

Deep Reinforcement Learning and Control

# Exploration by Curiosity and External Memory

Spring 2021, CMU 10-403

Katerina Fragkiadaki



# Exploration: It's all about modeling our uncertainty

- **Exploration**: trying out new things (new behaviours), with the hope of discovering higher rewards
- **Exploitation**: doing what **you know so far** will yield the highest reward

# Exploration with $\epsilon$ – greedy

- Are we exploring by adding random noise to our actions?
- Imagine what would happen if you go to a new museum and you try to explore its exhibits. What would be your strategy for efficient exploration?

# Exploration is all you need: no reward RL

- **Exploration**: trying out new things (new behaviors), with the hope of discovering higher rewards
- **We explore efficiently once we know what we do not know, and target our exploration to the unknown part of the space.**
- My goal is to maximize my learning progress, i.e., learn most about the world

# Motivation

Motivation: “Forces” that energize an organism to act and that direct its activity

- **Extrinsic** Motivation: being moved to do something because of some external reward (\$\$, a prize, etc.).
  - Problem: such rewards are sparse..
- **Intrinsic** Motivation: being moved to do something because it is inherently enjoyable (curiosity, exploration, novelty, surprise, incongruity, complexity...)
  - Gain: Task independent! Free of human supervision, no need to code up reward functions to incentivize the agent. A general loss functions that drives learning.



# Curiosity VS Survival

“As knowledge accumulated about the conditions that govern exploratory behavior and about how quickly it appears after birth, it seemed less and *less likely that this behavior could be a derivative of hunger, thirst, sexual appetite, pain, fear of pain, and the like*, or that stimuli sought through exploration are welcomed because they have previously accompanied satisfaction of these drives.”

D. E. Berlyne, *Curiosity and Exploration*, Science, 1966

Intrinsic Motivation different than Intrinsic Necessity: being moved to do something because it is necessary (eat, drink, find shelter from rain...)

# Curiosity and Never-ending Learning

Why should we care?

- Because curiosity is a general, task-independent cost function, that if we successfully incorporate to our learning machines, it may result in agents that (want to) improve with experience, like people do.
- Those intelligent agents would not require supervision by coding up reward functions for every little task, they would learn (almost) autonomously
- Curiosity-driven motivation is beyond satisfaction of hunger, thirst, and other biological activities (which arguably would be harder to code up in artificial agents..)

# Curiosity-driven exploration

Seek novelty/surprise:

- Visit *novel states*  $s$
- Observe *novel state transitions*  $(s, a) \rightarrow s'$

Q: How can we computationally formalize that?

A: Many answers provided in this lecture



# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

We will be using a standard model-free or model-based RL method and add exploration-based rewards in addition to the external environment rewards

# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

We will use the non-parametric model of the environment that we built to win the hardest RL environment, simply by methodic exploration: remembering where I have been and searching further towards places I have not been.

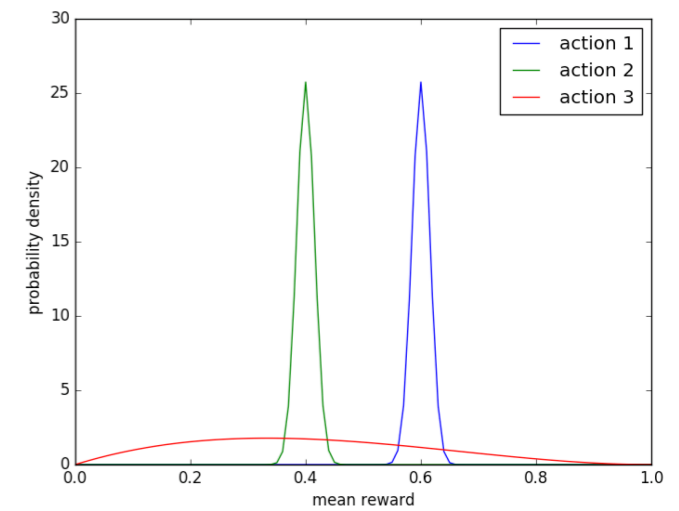
# Uncertainty-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

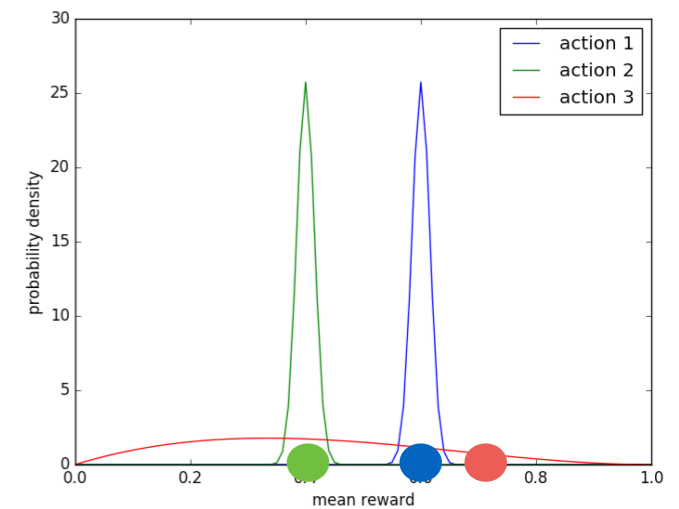


# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

1. Sample from it:  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$

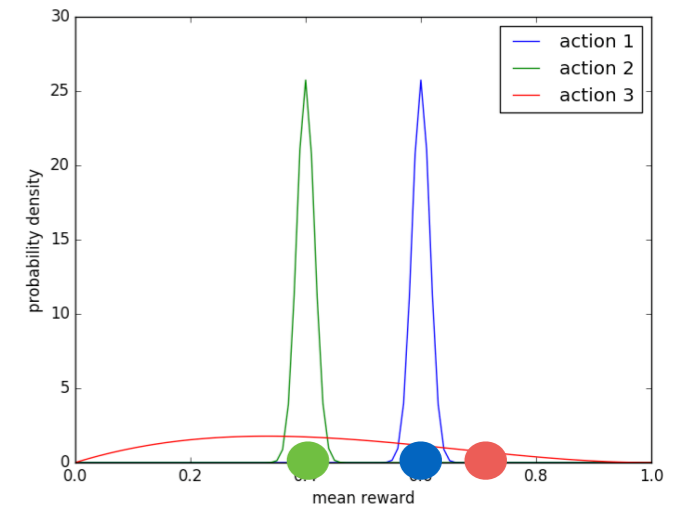


# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

1. Sample from it:  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$
2. Choose action:  $a = \arg \max_a \mathbb{E}_\theta[r(a)]$

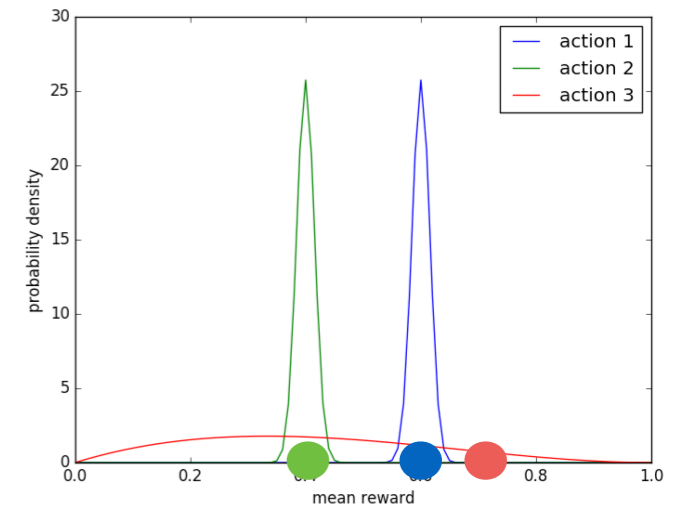


# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

1. Sample from it:  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$
2. Choose action:  $a = \arg \max_a \mathbb{E}_\theta[r(a)]$



Play the red arm!

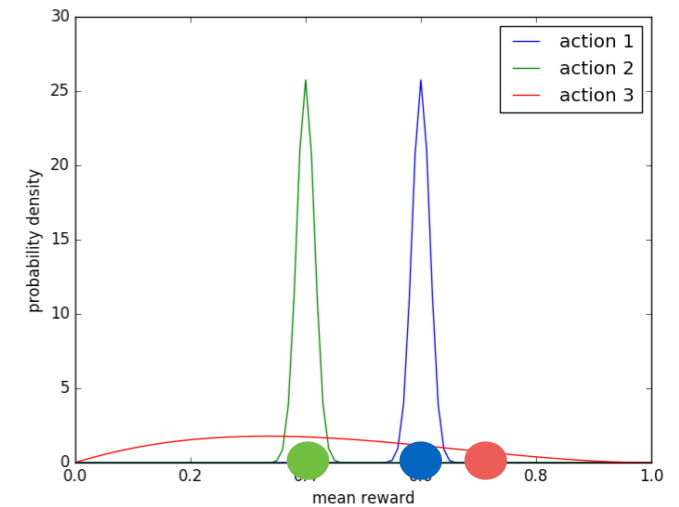


# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

1. Sample from it:  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$
2. Choose action:  $a = \arg \max_a \mathbb{E}_\theta[r(a)]$
3. Play action, observe reward



0.8!

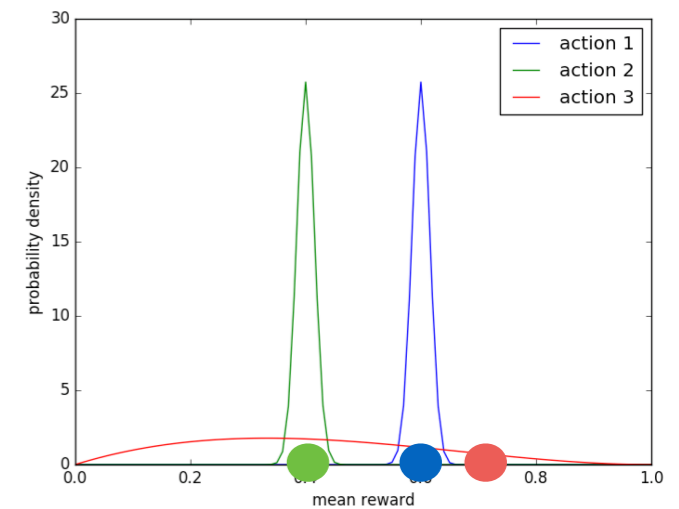
# Exploration in Bandits

## Thompson Sampling

Represent a **posterior distribution** of mean rewards of the arms, as opposed to **point estimates**.

1. Sample from it:  $\theta_1, \theta_2, \dots, \theta_k \sim \hat{p}(\theta_1, \theta_2 \dots \theta_k)$
2. Choose action:  $a = \arg \max_a \mathbb{E}_{\theta}[r(a)]$
3. Play action, observe reward
4. Update the mean reward distribution

- Can I do something like that for general MDPs?
- What is the equivalent of mean rewards for general MDP?



# Exploration via Posterior Sampling of Q functions

Represent a **posterior distribution** of  $Q$  functions, instead of a point estimate.

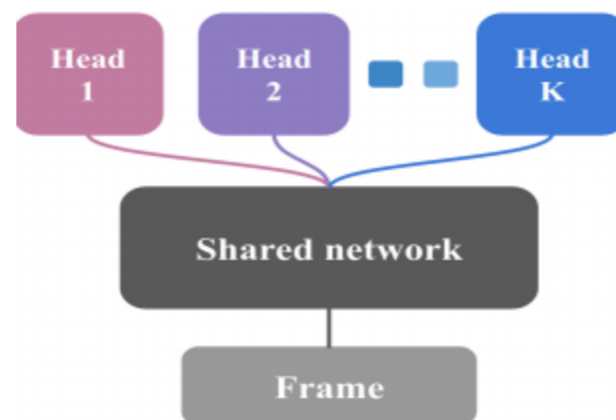
1. Sample from  $P(Q)$ :  $Q \sim P(Q)$
2. Choose actions according to this  $Q$  for one episode:  
$$a = \arg \max_a Q(a, s)$$
3. Update the  $Q$  distribution using the collected experience tuples

Then we do not need  $\epsilon$ -greedy for exploration! Better exploration by representing uncertainty over  $Q$ .

But how can we learn a **distribution of  $Q$  functions  $P(Q)$  if  $Q$  function is a deep neural network?**

# Exploration via Posterior Sampling of Q-functions

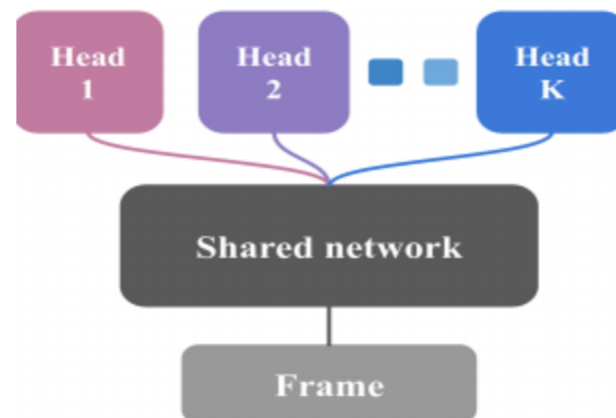
1. **Bayesian neural networks.** Estimate posteriors for the neural weights, as opposed to point estimates. We just saw that..
2. **Neural network ensembles.** Train multiple Q-function approximations each on using different subset of the data. A reasonable approximation to 1.
3. **Neural network ensembles with shared backbone.** Only the heads are trained with different subset of the data. A reasonable approximation to 2 with less computation.



4. **Ensembling by dropout.** Randomly mask-out (zero out) neural network weights, to create different neural nets, both at train and test time. reasonable approximation to 2.

# Exploration via Posterior Sampling of Q-functions

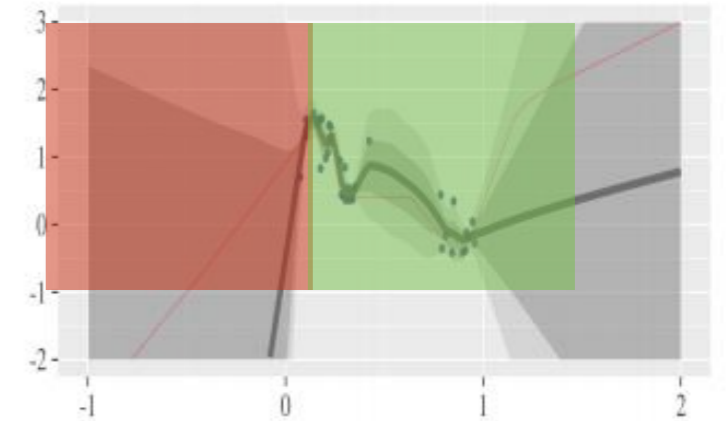
1. **Bayesian neural networks.** Estimate posteriors for the neural weights, as opposed to point estimates.
2. **Neural network ensembles.** Train multiple Q-function approximations each on using different subset of the data. A reasonable approximation to 1.
3. **Neural network ensembles with shared backbone.** Only the heads are trained with different subset of the data. A reasonable approximation to 2 with less computation.



4. **Ensembling by dropout.** Randomly mask-out (zero out) neural network weights, to create different neural nets, both at train and test time. Reasonable approximation to 2.

