Computer Engineering Department

Exploration in RL

Mohammad Hossein Rohban, Ph.D.

Spring 2025

Courtesy: Most of slides are adopted from CS 285, UC Berkeley.

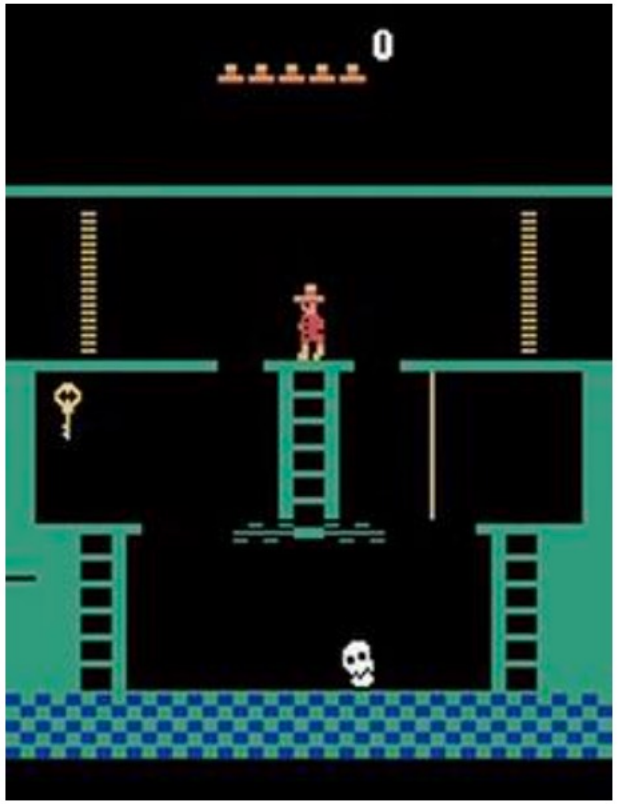
What's the problem?

this is easy (mostly)

Why?

this is impossible

Montezuma’s revenge



- Getting key = reward
- Opening door = reward
- Getting killed by skull = nothing (is it good? bad?)
- Finishing the game only weakly correlates with rewarding events
- We know what to do because we understand what these sprites mean!

Why exploration can be difficult?

- **Temporally extended** tasks like Montezuma's revenge become increasingly difficult based on
  - How extended the task is
  - How little you know about the rules
- Lets' assume a complex task
  - Consisting of multiple sub-task, each is a prerequisite for the next sub-task.
  - Each should be solved in a sequence to get a high reward
  - Epsilon greedy does not obviously help:
  - Suppose you mastered up to the k^{th} sub-task.
  - You have to exploit up to the k^{th} task and then explore onwards.
  - Now the chance to only explore in the sub-task (k+1) is $(1-\varepsilon)^{O(k)}\varepsilon^{O(1)}$.
  - For eps = 0.1, k = 5, this is ~ 6%. For eps = 0.5, this is ~ 3%.

Exploration and exploitation

- Two potential definitions of exploration problem:
  - How can an agent **discover** **high-reward strategies** that require a temporally **extended sequence of complex behaviors** that, **individually, are not rewarding**?
  - How can an agent decide whether to **attempt new behaviors** (to discover ones with higher reward) or **continue to do the best thing it knows so far**?

Optimal Exploration?

- Bayesian model of the environment. (POMDP with belief state)
- Optimize the expected reward under all uncertainties.
- Requires knowledge of state dynamic distribution class, the prior, and maintaining the belief state.
- Here we seek simpler solutions which could be extended to more complex scenarios.
- Compare the regret in such models against the Bayes' optimal approach.

$$\mathrm{Reg}(T) = TE[r(a^\star)] - \sum_{t=1}^{T} r(a_t)$$

expected reward of best action
(the best we can hope for in expectation)

actual reward of action
actually taken

Bandits

assume $r(a_i) \sim p_{\theta_i}(r_i)$

e.g., $p(r_i = 1) = \theta_i$ and $p(r_i = 0) = 1 - \theta_i$

$\theta_i \sim p(\theta)$, but otherwise unknown

this defines a POMDP with $\mathbf{s} = [\theta_1, \dots, \theta_n]$

belief state is $\hat{p}(\theta_1, \dots, \theta_n)$

how do we measure goodness of exploration algorithm?

regret: difference from optimal policy at time step T: $\mathrm{Reg}(T) = TE[r(a^\star)] - \sum_{t=1}^{T} r(a_t)$

expected reward of best action (the best we can hope for in expectation)

actual reward of action actually taken

- solving the POMDP yields the optimal exploration strategy
- but that's overkill: belief state is huge!
- we can do very well with much simpler strategies

Optimistic exploration

keep track of average reward $\hat{\mu}_a$ for each action a

exploitation: pick $a = \arg\max \hat{\mu}_a$

optimistic estimate: $a = \arg\max \hat{\mu}_a + C\underline{\sigma_a}$

some sort of variance estimate

intuition: try each arm until you are *sure* it's not great

example (Auer et al. Finite-time analysis of the multiarmed bandit problem):

$$a = \arg\max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$$

number of times we picked this action

$\mathrm{Reg}(T)$ is $O(\log T)$, provably as good as any algorithm

Probability matching/posterior sampling

assume $r(a_i) \sim p_{\theta_i}(r_i)$

this defines a POMDP with $\mathbf{s} = [\theta_1, \dots, \theta_n]$

belief state is $\underline{\hat{p}(\theta_1, \dots, \theta_n)}$

this is a *model* of our bandit

idea: sample $\theta_1, \dots, \theta_n \sim \hat{p}(\theta_1, \dots, \theta_n)$

pretend the model $\theta_1, \dots, \theta_n$ is correct

take the optimal action

update the model

- This is called posterior sampling or Thompson sampling
- Harder to analyze theoretically
- Can work very well empirically
- See: Chapelle & Li, “An Empirical Evaluation of Thompson Sampling.”

