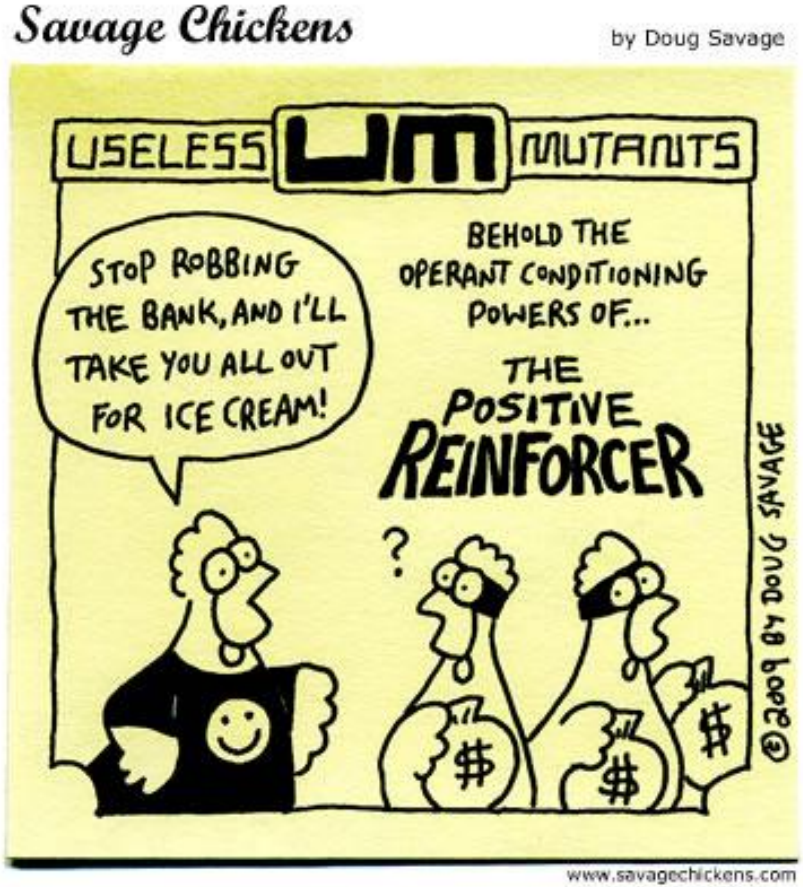

Savage Chickens
by Doug Savage
USELESS UM MUTANTS
STOP ROBBING THE BANK, AND I'LL TAKE YOU ALL OUT FOR ICE CREAM!
BEHOLD THE OPERANT CONDITIONING POWERS OF...
THE POSITIVE REINFORCER
?
©2009 BY DOUG SAVAGE
www.savagechickens.com

CIS 522 Lecture 19

Deep Reinforcement Learning III
4/2/20

Penn Engineering

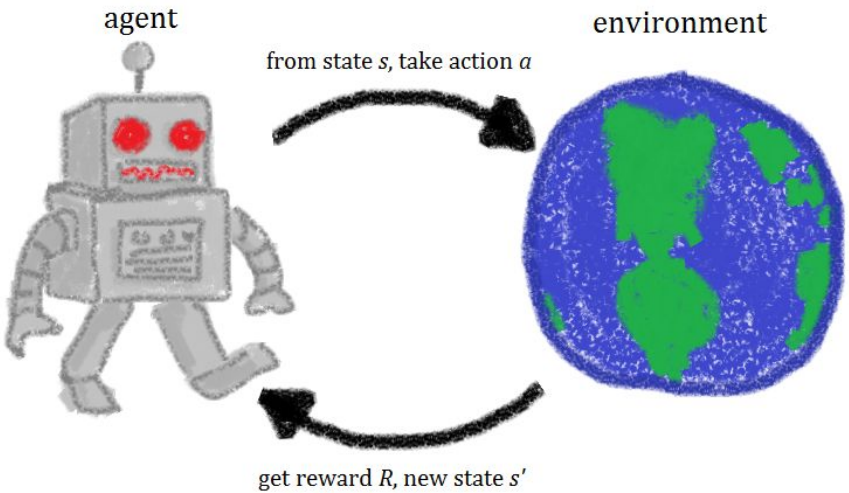
Today

- Challenges in deep RL
- Survey of current research topics
- Overview of other approaches in learning agents

Challenges in deep reinforcement learning

(Deep) RL

An ambitious goal: train an intelligent decision maker



(Deep) RL challenges

Still a number of challenges to be overcome before this goal is realized:

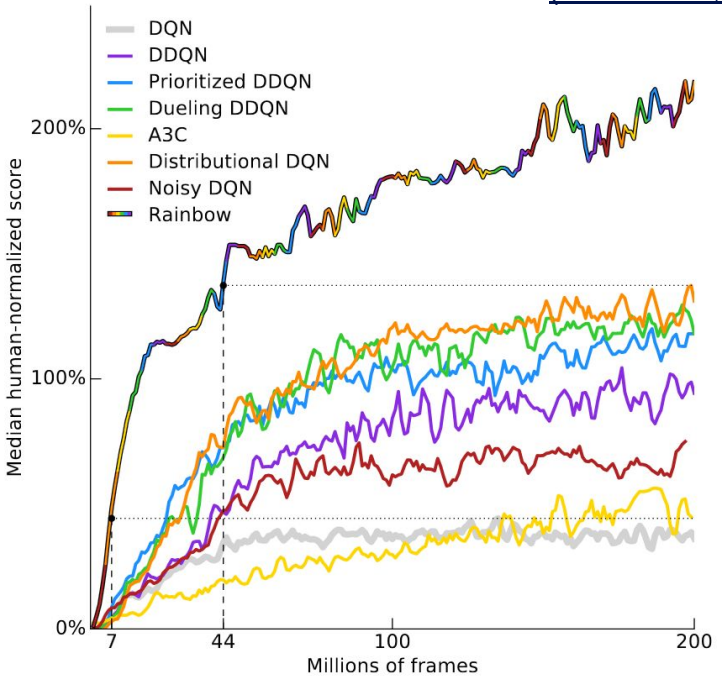
1. Sample efficiency
2. Sparse rewards/reward function design
3. Local optima
4. Generalization -- overfit to environments
5. Unstable and hard to reproduce