# Exploration via Posterior Sampling of Q-functions

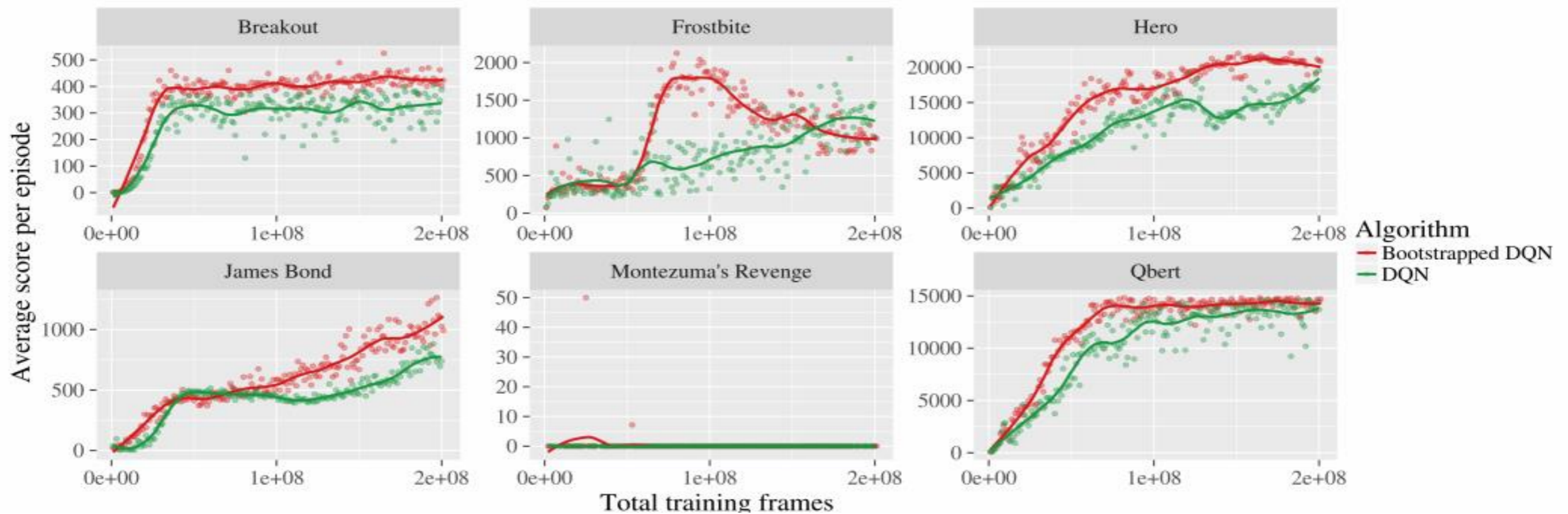
1. Sample from  $P(Q)$ :  $Q \sim P(Q)$
2. Choose actions according to this  $Q$  for one episode:  $a = \arg \max_a Q(a, s)$
3. Update the  $Q$  distribution using the collected experience tuples



With ensembles we achieve similar things as with Bayesian nets:

- The entropy of predictions of the network (obtained by sampling different heads) is high in the no data regime. Thus, Q function values will have **high entropy** there and encourage exploration.
- When Q values have **low entropy**, i exploit, i do not explore.

# Exploration via Posterior Sampling of Q-functions



# Curiosity-driven exploration-one way to do it

We will add **exploration reward bonuses** to the extrinsic (task-related) rewards:

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$$

Independent of the task in hand!

We would then be using rewards  $R^t(s, a, s')$  in our favorite RL method.



# Curiosity-driven exploration-one way to do it

We will add **exploration reward bonuses** to the extrinsic (task-related) rewards in our favorite model free RL method.

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$$

Independent of the task in hand

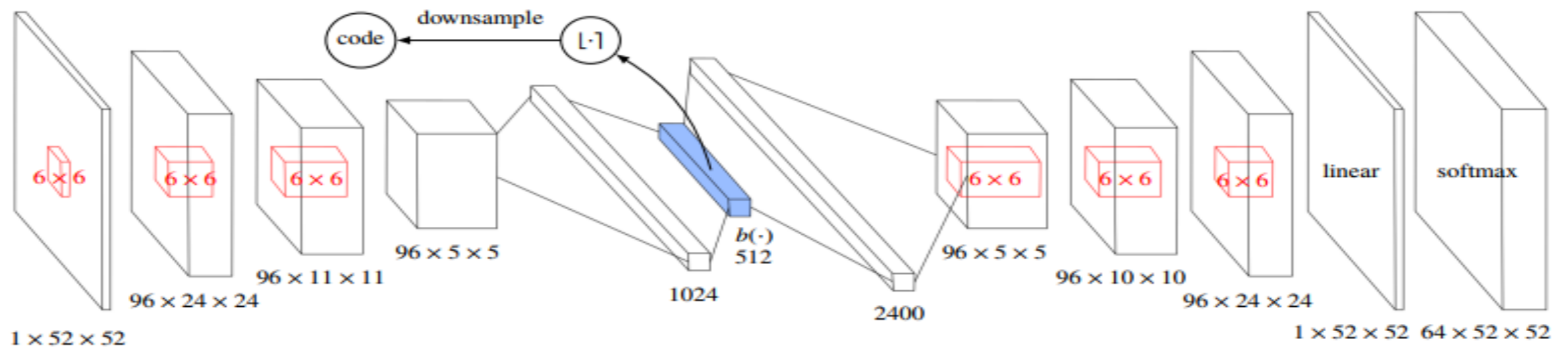
**Exploration reward bonuses are non stationary:** as the agent interacts with the environment, what is now new and novel, becomes old and known.

# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

# State Counting with DeepHashing

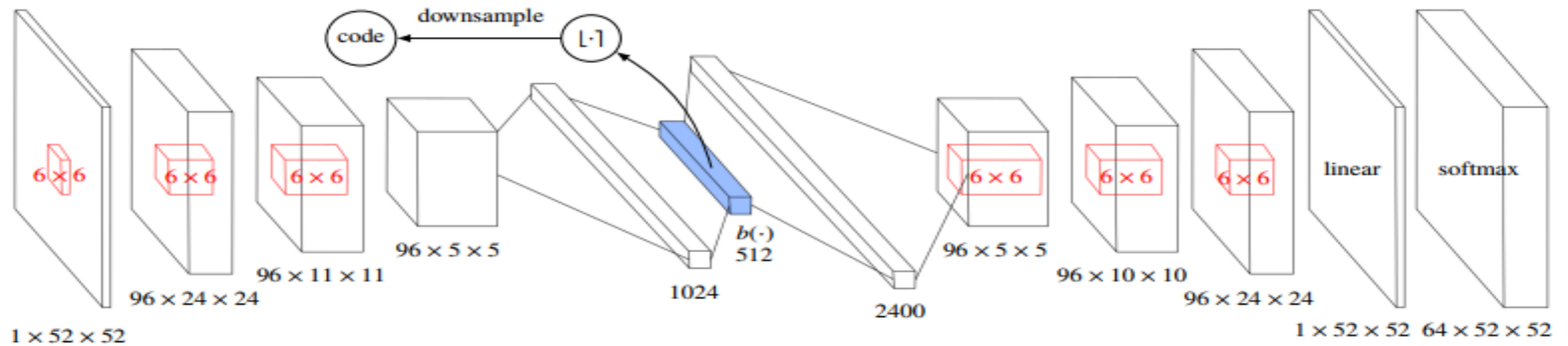
- We count states (images) but not in pixel space, but in latent compressed space.
- Compress  $s$  into a latent code, then count occurrences of the code.
- How do we get the image encoding? E.g, using autoencoders.



- Note: There is no guarantee such reconstruction loss will capture the important things that make two states to be similar or not policy wise..

# Exploration rewards

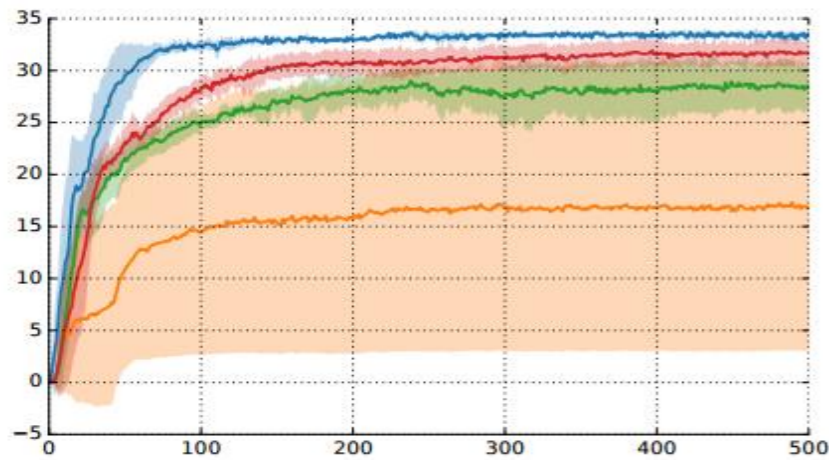
Add exploration reward bonuses that encourage policies to visit states with smaller counts:



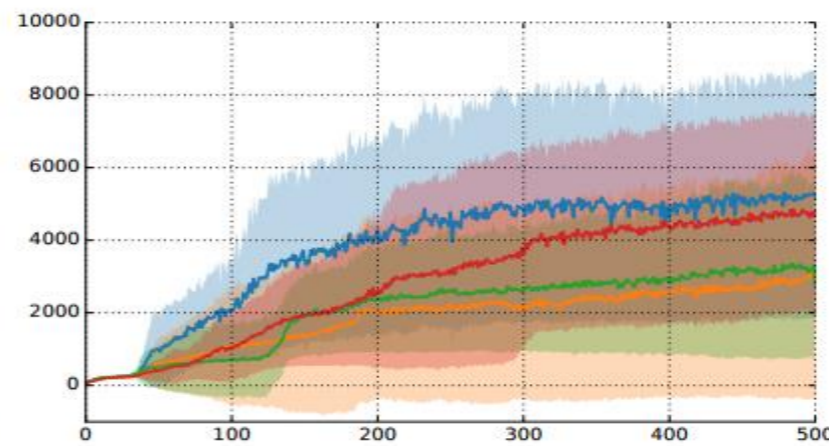
$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(\phi(s))}_{\text{intrinsic}}$$

# State Counting with DeepHash

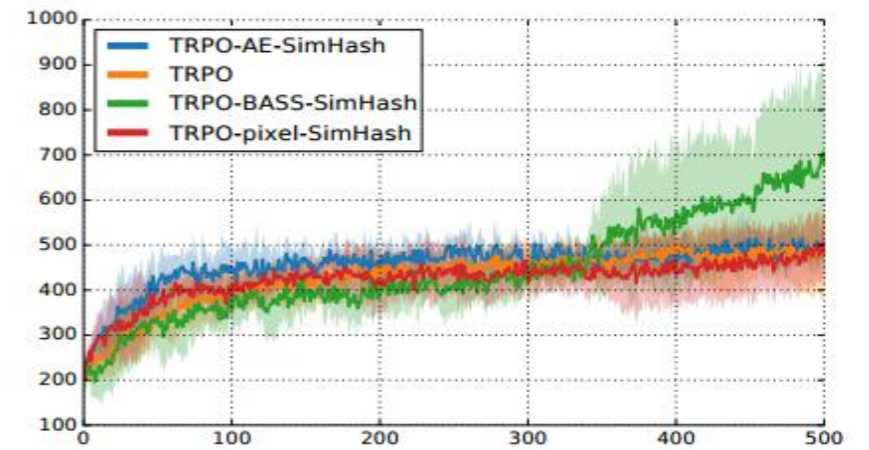
- We still count states (images) but not in pixel space, but in latent compressed space.
- Compress  $s$  into a latent code, then count occurrences of the code.
- How do we get the image encoding? E.g, using autoencoders.



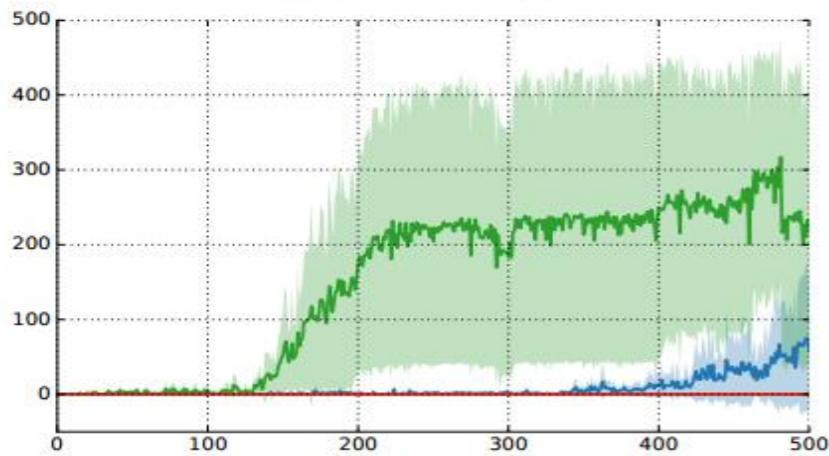
(a) Freeway



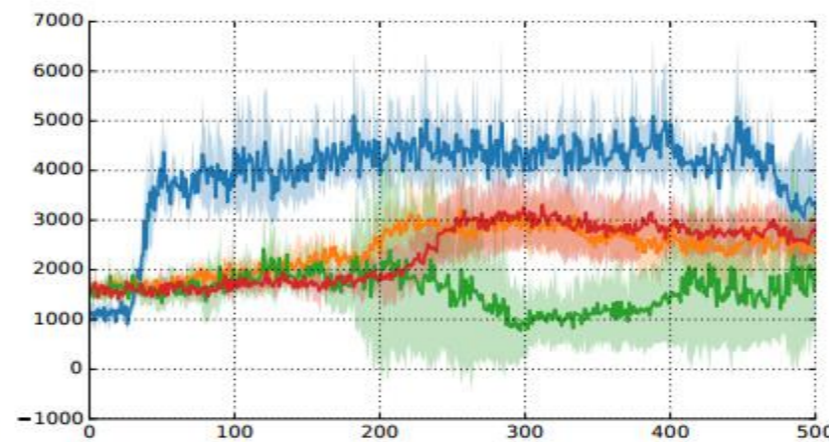
(b) Frostbite



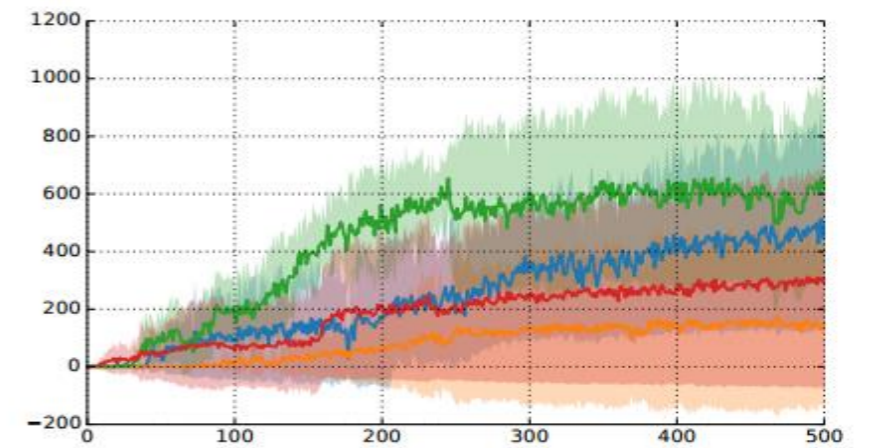
(c) Gravitar



(d) Montezuma's Revenge



(e) Solaris



(f) Venture

# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

# Exploration rewards

Add exploration reward bonuses that encourage policies to visit states that will cause the prediction model to fail.

model error!

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(\|T(s, a; \theta) - s'\|)}_{\text{intrinsic}}$$

Note: we will be using  $T(s, a; \theta)$  to denote the dynamics (transition) function.

# Computational Curiosity

- “The direct goal of curiosity and boredom is to improve the **world model**. The indirect goal is to ease the learning of new goal-directed action sequences.”
- “The same complex mechanism which is used for ‘normal’ goal-directed learning is used for implementing curiosity and boredom. There is no need for devising a separate system which aims at improving the world model.”
- “Curiosity Unit”: reward is a function of the **mismatch between model’s current predictions and actuality**. There is positive reinforcement whenever **the system fails to correctly predict the environment**.
- “Thus **the usual credit assignment process ... encourages certain past actions in order to repeat situations similar to the mismatch situation**.” (planning to make your (internal) world model to fail)



Jurgen Schmidhuber, 1991, 1991, 1997



# Computational Curiosity

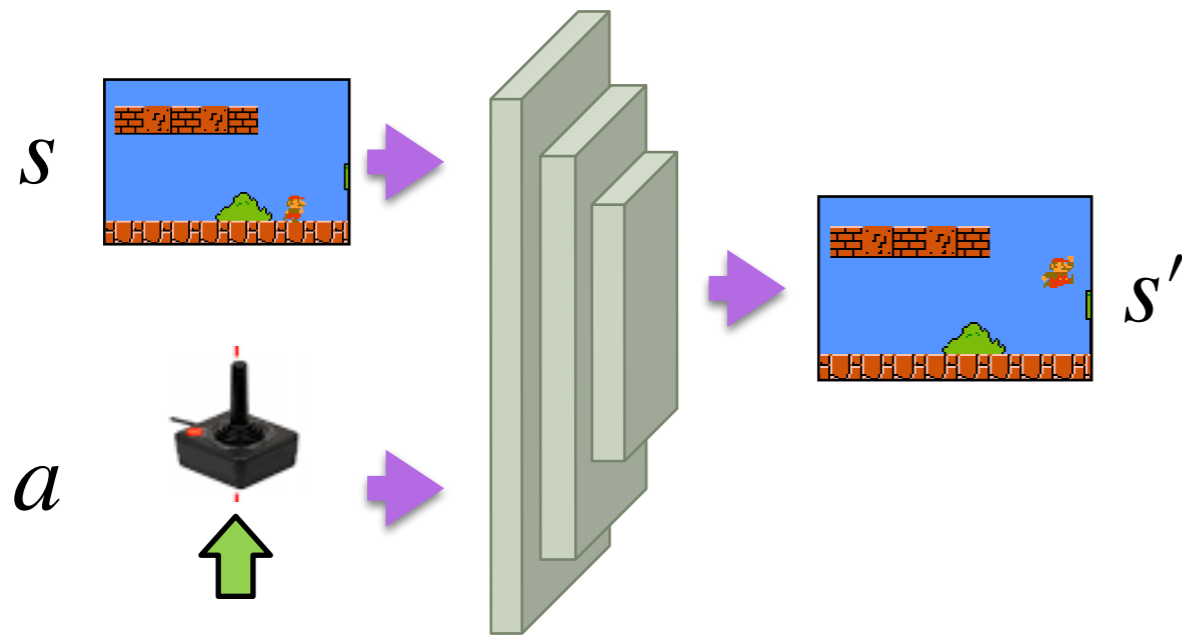
In other words:

- Model learning and model improvement can be cast as the goals of goal-seeking behaviour.
- My goal is not to beat Atari but to improve my Atari model.
- OK. What is my reward then that trying to maximize that reward will lead to fast model learning?



# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(s, a; \theta) - s'\|$



$$\min_{\theta} \|T(s, a; \theta) - s'\|$$

Here we predict the visual observation!

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$$

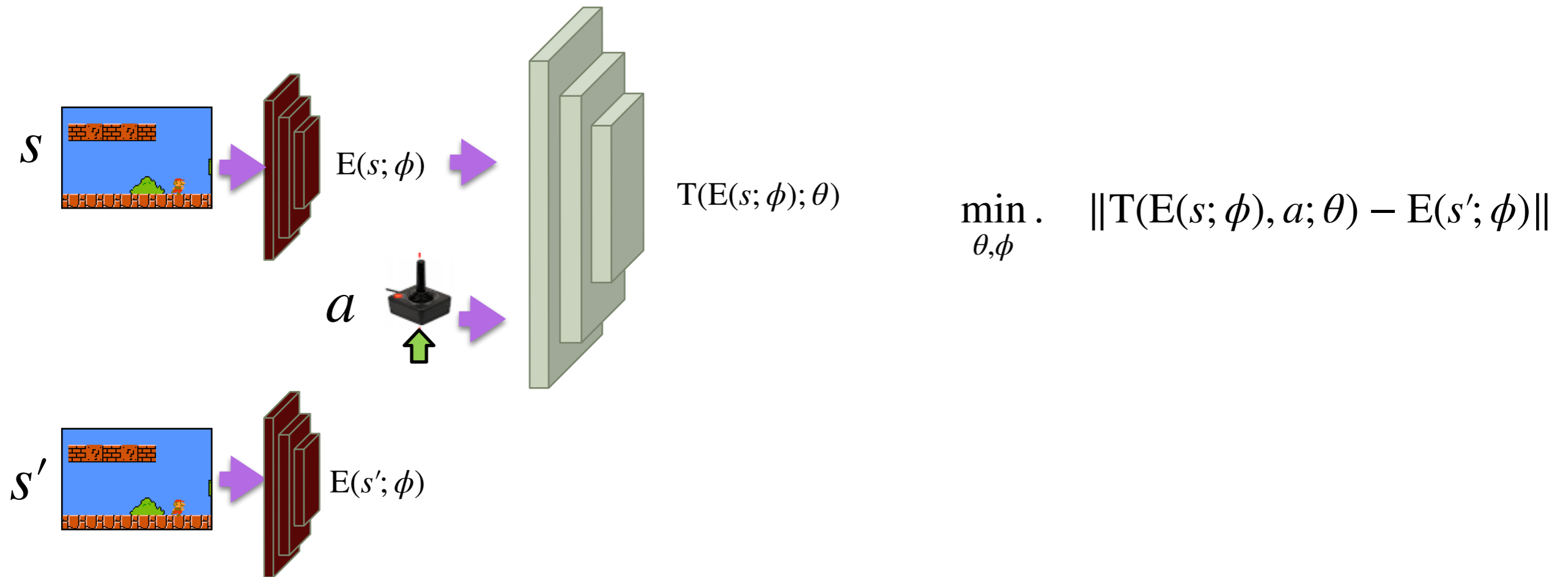
# Predicting Raw Sensory Input (Pixels)

Should our prediction model be predicting the input observations?

- **Observation prediction is difficult** especially for high dimensional observations.
- **Observation contains a lot of information unnecessary for planning**, e.g., dynamically changing backgrounds that the agent cannot control and/or are irrelevant to the reward.

# Learning Visual Dynamics

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$

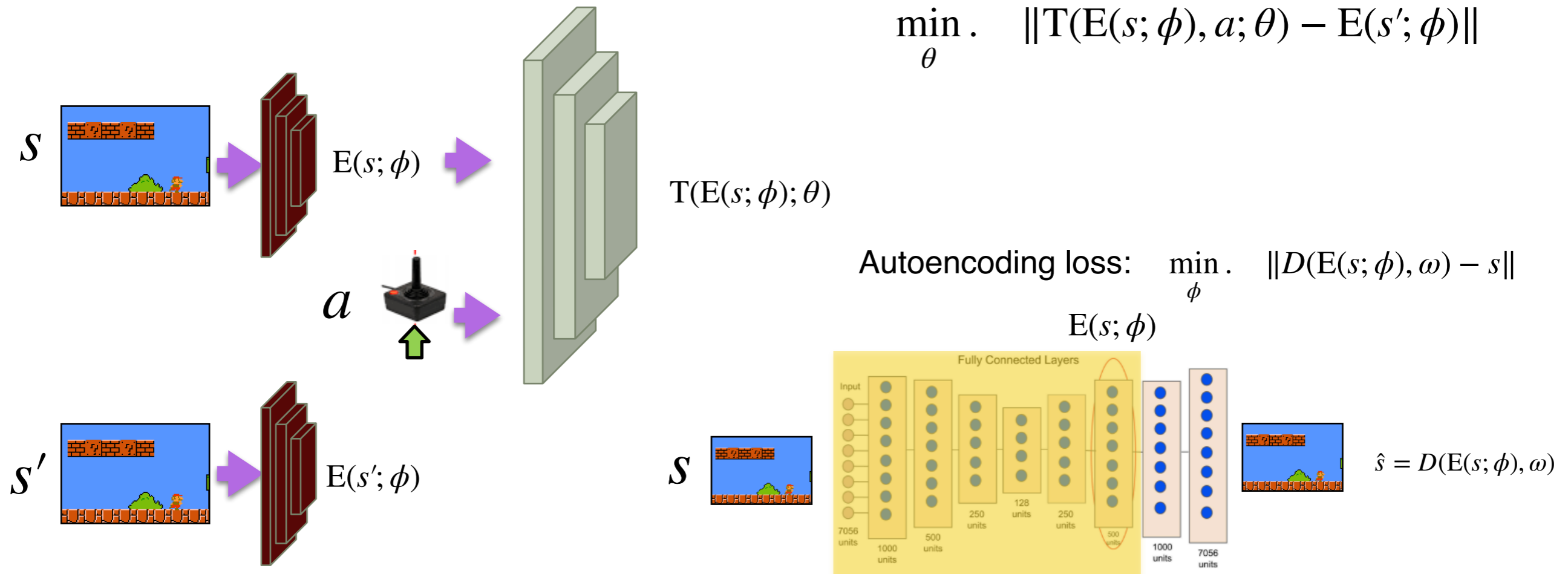


What is the problem with this optimization problem?

There is a trivial solution :-)

# Learning Visual Dynamics

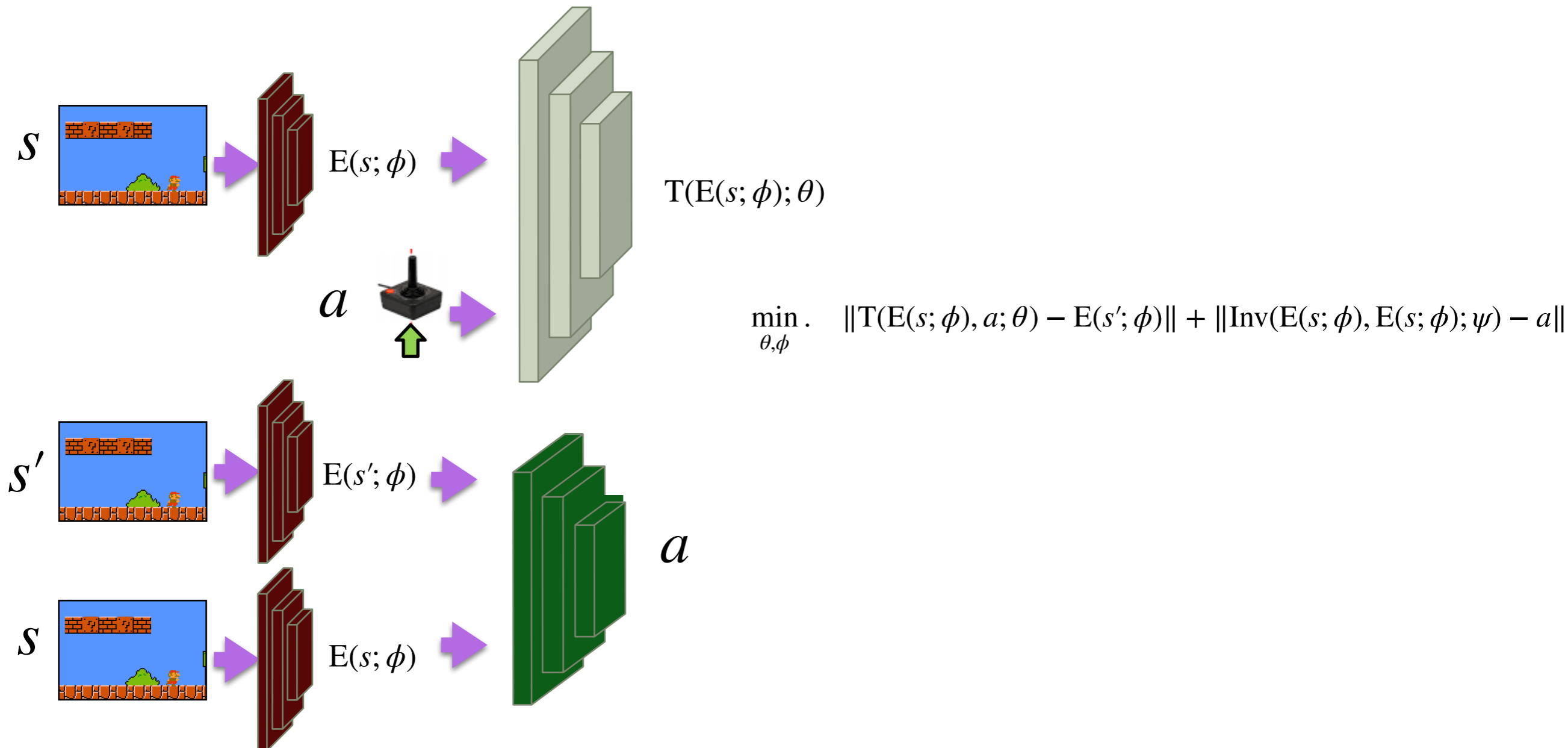
Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's learn image encoding using autoencoders (to avoid the trivial solution)
- ...and suffer the problems of autoencoding reconstruction loss that has little to do with our task

# Learning Visual Dynamics

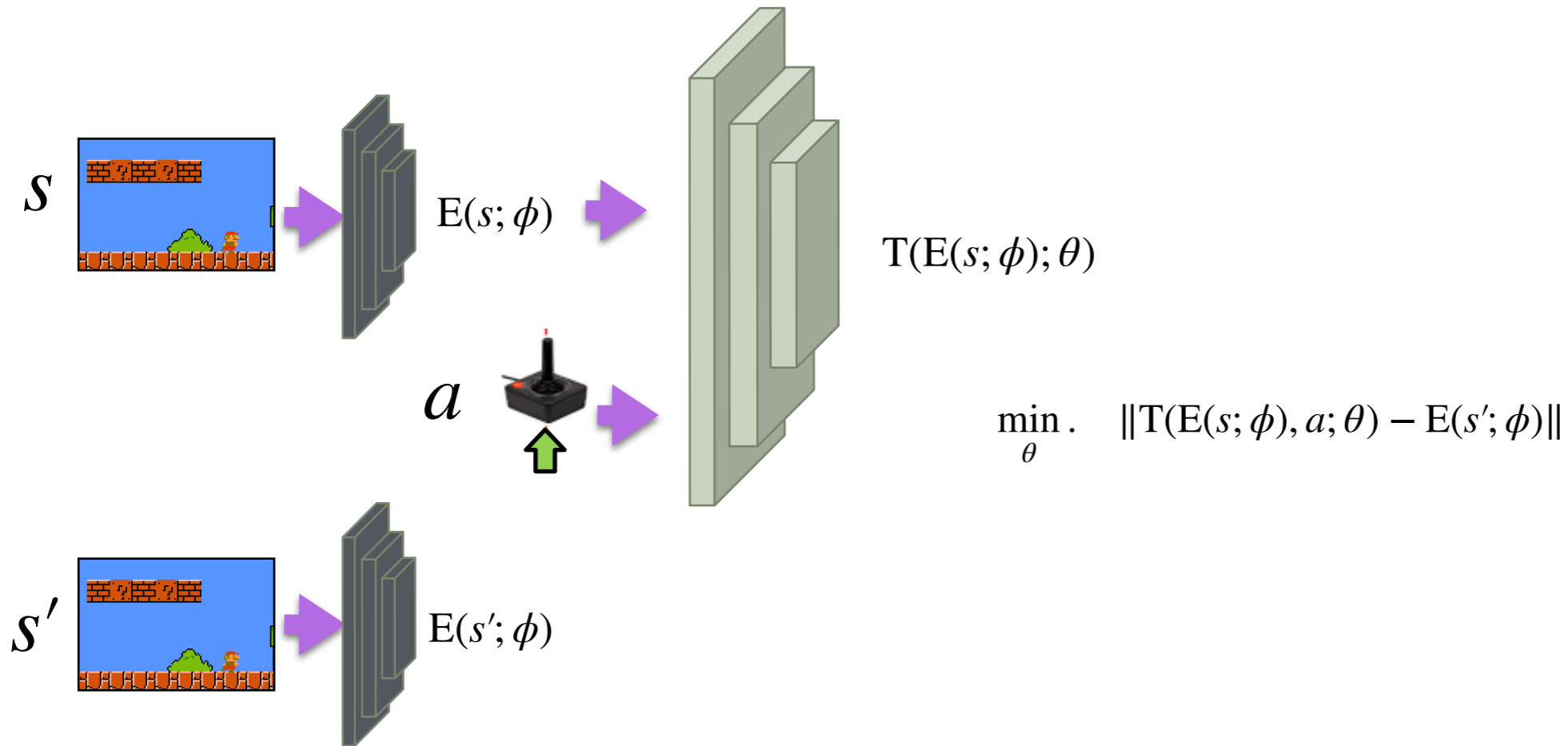
Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's couple forward and inverse models (to avoid the trivial solution)
- ...then we will only predict things that the agent can control

# Learning a Transition function

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|T(E(s; \phi), a; \theta) - E(s'; \phi)\|$



- Let's use random neural networks (networks initialized randomly and frozen thereafter)

# Task Versus Exploration rewards

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\mathbf{T}(\mathbf{E}(s; \phi), a; \theta) - \mathbf{E}(s'; \phi)\|$

Only task reward:  $R(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}}$

Task+curiosity:  $R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

Sparse task + curiosity:  $R^t(s, a, s') = \underbrace{r^T(s, a, s')}_{\text{extrinsic terminal}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$



# Task Versus Exploration rewards

Exploration reward bonus  $\mathcal{B}^t(s, a, s') = \|\mathbf{T}(\mathbf{E}(s; \phi), a; \theta) - \mathbf{E}(s'; \phi)\|$

Only task reward:  $R(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}}$

Task+curiosity:  $R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

Sparse task + curiosity:  $R^t(s, a, s') = \underbrace{r^T(s, a, s')}_{\text{extrinsic terminal}} + \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

**Only curiosity:**  $R^t(s, a, s') = \underbrace{\mathcal{B}^t(s, a, s')}_{\text{intrinsic}}$

Train an A3C agent under only curiosity reward.  
Will it learn to do something useful?

