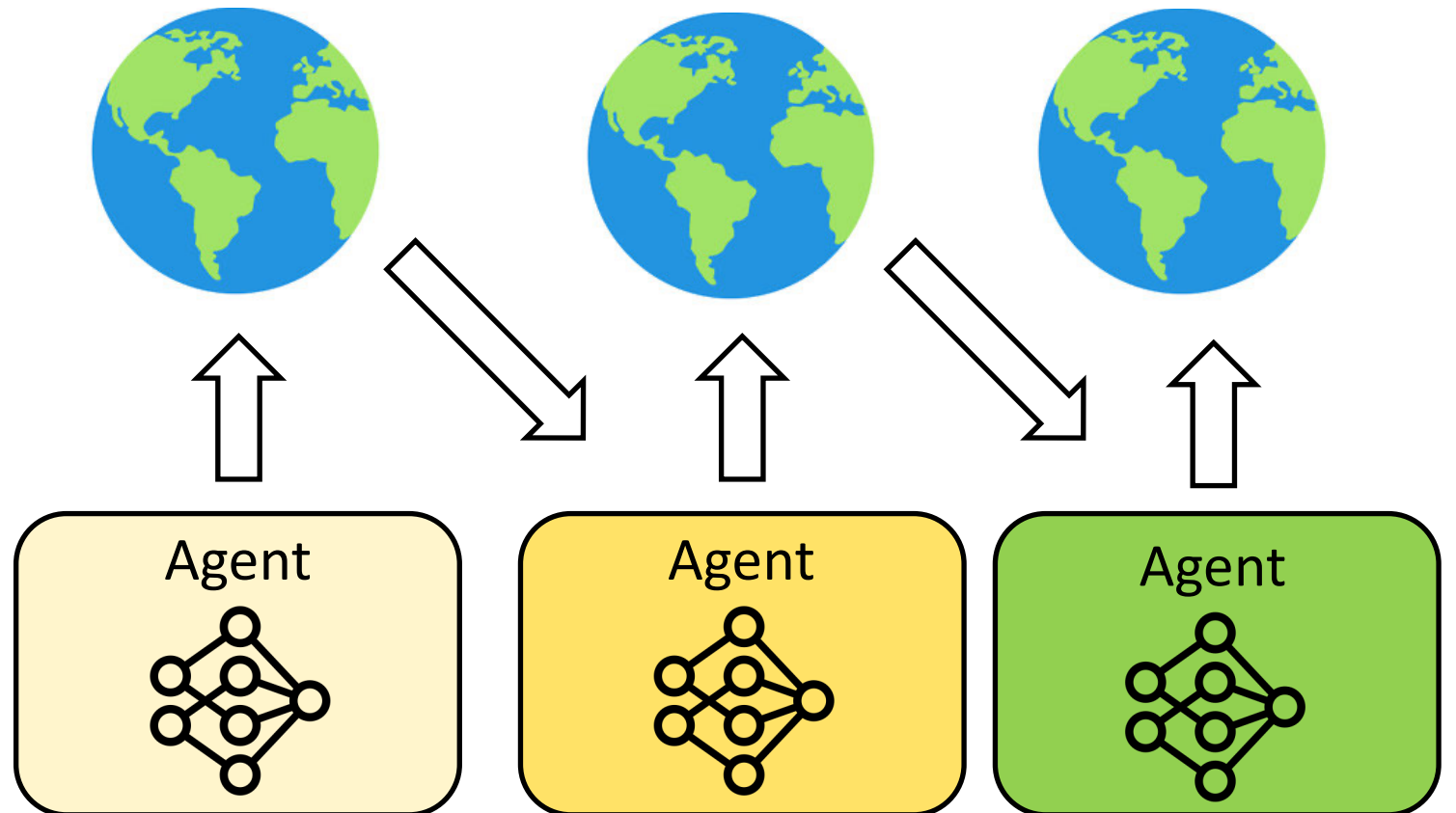
From 0-shot learning to few-shot learning

0-shot MTRL: No experience at test time

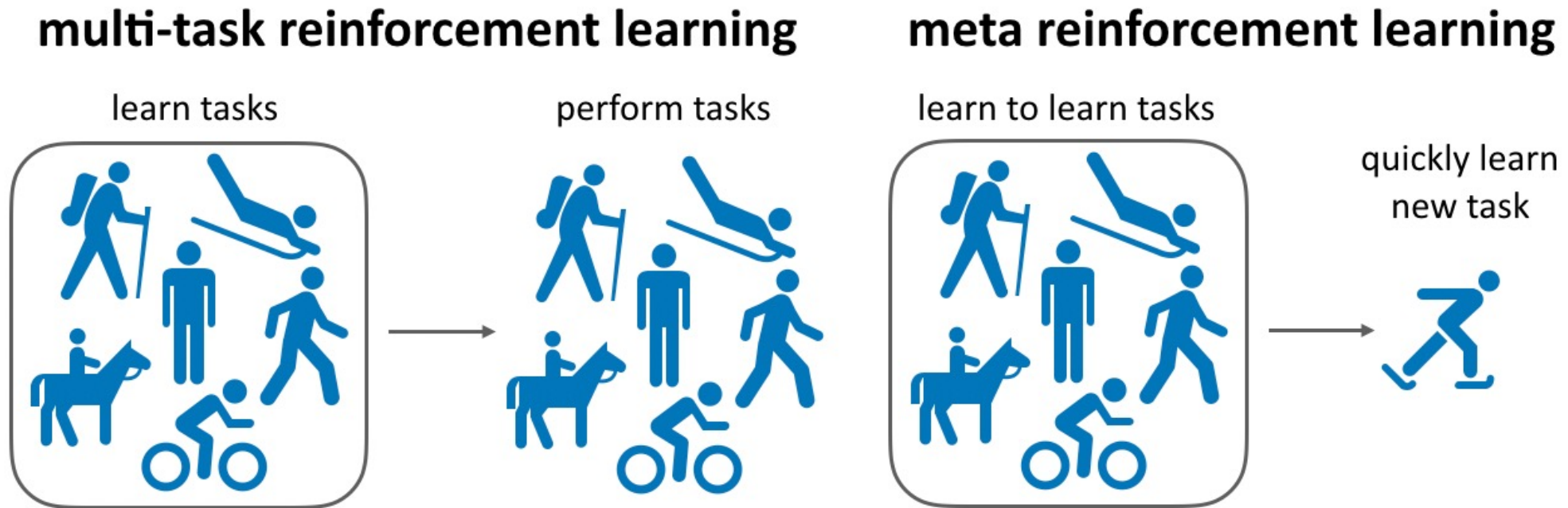


Meta-RL: Small amount of experience at test time

Fast adaptation with experience

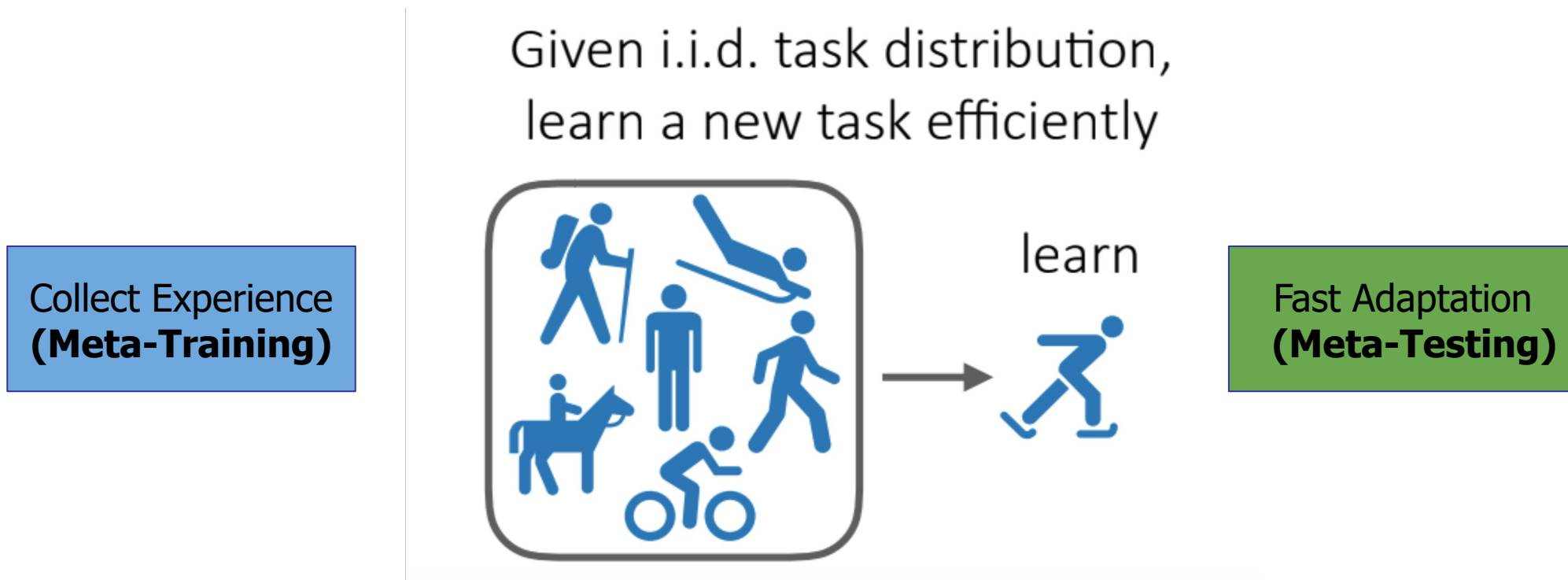


Connection to Contextual Multi-Task RL



- Multi-task policy evaluates 0-shot performance
- Meta-RL trains for good k-shot policy by “learning to learn”

Meta-Learning Problem for RL



- Given a distribution over tasks $p(\tau)$, learn an update function f_{θ} that can learn tasks drawn from $p(\tau)$ quickly!
- Leverage regularity across tasks to optimize for a fast RL algorithm

Meta-Learning Problem for RL

Standard RL:

Single reward function, single dynamics $\arg \max_{\theta} \mathbb{E}_{\pi_{\theta}} \left[\sum_t r(s_t, a_t) \right]$

Meta RL:

Distribution of tasks $p(\tau)$, optimize for update function f_{θ}

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right]$$

Encourages quick update

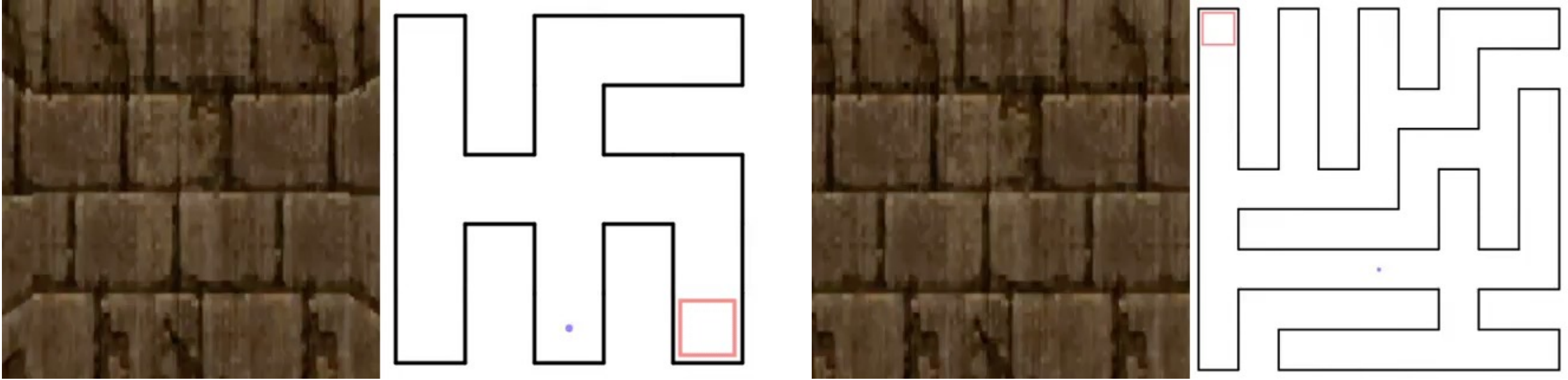
Per-task updated policy

where $\phi_i = f_{\theta}(\mathcal{D}_{\tau})$

Shared update function



Intuition behind Meta-RL




- Leverage regularity in task distribution to speed up learning
- Explore for some time before exploiting
- Minimizes regret not just maximizes reward

