Exploration in RL

Deep Reinforcement Learning

Mohammad Hossein Rohban, Ph.D.

Courtesy: Some slides are adopted from CS 285 Berkeley, and CS 234 Stanford, and Pieter Abbeel's compact series on RL.

RIML Lab, CE Department, Sharif University of Technology

Spring 2026

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 $(1-\epsilon)^{O(k)}\epsilon^{O(1)}$.
  - 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.

$$Reg(T) = T\underbrace{E[r(a^*)]}_{\text{Expected Reward of best action}} - \sum_{t=1}^{T} \underbrace{r(a_t)}_{\text{Actual reward of action actually taken}}$$

Bandits

- 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

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:

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

Optimistic exploration

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

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

optimistic estimate: $a = \arg max \hat{\mu}_a + C\sigma_a$) σ_a is small some sort of variance estimate)

✦ intuition: try each arm until you are <u>sure</u> it's not great

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

bonus

Q

r

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

($N(a)$ is number of times we picked this action)

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

Probability matching/posterior sampling

- 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."

```
assume r(a_i) ~ p_θi(r_i)
this defines a POMDP with s = [θ_1, …, θ_n]
belief state is p̂(θ_1, …, θ_n)          this is a model of our bandit

idea: sample θ_1, …, θ_n ~ p̂(θ_1, …, θ_n)

pretend the model θ_1, …, θ_n is correct

take the optimal action

update the model
```

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 $\hat{p}(\theta_1, \dots, \theta_n)$ — 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

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 \quad (e.g., y might be $r(a)$)
  - the lower the entropy, the more precisely we know z
  - $$IG(z, y) = E_y[\mathcal{H}(\hat{p}(z)) - \mathcal{H}(\hat{p}(z)|y)]$$
  - typically depends on action, so we have $IG(z, y|a)$

Information gain example

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

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 $p(r_a)$)

$g(a) = IG(\theta_a, r_a|a)$ -- information gain of a

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

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

General themes

- UCB:

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

- Thompson sampling:

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

$$a = \underset{a}{\text{argmax}} E_{\theta_a}[r(a)]$$

- Info gain:

$$IG(z, y|a)$$

- 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

Stochastic Process

$P(Q)$

$= P(Q(s_1, a_1), Q(s_2, a_2) \dots)$

$Q(a)$

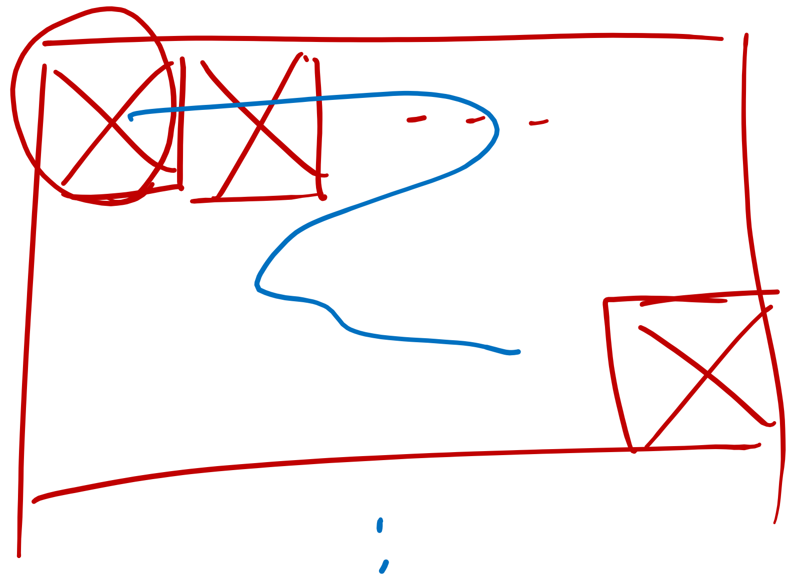
$Q(\underline{a_1})\ Q(\underline{a_2}) \cdots Q(\underline{a_k})$

$\underline{\forall s,a\ Q(s,a)}$

Optimistic exploration in RL

$+c\sqrt{\frac{1}{n(s,a)}}$

Q-table



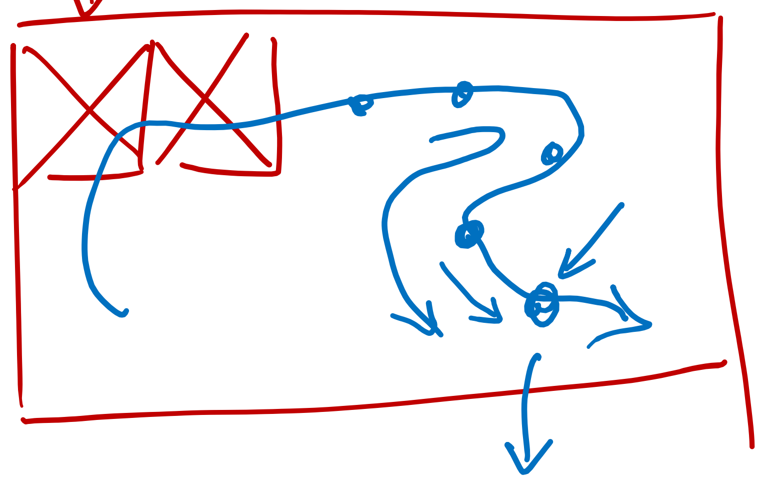
Scenario 1

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

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

- can we use this idea with MDPs?

$r + c\sqrt{\frac{1}{N(s,a)}}$



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

$B(\circ)$

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

Here, B 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

$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r(s,a) + \gamma \max_{a'} Q(s',a')\right]$

