Transfer and Multi-Task Learning

CS 285: Deep Reinforcement Learning, Decision Making, and Control

Sergey Levine

Today's Lecture

1. Transfer from prior tasks to learn new tasks more quickly
2. Forward transfer: train on source task in such a way as to do better on target task
3. Randomization of source tasks
4. Multi-task transfer
5. Contextual policies
6. Modular policies

- Goals:
  - Understand (at a high level) the landscape of research work on transfer learning

What's the problem?

this is easy (mostly)

Why?

this is impossible

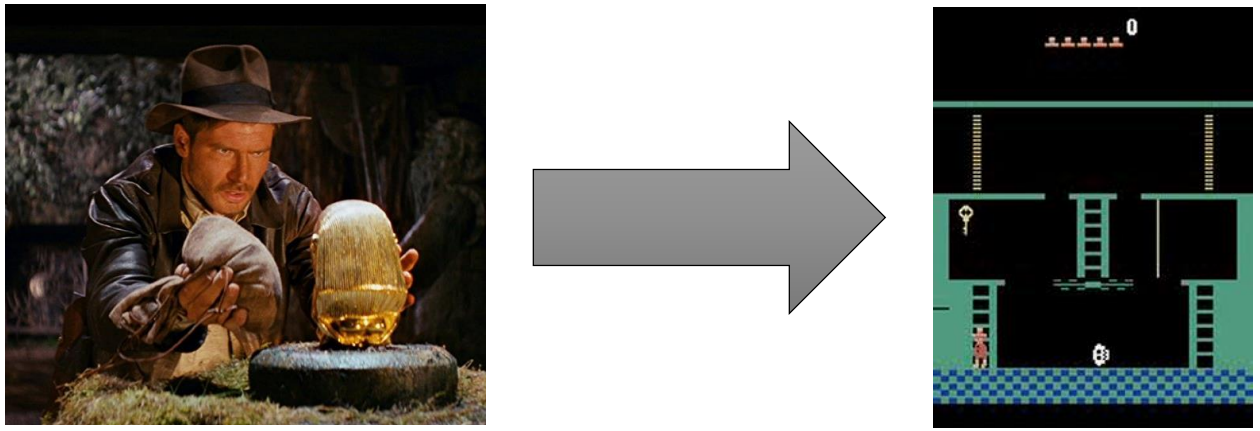
Montezuma’s revenge

- Getting key = reward
- Opening door = reward
- Getting killed by skull = bad

Montezuma's revenge

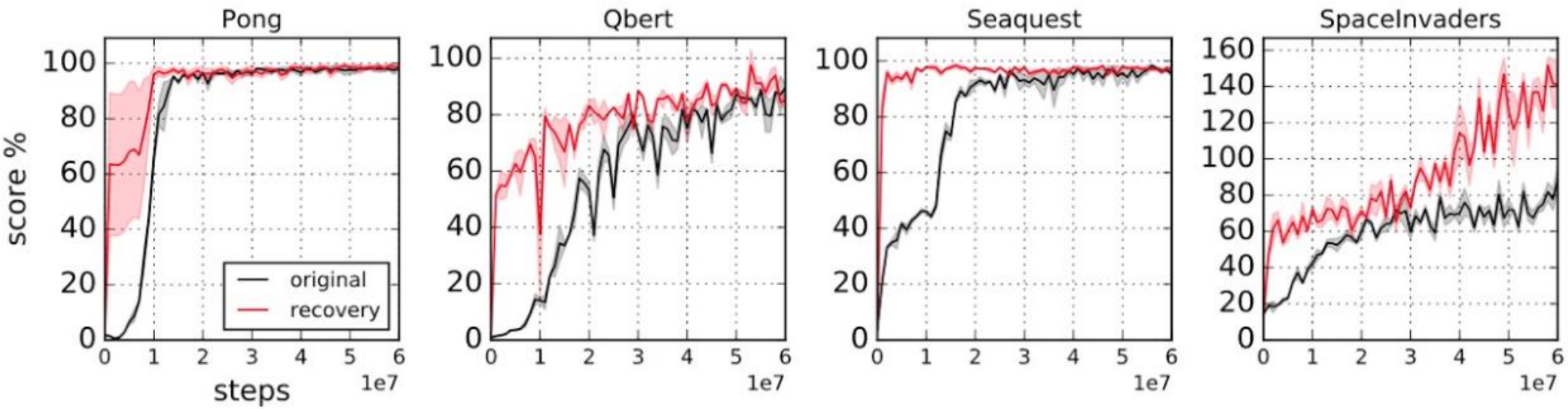
- We know what to do because we **understand** what these sprites mean!
- Key: we know it opens doors!
- Ladders: we know we can climb them!
- Skull: we don't know what it does, but we know it can't be good!
- **Prior understanding of problem structure can help us solve complex tasks quickly!**

Can RL use the same prior knowledge as us?



- If we've solved prior tasks, we might acquire useful knowledge for solving a new task
- How is the knowledge stored?
  - Q-function: tells us which actions or states are good
  - Policy: tells us which actions are potentially useful
    - some actions are never useful!
  - Models: what are the laws of physics that govern the world?
  - Features/hidden states: provide us with a good representation
    - Don't underestimate this!

Aside: the representation bottleneck



To decouple reinforcement learning from representation learning, we decapitate an agent by destroying its policy and value outputs and then re-train end-to-end. The representation remains and the policy is swiftly recovered. **The gap between initial optimization and recovery shows a representation learning bottleneck**.

slide adapted from E. Schelhamer, “Loss is its own reward”

Transfer learning terminology

transfer learning: using experience from one set of tasks for faster learning and better performance on a new task

in RL, task = MDP!

source domain

target domain

"shot": number of attempts in the target domain

0-shot: just run a policy trained in the source domain

1-shot: try the task once

few shot: try the task a few times

How can we frame transfer learning problems?

No single solution! Survey of various recent research papers

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetune on the new task
   c) Randomize source domain
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Generate highly randomized source domains
   b) Model-based reinforcement learning
   c) Model distillation
   d) Contextual policies
   e) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

How can we frame transfer learning problems?

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetune on the new task
   c) Randomize source domain
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Generate highly randomized source domains
   b) Model-based reinforcement learning
   c) Model distillation
   d) Contextual policies
   e) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

Try it and hope for the best

Policies trained for one set of circumstances might just work in a new domain, but no promises or guarantees

Try it and hope for the best

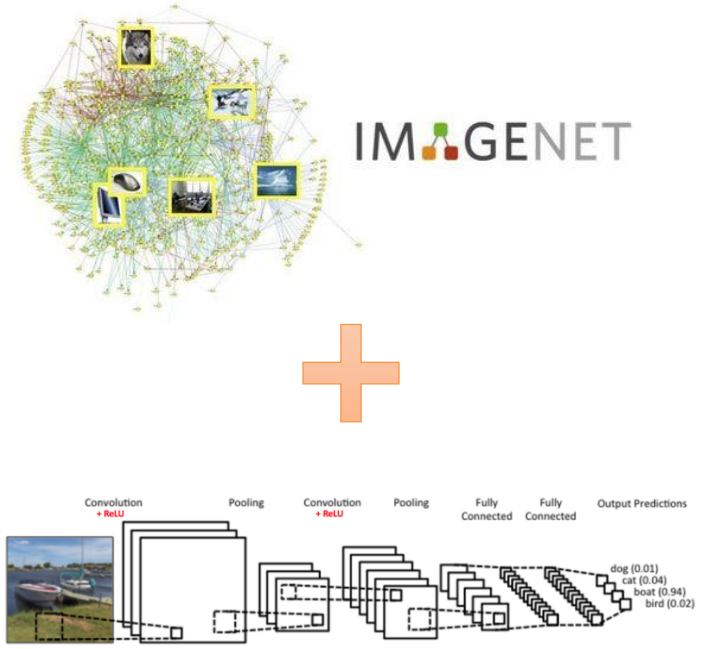
Policies trained for one set of circumstances might just work in a new domain, but no promises or guarantees

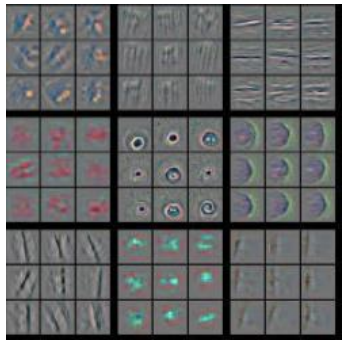
Levine*, Finn*, et al. ‘16

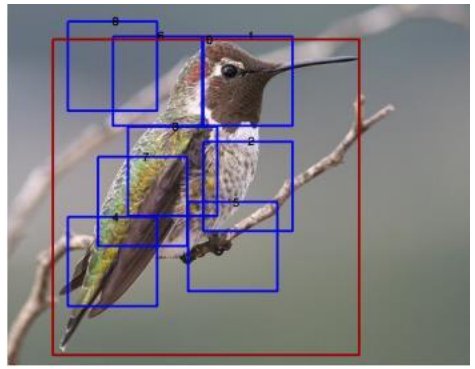
Devin et al. ‘17

Finetuning

The most popular transfer learning method in (supervised) deep learning!



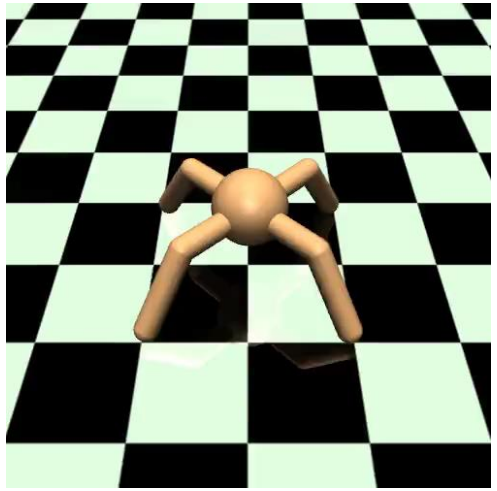




Where are the “ImageNet” features of RL?

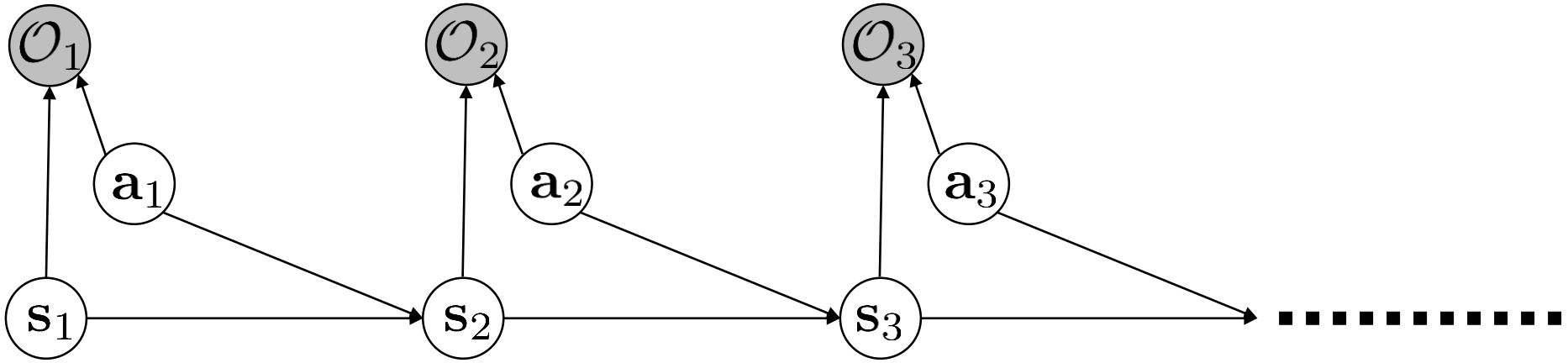
Challenges with finetuning in RL

1. RL tasks are generally much less diverse
   - Features are less general
   - Policies & value functions become overly specialized
2. Optimal policies in fully observed MDPs are deterministic
   - Loss of exploration at convergence
   - Low-entropy policies adapt very slowly to new settings



Finetuning with maximum-entropy policies

How can we increase diversity and entropy?