General Structure of Meta-RL Algorithms

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right] \longleftarrow \text{Outer loop}$$

$$\text{where } \phi_i = f_{\theta}(\mathcal{D}_{\tau}) \longleftarrow \text{Inner loop}$$

- 
1. Sample a batch of tasks from $p(\tau)$
 2. collect data pre-update
 3. Compute update according to $\phi_i = f_{\theta}(\mathcal{D}_{\tau})$
 4. Sample data from ϕ_i post-update to evaluate the update
 5. Optimize for update function f_{θ}

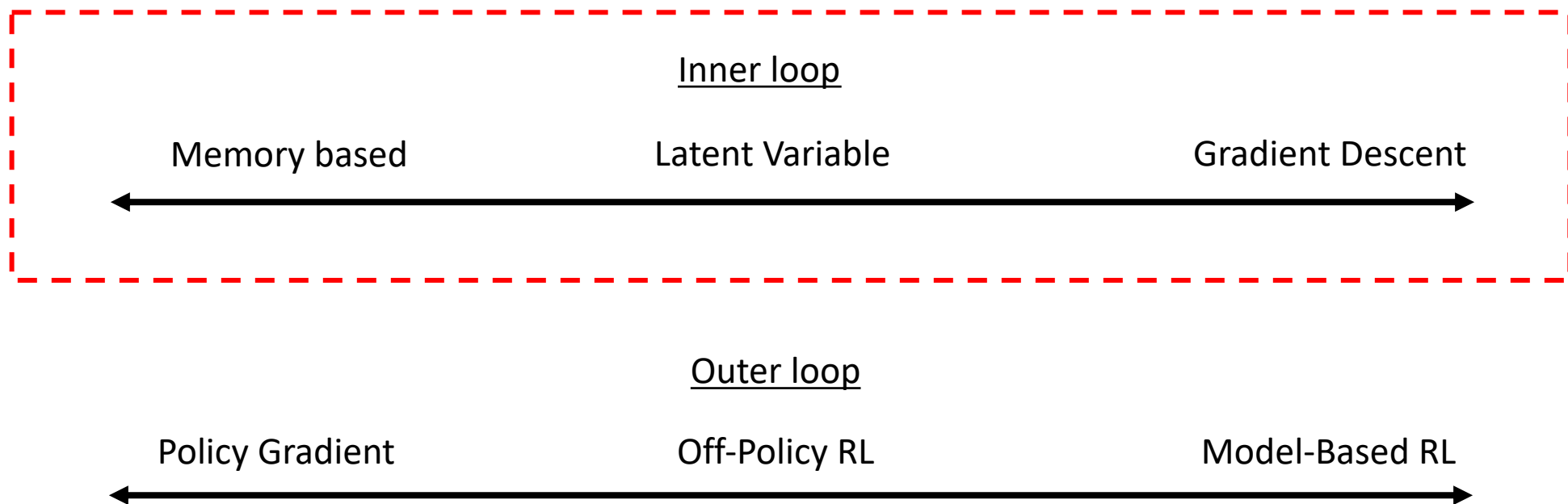
Solution Techniques for Meta-RL Problems

Main design choices:

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right] \longleftarrow \text{Outer loop}$$

where $\phi_i = f_{\theta}(\mathcal{D}_{\tau}) \longleftarrow \text{Inner loop}$

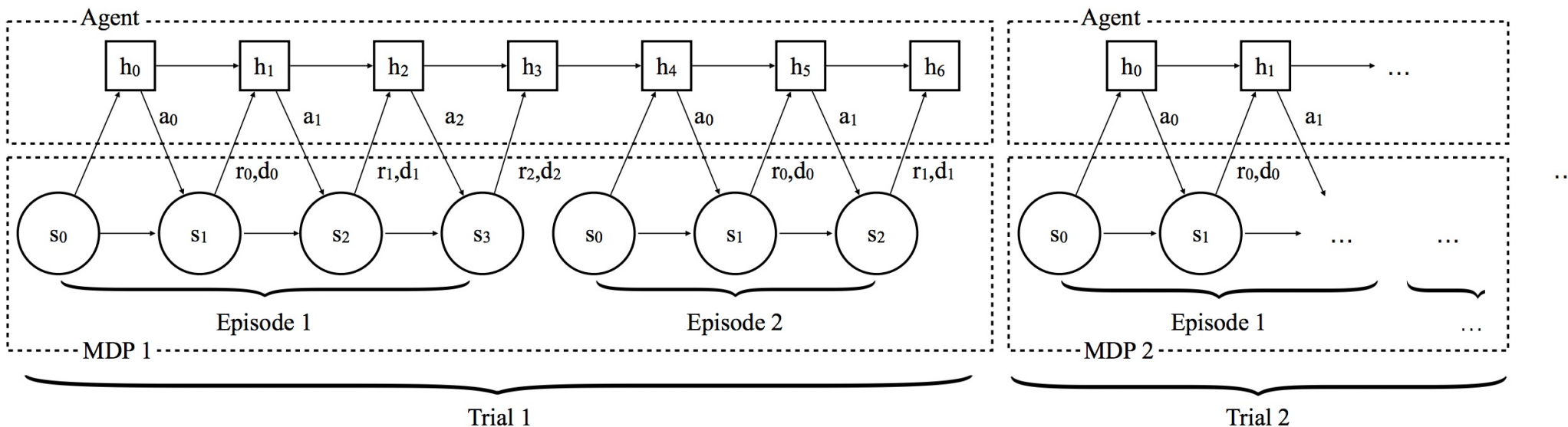
- Parameterization of f_{θ} for inner loop
- Algorithm for outer loop optimization



Memory Based Meta-RL

Idea: Make the update function forward pass of an RNN

- Learn RNN that takes in past s , a , $\mathbf{r}(s, a)$, produce action.
- Maintain hidden state across episodes
- Maximize sum of returns across episodes




Memory Based Meta-RL

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right]$$

where $\phi_i = f_{\theta}(\mathcal{D}_{\tau})$

Combine inner and
outer loop into black
box RNN

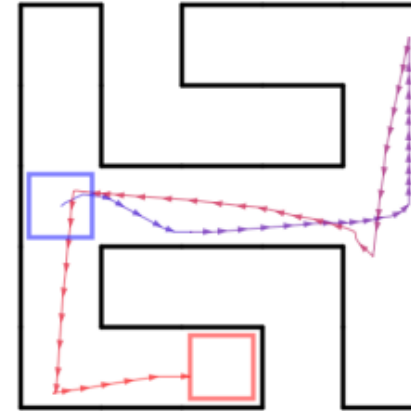
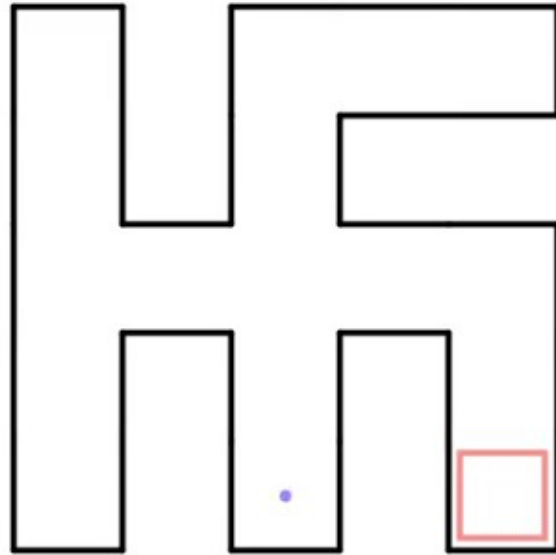
Meta-Training

- 
1. Sample a batch of tasks from $p(\tau)$
 2. Collect data using RNN across episodes for each task, with persistent hidden state and rewards available to the policy
 3. Optimize RNN policy via policy gradient BPTT

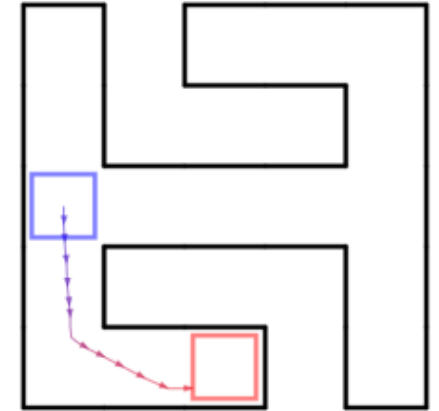
Meta-Testing

1. Simply run the RNN forward pass across episodes

Memory Based Meta-RL



(a) Good behavior, 1st episode



(b) Good behavior, 2nd episode

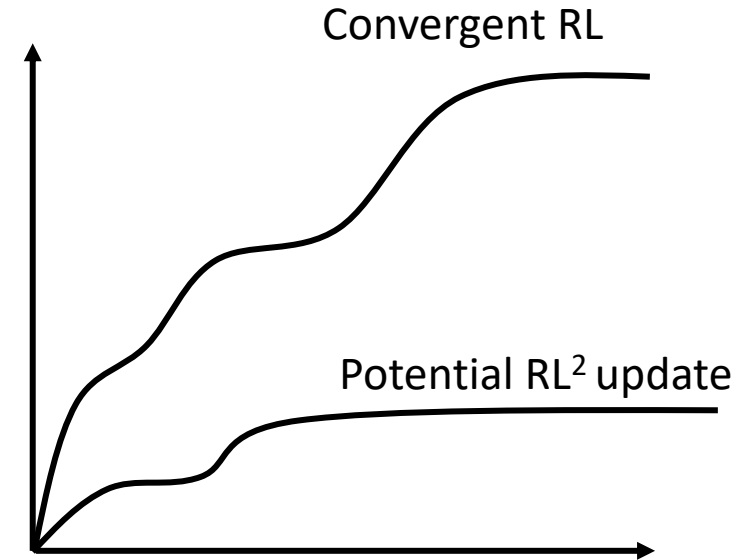
How well does memory based meta-RL work?

Pros:

- Simple, easy to implement
- Arbitrarily flexible inner loop
- Generally stable optimization

Cons:

- No guaranteed improvement during meta-test time
- Poor performance OOD



Optimization Based Meta-RL

Idea:

What if we force $f(\theta)$ to be convergent?

