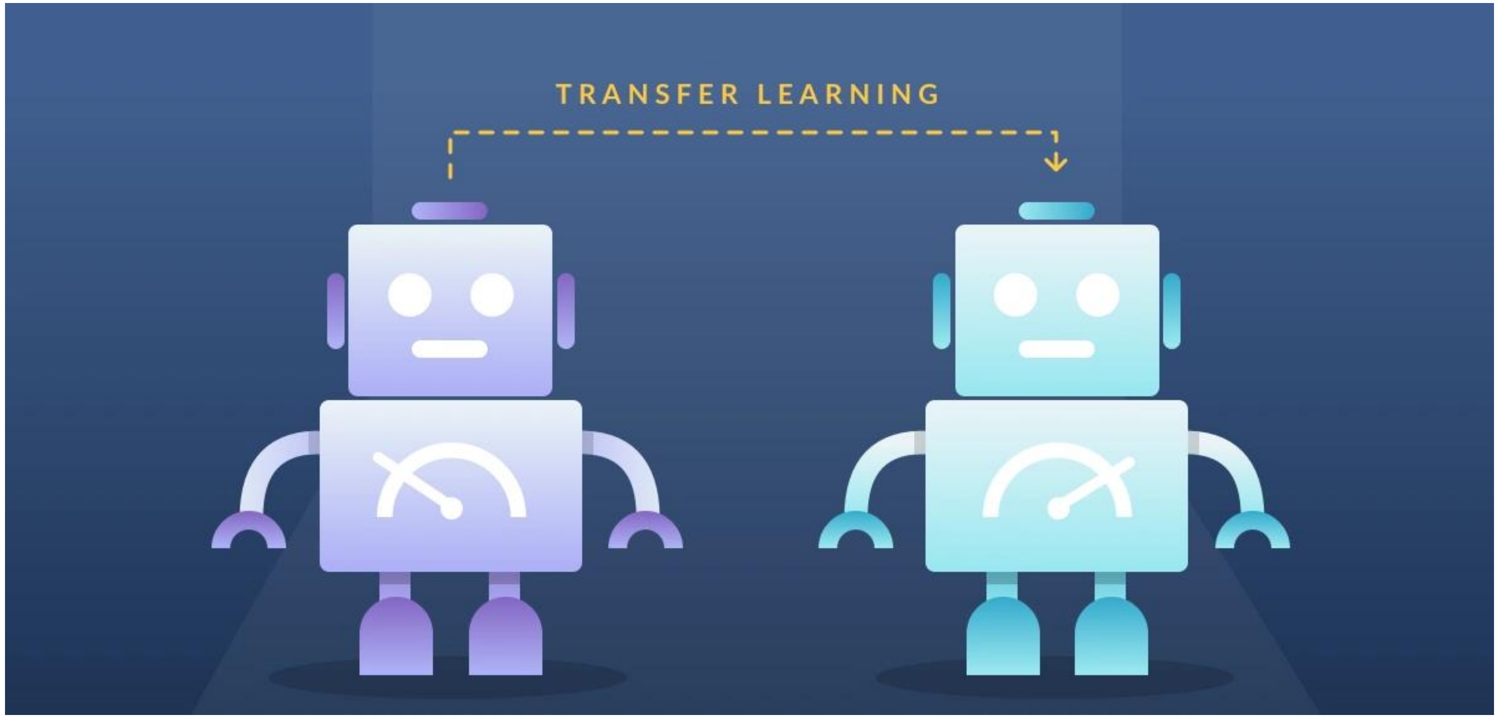
CSCE-642 Reinforcement Learning
Transfer Learning



Instructor: Guni Sharon

Based on slides by Sergey Levine

Transferring knowledge between models

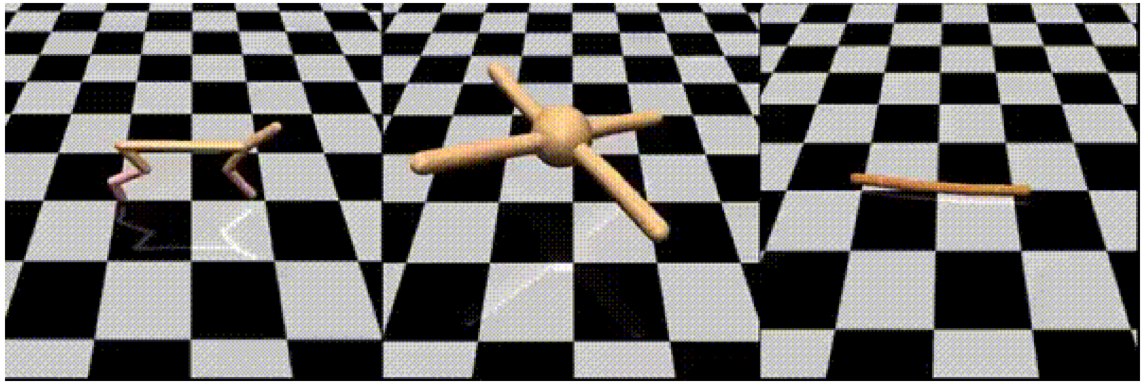
- "Forward" transfer: train on one task, transfer to a new task
  - Just try it and hope for the best
  - Finetune on the new task
  - Architectures for transfer: progressive networks
  - Randomize source task domain
- Multi-task transfer: train on many tasks, transfer to a new task
  - Model-based reinforcement learning
  - Model distillation
  - Modular policy networks
- Multi-task meta-learning: learn to learn from many tasks
  - RNN-based meta-learning
  - Gradient-based meta-learning

Finetuning

- **Task**: train a conv-net to detect dogs using 10,000 images
- **Solution**: train the network from scratch
- Conv-nets have a huge number of parameters (millions)
  - Training on a small dataset (<< #parameters) often result in overfitting
- Fine-tune an existing network that is trained on a large dataset like the ImageNet (1.2M labeled images)
  - Continue training it (i.e., back-propagation) on the smaller dataset
- The pre-trained model will already have learned features that are relevant* to our own classification problem

Challenges with finetuning in RL

- RL tasks are domain specific
  - Features are less general
  - Policies & value functions become overly specialized
- Learned policies tend to be of low entropy
  - Loss of exploration at convergence
  - Low-entropy policies adapt very slowly to new settings