Other approaches (e.g. supervised algorithms, genetic algorithms) work better in many cases

1. Learning efficiency

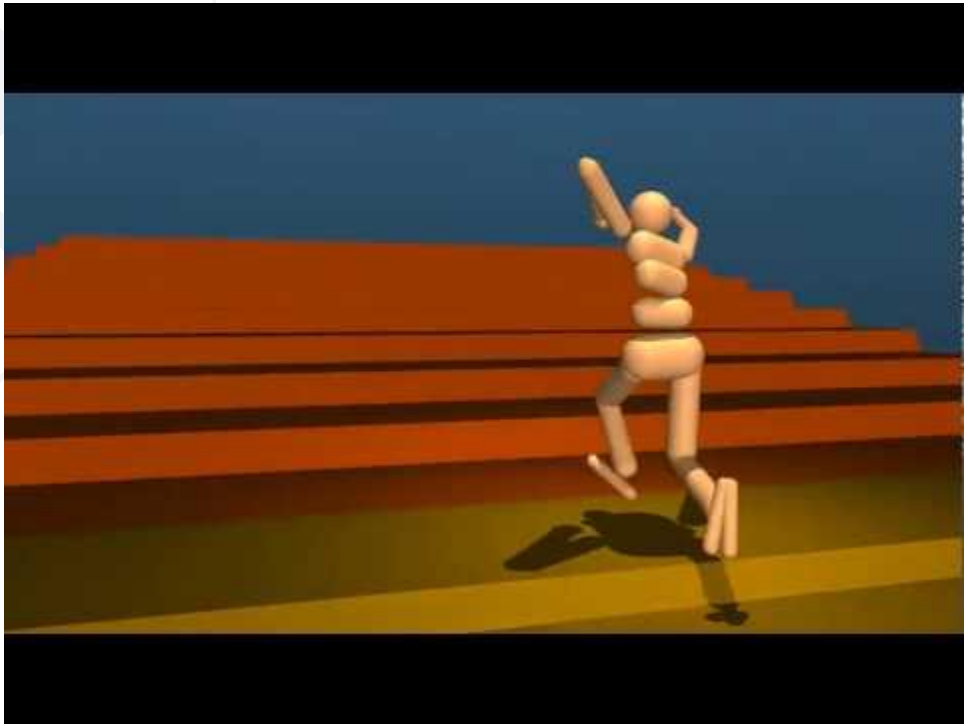
- Rainbow DQN (2017) efficiency: 18m frames to reach human performance (83 hours of play)

(Hessel et al, 2017)



1. Learning efficiency

- Parkour paper (2017)[1]: 6400 CPU hours to train

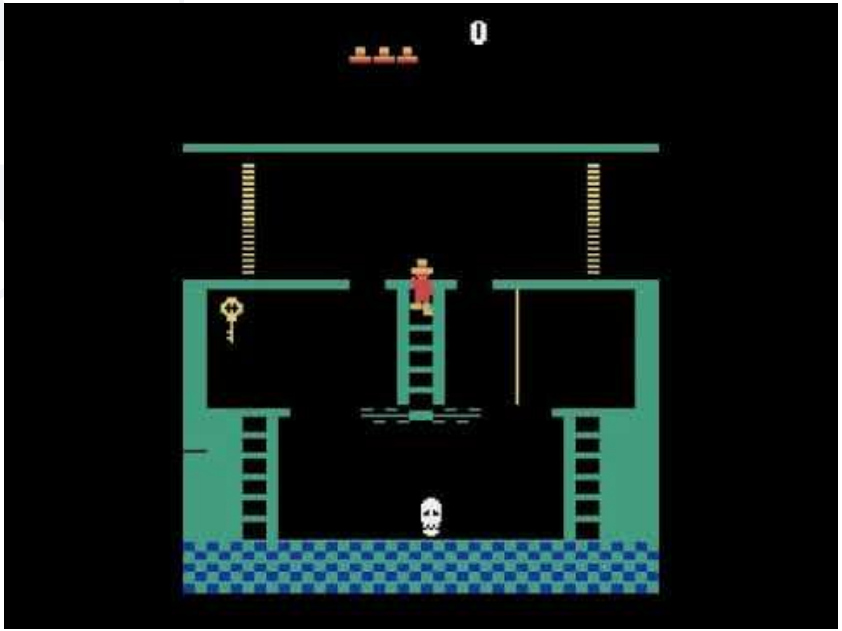


1. Learning efficiency

- AlphaStar (2019)[1]: competitive play at Starcraft, after 200 years of human experience in training

2. Sparse rewards

- How to efficiently explore when only rewarded at end of level?
- DQN playing Montezuma's revenge:



2. Sparse rewards

Can try ‘reward shaping’: (Coast Runners 7)

https://openai.com/blog/faulty-reward-functions/

3. Local optima

- Can easily reach sub-optimal solutions
- The explore-exploit problem again

https://www.alexirpan.com/2018/02/14/rl-hard.html

4. Generalization

- A trained model rarely extrapolates to any other task
- A3C agent trained to play Breakout, tested on small modifications:

Schema networks (from Vicarious AI)[1]

5. Other issues

- Reproducibility
- Stability
- Challenges in coding/implementing:
  - Many training steps + hyperparameters to tune + repeated runs to get statistical significance
  - Very close nursing on ‘toy’ problems
  - Code can fail silently

5. Other issues: reproducibility

- Key hyperparameters may be unreported
- Challenging to reproduce results:

"When I started working at Google Brain, one of the first things I did was implement the algorithm from the Normalized Advantage Function paper. I figured it would only take me about 2-3 weeks. I had several things going for me: some familiarity with Theano (which transferred to TensorFlow well), some deep RL experience, and the first author of the NAF paper was interning at Brain, so I could bug him with questions."