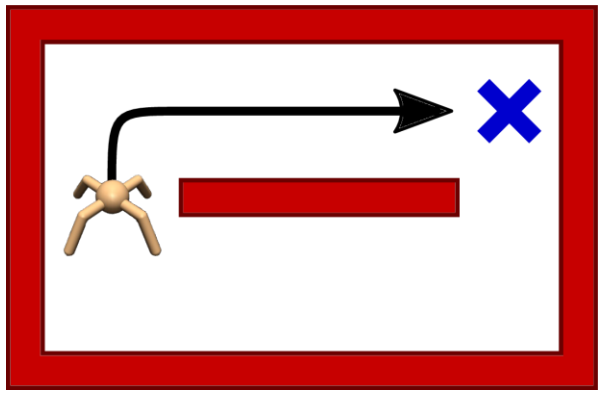


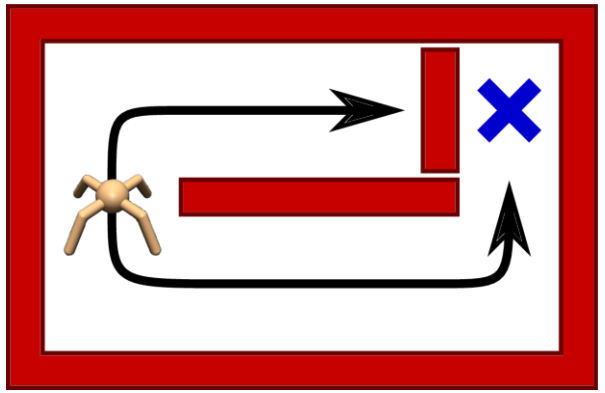
$$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s},\mathbf{a}) - V(\mathbf{s})) \text{ optimizes } \sum_t E_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]$$

policy entropy

Act **as randomly as possible** while collecting high rewards!

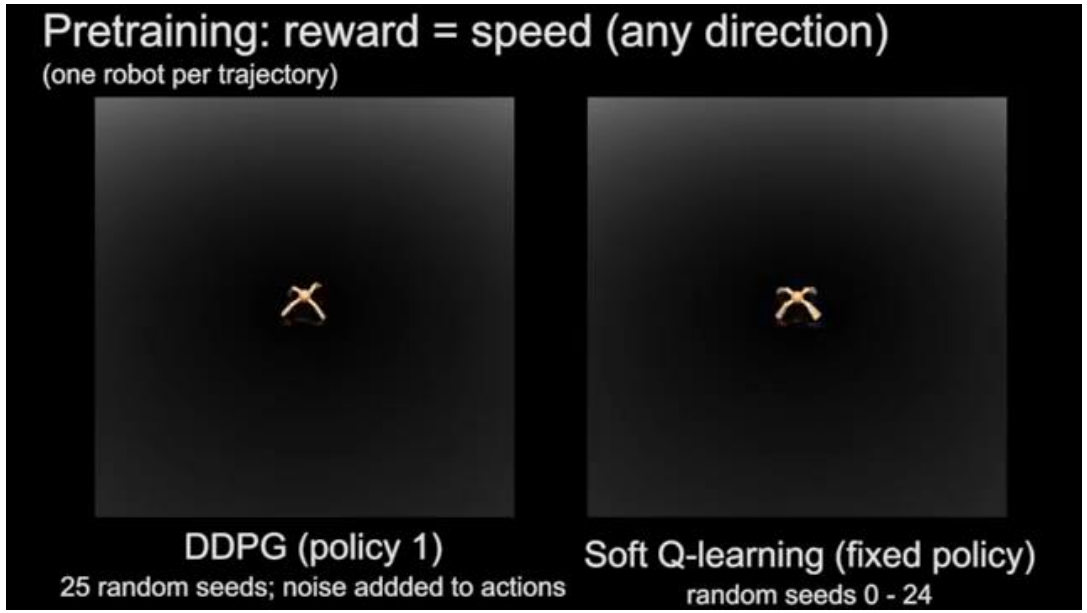
Example: pre-training for robustness

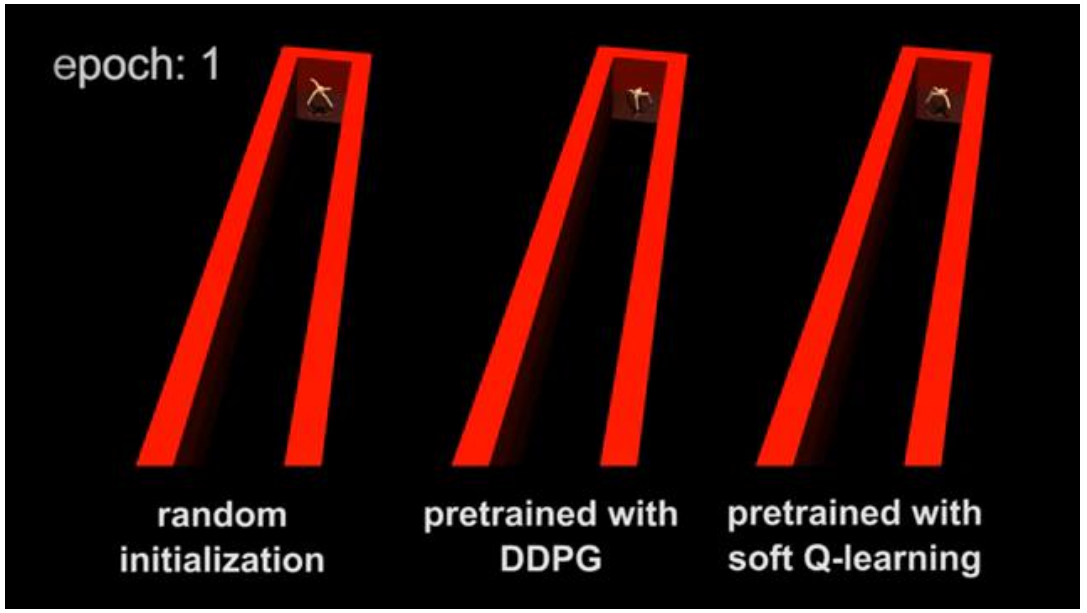


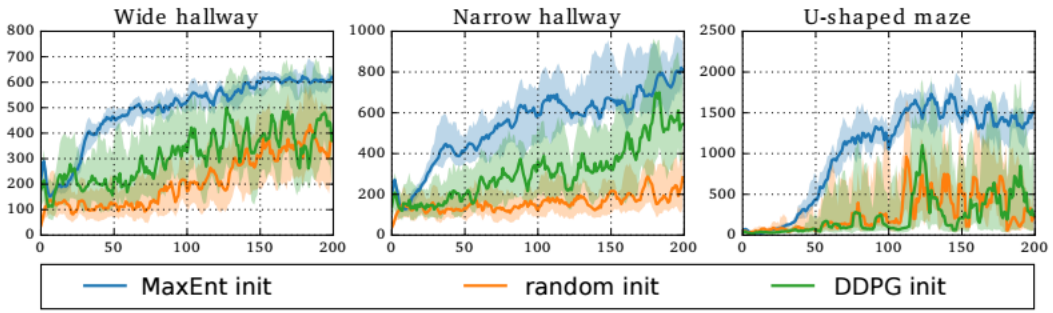


Learning to solve a task **in all possible ways** provides for more robust transfer!

Example: pre-training for diversity







Haarnoja*, Tang*, et al. “Reinforcement Learning with Deep Energy-Based Policies”

Finetuning in RL: suggested readings

Finetuning via MaxEnt RL: Haarnoja*, Tang*, et al. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**

Finetuning from transferred visual features (via VAE): Higgins et al. **DARLA: improving zero-shot transfer in reinforcement learning.** 2017.

Pretraining with hierarchical RL methods:

Andreas et al. **Modular multitask reinforcement learning with policy sketches.** 2017.

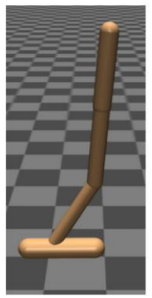
Florensa et al. **Stochastic neural networks for hierarchical reinforcement learning.** 2017.

…and many many others!

What if we can manipulate the source domain?

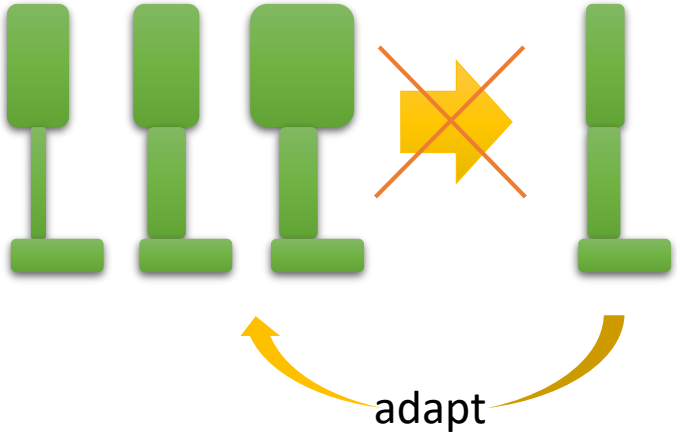
- So far: source domain (e.g., empty room) and target domain (e.g., corridor) are fixed
- What if we can **design** the source domain, and we have a **difficult** target domain?
  - Often the case for simulation to real world transfer
- Same idea: the more diversity we see at training time, the better we will transfer!