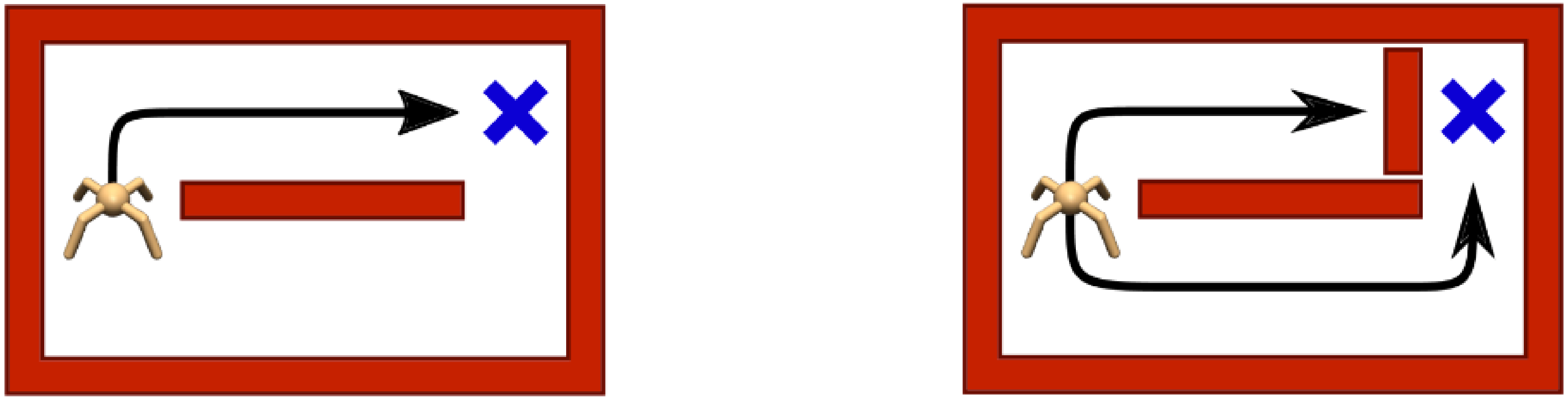
Remedy 1: encourge high entropy

- Act as randomly as possible
- High entropy policies -> More exploration -> increased adaptability to new tasks
- $\pi^* = \operatorname{argmax}_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}\left(\pi(\cdot\,|s_t)\right)\right]$

Policy entropy

- Find the optimal high entropy policy using a MaxEnt algorithms e.g., SAC (slide deck 16)

Example: pre-training for robustness



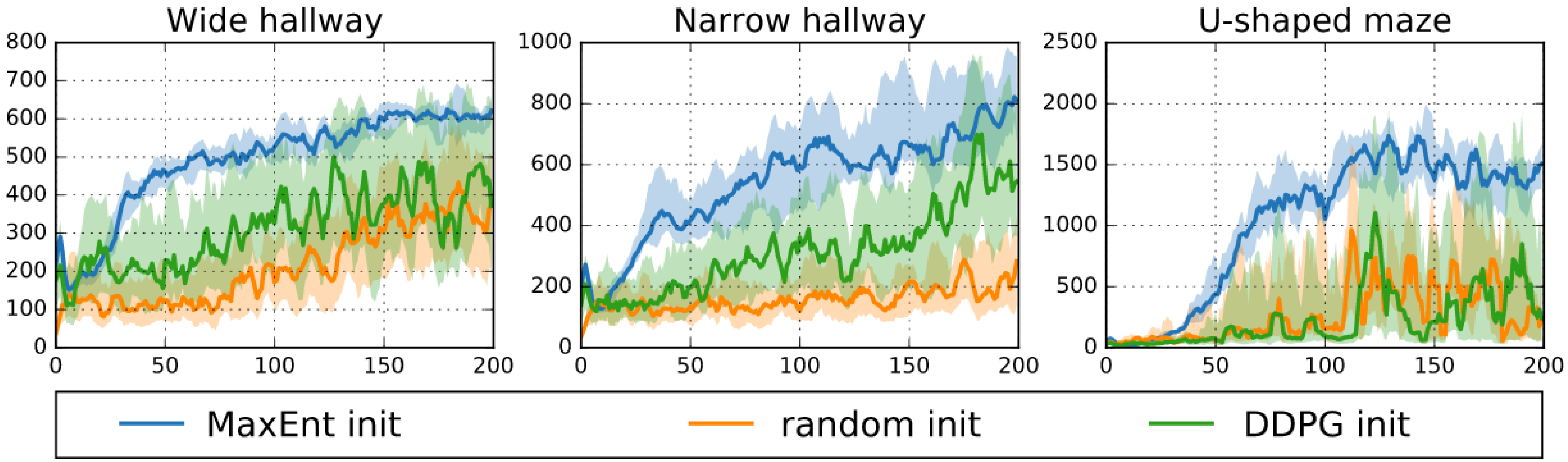
- Learning to solve a task in all possible ways provides for more robust transfer!

Soft Q-learning

Fine-tuning a pretrained policy
in a new environment

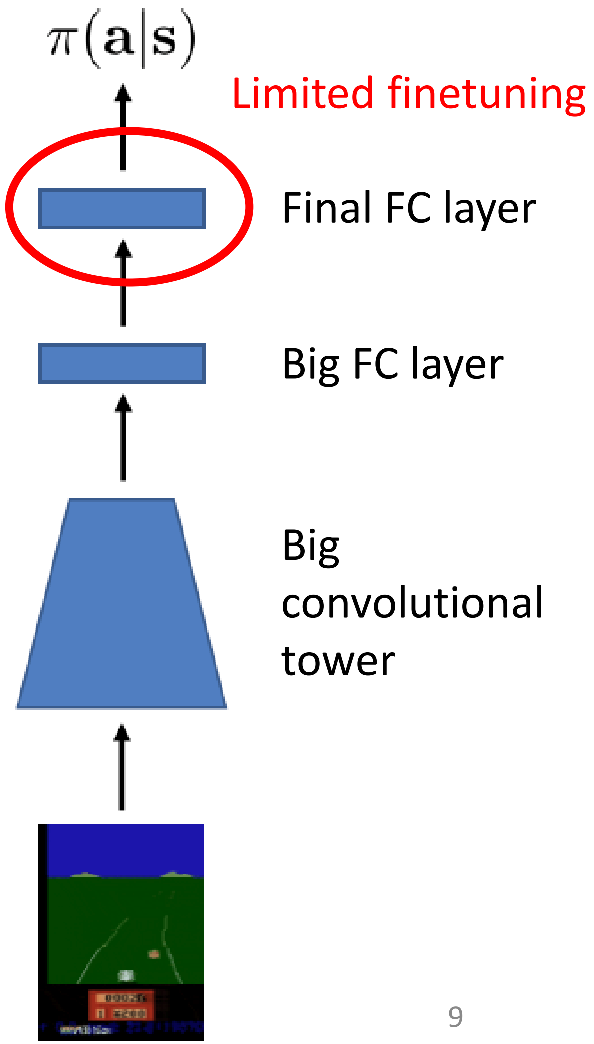
Haarnoja et al. “Reinforcement Learning with Deep Energy-Based Policies”

