Exploration: Part 2

CS 294-112: Deep Reinforcement Learning

Sergey Levine

Class Notes

1. Homework 4 due next Wednesday!

Recap: what's the problem?

this is easy (mostly)

Why?

this is impossible

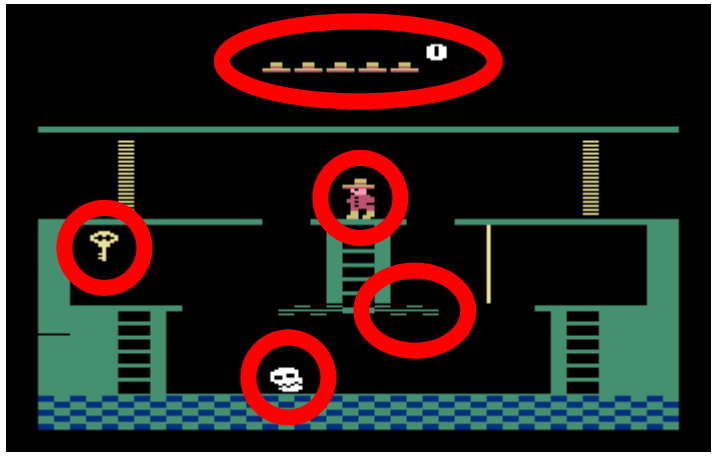
Recap: classes of exploration methods in deep RL

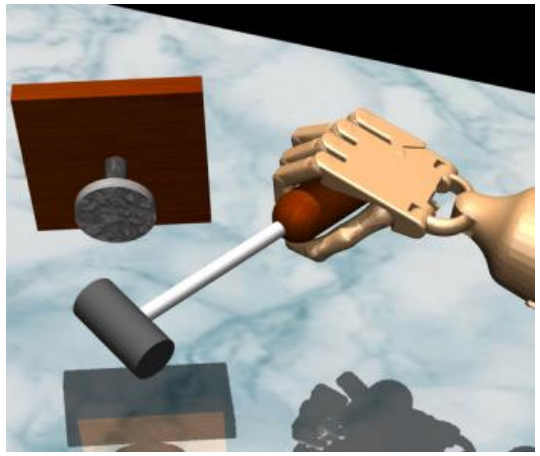
- Optimistic exploration:
  - new state = good state
  - requires estimating state visitation frequencies or novelty
  - typically realized by means of exploration bonuses
- Thompson sampling style algorithms:
  - learn distribution over Q-functions or policies
  - sample and act according to sample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

Count-based exploration

$$\text{use } r^+(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \mathcal{B}(N(\mathbf{s}))$$

But wait… what's a count?

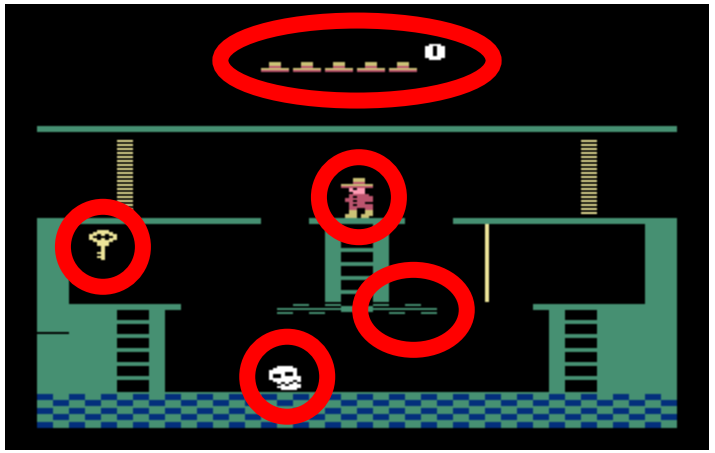


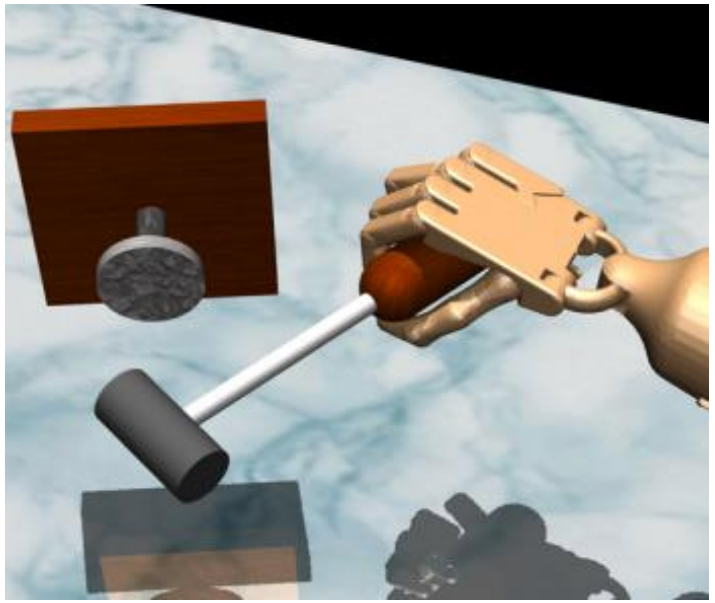


Uh oh… we never see the same thing twice!

But some states are more similar than others

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"

$$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{1}{\hat{N}(\mathbf{s})}}$$

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

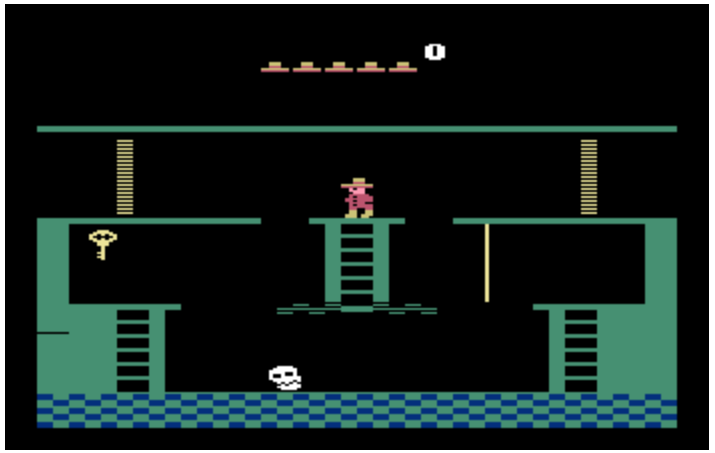
$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \qquad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)+1}{\hat{n}+1}$$