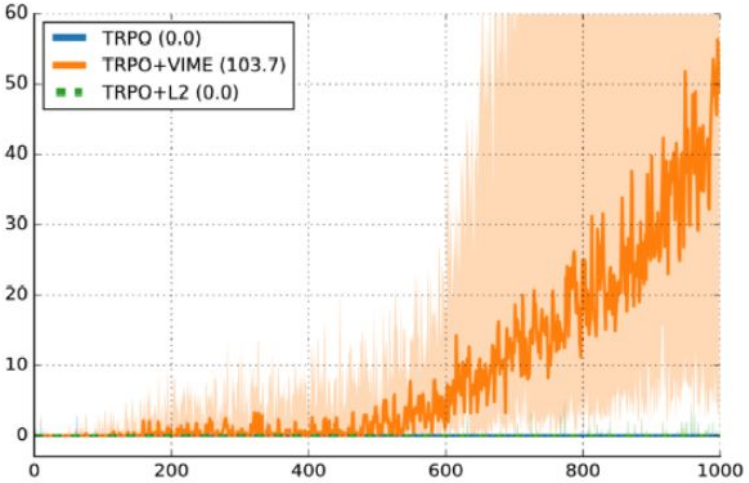
"...This ended up taking six weeks."

Alex Irpan

https://www.alexirpan.com/2018/02/14/rl-hard.html

5. Other issues: stability

- Even for fixed hyperparameters, performance can vary wildly between runs
- Performance on Mujoco task HalfCheetah (2D Cheetah robot)



"Variational Information Maximizing Exploration" (Houthooft et al, NIPS 2016)

Applications

- (Deep) RL currently has few industrial applications
- Still a very active area of research
- Current or potential applications:
  - GPU device management: Mirhoseini et al 2017
  - Data-center power management[1]
  - Finance?
  - Personalized medicine
    - Sepsis treatment in ICUs: Raghu et al 2017
      - Patients respond very differently to treatment, dynamic treatment plan
    - Multi-armed bandits (a sub-field of RL that focuses on exploration-exploitation in a simplified setting)

Current topics in deep RL

(Some) Current topics in deep RL

- Sample efficiency*:
  - Combining model-based and model-free RL
  - Sim2real
- Dealing with sparse rewards
- Generalization:
  - Meta-learning

* A loose categorization of topics

Learning efficiency

- Knowing or learning the dynamics can allow internal simulation, reducing number of real-world samples required
- Combined model-based and model-free RL can be fruitful

Imagination-Augmented Agents for Deep Reinforcement Learning

Théophane Weber* Sébastien Racanière* David P. Reichert* Lars Buesing
Arthur Guez Danilo Rezende Adria Puigdomènech Badia Oriol Vinyals
Nicolas Heess Yujia Li Razvan Pascanu Peter Battaglia
Demis Hassabis David Silver Daan Wierstra
DeepMind

Abstract

We introduce Imagination-Augmented Agents (I2As), a novel architecture for deep reinforcement learning combining model-free and model-based aspects. In contrast to most existing model-based reinforcement learning and planning methods, which prescribe how a model should be used to arrive at a policy, I2As learn to interpret predictions from a learned environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in deep policy networks. I2As show improved data efficiency, performance, and robustness to model misspecification compared to several baselines.

Learning efficiency

1. Learn environment model (a supervised learning problem: $f(s_t, a_t) \rightarrow s_{t+1}$)
2. Learn encoding of environment roll-outs that usefully augment a model-free algorithm

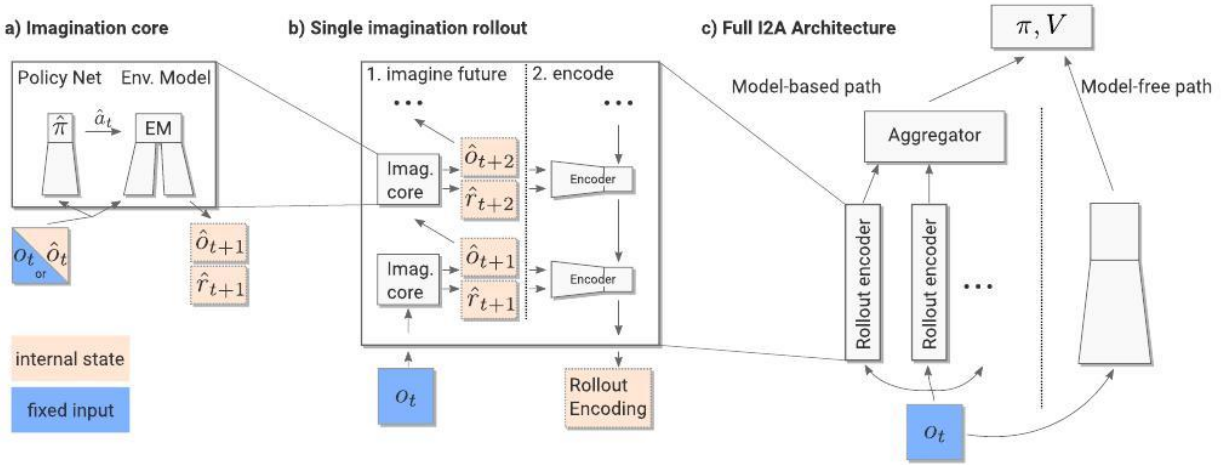


Figure 1: *I2A architecture*. $\hat{\cdot}$ notation indicates imagined quantities. *a)*: the imagination core (IC) predicts the next time step conditioned on an action sampled from the rollout policy $\hat{\pi}$. *b)*: the IC imagines trajectories of features $\hat{f} = (\hat{o}, \hat{r})$, encoded by the rollout encoder. *c)*: in the full I2A, aggregated rollout encodings and input from a model-free path determine the output policy π.