EPOpt: randomizing physical parameters

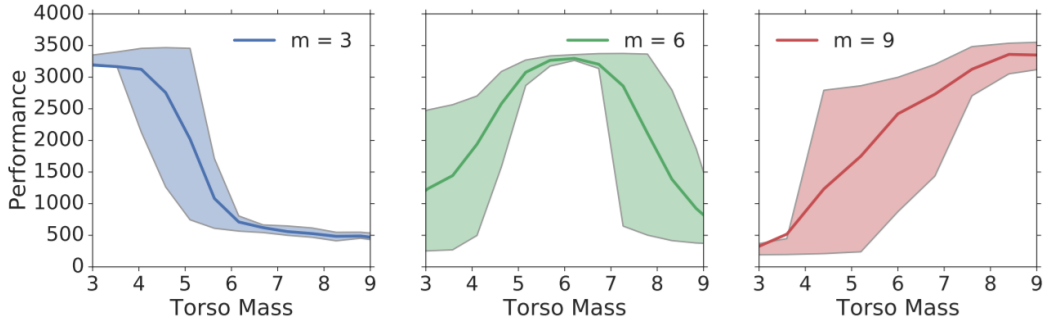


train

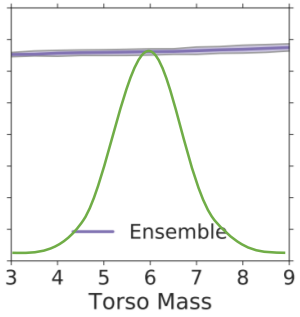
test



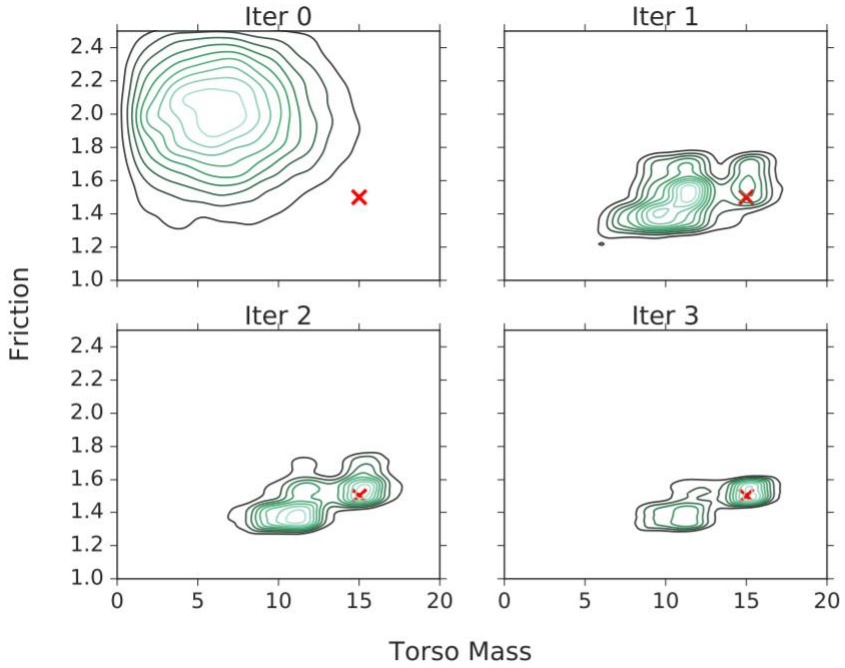
training on single torso mass



training on model ensemble

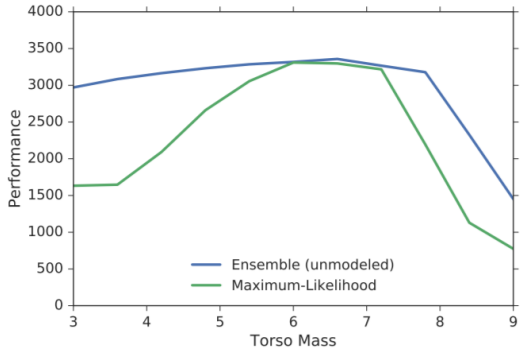


ensemble adaptation



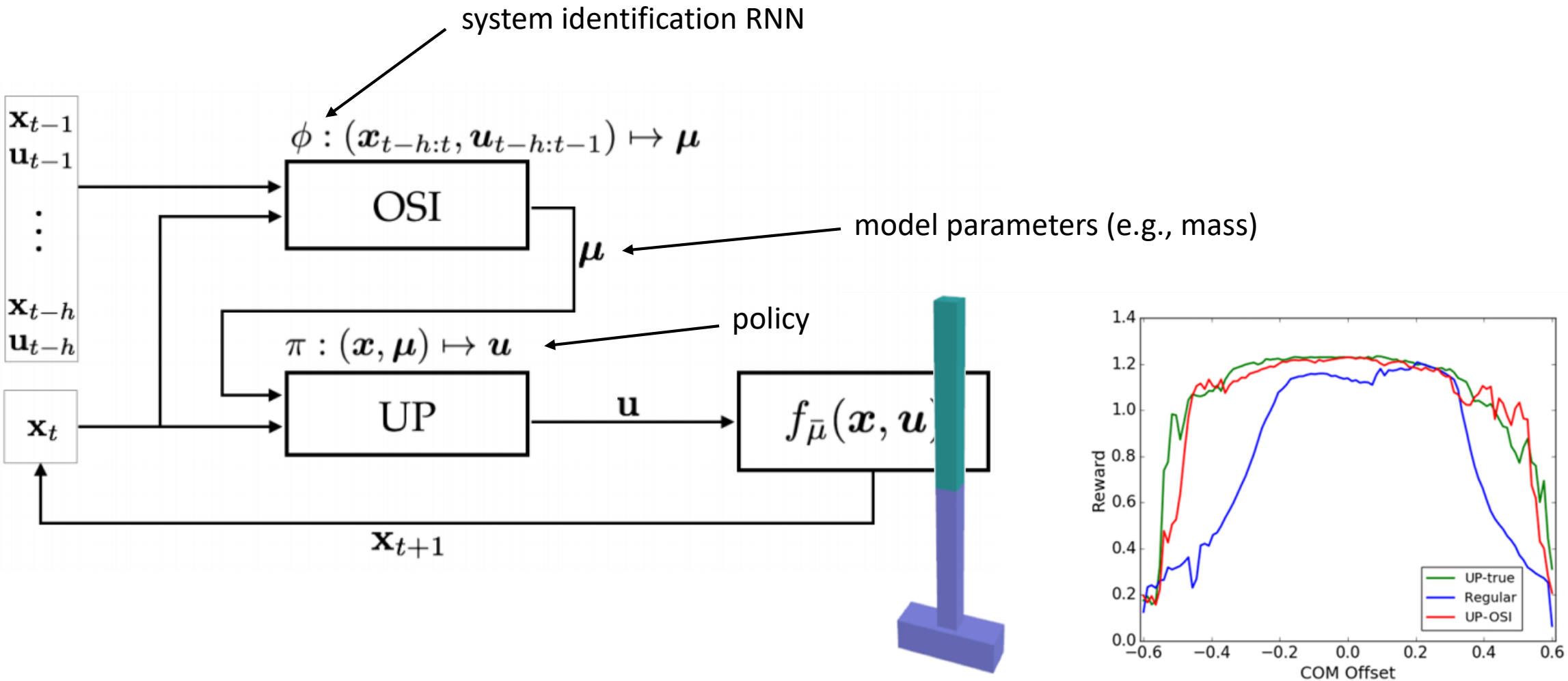
unmodeled effects

Hopper	μ	σ	low	high
~~mass~~	~~6.0~~	~~1.5~~	~~3.0~~	~~9.0~~
ground friction	2.0	0.25	1.5	2.5
joint damping	2.5	1.0	1.0	4.0
armature	1.0	0.25	0.5	1.5
Half-Cheetah	μ	σ	low	high
~~mass~~	~~6.0~~	~~1.5~~	~~3.0~~	~~9.0~~
ground friction	0.5	0.1	0.3	0.7
joint damping	1.5	0.5	0.5	2.5
armature	0.125	0.04	0.05	0.2



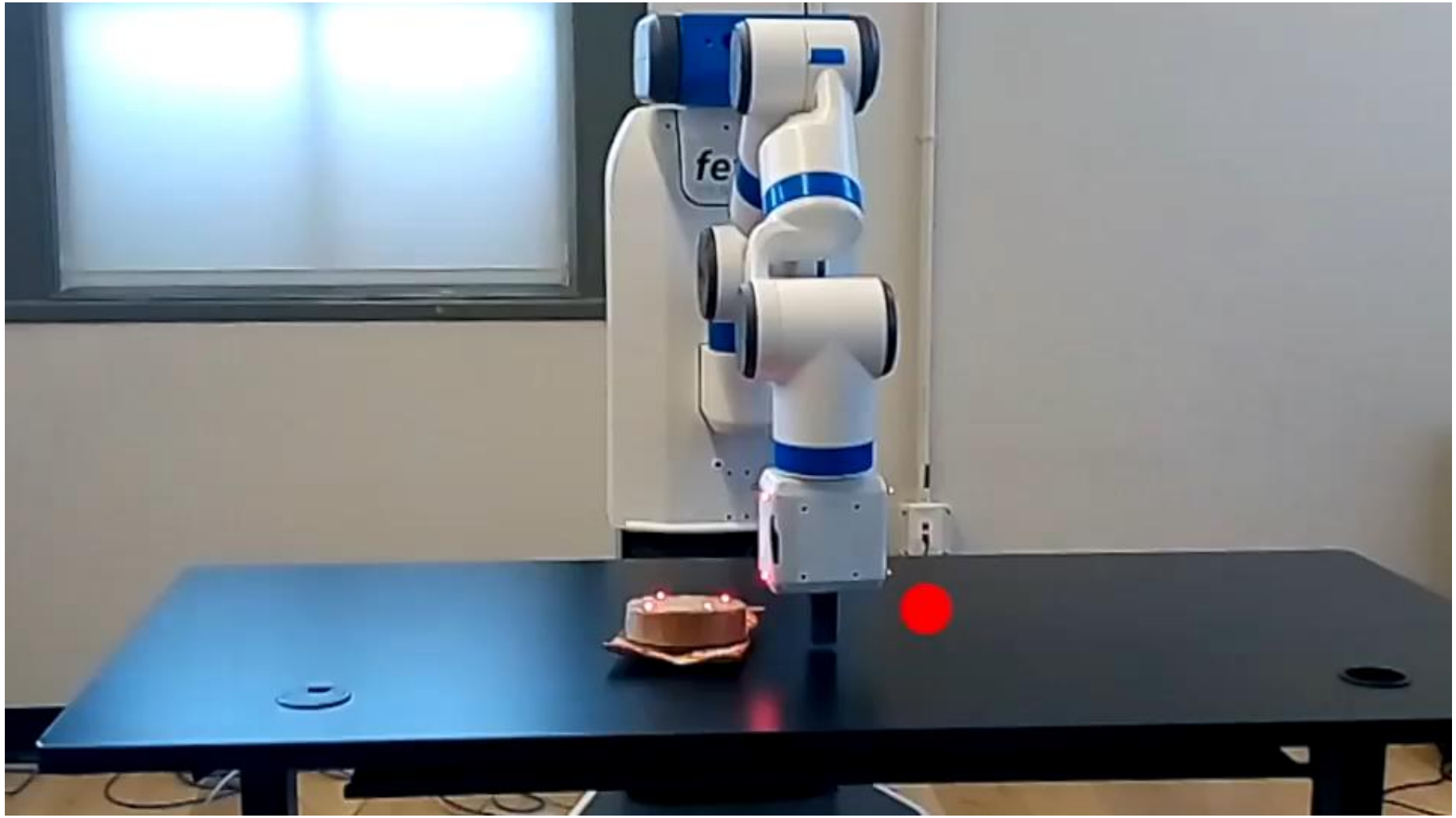
Rajeswaran et al., “EPOpt: Learning robust neural network policies…”

Preparing for the unknown: explicit system ID



Yu et al., “Preparing for the Unknown: Learning a Universal Policy with Online System Identification”

Another example



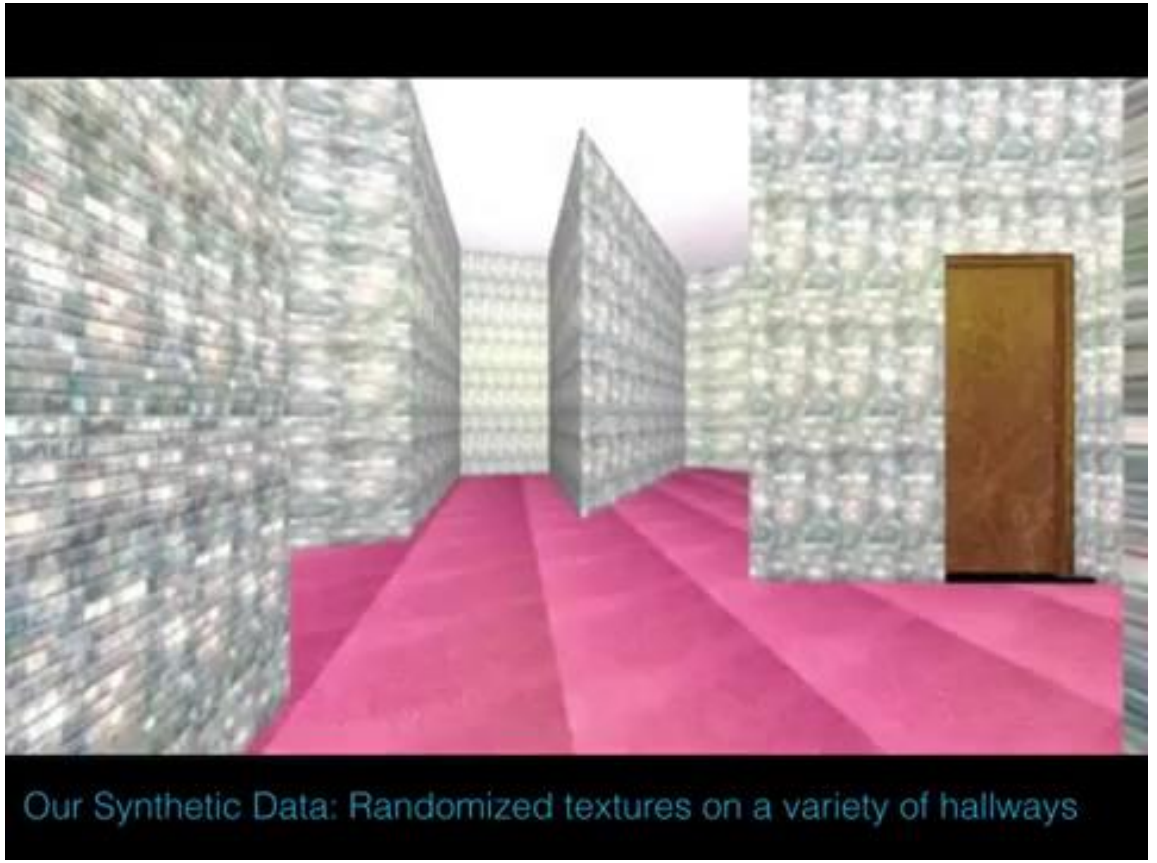
Xue Bin Peng et al., “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”