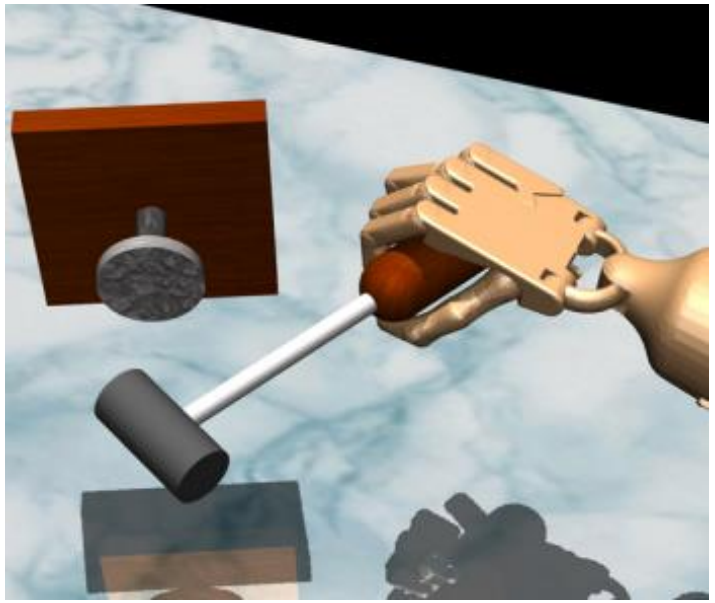
two equations and two unknowns!

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

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

What kind of model to use?



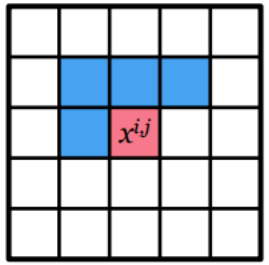


$$p_\theta(\mathbf{s})$$

need to be able to output densities, but doesn't necessarily need to produce great samples

opposite considerations from many popular generative models in the literature (e.g., GANs)

Bellemare et al.: "CTS" model: condition each pixel on its top-left neighborhood



$$p_\theta(\mathbf{s}) = \prod_{i,j} p_{\theta_{i,j}}(x^{i,j} | x^{i-1,j}, x^{i,j-1}, x^{i-1,j-1}, x^{i-1,j+1})$$

Counting with hashes

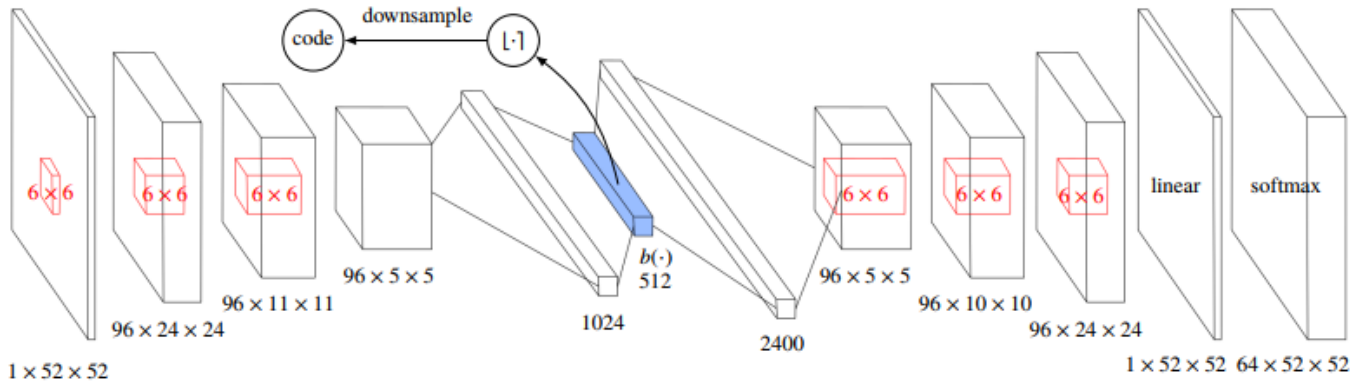
What if we still count states, but in a different space?

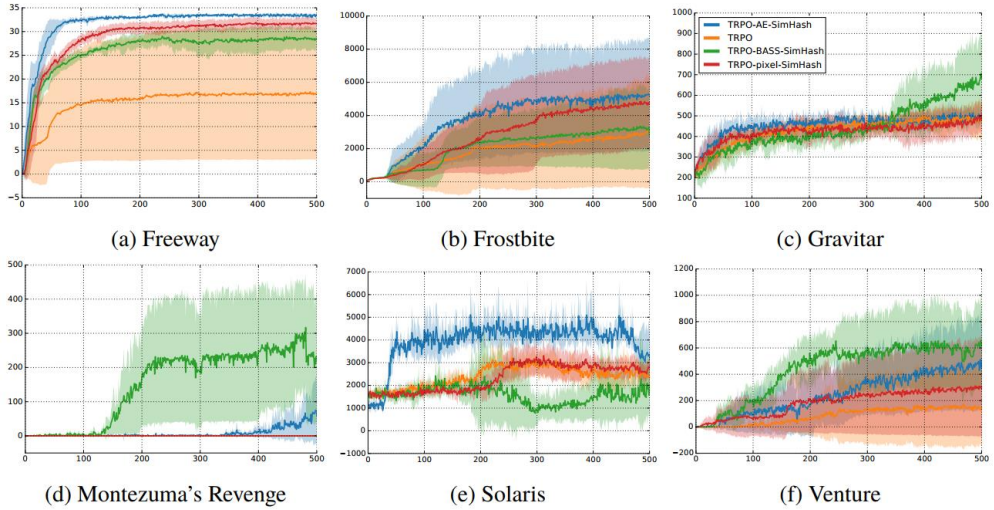
idea: compress $\mathbf{s}$ into a k-bit code via $\phi(\mathbf{s})$, then count $N(\phi(\mathbf{s}))$

shorter codes = more hash collisions

similar states get the same hash? maybe

improve the odds by *learning* a compression:





Tang et al. "#Exploration: A Study of Count-Based Exploration"

Implicit density modeling with exemplar models

$p_\theta(\mathbf{s})$ need to be able to output densities, but doesn't necessarily need to produce great samples

Can we explicitly compare the new state to past states?

Intuition: the state is **novel** if it is **easy** to distinguish from all previous seen states by a classifier

for each observed state $\mathbf{s}$, fit a classifier to classify that state against all past states $\mathcal{D}$, use classifier error to obtain density

$$p_\theta(\mathbf{s}) = \frac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$$

probability that classifier assigns that $\mathbf{s}$ is "positive"
positives: $\{\mathbf{s}\}$
negatives: $\mathcal{D}$

Fu et al. "EX2: Exploration with Exemplar Models…"

Implicit density modeling with exemplar models

hang on... aren't we just checking if $\mathbf{s} = \mathbf{s}$?