Information gain

Bayesian experimental design:

say we want to determine some latent variable z (e.g., z might be the optimal action, or its value)

which action do we take?

let $\mathcal{H}(\hat{p}(z))$ be the current entropy of our z estimate

let $\mathcal{H}(\hat{p}(z)|y)$ be the entropy of our z estimate after observation y (e.g., y might be $r(a)$)

the lower the entropy, the more precisely we know z

$$\mathrm{IG}(z, y) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)]$$

typically depends on action, so we have $\mathrm{IG}(z, y|a)$

Information gain example

$$\mathrm{IG}(z, y|a) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)|a]$$

how much we learn about z from action a, given current beliefs

Example bandit algorithm:
Russo & Van Roy "Learning to Optimize via Information-Directed Sampling"

$y = r_a$, $z = \theta_a$ (parameters of model $p(r_a)$)

$g(a) = \mathrm{IG}(\theta_a, r_a|a)$ – information gain of a

$\Delta(a) = E[r(a^\star) - r(a)]$ – expected suboptimality of a

choose a according to $\arg\min_a \dfrac{\Delta(a)^2}{g(a)}$

don't take actions that you're sure are suboptimal

don't bother taking actions if you won't learn anything

General themes

UCB:

$$a = \arg\max \hat{\mu}_a + \sqrt{\frac{2 \ln T}{N(a)}}$$

Thompson sampling:

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

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

Info gain:

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

- Most exploration strategies require some kind of uncertainty estimation (even if it's naïve)
- Usually assumes some value to new information
  - Assume unknown = good (optimism)
  - Assume sample = truth
  - Assume information gain = good

Optimistic exploration in RL

UCB: $$a = \arg\max \hat{\mu}_a + \underbrace{\sqrt{\frac{2\ln T}{N(a)}}}_{\text{"exploration bonus"}}$$

lots of functions work, so long as they decrease with $N(a)$

can we use this idea with MDPs?

count-based exploration: use $N(\mathbf{s}, \mathbf{a})$ or $N(\mathbf{s})$ to add *exploration bonus*

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

bonus that decreases with $N(\mathbf{s})$

use $r^+(\mathbf{s}, \mathbf{a})$ instead of $r(\mathbf{s}, \mathbf{a})$ with any model-free algorithm

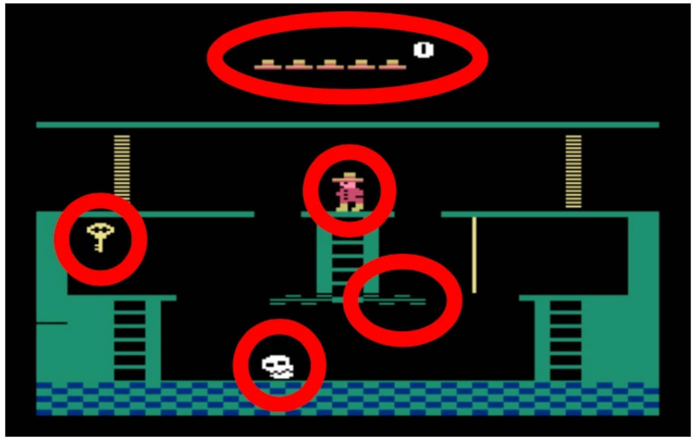
+ simple addition to any RL algorithm

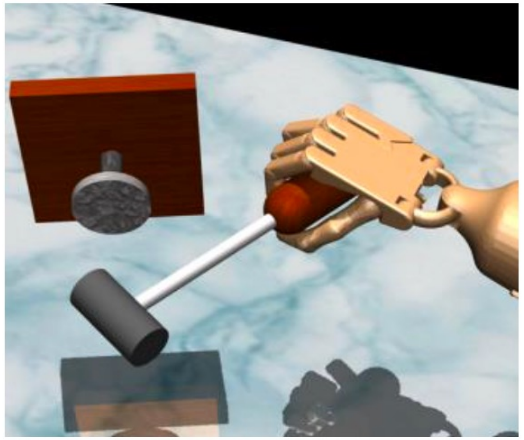
- need to tune bonus weight

The trouble with counts

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

But wait… what's a count?





Uh oh… we never see the same thing twice!

But some states are more similar than others

Fitting generative models

idea: fit a density model $p_\theta(\mathbf{s})$ (or $p_\theta(\mathbf{s}, \mathbf{a})$)

$p_\theta(\mathbf{s})$ might be high even for a new $\mathbf{s}$ $P'(s) \leftarrow Alg(P(s), s)$

if $\mathbf{s}$ is similar to previously seen states

can we use $p_\theta(\mathbf{s})$ to get a "pseudo-count"?

if we have small MDPs

the true probability is:

$$P(\mathbf{s}) = \frac{N(\mathbf{s})}{n}$$

count

probability/density

total states visited

after we see $\mathbf{s}$, we have:

$$P'(\mathbf{s}) = \frac{N(\mathbf{s}) + 1}{n + 1}$$

$N(s)$

n

Exploring with pseudo-counts

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

take a step i and observe $\mathbf{s}_i$ a_{i-1}

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" → $(s_{i-1}, s_i, r_i^+, a_{i-1})$

$\frac{1}{\sqrt{\hat{N}(s)}}$

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

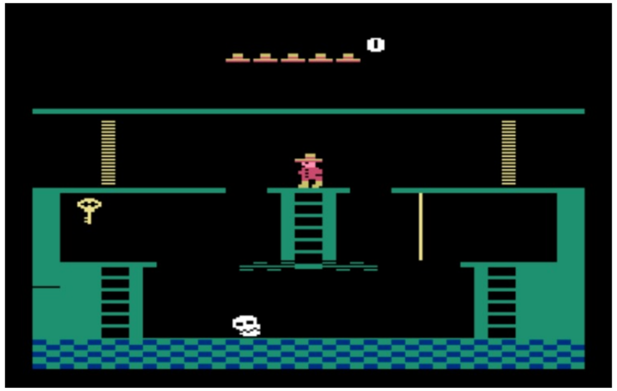
$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \qquad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i) + 1}{\hat{n} + 1}$$

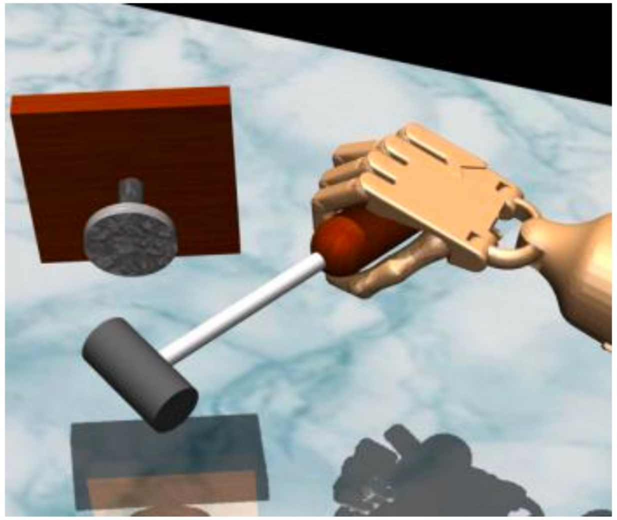
two equations and two unknowns!