Learning efficiency

- Improves performance on games that require multi-step planning (and where steps may be irreversible) -- block manipulation game Sokoban

Figure 3: *Random examples of procedurally generated Sokoban levels.* The player (green sprite) needs to push all 4 boxes onto the red target squares to solve a level, while avoiding irreversible mistakes. Our agents receive sprite graphics (shown above) as observations.

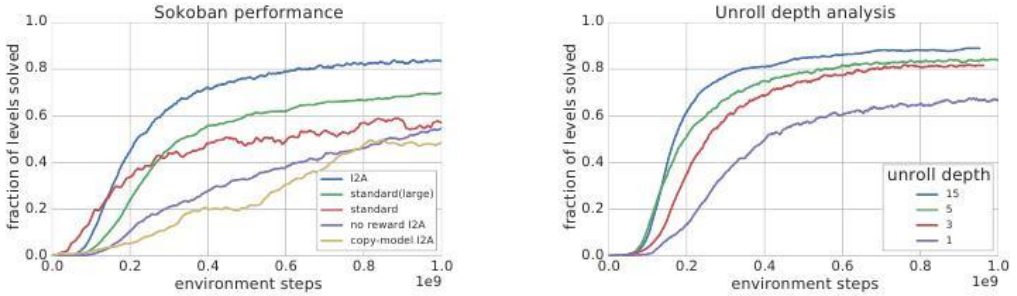
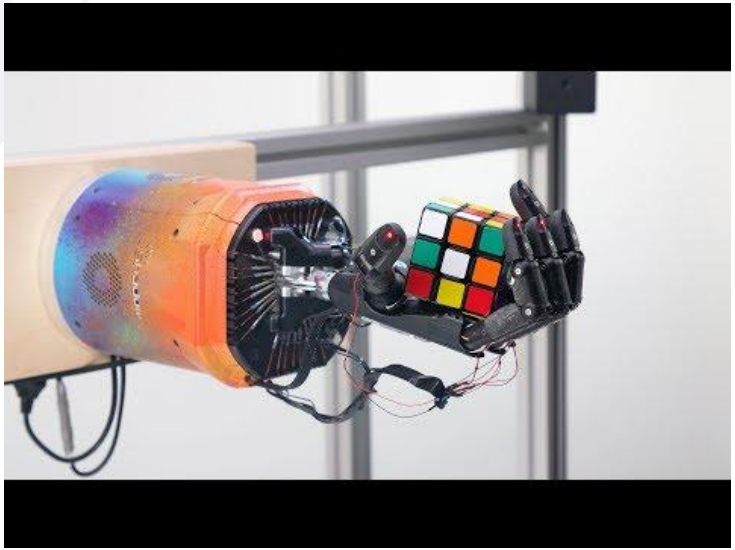


Figure 4: *Sokoban learning curves. Left:* training curves of I2A and baselines. Note that I2A use additional environment observations to pretrain the environment model, see main text for discussion. *Right:* I2A training curves for various values of imagination depth.

Sim2real

- Using simulated experience, learn and transfer to the real setting
- OpenAI manipulation of the Rubik's cube[1]
- Steps to solve the cube are given by a symbolic solving algorithm (this is only solving the cube manipulation… still very difficult!)



Sim2real: OpenAI's Rubik's cube solver

- Uses a standard RL algorithm: Proximal policy optimization (PPO)
- Maximize change in expected return, while minimizing change in policy
- A default choice in many deep RL problems

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta) \right],$$

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right),$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

Advantage function: (reward from this action, minus baseline)

$$A(s_t, a_a) = Q(s_t, a_t) - V(s_t)$$

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

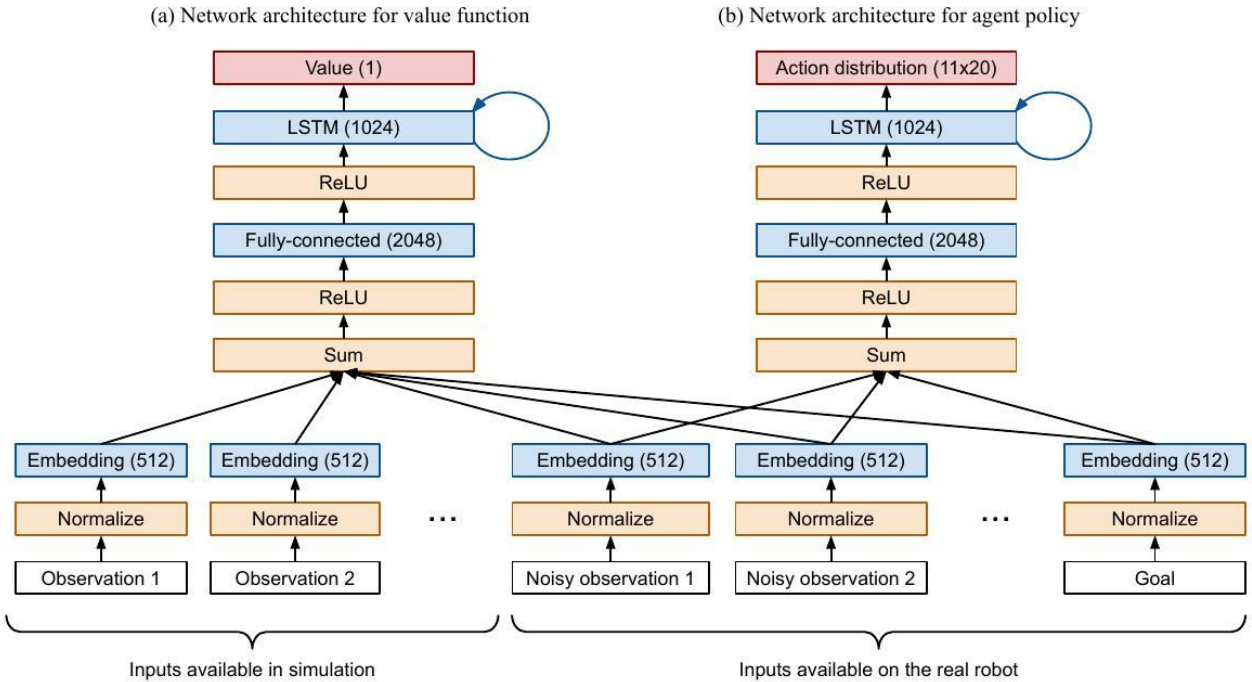
typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Sim2real: OpenAI's Rubik's cube solver

