# Introduction to Deep RL, Part 2

Joshua Achiam

OpenAI

February 2, 2019

# 2.1: What is RL currently achieving?

# What RL Can Currently Do: RL in Simulation

If you have infinite simulator data and well-defined rewards, you can make substantial progress on extremely hard problems!

- Atari
- Simulated robotics
- Go (Deepmind's AlphaZero)
- Dota (OpenAI Five)
- Starcraft (Deepmind's AlphaStar)

# Spotlight on AlphaGo

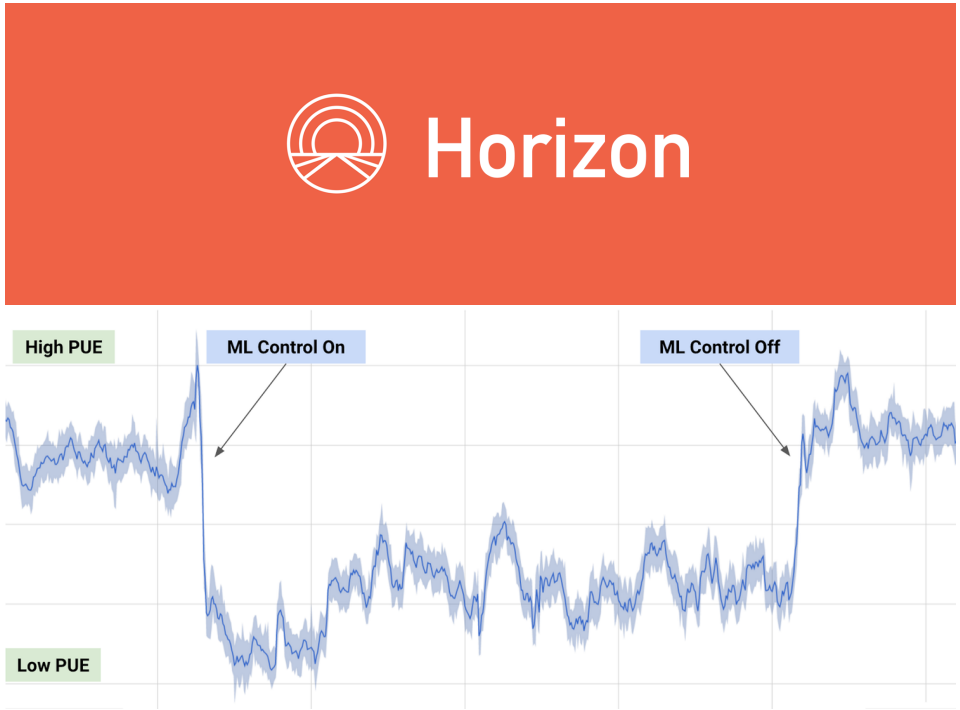

Hard to overstate what an unbelievable accomplishment this was

Previously: Go was considered unassailable stronghold for human experts, AI 10+ years away

# What RL Can Currently Do: RL in the Real World

RL is beginning to see profitable real-world applications!

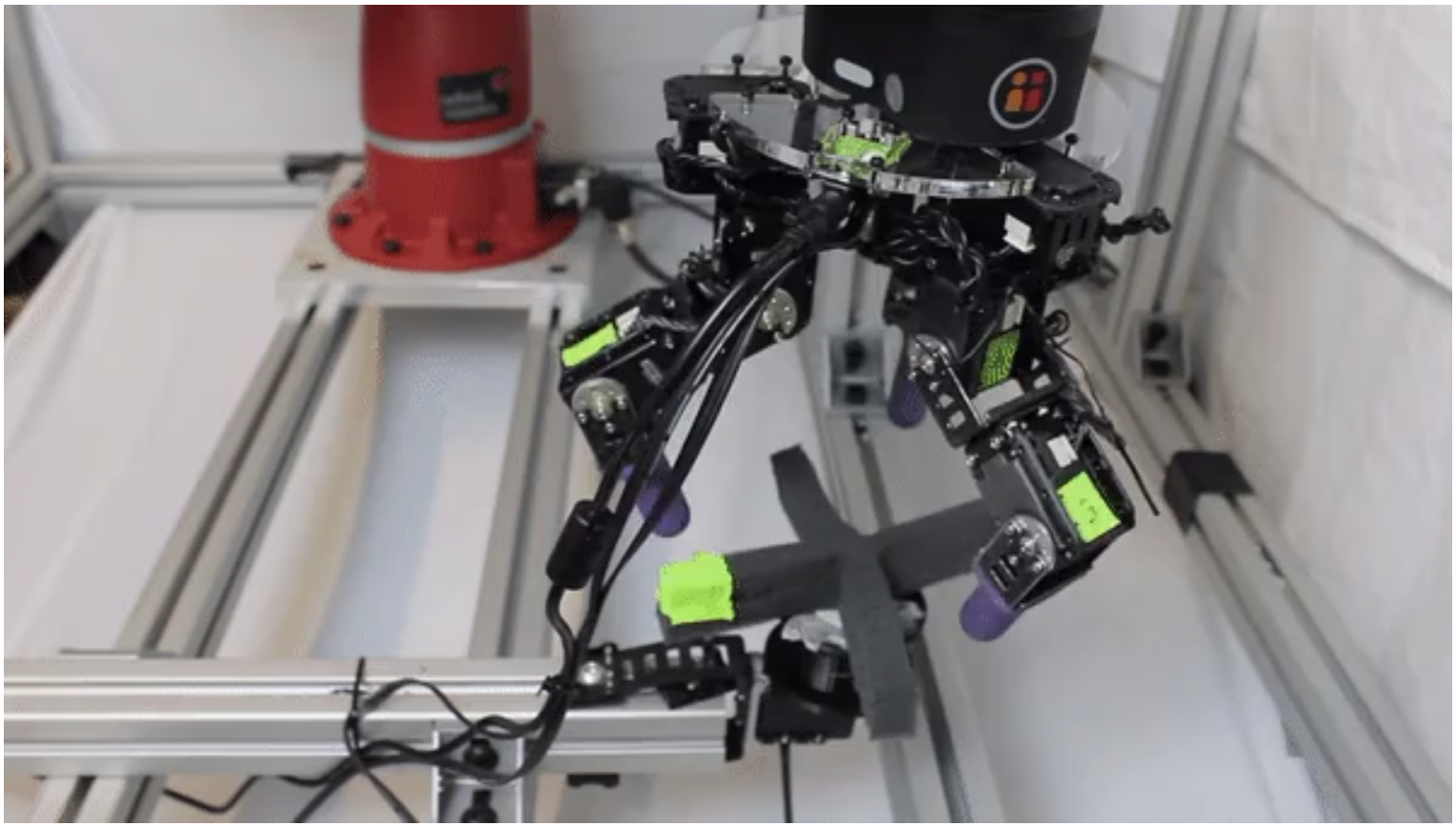
- Facebook uses RL (DQN) for push notifications (Horizon)
- DeepMind integrated RL into data center cooling
- Several promising early efforts at applying RL for robotics