# Policy Transfer

Policies trained with A3C using only curiosity rewards  
Prediction error using forward/inverse model coupling

## Trained on Level-1



## Testing on Level-2



# Limitation of Prediction Error as Bonus

- Agent will be rewarded even though the model cannot improve.
- The agent is attracted forever in the most noisy states, with unpredictable outcomes.
- If we give the agent a TV and a remote, it becomes a couch potato!



# Curiosity-driven exploration

- Ensembles of Q functions: modeling uncertainty of Q values
- State counting: the lower the count of the state the higher the exploration bonus
- Model prediction error: the higher the prediction error the higher the curiosity
- **Reachability: the least reachable a state from a set of already reached states in my memory, the higher the exploration bonus**
- Non-parametric memory of states and their transitions (reachability) of one to the other. Explore by maximizing coverage.

# EPISODIC CURIOSITY THROUGH REACHABILITY

Nikolay Savinov<sup>\*1</sup>   Anton Raichuk<sup>\*1</sup>   Raphaël Marinier<sup>\*1</sup>   Damien Vincent<sup>\*1</sup>  
Marc Pollefeys<sup>3</sup>   Timothy Lillicrap<sup>2</sup>   Sylvain Gelly<sup>1</sup>

<sup>1</sup>Google Brain, <sup>2</sup>DeepMind, <sup>3</sup>ETH Zürich

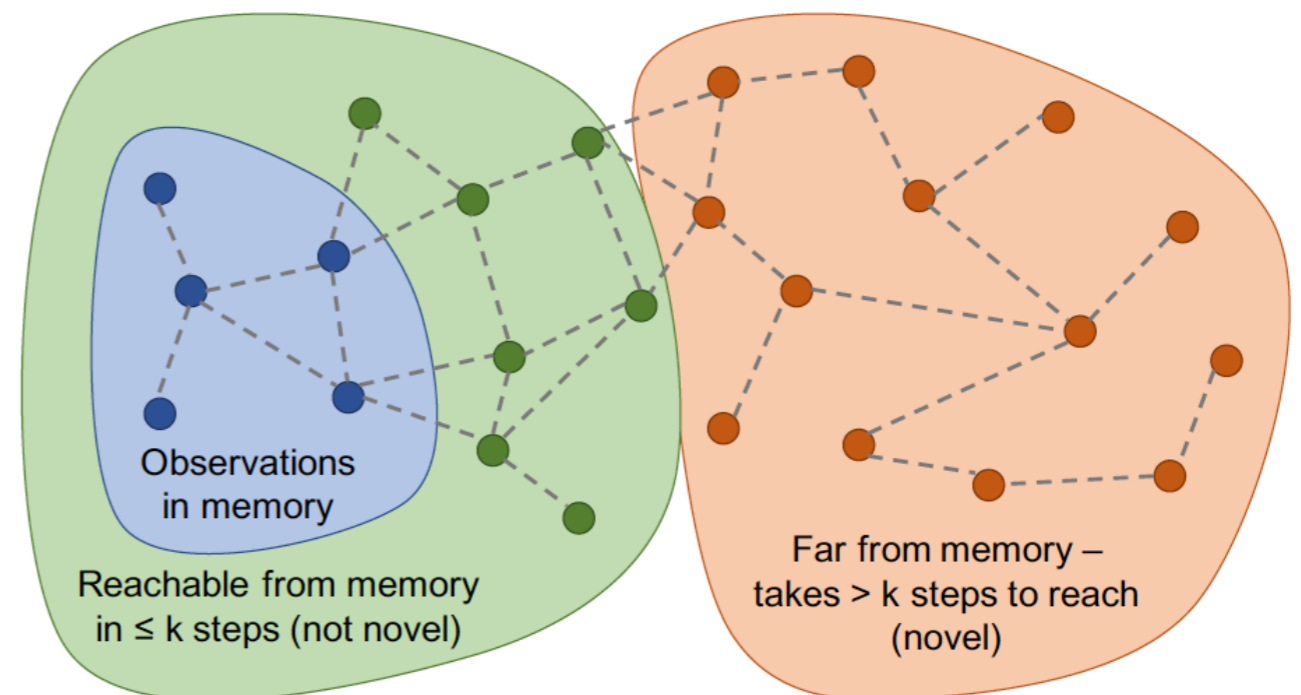
- We will be using augmented rewards as before

$$R^t(s, a, s') = \underbrace{r(s, a, s')}_{\text{extrinsic}} + \underbrace{\mathcal{B}^t(s, \mathcal{M})}_{\text{intrinsic}}, \text{ where } \mathcal{M} \text{ is a non-parametric}$$

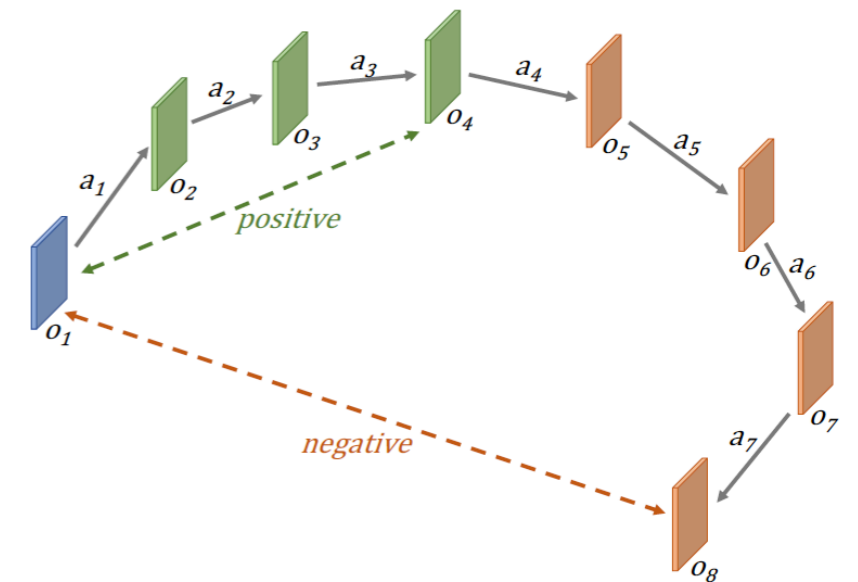
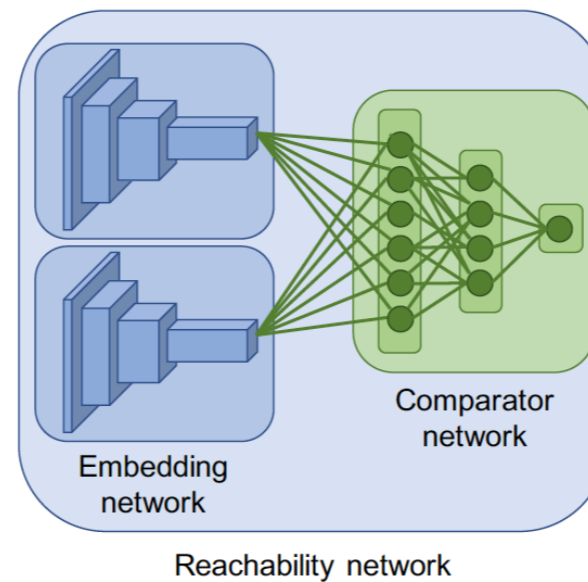
memory structure populated with embeddings of past image observations.

- Curiosity reward will use a comparator neural net, that takes as input two images and predicts whether they are close (few actions apart) or far
- We will plug those rewards into PPO, a model-free RL method

non-parametric memory structure

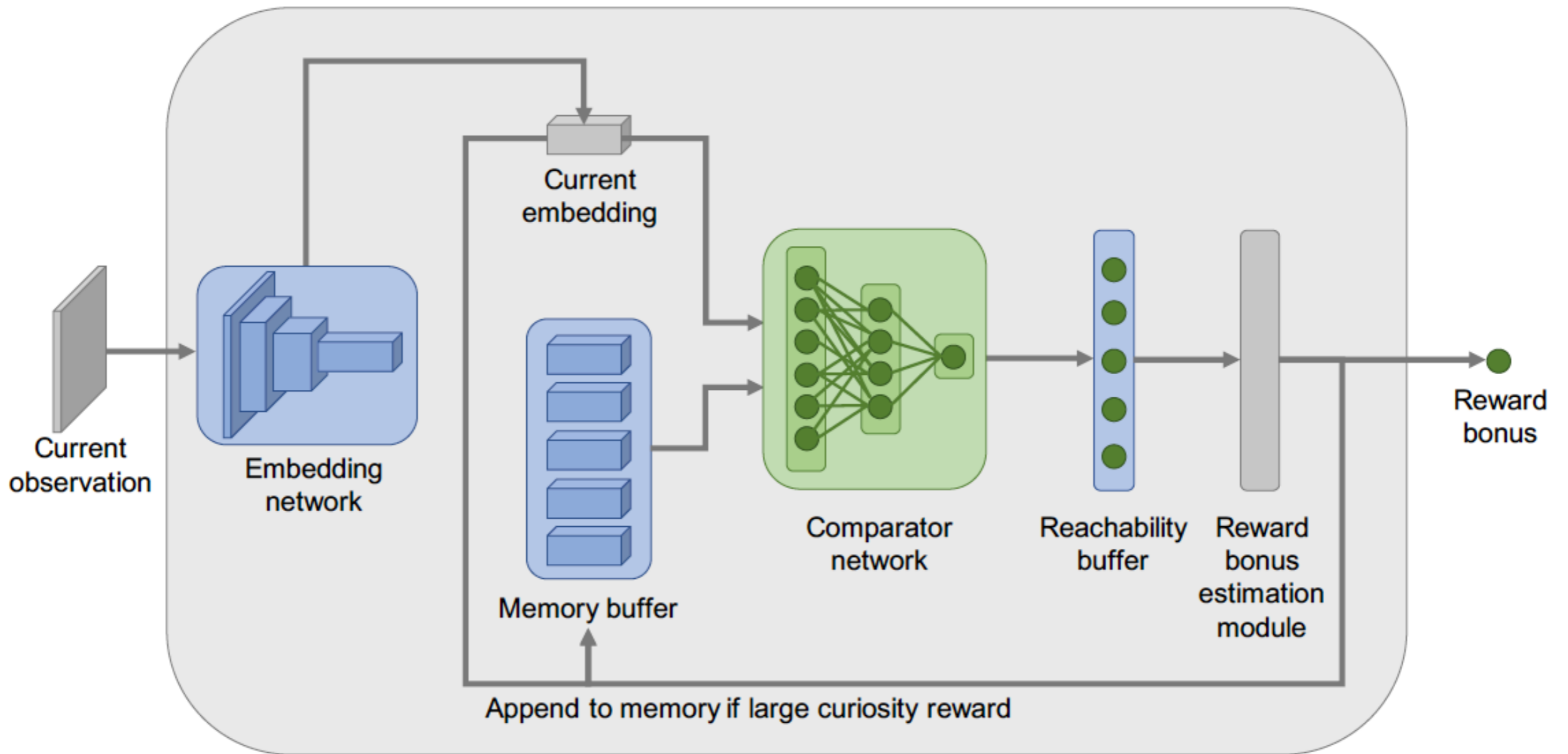


comparator network trained with temporal contrastive learning



$$\mathcal{L}_\phi(o, o^+, o^-) = \|E(o, \phi) - E(o^+, \phi)\| + \max(0, \gamma - \|E(o, \phi) - E(o^-, \phi)\|)$$

At each time step the agent compares the current observation with the ones in memory. If it is novel (takes more steps to reach than a threshold) then agent get rewarded, and the novel observation is added into memory.



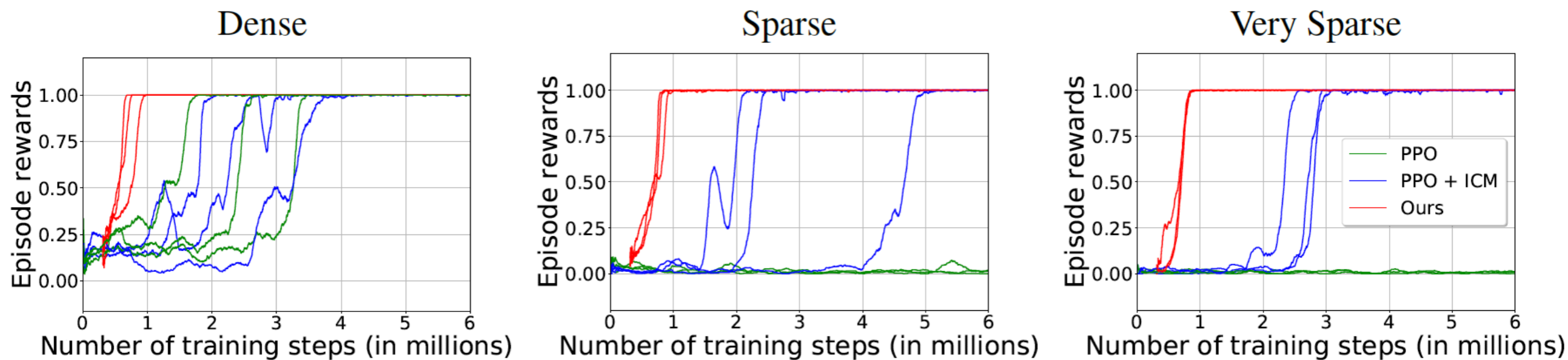


Figure 6: Task reward as a function of training step for *VizDoom* tasks. Higher is better. We use the offline version of our algorithm and shift the curves for our method by the number of environment steps used to train R-network — so the comparison is fair. We run every method with a repeat of 3 (same as in prior work (Pathak et al., 2017)) and show all runs. No seed tuning is performed.



---

# Go-Explore: a New Approach for Hard-Exploration Problems

---

**Adrien Ecoffet**

**Joost Huizinga**

**Joel Lehman**

**Kenneth O. Stanley\***

**Jeff Clune\***

Uber AI Labs

San Francisco, CA 94103

`adrienle, jhuizinga, joel.lehman, kstanley, jeffclune@uber.com`

\*Co-senior authors

Structured memory as state connectivity map

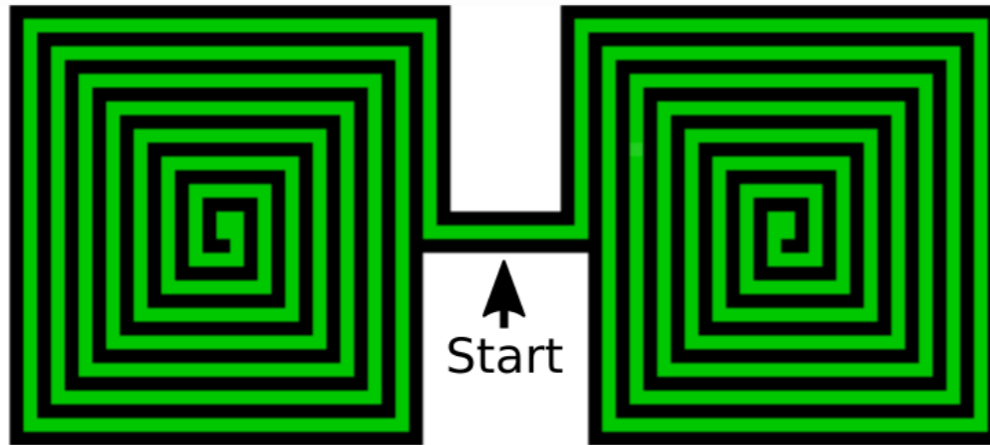
# Intrinsic Motivation

- Helps, but why doesn't it work better?

