Exploration (Part 2) and Transfer Learning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

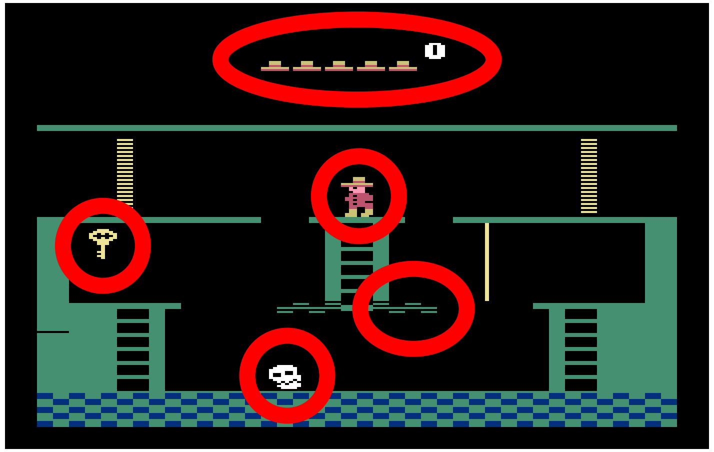
Class Notes

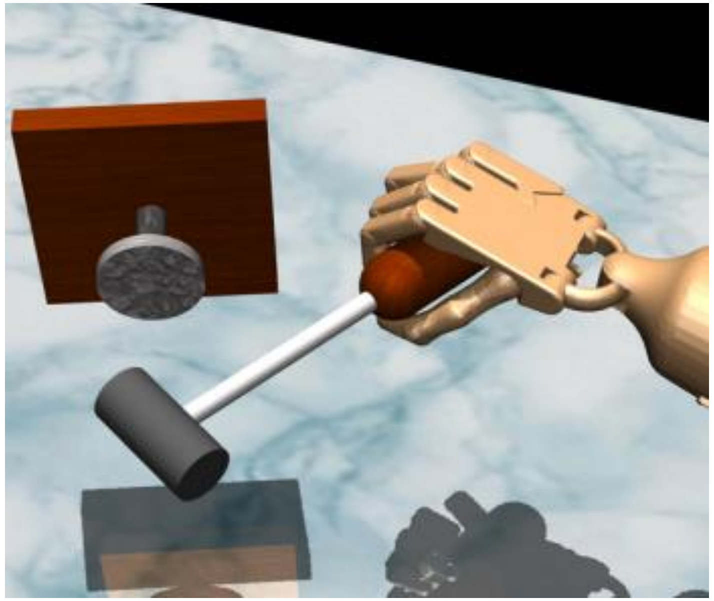
1. Homework 4 due today! Last one!

Recap: classes of exploration methods in deep RL

- Optimistic exploration:
  - new state = good state
  - requires estimating state visitation frequencies or novelty
  - typically realized by means of exploration bonuses
- Thompson sampling style algorithms:
  - learn distribution over Q-functions or policies
  - sample and act according to sample
- Information gain style algorithms
  - reason about information gain from visiting new states

Recap: exploring with pseudo-counts





fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far

take a step i and observe $\mathbf{s}_i$

fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \cup \mathbf{s}_i$

use $p_\theta(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$

set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$ ← "pseudo-count"

how to get $\hat{N}(\mathbf{s})$? use the equations

$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \qquad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)+1}{\hat{n}+1}$$

two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_\theta(\mathbf{s}_i) \qquad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_\theta(\mathbf{s}_i)} p_\theta(\mathbf{s}_i)$$

Bellemare et al. "Unifying Count-Based Exploration…"

Posterior sampling in deep RL

Thompson sampling:

$$\theta_1, \ldots, \theta_n \sim \hat{p}(\theta_1, \ldots, \theta_n)$$

$$a = \arg\max_a E_{\theta_a}[r(a)]$$

What do we sample?

How do we represent the distribution?

bandit setting: $\hat{p}(\theta_1, \ldots, \theta_n)$ is distribution over *rewards*

MDP analog is the Q-function!

1. sample Q-function Q from $p(Q)$
2. act according to Q for one episode
3. update $p(Q)$

since Q-learning is off-policy, we don't care which Q-function was used to collect data

how can we represent a distribution over functions?

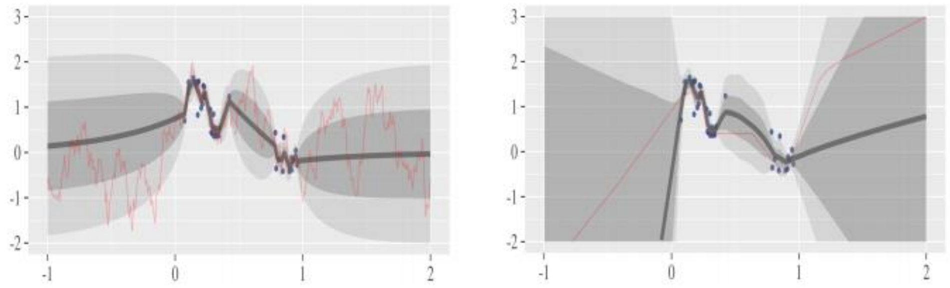
Osband et al. "Deep Exploration via Bootstrapped DQN"

Bootstrap

given a dataset $\mathcal{D}$, resample with replacement N times to get $\mathcal{D}_1, \dots, \mathcal{D}_N$

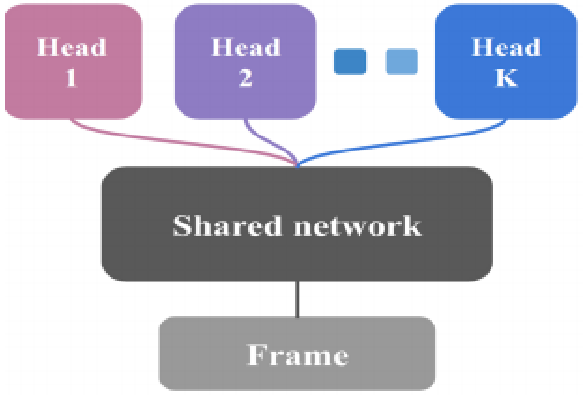
train each model f_{θ_i} on $\mathcal{D}_i$

to sample from $p(\theta)$, sample $i \in [1, \dots, N]$ and use f_{θ_i}



(b) Gaussian process posterior (c) Bootstrapped neural nets

training N big neural nets is expensive, can we avoid it?

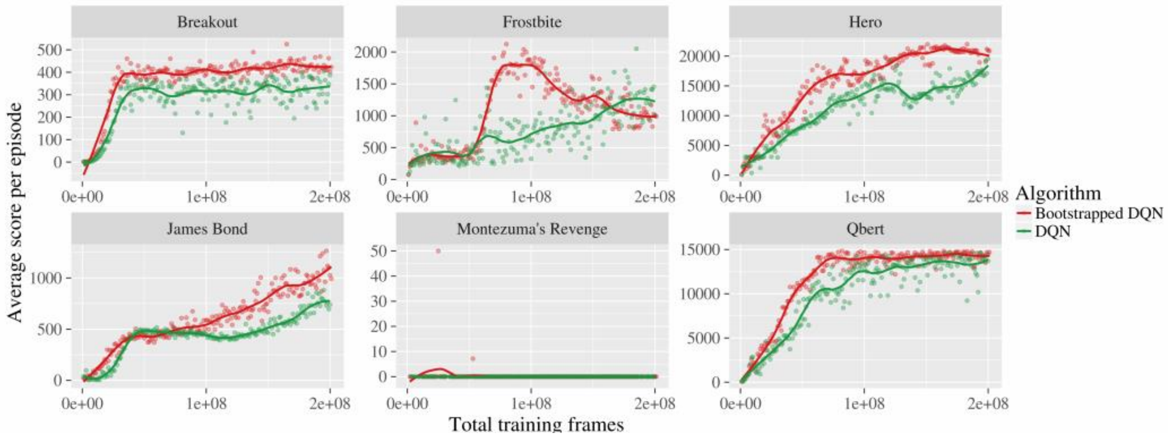


Osband et al. "Deep Exploration via Bootstrapped DQN"

Why does this work?

Exploring with random actions (e.g., epsilon-greedy): oscillate back and forth, might not go to a coherent or interesting place

Exploring with random Q-functions: commit to a randomized but internally consistent strategy for an entire episode



+ no change to original reward function

- very good bonuses often do better

Osband et al. "Deep Exploration via Bootstrapped DQN"

Reasoning about information gain (approximately)

Info gain: $\mathrm{IG}(z, y|a)$

information gain about *what*?	
information gain about reward $r(\mathbf{s}, \mathbf{a})$?	not very useful if reward is sparse
state density $p(\mathbf{s})$?	a bit strange, but actually makes sense!
information gain about dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$?	good proxy for *learning* the MDP, though still heuristic

Generally intractable to use exactly, regardless of what is being estimated!