$+ c\sqrt{\frac{1}{N(s,a)}}$

Q

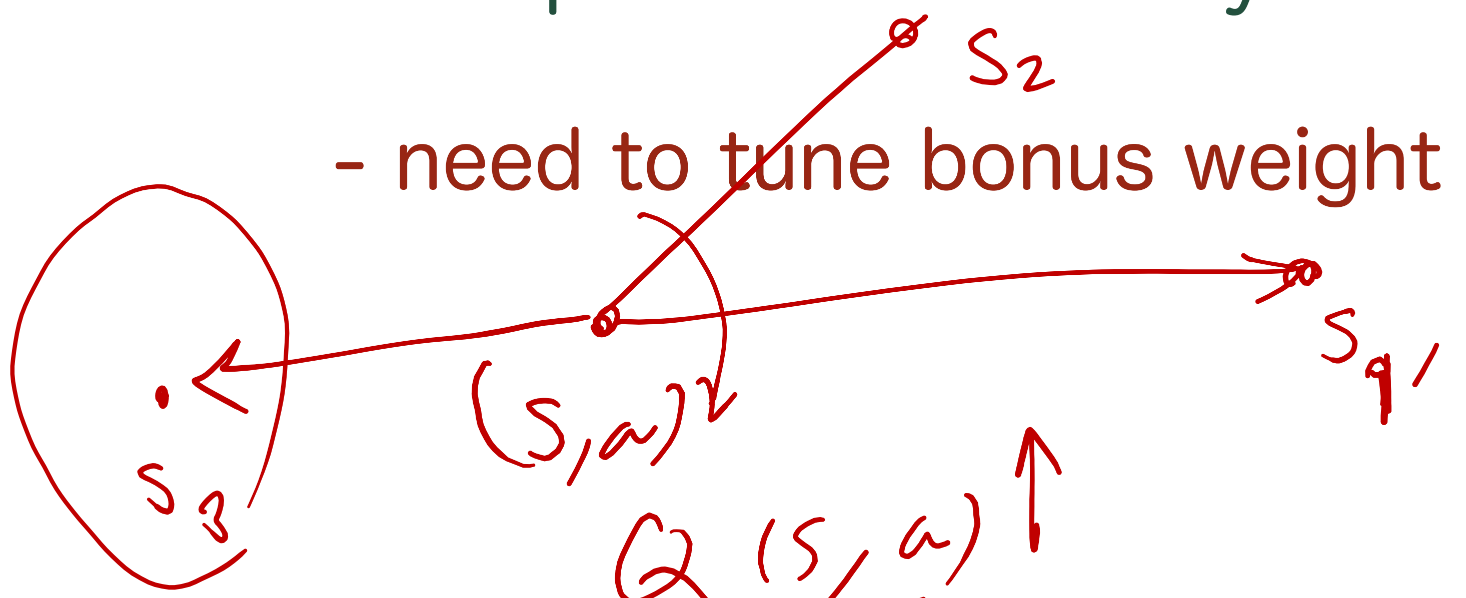
$B(S_2)\downarrow$

+ simple addition to any RL algorithm

- need to tune bonus weight

$r=0$

$B(S_1)\uparrow$



$Q(s,a)\uparrow$

13

The trouble with counts

$$\hat{Q}(a) + \overbrace{\sqrt{\frac{\log T}{N(a)+\epsilon}}}^{\text{bonus}}$$

$\hat{Q}(a)$: return

T: # time steps passed

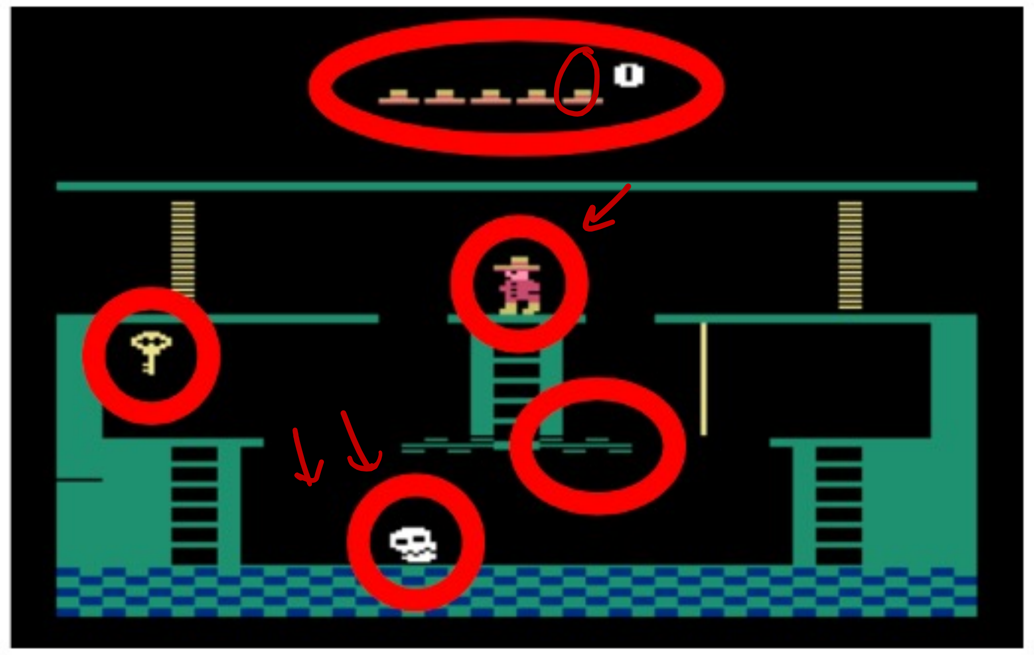
$N(a)$: # action a was explored

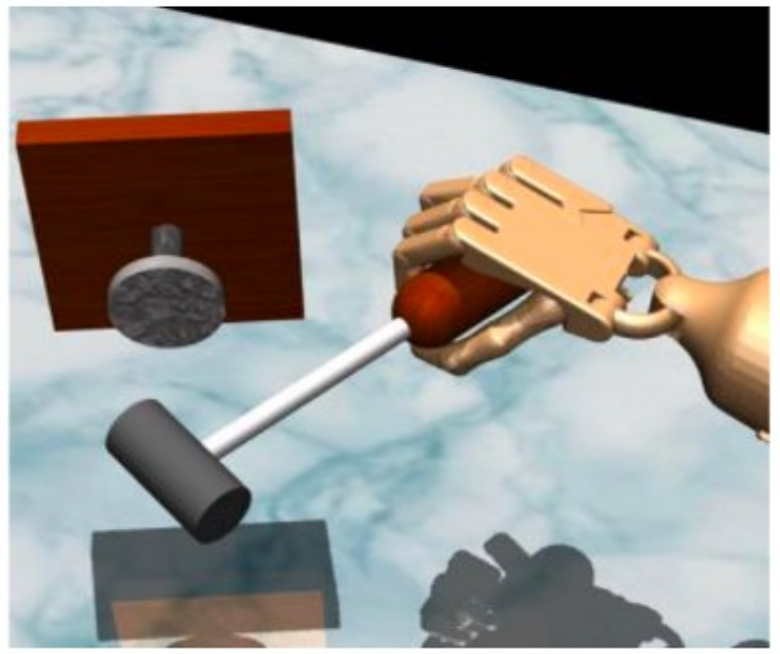
Use $r^+(s,a) = r(s,a) + \mathcal{B}(N(s))$

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

$N(s)$: # times that state s was visited.

But wait… what's a count?





full RL: bonus should be added to all intermediate states in the trajectory leading to s, a

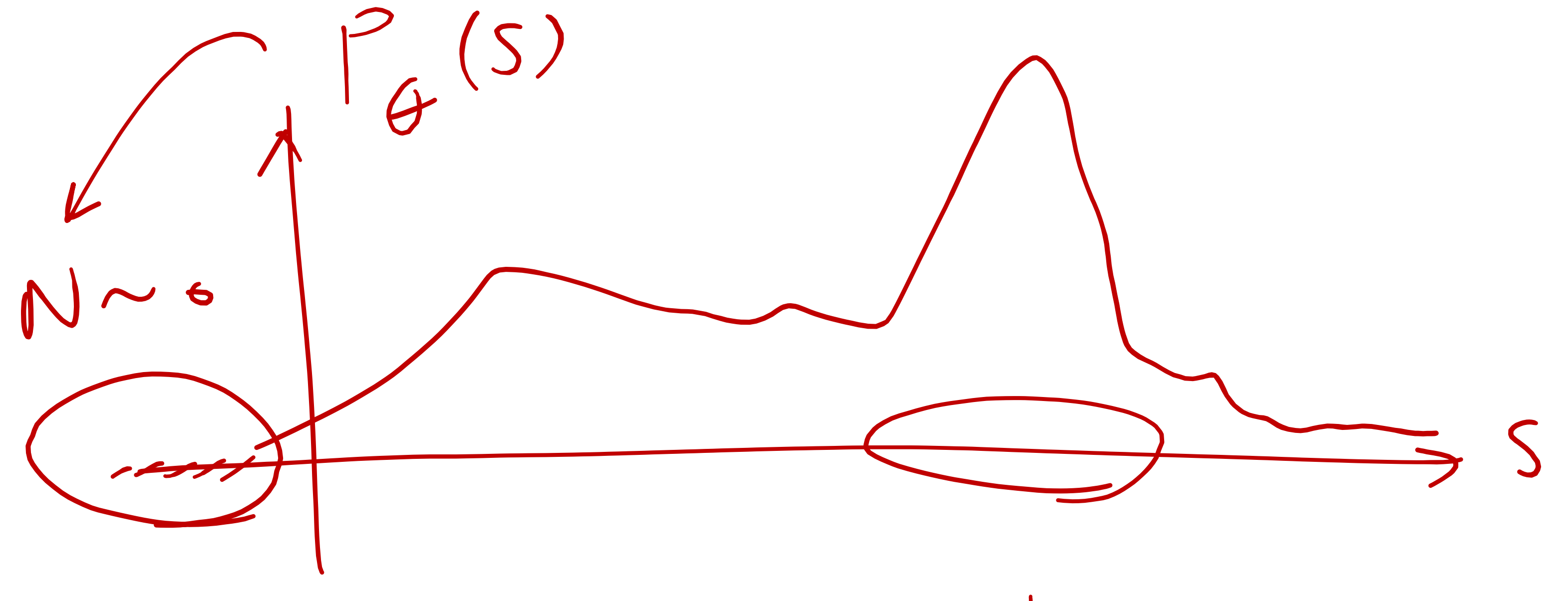
$$\hat{Q}(s_{n-1}, a_{n-1}) \leftarrow s, a \quad \hat{Q}(s,a) \quad s \quad \underbrace{s_{n+1}, a_{n+1}, \dots}$$

$$r(s_{n-1}, a_{n-1}) + \gamma \max_{a_n} \hat{Q}(s_n, a_n)$$

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 $\mathbf{s}$
- 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 after we see $\mathbf{s}$, we have:
  - the true probability is:

Knowns: P, P'

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

  - After we see s, we have:

$D = \{s_0, s_1, \dots, s_n\}$

$D \cup \{s\}$

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

Pseudo-count

$N(s)$ $\quad$ $N(s) \leftarrow N(s)+1$

n $\quad$ $n+1$

$1 \longrightarrow s$

n $\quad$ $n+1$

RL alg.

$P(s)$

$P'(s)$
(updated $P(s)$
considering $\{s\}$
added to D)

Exploring with pseudo-counts

① fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far
take a step i and observe $\mathbf{s}_i$
③ 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 $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$ ($(\hat{N}(\mathbf{s}))$ is pseudo-count)

RL training step

retraining on $\mathcal{D} \cup \{s_i\}$

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

reward to go

$\rightarrow Q(s,a)$

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

two equations and two unknowns!

$\mathcal{D} = \{s_1 \cdots s_{n-1}\}$

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

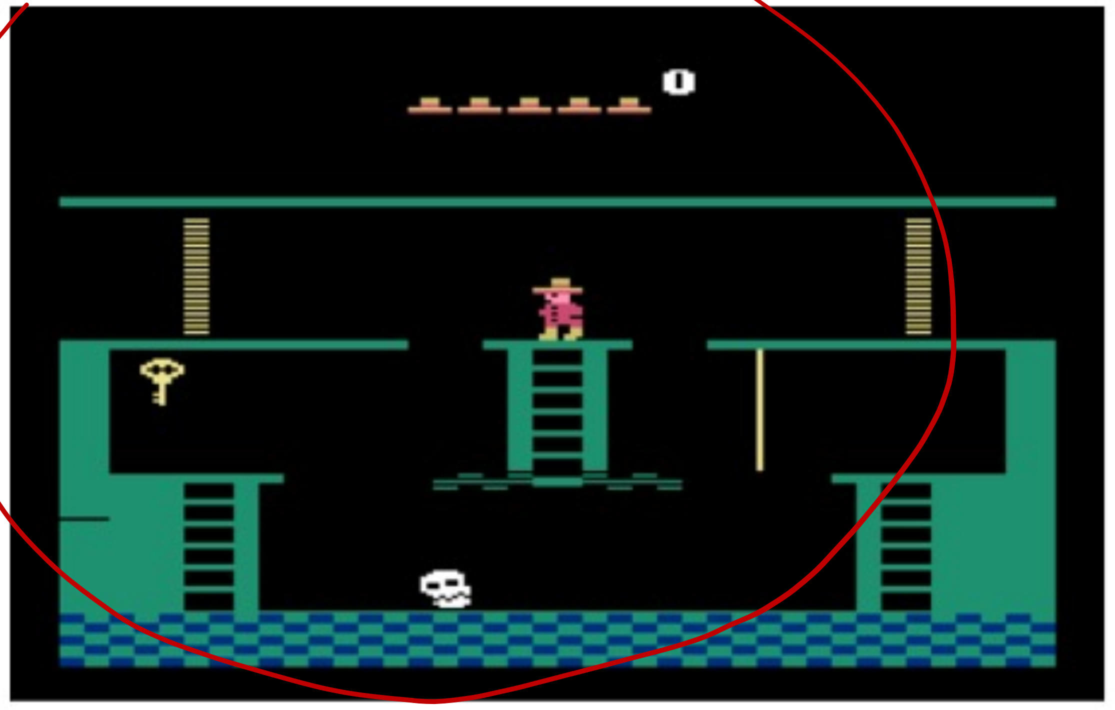
What kind of generative modeling to use?