## 2.1.1: Spotlight on RL for Real-World Robotics

# Tasks that can't easily be simulated

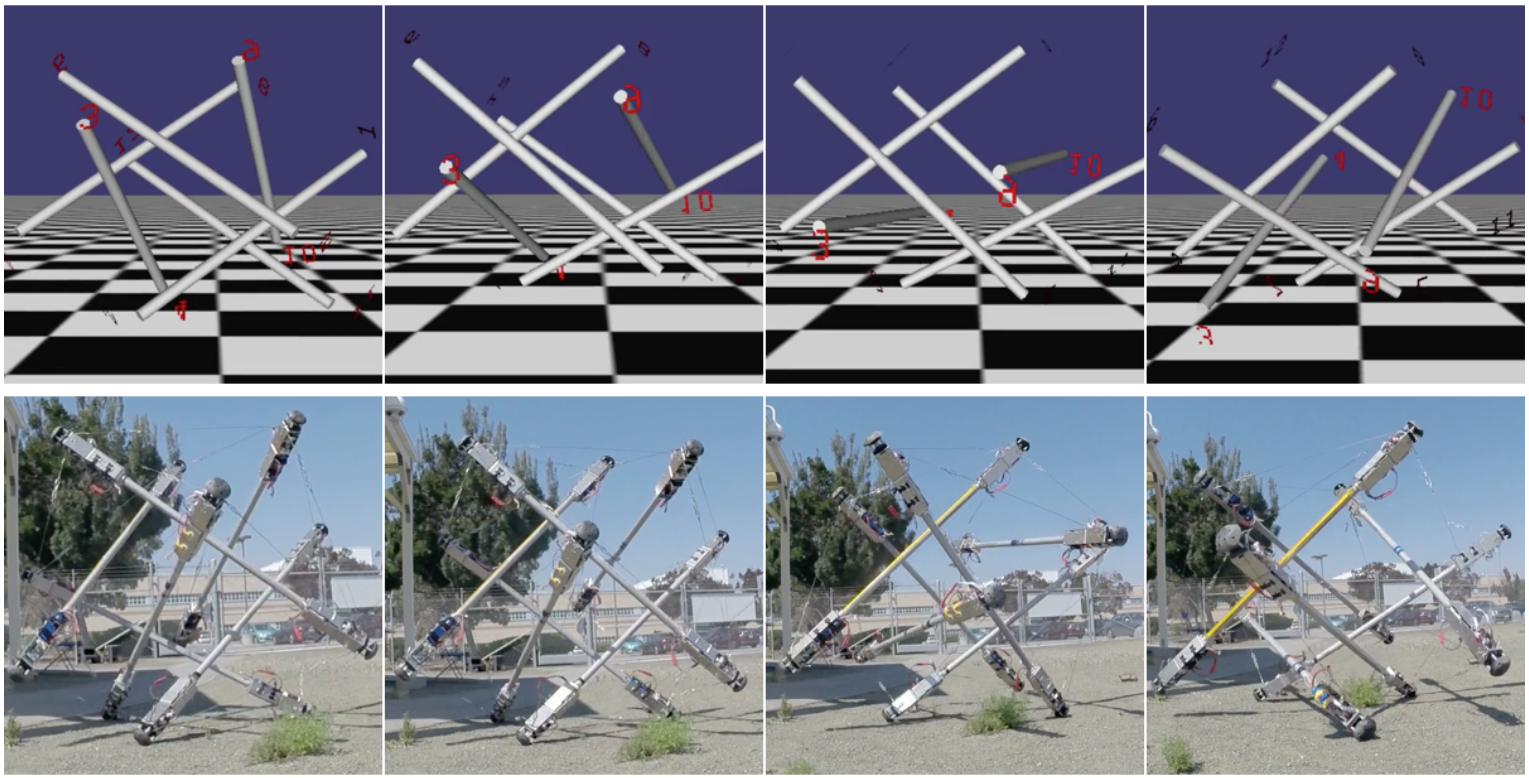
Zhu et al. trained low-cost robots from scratch in the real world on hard-to-simulate tasks using natural policy gradient. (Note: observation space here is hand state and valve state)



[1]Zhu et al, 2018: "Dexterous Manipulation with Deep Reinforcement Learning: Efficient, General, and Low-Cost"