- Large policy and value function networks:



Akkaya et al 2019

Sim2real: OpenAI's Rubik's cube solver

- Large (separately trained) vision network to extract face angles/positions:

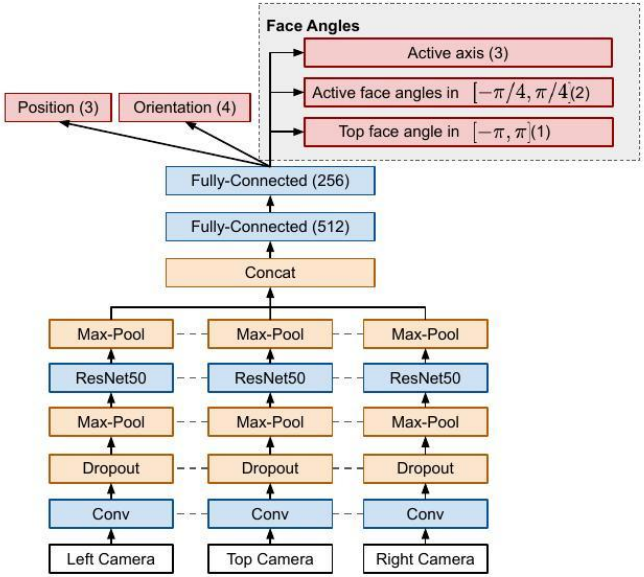
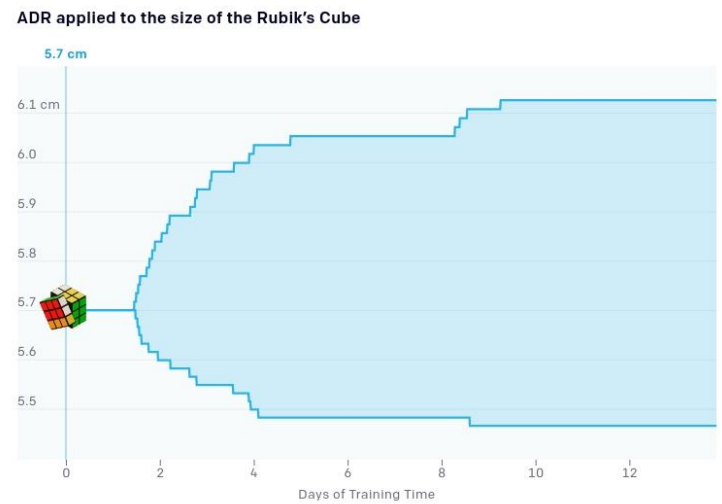


Figure 14: Vision model architecture, which is largely built upon a ResNet50 [43] backbone. Network weights are shared across the three camera frames, as indicated by the dashed line. Our model produces the position, orientation, and a specific representation of the six face angles of the Rubik's cube. We specify ranges with $[\ldots]$ and dimensionality with $(\ldots)$.

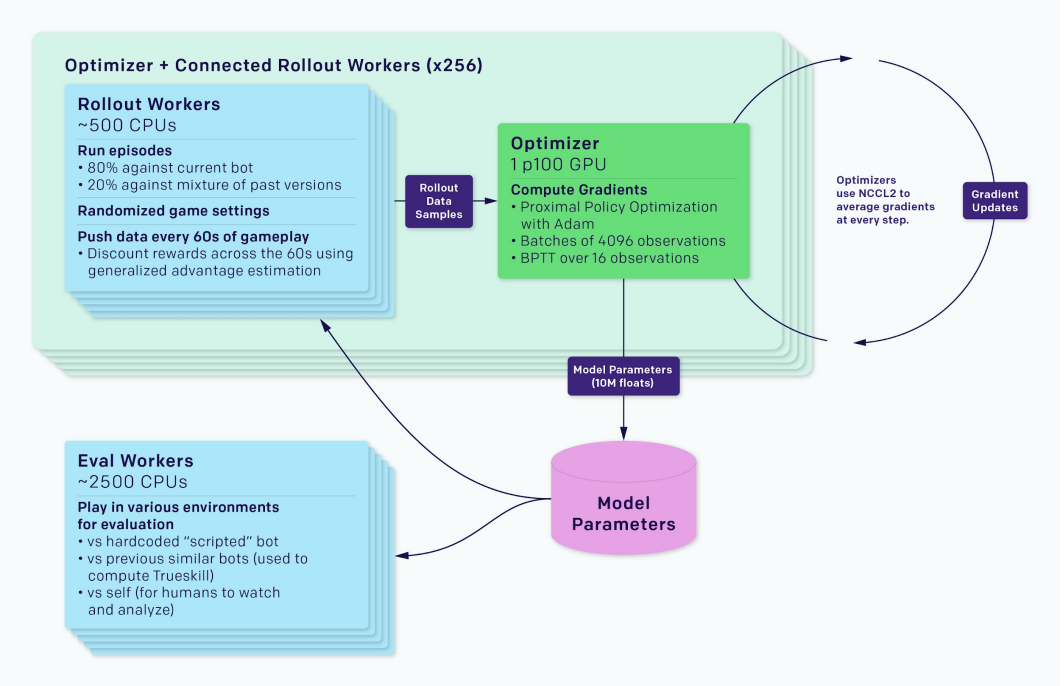
Sim2real: OpenAI's Rubik's cube solver

- Run simulated hand+cube (Mujoco) in diverse range of environments
- Vary: mass, cube size, friction, cube appearance
- Automatic domain randomization:
  - Increase variance in domains once performance plateaus