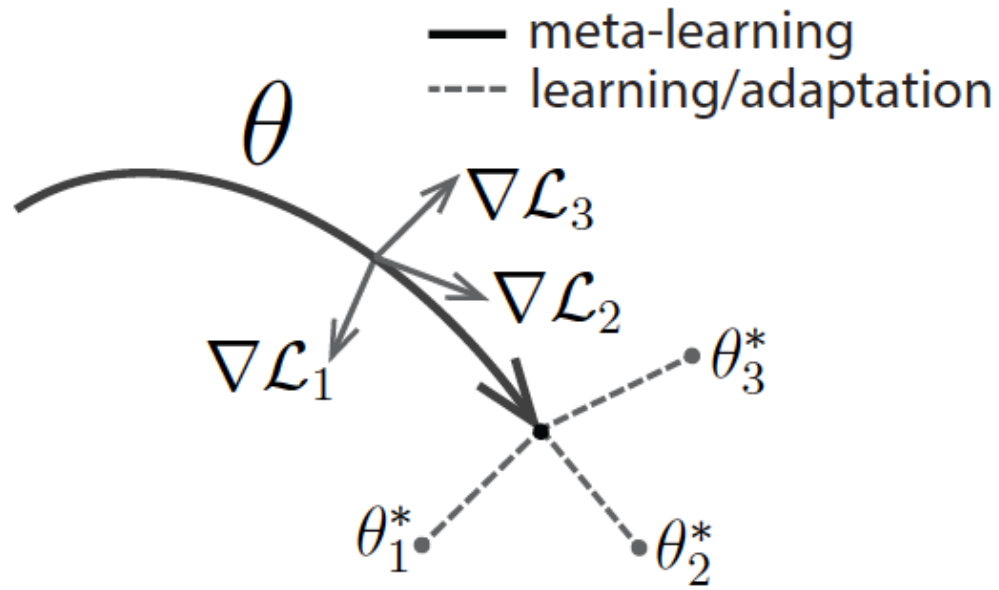
Force $f(\theta)$ to be a convergent optimization algorithm like SGD

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right]$$

$$\phi_i = f_{\theta}(\mathcal{M}_i)$$

↑ Restrict to be convergent optimization

MAML: Gradient Based Meta-RL



$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r_{\tau}(s_t, a_t) \right] \right]$$

$$\phi_i = \theta + \alpha \nabla_{\theta} \mathbb{E}_{\pi_{\theta}} \left[\sum_t r_{\tau}(s_t, a_t) \right]$$

Learn most fine-tunable initial parameters, such that 1-step of SGD is good

Pseudocode for Gradient Based RL



1. Sample a batch of tasks from $p(\tau)$

2. collect data pre-update from π_θ

3. Compute update according to $\phi_i = \theta + \alpha \nabla_\theta \mathbb{E}_{\pi_\theta} \left[\sum_t r_\tau(s_t, a_t) \right]$

4. Sample data from ϕ_i post-update

5. Optimize for initial parameters by PG in outer loop

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r_\tau(s_t, a_t) \right] \right]$$

$$\phi_i = \theta + \alpha \nabla_\theta \mathbb{E}_{\pi_\theta} \left[\sum_t r_\tau(s_t, a_t) \right]$$

Second order gradients
via bi-level optimization

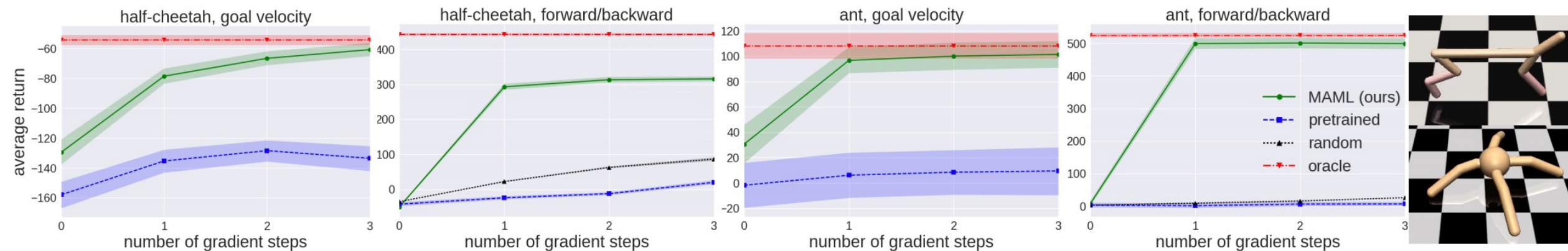
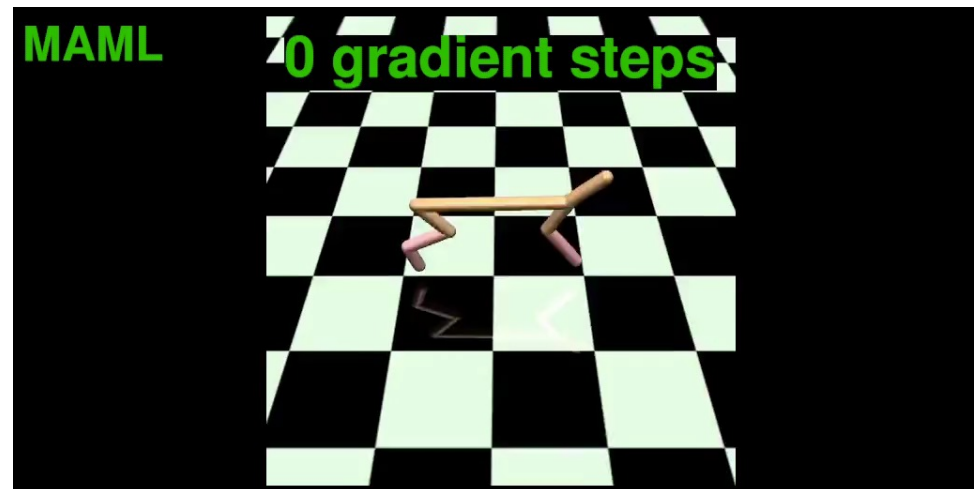
How well does it work?

Tasks:

Half cheetah: goal velocity,

Half cheetah: forward/backward

Ant: forward/backward



How well does it work?

Pros:

Consistent, worst case performance is PG

Only need to learn initialization

Cons:

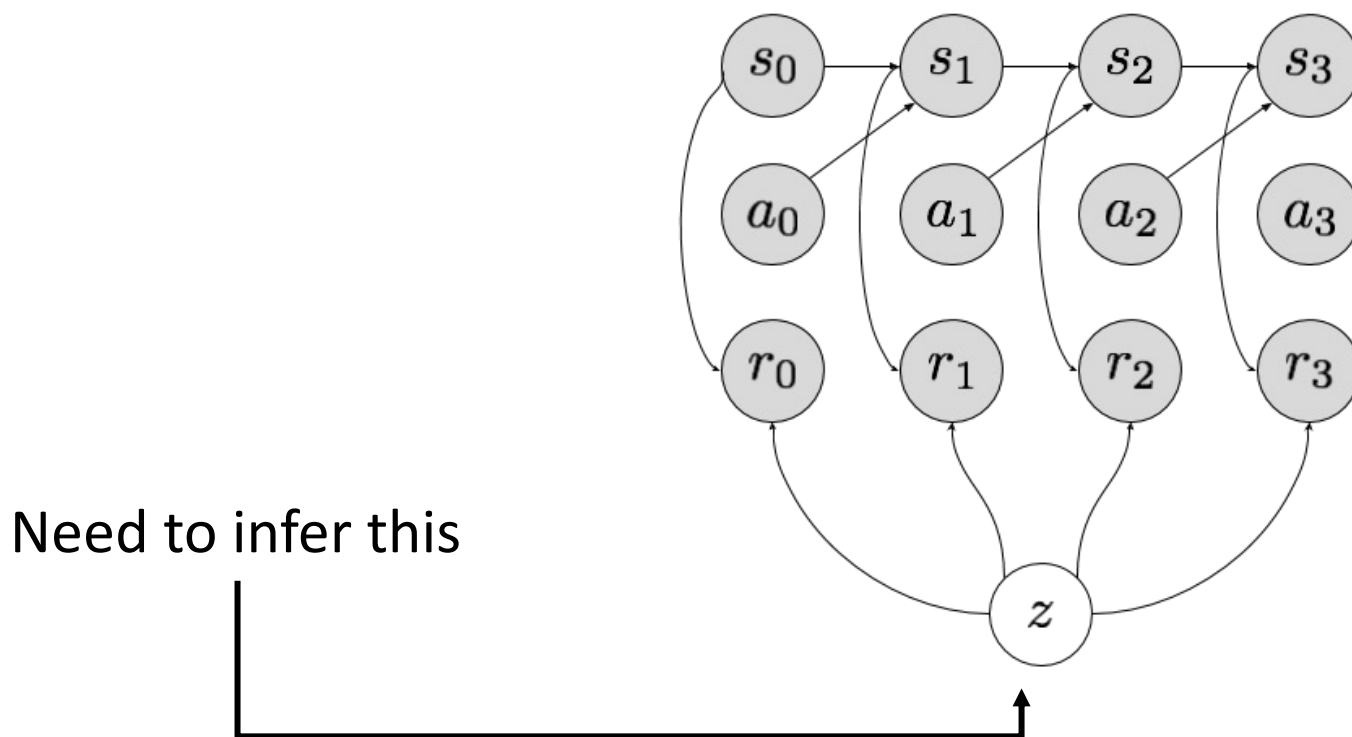
Second order gradients needed

Potentially less expressive update

Latent Variable Models for Meta-RL

Think of meta-RL similar to multi-task RL, but context ω_i is a hidden variable that must be inferred

Meta-RL as a POMDP



Recasting meta-RL as context inference

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim p(\tau)} \left[\mathbb{E}_{\pi_{\phi_i}} \left[\sum_t r(s_t, a_t) \right] \right]$$

where $\phi_i = f_{\theta}(\mathcal{D}_{\tau})$

Infer latent variable from
experience


$$q_{\theta}(z | s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T, a_T, r_T)$$

Deploy latent conditioned
policy


$$\pi_{\theta}(a | s, z)$$

Recasting meta-RL as context inference

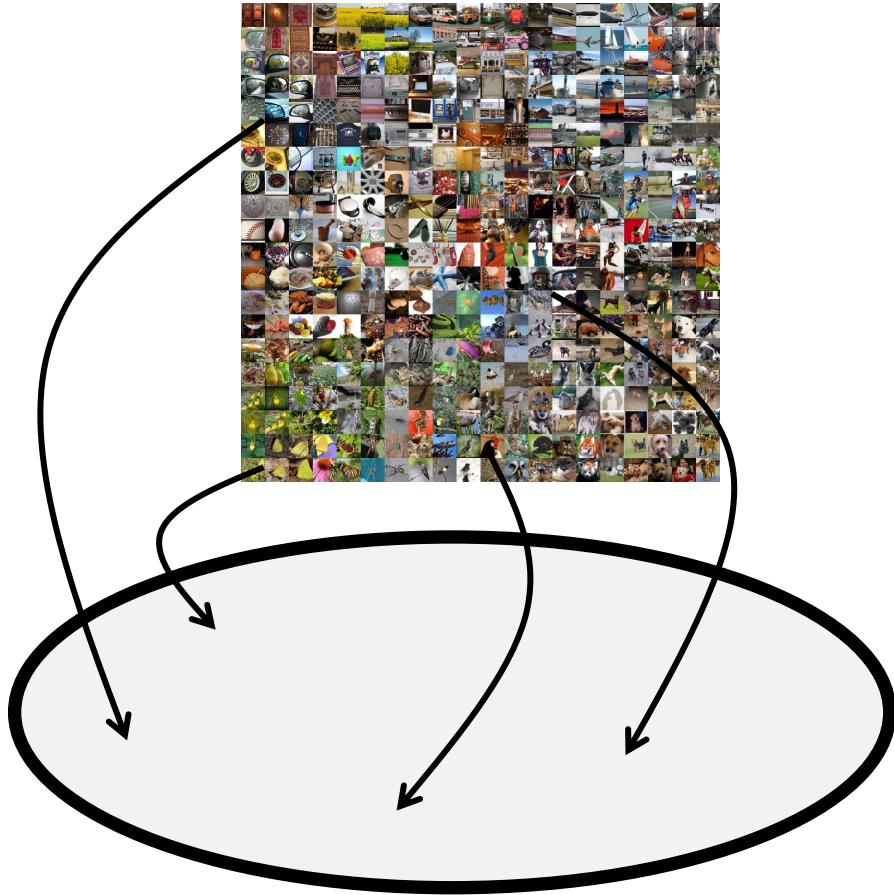
Meta-Training

- 
1. Sample a batch of tasks from $p(\tau)$
 2. Sample trajectories $\{s_0, a_0, r_0, \dots, s_T, a_T, r_T\}_{I=1}^N$
 3. Train $q_\theta(z|s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T, a_T, r_T)$ and $\pi_\theta(a|s, z)$ to maximize rewards via RL (+ some regularization)