# Robots that are hard to control conventionally

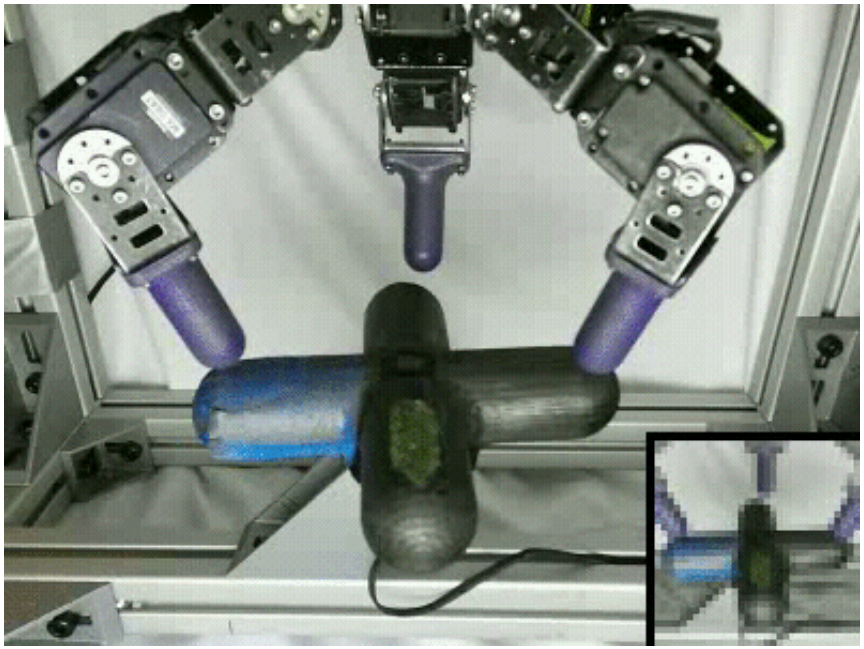
Zhang et al. trained a tensegrity robot with deep RL in simulation and demonstrated transfer to the real world



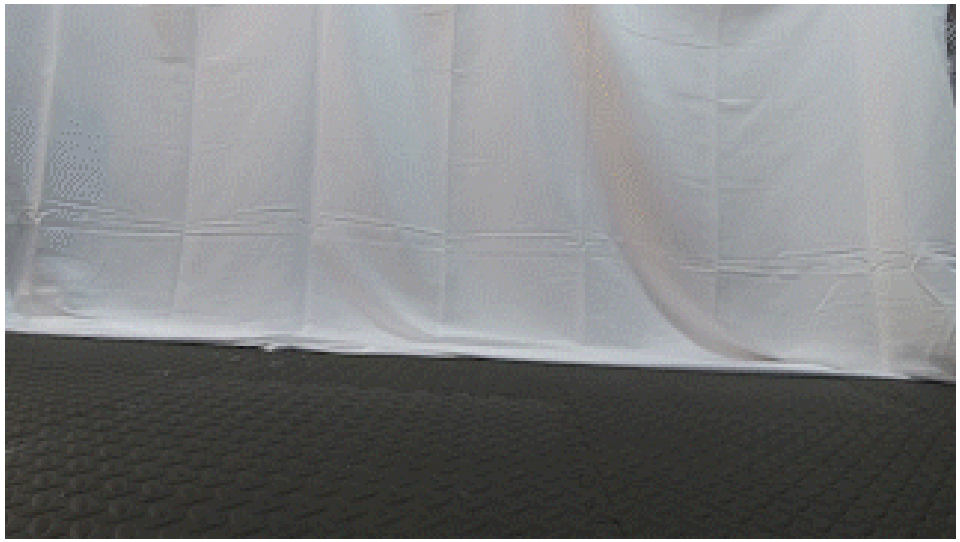
[1] Zhang et al, 2016: "Deep Reinforcement Learning for Tensegrity Robot Locomotion"

# Soft Actor-Critic

Haarnoja et al. trained robots in the real world with Soft Actor-Critic, and demonstrated efficient and robust control on hard domains



Manipulation from visual input, agent sees lower-right (took 20 hours to learn)



Robust control of a legged robot (took 2 hours to learn)

[1] Haarnoja et al, 2018: "Soft Actor-Critic Algorithms and Applications"

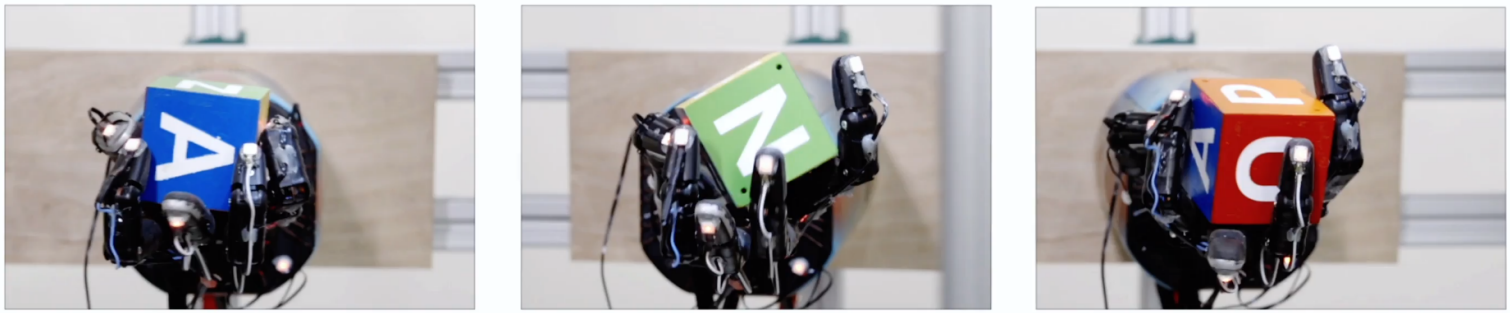
# Anymal

Hwangbo et al. used real data to learn a better simulator, and then used lots of simulator data to train complex locomotion and recovery policies with TRPO for the Anymal robot:


[1]Hwangbo et al, 2018: "Learning agile and dynamic motor skills for legged robots"

# Learning Dexterity

OpenAI et al. trained policies with PPO to dextrously manipulate a complex hand robot, using sim2real by domain randomization:




[1] OpenAI et al, 2018: "Learning Dexterous In-Hand Manipulation"

## 2.2: What are the challenges in modern RL?

# Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?

# Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?

# Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?
- **Generalization**: Policies trained for one task are brittle and cannot be used outside of training environment: how can we make them generalize?

# Some grand challenges for RL

Let's talk about where RL fails, what we need that we aren't getting, and what research is happening.