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

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

What kind of generative modeling to use?





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

$s_1, \dots, s_m$

s

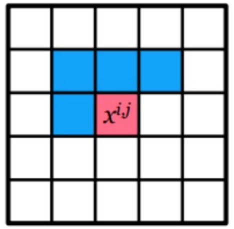
neigh(s) $\rightarrow \boxed{\theta} \rightarrow$

$$P_\theta(s_1, \dots, s_m) = p(s_1)\,p(s_2|s_1)\,p(s_3|s_2,s_1)\cdots p(s_m|s_{m-1},\dots,s_1)$$

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

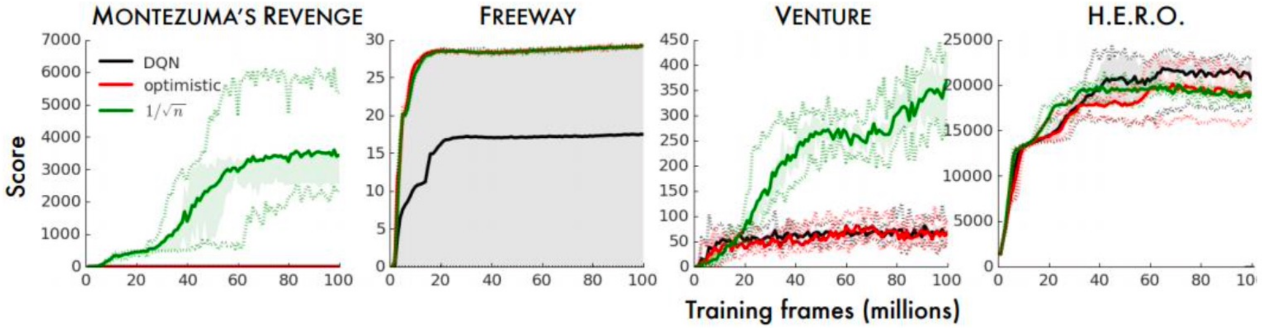
opposite considerations from many popular generative models in the literature (e.g., GANs)

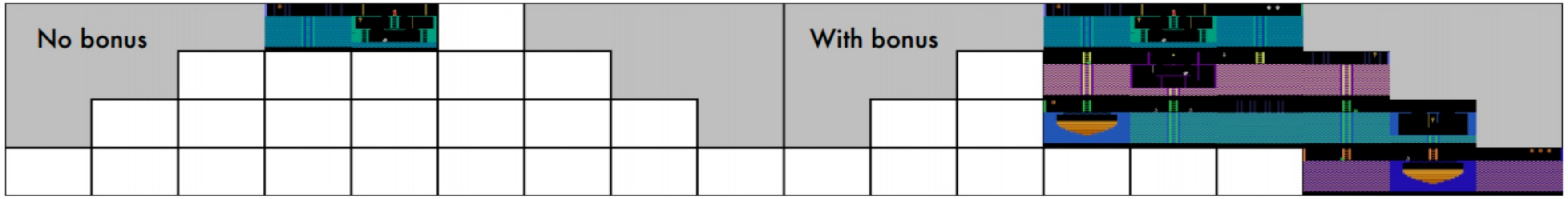
Bellemare et al.: "CTS" model: condition each pixel on its top-left neighborhood



Other models: stochastic neural networks, compression length, EX2

Does it work?





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

Counting with hashes

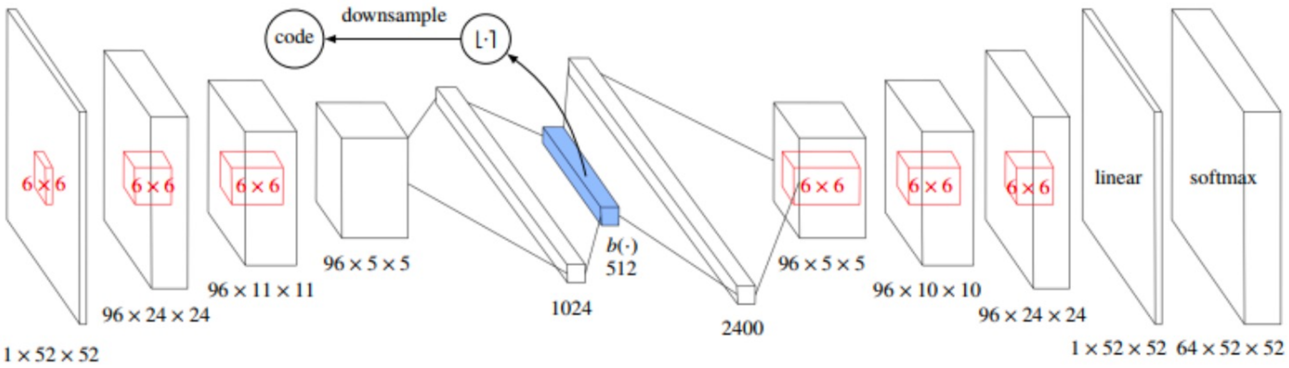
What if we still count states, but in a different space?