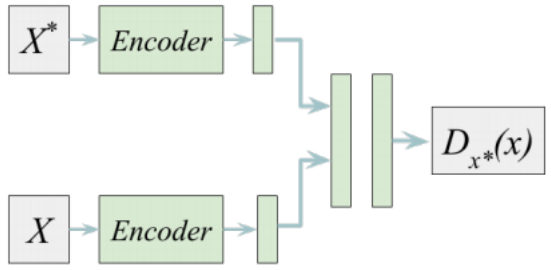
if $\mathbf{s} \in \mathcal{D}$, then the optimal $D_{\mathbf{s}}(\mathbf{s}) \neq 1$

in fact: $D^{\star}_{\mathbf{s}}(\mathbf{s}) = \dfrac{1}{1 + p(\mathbf{s})}$ → $p_\theta(\mathbf{s}) = \dfrac{1 - D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$

in reality, each state is unique, so we *regularize* the classifier

isn't one classifier per state a bit much?

train one *amortized* model: single network that takes in exemplar as input!



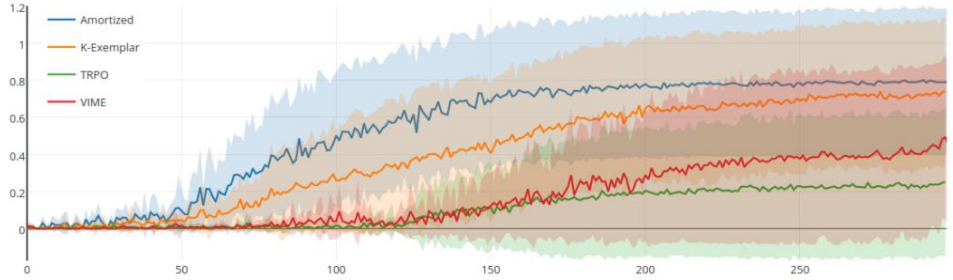


Figure 9: DoomMyWayHome+

Fu et al. "EX2: Exploration with Exemplar Models…"

Posterior sampling in deep RL

Thompson sampling:

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

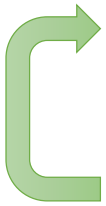
$a = \arg\max_a E_{\theta_a}[r(a)]$

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

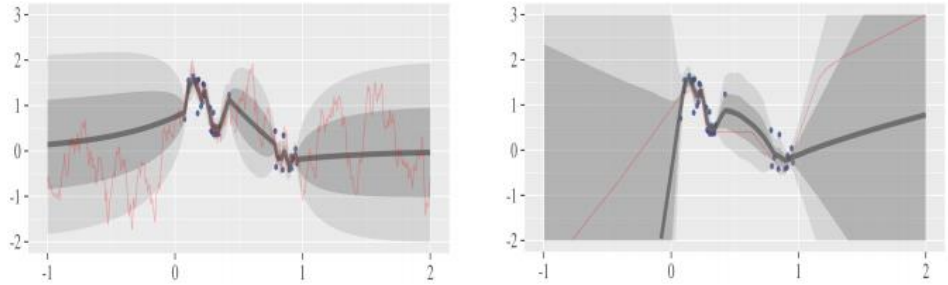
how can we represent a distribution over functions?

Bootstrap

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

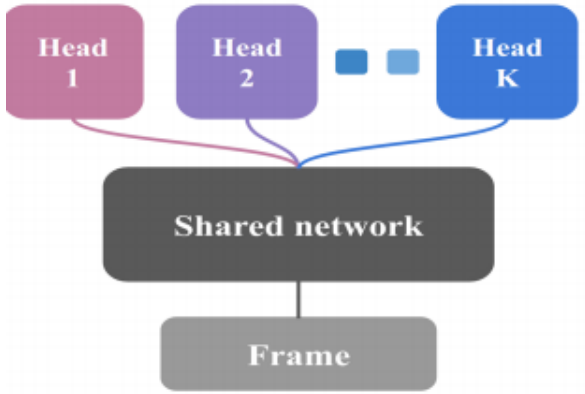
train each model f_{θ_i} on $\mathcal{D}_i$

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



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

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

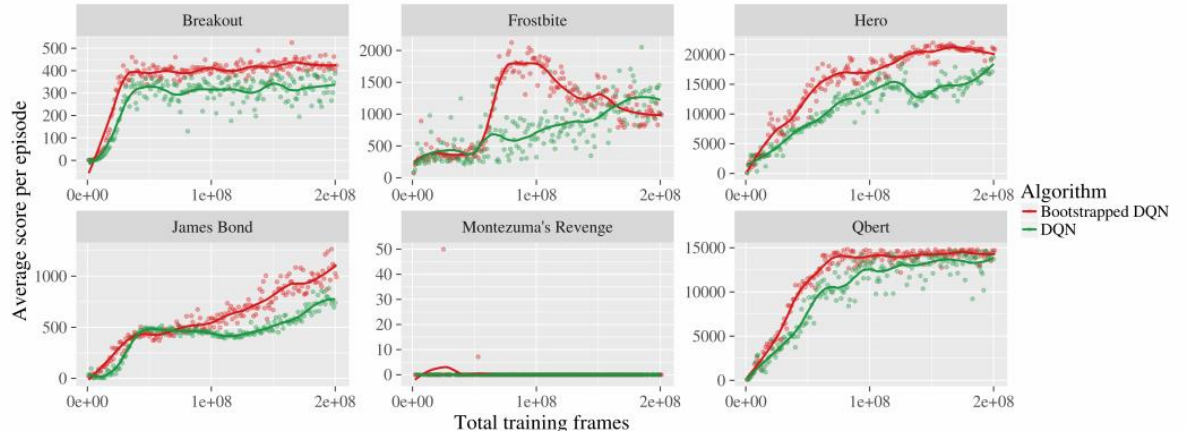


Osband et al. “Deep Exploration via Bootstrapped DQN”

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 somewhat 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 $D_{\mathrm{KL}}(p(z|y)\|p(z))$

learn about *transitions* $p_\theta(s_{t+1}|s_t, a_t)$: $z = \theta$

$y = (s_t, a_t, s_{t+1})$

$$D_{\mathrm{KL}}(p(\theta|h, s_t, a_t, s_{t+1})\|p(\theta|h))$$

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

history of all prior transitions

newly observed transition

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)

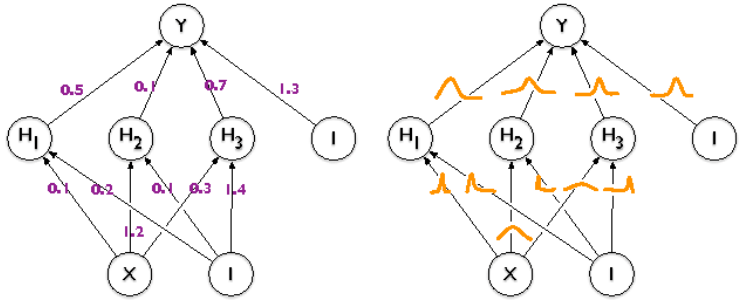
VIME implementation:

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

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

history of all prior transitions

newly observed transition



$q(\theta|\phi) \approx p(\theta|h)$ 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