$$P(s^{(i)} \mid s^{(1)} \dots s^{(i-1)}) := P(s^{(i)} \mid \text{neighbors of } s^{(i)})$$

s_i → θ → log

Neigh(s_i) $P(s^{(i)} \mid$

Neigh(s_i)



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

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

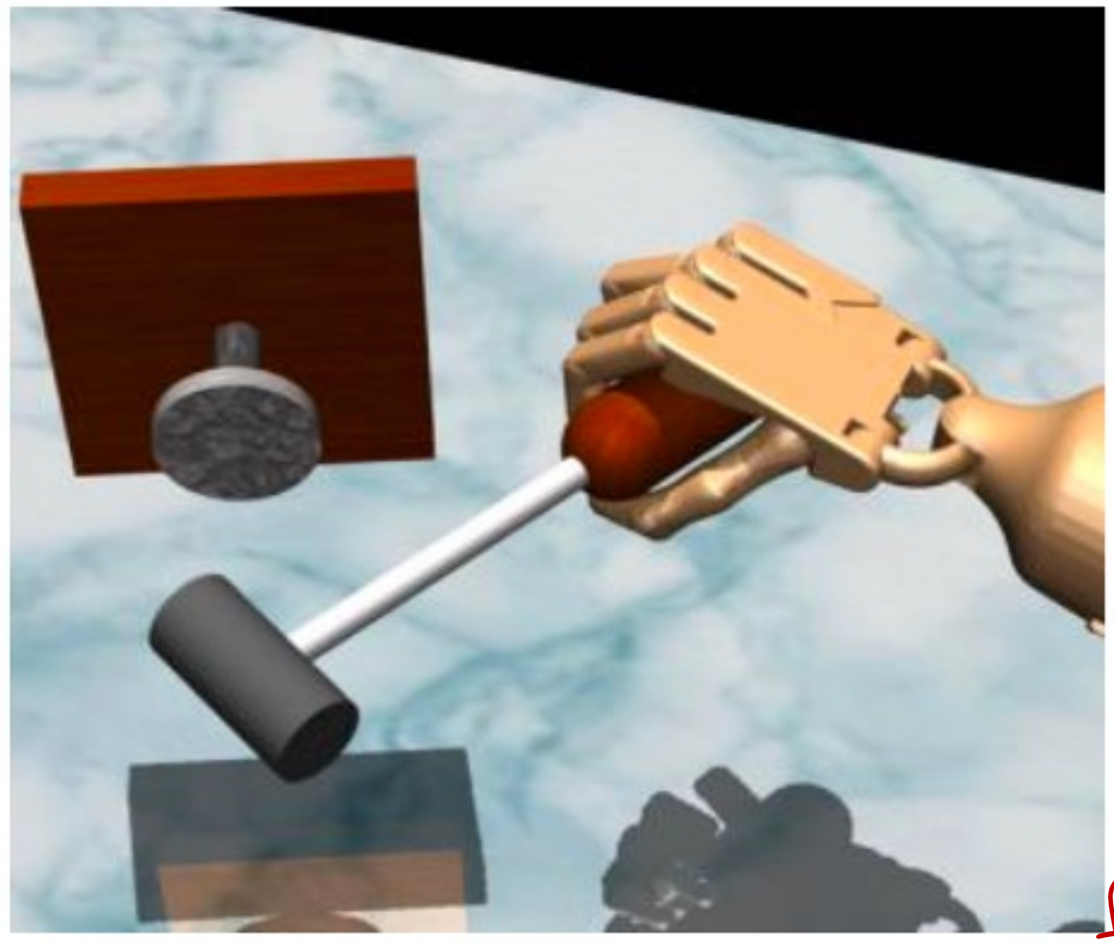
opposite considerations from many popular generative models in the literature (e.g., GANs)

(S)

Image

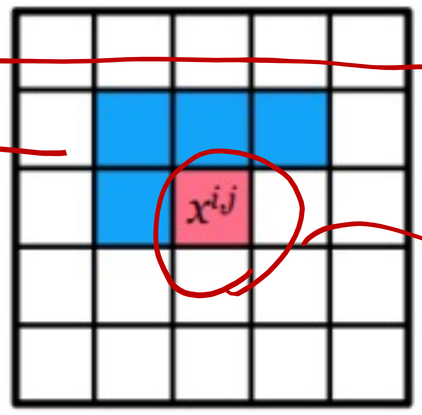
elements of s.

$s = (s^{(1)} \dots s^{(d)})$



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

$s^{(1)}$ $s^{(2)}$



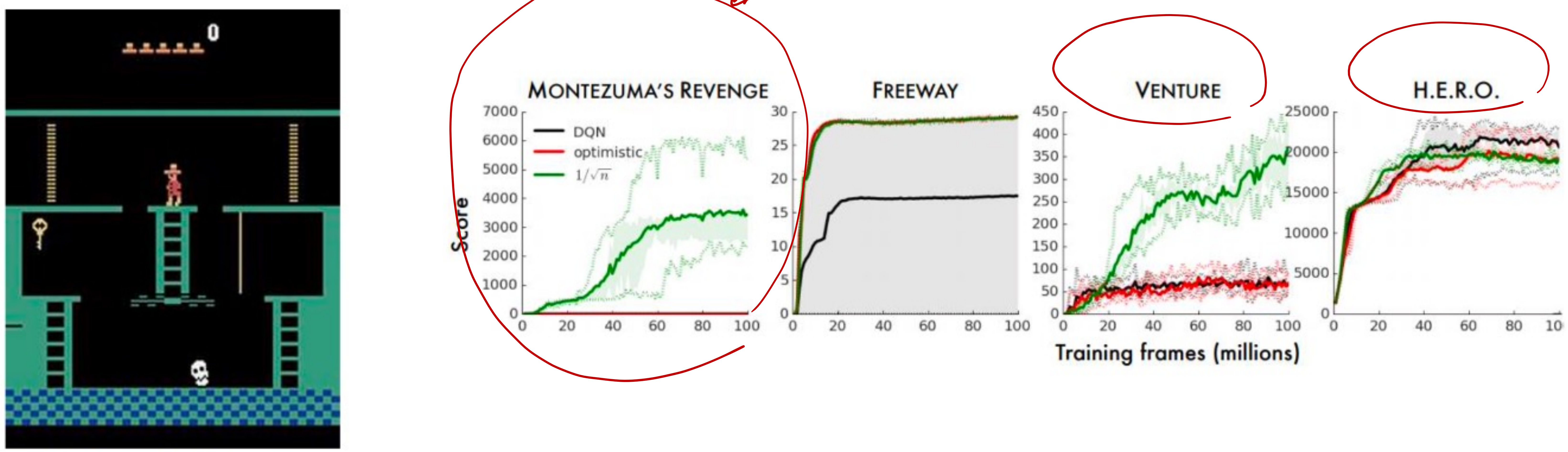
$s^{(i)}$

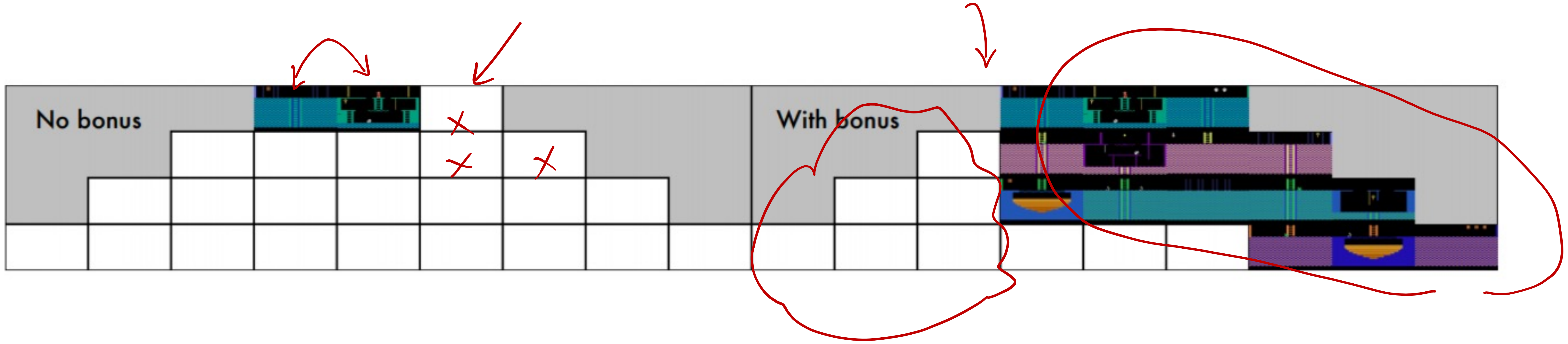
Other models: stochastic neural networks, compression length, EX2

d-1 d

$$\log P(s^{(1)} \dots s^{(d)}) = \log \underbrace{P_\theta(s^{(1)})}_{f(s_1)} + \log \underbrace{P_\theta(s^{(2)} \mid s^{(1)})}_{f(s_2, s_1)} + \log P_\theta(s^{(3)} \mid s^{(2)}, s^{(1)}) \dots$$

Does it work?