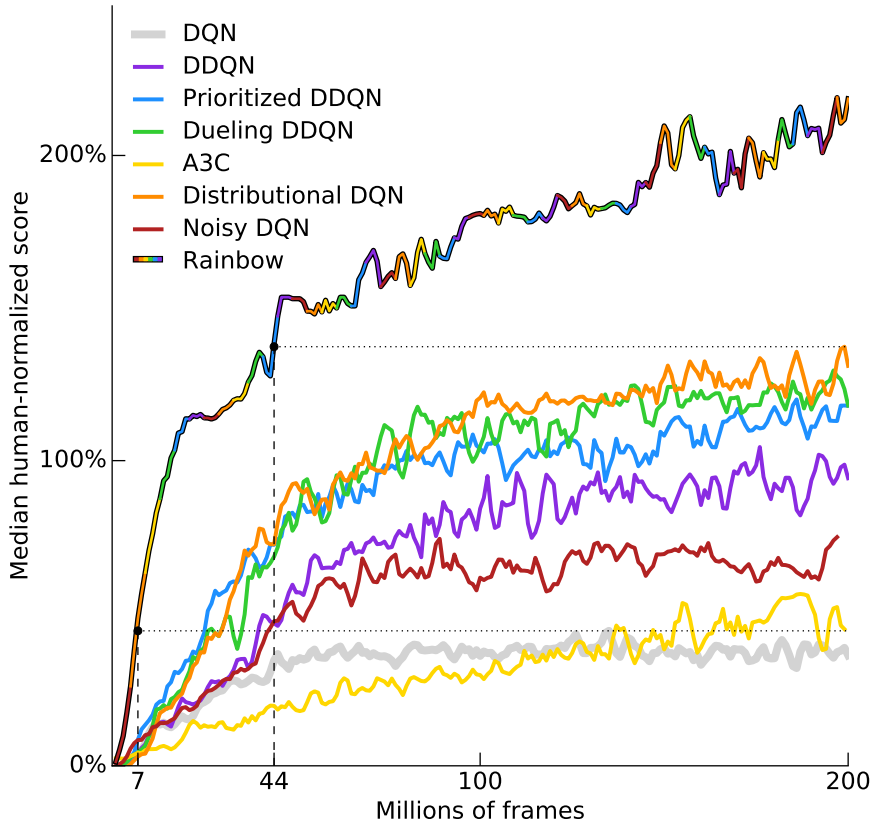
- **Sample efficiency**: Modern deep RL algorithms are very inefficient with data, requiring in some cases 100x or more experience than a human to achieve good performance. How can we make them faster?
- **Exploration**: Modern RL is terrible at environments where rewards are very rare—how can we get agents to try out new things?
- **Generalization**: Policies trained for one task are brittle and cannot be used outside of training environment: how can we make them generalize?
- **Safety**: How can we make sure that agents trained by deep RL behave in ways consistent with human preferences?

## 2.2.1: Research on Improving Sample Efficiency

# Combining Model-Free Improvements

**Rainbow DQN**: combines...

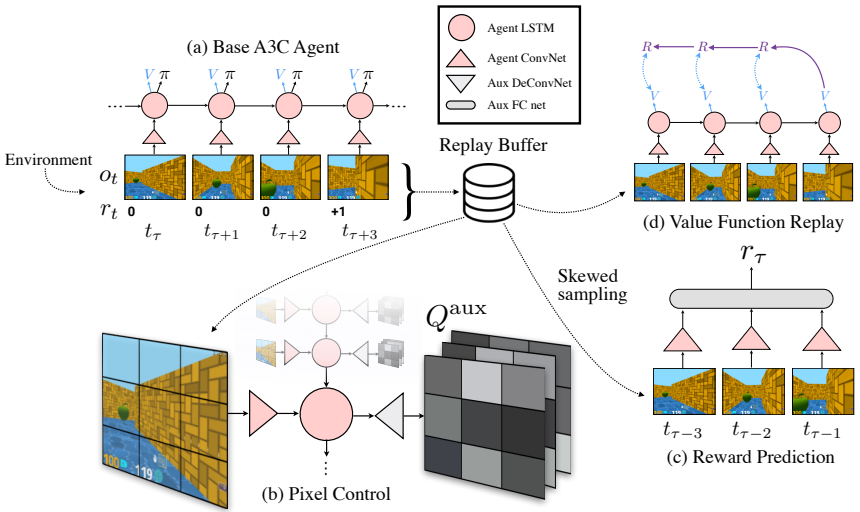
- Dueling architecture
- Double Q-Learning
- Prioritized experience replay
- N-step Q-Learning
- Distributional Q-Learning
- Parameter-space noise

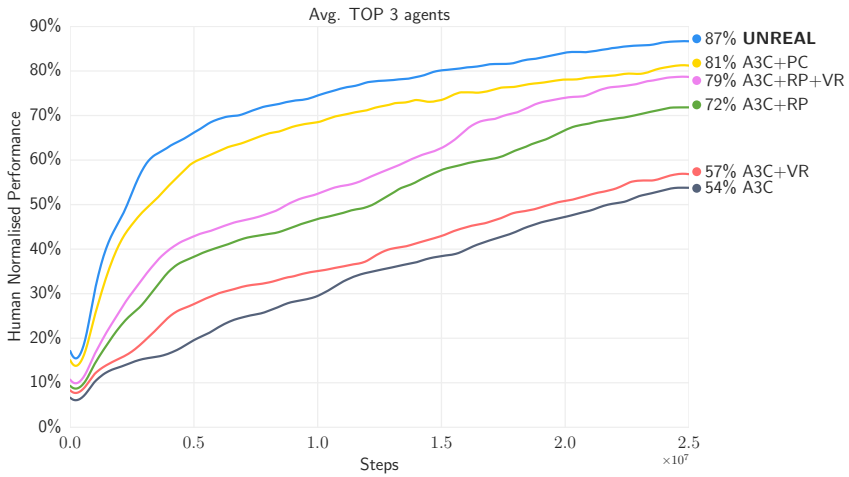



[1]Hessel et al, 2017: "Rainbow: Combining Improvements in Deep Reinforcement Learning"