Reasoning about information gain (approximately)

Generally intractable to use exactly, regardless of what is being estimated

A few approximations:

prediction gain: $\log p_{\theta'}(\mathbf{s}) - \log p_{\theta}(\mathbf{s})$ (Schmidhuber '91, Bellemare '16)

intuition: if density changed a lot, the state was novel

variational inference: (Houthooft et al. "VIME")

IG can be equivalently written as $D_{\mathrm{KL}}(p(\theta|h, \underline{s_t, a_t, s_{t+1}})\|p(\theta|h))$

model parameters for $p_\theta(s_{t+1}|s_t, a_t)$

newly observed transition

history of all prior transitions

intuition: a transition is more informative if it causes belief over θ to change

idea: use variational inference to estimate $q(\theta|\phi) \approx p(\theta|h)$

given new transition (s, a, s'), update ϕ to get ϕ'

Reasoning about information gain (approximately)

VIME implementation:

IG can be equivalently written as $D_{\mathrm{KL}}(p(\theta|h, \underline{s_t, a_t, s_{t+1}})\|p(\theta|h))$

model parameters for $p_\theta(s_{t+1}|s_t, a_t)$

newly observed transition

history of all prior transitions

$q(\theta|\phi) \approx p(\theta|h)$ specifically, optimize variational lower bound $D_{\mathrm{KL}}(q(\theta|\phi)\|p(h|\theta)p(\theta))$

represent $q(\theta|\phi)$ as product of independent Gaussian parameter distributions
with mean ϕ (see Blundell et al. "Weight uncertainty in neural networks")

given new transition (s, a, s'), update ϕ to get ϕ'
this corresponds to updating the network weight means and variances

use $D_{\mathrm{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ as approximate bonus

Houthooft et al. "VIME"

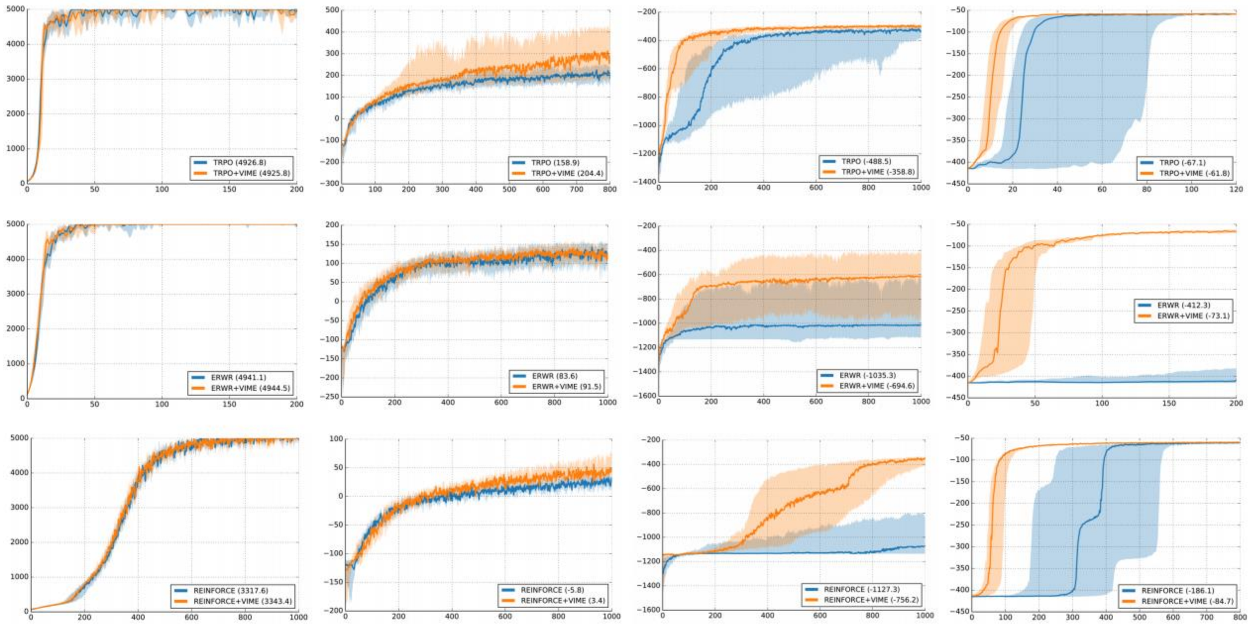
Reasoning about information gain (approximately)

VIME implementation:

IG can be equivalently written as $D_{\mathrm{KL}}(p(\theta|h, s_t, a_t, s_{t+1}) \| p(\theta|h))$

$q(\theta|\phi) \approx p(\theta|h)$ specifically, optimize variational lower bound $D_{\mathrm{KL}}(q(\theta|\phi) \| p(h|\theta)p(\theta))$

use $D_{\mathrm{KL}}(q(\theta|\phi') \| q(\theta|\phi))$ as approximate bonus



(a) CartPole (b) CartPoleSwingup (c) DoublePendulum (d) MountainCar

Approximate IG:

+ appealing mathematical formalism

- models are more complex, generally harder to use effectively

Houthooft et al. “VIME”

Exploration with model errors

$D_{\mathrm{KL}}(q(\theta|\phi')\|q(\theta|\phi))$ can be seen as change in network (mean) parameters ϕ

if we forget about IG, there are many other ways to measure this

Stadie et al. 2015:

- encode image observations using auto-encoder
- build predictive model on auto-encoder latent states
- use model error as exploration bonus

Schmidhuber et al. (see, e.g. “Formal Theory of Creativity, Fun, and Intrinsic Motivation):

- exploration bonus for model error
- exploration bonus for model gradient
- many other variations

Many others!

Suggested readings

Schmidhuber. (1992). **A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers.**

Stadie, Levine, Abbeel (2015). **Incentivizing Exploration in Reinforcement Learning with Deep Predictive Models.**

Osband, Blundell, Pritzel, Van Roy. (2016). **Deep Exploration via Bootstrapped DQN.**

Houthooft, Chen, Duan, Schulman, De Turck, Abbeel. (2016). **VIME: Variational Information Maximizing Exploration.**

Bellemare, Srinivasan, Ostroviski, Schaul, Saxton, Munos. (2016). **Unifying Count-Based Exploration and Intrinsic Motivation.**

Tang, Houthooft, Foote, Stooke, Chen, Duan, Schulman, De Turck, Abbeel. (2016). **#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning.**

Fu, Co-Reyes, Levine. (2017). **EX2: Exploration with Exemplar Models for Deep Reinforcement Learning.**

Next: transfer learning and meta-learning

1. The benefits of sharing knowledge across tasks
2. The transfer learning problem in RL
3. The meta-learning problem statement, algorithms

- Goals:
  - Understand how reinforcement learning algorithms can benefit from structure learned on prior tasks
  - Understand prior work on transfer learning
  - Understand meta-learning, how it differs from transfer learning

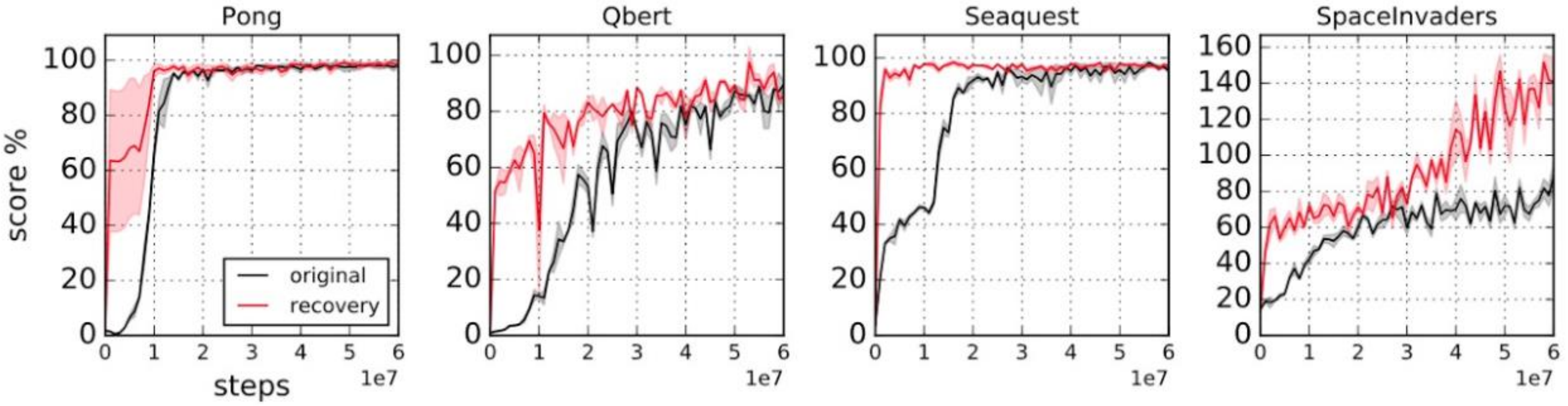
Back to Montezuma's Revenge

- 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!
  - 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

slide adapted from C. Finn

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) Architectures for transfer: progressive networks
   c) Finetune on the new task
   d) 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

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) Architectures for transfer: progressive networks
   d) 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

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

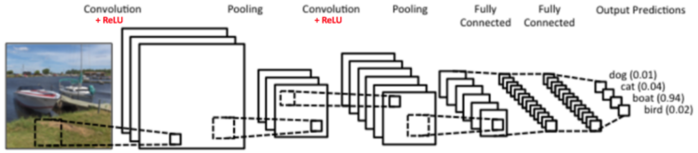
Policies trained for one set of circumstances might just work in a new domain, but no promises or guarantees

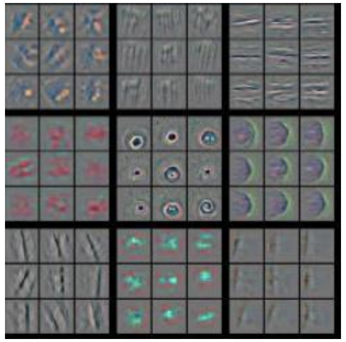
Levine*, Finn*, et al. '16

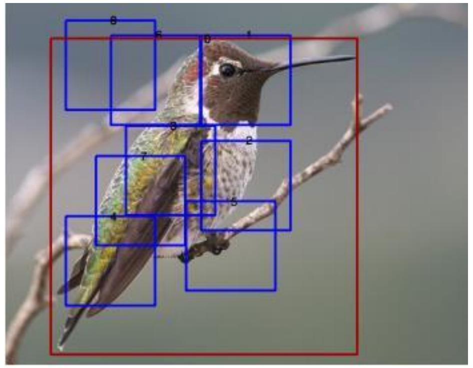
Devin et al. '17

Finetuning

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



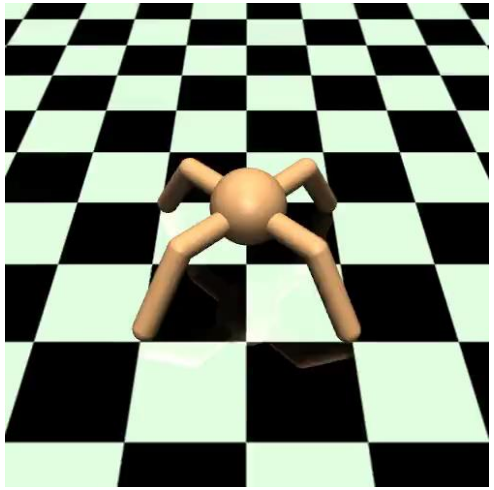




Where are the “ImageNet” features of RL?