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

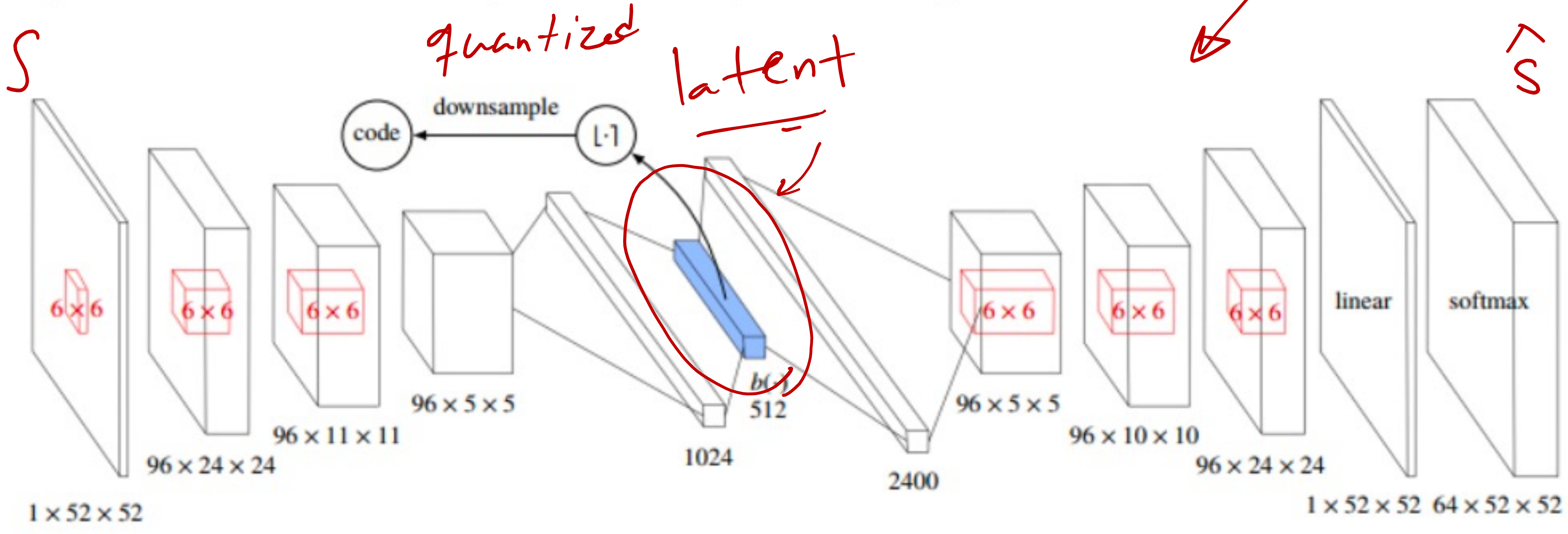
Counting with hashes

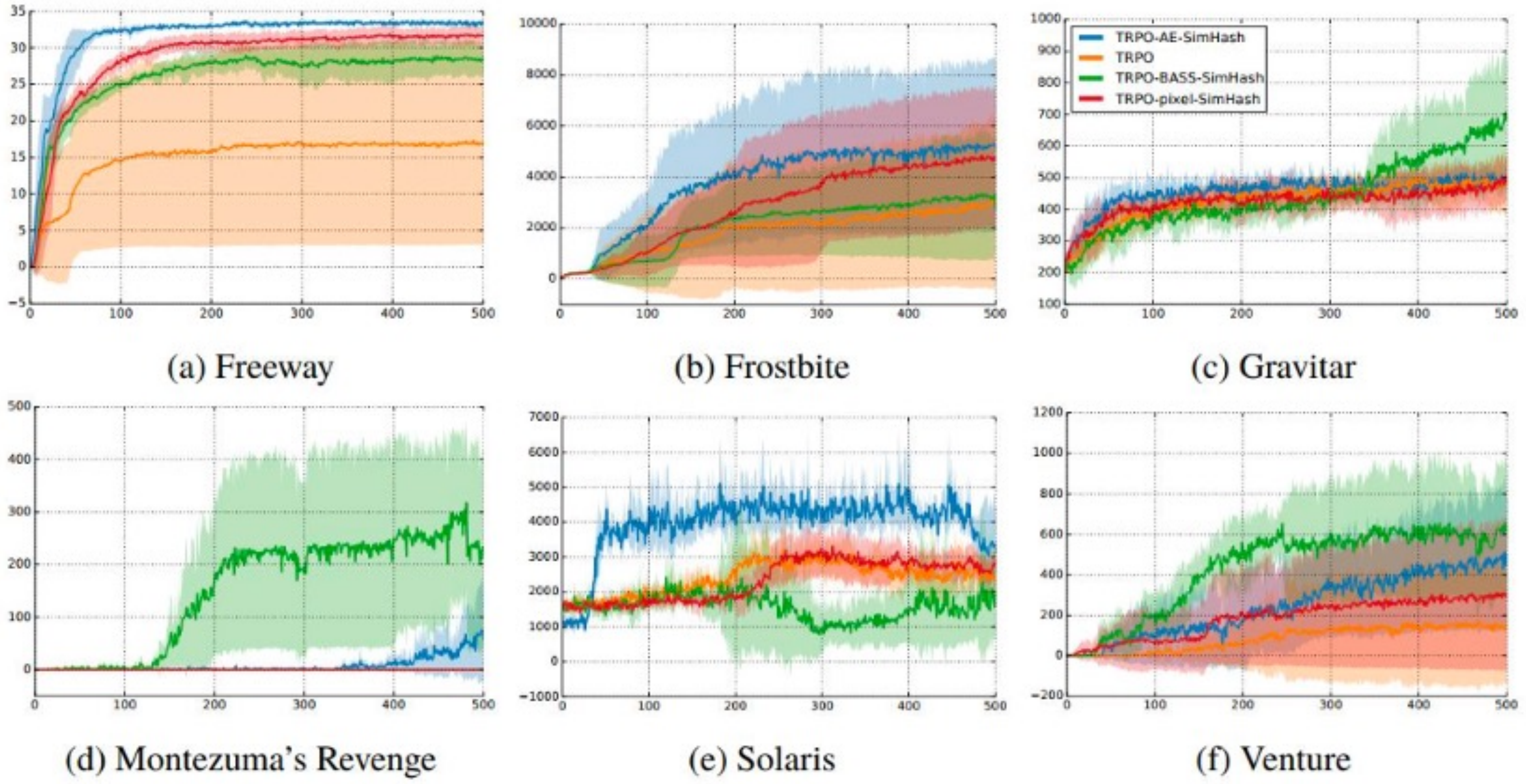
<u>What if we still count states, but in a different space?</u>

- 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:

reconstruction



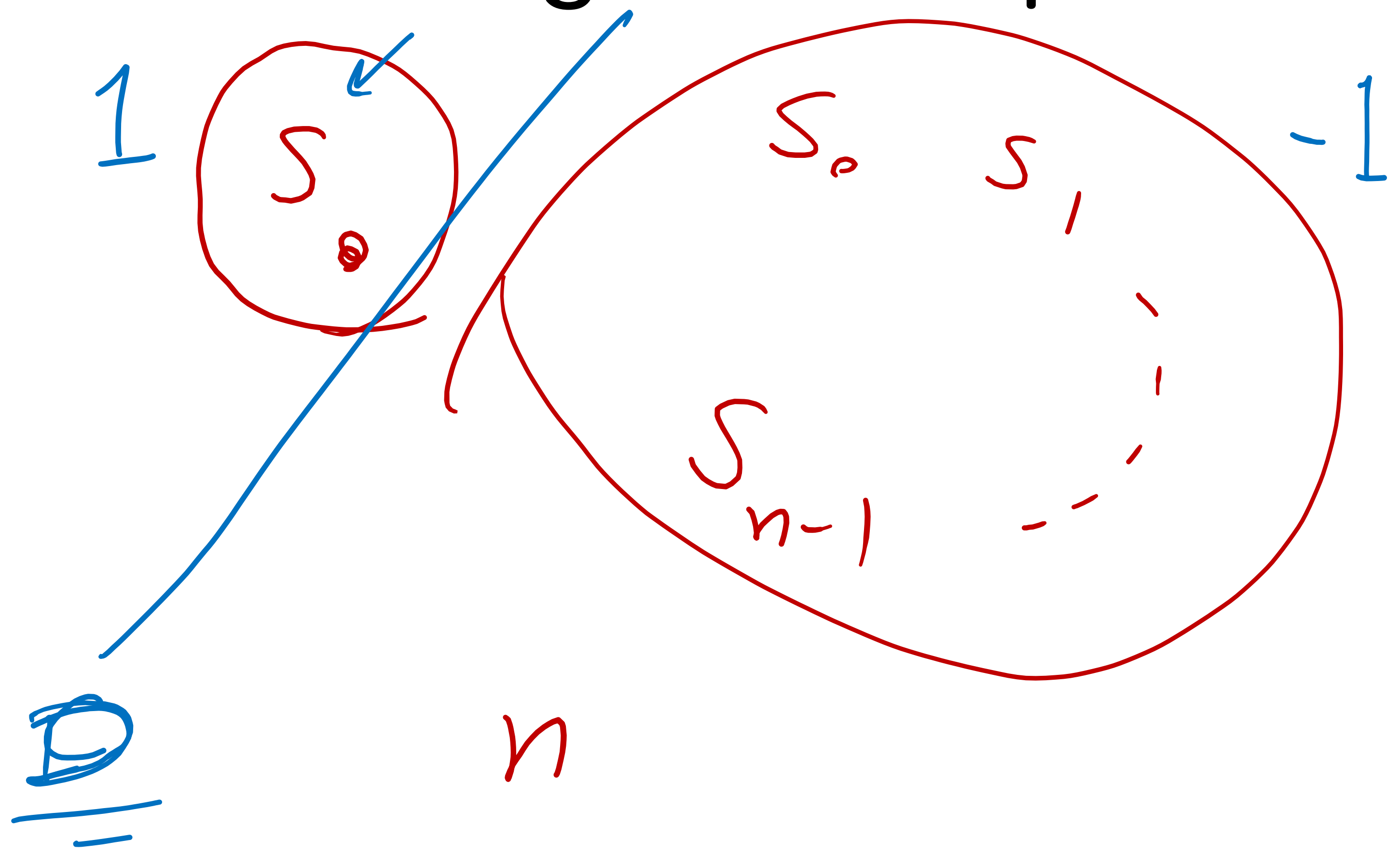


Tang et al. "#Exploration: A Study of Count-Based Exploration"

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

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



Implicit density modeling with exemplar models (Cont.)