# Accelerating Feature Learning with Unsupervised Learning

**UNREAL** uses various unsupervised auxilliary tasks to speed up learning:

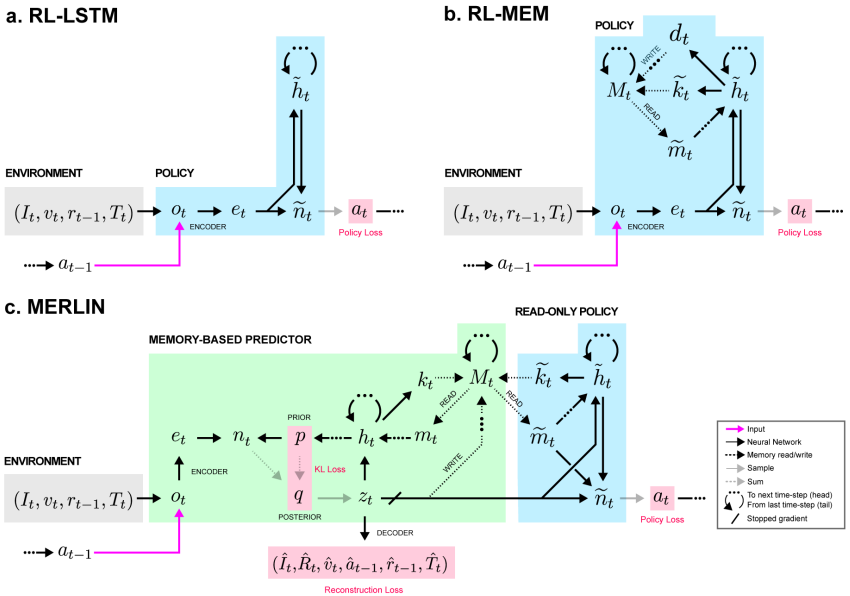


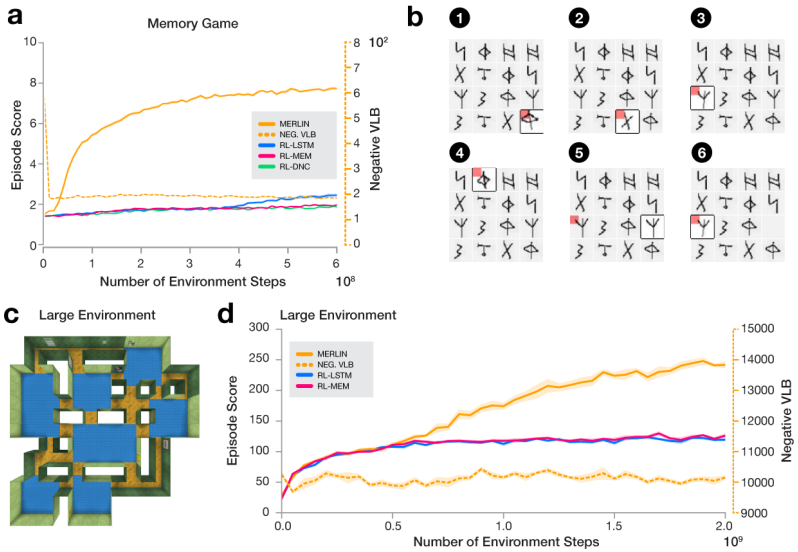




[1] Jaderberg et al, 2016: "Reinforcement Learning with Unsupervised Auxiliary Tasks"

# Combining with Memory Mechanisms

**MERLIN** combines unsupervised learning and attention-based memory to improve over baseline architectures:



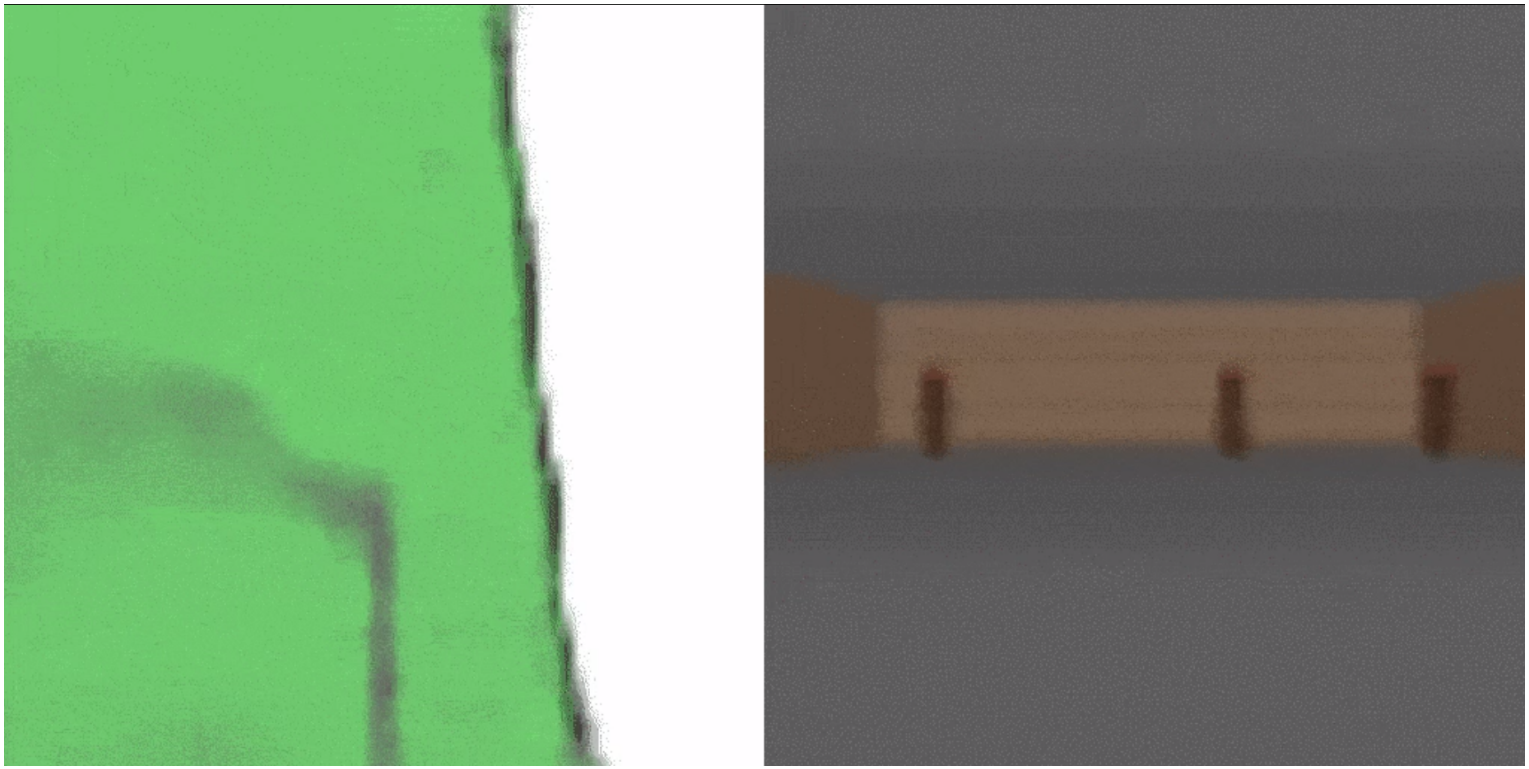




Note: this avenue may not help on arbitrary problems, but seems likely to help on sophisticated partially-observed ones