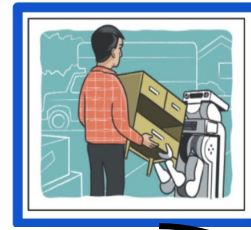
Meta-Testing

- 
1. Sample z from prior $p(z)$
 2. Sample trajectories from $\pi_\theta(a|s, z)$ and z
 3. Update $p(z)$ to posterior $q_\theta(z|s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T, a_T, r_T)$

Latent Variable Model Intuition



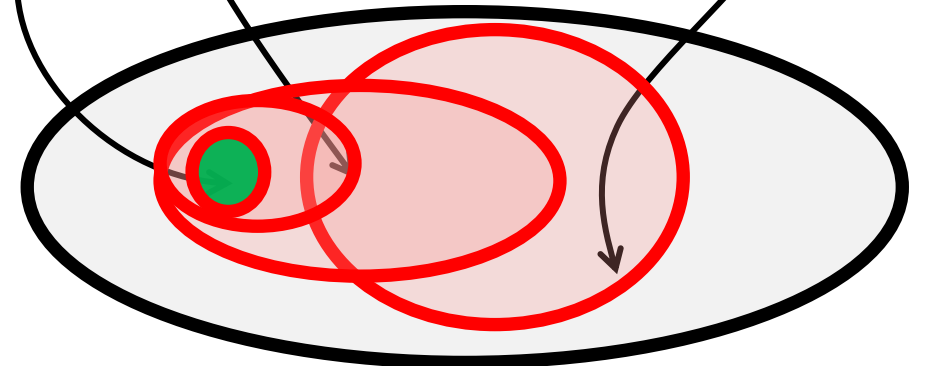
Different images correspond to different z



...



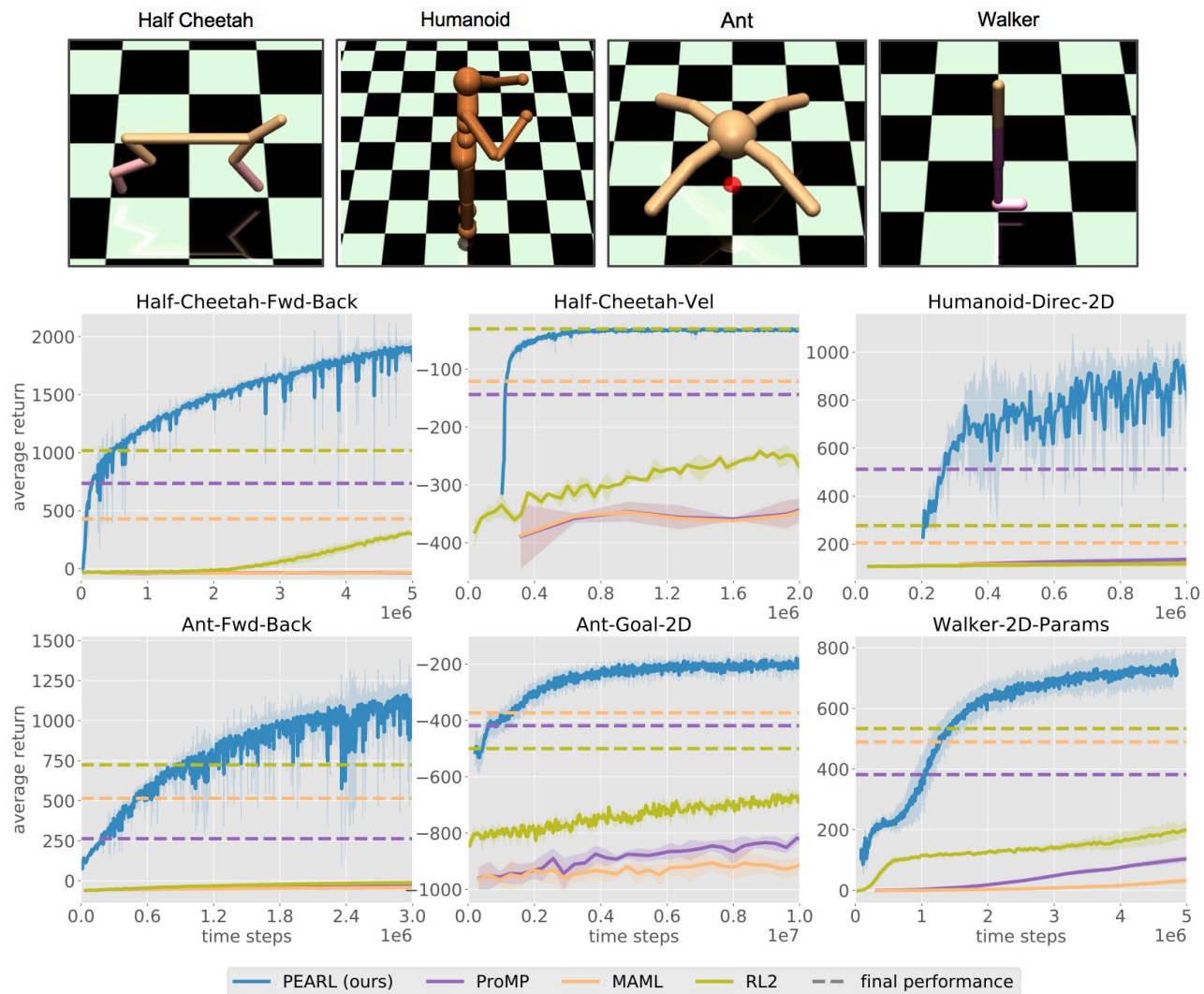
Latent Space



Different tasks correspond to different z
Quick search happens in z space

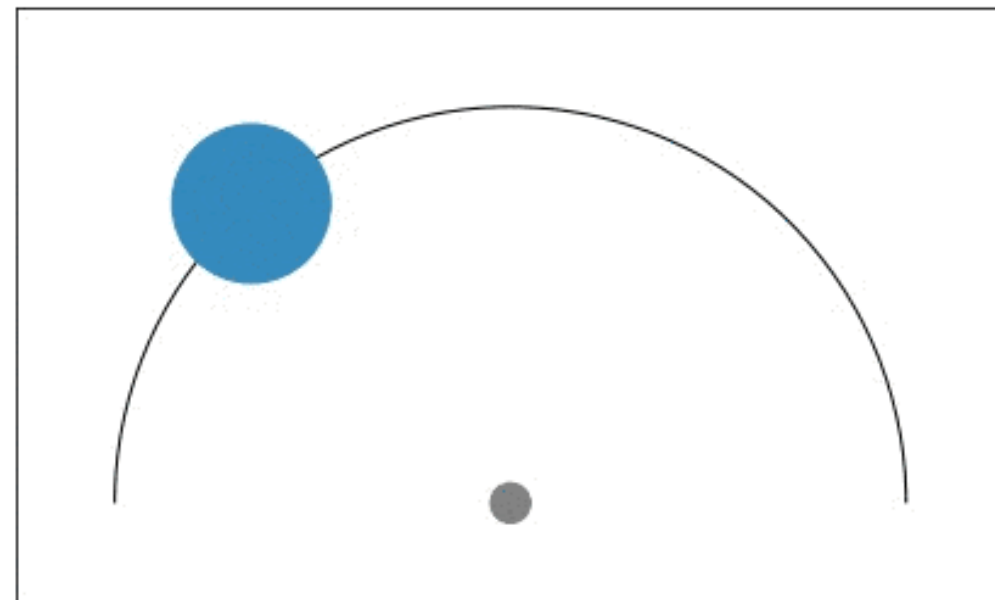
How well does it work?

Quantitative:



Gains mainly from off-policy RL

Exploration:



How well does it work?



How well does it work?

Pros:

Easy to run with off-policy RL

Can be very efficient, trained offline, etc

Might be easy to incorporate priors into inference network

Cons:

Exploration may be suboptimal

May need a huge context variable, hard to optimize/generalize

So meta-RL is cool, does it actually work?

Industrial insertion → adapting to different plug shapes



US-AC-plug

NEMA14-30P

Metal-peg-rec

Metal-peg-rd

UK-AC-plug

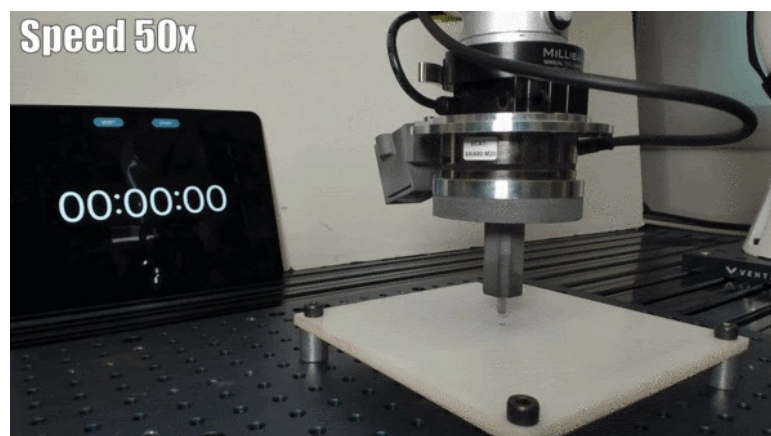
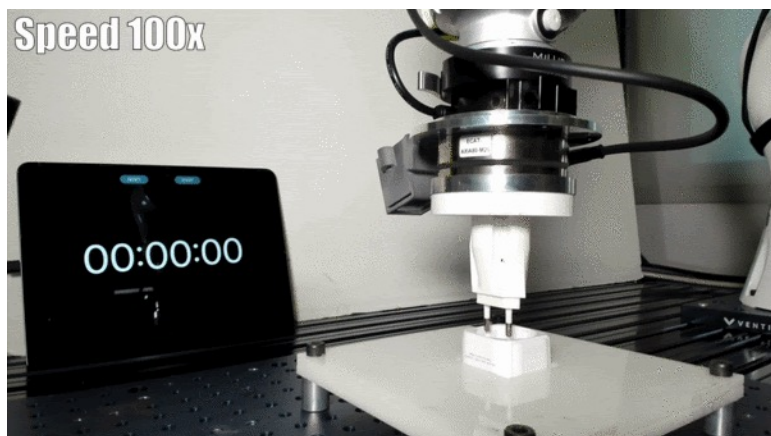
Car-plug-4p