CAD2RL: randomization for real-world control

also called domain randomization

Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”

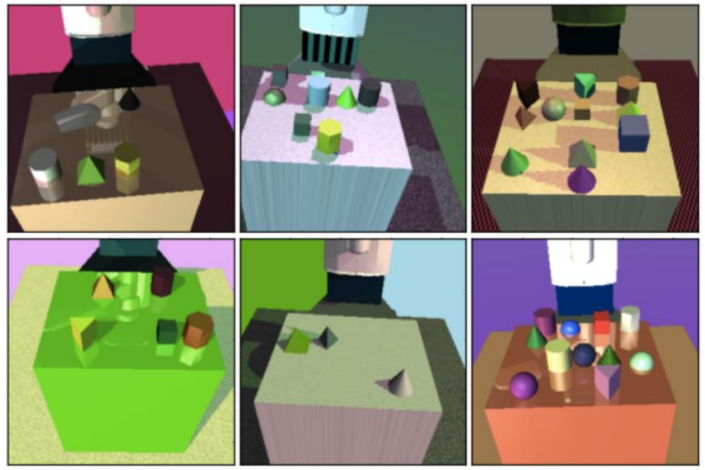
CAD2RL: randomization for real-world control



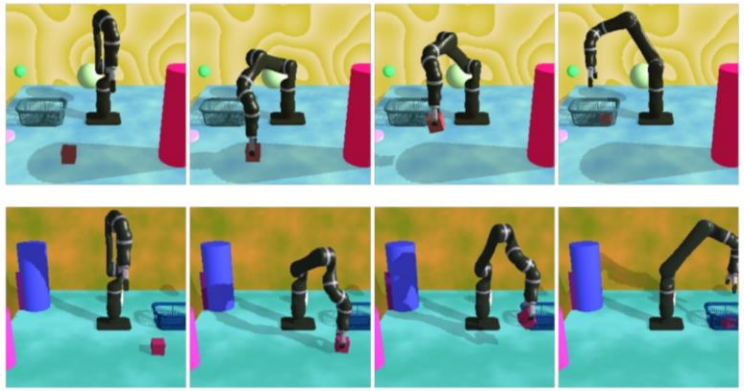
Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”

Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”

Randomization for manipulation



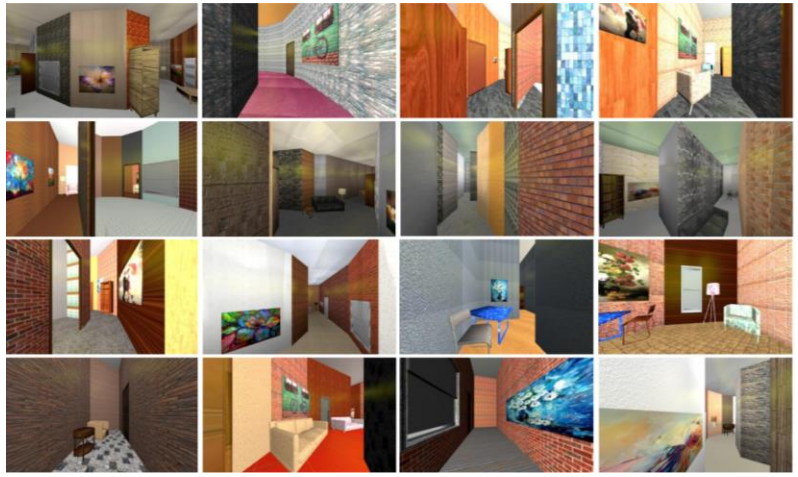
Tobin, Fong, Ray, Schneider, Zaremba, Abbeel



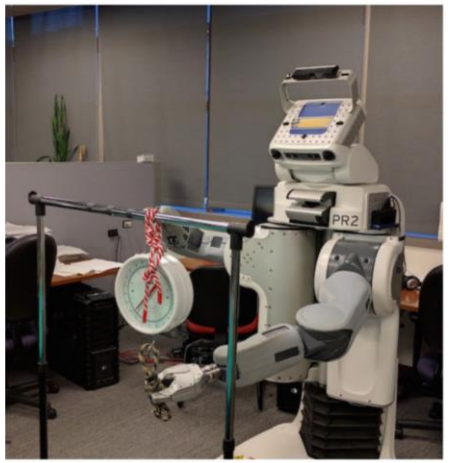
James, Davison, Johns

What if we can peek at the target domain?

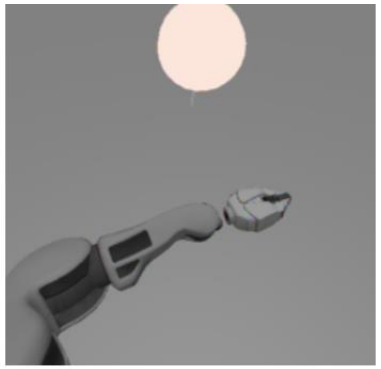
- So far: pure 0-shot transfer: learn in source domain so that we can succeed in **unknown** target domain
- Not possible in general: if we know nothing about the target domain, the best we can do is be as robust as possible
- What if we saw a few images of the target domain?



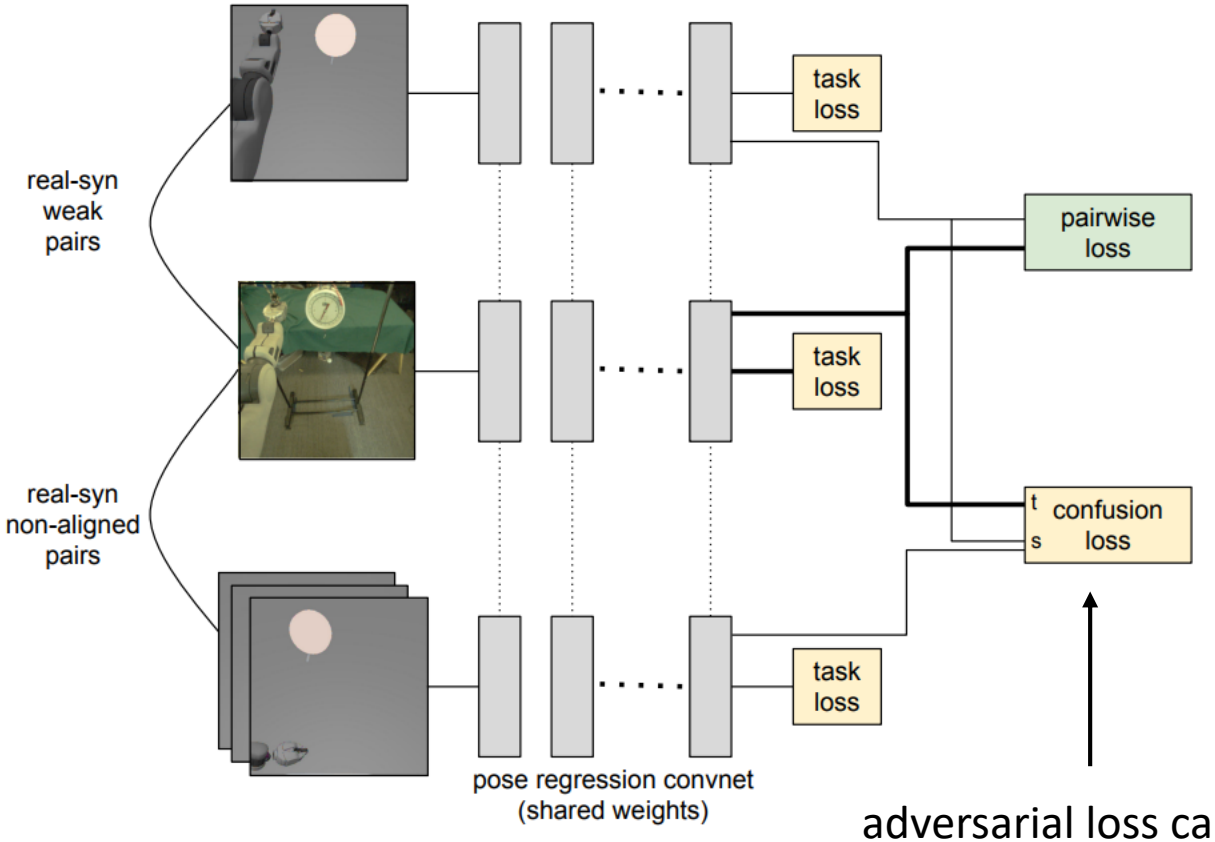
Better transfer through domain adaptation