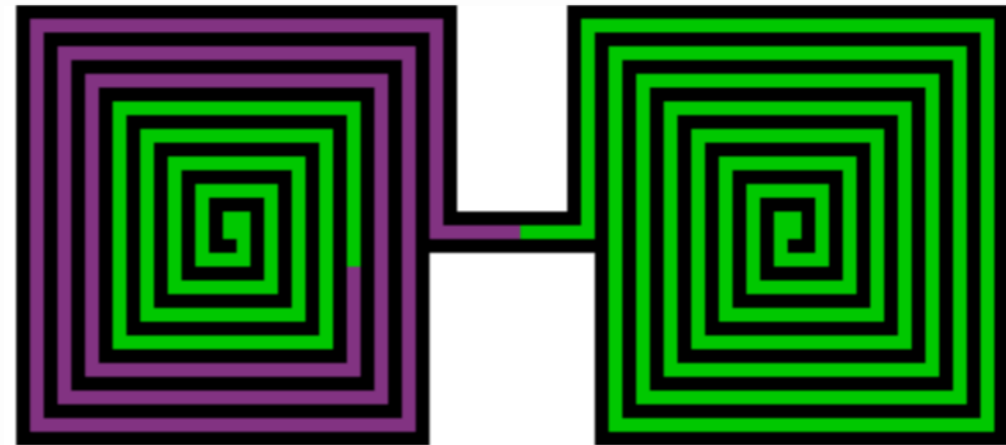


# “Detachment”

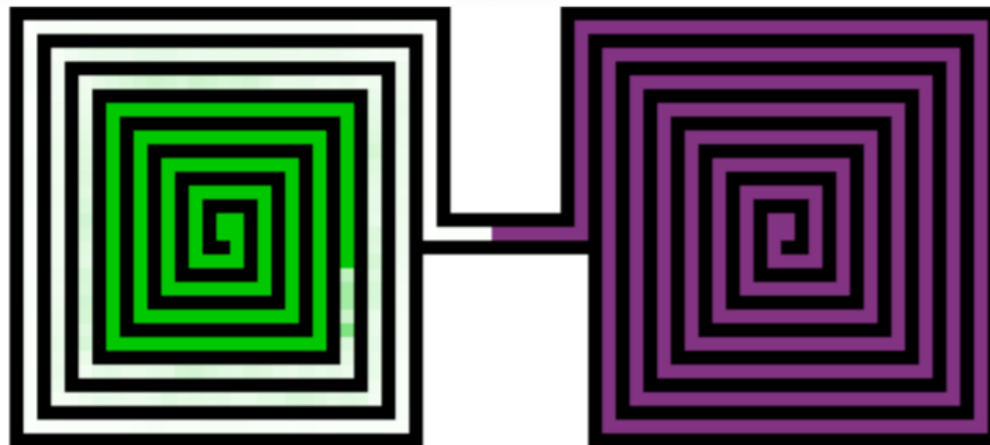
1. Intrinsic reward (green) is distributed throughout the environment



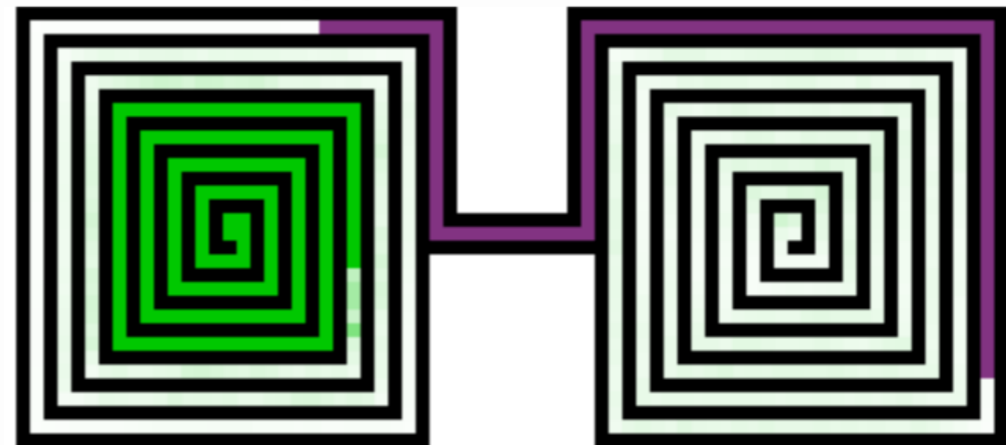
2. An IM algorithm might start by exploring (purple) a nearby area with intrinsic reward



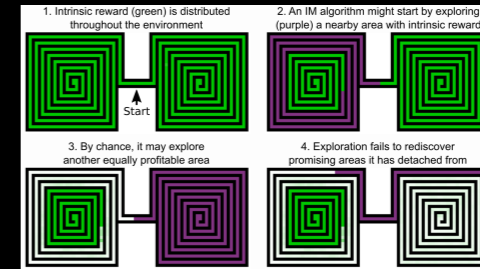
3. By chance, it may explore another equally profitable area



4. Exploration fails to rediscover promising areas it has detached from



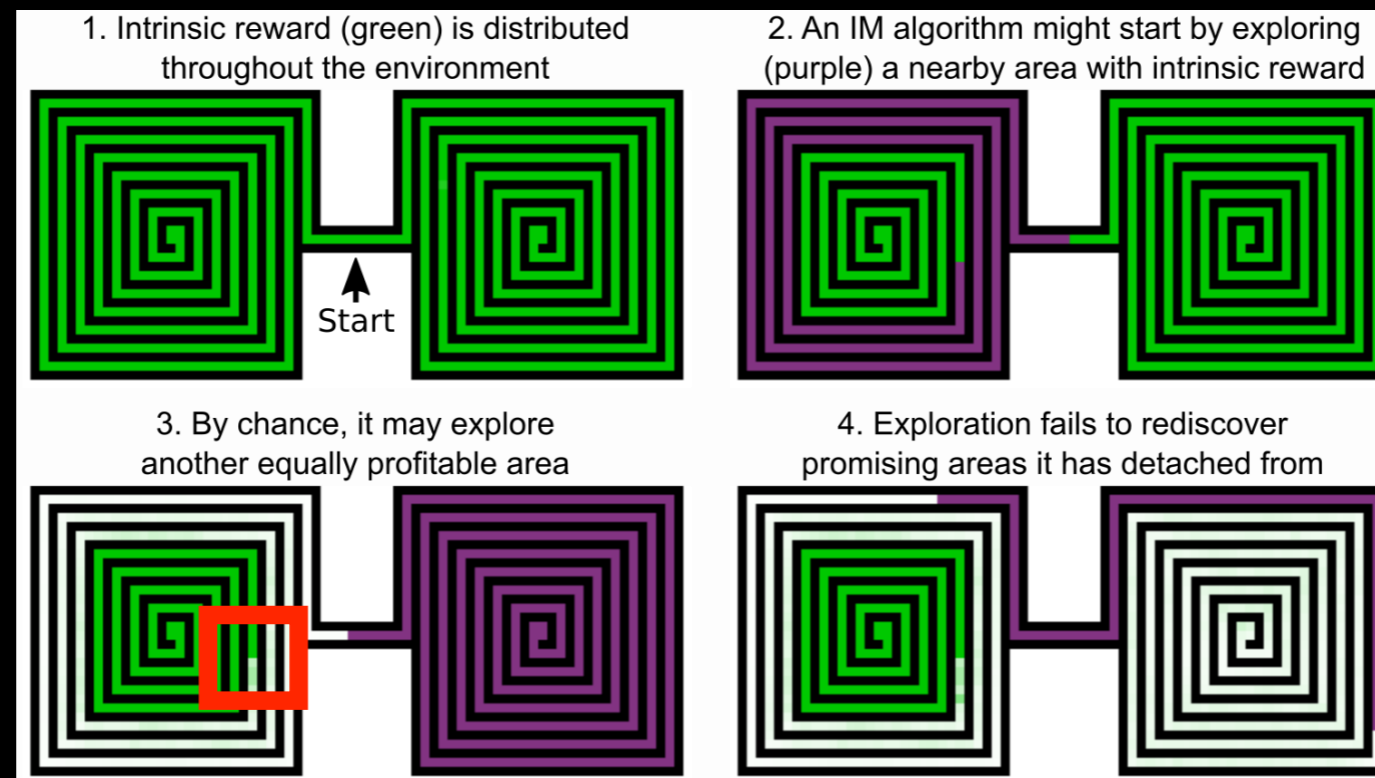
# Detachment



- Replay buffers
  - Should remember in theory, but forget/fail in practice
    - replay buffer size must be very large
    - but that causes optimization/stability issues

# Detachment

- Proposal: **explicitly remember**
  - where promising locations are
  - how to get back to them



---

# Go-Explore: a New Approach for Hard-Exploration Problems

---

**Adrien Ecoffet**

**Joost Huizinga**

**Joel Lehman**

**Kenneth O. Stanley\***

**Jeff Clune\***

Uber AI Labs

San Francisco, CA 94103

adrienle, jhuizinga, joel.lehman, kstanley, jeffclune@uber.com

\*Co-senior authors

Book-keep an archive of trajectories that—accidentally through exploration—reached a particular state in the world and the state reached.

We will be updating the archive with:

- New states reached and the corresponding trajectories
- Better (shorter) trajectories to reach already archived states
- We will be resetting ourselves to such states and continue exploring updating our **state connectivity map**.

Then, we will turn such single trajectories into robust policies.

# “Derailment”

- Most RL algorithms:
  - take promising policy, perturb it, hope it explores further
  - most likely breaks policy!
    - especially as length, complexity, & precision of sequence increases



# Derailment

- Insight: **First return, then explore**





# Derailment

- Insight: **First return, then explore**
- counter: hurt robustness?



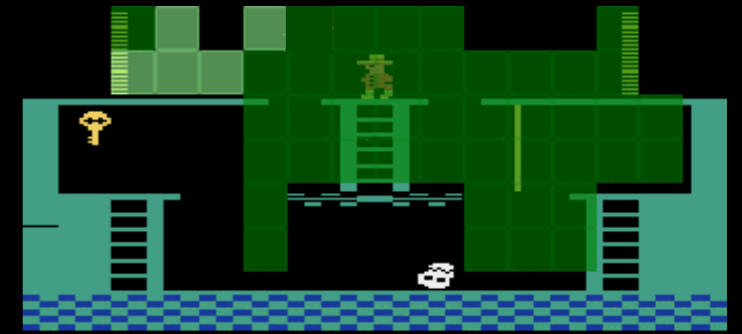
# Go-Explore Strategy

- Phase 1: **First solve**
- Phase 2: **Then robustify**
  - pay the cost to robustify only once you know *what* needs to be robustified



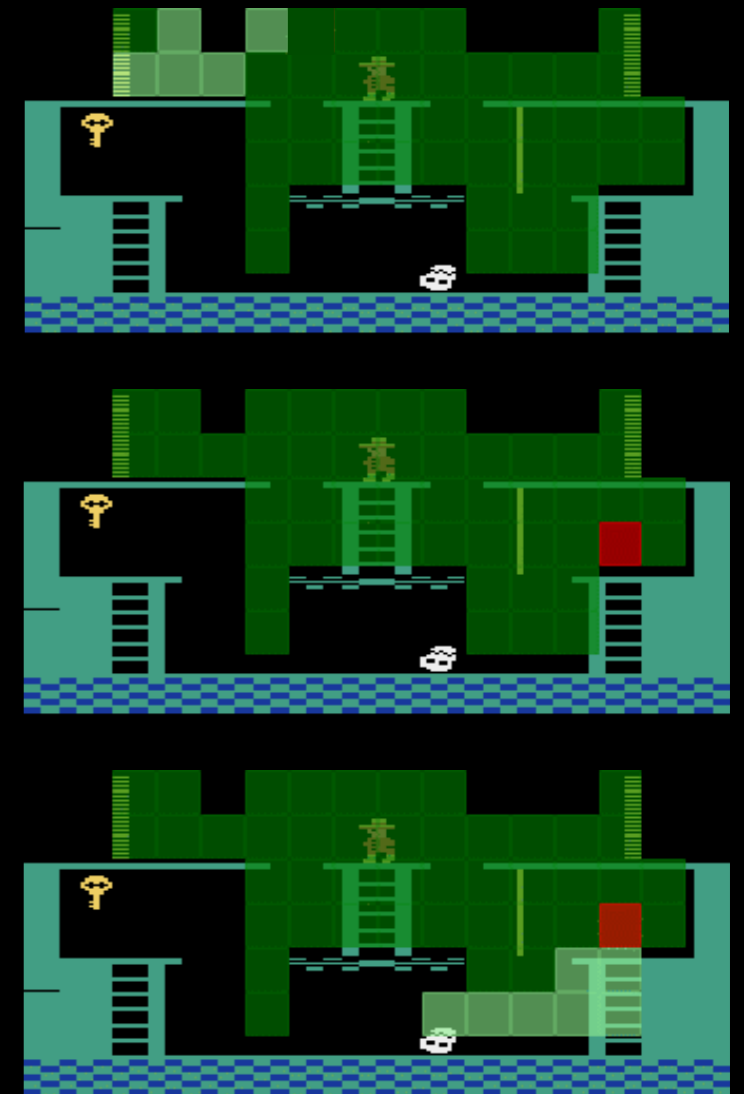
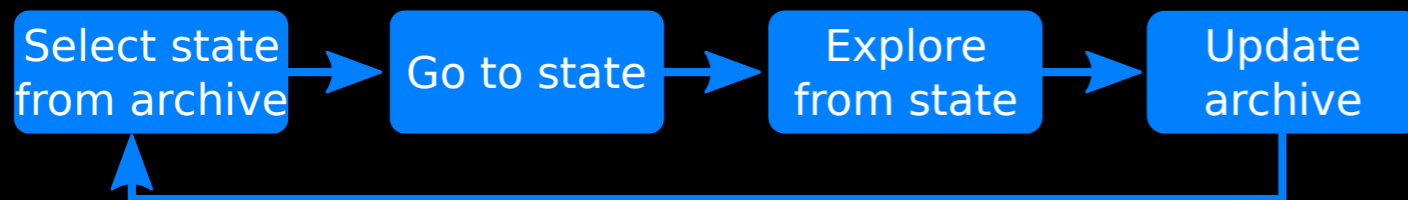
# Go-Explore: Phase 1

- initialization:
  - take random actions, store cells visited



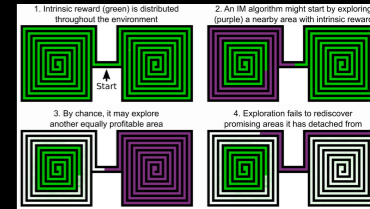
# Go-Explore: Phase 1

- Phase 1: explore until solved
  - A. choose a cell from archive
  - B. Go back to it
  - C. Explore from it
  - D. add newly found cells to archive
    - if better, replace old way of reaching cell

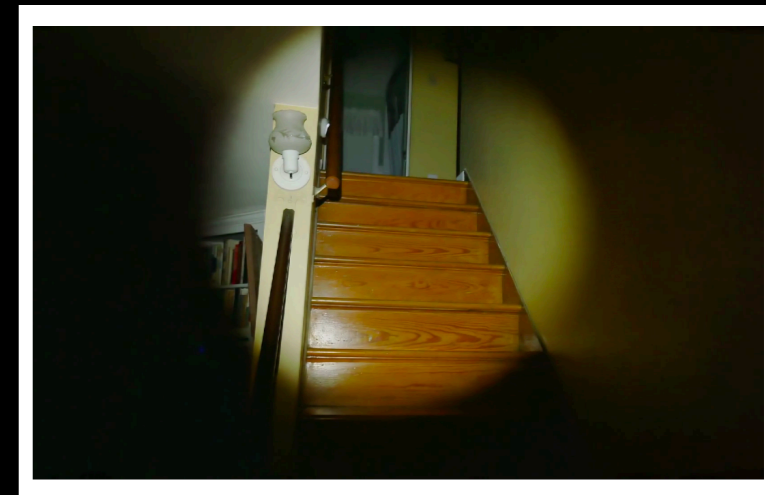


# Avoids Detachment

By Remembering Promising Exploration Stepping Stones



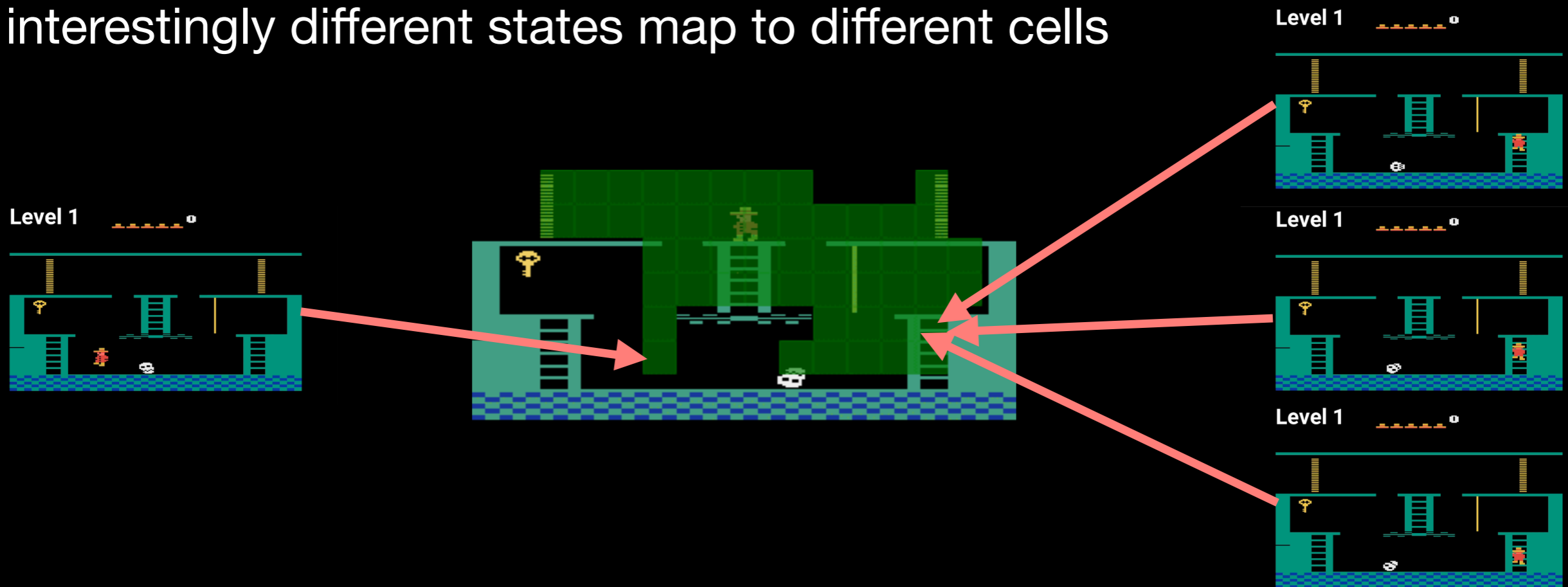
- Intrinsic motivation:
  - narrow beam mining intrinsic motivation and moving on
- Go-Explore
  - continuously expands sphere of knowledge



We are building a room connectivity map of the house!