- Performance in the downstream task with fine-tuning over (MaxEnt) or (DDPG). The x-axis shows the training iterations. The y-axis shows the average discounted return. Solid lines are average values over 10 random seeds. Shaded regions correspond to one standard deviation.



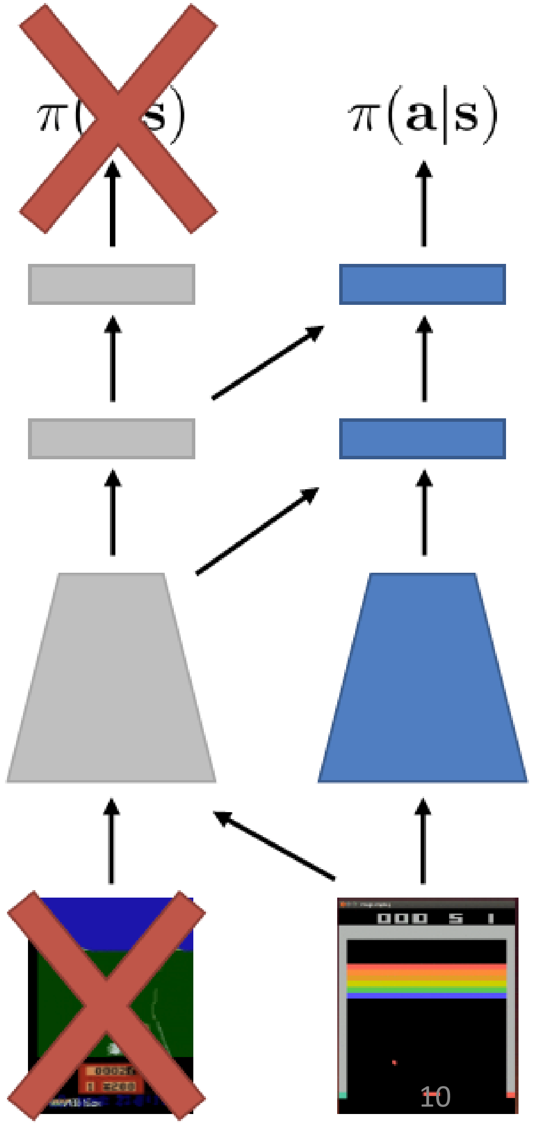
Architectures for transfer: progressive networks

- 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
  - Avoid wiping 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
  - Avoid wiping out initialization
- Idea 2: add *new* layers for the new task
  - Freeze the old layers, so no forgetting