simulated images



real images

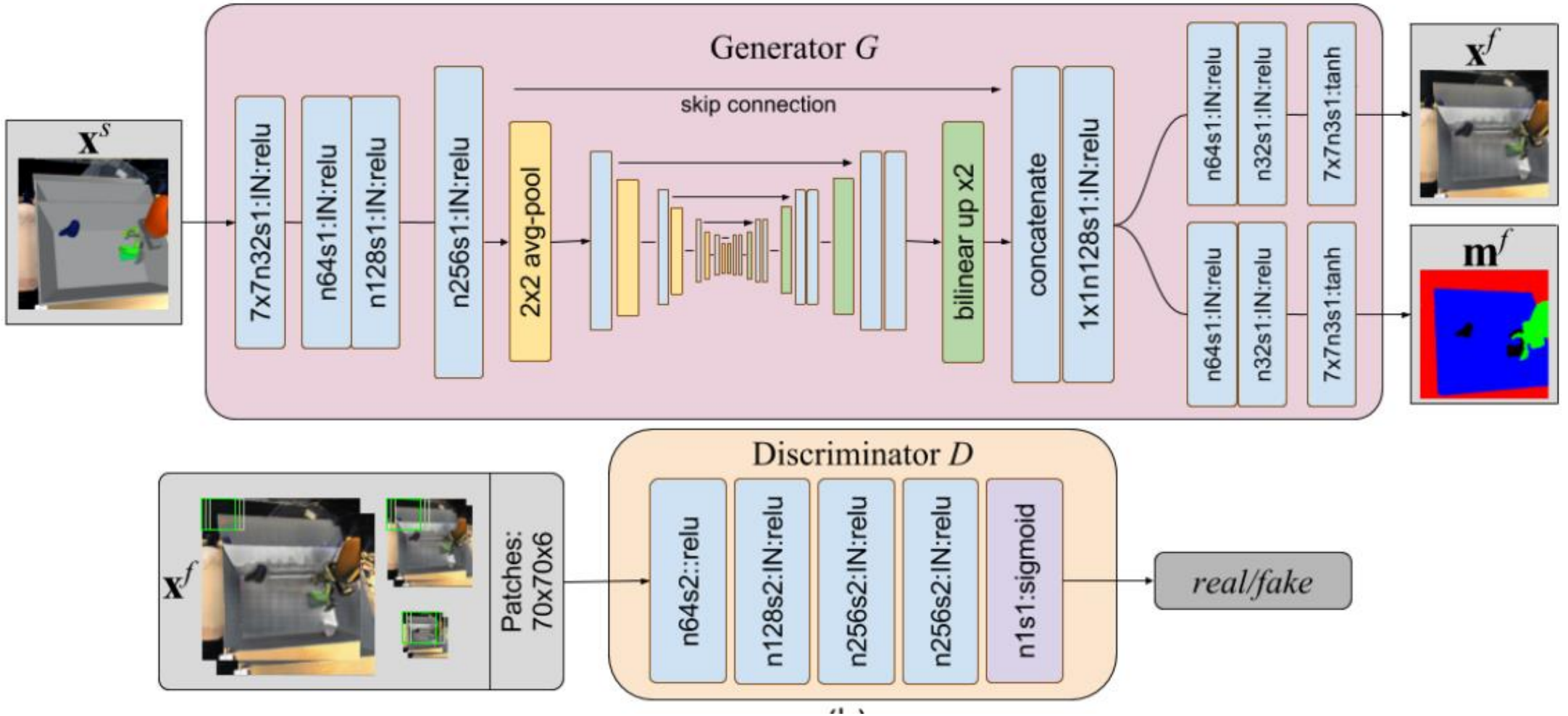


adversarial loss causes internal CNN features to be indistinguishable for sim and real

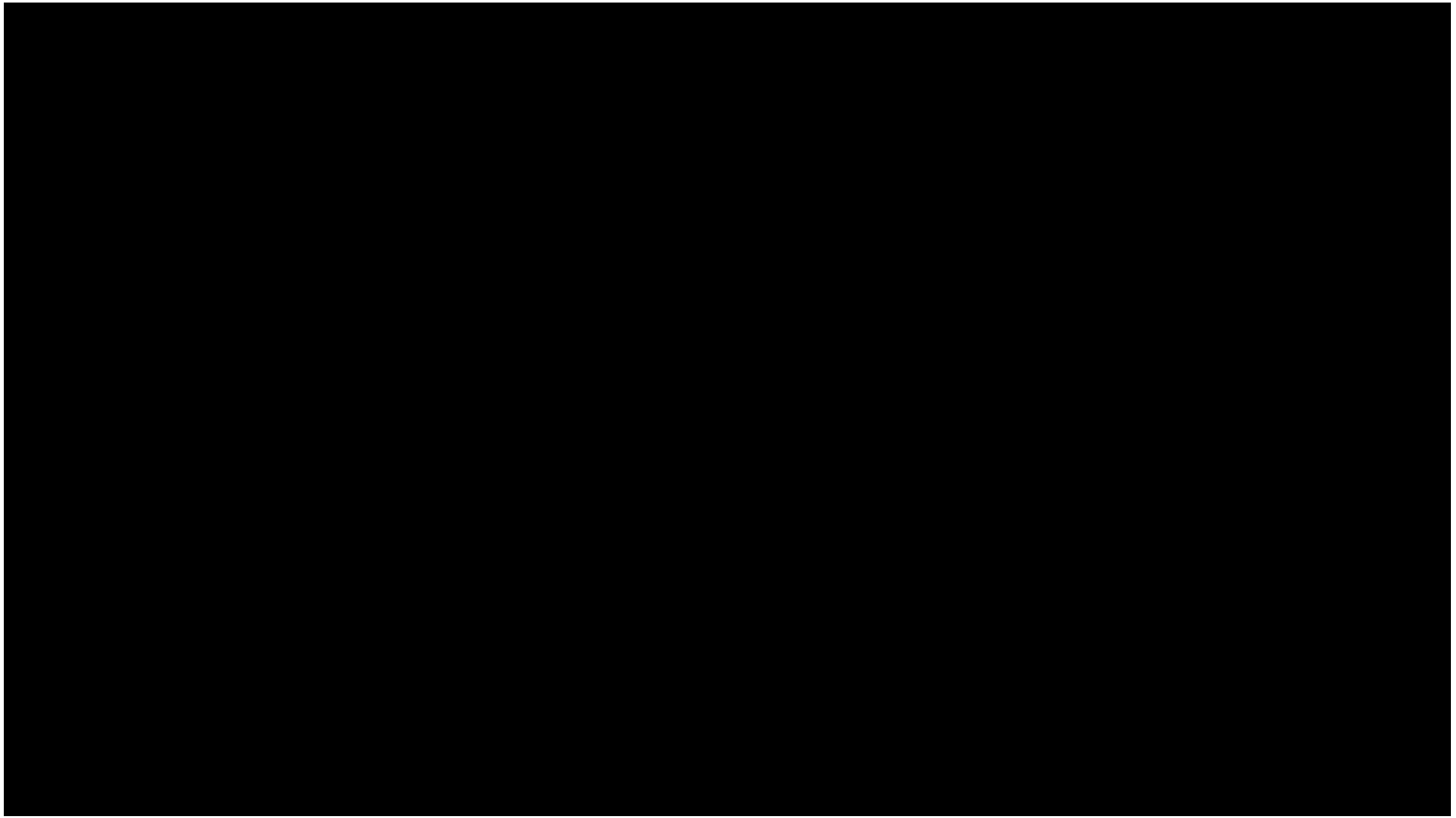
Tzeng*, Devin*, et al., “Adapting Visuomotor Representations with Weak Pairwise Constraints”

Domain adaptation at the pixel level

can we *learn* to turn synthetic images into *realistic* ones?



Bousmalis et al., “Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping”



Bousmalis et al., “Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping”

Forward transfer summary

- Pretraining and finetuning
  - Standard finetuning with RL is hard
  - Maximum entropy formulation can help
- How can we modify the source domain for transfer?
  - Randomization can help a lot: the more diverse the better!
- How can we use modest amounts of target domain data?
  - Domain adaptation: make the network unable to distinguish observations from the two domains
  - …or modify the source domain observations to look like target domain
  - Only provides **invariance** – assumes all differences are functionally irrelevant; this is not always enough!

Source domain randomization and domain adaptation suggested readings

Rajeswaran, et al. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.**

Yu et al. (2017). **Preparing for the Unknown: Learning a Universal Policy with Online System Identification.**

Sadeghi & Levine. (2017). **CAD2RL: Real Single Image Flight without a Single Real Image.**

Tobin et al. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.**

James et al. (2017). **Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task.**

Tzeng*, Devin*, et al. (2016). **Adapting Deep Visuomotor Representations with Weak Pairwise Constraints.**

Bousmalis et al. (2017). **Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping.**

… and many many others!

Break

How can we frame transfer learning problems?

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetune on the new task
   c) Randomize source task domain
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Model-based reinforcement learning
   b) Model distillation
   c) Contextual policies
   d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

Multiple source domains

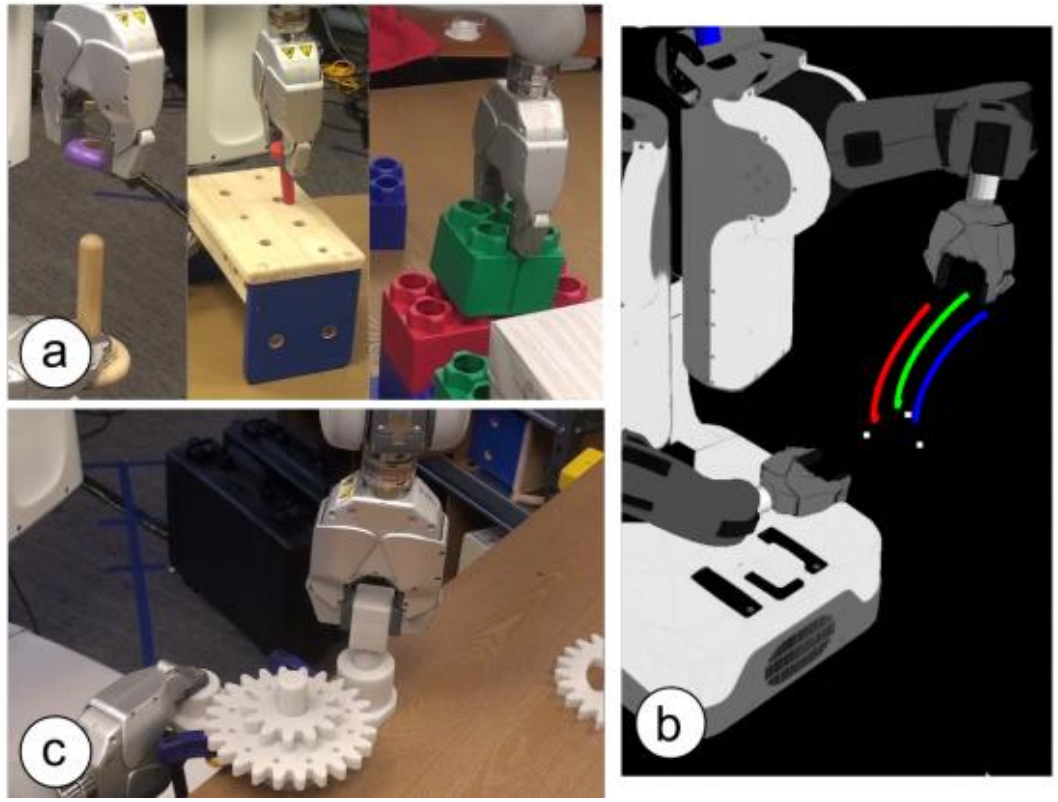
- So far: more diversity = better transfer
- Need to design this diversity
  - E.g., simulation to real world transfer: randomize the simulation
- What if we transfer from multiple *different* tasks?
  - In a sense, closer to what people do: build on a lifetime of experience
  - Substantially harder: past tasks don't directly tell us how to solve the task in the target domain!