Metal-peg-sq

Car-plug-3p

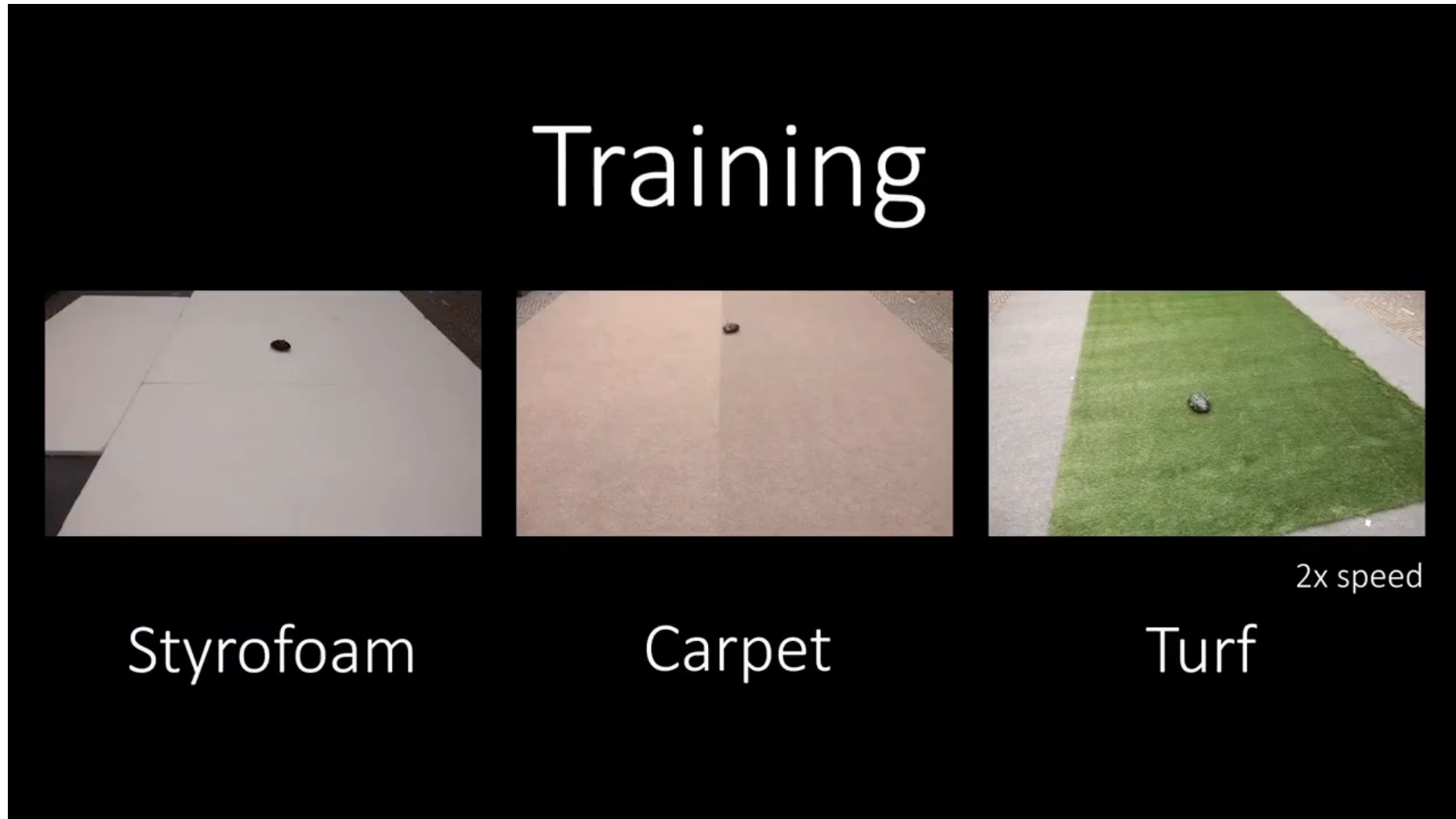
EU-AC-plug

Ours	100/100	100/100	100/100	100/100	100/100	100/100	99/100	75/100	99/100
AWAC	87/100	93/100	96/100	99/100	100/100	100/100	90/100	64/100	100/100

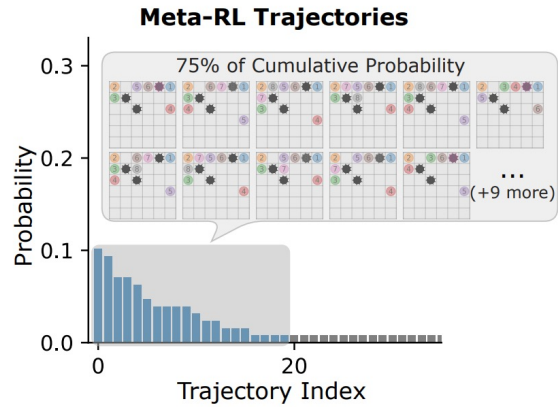
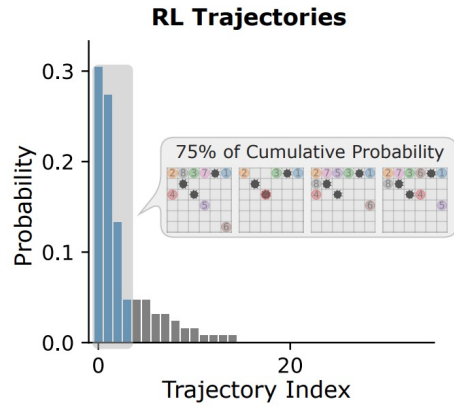
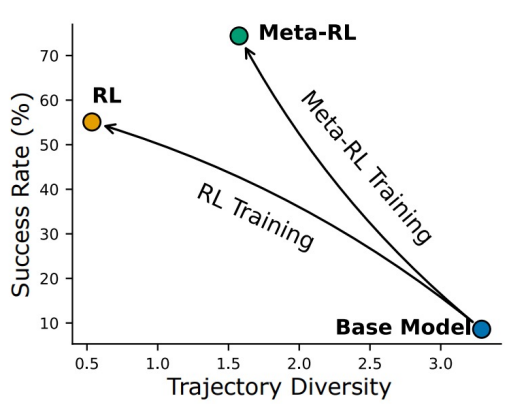


So meta-RL is cool, does it actually work?

Adapting to different terrains/robot conditions



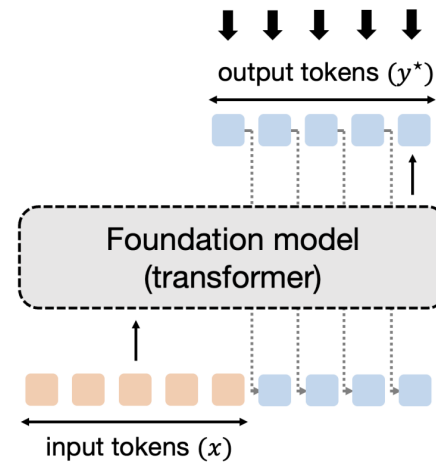
Meta-RL for LLMs



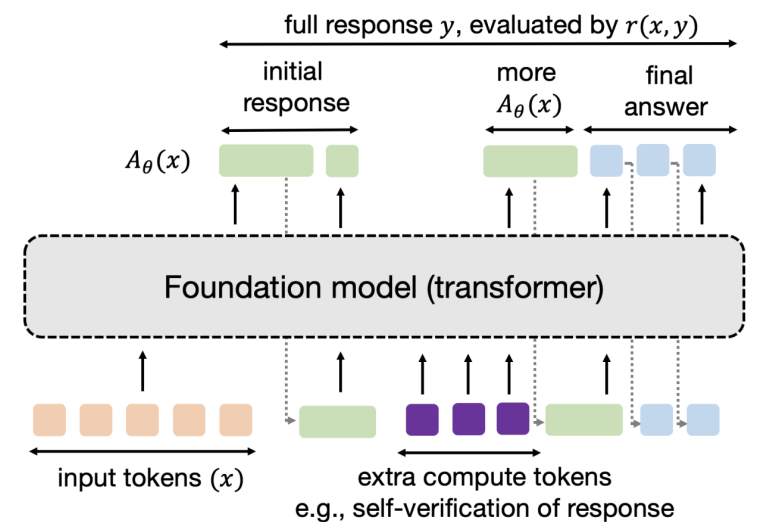
(Tool Use)

(Test-time Compute)

Learning "what": Train the model to fit the target correct output y^*



Learning "how": Train algorithm $A_\theta(x)$ to spend **extra test compute**, search over **responses** & discover the **final answer**.



Check out Aviral Kumar's work!

Meta-RL for Long-Context Learning

The screenshot shows a web browser window with the URL `codeinplace.stanford.edu/diagnostic/feedback`. The page has a navigation bar with tabs for "Overview", "Question 1", "Question 2", "Question 3", "Question 4", "Question 5", and "Wrap-Up". Below the navigation bar are "Back", "Feedback", and "Next" buttons. The main content is divided into two columns. The left column contains the question text: "GETTING INPUT FROM USER" and "This question requires you to get input from the user, convert it to a number, and save it as a variable. Did you correctly do all of these steps?". Below this is a purple feedback box with the text: "Close. There is a minor error with your logic to get input from user. This could be something like forgetting to convert user input to a float". At the bottom of the left column are thumbs up and thumbs down icons, and a text input field labeled "Please explain (optional)". The right column is titled "Your Solution" and contains Python code. The code is:

```
def main():  
    # TODO write your solution here  
    height=input("Enter your height in meters: ")  
    if height < 1.6:  
        print("Below minimum astronaut height")  
    if height > 1.9:  
        print("Above maximum astronaut height")  
    if height >= 1.6 and height <= 1.9:  
        print("Correct height to be an astronaut")  
  
if __name__ == "__main__":  
    main()
```

AI generated feedback

Students evaluate the feedback

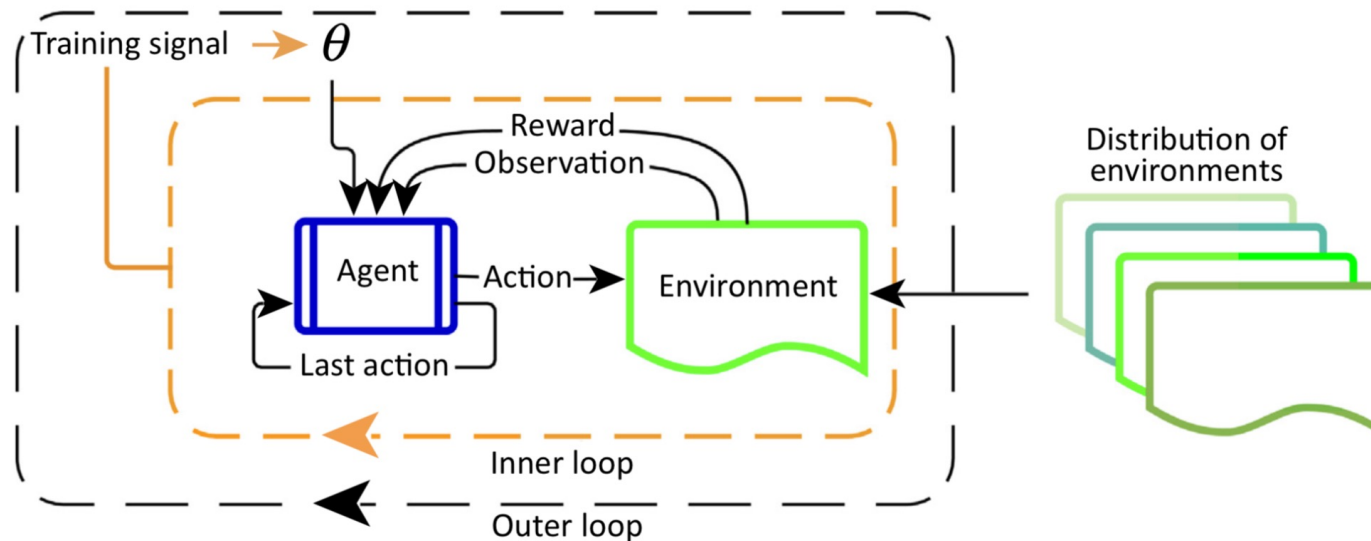
Algorithm uses attention to highlight where in the code the error comes from

Syntax error (missing ") here would prevent auto graders from being useful.