[1]Wayne et al, 2018: "Unsupervised Predictive Memory in a Goal-Directed Agent"

# Model-Based RL

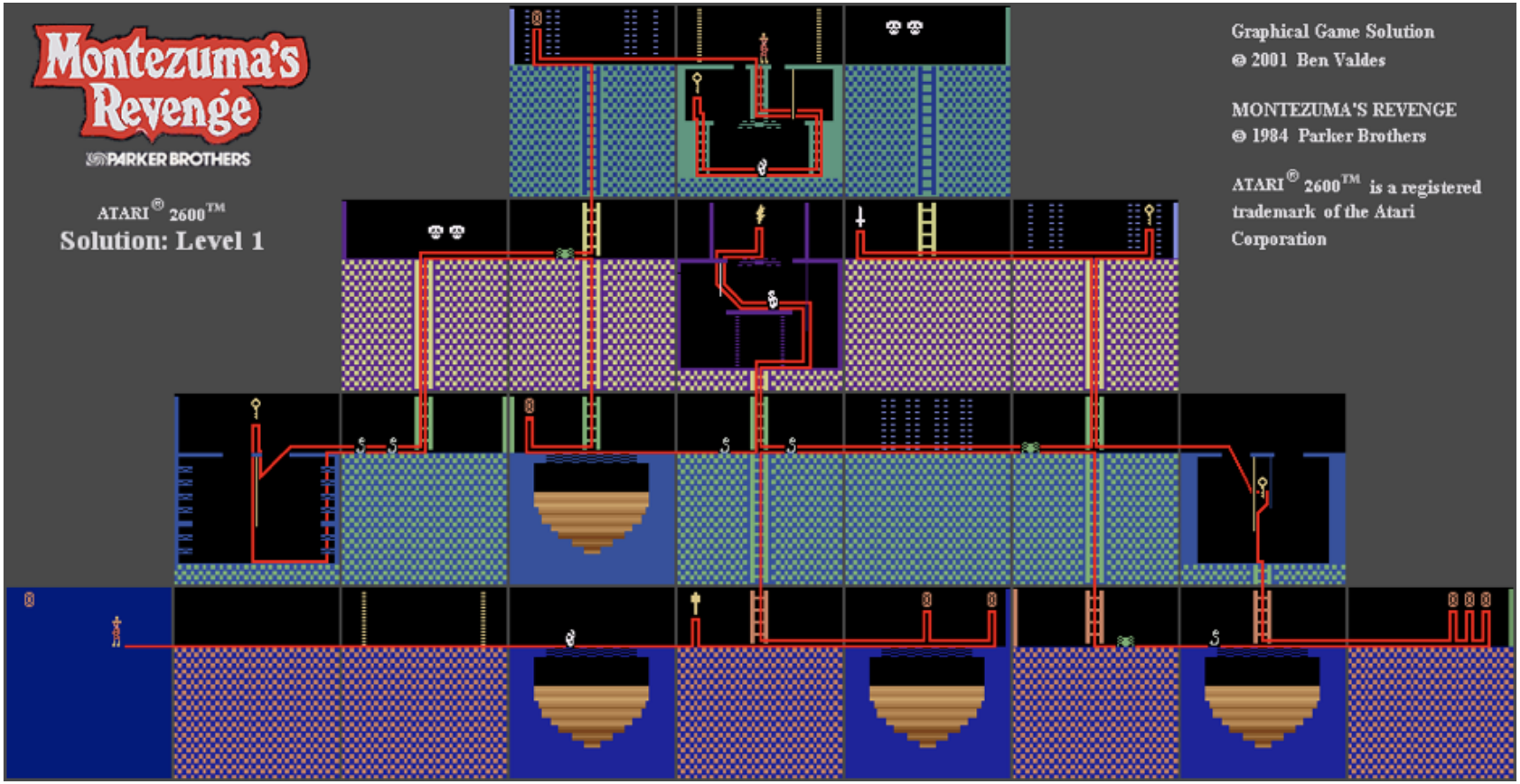
Techniques like **World Models**, that allow an agent to learn from simulated experience, are another way to bring down the needed number of real samples:



[1] Ha and Schmidhuber, 2018: "World Models"

# 2.2.2: Research on Improving Exploration

# AKA, the Quest to Solve Montezuma's Revenge




- Famously hard Atari game
- Most RL algorithms do terribly
- Very sparse rewards, many actions lead to death
- But humans can do just fine!