# Cell Representations

- For large state spaces (e.g. Atari), need conflation
  - similar states map to same cell
  - interestingly different states map to different cells



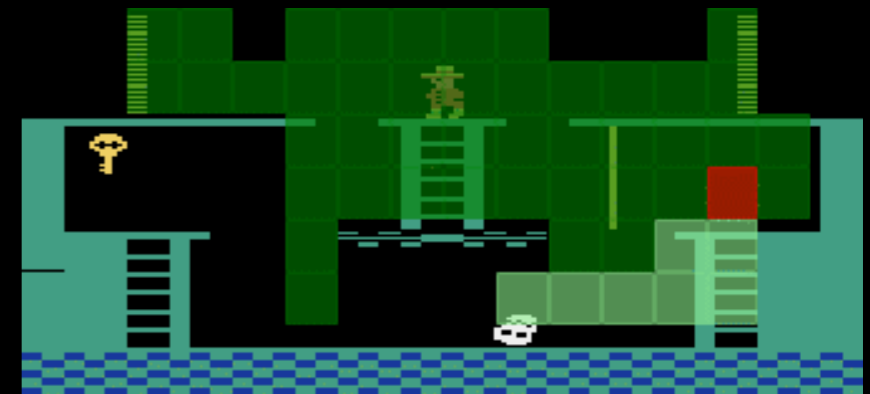
# Cell Representations

- First attempt: downsampling images
  - dumb
  - fast
  - no game-specific domain knowledge



# Choosing Cells

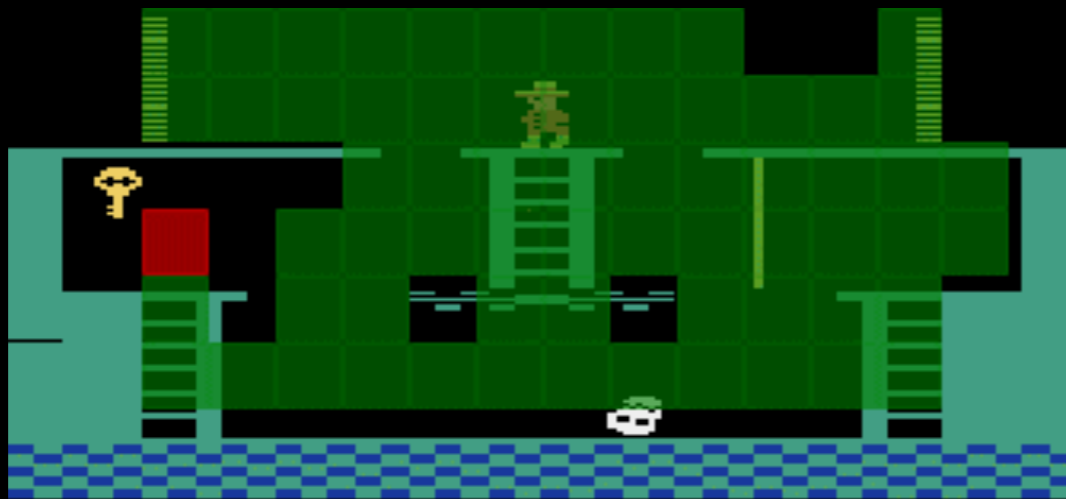
- Prefer cells that are
  - led to new states (productive)
  - less tested (newer)





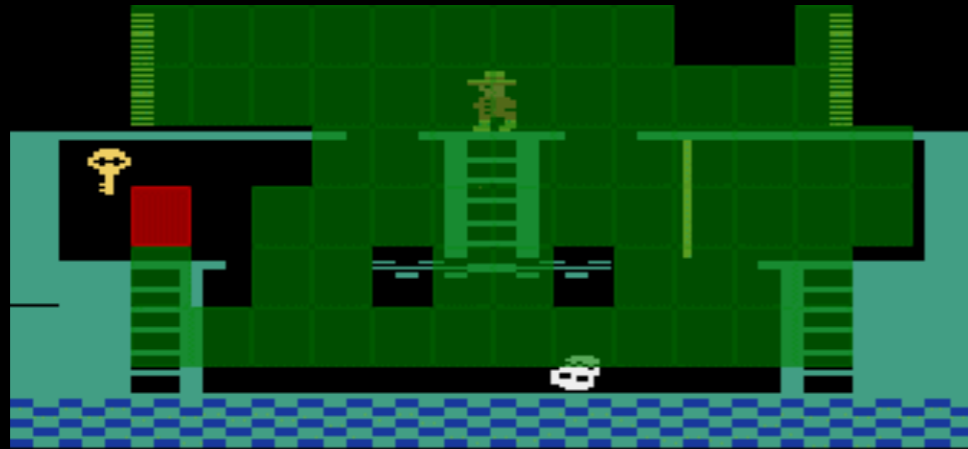
# Returning to Cells

- resettable & deterministic: reset state (or replay actions)
- stochastic environment:
  - goal-conditioned policy
    - e.g. UVFA (Schaul et al. 2015), HER (Andrychowicz et al. 2017)
    - other ideas



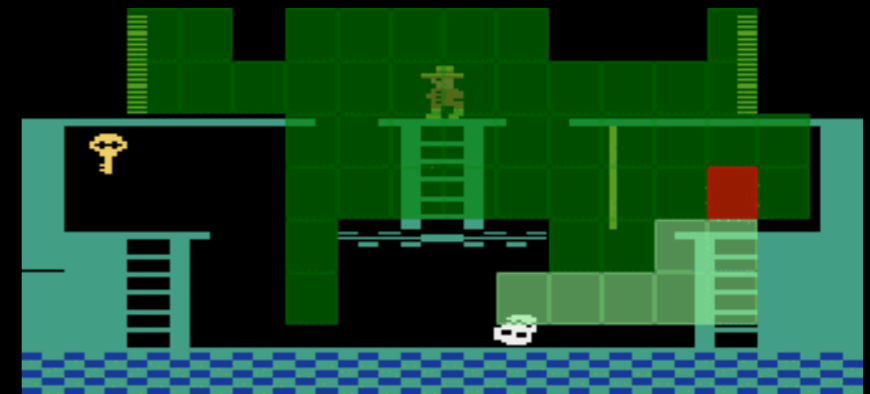
# Returning to Cells

- save action-sequence trajectories to cells
  - open loop
  - no neural network!



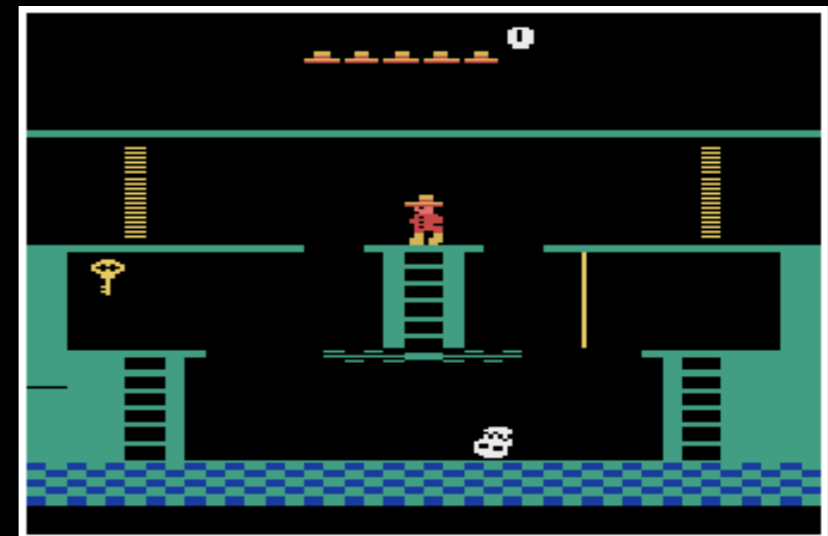
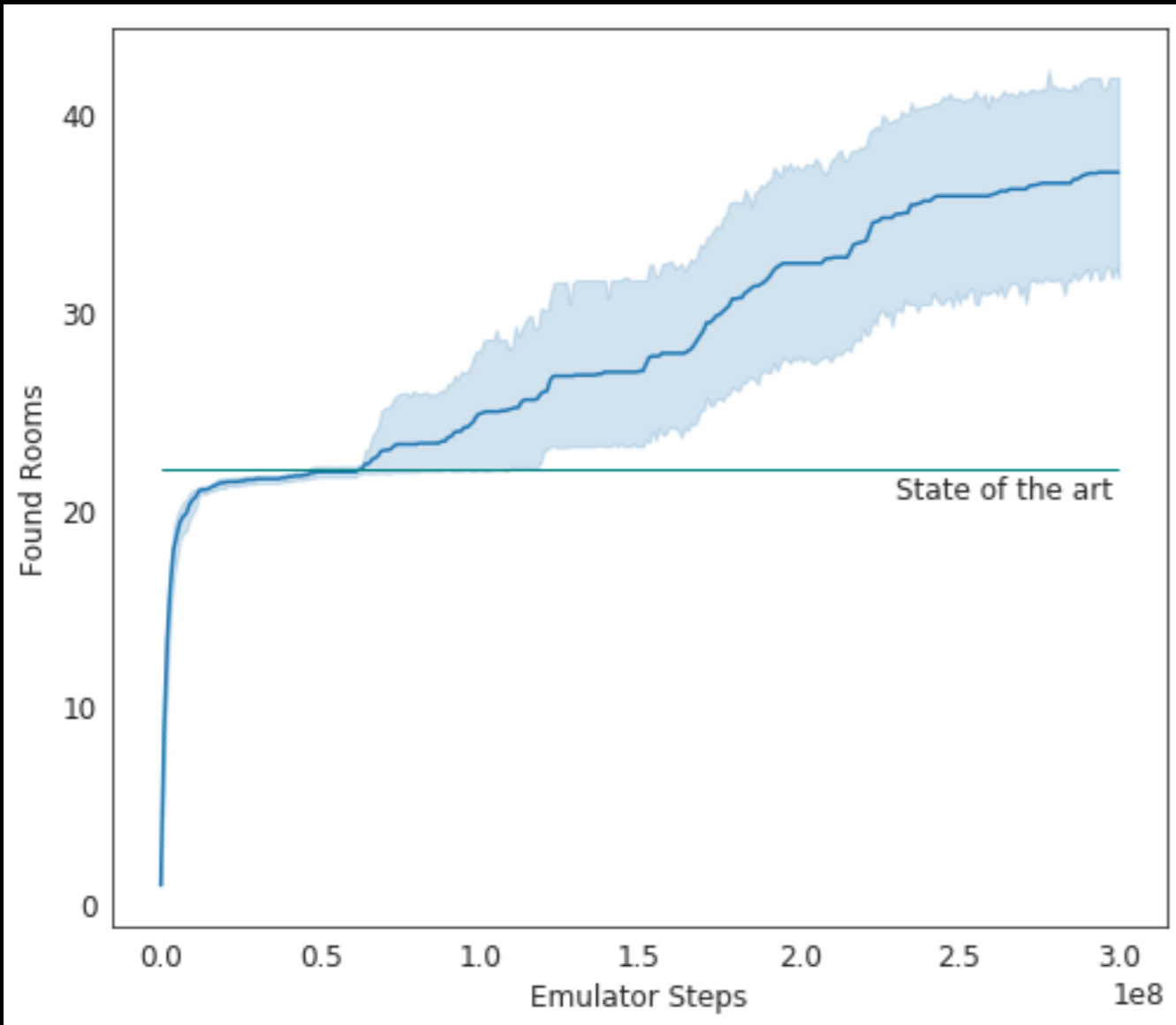
# Exploration

- after returning to a cell
- take random actions ( $k=100$ )



# Montezuma's Revenge Results: Phase 1

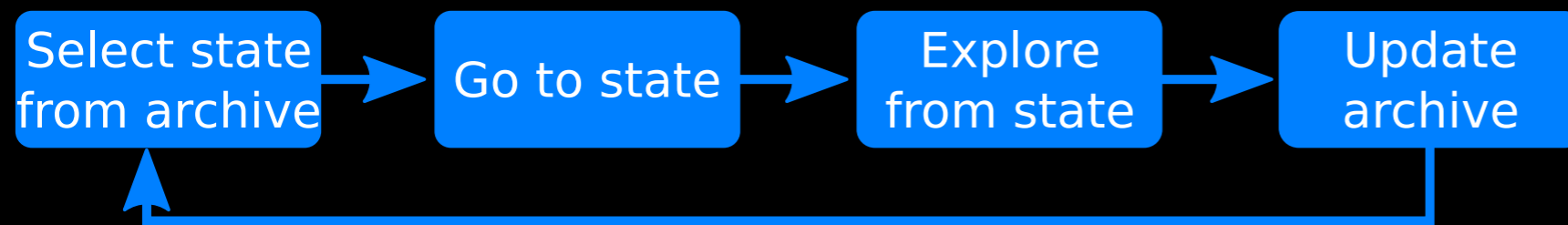
- Solves level 1 65% of runs



# Go-Explore

Separates learning a solution into two phases

## Phase 1: Explore Until Solved



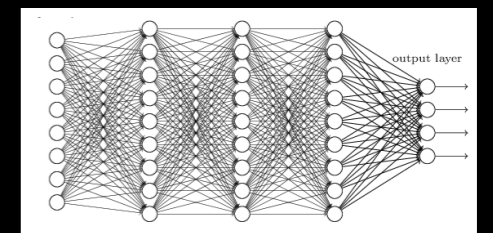
no neural networks  
(in current work)



## Phase 2: Robustify (if necessary)



produces neural network  
robust to stochasticity



## Phase 2: Robustify

- Imitation learning can work well with human demonstrations
  - including on Montezuma's Revenge & Pitfall (e.g. Aytar 2018)
- Go-Explore Phase 1 generates solution demonstrations
  - automatically
  - quickly
  - cheaply
  - as many as you want