Quick personalization to new code snippets and users

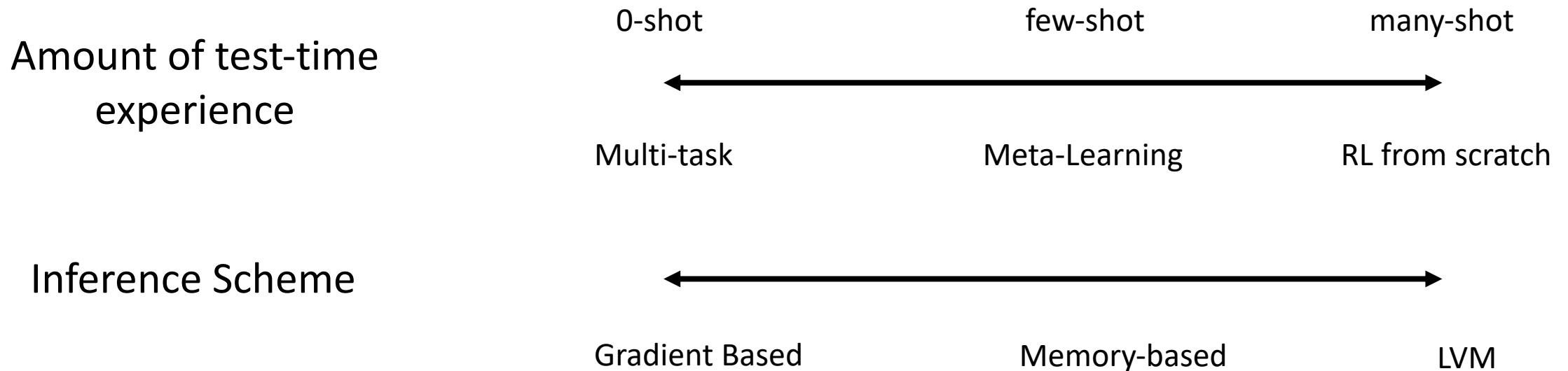
Takeaways from meta-RL

- Meta-RL takes multi-task RL from 0-shot to few-shot
- Meta-RL algorithms can be viewed as choices on top of bi-level optimization
 - memory based, gradient based, latent variable
- Meta-RL can allow adaptation when context is unknown or hard to describe



Putting things in perspective

- Multi-task (and meta) RL takes RL from specialists to generalists (well, kind of)
- The landscape can be understood along 2 axes



Some heavily biased readings

Multi-Task RL

1. Gradient conflict: Gradient Surgery for Multi-Task Learning (Yu et al 2020), Multi-Task Learning as Multi-Objective Optimization (Sener et al 2019)
2. Divide and Conquer: Distal: Robust Multitask Reinforcement Learning (Teh et al 2017), Divide-and-Conquer Reinforcement Learning (Ghosh et al 2018)
3. Multi-task RL at scale: MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale (Kalashnikov et al 2021), BC-Z: (Jang et al 2022), Do As I Can, Not As I Say: Grounding Language in Robotic Affordances (Ahn et al 2022)

Meta-RL

4. Meta-RL overview, older papers by Schimdhuber/Hochreiter
5. Recurrent meta-RL: RL² (Duan et al), L2RL (Wang et al), SNAIL (Mishra et al), CNP (Garnelo et al 2018)
6. Gradient-based meta-RL: MAML (Finn et al), REPTILE (Nichols et al), ProMP (Clavera et al), Antoniu 2018, Bechtle 2019
7. Latent variable meta-RL: PEARL (rakelly et al), VariBAD (zintgraf et al), MAESN (Gupta et al), Zhang et al 2020
8. Model-based meta-RL: Clavera and Nagabandi 2019, Harrison and Sharma 2020, MIER (Mendonca et al)
9. Exploration in meta-RL: MAESN (Gupta et al), DREAM (Liu et al), GMPS (Mendonca et al)
10. Supervision in meta-RL: UMRL (Gupta et al), CARML (Jabri et al), UML (Hsu et al)

Lecture Outline

From specialists to generalists



Multi-Task Reinforcement Learning



Meta-Reinforcement Learning



Frontiers

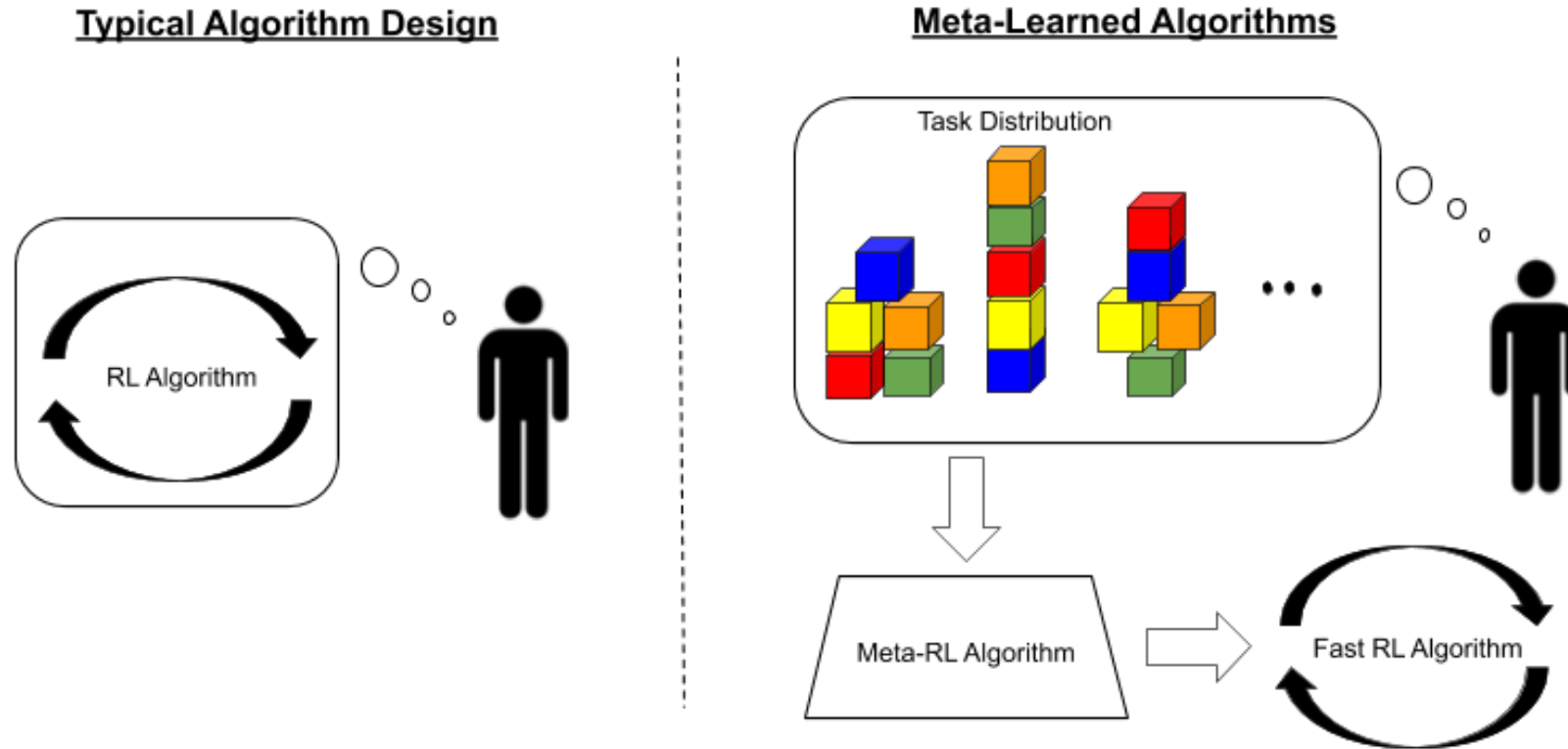
Frontiers of meta-RL

Major issues persist, active research in the following areas:

1. Supervision
2. Exploration
3. ... many others we won't have time to cover

Supervision in meta-RL

Burden of hand-design shifted from algorithm to task distribution

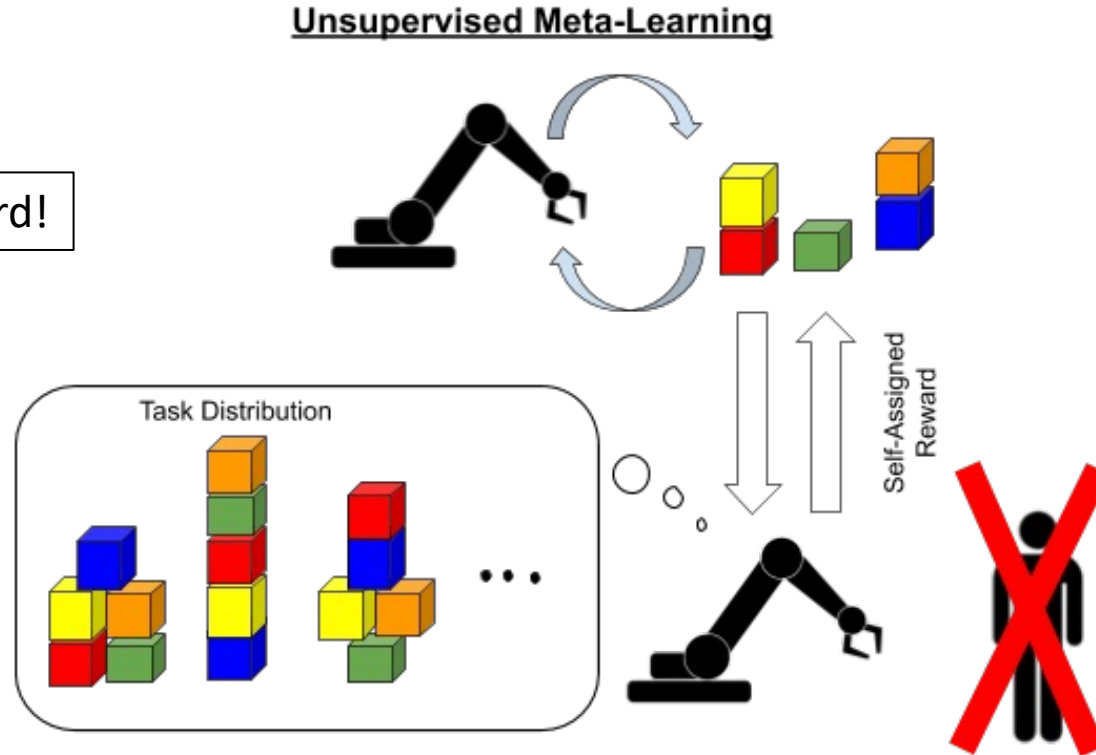


- But where do the tasks come from?