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

ϕ

Houthooft et al. "VIME"

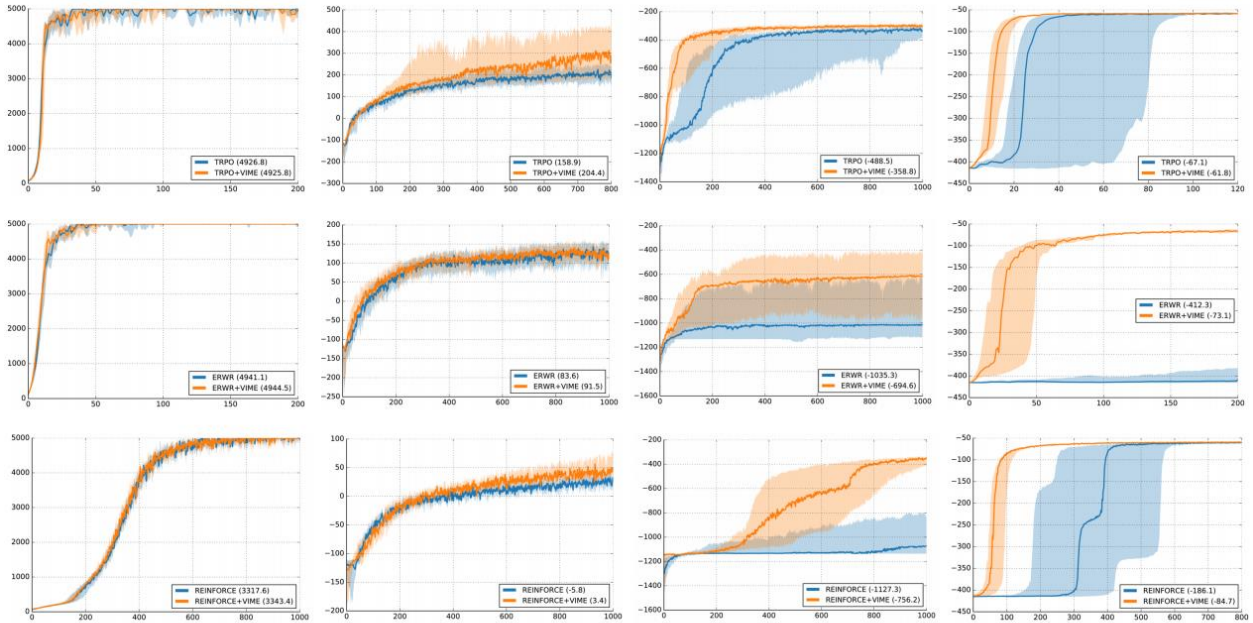
Reasoning about information gain (approximately)

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!

Recap: classes of exploration methods in deep RL

- Optimistic exploration:
  - Exploration with counts and pseudo-counts
  - Different models for estimating densities
- Thompson sampling style algorithms:
  - Maintain a distribution over models via bootstrapping
  - Distribution over Q-functions
- Information gain style algorithms
  - Generally intractable
  - Can use variational approximation to information gain

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.**

Break

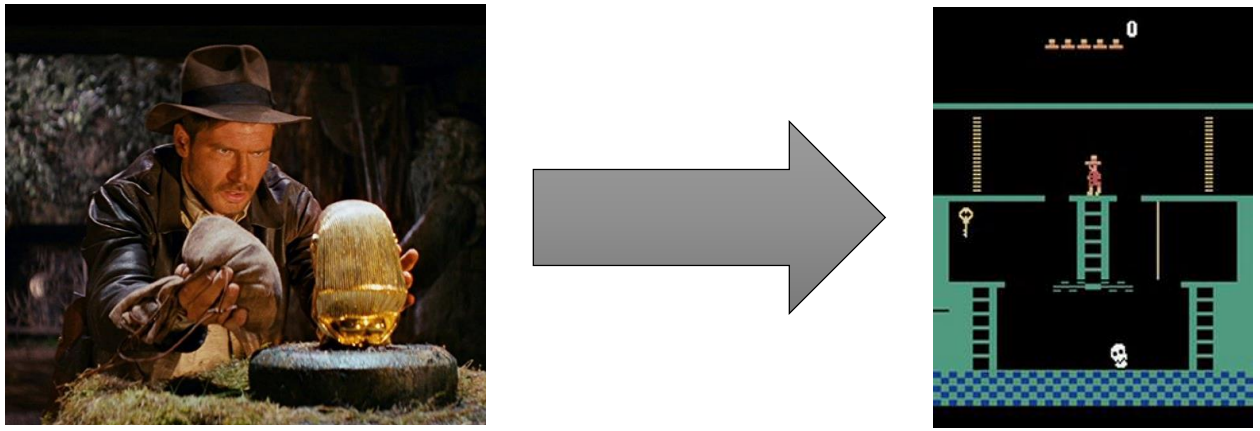
Next: transfer learning

1. The benefits of sharing knowledge across tasks
2. The transfer learning problem in RL
3. Transfer learning with source and target domains
4. Next week: multi-task learning, meta-learning

Back to Montezuma's Revenge

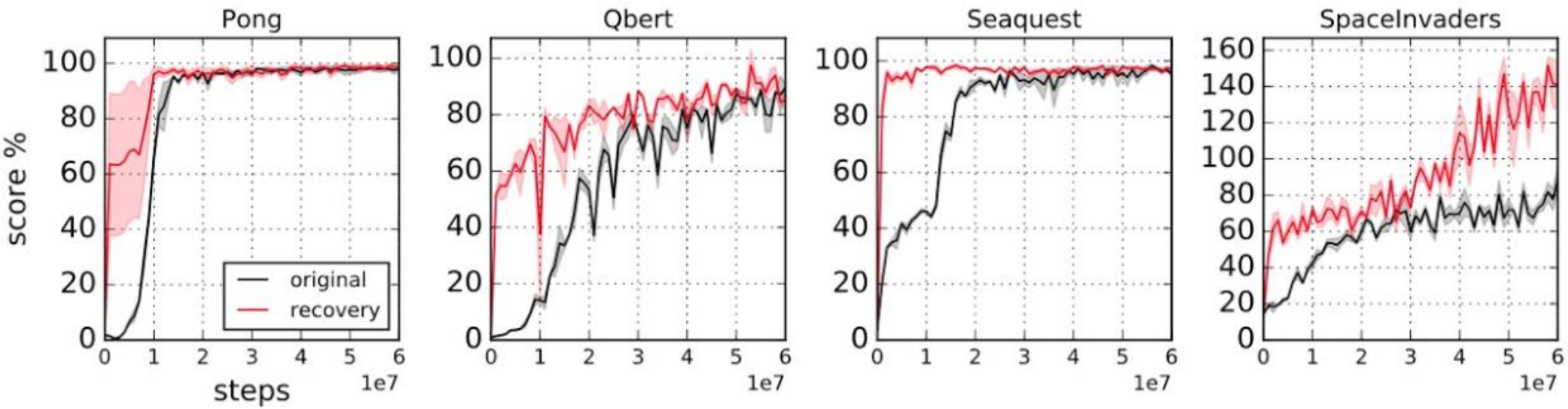
- We know what to do because we **understand** what these sprites mean!
- Key: we know it opens doors!
- Ladders: we know we can climb them!
- Skull: we don't know what it does, but we know it can't be good!
- **Prior understanding of problem structure can help us solve complex tasks quickly!**