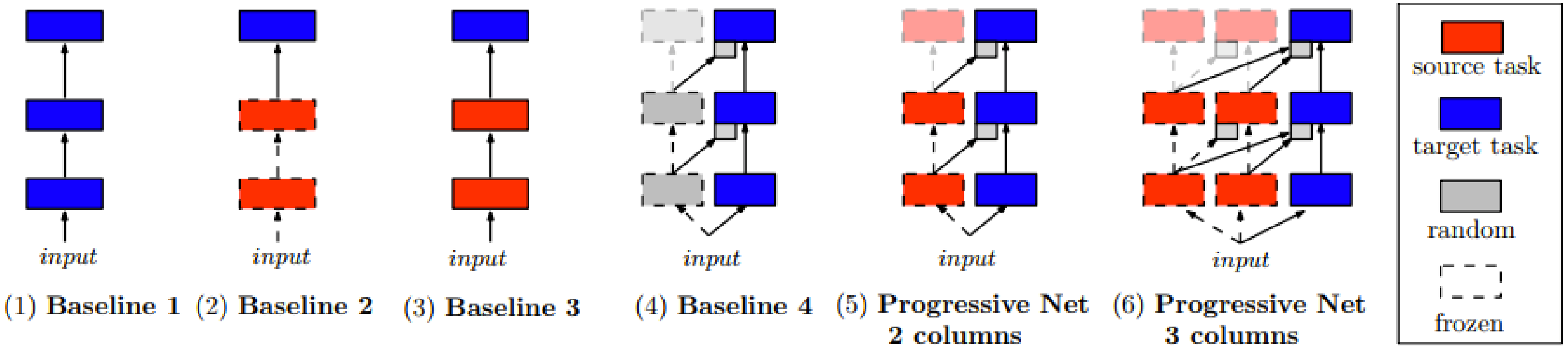


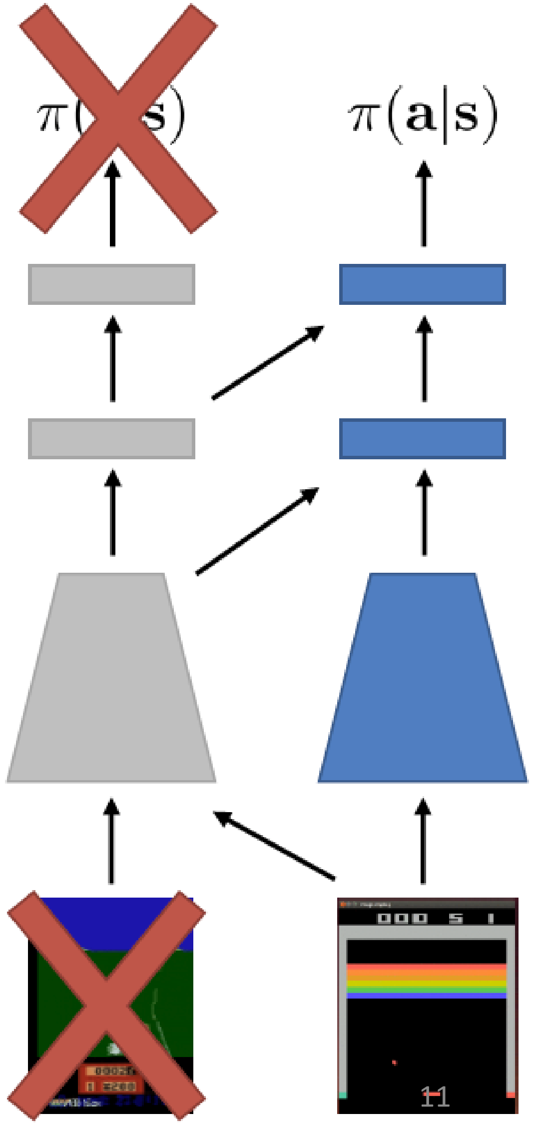
[Rusu et al. “Progressive Neural Networks”]

Architectures for transfer: progressive networks

	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”]

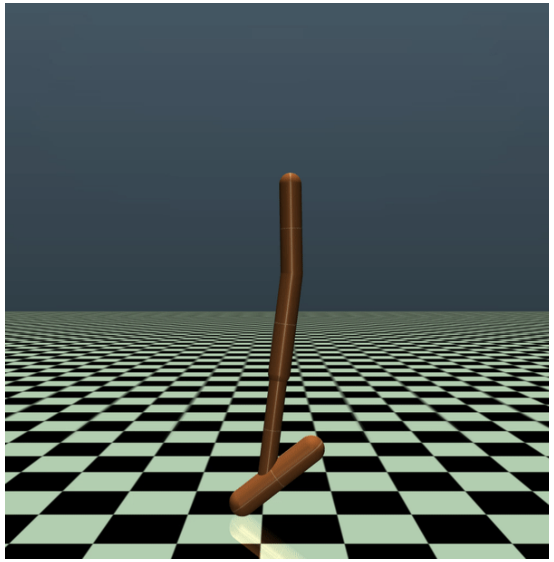
Finetuning summary

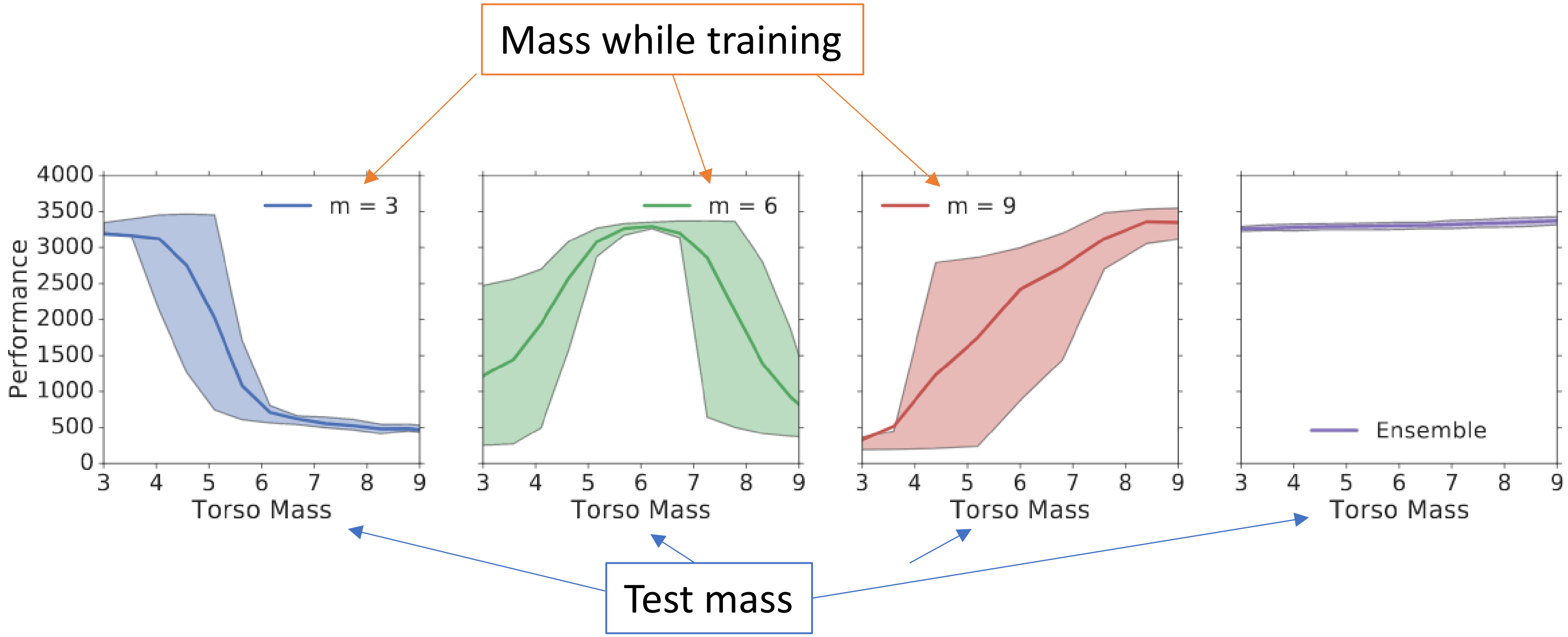
- Try and hope for the best
  - Sometimes there is enough variability during training to generalize
- Finetuning
  - Task overfitting issues with finetuning in RL
  - Maximum entropy training can help
- Architectures for finetuning: progressive networks
  - Addresses some overfitting and expressivity 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?
  - Often the case for simulation to real world transfer
- Same idea: the more diversity we see at training time, the better we will transfer!