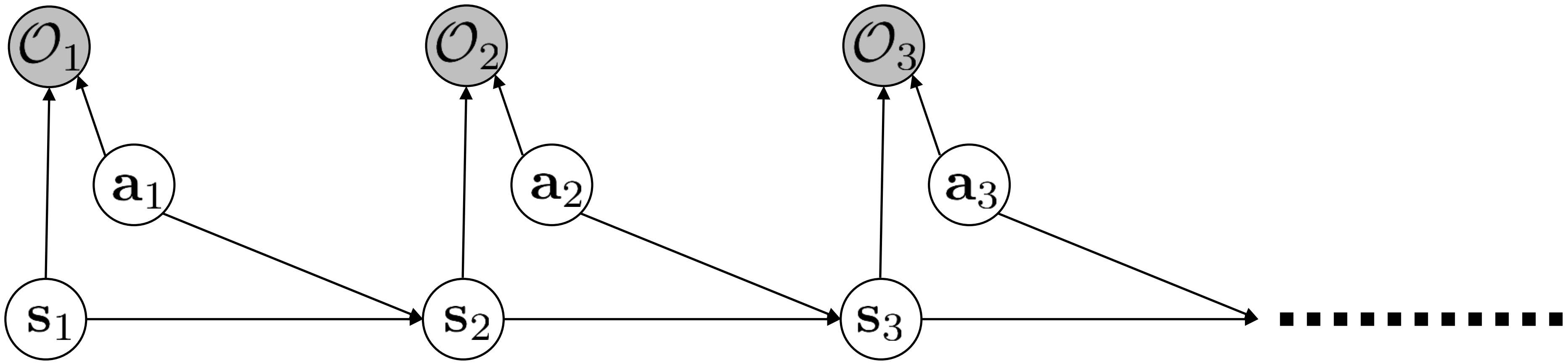
Challenges with finetuning in RL

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



Finetuning with maximum-entropy policies

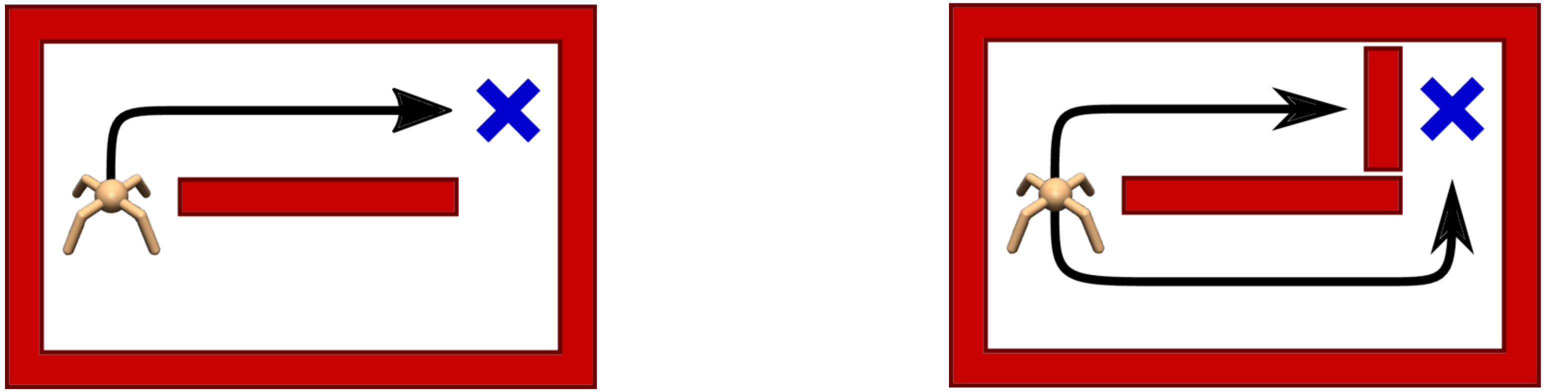
How can we increase diversity and entropy?



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

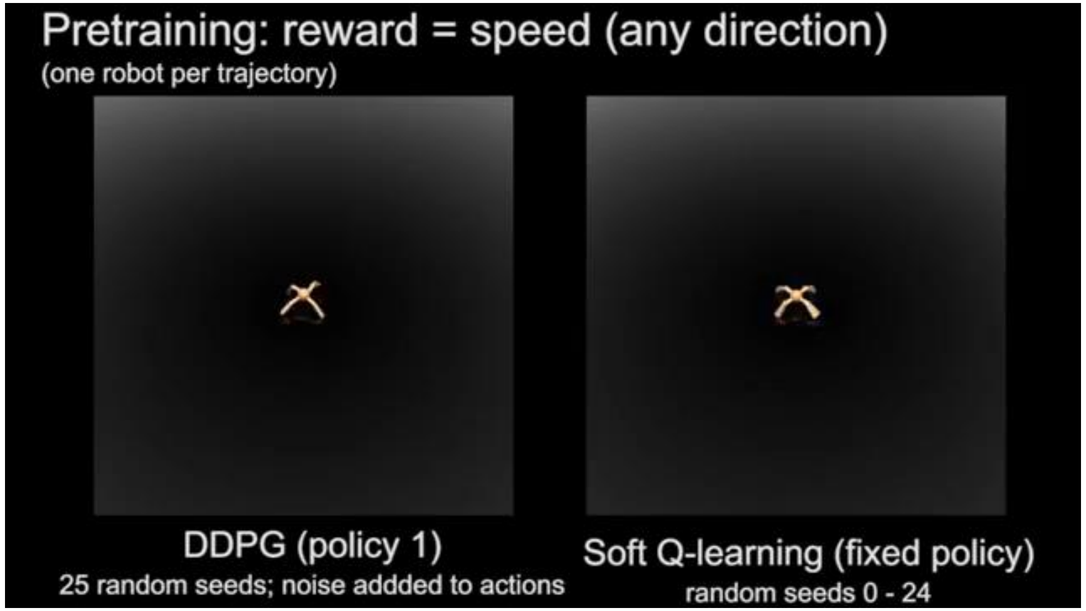
Act **as randomly as possible** while collecting high rewards!

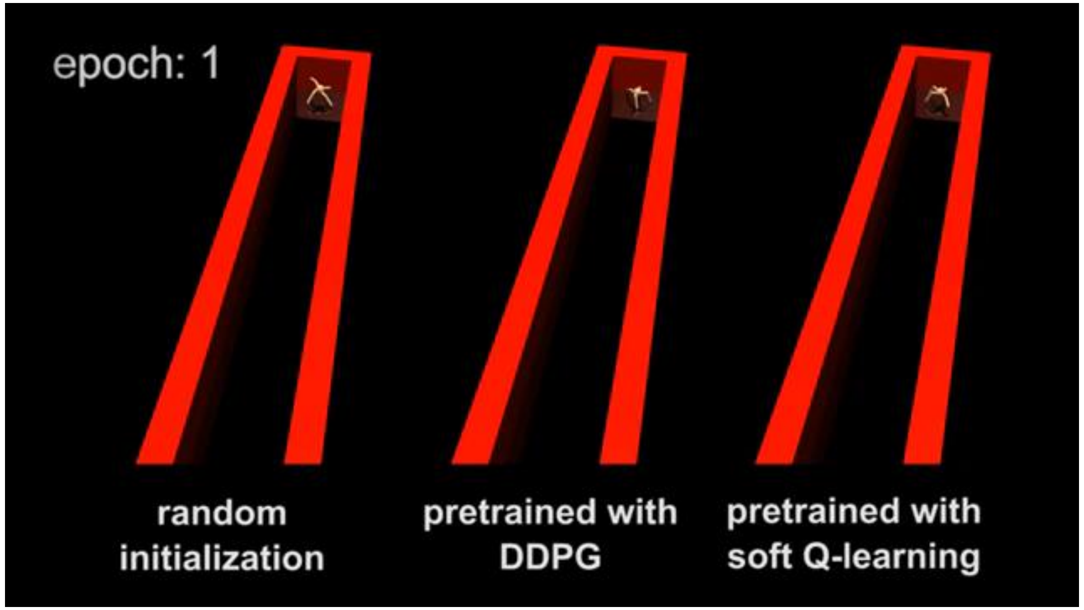
Example: pre-training for robustness

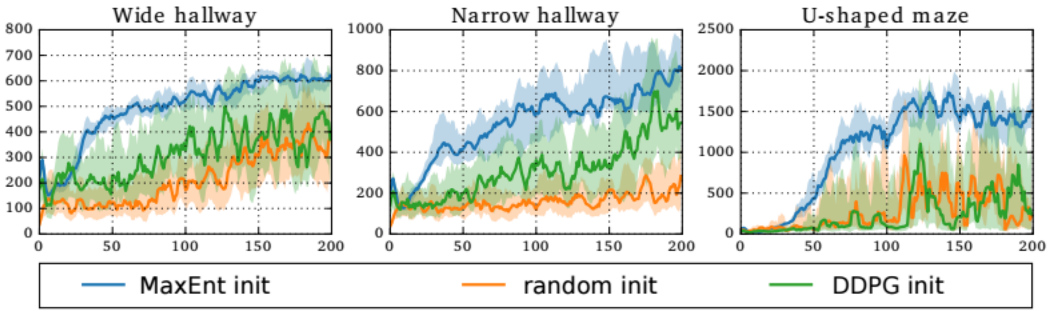


Learning to solve a task **in all possible ways** provides for more robust transfer!