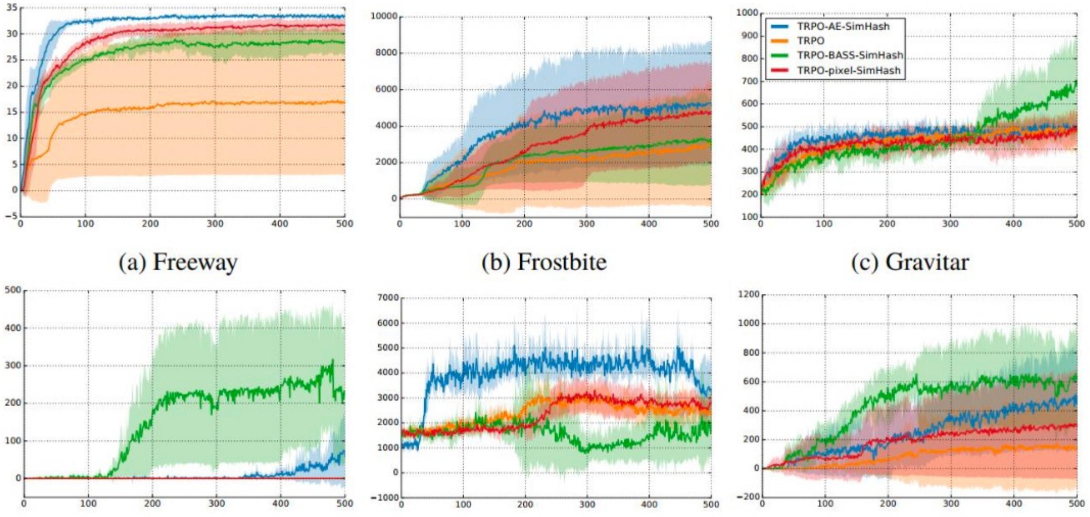
idea: compress $\mathbf{s}$ into a k-bit code via $\phi(\mathbf{s})$, then count $N(\phi(\mathbf{s}))$

shorter codes = more hash collisions

similar states get the same hash? maybe

improve the odds by *learning* a compression:





(a) Freeway (b) Frostbite (c) Gravitar

(d) Montezuma's Revenge (e) Solaris (f) Venture

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

$$D_{x^*} = \underset{D \in \mathcal{D}}{\arg\max} \left(E_{\delta_{x^*}}[\log D(x)] + E_{P_{\mathcal{X}}}[\log 1 - D(x)] \right) . \tag{1}$$

Proposition 1. *(Optimal Discriminator) For a discrete distribution $P_{\mathcal{X}}(x)$, the optimal discriminator D_{x^*} for exemplar x^* satisfies*

$$D_{x^*}(x) = \frac{\delta_{x^*}(x)}{\delta_{x^*}(x) + P_{\mathcal{X}}(x)} \qquad \textit{and} \qquad D_{x^*}(x^*) = \frac{1}{1 + P_{\mathcal{X}}(x^*)}.$$

Proof. The proof is obtained by taking the derivative of the loss in Eq. (1) with respect to $D(x)$, setting it to zero, and solving for $D(x)$. □

Implicit density modeling with exemplar models

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

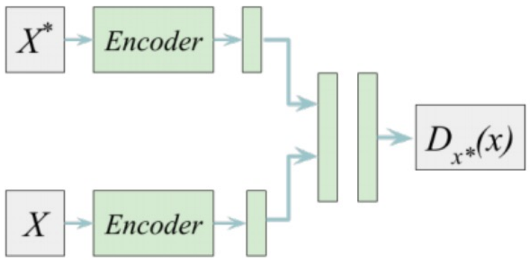
if $\mathbf{s} \in \mathcal{D}$, then the optimal $D_{\mathbf{s}}(\mathbf{s}) \neq 1$

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

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!



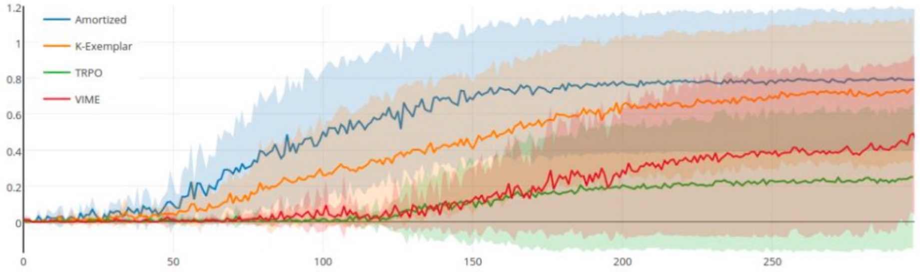


Figure 9: DoomMyWayHome+

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

Heuristic estimation of counts via errors

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

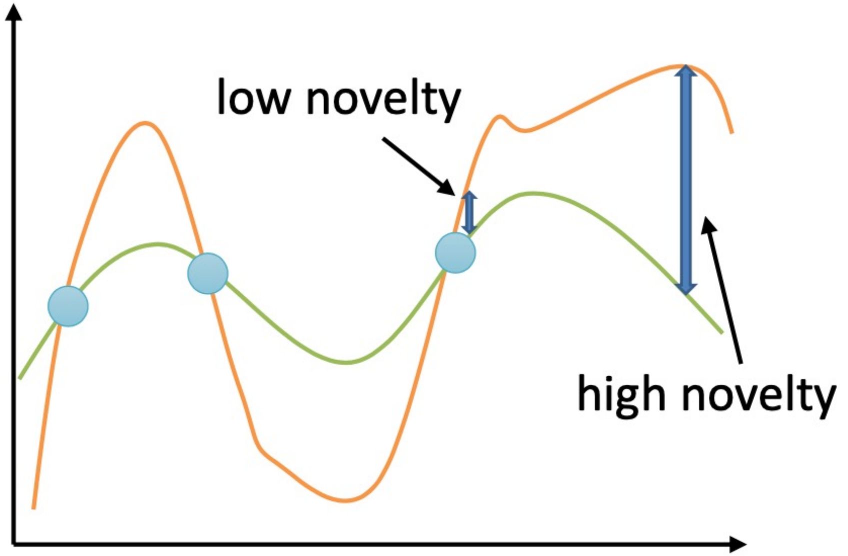
…and doesn't even need to output great densities

…just need to tell if state is **novel** or not!

let's say we have some **target** function $f^\star(\mathbf{s}, \mathbf{a})$

given our buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, fit $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$

use $\mathcal{E}(\mathbf{s}, \mathbf{a}) = ||\hat{f}_\theta(\mathbf{s}, \mathbf{a}) - f^\star(\mathbf{s}, \mathbf{a})||^2$ as bonus



Heuristic estimation of counts via errors

what should we use for $f^\star(\mathbf{s}, \mathbf{a})$?

one common choice: set $f^\star(\mathbf{s}, \mathbf{a}) = \mathbf{s}'$ – i.e., next state prediction

even simpler: $f^\star(\mathbf{s}, \mathbf{a}) = f_\phi(\mathbf{s}, \mathbf{a})$, where ϕ is a *random* parameter vector

Burda et al. Exploration by random network distillation. 2018.