Unsupervised Meta-RL

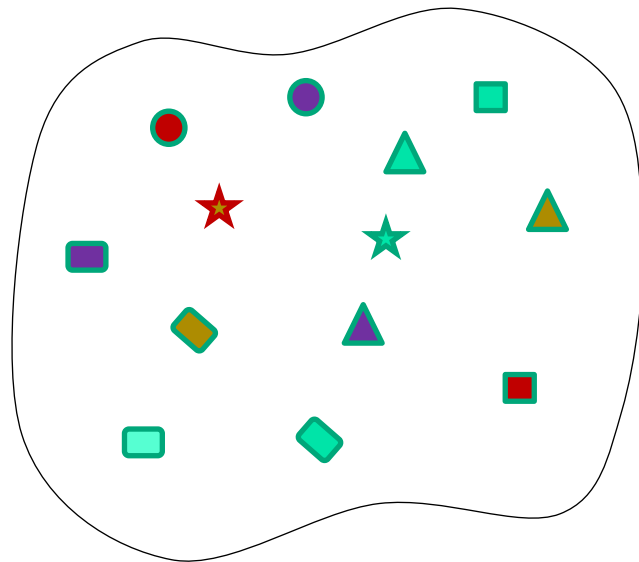
Agent proposes it's own task distribution and reward, and learns with meta-RL

No meta-training reward!

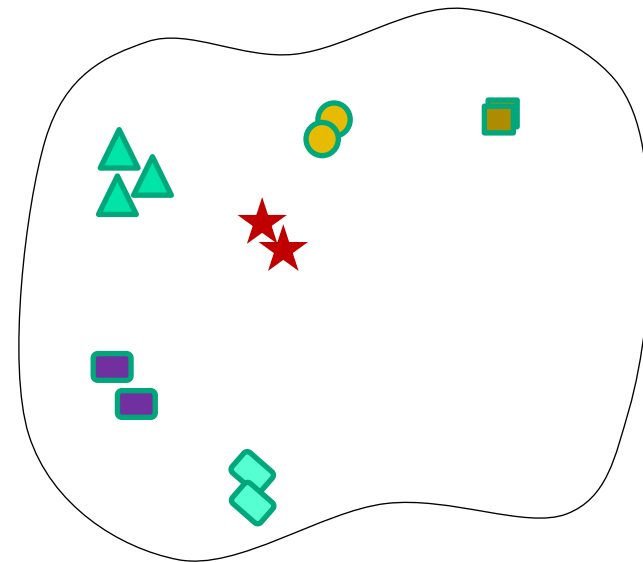


Unsupervised task proposals

- Propose tasks that have good coverage, yet are structured
- Mutual information maximization yields controllable uniform distributions



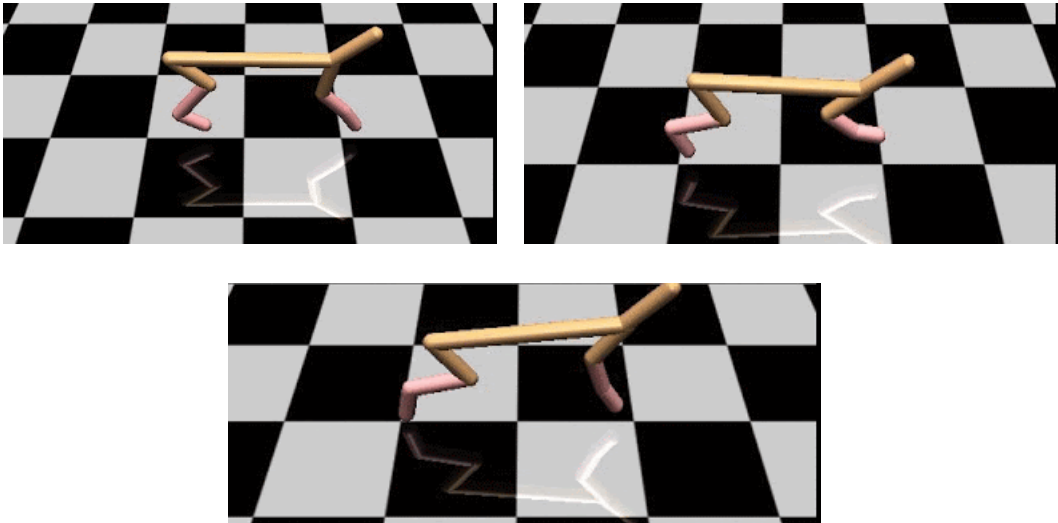
Diversity



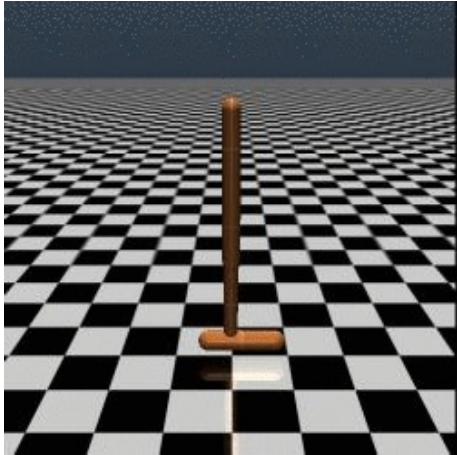
Structure

Proposed Tasks for Unsupervised Meta-RL

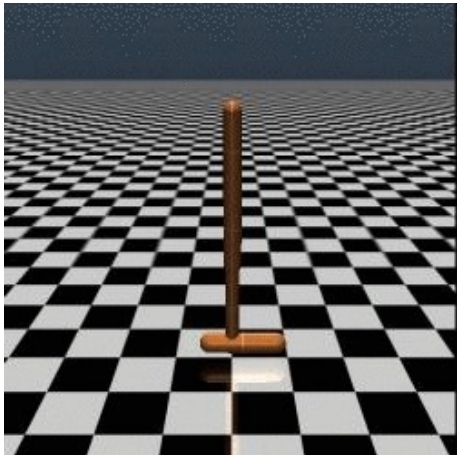
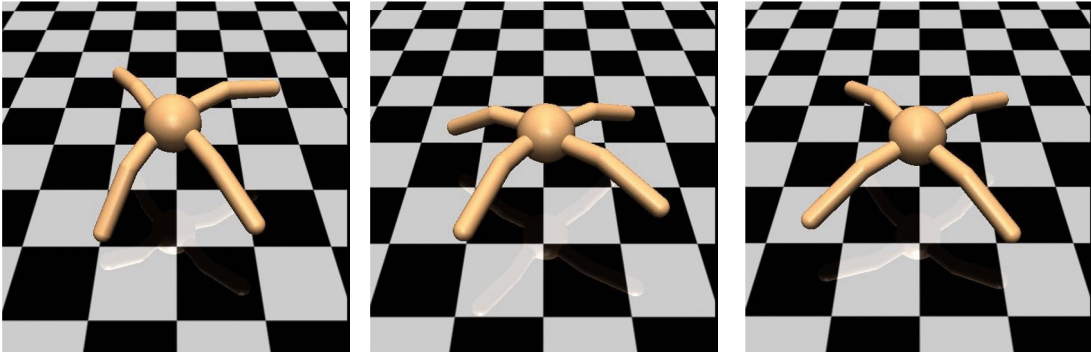
Cheetah