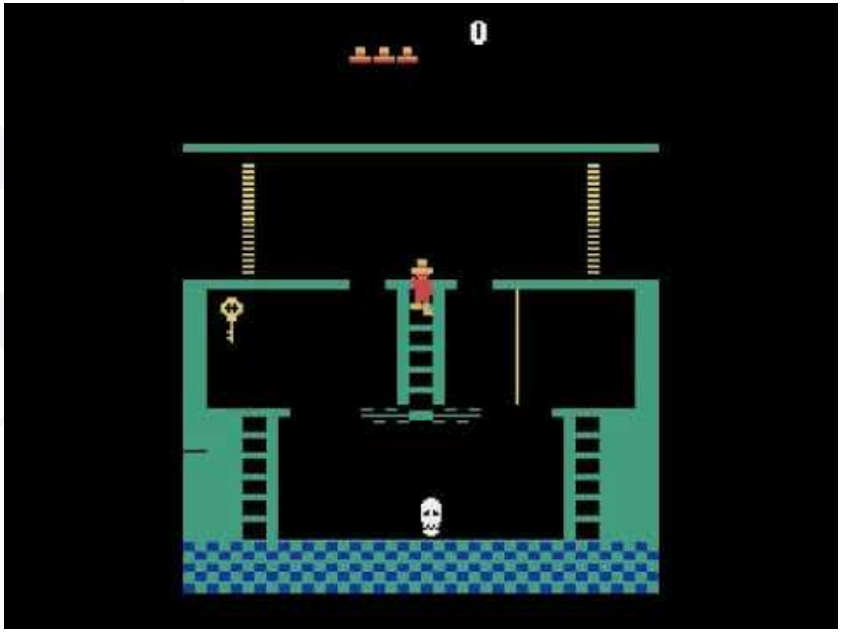


Sim2real: OpenAI's Rubik's cube solver

- Huge distributed training infrastructure (Rapid)
- Train entirely in simulation
- Success rate in real:
  - 60% with 15-steps-to-completed initial configurations
  - 20% with 26-steps initial



Sparse rewards



How do we approach playing a game where rewards are sparse and/or goals are unclear?

Sparse rewards

- Encourage exploration in the absence of a strong reward signal
- Curiosity-based/intrinsic motivation approaches

Curiosity-driven Exploration by Self-supervised Prediction

Deepak Pathak [1] **Pulkit Agrawal** [1] **Alexei A. Efros** [1] **Trevor Darrell** [1]

Abstract

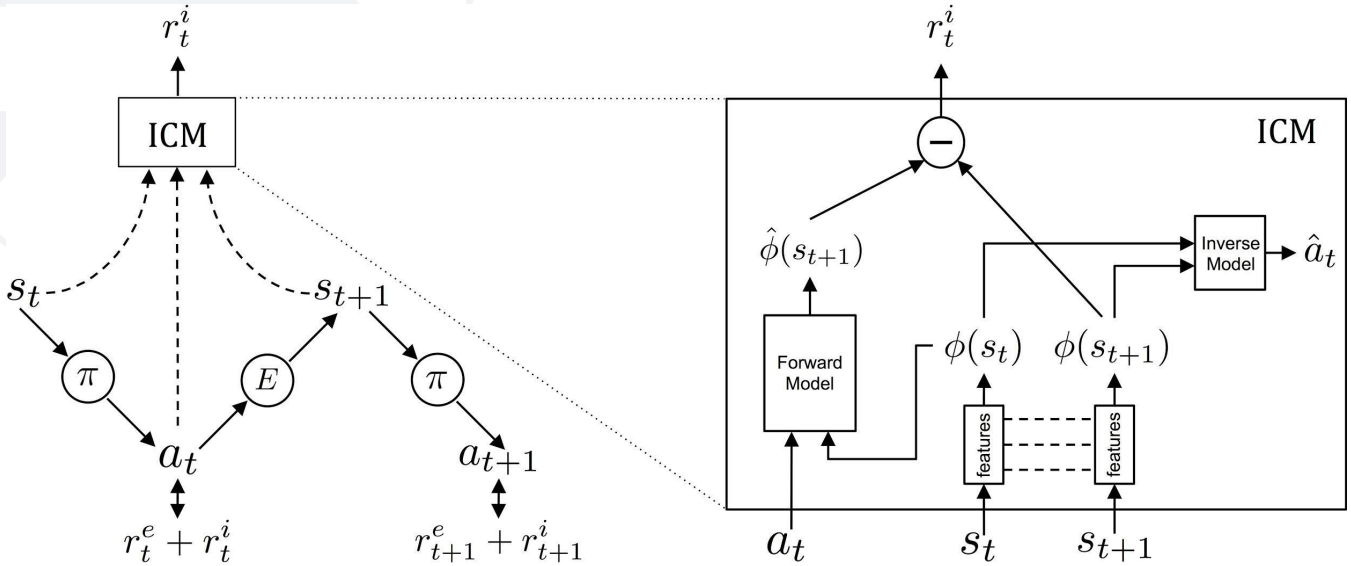
In many real-world scenarios, rewards extrinsic to the agent are extremely sparse, or absent altogether. In such cases, curiosity can serve as an intrinsic reward signal to enable the agent to explore its environment and learn skills that might be useful later in its life. We formulate curiosity as the error in an agent's ability to predict the consequence of its own actions in a visual feature space learned by a self-supervised inverse dynamics model. Our formulation scales to high-dimensional continuous state spaces like images, bypasses the difficulties of directly predicting pixels, and, critically, ignores the aspects of the environment that cannot affect the agent.