Example: pre-training for diversity



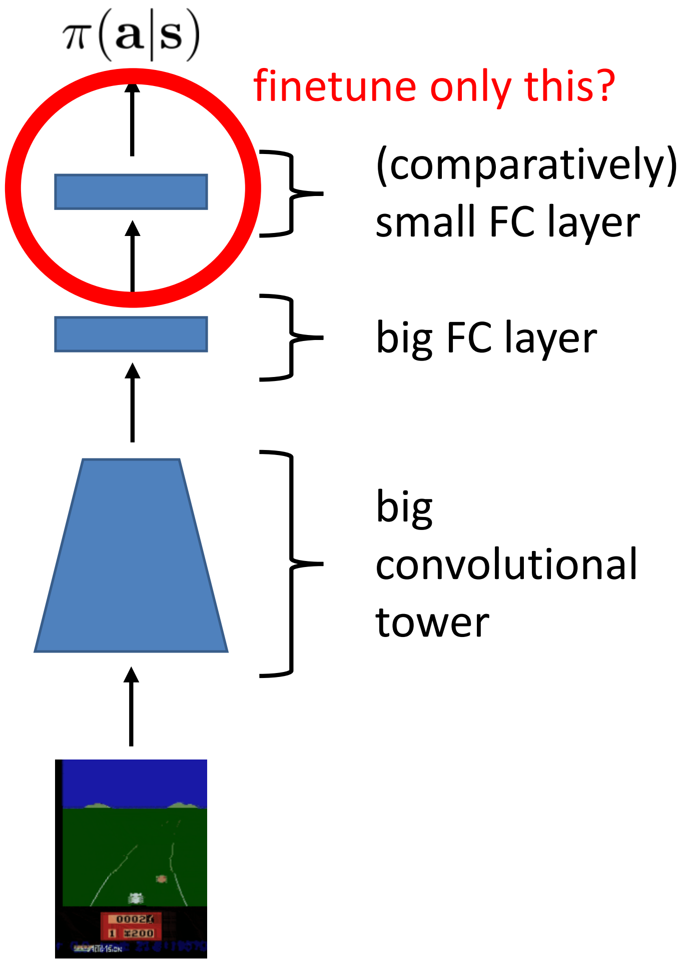




Haarnoja*, Tang*, et al. “Reinforcement Learning with Deep Energy-Based Policies”

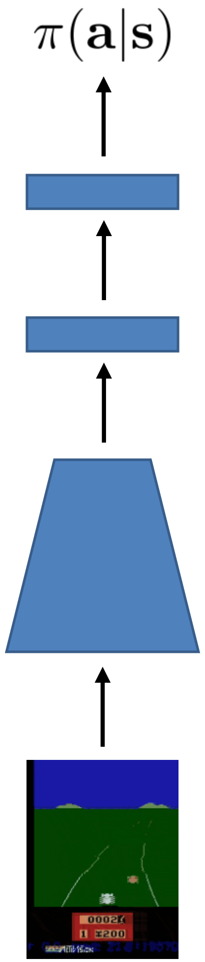
Architectures for transfer: progressive networks

- An issue with finetuning
  - Deep networks work best when they are big
  - When we finetune, we typically want to use a little bit of experience
  - Little bit of experience + big network = overfitting
  - Can we somehow finetune a *small* network, but still pretrain a *big* network?
- Idea 1: finetune just a few layers
  - Limited expressiveness
  - Big error gradients can wipe out initialization



Architectures for transfer: progressive networks

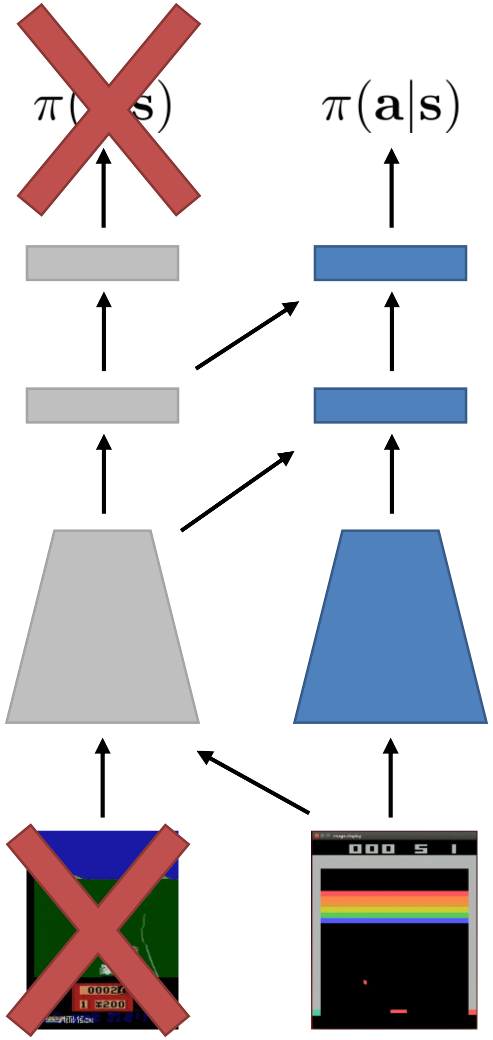
- An issue with finetuning
  - Deep networks work best when they are big
  - When we finetune, we typically want to use a little bit of experience
  - Little bit of experience + big network = overfitting
  - Can we somehow finetune a *small* network, but still pretrain a *big* network?
- Idea 1: finetune just a few layers
  - Limited expressiveness
  - Big error gradients can wipe out initialization
- Idea 2: add *new* layers for the new task
  - Freeze the old layers, so no forgetting



Rusu et al. "Progressive Neural Networks"

Architectures for transfer: progressive networks

- An issue with finetuning
  - Deep networks work best when they are big
  - When we finetune, we typically want to use a little bit of experience
  - Little bit of experience + big network = overfitting
  - Can we somehow finetune a *small* network, but still pretrain a *big* network?
- Idea 1: finetune just a few layers
  - Limited expressiveness
  - Big error gradients can wipe out initialization
- Idea 2: add *new* layers for the new task
  - Freeze the old layers, so no forgetting



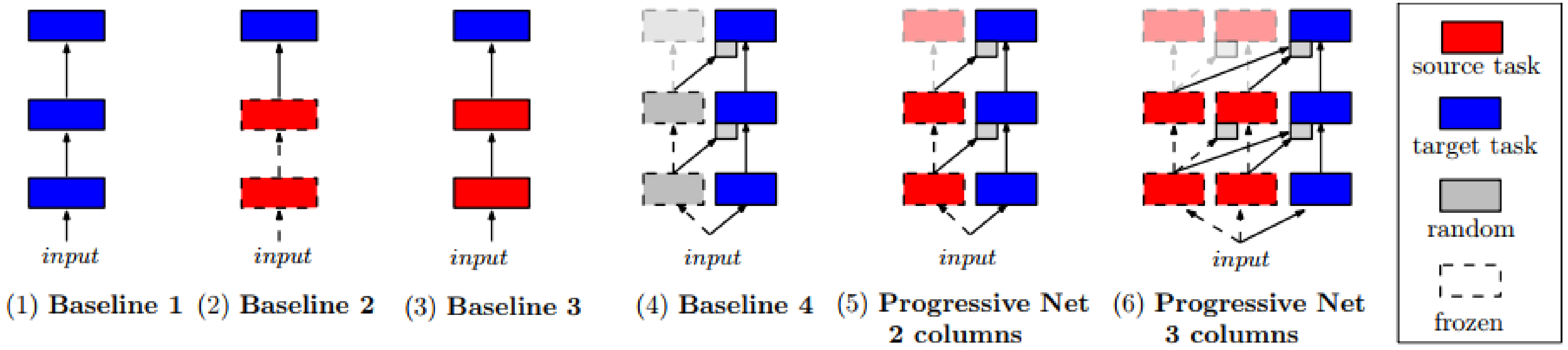
Rusu et al. "Progressive Neural Networks"

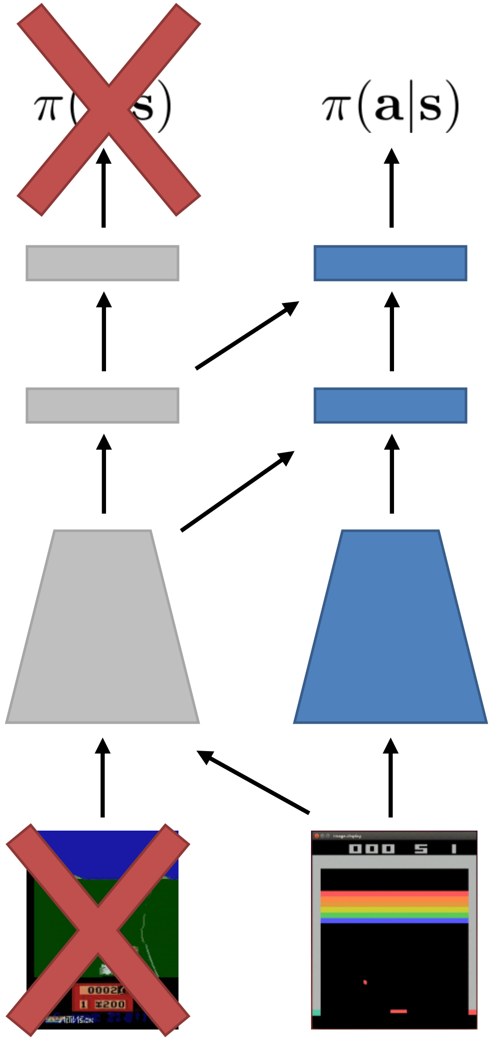
Architectures for transfer: progressive networks

Does it work? sort of…

	Pong Soup		Atari		Labyrinth	
	Mean (%)	Median (%)	Mean (%)	Median (%)	Mean (%)	Median (%)
Baseline 1	100	100	100	100	100	100
Baseline 2	35	7	41	21	88	85
Baseline 3	181	160	133	110	235	112
Baseline 4	134	131	96	95	185	108
Progressive 2 col	209	169	132	112	**491**	**115**
Progressive 3 col	**222**	**183**	140	111	—	—
Progressive 4 col	—	—	**141**	**116**	—	—

Table 1: Transfer percentages in three domains. Baselines are defined in Fig. 3.