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

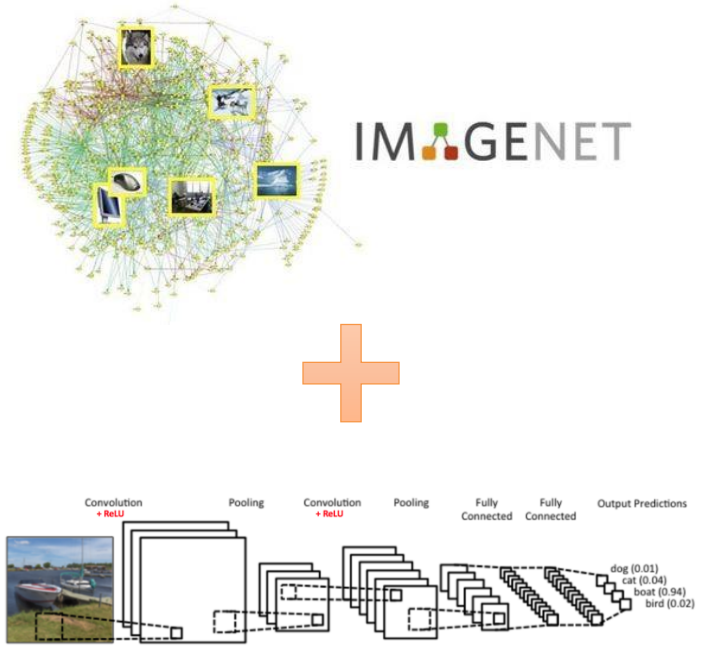
Policies trained for one set of circumstances might just work in a new domain, but no promises or guarantees

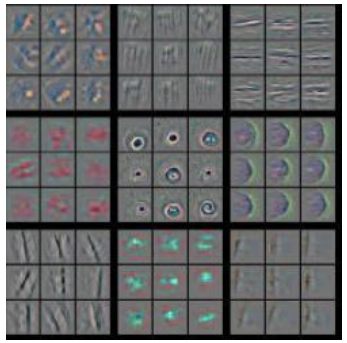
Levine*, Finn*, et al. ‘16

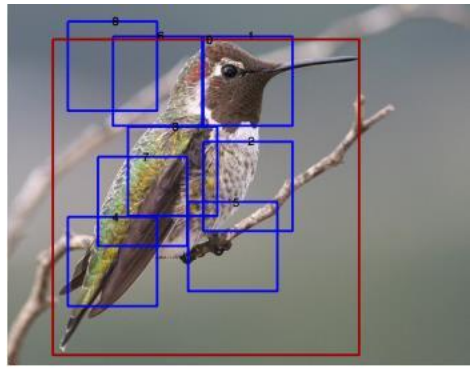
Devin et al. ‘17

Finetuning

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



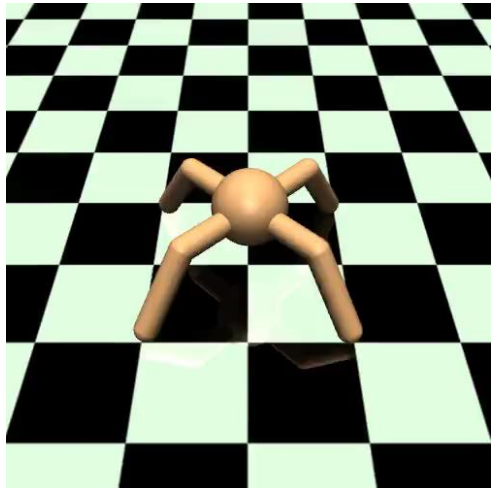




Where are the “ImageNet” features of RL?

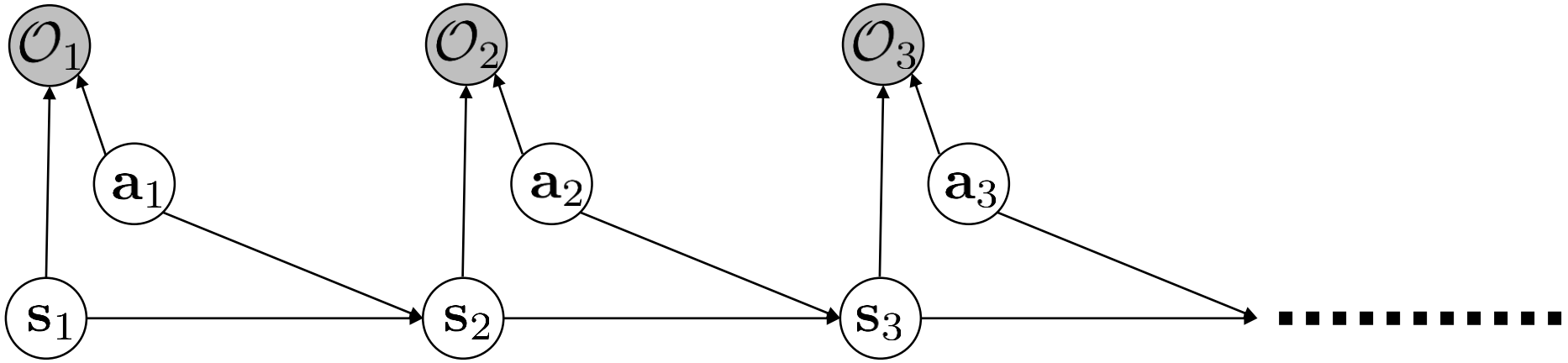
Challenges with finetuning in RL

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?