Posterior sampling in deep RL

$p(Q|\underline{x}) = \dfrac{p(\underline{x}|Q)\,p(Q)}{p(\underline{x})}$

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"

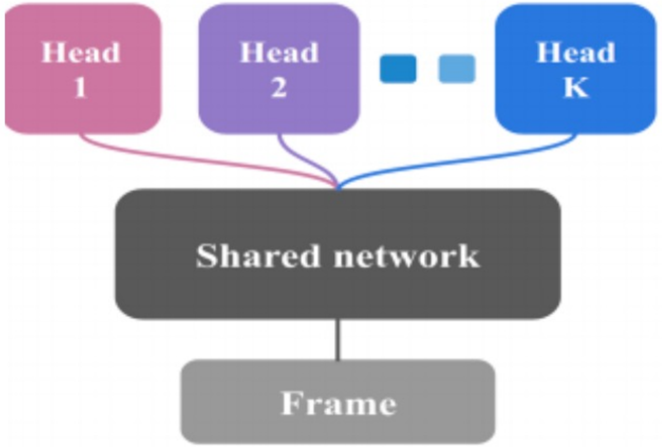
Bootstrap

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}

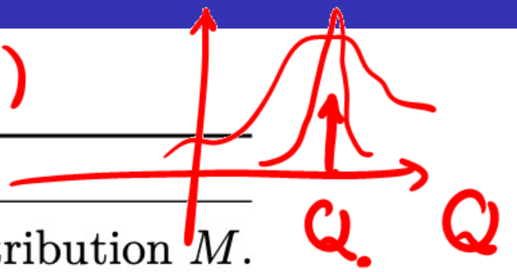
training N big neural nets is expensive, can we avoid it?



Osband et al. "Deep Exploration via Bootstrapped DQN"

Bootstrap

$P(Q) \rightarrow \delta(Q - Q_0)$



Algorithm 1 Bootstrapped DQN

Input: Value function networks Q with K outputs $\{Q_k\}_{k=1}^{K}$. Masking distribution M.
Let B be a replay buffer storing experience for training.
for each episode **do**
    Obtain initial state from environment s_0
    Pick a value function to act using $k \sim \text{Uniform}\{1, \dots, K\}$
    for step $t = 1, \dots$ until end of episode **do**
        Pick an action according to $a_t \in \arg\max_a Q_k(s_t, a)$
        Receive state s_{t+1} and reward r_t from environment, having taking action a_t
        Sample bootstrap mask $m_t \sim M$
        Add $(s_t, a_t, r_{t+1}, s_{t+1}, m_t)$ to replay buffer B
    end for
end for

Sample

$$Q_m(s_t, a_t) \leftarrow Q_m(s_t, a_t) + \alpha\left(\left[r_{t+1} + \gamma \max_a \bar{Q}_m(s_{t+1}, a)\right] - Q_m(s_t, a_t)\right)$$

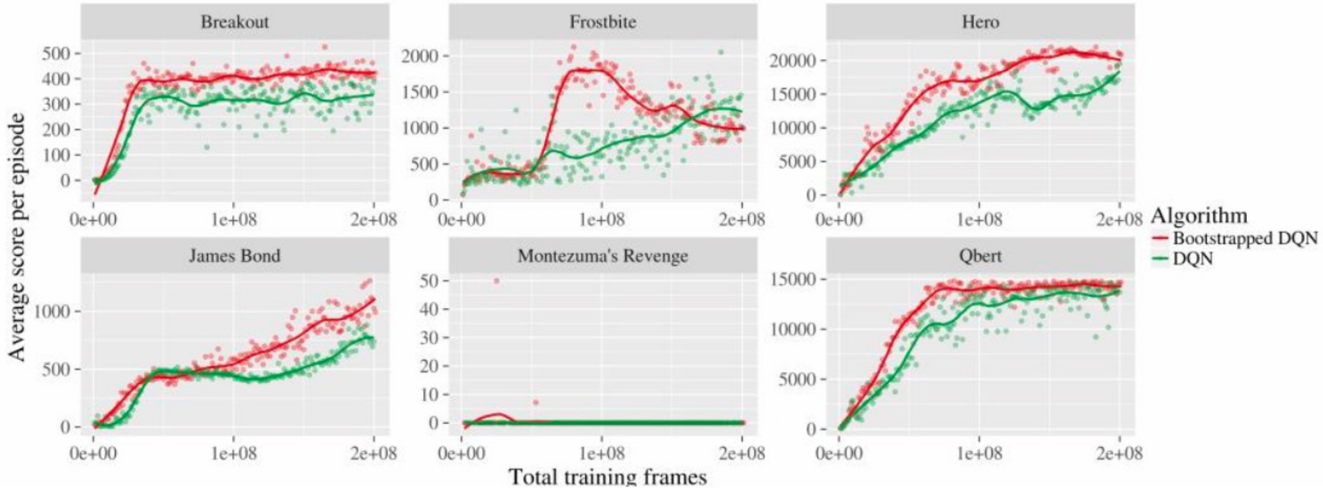
Sample $(s_t, a_t, s_{t+1}, r_{t+1}, m_t)$ from B

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”

Go-Explore Method

First return, then explore

Adrien Ecoffet*[1], Joost Huizinga*[1], Joel Lehman[1], Kenneth O. Stanley[1] & Jeff Clune[1,2]

[1] *Uber AI, San Francisco, CA, USA*

[2] *OpenAI, San Francisco, CA, USA (work done at Uber AI)*

* *These authors contributed equally to this work*

Authors' note: This is the pre-print version of this work. You most likely want the updated, published version, which can be found as an unformatted PDF at `https://adrien.ecoffet.com/files/go-explore-nature.pdf` or as a formatted web-only document at `https://tinyurl.com/Go-Explore-Nature`.

Please cite as:

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O. and Clune, J. First return, then explore. *Nature* **590,** 580–586 (2021). https://doi.org/10.1038/s41586-020-03157-9

Go-Explore Method

Sparse Rewards: Requires a lot of exploration;

e.g. task: reaching coffee machine.
define reward as whether or not the machine has been reached.