(a) learn to explore in Level-1 (b) explore faster in Level-2

Figure 1. Discovering how to play *Super Mario Bros* **without rewards**. (a) Using only curiosity-driven exploration, the agent makes significant progress in Level-1. (b) The gained knowledge helps the agent explore subsequent levels much faster than when starting from scratch. Watch the video at `http://pathak22.github.io/noreward-rl/`

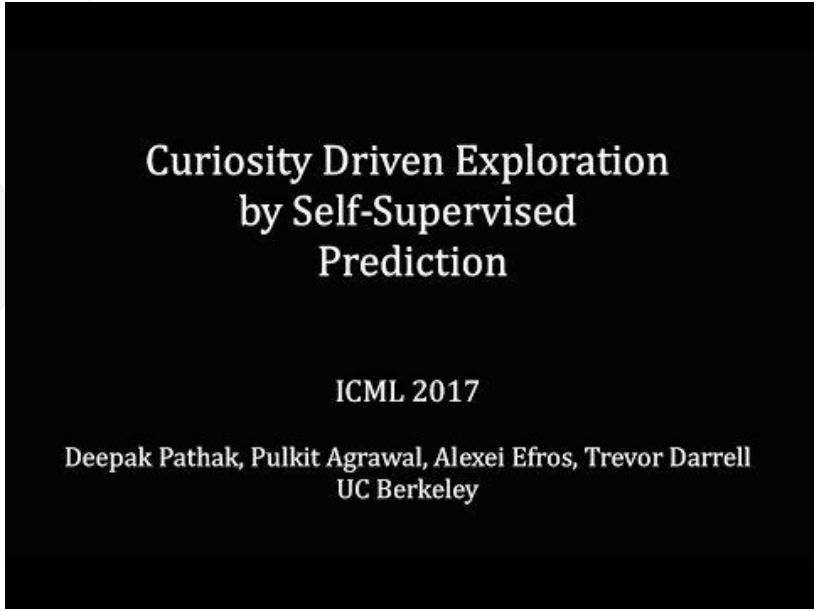
Sparse rewards

- Augment external reward with reward based on surprise (forward prediction error)
- Train with standard RL algorithm (e.g. A3C)



Sparse rewards

- Can play Mario with only curiosity



Sparse rewards

- Exploration bonuses can help with good performance on Montezuma's Revenge

EXPLORATION BY RANDOM NETWORK DISTILLATION

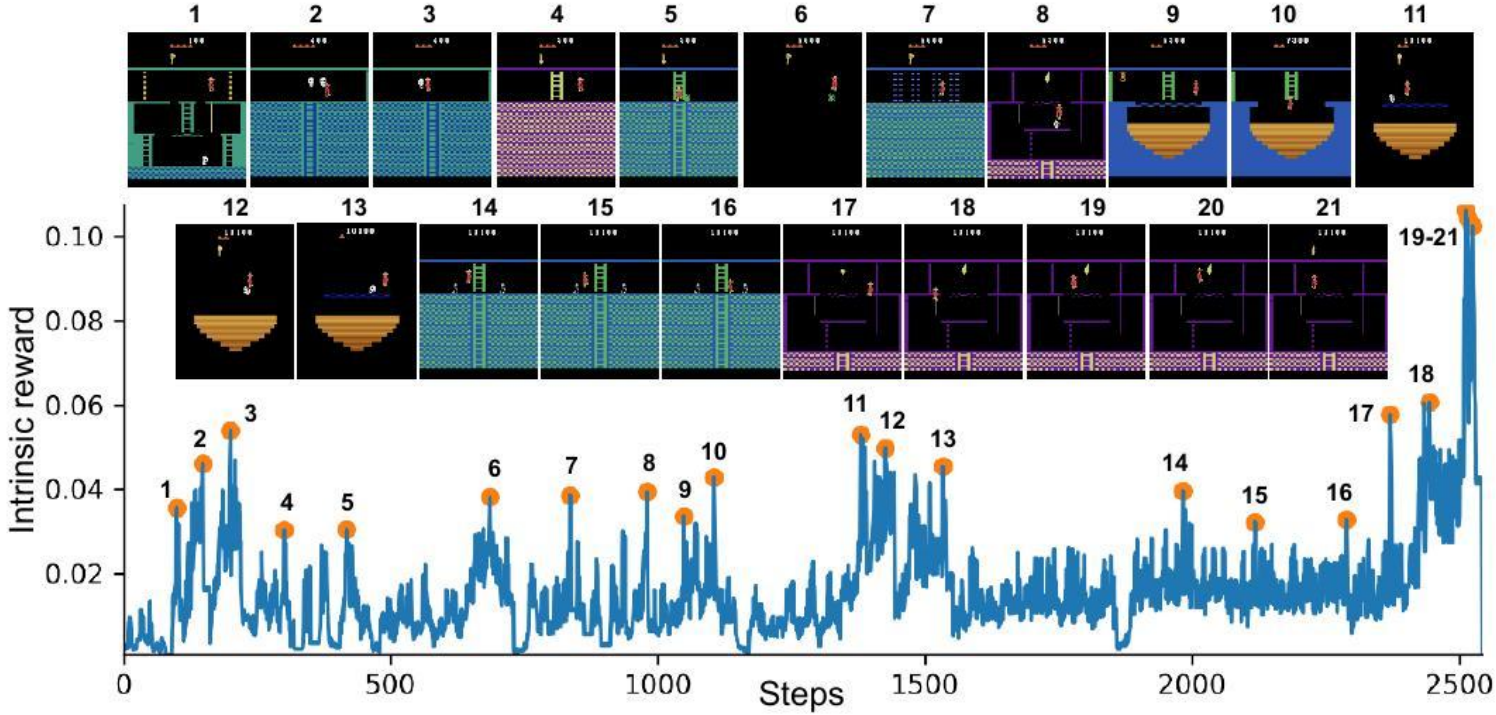
Yuri Burda* OpenAI
Harrison Edwards* OpenAI
Amos Storkey Univ. of Edinburgh
Oleg Klimov OpenAI