Hopper



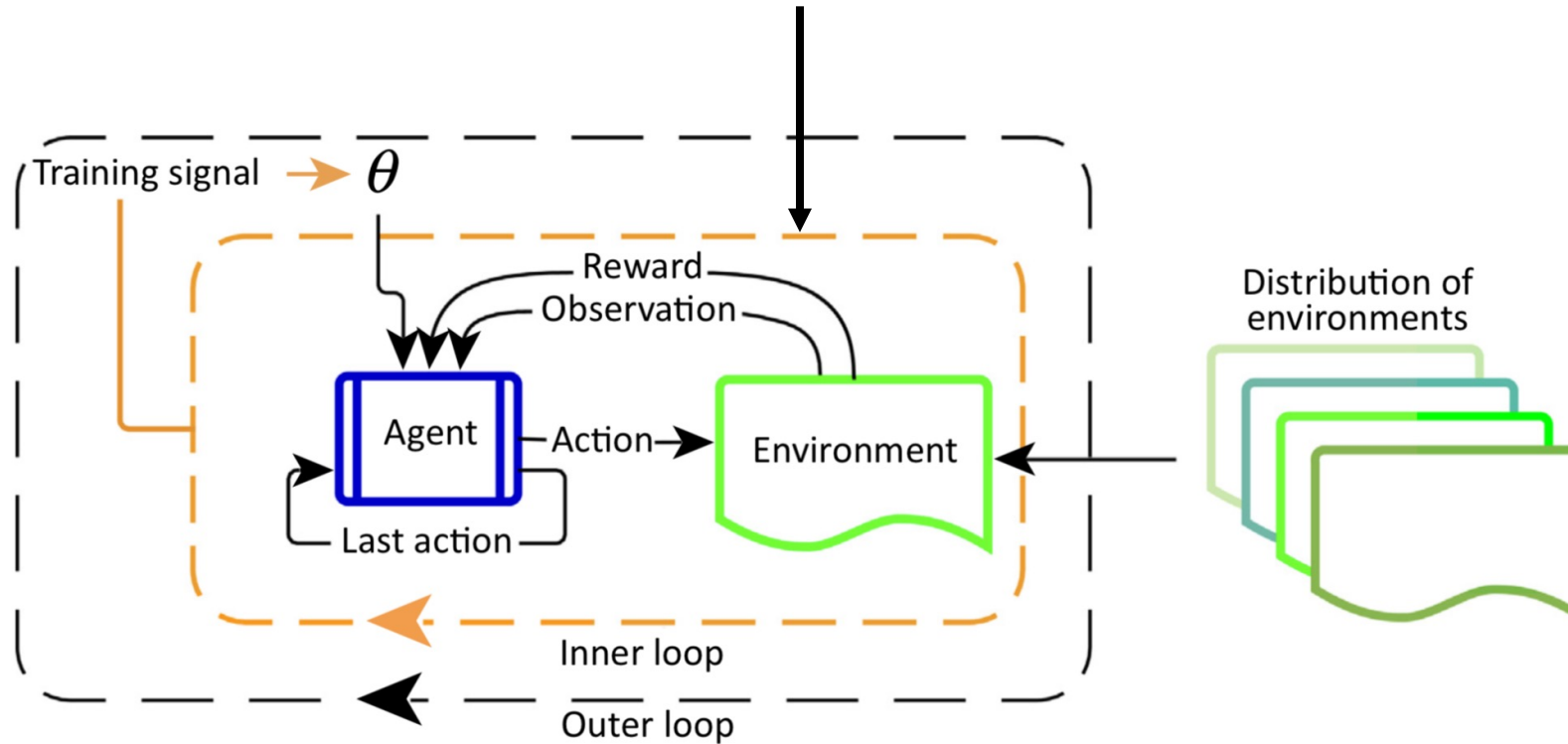
Ant



Exploration in meta-RL

Chicken and Egg Exploration Problem for Exploration in Meta-RL

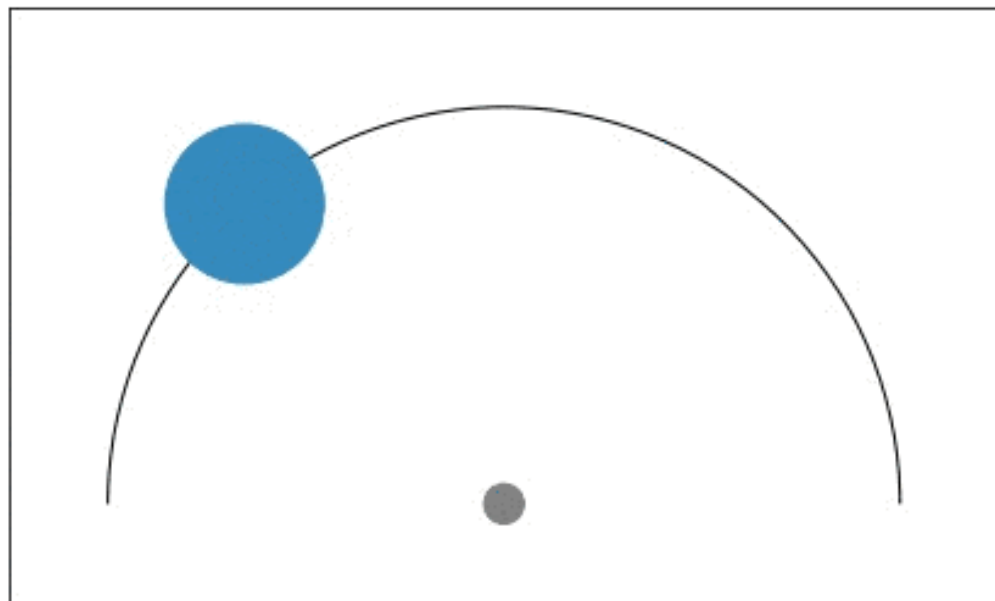
Meta-training may get no signal,
need some training wheels



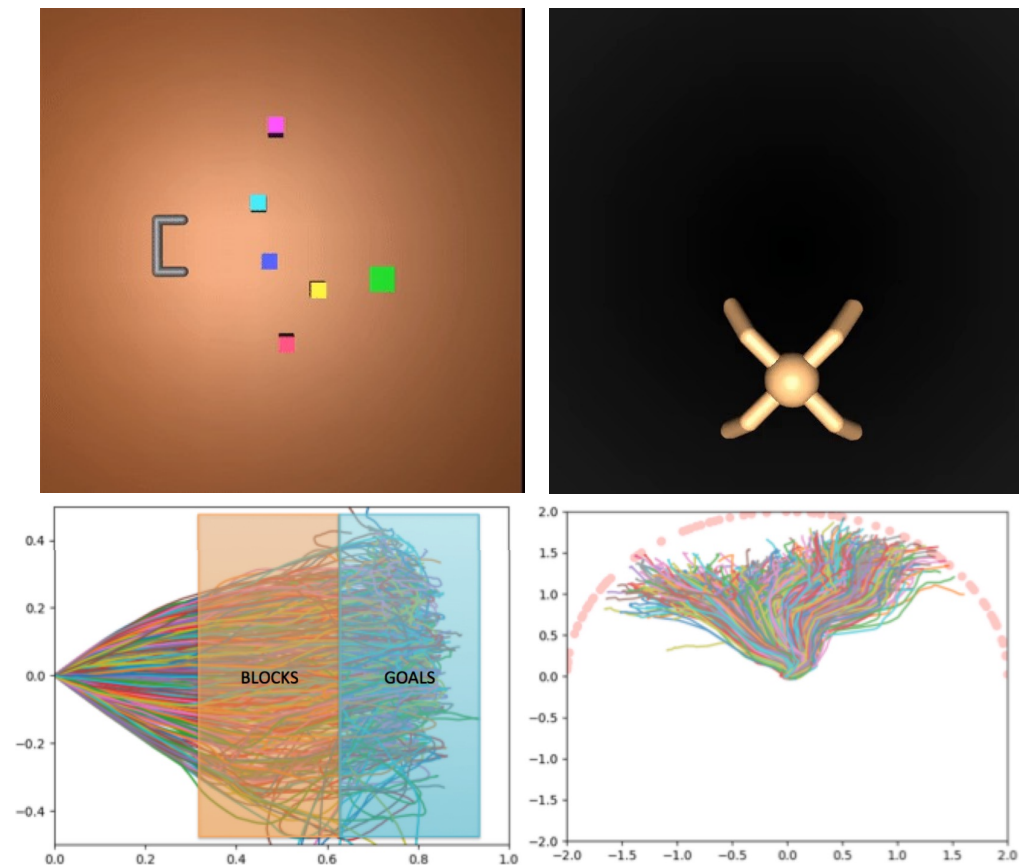
Exploration in meta-RL

Ideas: Assume dense reward available meta-train, sparse reward at meta-test

PEARL (Rakelly et al 2019)

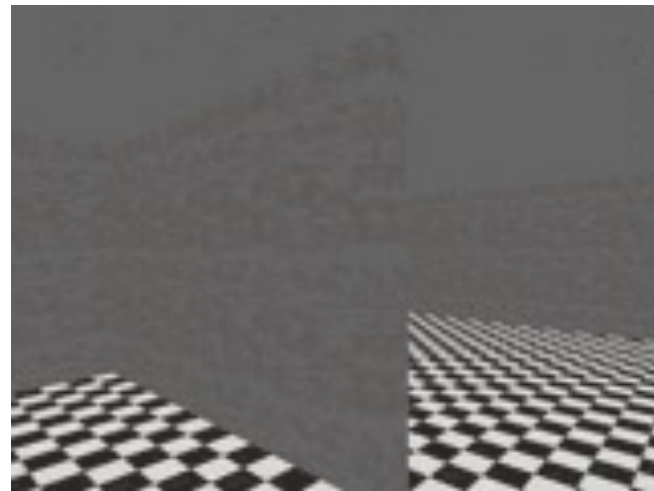
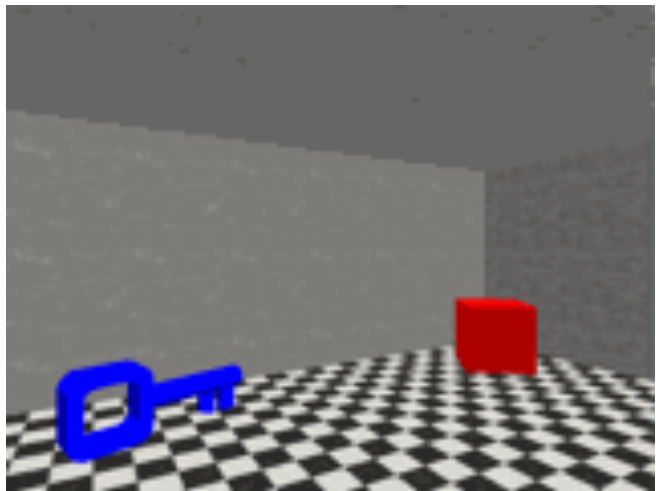
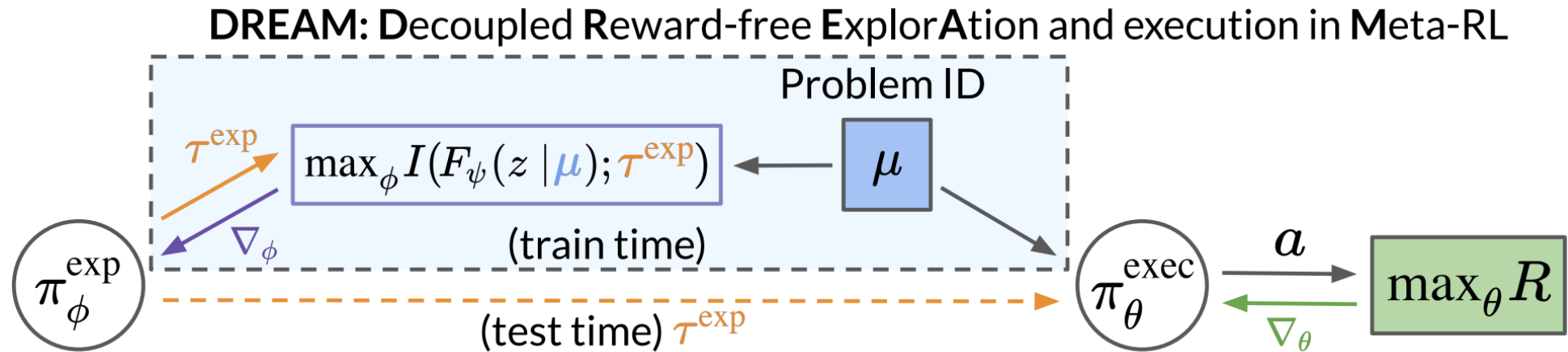


MAESN (Gupta et al 2018)



Exploration in meta-RL

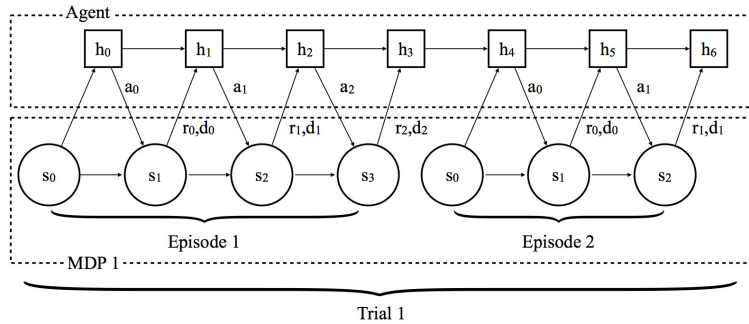
Ideas: Assume context is available at train time, decouple explore and exploit



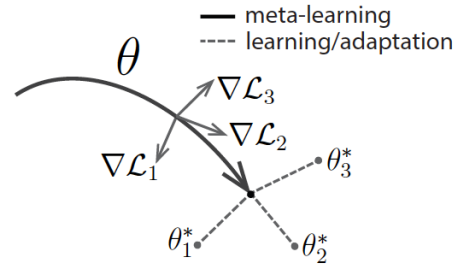
Landscape of Meta-RL

Methods

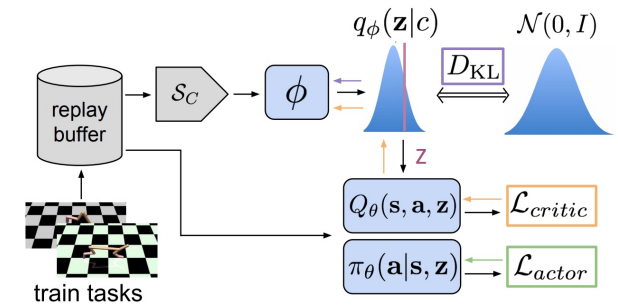
Memory Based



Gradient Based



Latent Variable



Frontiers

Supervision in Meta-RL

Exploration in Meta-RL

...

Takeaways for things to work on

Multi-task RL

- Optimization is hard
 - Gradient surgery/distillation may work, but not always
- Generalization is hard
 - How to represent/inductive bias/capacity, etc.
- Systems are hard
 - Building real world multi-task RL systems need many considerations

Meta RL

- Exploration is hard
 - Circular challenge of exploring to learn how to explore
- Generalization is hard
 - Avoiding meta-overfitting/meta-underfitting is tough
- Sourcing task distributions is hard:
 - Constructing real life examples of task distributions is tough.

What did we learn in this class?