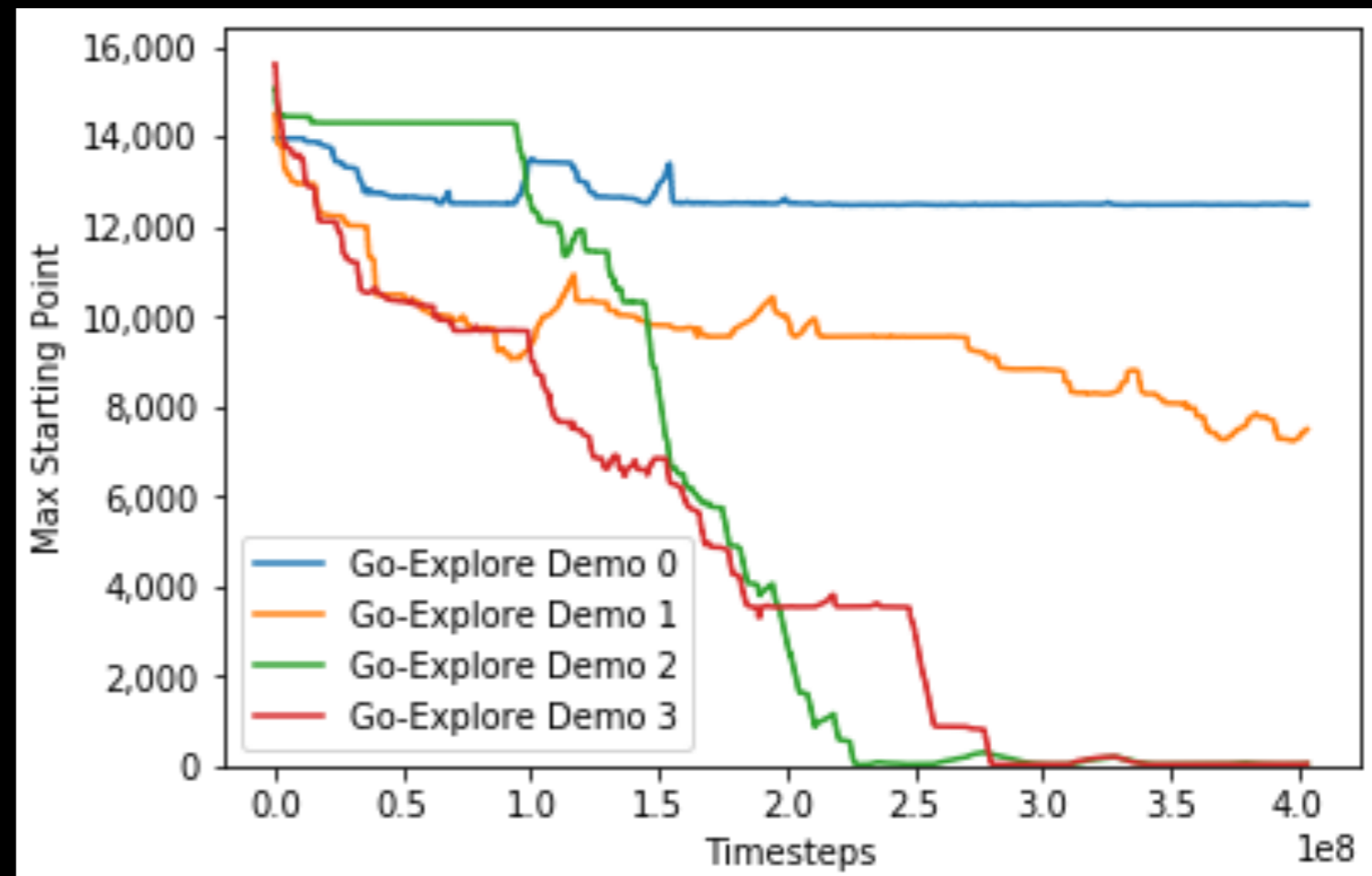
# Phase 2: Robustify

- Most imitation learning algorithms should work
- We chose the “backward algorithm”
  - from Salimans & Chen 2018 (this workshop!)
  - similar: Backplay from Resnick et al. 2018



# Many Demonstrations

- Backwards Imitation somewhat unreliable from one demonstration
- We modified it to learn from many
  - here, 4
- Add no-ops at beginning
- Success rate: 100%



Example Successful Attempt



---

**Algorithm 1** Demonstration-Initialized Rollout Worker

---

```
1: Input: a human demonstration  $\{(\tilde{s}_t, \tilde{a}_t, \tilde{r}_t, \tilde{s}_{t+1}, \tilde{d}_t)\}_{t=0}^T$ , number of starting points  $D$ , effective RNN memory length  $K$ , batch rollout length  $L$ .
2: Initialize starting point  $\tau^*$  by sampling uniformly from  $\{\tau - D, \dots, \tau\}$ 
3: Initialize environment to demonstration state  $\tilde{s}_{\tau^*}$ 
4: Initialize time counter  $i = \tau^* - K$ 
5: while TRUE do
6:   Get latest policy  $\pi(\theta)$  from optimizer
7:   Get latest reset point  $\tau$  from optimizer
8:   Initialize success counter  $W = 0$ 
9:   Initialize batch  $\mathcal{D} = \{\}$ 
10:  for step in  $0, \dots, L - 1$  do
11:    if  $i \geq \tau^*$  then
12:      Sample action  $a_i \sim \pi(s_i, \theta)$ 
13:      Take action  $a_i$  in the environment
14:      Receive reward  $r_i$ , next state  $s_{i+1}$  and done signal  $d_{i+1}$ 
15:       $m_i = \text{TRUE}$  ▷ We can train on this data
16:    else ▷ Replay demonstration to initialize RNN state of policy
17:      Copy data from demonstration  $a_i = \tilde{a}_i, r_i = \tilde{r}_i, s_{i+1} = \tilde{s}_{i+1}, d_i = \tilde{d}_i$ .
18:       $m_i = \text{FALSE}$  ▷ We should mask out this transition in training
19:    end if
20:    Add data  $\{s_i, a_i, r_i, s_{i+1}, d_i, m_i\}$  to batch  $\mathcal{D}$ 
21:    Increment time counter  $i \leftarrow i + 1$ 
22:    if  $d_i = \text{TRUE}$  then
23:      if  $\sum_{t \geq \tau^*} r_t \geq \sum_{t \geq \tau^*} \tilde{r}_t$  then ▷ As good as demo
24:         $W \leftarrow W + 1$ 
25:      end if
26:      Sample next starting point  $\tau^*$  uniformly from  $\{\tau - D, \dots, \tau\}$ 
27:      Set time counter  $i \leftarrow \tau^* - K$ 
28:      Reset environment to state  $\tilde{s}_{\tau^*}$ 
29:    end if
30:  end for
31:  Send batch  $\mathcal{D}$  and counter  $W$  to optimizer
32: end while
```

---

---

**Algorithm 2** Optimizer

---

```
1: Input: number of parallel agents  $M$ , starting point shift size  $\Delta$ , success threshold  $\rho$ , initial parameters  $\theta_0$ , demonstration length  $T$ , learning algorithm  $\mathcal{A}$  (e.g. PPO, A3C, Impala, etc.)
2: Set the reset point  $\tau = T$  to the end of the demonstration
3: Start rollout workers  $i = 0, \dots, M - 1$ 
4: while  $\tau > 0$  do
5:   Gather data  $\mathcal{D} = \{\mathcal{D}_0, \dots, \mathcal{D}_{M-1}\}$  from rollout workers
6:   if  $\sum_{\mathcal{D}} [W_i] / \sum_{\mathcal{D}} [d_{i,t}] \geq \rho$  then ▷ The workers are successful sufficiently often
7:      $\tau \leftarrow \tau - \Delta$ 
8:   end if
9:    $\theta \leftarrow \mathcal{A}(\theta, \mathcal{D})$  ▷ Make sure to mask out demo transitions
10:  Broadcast  $\theta, \tau$  to rollout workers
11: end while
```

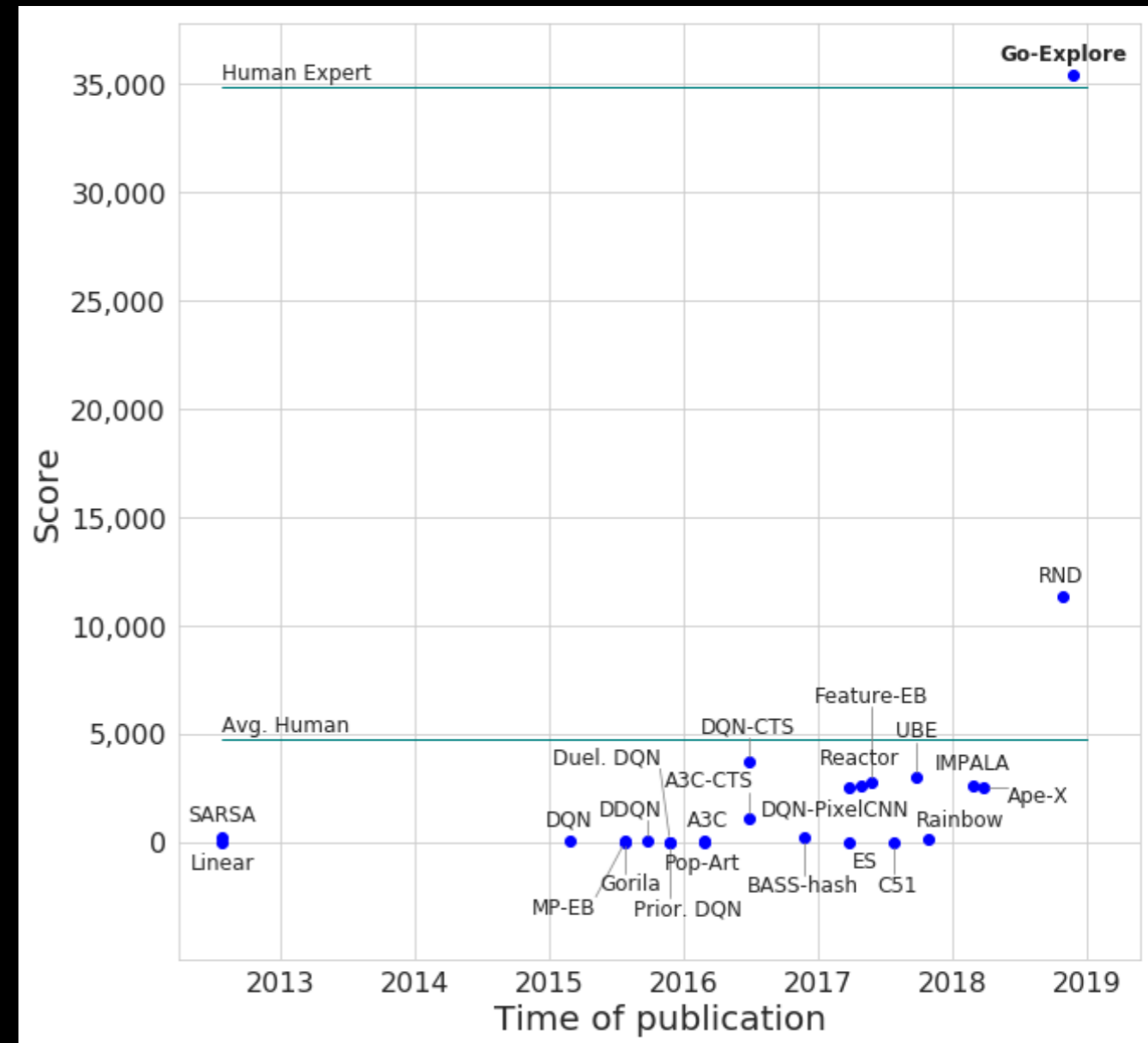
---

Reset time point

Solve the subtask from  $\tau$  till  $T$

# Results with Robust Deep Neural Networks

- 3x previous state of the art!
- Robust to stochasticity
  - random # no-ops up to 30
- No game-specific domain knowledge



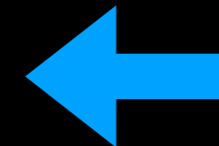
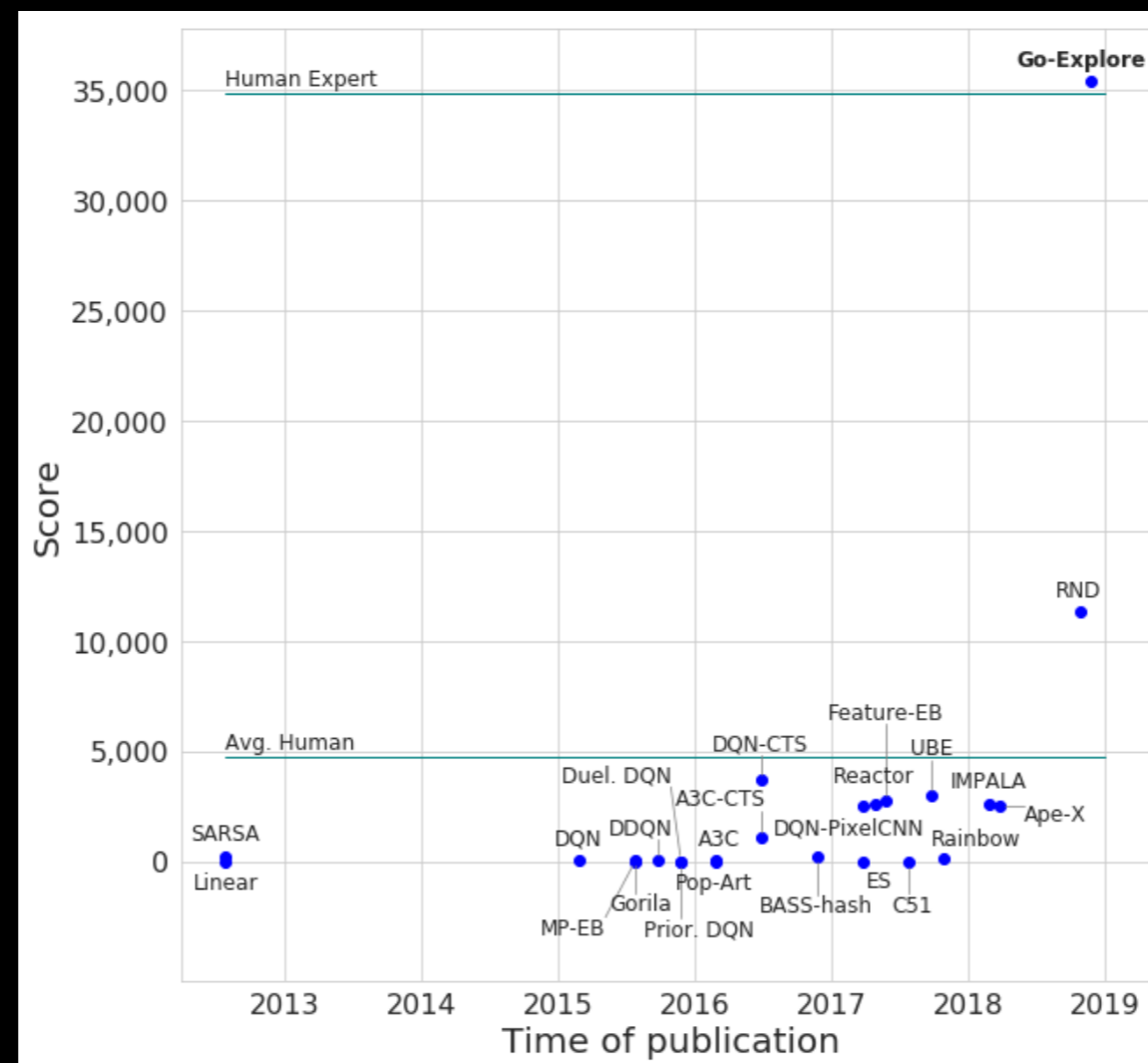
# Go-Explore created a healthy debate

- When should we **require** stochasticity?
  - **test time (only)?**
  - **training time too?**