# Intrinsic Motivation

Various **intrinsic motivation** schemes exist, which work by adding a non-environment-dependent reward:

$$R(s, a, s') \rightarrow R(s, a, s') + \eta R_{int}(s, a, s')$$

Usually, intrinsic reward prefers **new experiences**. Examples:

- Pseudocount-based methods[1]: approximates number of state visitations with density model, rewards are:

$$R_{int}(s, a, s') \propto \frac{1}{\sqrt{N(s)}}$$

- Information gain[2]: approximates change in info state about environment model based on new experience

$$R_{int}(s, a, s') \approx D_{KL}(P(\mathcal{E}|\mathcal{D}_t, s_{t+1})||P(\mathcal{E}|\mathcal{D}_t))$$

[1] Bellemare et al, 2016: "Unifying Count-Based Exploration and Intrinsic Motivation"
[2] Houthooft et al, 2016: "Variational Information Maximizing Exploration"

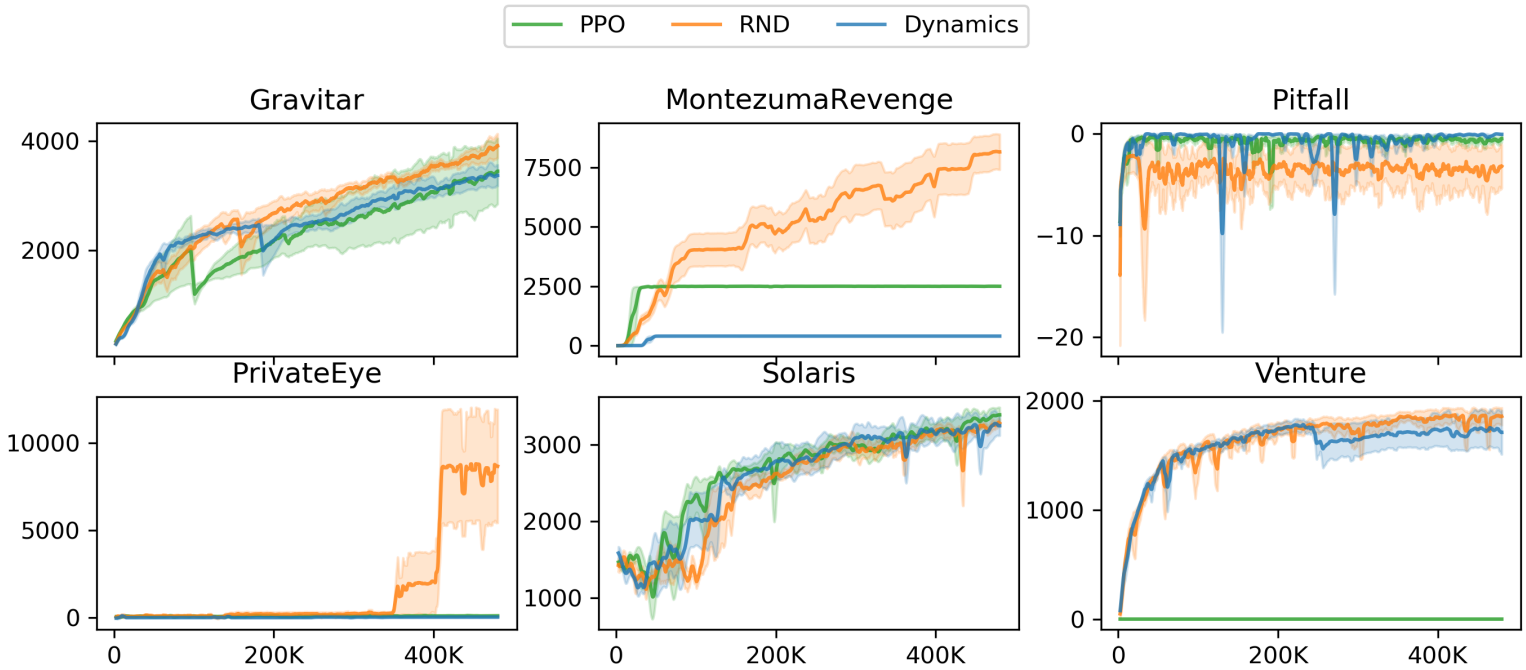
# Random Network Distillation

**Random Network Distillation**[3]: A new and appealingly simple form of intrinsic motivation:

$$R_{int}(s) = \|f_\phi(s) - f_\xi(s)\|,$$

where $f_\phi$ is a random neural network (the target) and $f_\xi$ is trained to match $f_\phi$

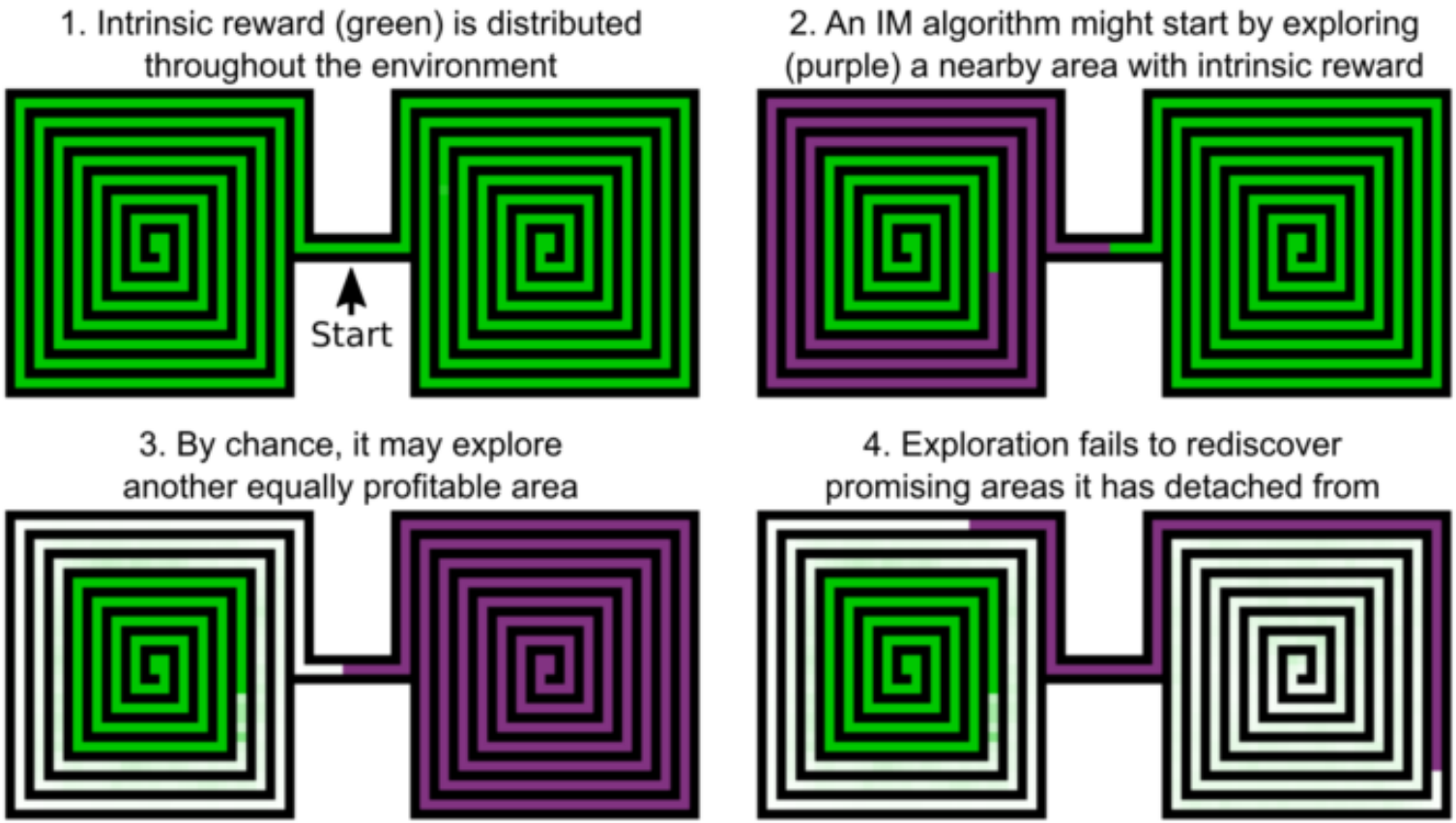
- Exploits generalization properties of networks: $f_\phi$ matches $f_\xi$ on old states but not new states!