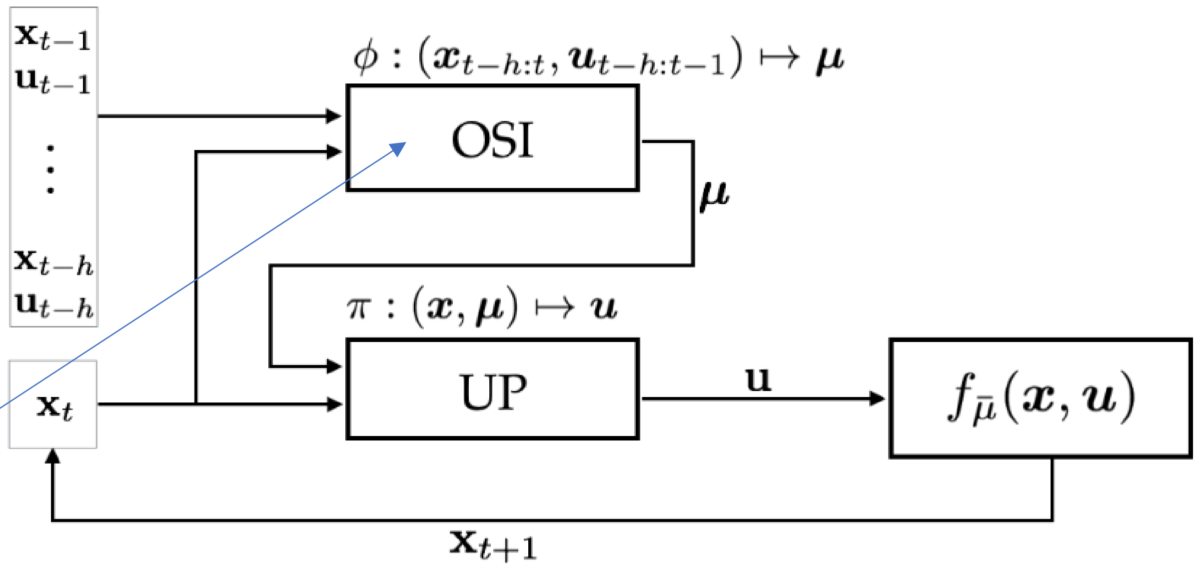
EPOpt: randomizing physical parameters



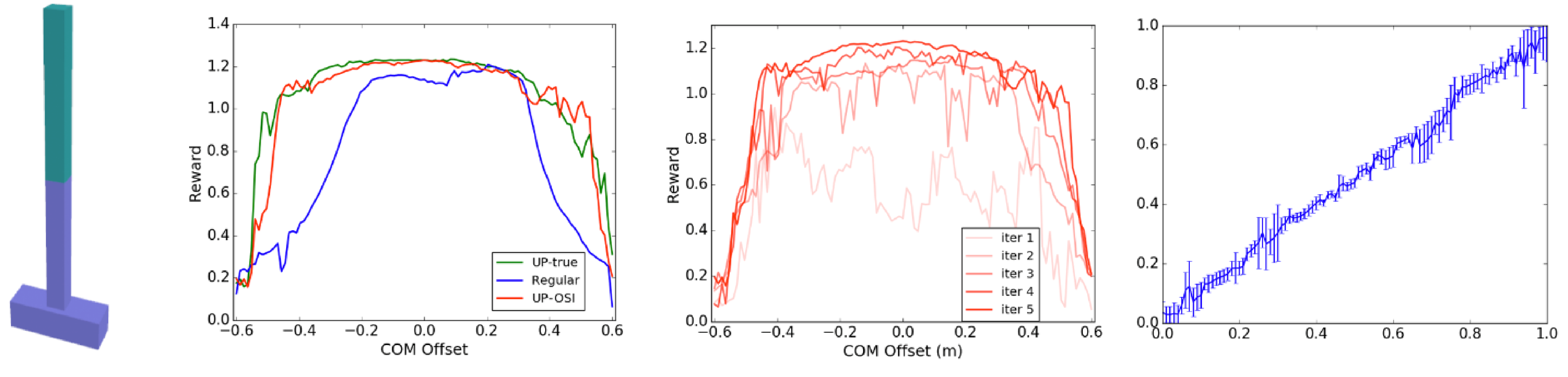


Rajeswaran et al. “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles”

Preparing for the unknown: explicit system ID



Learn the model's physics and send it as input to the controller



(a) (b) (c) (d)

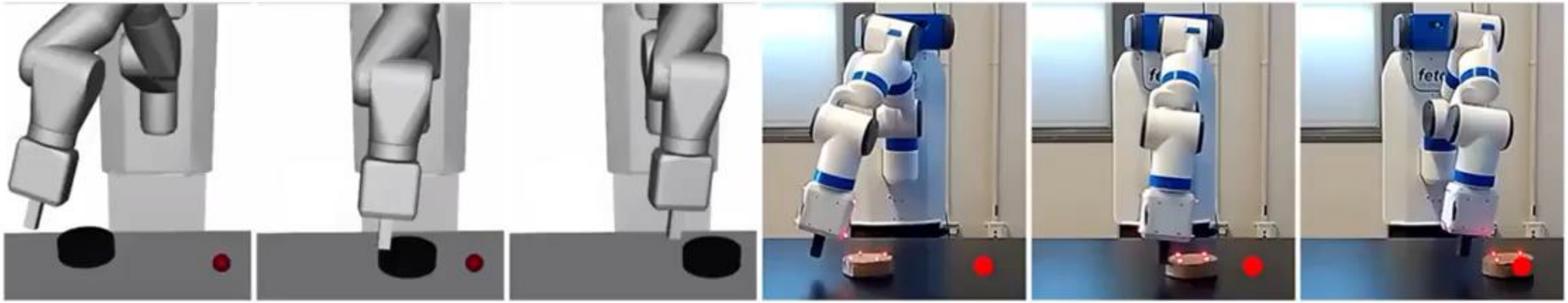
Yu et al., "Preparing for the Unknown: Learning a Universal Policy with Online System Identification"

Sim-to-Real Transfer of Robotic Control with Dynamics Randomization

Xue Bin Peng[1,2], Marcin Andrychowicz[2], Wojciech Zaremba[2], Pieter Abbeel[1,2]

[1]Electrical Engineering and Computer Sciences, UC Berkeley, USA

[2]OpenAI, USA



 Wed AM Pod T.8

$(CAD)^2RL$:

Real Single-Image Flight without a Single Real Image

Fereshteh Sadeghi

University of Washington

Sergey Levine

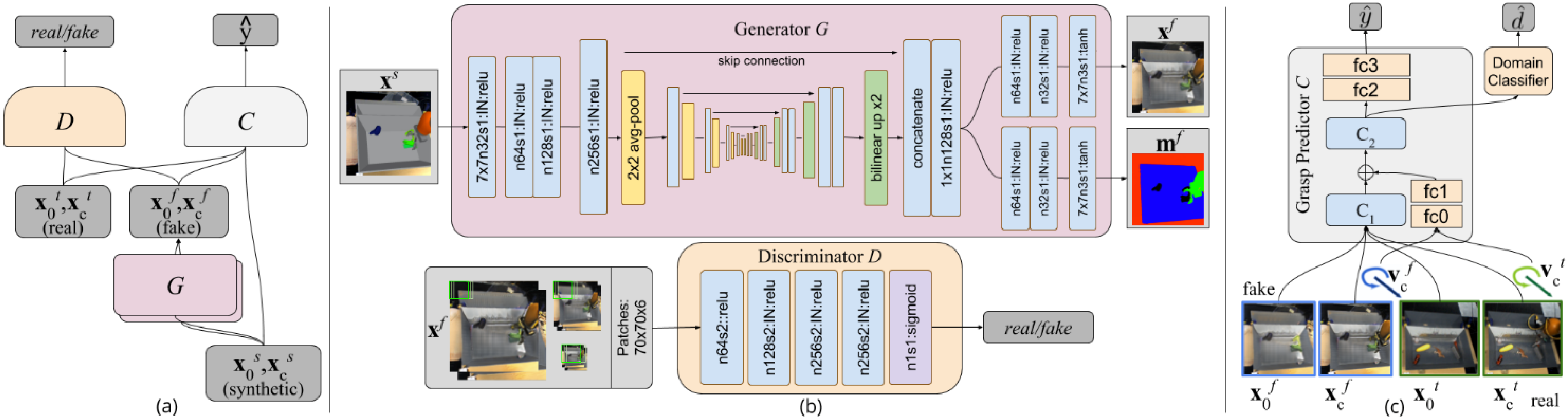
University of California, Berkeley

What if we can peek at the target domain?

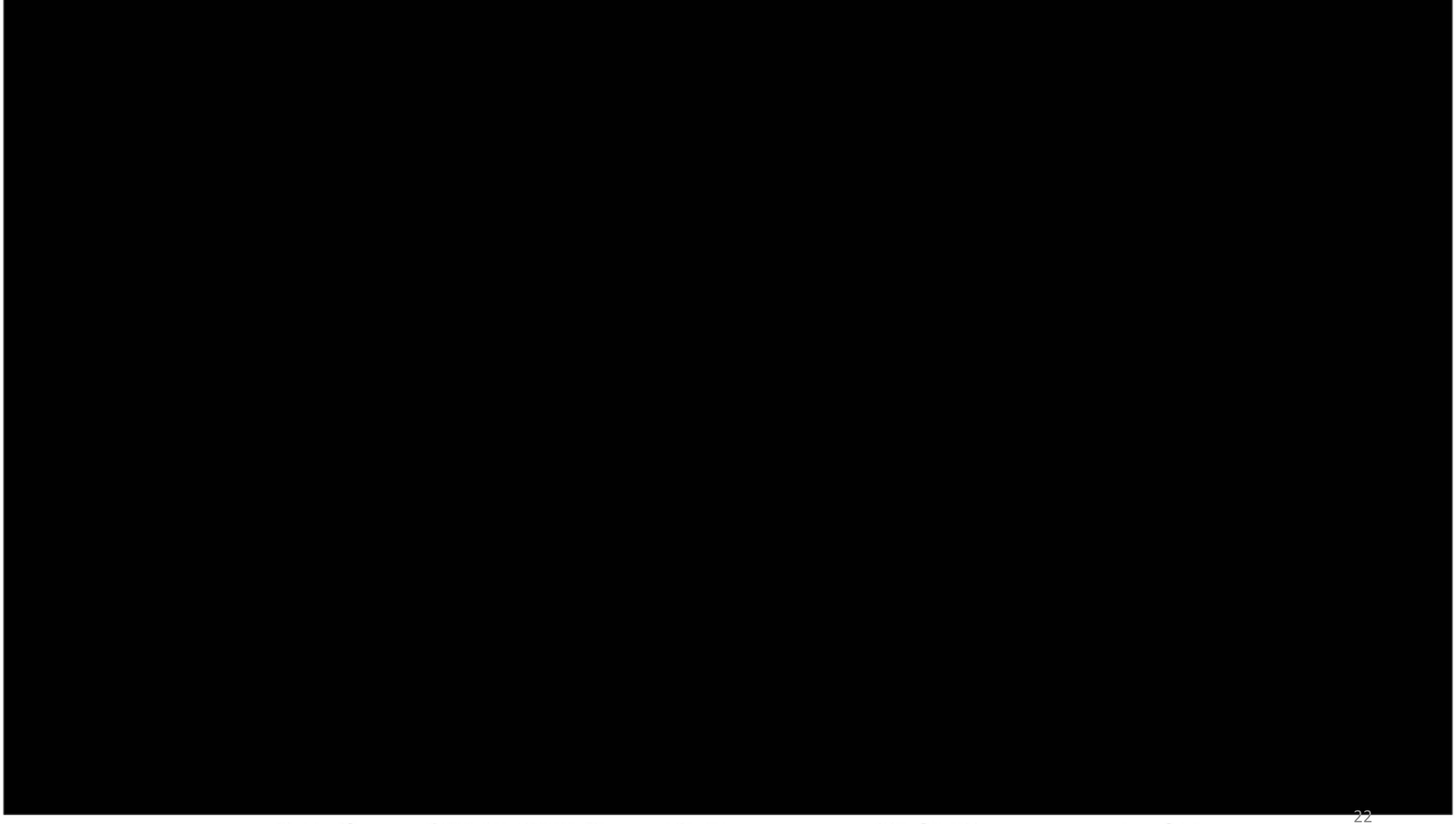
- So far: pure 0-shot transfer: learn in source domain so that we can succeed in **unknown** target domain
- If we know nothing about the target domain, the best we can do is be as robust as possible
- What if we saw a few images of the target domain?