Previous states

$$D_{x^\star} = \underset{D \in \mathcal{D}}{\arg max} \left(E_{\delta_{x^\star}}[\log D(x)] + E_{P_\mathcal{X}}[\log(1 - D(x))] \right). \text{ (1)} \longrightarrow \frac{\partial L}{\partial D_{x^\star}}$$

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

$$D_{x^\star}(x) = \frac{\delta_{x^\star}(x)}{\delta_{x^\star}(x) + P_\mathcal{X}(x)} \; and \; D_{x^\star}(x^\star) = \frac{1}{1 + P_\mathcal{X}(x^\star)}.$$

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 $D(x)$.

$$\frac{1}{D(x^\star)} \cdot 1 - \frac{1}{1 - D(x^\star)} \cdot P(x^\star) = 0$$

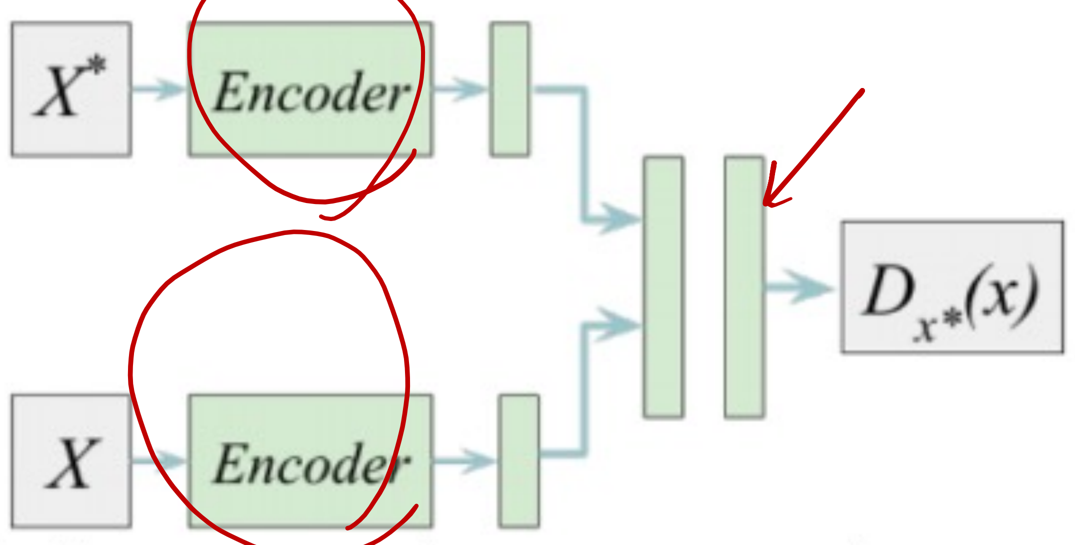
Implicit density modeling with exemplar models (Cont.)

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:

$$D^{\star}_{\mathbf{s}}(\mathbf{s}) = \frac{1}{1+p(\mathbf{s})} \Longrightarrow p_{\theta}(\mathbf{s}) = \frac{1-D_{\mathbf{s}}(\mathbf{s})}{D_{\mathbf{s}}(\mathbf{s})}$$

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!



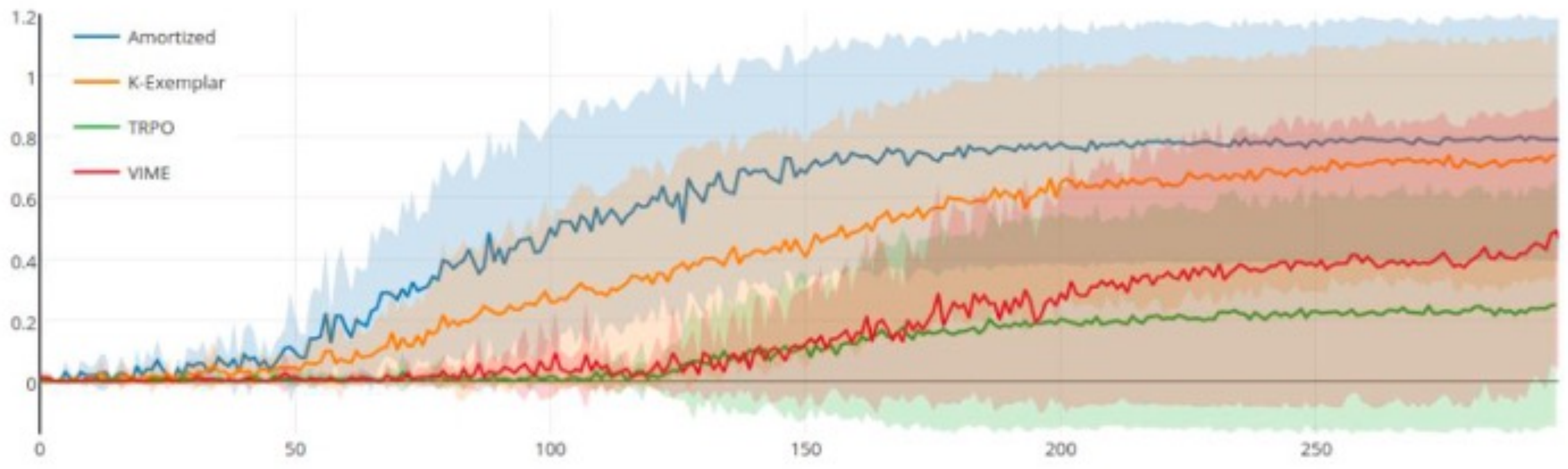


Figure 9: DoomMyWayHome+

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

Heuristic estimation of counts via errors

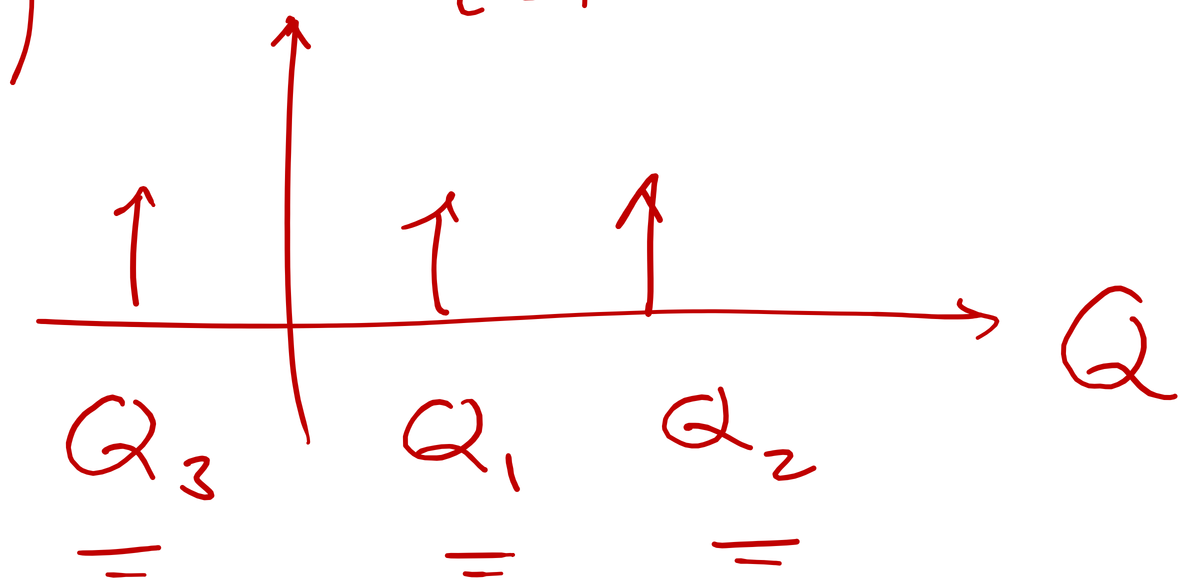
$p_\theta(\mathbf{s})$, need to be able to output densities, but doesn't necessarily need to produce great samples

...and doesn't even need to output great densities

...just need to tell if state is novel or not!

R.V.