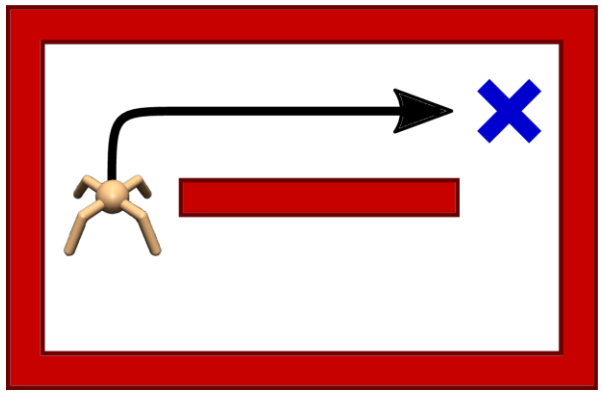


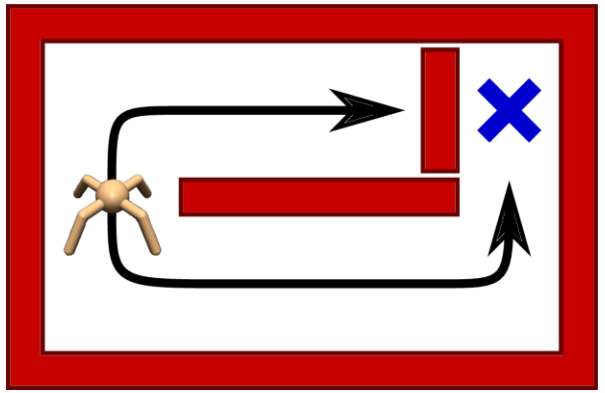
$$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s},\mathbf{a}) - V(\mathbf{s})) \text{ optimizes } \sum_t E_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]$$

policy entropy

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

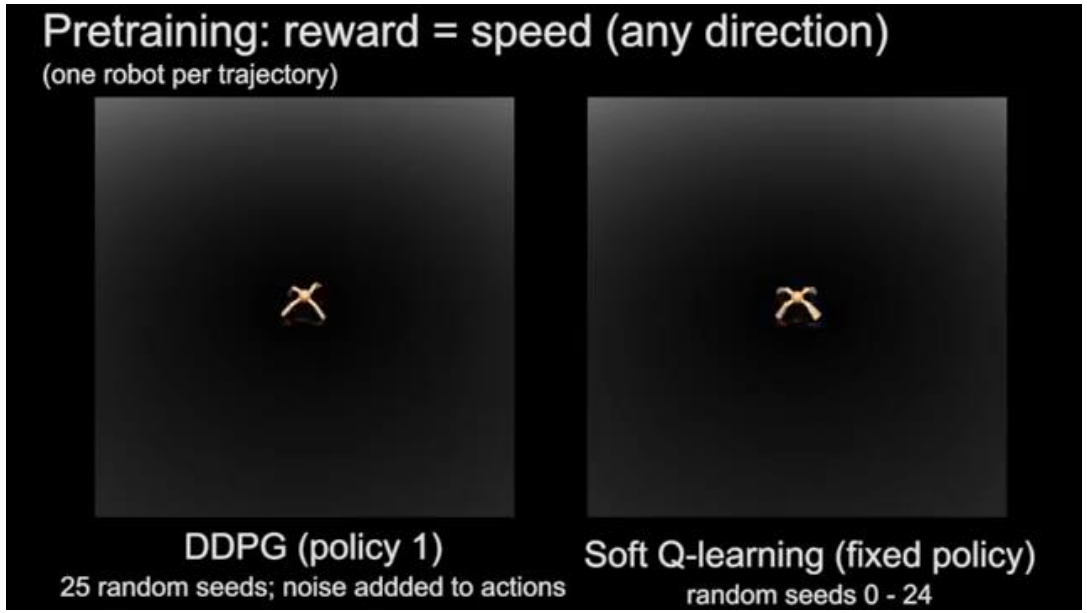
Example: pre-training for robustness

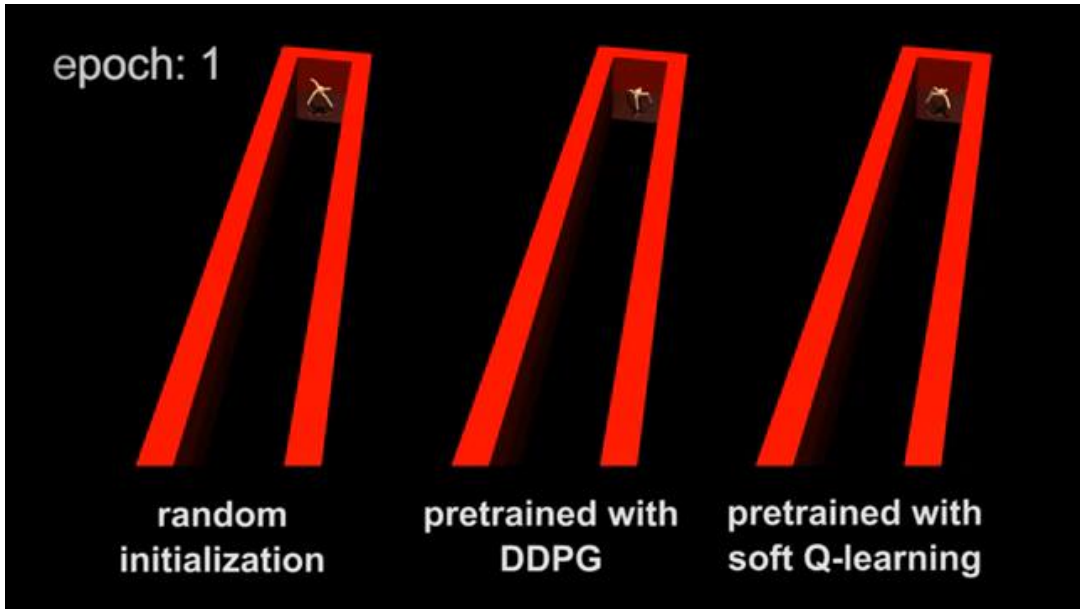


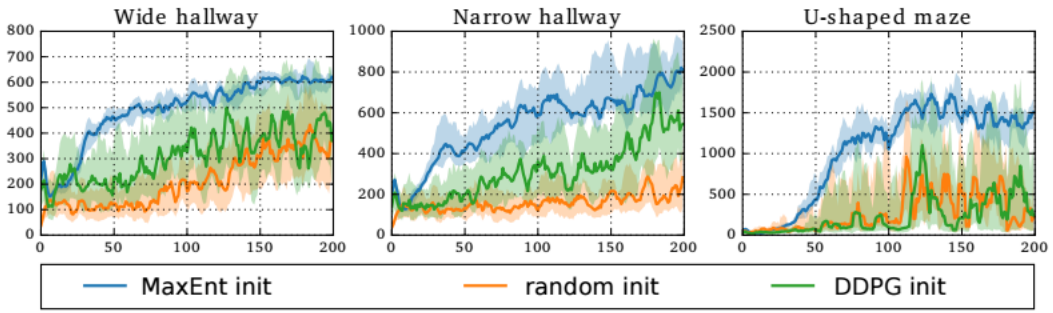


Learning to solve a task **in all possible ways** provides for more robust transfer!