Dense Rewards:

e.g. define reward as the inverse Euclidean distance to the machine.

- Deceptive
- Reward Hacking

Solving two major issues

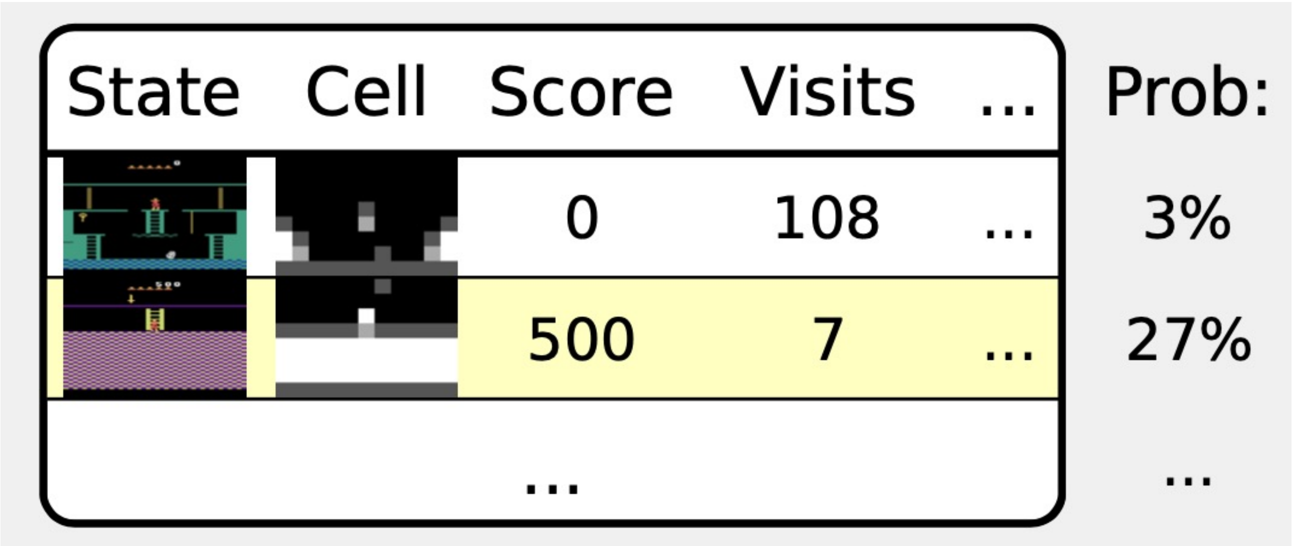
Solves two issue:

Detachment: the algorithm loses track of previously seen and promising states

Derailment: exploratory mechanisms of the policy prevents it from reaching previously visited states

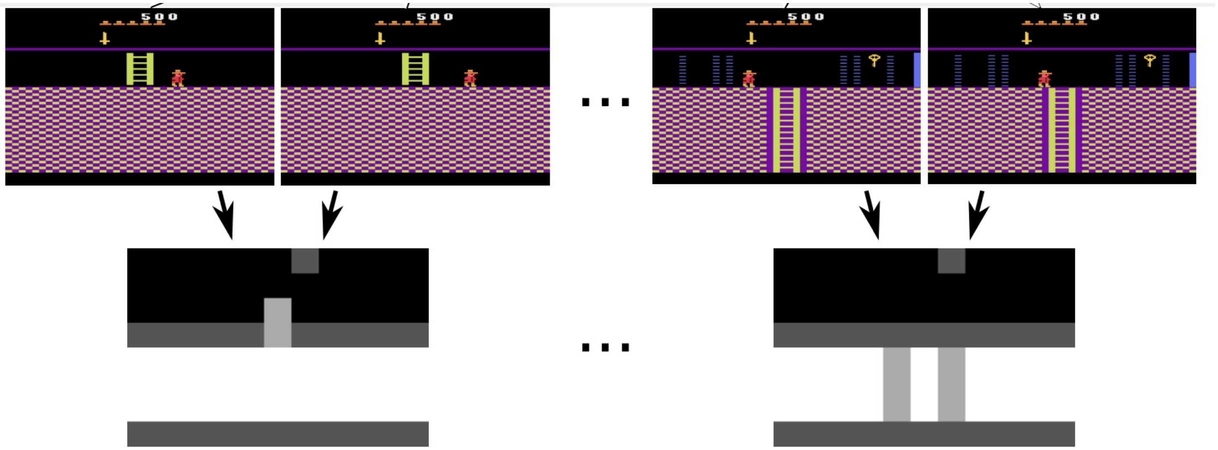
State Archive

- Builds an archive of the different states visited in the environment.
- Probabilistically selects a state to return to from the archive.
- Goes back (i.e. returns) to that state, then explores from that state.
- Updates the archive with all novel states encountered



How to build archive

Non-trivial environments have too many states to store explicitly.
Similar states are grouped into cells.
States are only considered novel if they are in a cell that does not yet exist in the archive.



To maximize performance, if a state maps to an already known cell, but is associated with a better trajectory, that state and its associated trajectory will replace the state and trajectory currently associated with that cell.

How to return to a state?