$P(Q) = \sum_{i=1}^{k} \delta(Q - Q_i)$



$\neq P(Q_1 \cdots Q_k)$

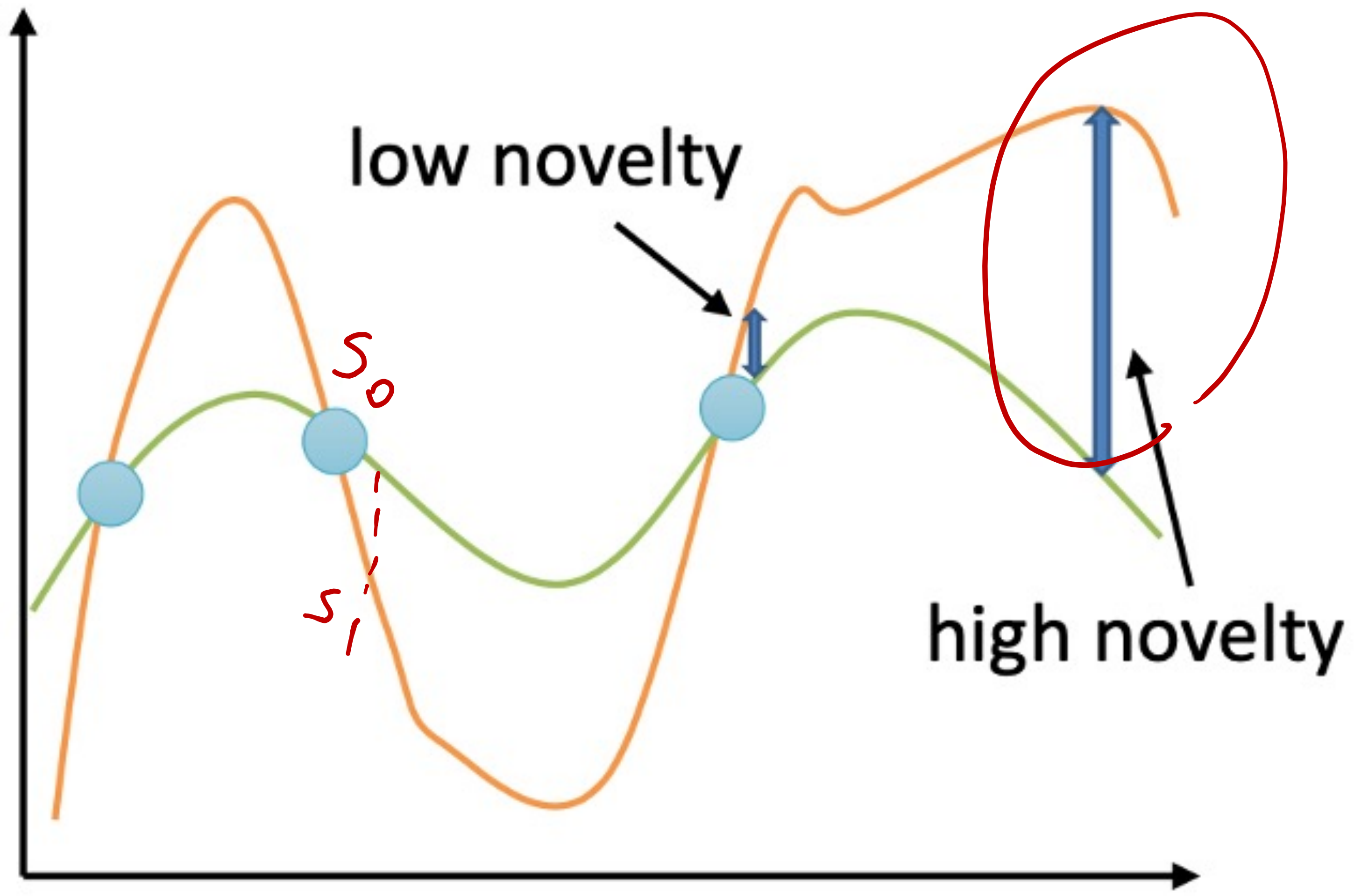
let's say we have some **target** function $f^\star(\mathbf{s}, \mathbf{a})$

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

NN

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

$\bar{r} = r + B(s, a)$



Heuristic estimation of counts via errors (Cont.)

- 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

Posterior sampling in deep RL

Thompson sampling:

$\theta_1, \dots, \theta_n \sim \hat{p}(\theta_1, \dots, \theta_n)$ What do we sample?

$a = \arg\max_a E_{\theta_a}[r(a)]$ How do we represent the distribution?

Bandit

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

$Q_1 \cdots Q_k$

MDP analog is the Q-function!

full RL

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?

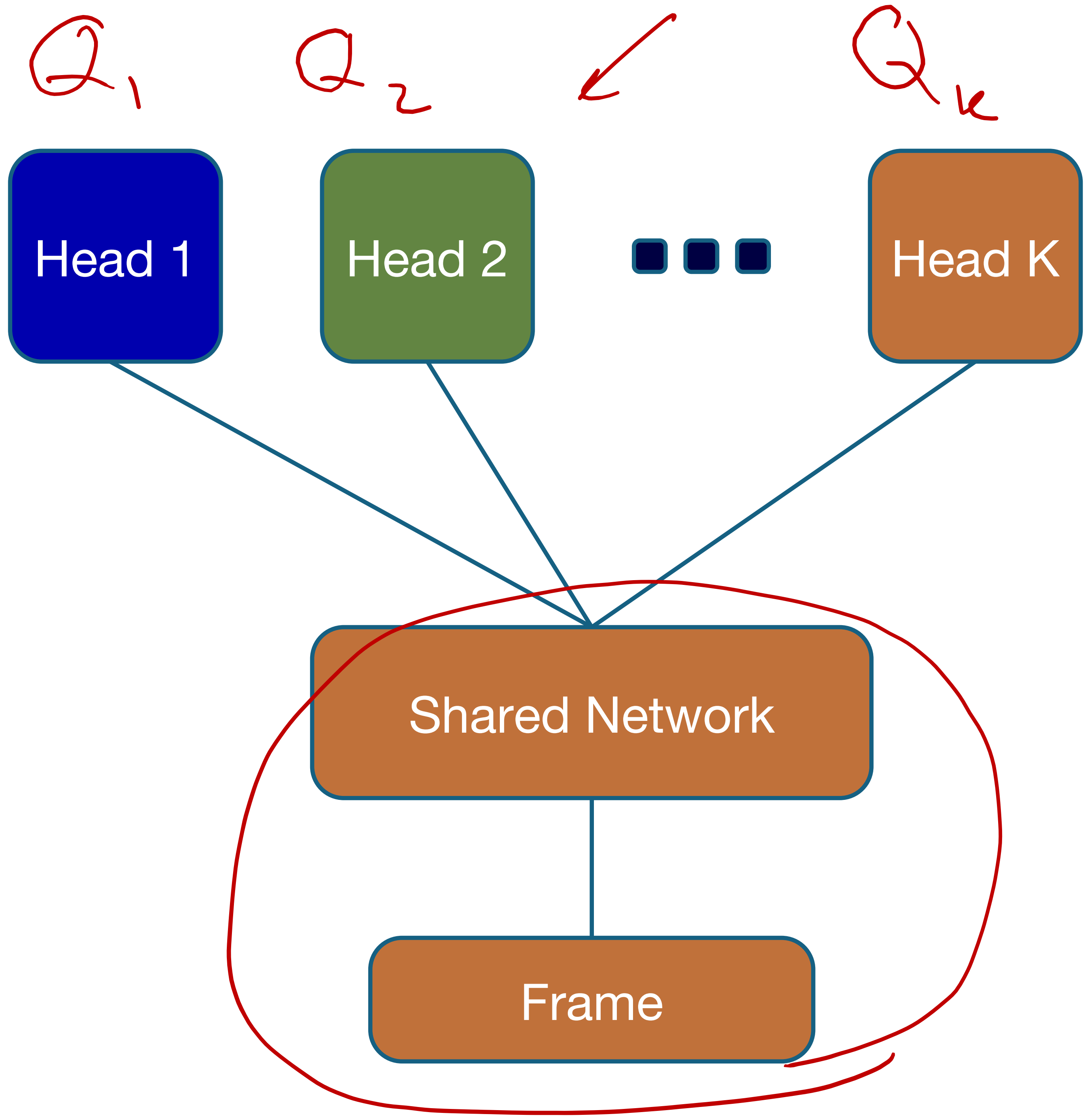
Bootstrap

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

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?



Bootstrap (Cont.)

Bootstrapped DQN

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

$s, a, s', r,$ tag

$1 \leq \quad \leq K$

fixed in episode k

a mini batch of samples

train Q_k based on samples with $m = k$

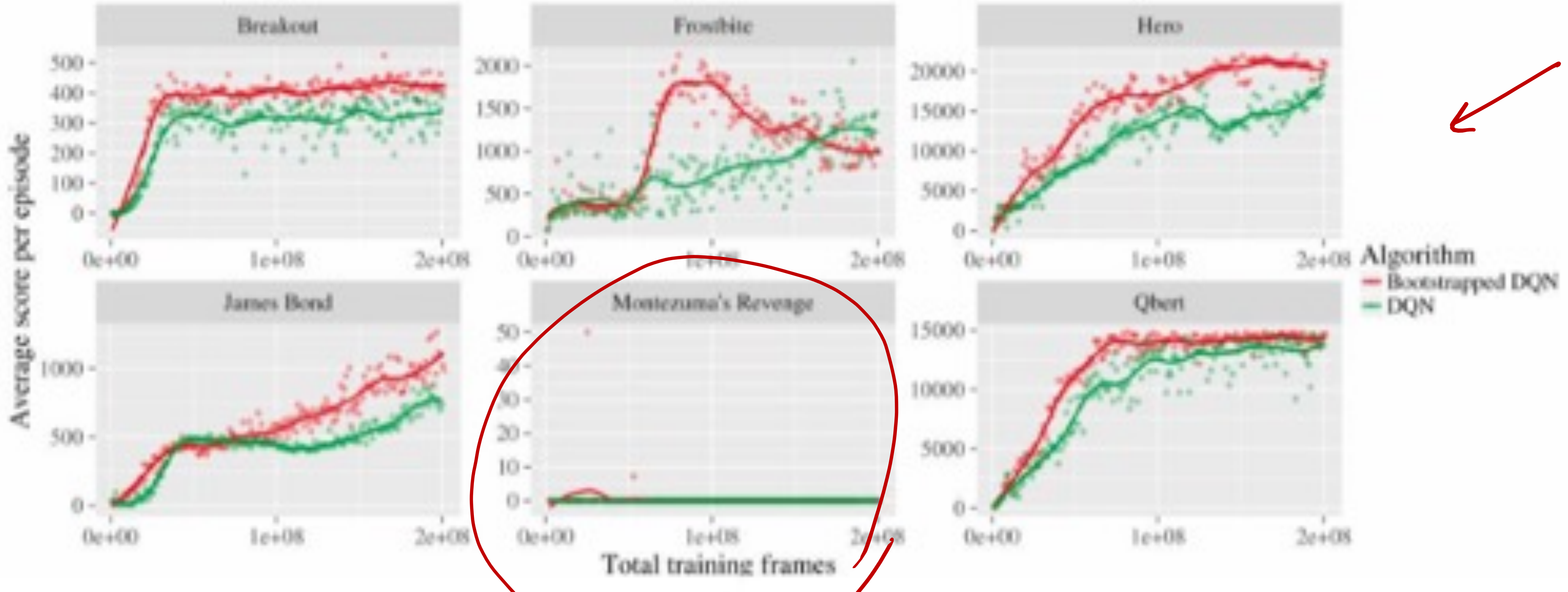
Why does this work?