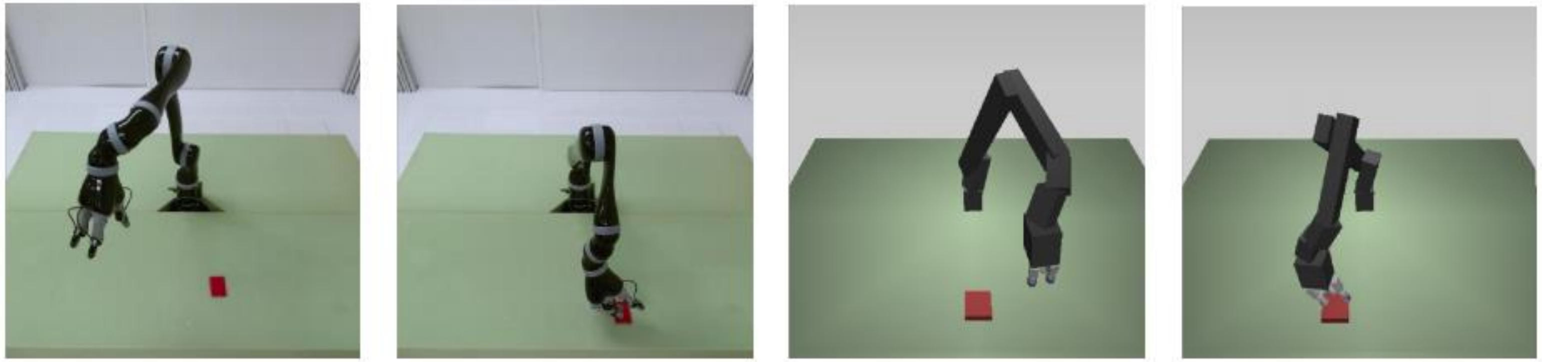


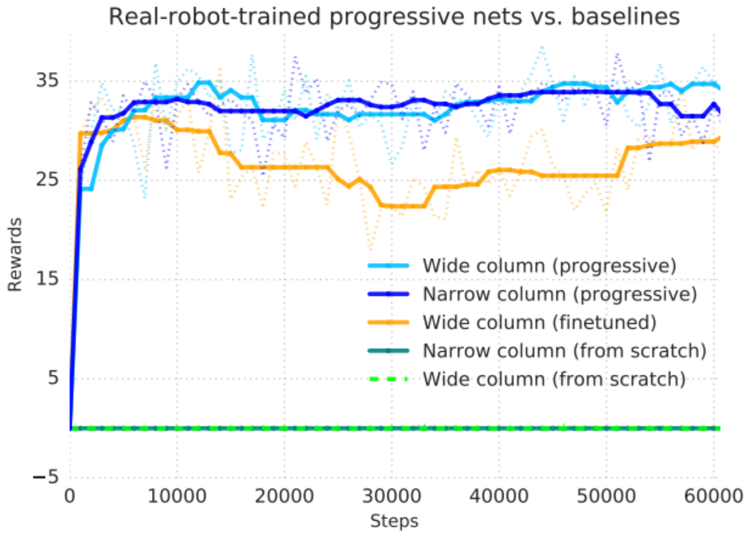


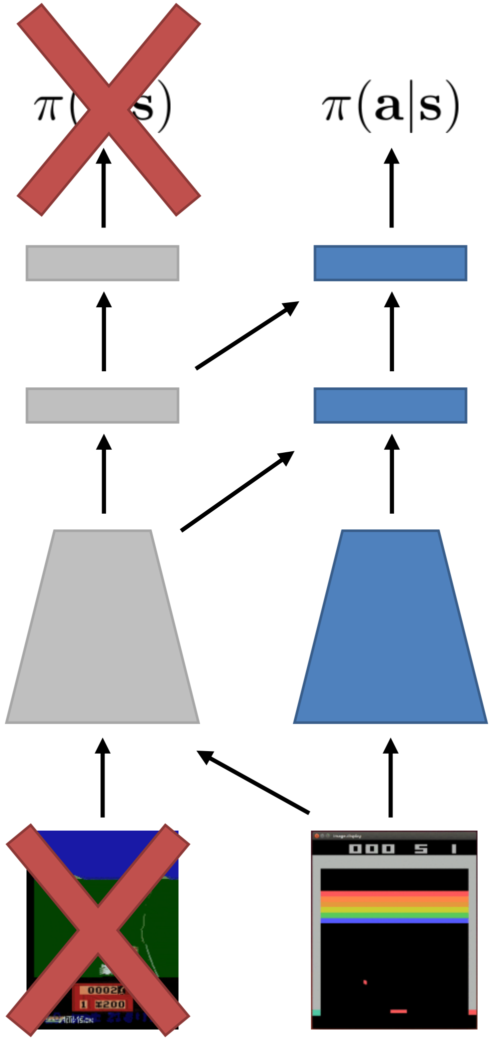
Rusu et al. “Progressive Neural Networks”

Architectures for transfer: progressive networks

Does it work? sort of…







+ alleviates some issues with finetuning

- not obvious how serious these issues are

Rusu et al. “Progressive Neural Networks”

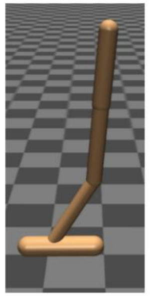
Finetuning summary

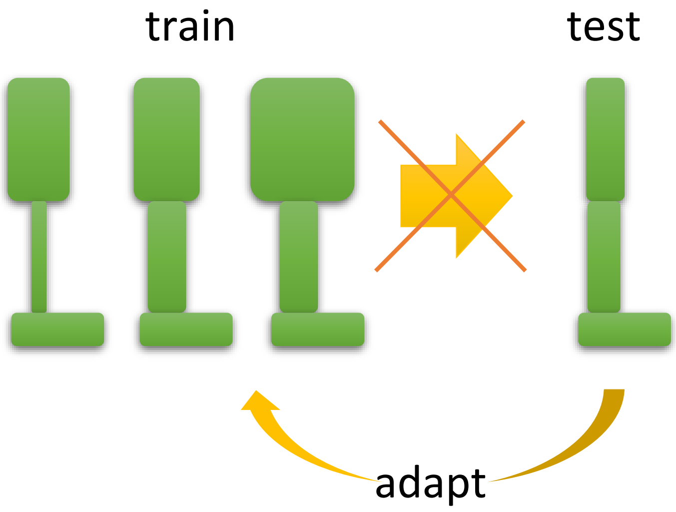
- Try and hope for the best
    - Sometimes there is enough variability during training to generalize
- Finetuning
    - A few issues with finetuning in RL
    - Maximum entropy training can help
- Architectures for finetuning: progressive networks
    - Addresses some overfitting and expressivity problems by construction

What if we can manipulate the source domain?

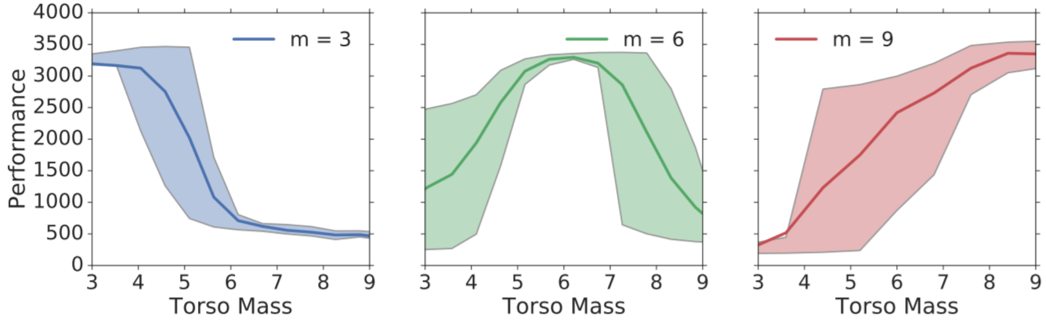
- So far: source domain (e.g., empty room) and target domain (e.g., corridor) are fixed
- What if we can **design** the source domain, and we have a **difficult** target domain?
  - Often the case for simulation to real world transfer
- Same idea: the more diversity we see at training time, the better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