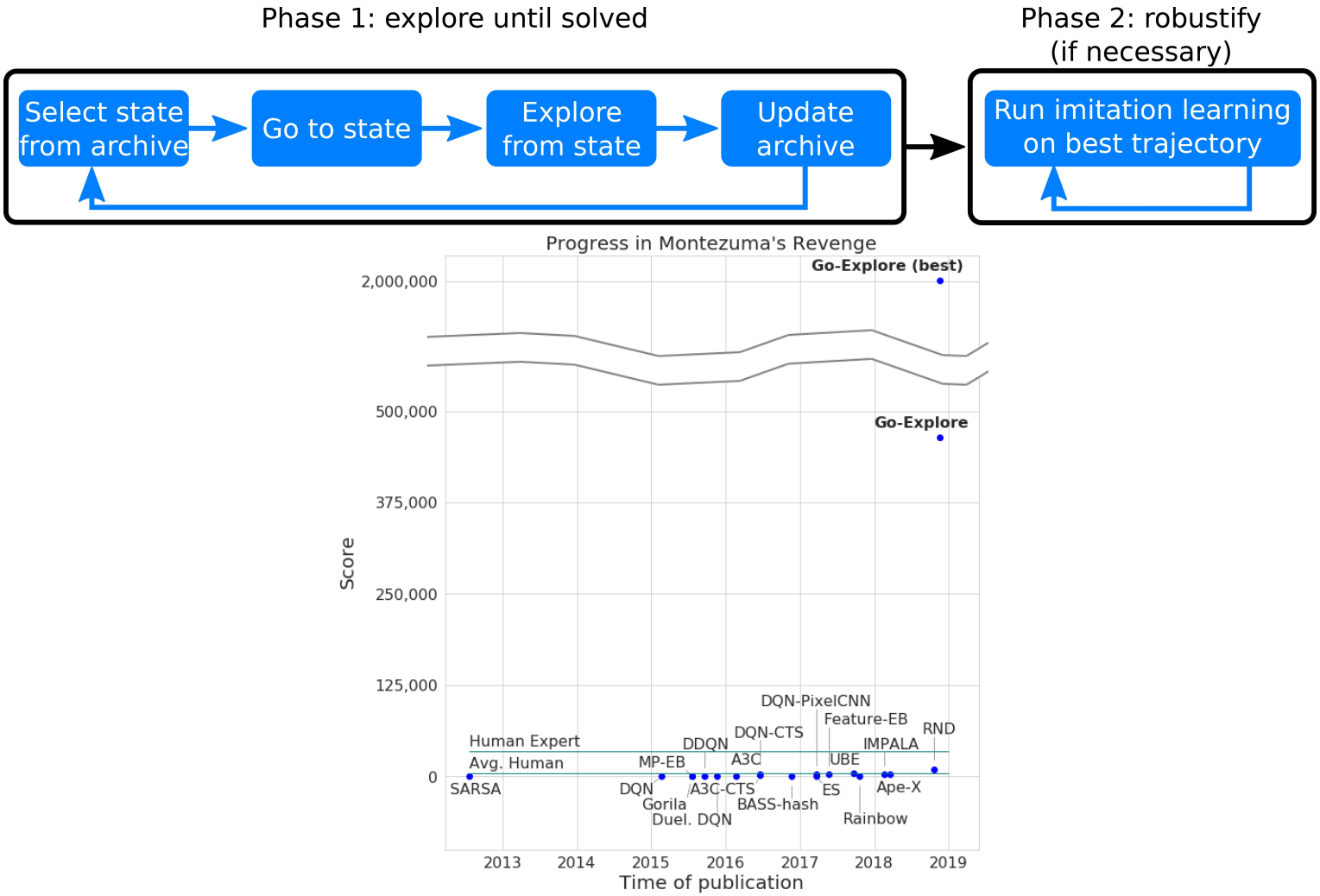

[3] Burda et al, 2018: "Exploration by Random Network Distillation"

# Go-Explore

Ecoffet et al, 2018 identify a problem with intrinsic motivation that they call **detachment**:



They rectify detachment with **Go-Explore**[4] algorithm:




[4] Ecoffet et al, 2018: "Go-Explore: a New Approach for Hard-Exploration Problems"

# 2.3: Spinning Up in Deep RL

# Getting started

If you haven't already:

- Brush up on math! Be familiar with vectors, matrices, gradients, gradient descent, random variables, expectations, variance, and a few other things.
- Brush up on deep learning! **As much as you can get.**
- Become familiar with TF or PyTorch! **You need to tinker to learn.**
- Stay fresh on RL basics! **Fundamentals go a long way.**