Example: pre-training for diversity



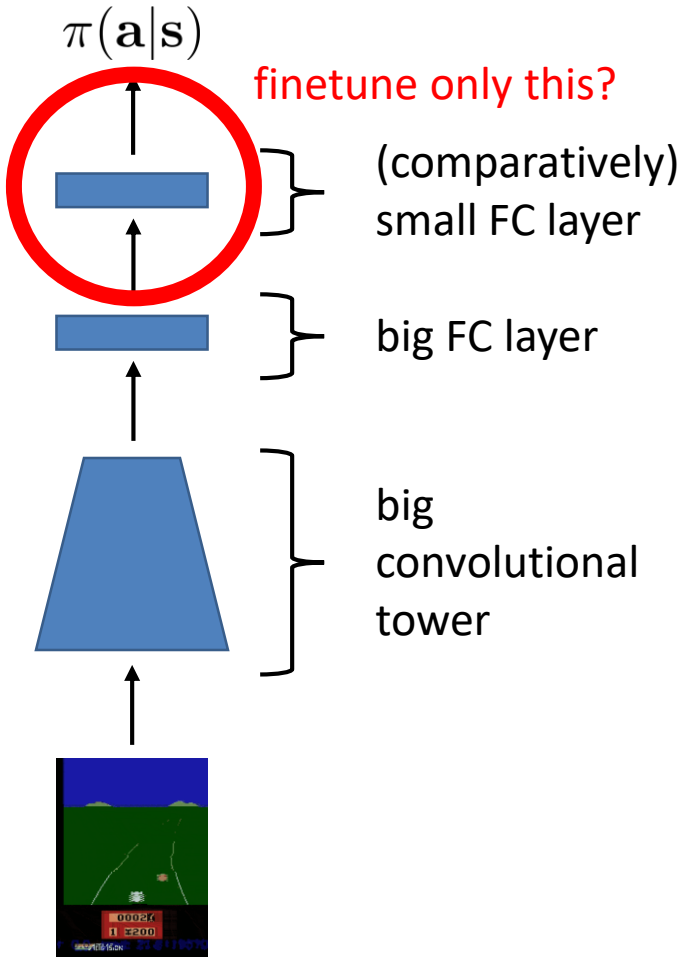




Haarnoja*, Tang*, et al. “Reinforcement Learning with Deep Energy-Based Policies”

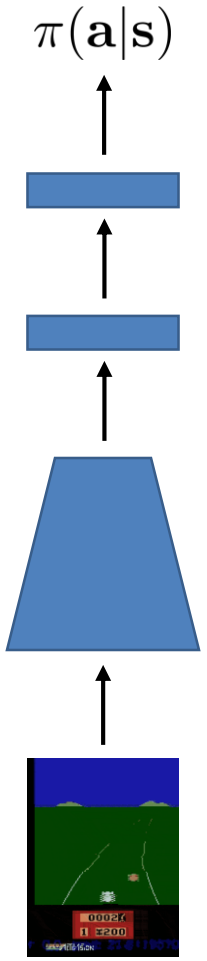
Architectures for transfer: progressive networks

- 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

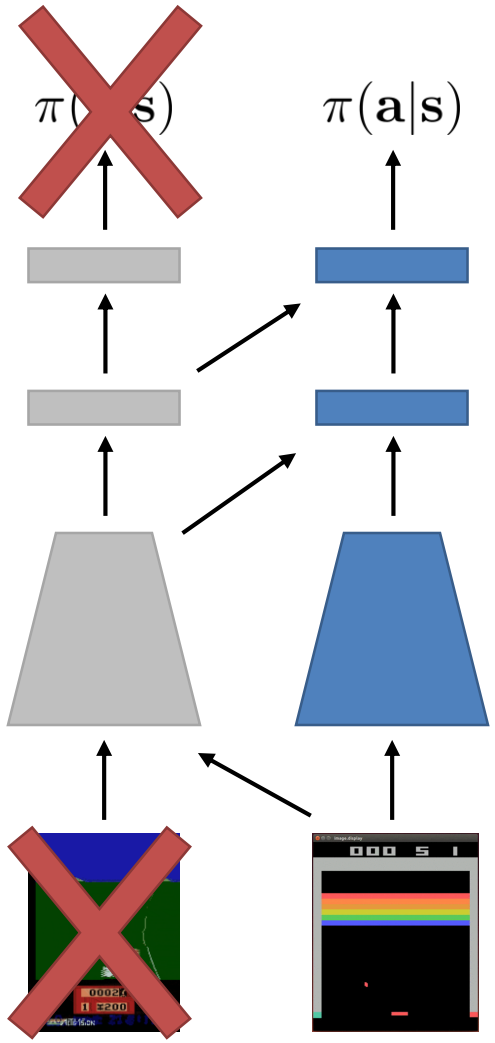
- An issue with finetuning
  - Deep networks work best when they are big
  - When we finetune, we typically want to use a little bit of experience
  - Little bit of experience + big network = overfitting
  - Can we somehow finetune a *small* network, but still pretrain a *big* network?
- Idea 1: finetune 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



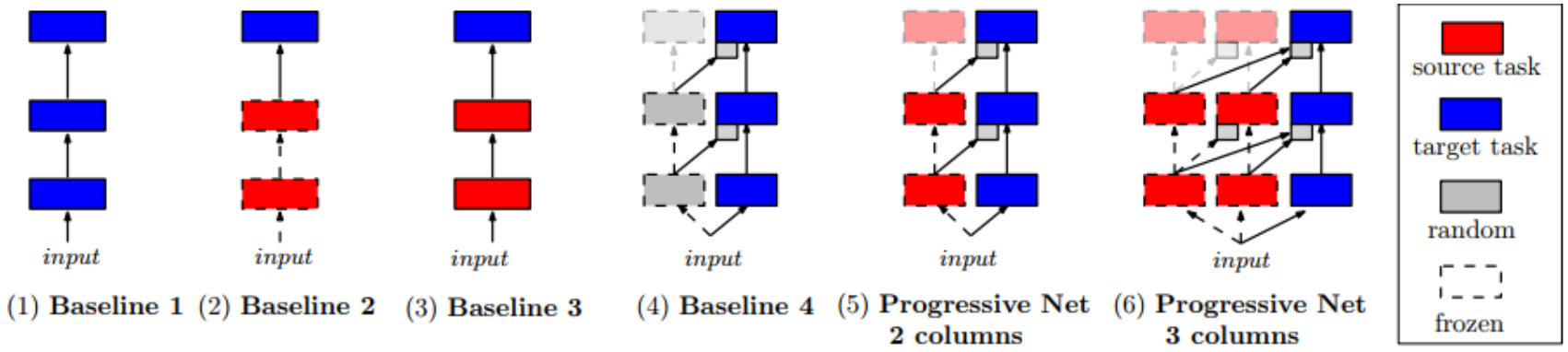
Rusu et al. “Progressive Neural Networks”