EPOpt: randomizing physical parameters

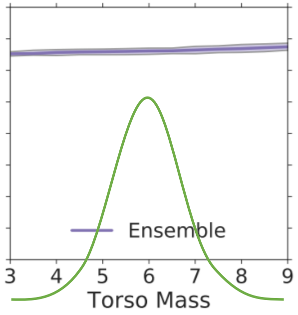




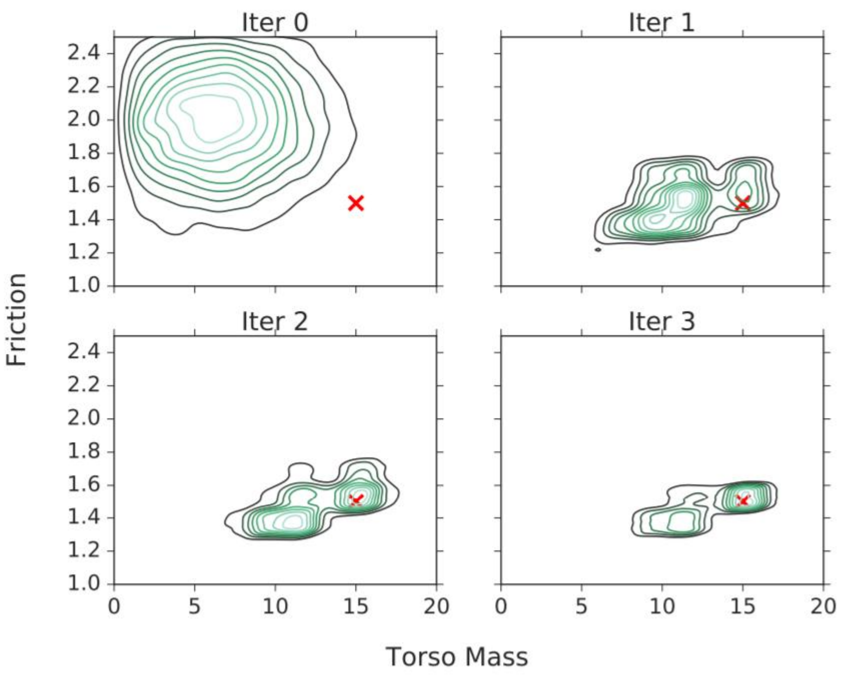
training on single torso mass



training on model ensemble

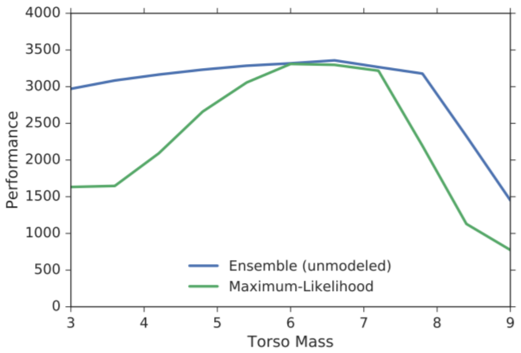


ensemble adaptation



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



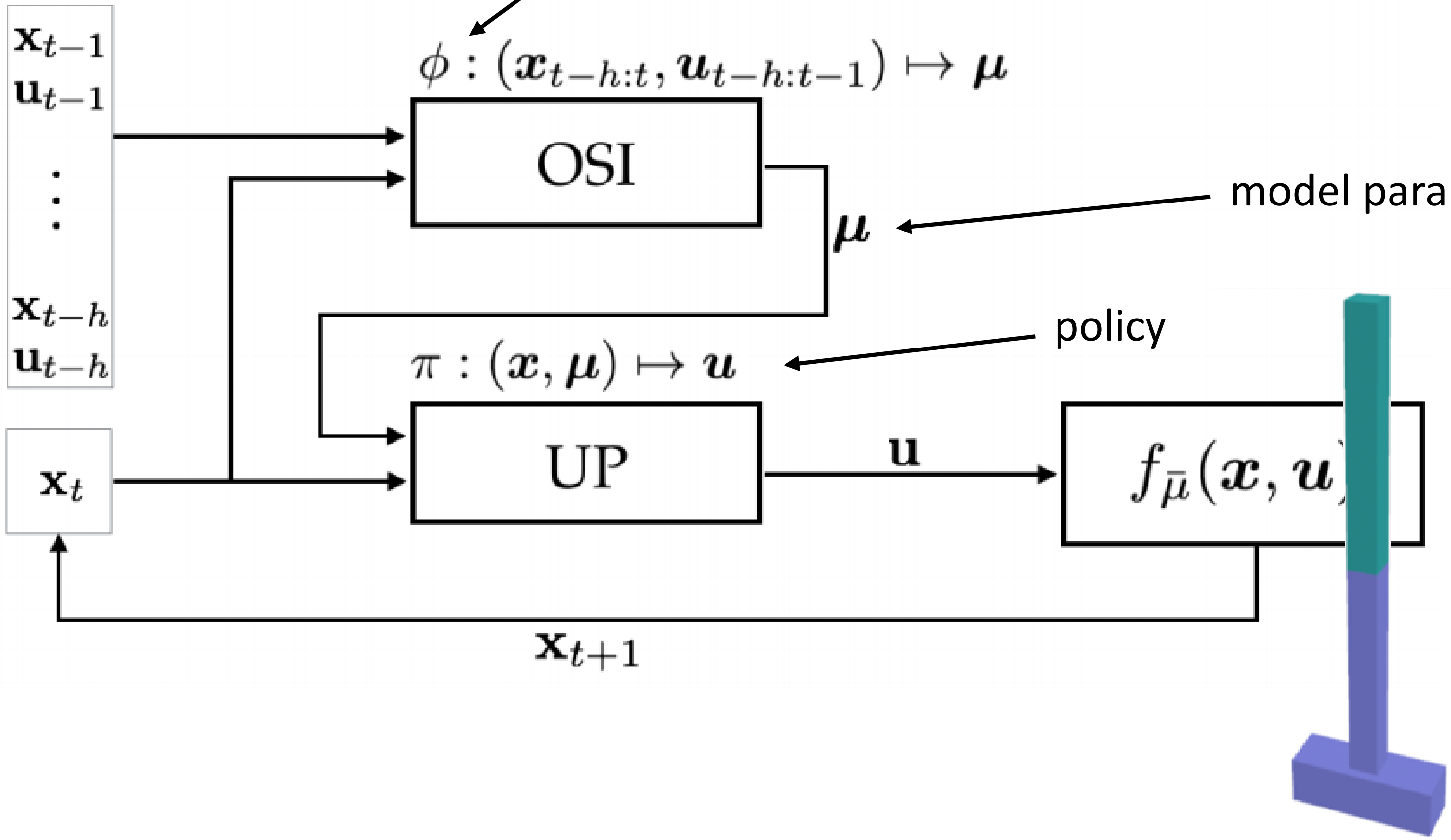
Rajeswaran et al., “EPOpt: Learning robust neural network policies...”

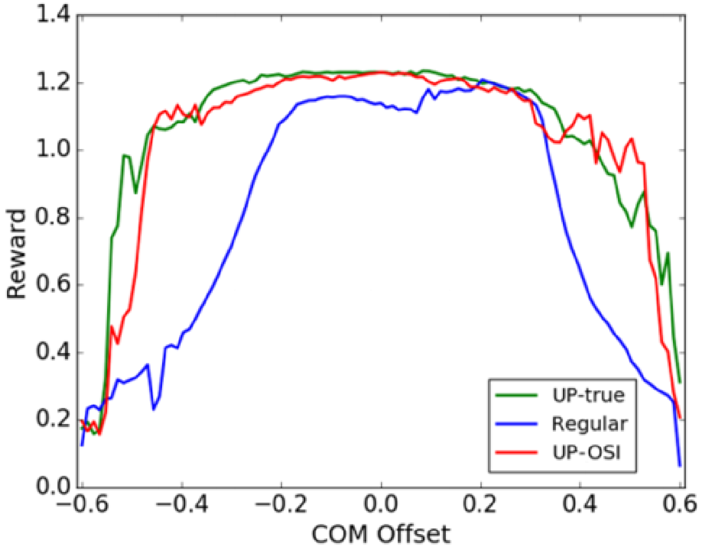
Preparing for the unknown: explicit system ID

system identification RNN

model parameters (e.g., mass)