Domain adaptation at the pixel level

Use a GAN to train synthetic images to resemble *realistic* ones



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 inefficient
  - Maximum entropy formulation can help
  - Progressive nets 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
  - Modify the source domain observations to look like target domain

Forward transfer suggested readings

- Haarnoja*, Tang*, et al. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**
- Rusuet al. (2016). **Progress Neural Networks.**
- Rajeswaran, et al. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.**
- 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.**
- Tzeng*, Devin*, et al. (2016). **Adapting Deep Visuomotor Representations with Weak Pairwise Constraints.**
- Bousmaliset al. (2017). **Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping.**

Transferring knowledge between models

- "Forward" transfer: train on one task, transfer to a new task
  - Just try it and hope for the best
  - Finetune on the new task
  - Architectures for transfer: progressive networks
  - Randomize source task domain
- Multi-task transfer: train on many tasks, transfer to a new task
  - Model-based reinforcement learning
  - Model distillation
  - Modular policy networks
- Multi-task meta-learning: learn to learn from many tasks
  - RNN-based meta-learning
  - Gradient-based meta-learning

Multiple source domains

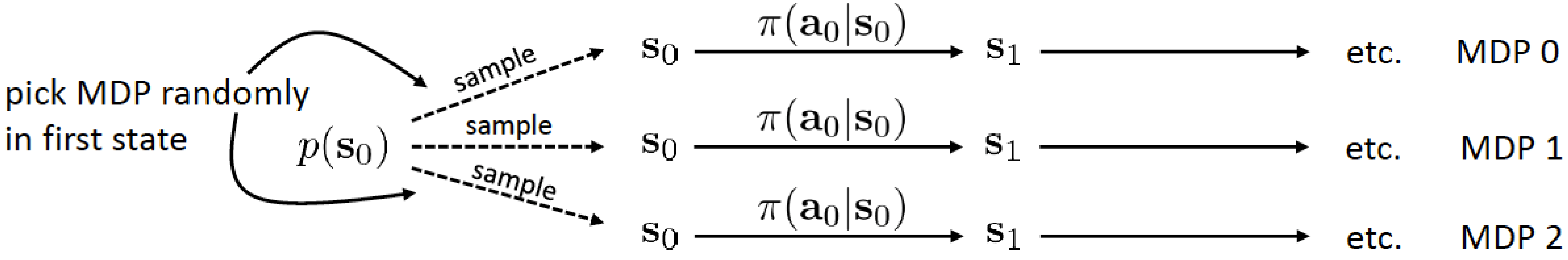
- So far: more diversity -> better transfer
- Need to design this diversity
  - E.g., simulation to real world transfer: randomize the simulation
- What if we transfer from multiple *different* tasks?
  - In a sense, closer to what people do: build on a lifetime of experience
  - Substantially harder: 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
  - Fu et al. “One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors”

Another approach: Solve multiple tasks with the same policy

- Sometimes learning a model is very hard
- Can we learn a multi-task policy that can *simultaneously* perform many tasks?
- Idea 1: construct a joint MDP



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

Background: Ensembles & Distillation

- **Ensemble models:** single models are often not robust, instead train many models and average their predictions
  - This is how most ML competitions (e.g., Kaggle) are won
  - This is very expensive at test time
- **Can we make a single model that is as good as an ensemble?**
- **Distillation:** train on the ensemble's predictions as "soft" targets

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

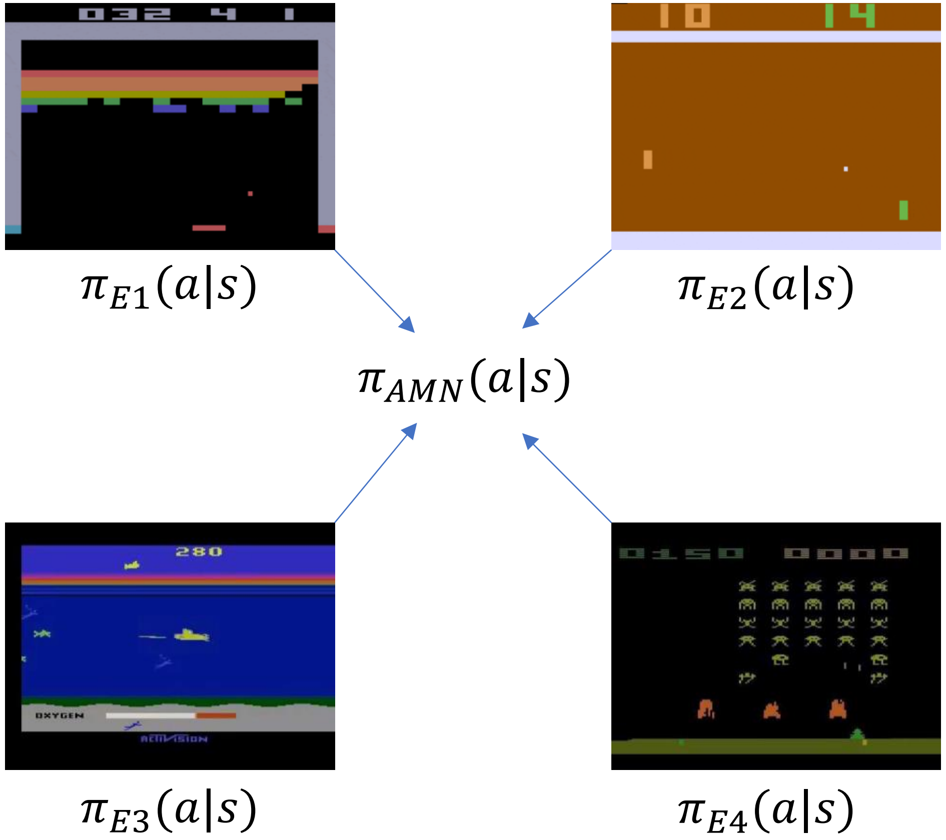
Logit

Temperature

- **Intuition:** more knowledge in soft targets than hard labels!