Model-based reinforcement learning

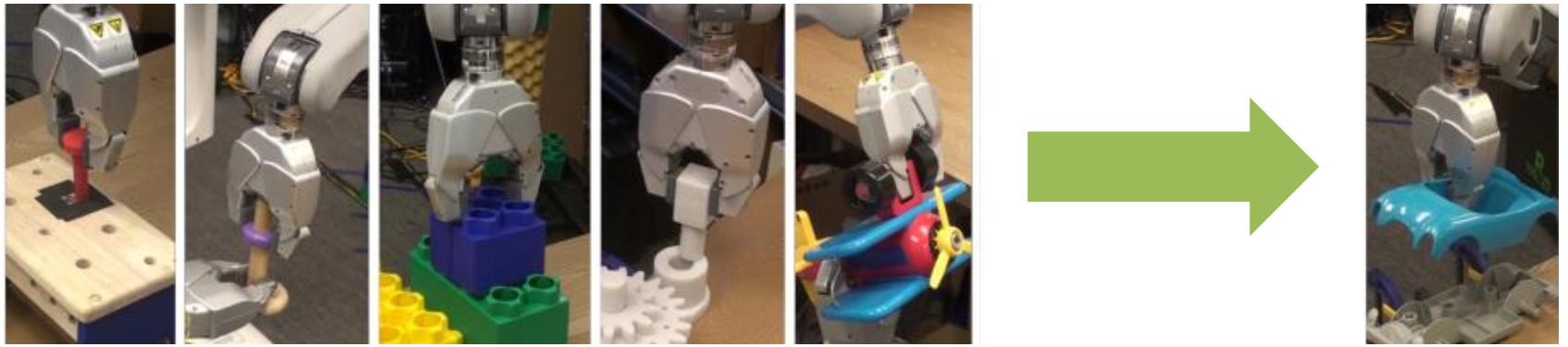
- If the past tasks are all different, what do they have in common?
- Idea 1: the laws of physics
  - Same robot doing different chores
  - Same car driving to different destinations
  - Trying to accomplish different things in the same open-ended video game
- Simple version: train model on past tasks, and then use it to solve new tasks
- More complex version: adapt or finetune the model to new task
  - Easier than finetuning the policy if task is very different but physics are mostly the same

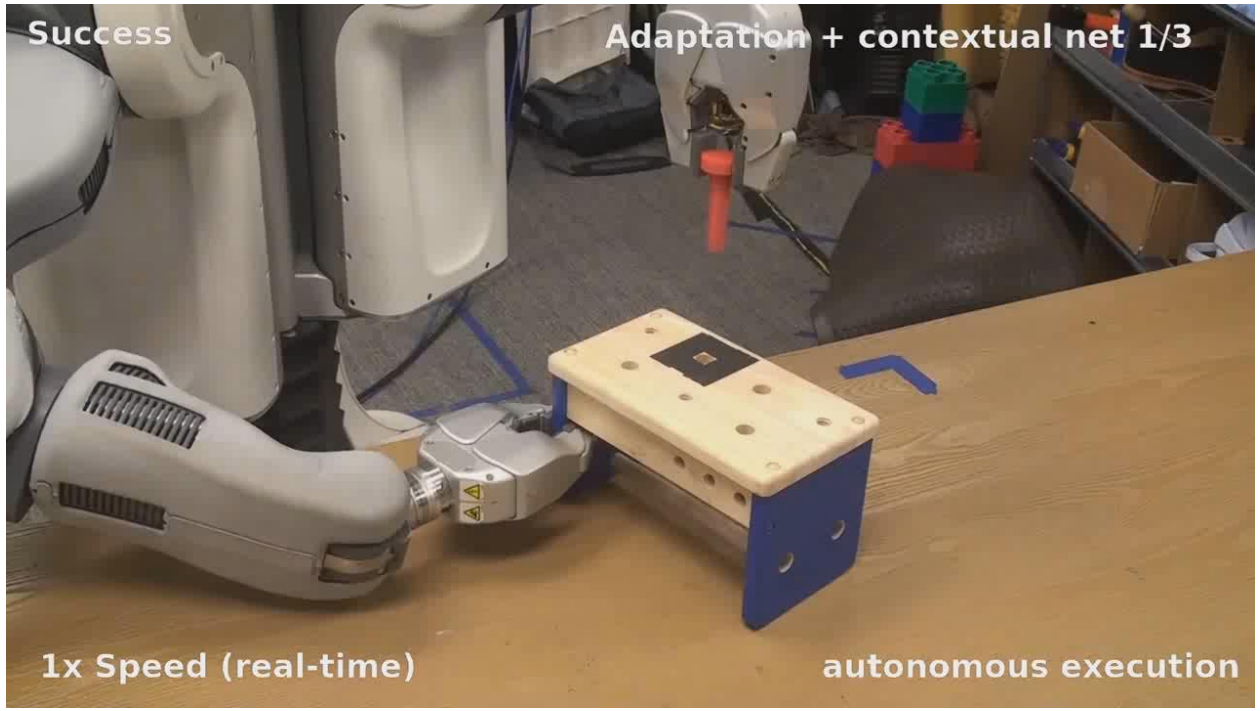
Model-based reinforcement learning

Example: 1-shot learning with model priors



Fu et al., “One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation…”

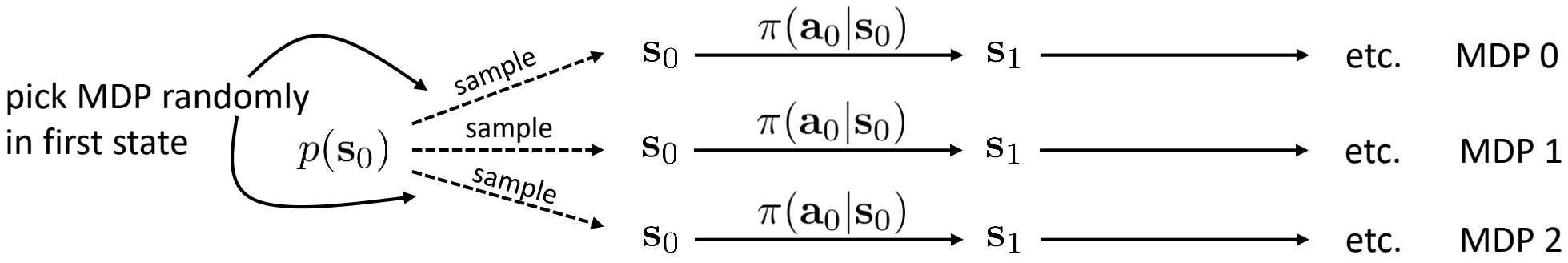




Fu et al., “One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation…”

Can we solve multiple tasks at once?

- Sometimes learning a model is very hard
- Can we learn a multi-task policy that can *simultaneously* perform many tasks?
- This policy might then be easier to finetune to new tasks
- Idea 1: construct a joint MDP



- Idea 2: train in each MDP separately, and then combine the policies

Actor-mimic and policy distillation

Goal: learn a single policy that can play all Atari games

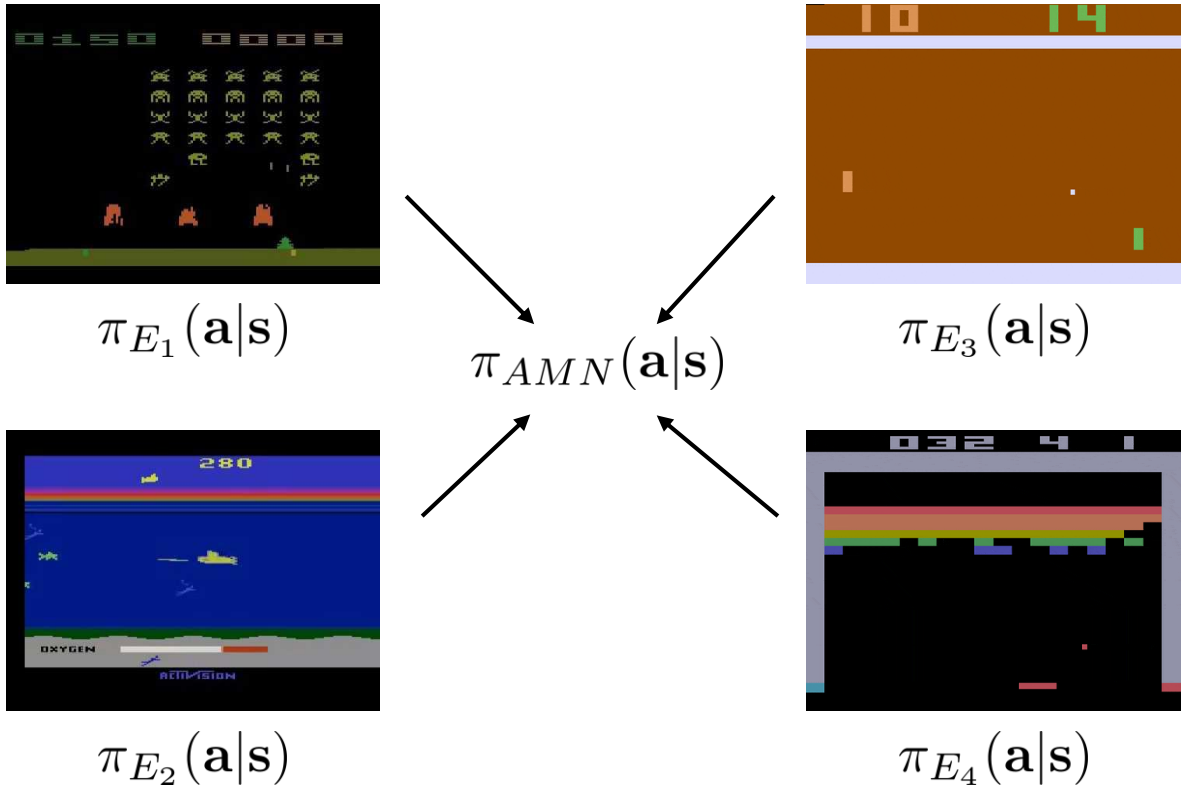
POLICY DISTILLATION

Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre, Guillaume Desjardins,
James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu & Raia Hadsel
Google DeepMind

ACTOR-MIMIC DEEP MULTITASK AND TRANSFER REINFORCEMENT LEARNING

Emilio Parisotto, Jimmy Ba, Ruslan Salakhutdinov
Department of Computer Science
University of Toronto

Distillation for Multi-Task Transfer



$$\mathcal{L} = \sum_{\mathbf{a}} \pi_{E_i}(\mathbf{a}|\mathbf{s}) \log \pi_{AMN}(\mathbf{a}|\mathbf{s})$$

(just supervised learning/distillation)

analogous to guided policy search, but for transfer learning

-> see model-based RL slides

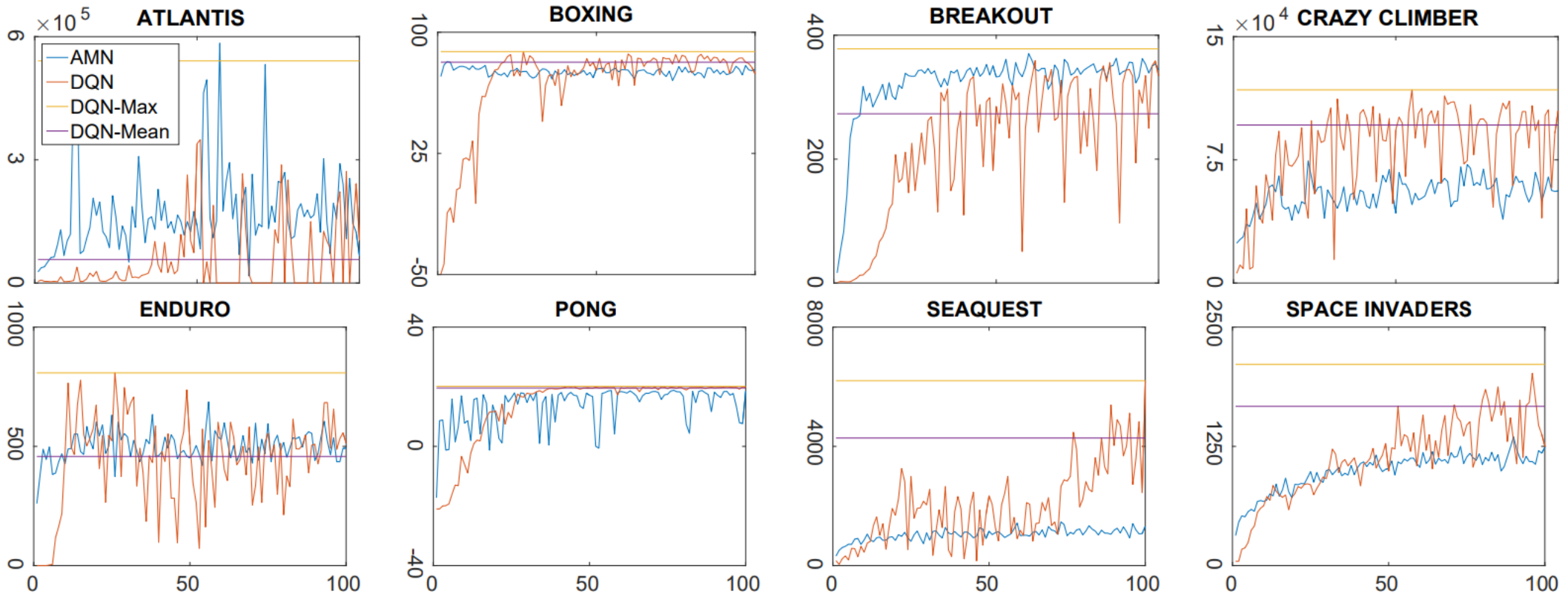
some other details

(e.g., feature regression objective)