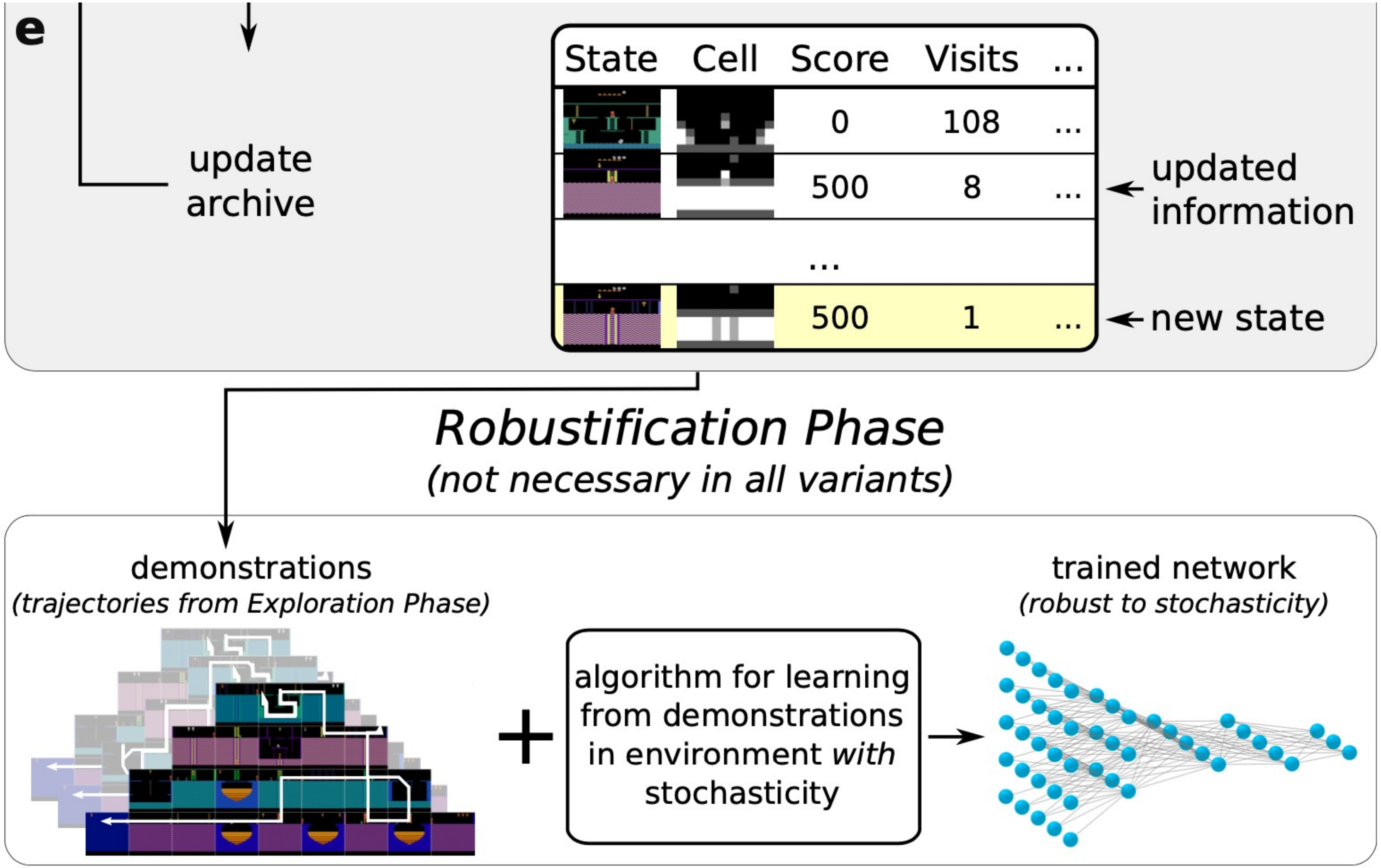
The inner state of any simulator can be saved and restored at a later time, making it possible to instantly return to a previously seen state.

Else, one can train a goal-conditioned policy for this purpose.

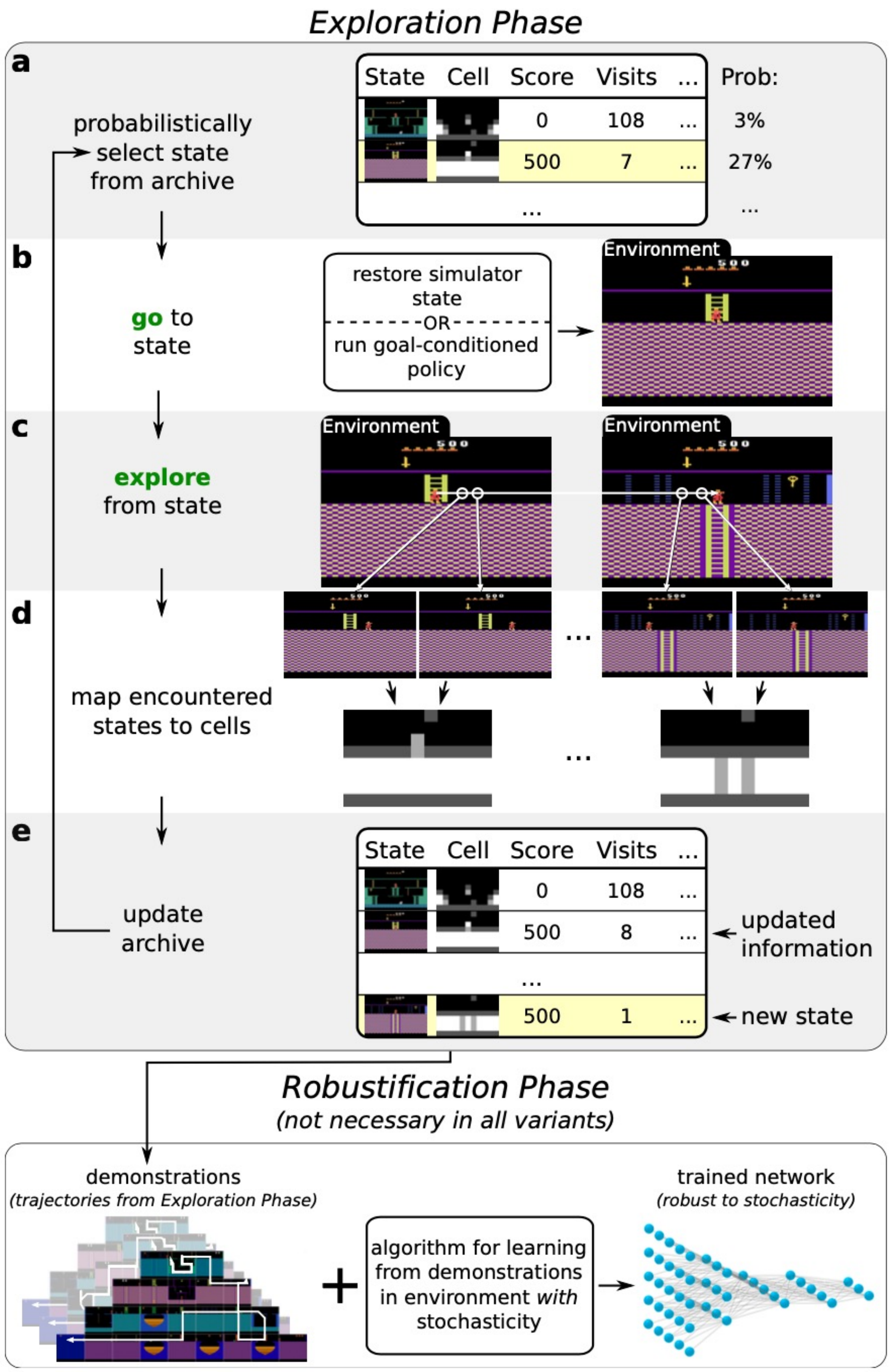
However, returning without a policy also means that this exploration process, which we call the exploration phase, does not produce a policy robust to the inherent stochasticity of the real world.