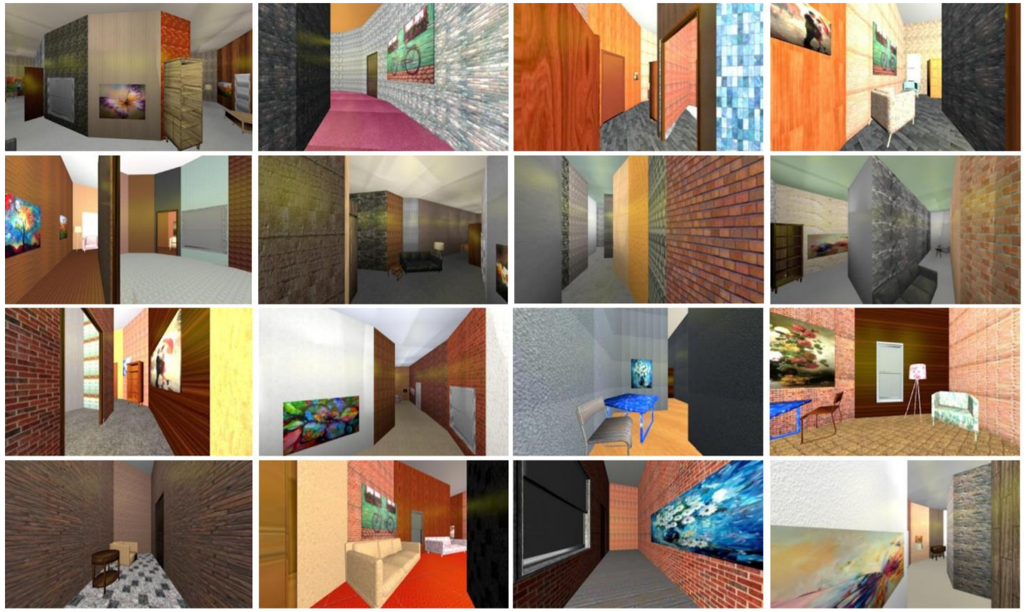
policy





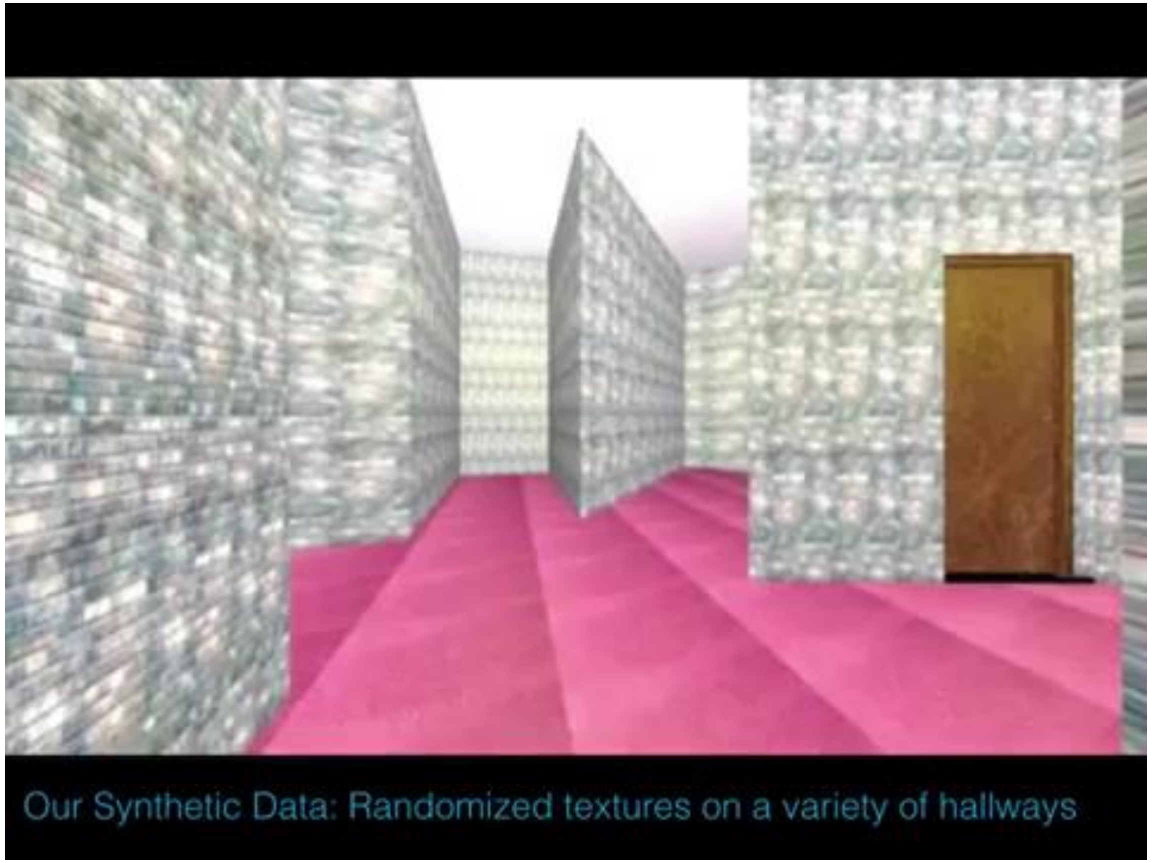
Yu et al., "Preparing for the Unknown: Learning a Universal Policy with Online System Identification"

CAD2RL: randomization for real-world control



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

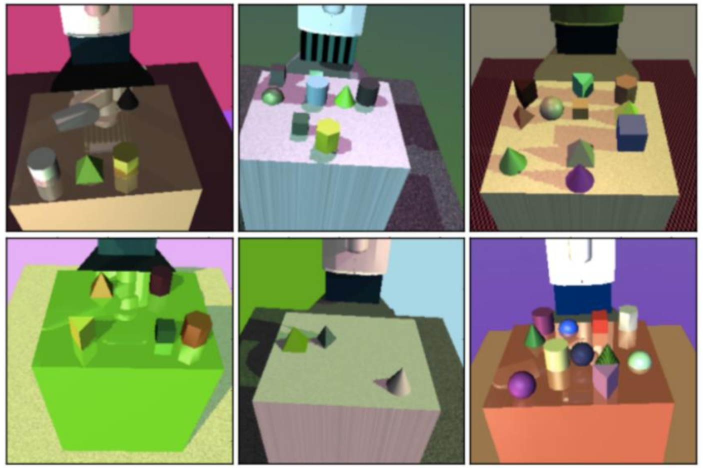
CAD2RL: randomization for real-world control



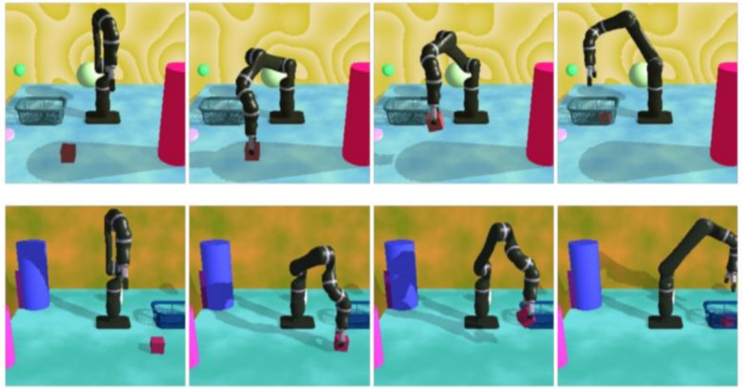
Sadeghi et al., “CAD2RL: Real Single-Image Flight without a Single Real Image”

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

Randomization for manipulation



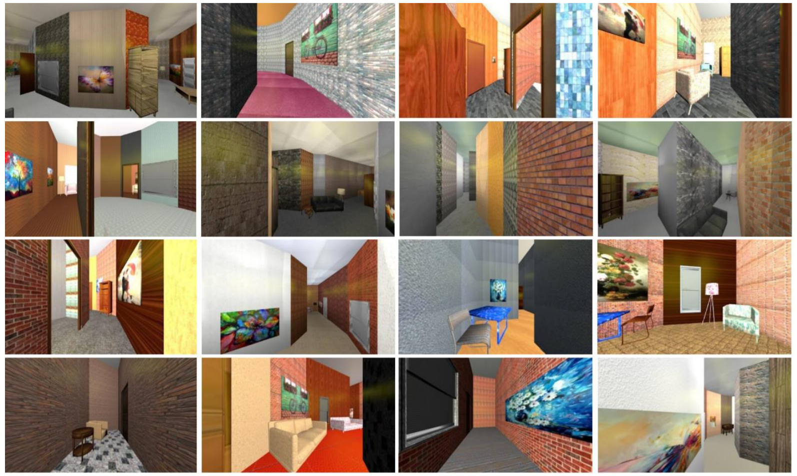
Tobin, Fong, Ray, Schneider, Zaremba, Abbeel



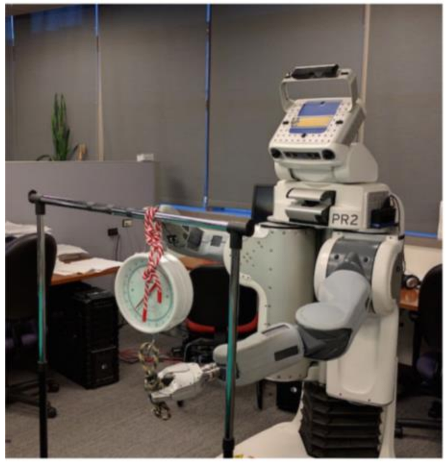
James, Davison, Johns

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?