Architectures for transfer: progressive networks

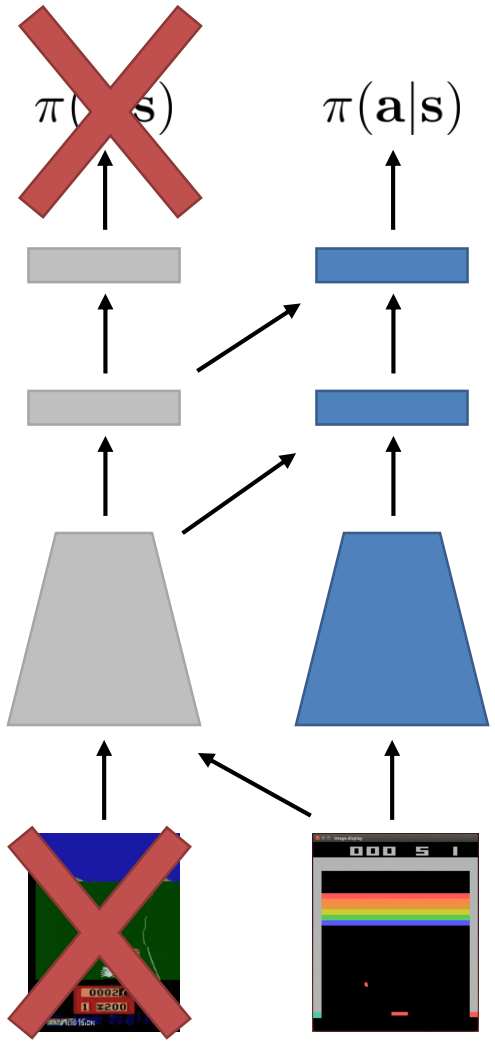
Does it work? sort of…

	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.



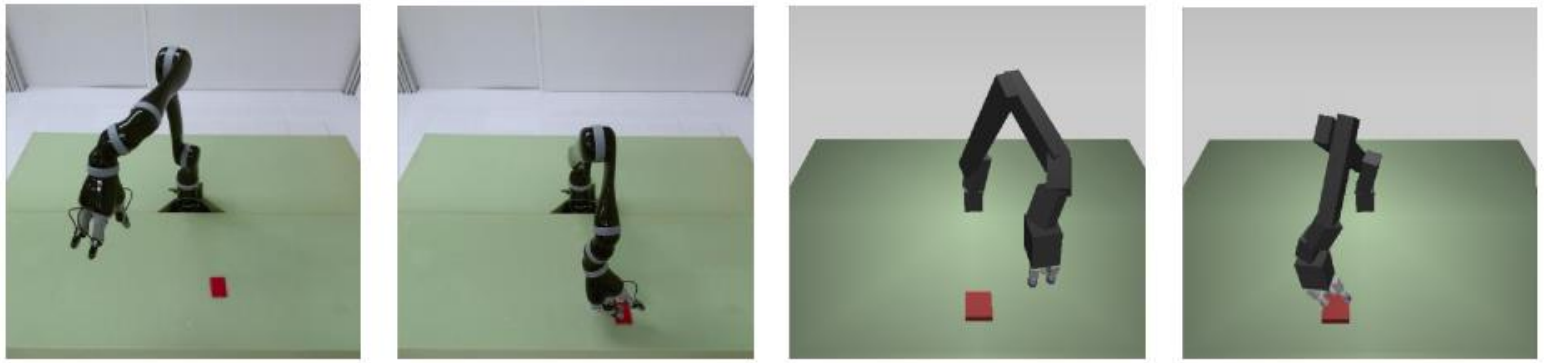
(1) Baseline 1 (2) Baseline 2 (3) Baseline 3 (4) Baseline 4 (5) Progressive Net 2 columns (6) Progressive Net 3 columns

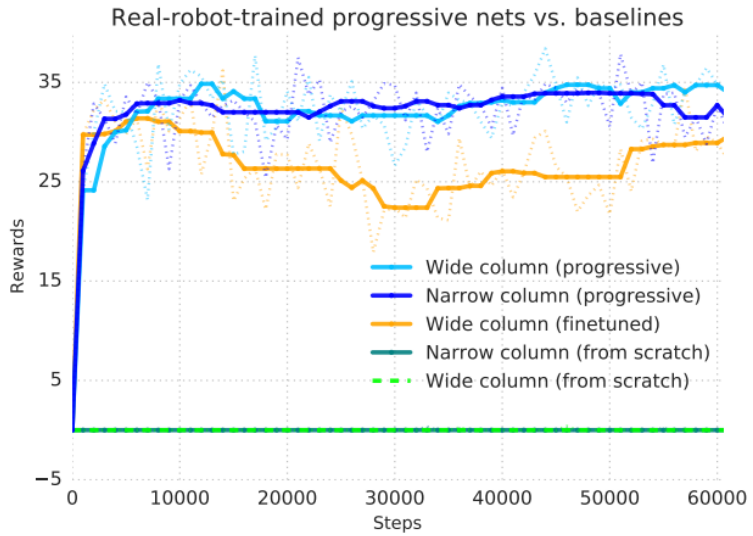


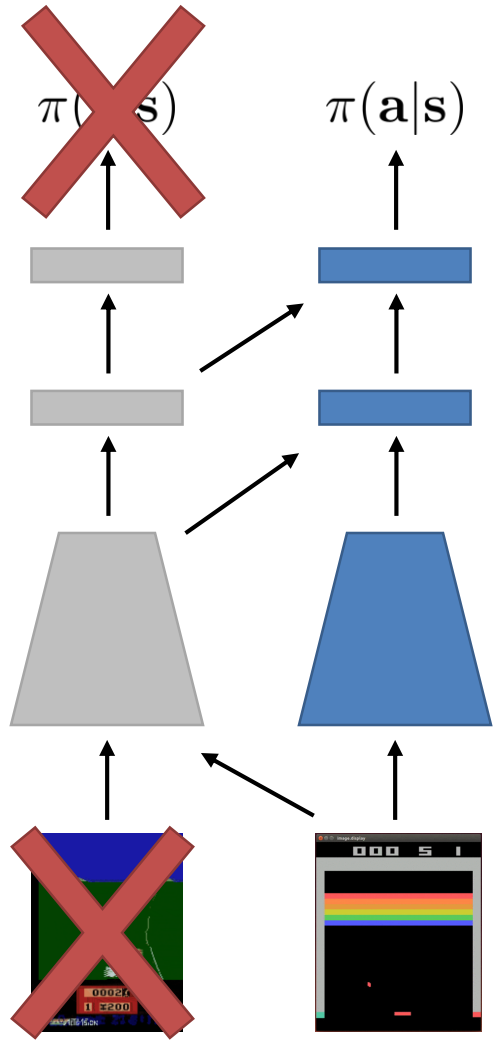
Rusu et al. “Progressive Neural Networks”

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

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) Architectures for transfer: progressive networks
   c) Fi[illegible]
2. Mu[illegible]sk
   a) [illegible]
   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

more on this next time!