– see paper

Parisotto et al. “Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning”

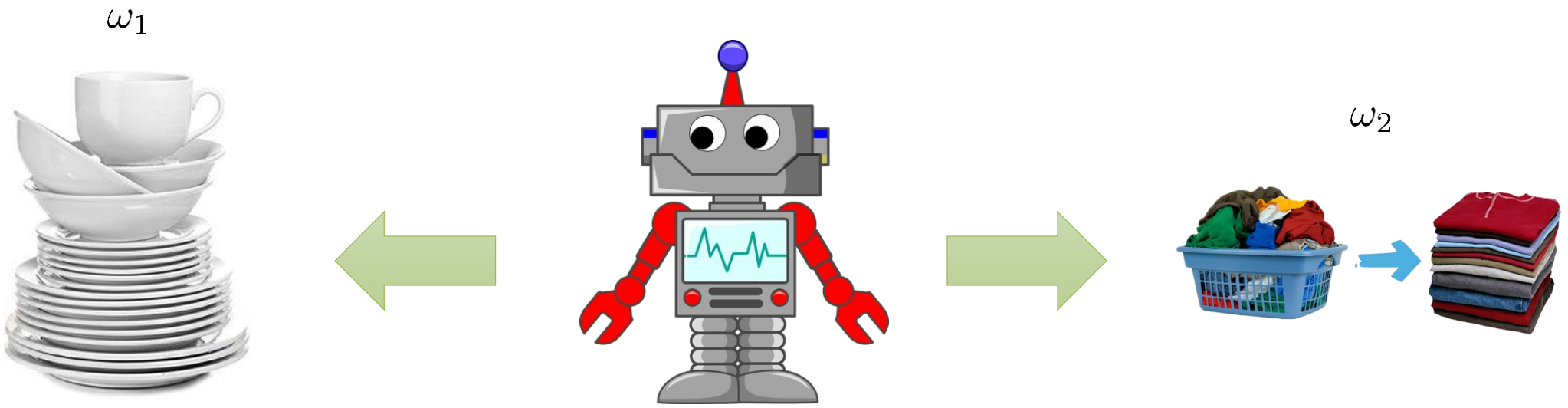
Distillation Transfer Results



Parisotto et al. “Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning”

How does the model know what to do?

- So far: what to do is apparent from the input (e.g., which game is being played)
- What if the policy can do *multiple* things in the *same* environment?

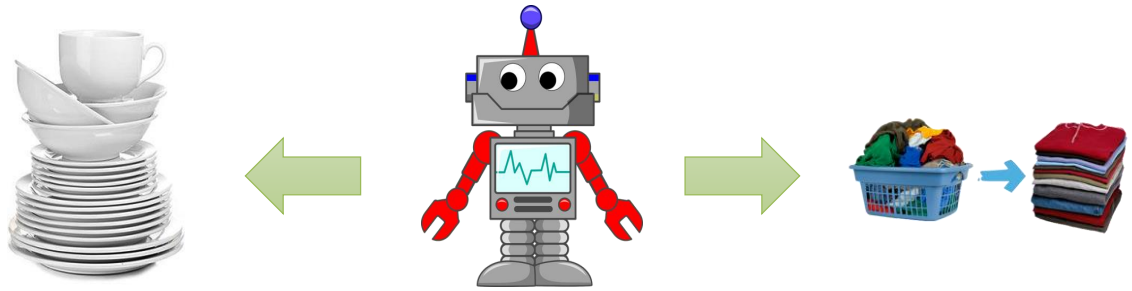


Contextual policies

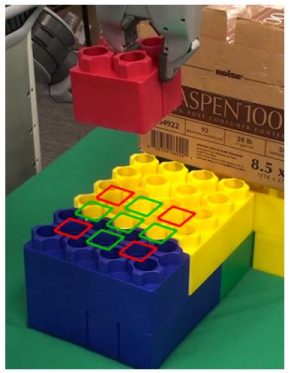
standard policy: $\pi_\theta(\mathbf{a}|\mathbf{s})$

contextual policy: $\pi_\theta(\mathbf{a}|\mathbf{s}, \omega)$

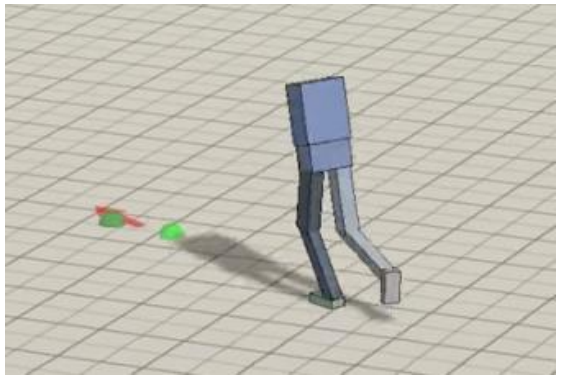
e.g., do dishes or laundry



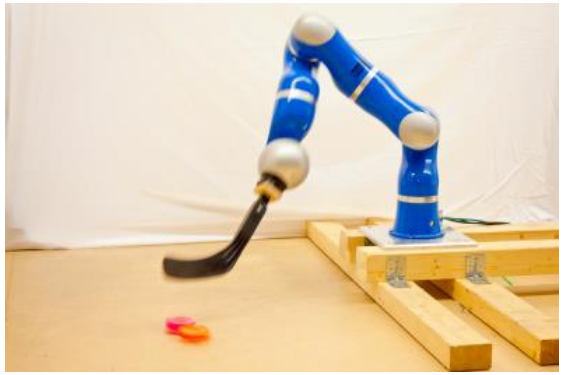
formally, simply defines augmented state space: $\tilde{\mathbf{s}} = \begin{bmatrix} \mathbf{s} \\ \omega \end{bmatrix}$ $\tilde{\mathcal{S}} = \mathcal{S} \times \Omega$



ω: stack location



ω: walking direction



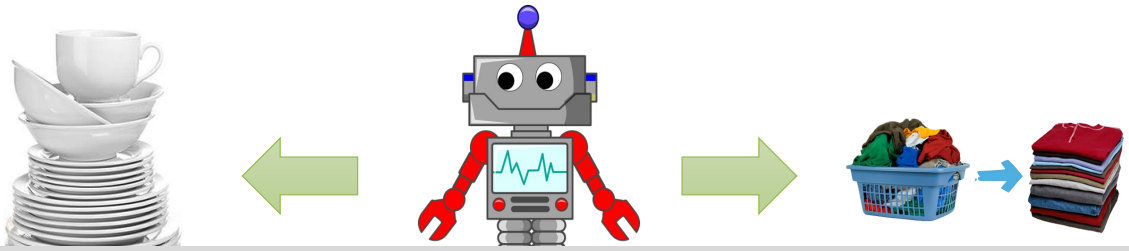
ω: where to hit puck

images: Peng, van de Panne, Peters

Contextual policies

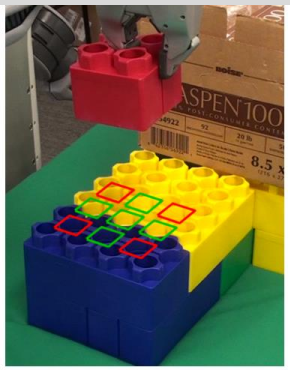
standard policy: $\pi_\theta(\mathbf{a}|\mathbf{s})$

contextual policy: $\pi_\theta(\mathbf{a}|\mathbf{s}, \omega)$

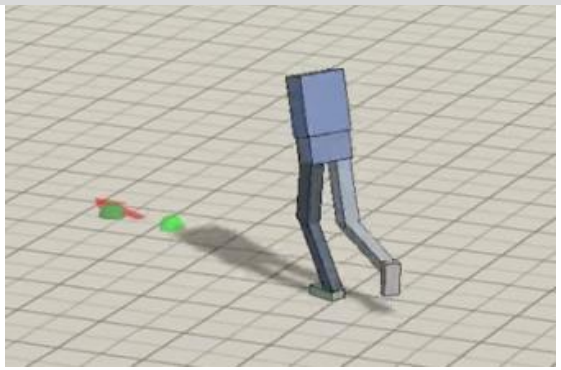


forr

will discuss more in the context of meta-learning!



ω: stack location



ω: walking direction

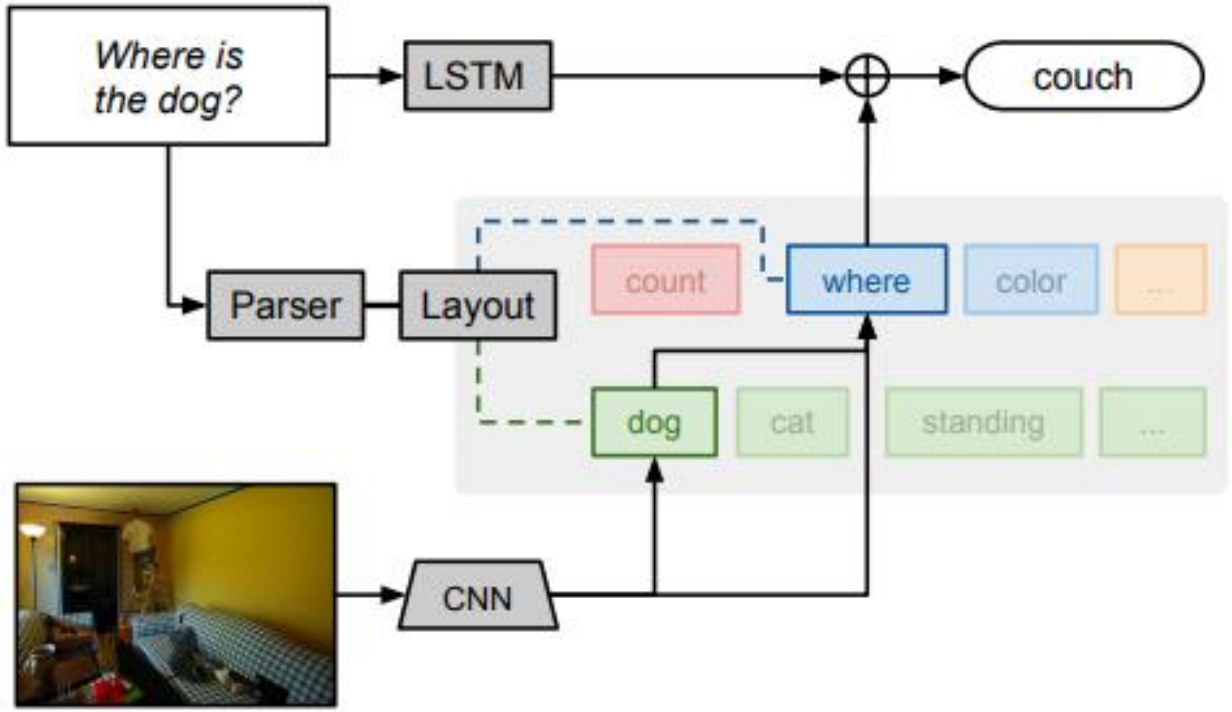
ω: where to hit puck

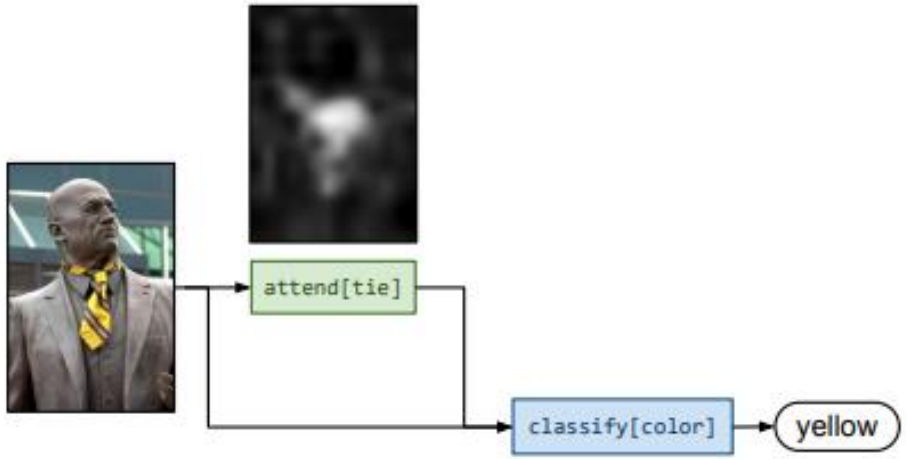
images: Peng, van de Panne, Peters

Architectures for multi-task transfer

- So far: single neural network for all tasks (in the end)
- What if tasks have some shared parts and some distinct parts?
  - Example: two cars, one with camera and one with LIDAR, driving in two different cities
  - Example: ten different robots trying to do ten different tasks
- Can we design architectures with *reusable components*?