After the exploration phase is complete, these trajectories make it possible to train a robust and high-performing policy by Learning from Demonstrations (LfD).

How to return to a state?



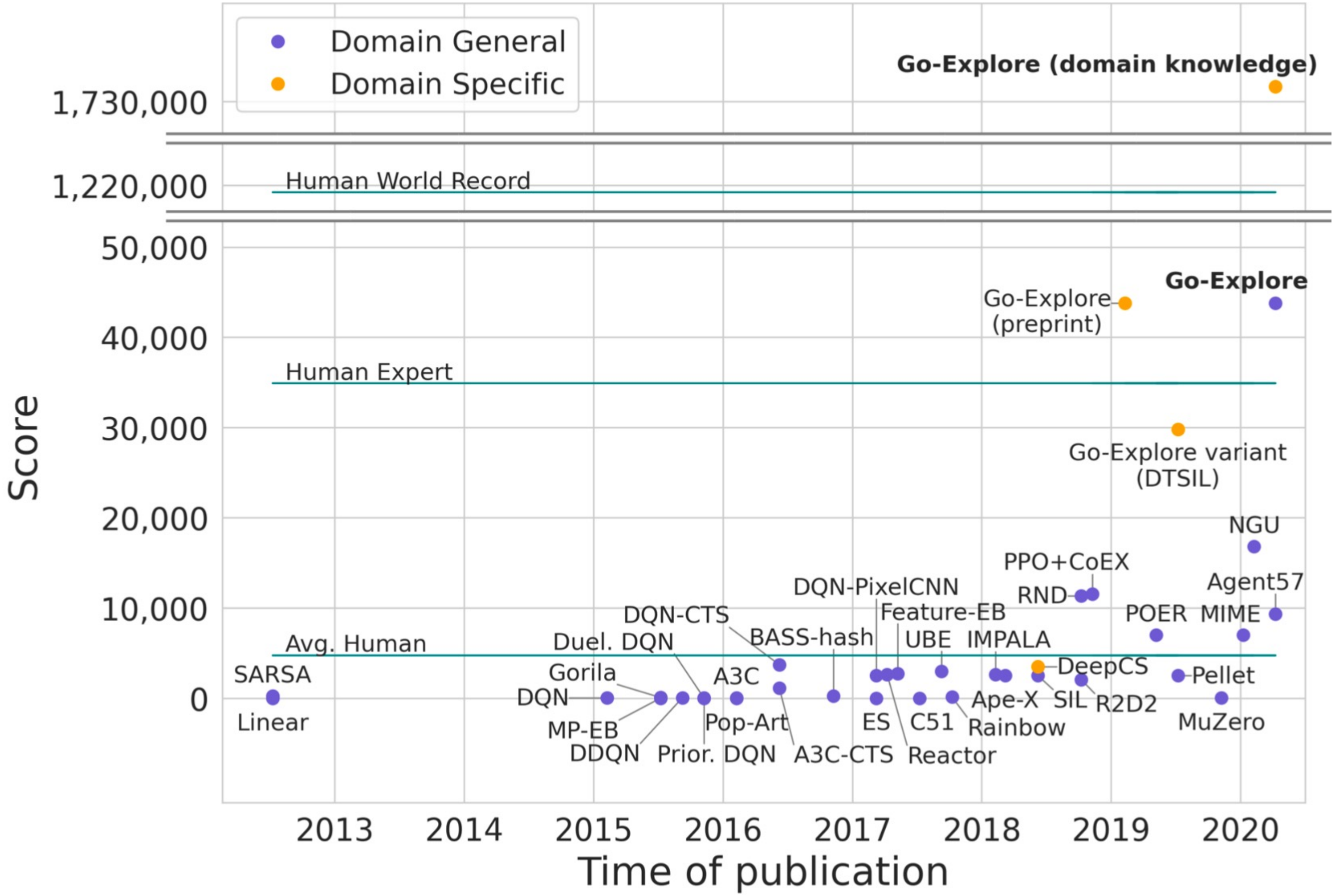
All in one view



Results

Game	Exploration Phase	Robustification Phase	SOTA	Avg. Human
Berzerk	131,216	**197,376**	1,383	2,630
Bowling	247	**260**	69	160
Centipede	613,815	**1,422,628**	10,166	12,017
Freeway	34	**34**	**34**	30
Gravitar	13,385	**7,588**	3,906	3,351
MontezumaRevenge	24,758	**43,791**	11,618	4,753
Pitfall	6,945	**6,954**	0	6,463
PrivateEye	60,529	**95,756**	26,364	69,571
Skiing	-4,242	**-3,660**	-10,386	-4,336
Solaris	20,306	**19,671**	3,282	12,326
Venture	3,074	**2,281**	1,916	1,187

Results


Domain General
Domain Specific
Go-Explore (domain knowledge)
1,730,000
1,220,000
Human World Record
50,000
40,000
30,000
20,000
10,000
0
Score
Go-Explore
Go-Explore (preprint)
Human Expert
Go-Explore variant (DTSIL)
NGU
PPO+CoEX
RND
Agent57
DQN-PixelCNN
Feature-EB
DQN-CTS
POER
MIME
BASS-hash
UBE
IMPALA
Avg. Human
Duel. DQN
DeepCS
SARSA
Gorila
A3C
Pellet
DQN
Ape-X
SIL
R2D2
Linear
MP-EB
Pop-Art
ES
C51
Rainbow
MuZero
DDQN
Prior. DQN
A3C-CTS
Reactor
2013
2014
2015
2016
2017
2018
2019
2020
Time of publication