# Stochasticity at **Test** Time

- Classic way: random number up to 30 no-ops

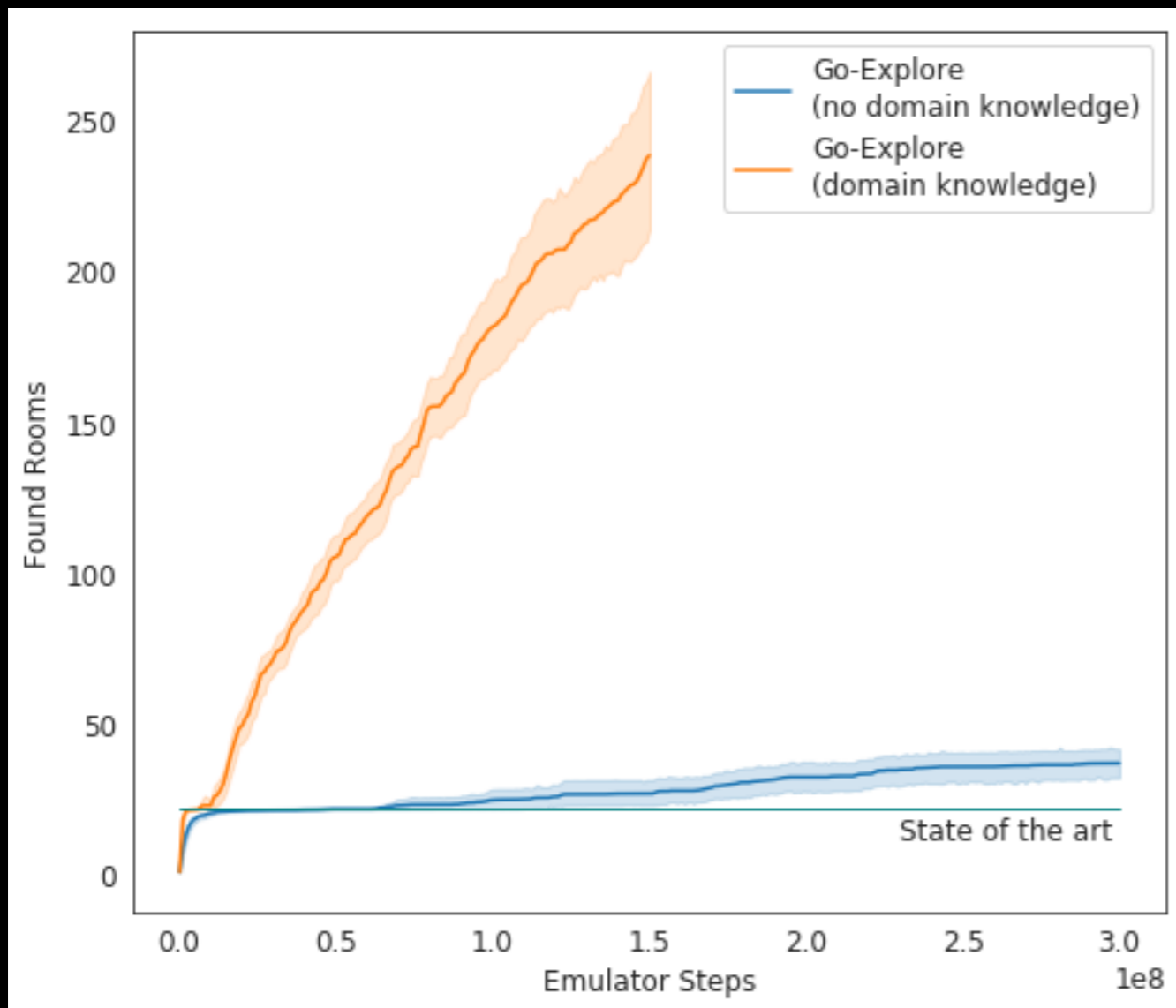


# Adding Domain Knowledge

- Go-Explore can add it via cell representation
  - Important notes:
    - final post-robustification policies still play from pixels only
      - do not consume domain knowledge at eval time
    - wrote simple code to extract info from pixels (not emulator)
  - Montezuma's Revenge
    - unique combinations of: x, y location, room, level, numKeysHeld
  - Pitfall
    - $\langle x, y \rangle$  location, room

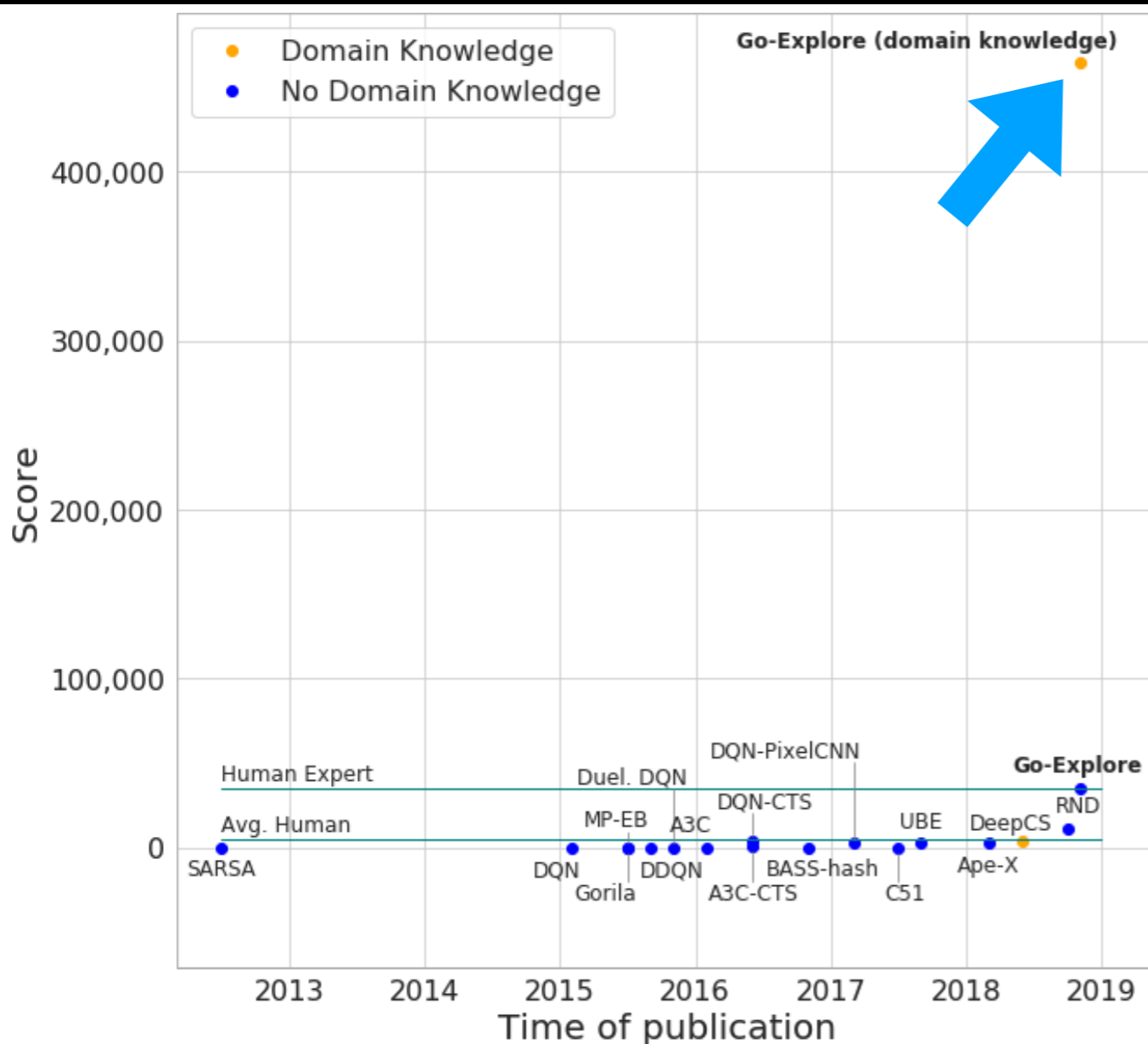
# Montezuma's Results with Domain Knowledge

## Phase 1: Exploration (deterministic eval)



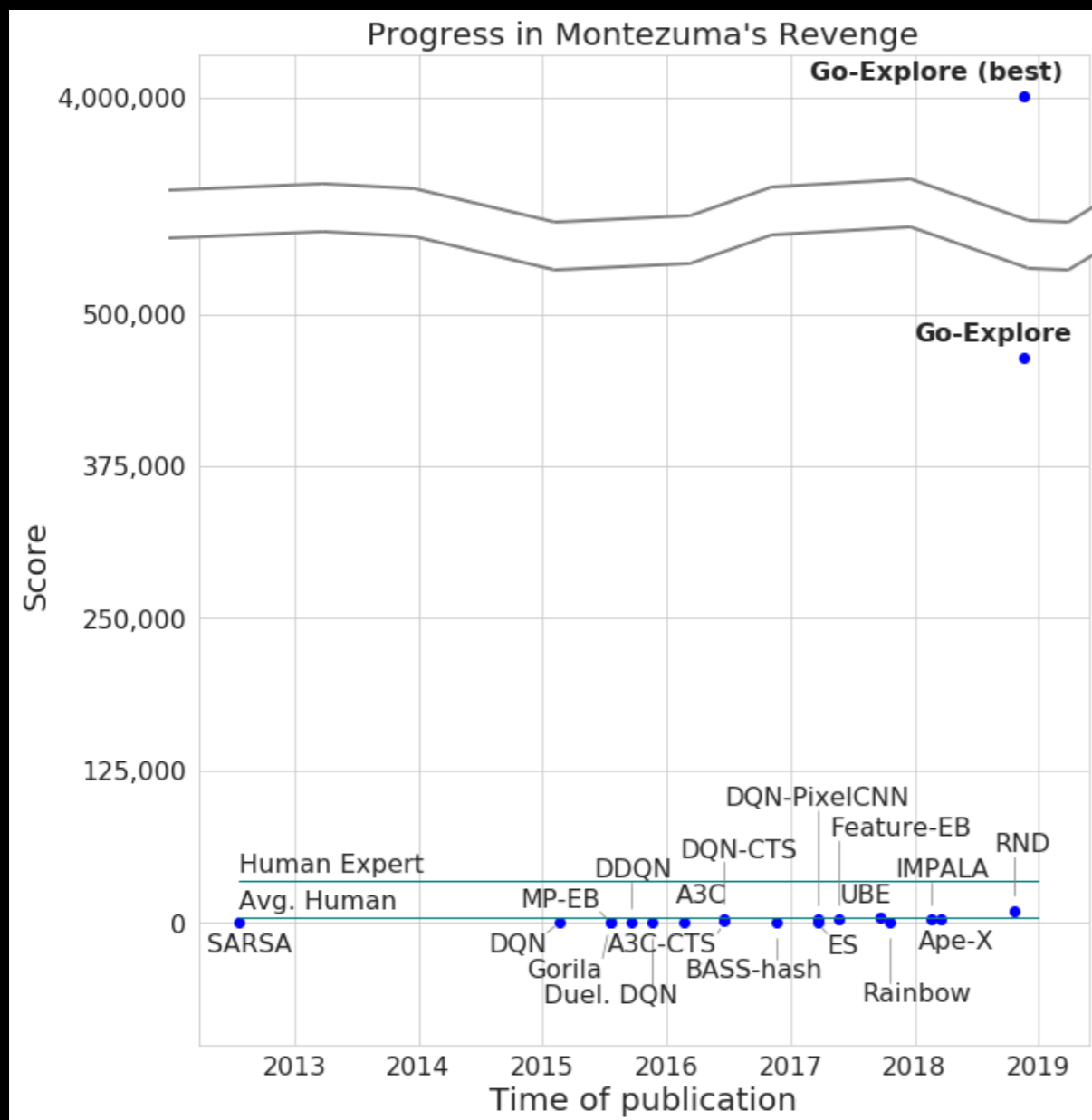
- Solves 9 levels on average
- in half the time

# Results with Robust Deep Neural Networks



- Solves all 3 unique levels
- Levels 3+ nearly identical
  - slight timing differences
  - score changes state/inputs
- On average
  - scores over 469,209!
  - solves 29 levels!
- Sticky action eval:
  - scores 281,264
  - level 18

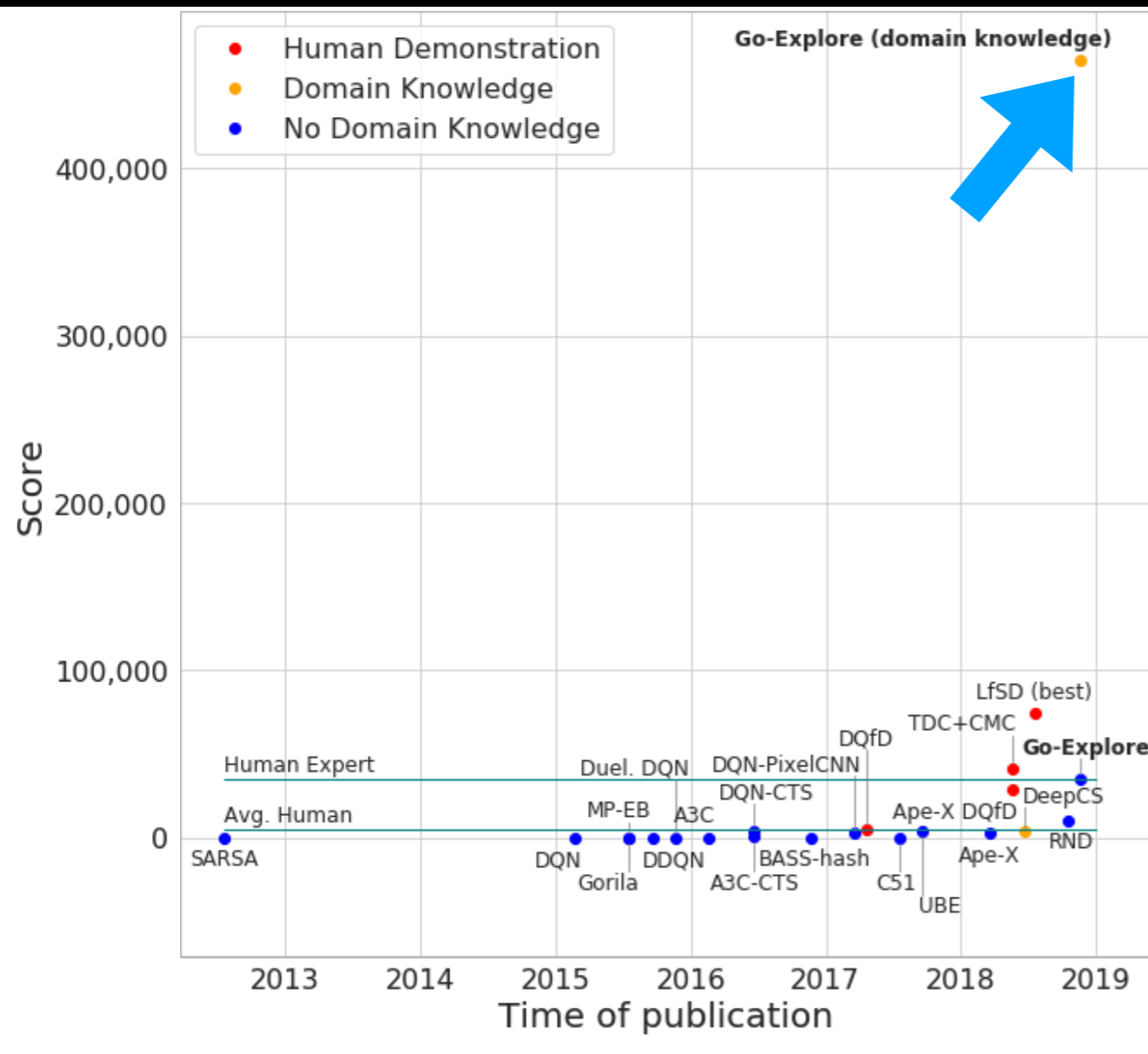
# Results with Robust Deep Neural Networks



- Increased Gym's time limits
- Best neural network
  - scores over 4 million
  - reaches level 295
- Beats human world record
  - 1,219,200
  - achieves strictest definition of "super-human"



# Results with Robust Deep Neural Networks



- Even beats previous work from human demonstrations
  - better demos
  - more demos

# Not Expensive!

- Solving Level 1 of Montezuma's Revenge
  - Phase 1: Exploration
    - ~1 hour!
    - single machine (22 CPUs, 50GB RAM)
    - runs produce ~4GB of data
  - Phase 2: Robustification
    - ~1 day
      - 16 GPUS
- **All told: ~1 day on modest hardware**
  - compare to billions of frames, thousands of workers

## Similar to graph search? (e.g. breadth-first search)

- Yes, but
  1. Go-Explore adds Phase 2: Robustification to handle noise
  2. Such algorithms are intractable in raw high-D state space

# Similar to graph search? (e.g. breadth-first search)

- Yes, but
  1. Go-Explore adds Phase 2: Robustification to handle noise
  2. Such algorithms are intractable in raw high-D state space
    - Go-Explore insight: run these great graph search algorithms in low-D conflated spaces!
      - adds many challenges
        - which cells can you reach from current cell? how (have to search)? can't replace subpaths. etc.
      - requires many algorithmic innovations
      - new research area: porting classic graph algorithms to high-D (conflated representations): BFS, DFS, Dijkstra's, A-Star, etc.