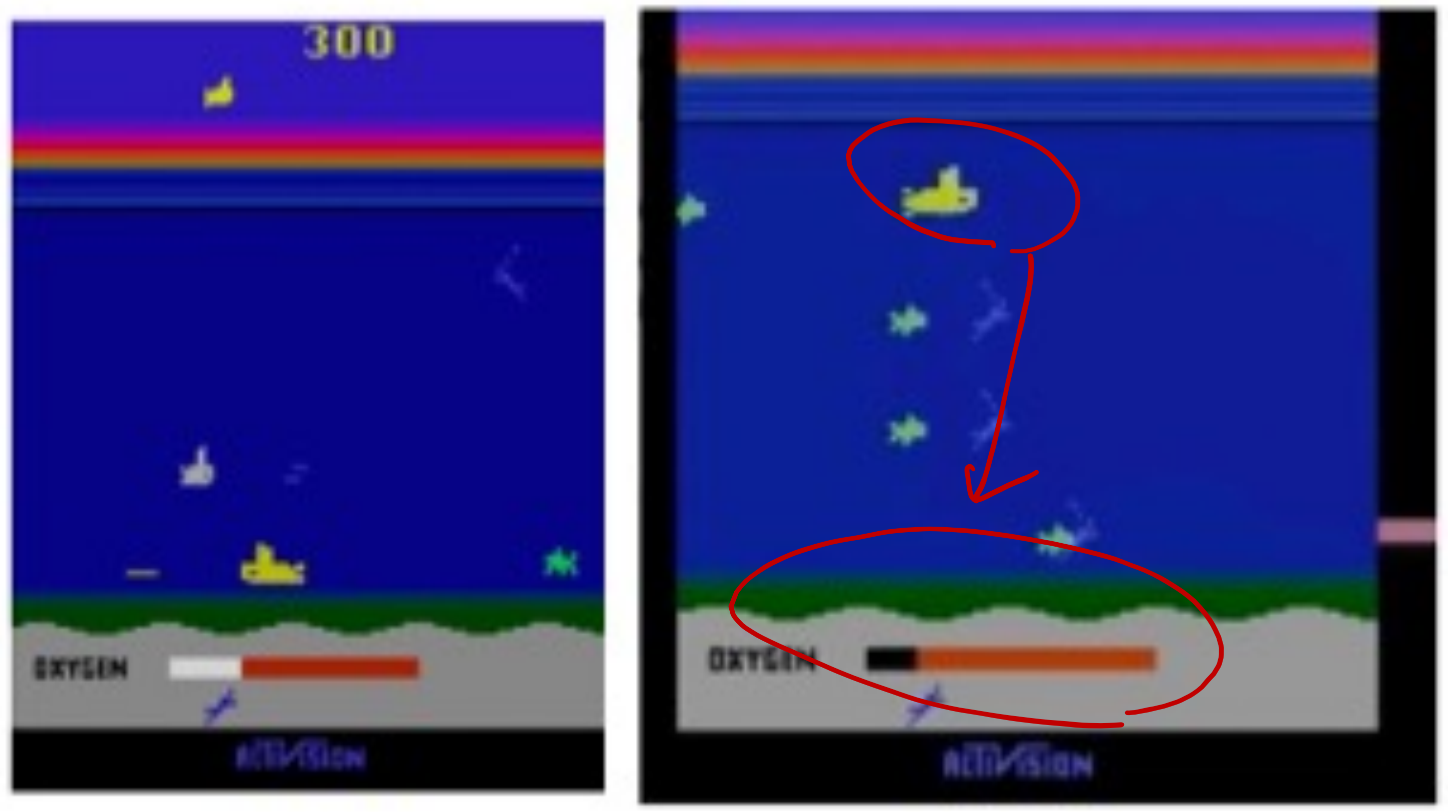
- Exploring with random actions (e.g., epsilon-greedy): oscillate back and forth, might not go to a coherent or interesting place
- Exploring with random Q-functions: commit to a randomized but internally consistent strategy for an entire episode

ϵ - greedy on Q-learning

+ 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*

I] 16 Sep 2021

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 (Cont.)

- Sparse Rewards: Requires a lot of exploration;
  - e.g. task: reaching coffee machine.
  - define reward as whether or not the machine has been reached.

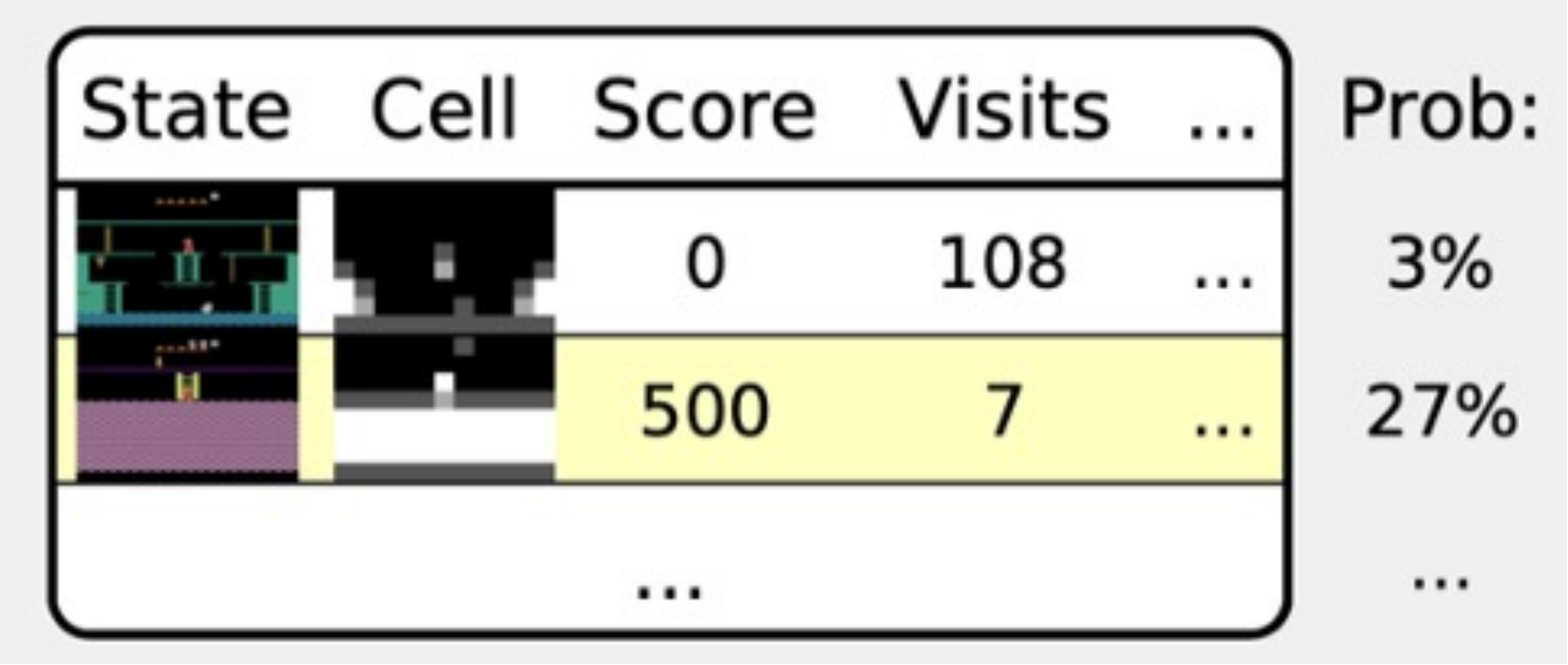
- Dense Rewards:
  - e.g. define reward as the inverse Euclidean distance to the machine.
  - Deceptive
  - Reward Hacking