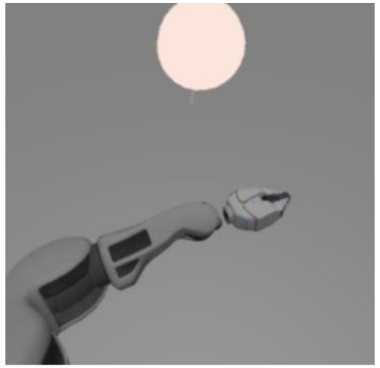


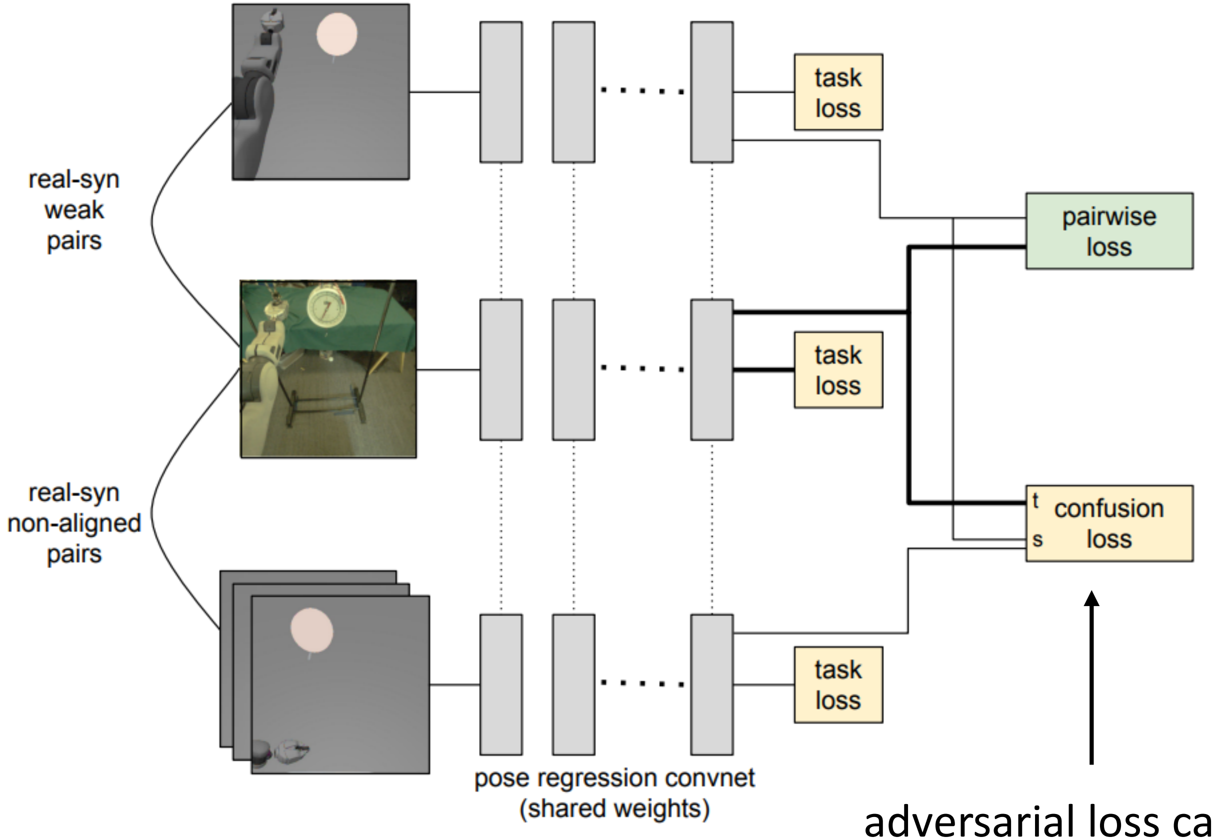
Better transfer through domain adaptation



simulated images

real images



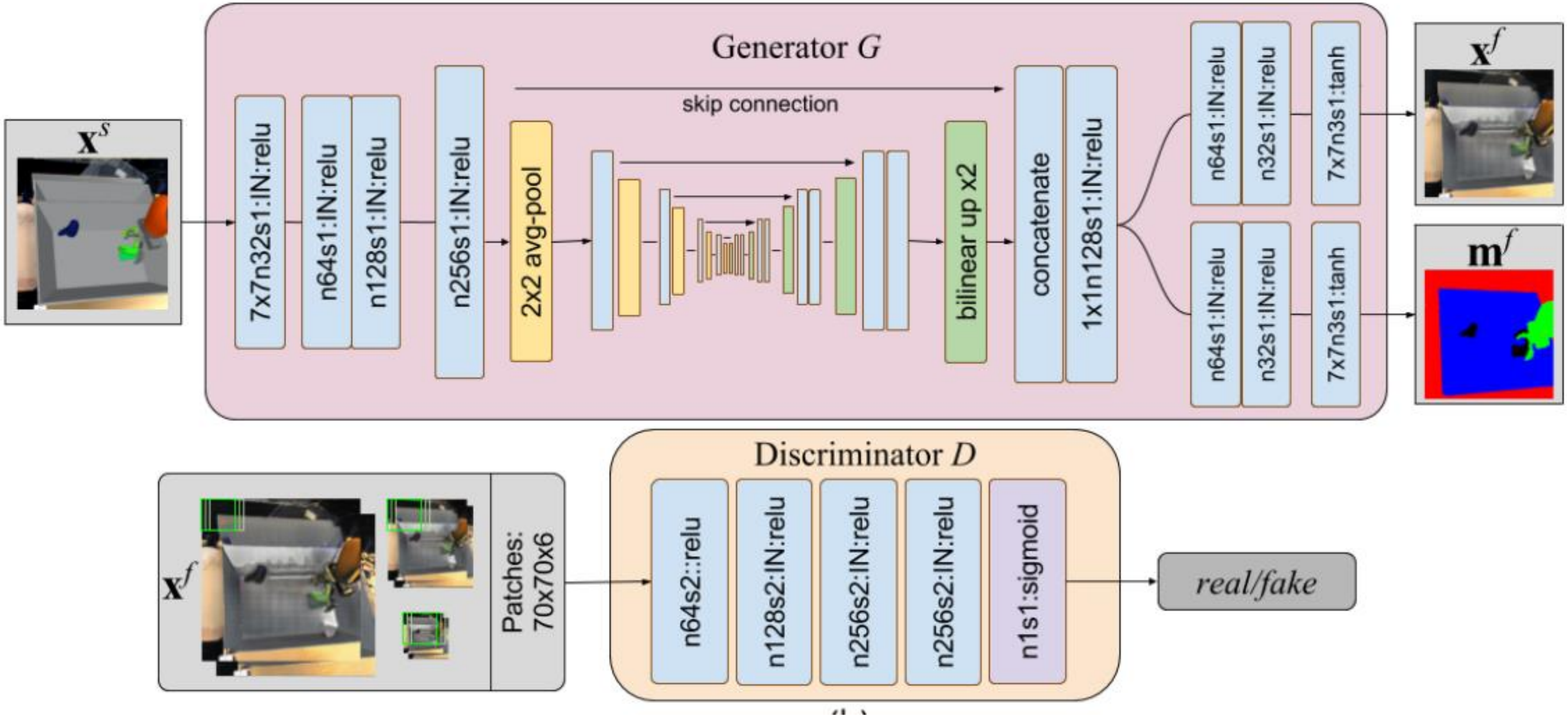


adversarial loss causes internal CNN features to be indistinguishable for sim and real

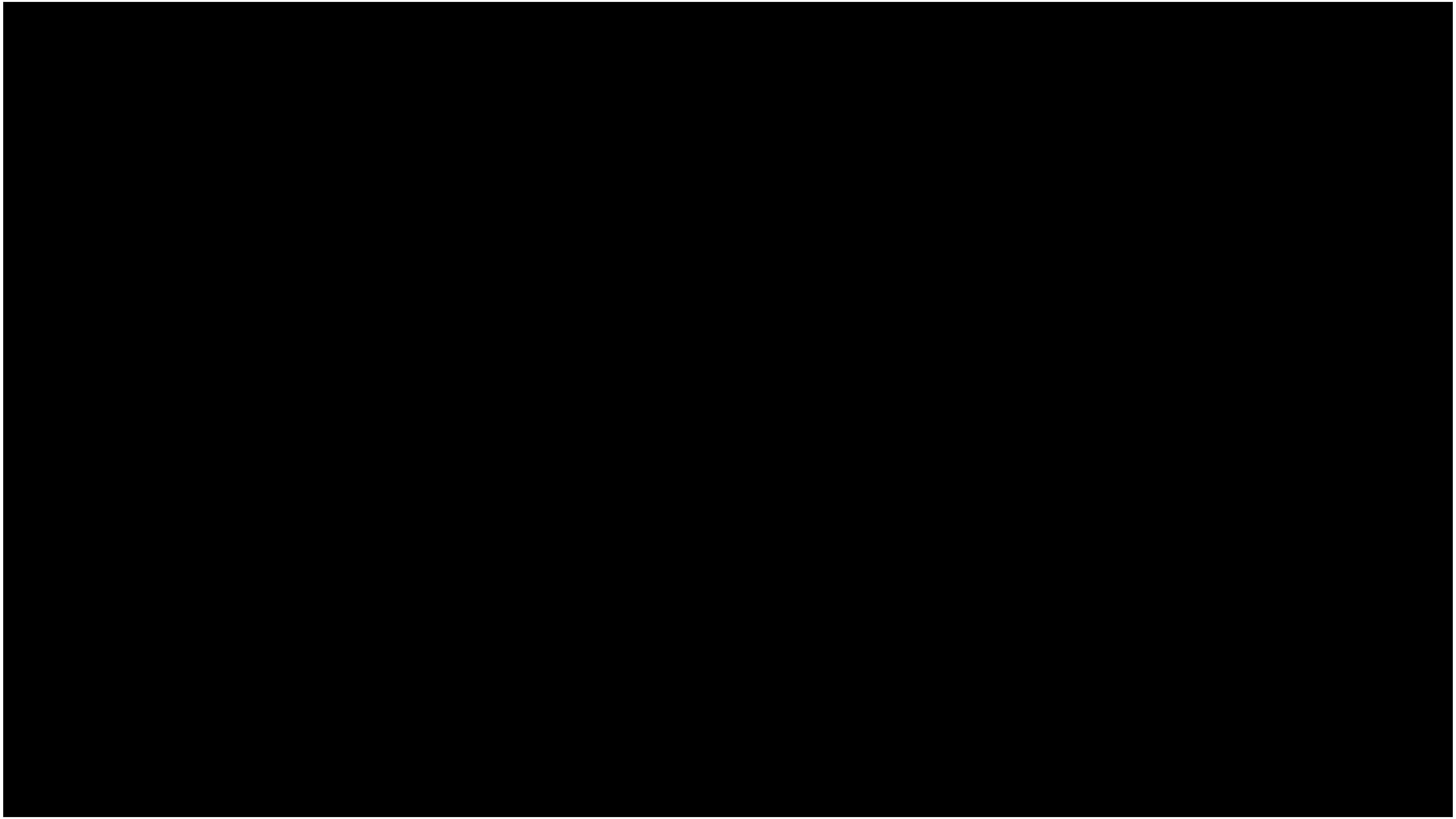
Tzeng*, Devin*, et al., “Adapting Visuomotor Representations with Weak Pairwise Constraints”

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!

Forward transfer suggested readings

Haarnoja*, Tang*, Abbeel, Levine. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**

Rusu et al. (2016). **Progress Neural Networks.**

Rajeswaran, Ghotra, Levine, Ravindran. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.**

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

Tobin, Fong, Ray, Schneider, Zaremba, Abbeel. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.**

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.**

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

more on this next time!

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