Modular networks in deep learning

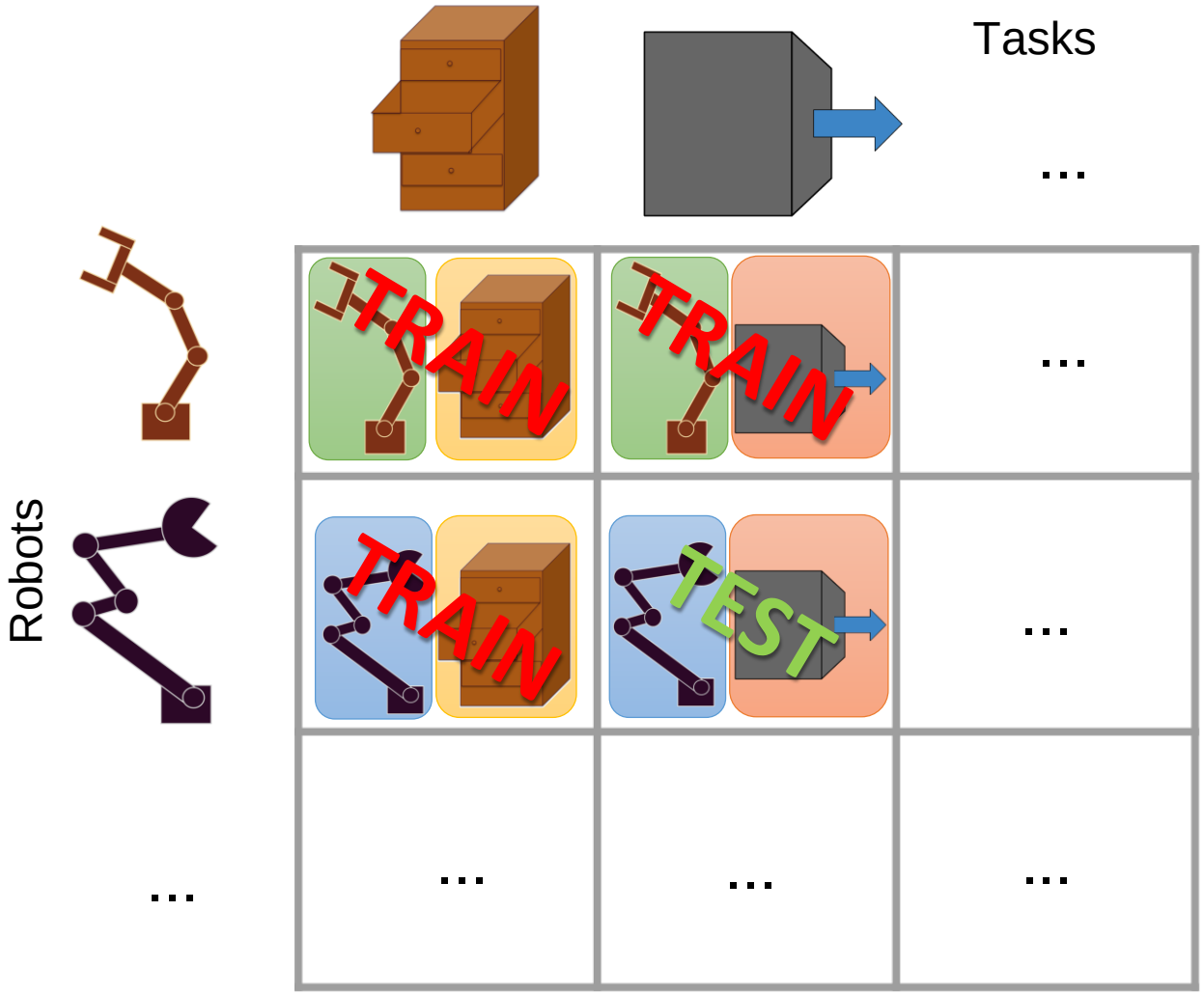


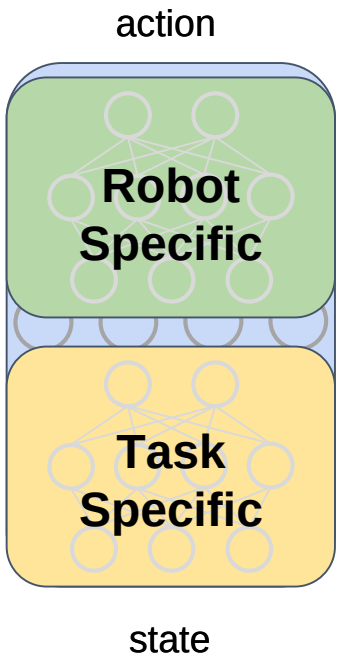


(a) NMN for answering the question *What color is his tie?* The `attend[tie]` module first predicts a heatmap corresponding to the location of the tie. Next, the `classify[color]` module uses this heatmap to produce a weighted average of image features, which are finally used to predict an output label.

Andreas et al. “Neural Module Networks.” 2015

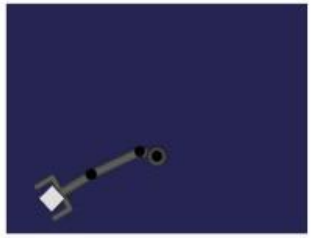
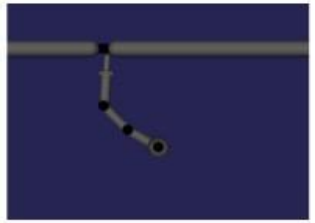
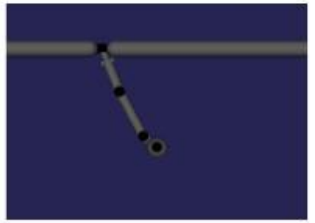
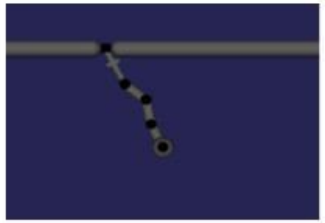
Modular networks in RL

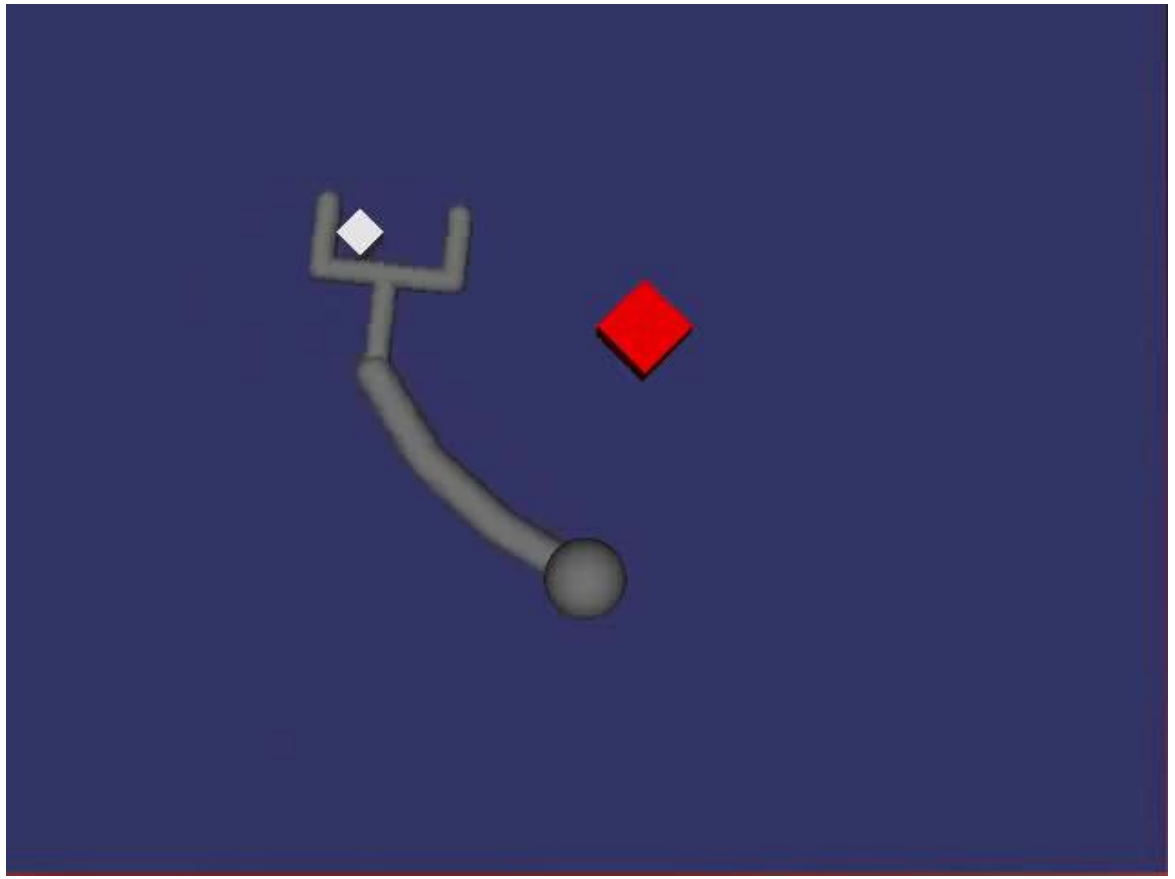




Devin*, Gupta*, et al. “Learning Modular Neural Network Policies…”

Modular networks in RL

Robots / Tasks	3link	3link different config	4link
Reach			
Push			Unseen World
Peg insert			



Multi-task learning summary

- More tasks = more diversity = better transfer
- Often easier to obtain multiple different but relevant prior tasks
- Model-based RL: transfer the physics, not the behavior
- Distillation: combine multiple policies into one, for concurrent multi-task learning (accelerate all tasks through sharing)
- Contextual policies: policies that are told *what* to do
- Architectures for multi-task learning: modular networks

Suggested readings

Fu etal. (2016). **One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors.**

Rusu et al. (2016). **Policy Distillation.**

Parisotto et al. (2016). **Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning.**

Devin*, Gupta*, et al. (2017). **Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer.**

How can we frame transfer learning problems?

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetu
   c) Archit
   d) Rando
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Model-based reinforcement learning
   b) Model distillation
   c) Contextual policies
   d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

more on this next time!