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

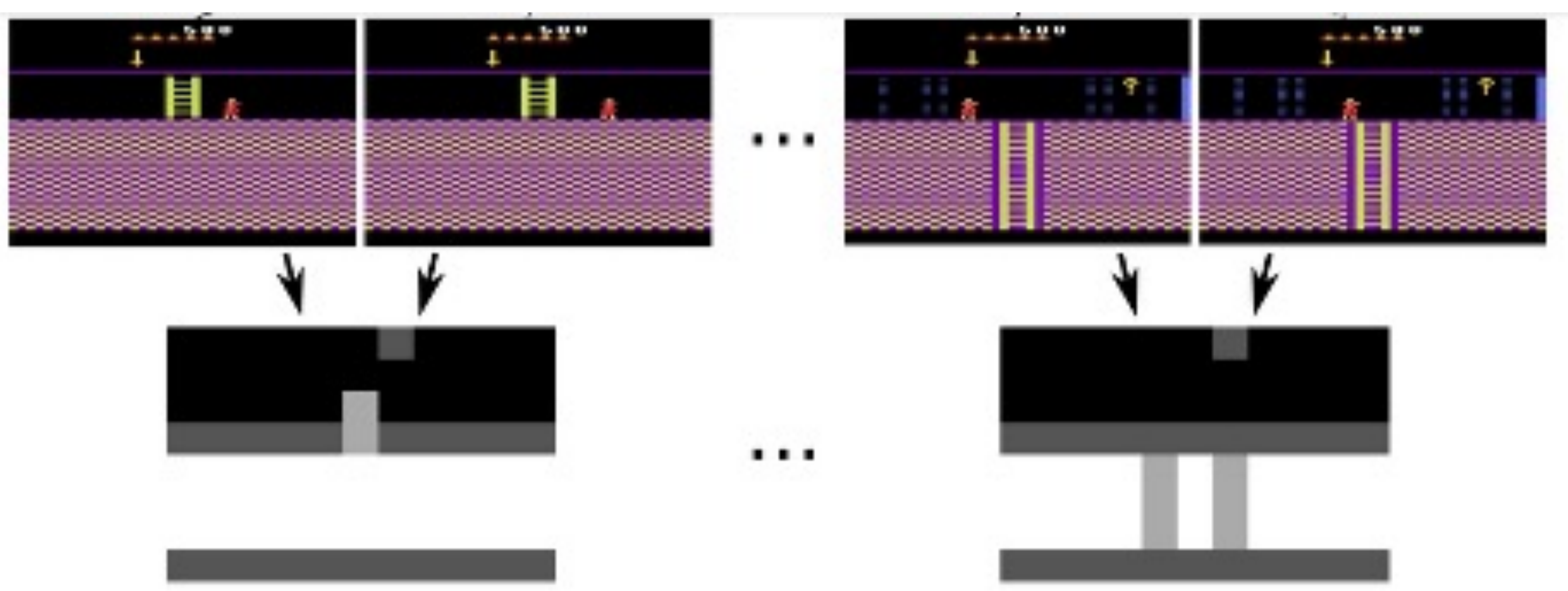
State Archive

- 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

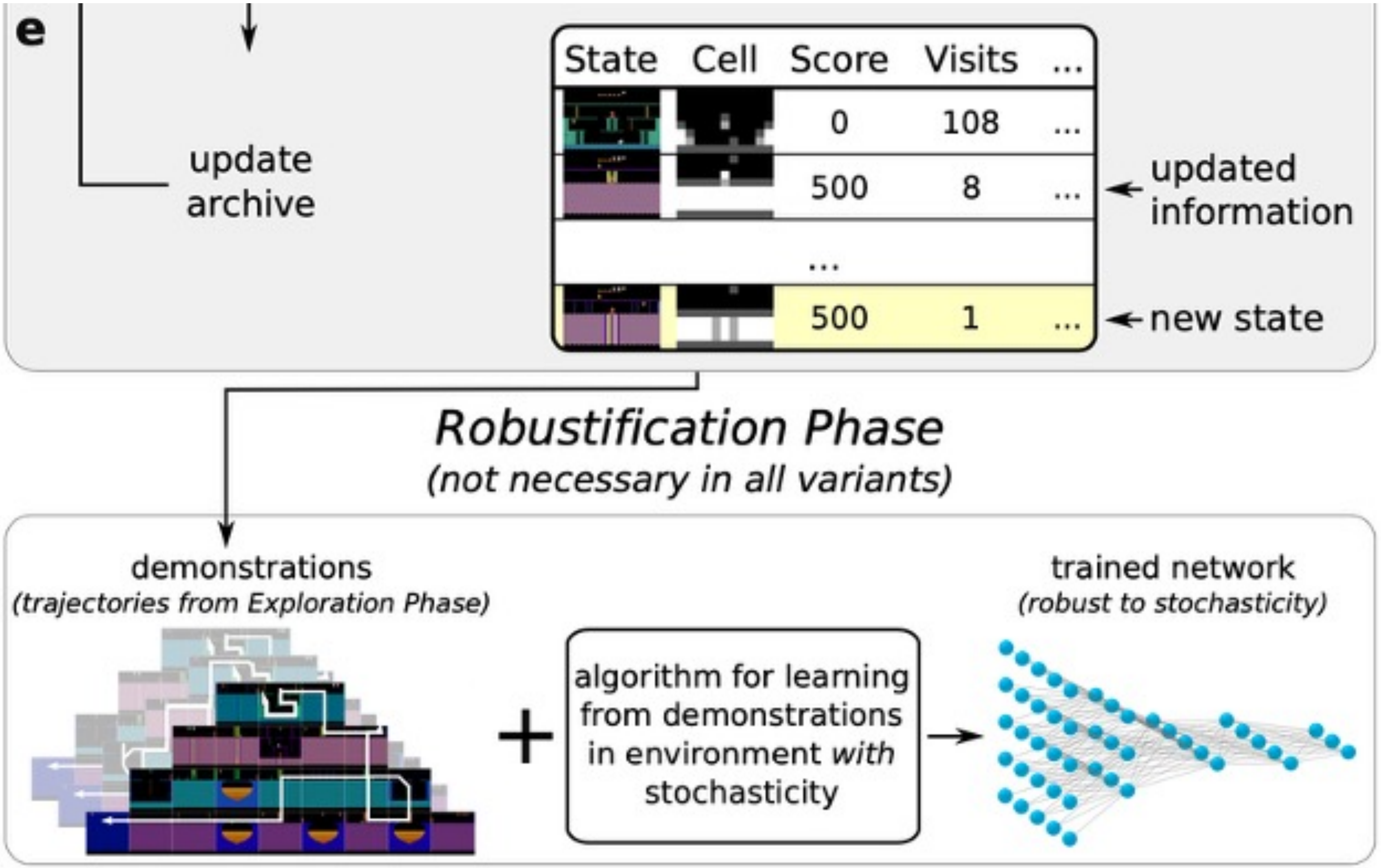
- Non-trivial environments have too many states to store explicitly.
- Similar states are grouped into cells.
- States are only considered novel if they are in a cell that does not yet exist in the archive.
- To maximize performance, if a state maps to an already known cell, but is associated with a better trajectory, that state and its associated trajectory will replace the state and trajectory currently associated with that cell.



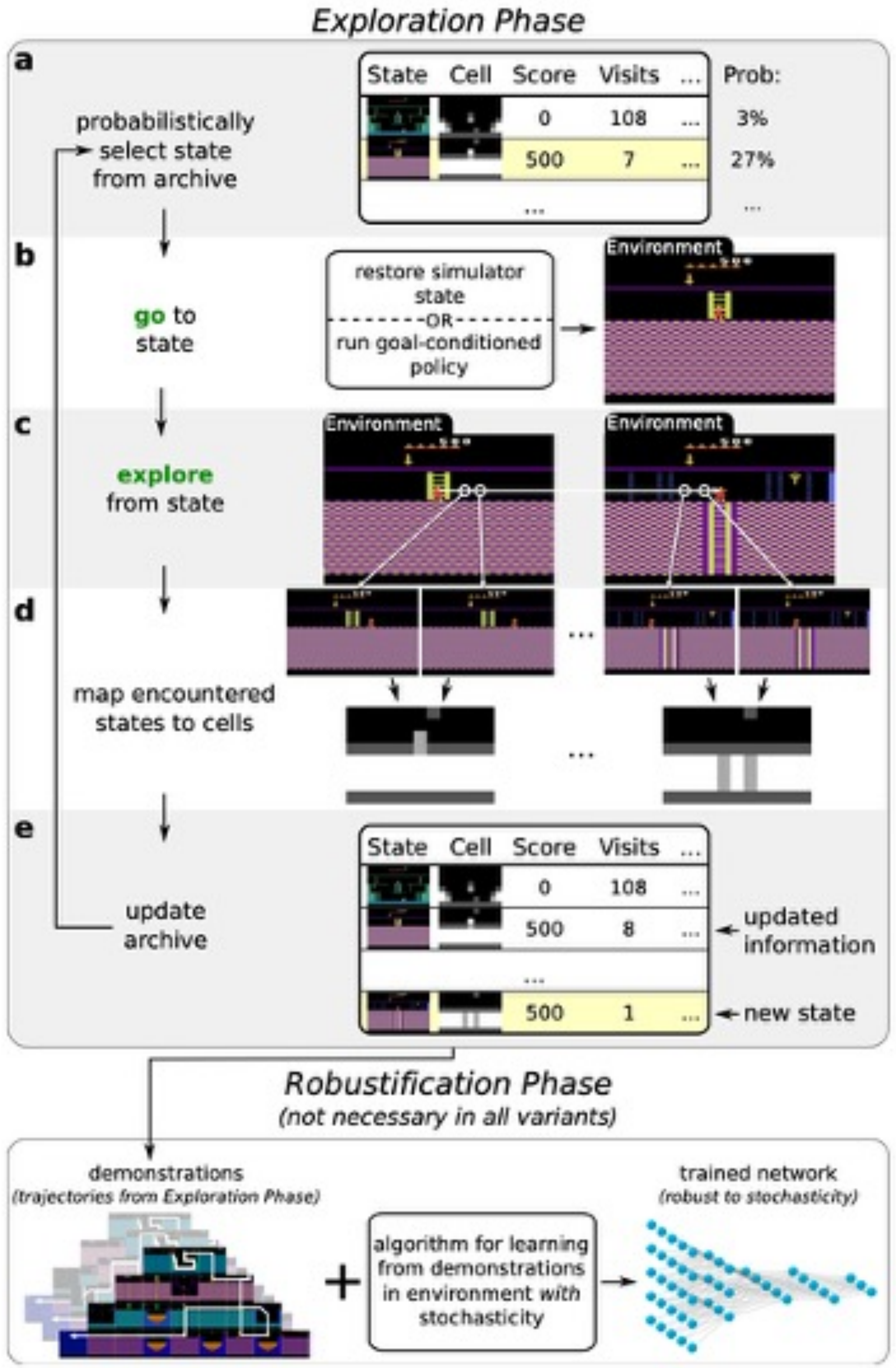
How to return to a state?

- 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? (Cont.)



All in one view



36

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 (Cont.)