Hinton et al. "Distilling the Knowledge in a Neural Network"

Distillation for Multi-Task Transfer



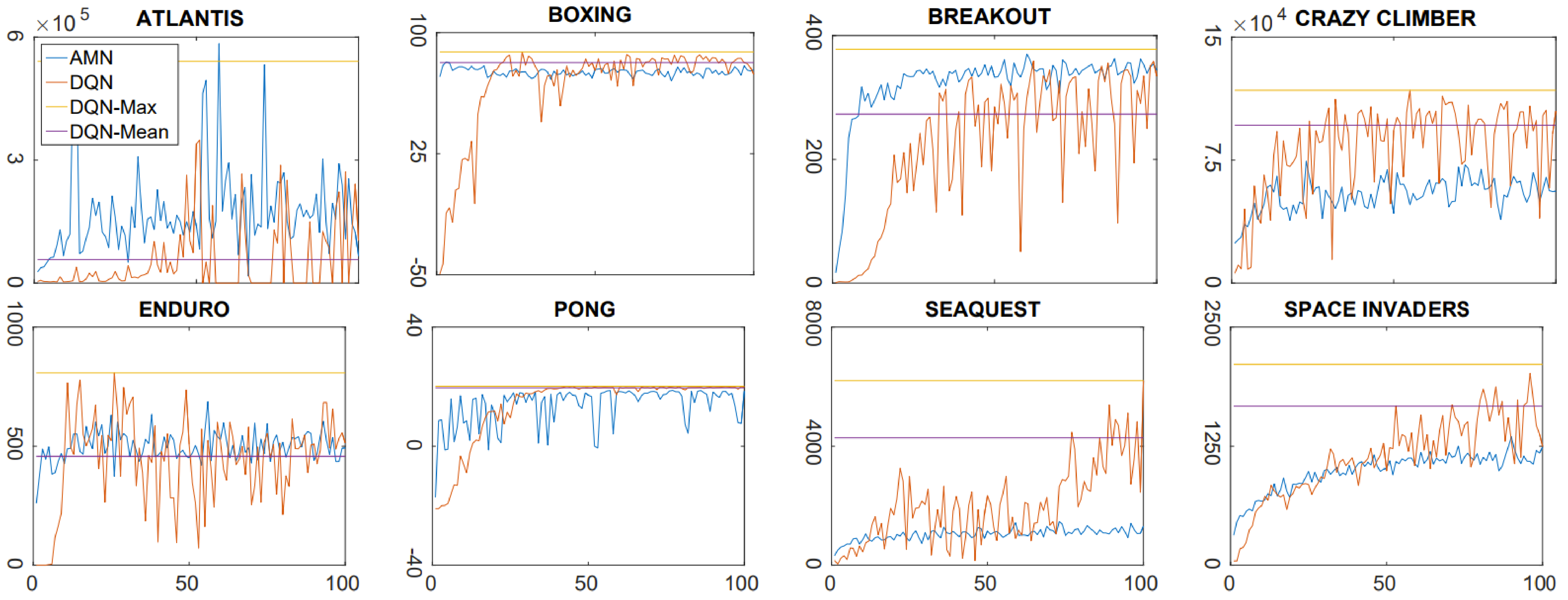
Given a state from a source task s_i, we define the policy objective over the multitask network as the cross-entropy between the expert network's policy and the current multitask policy, Actor-Mimic Network (AMN) policy:

$$\mathcal{L}^{i}_{policy}(\theta) = \sum_{a \in \mathcal{A}_{E_i}} \pi_{E_i}(a|s) \log \pi_{\text{AMN}}(a|s;\theta)$$

In contrast to the Q-learning objective which recursively relies on itself as a target value, we now have a stable supervised training signal (the expert network output) to guide the multitask network.

Parisottoet al. "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning"

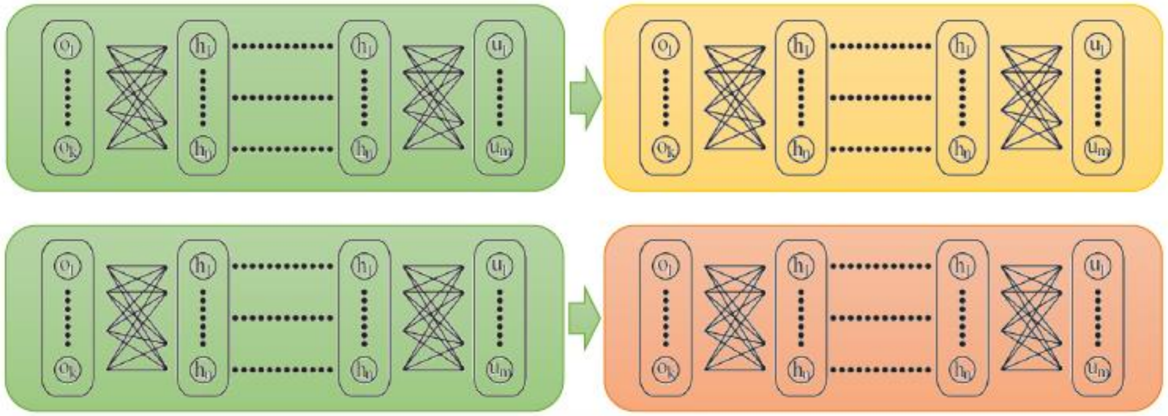
Distillation Transfer Results



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

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 Policies:



Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer

Coline Devin*, Abhishek Gupta*, Trevor Darrell, Pieter Abbeel, Sergey Levine

*These authors contributed equally
Berkeley Artificial Intelligence Research, Department of Electrical Engineering and Computer Science,
University of California, Berkeley

Multi-task learning summary

- More tasks -> more diversity -> better transfer
- 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)
- 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.**
- Rusuet al. (2016). **Policy Distillation.**
- Parisottoet al. (2016). **Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning.**
- Devin, et al. (2017). **Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer.**

What next?

- **Lecture**: Imitation Learning
- **Assignments**:
  - DDPG, by No. 25, EoD
  - A2C, by Nov. 18, EoD
  - REINFORCE, by Nov. 11, EOD
- **Quiz (on Canvas)**:
  - Soft Actor-Critic, by Nov. 13, EoD
- **Project**:
  - Final Project, by Dec. 2, EoD

Sergey Levin's lecture

- https://www.youtube.com/watch?v=brLZ2ny40n4&list=PLkFD6_40KJIxJMR-j5A1mkxK26gh_qg37&index=8&t=0s