ABSTRACT

We introduce an exploration bonus for deep reinforcement learning methods that is easy to implement and adds minimal overhead to the computation performed. The bonus is the error of a neural network predicting features of the observations given by a fixed randomly initialized neural network. We also introduce a method to flexibly combine intrinsic and extrinsic rewards. We find that the random network distillation (RND) bonus combined with this increased flexibility enables significant progress on several hard exploration Atari games. In particular we establish state of the art performance on Montezuma's Revenge, a game famously difficult for deep reinforcement learning methods. To the best of our knowledge, this is the first method that achieves better than average human performance on this game without using demonstrations or having access to the underlying state of the game, and occasionally completes the first level.

Sparse rewards

- Exploration bonuses can help with good performance on Montezuma's Revenge



Generalization

- Good performance in deep RL often requires overfitting
- Comes at cost of efficient transfer to other environments
- But we can learn new (e.g.) games relatively quickly (e.g. minutes of play)

What differences are there in how we learn a new game, compared with a reinforcement learning algorithm (e.g. DQN)?

Generalization

How do we learn Atari?

Investigating Human Priors for Playing Video Games

Rachit Dubey [1] Pulkit Agrawal [1] Deepak Pathak [1] Thomas L. Griffiths [1] Alexei A. Efros [1]

Abstract

What makes humans so good at solving seemingly complex video games? Unlike computers, humans bring in a great deal of prior knowledge about the world, enabling efficient decision making. This paper investigates the role of human priors for solving video games. Given a sample game, we conduct a series of ablation studies to quantify the importance of various priors on human performance. We do this by modifying the video game environment to systematically mask different types of visual information that could be used by humans as priors. We find that removal of some prior knowledge causes a drastic degradation in the speed with which human

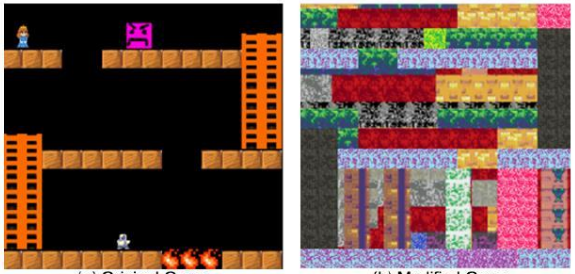


Figure 1. **Motivating example.** (a) A simple platformer game. (b) The same game modified by re-rendering the textures. Despite the two games being structurally the same, humans took twice as long to finish the second game. In comparison, the performance of an RL agent was approximately the same for the two games.

Generalization

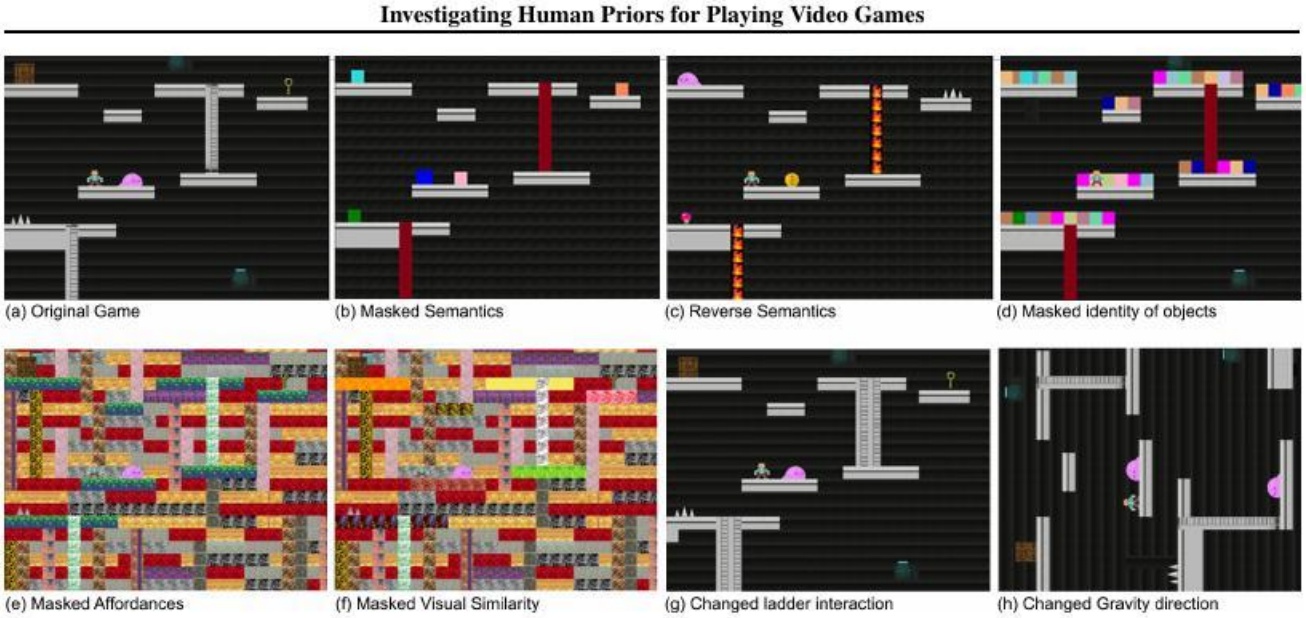


Figure 2. **Various game manipulations**. (a) Original version of the game. (b) Game with masked objects to ablate semantics prior. (c) Game with reversed associations as an alternate way to ablate semantics prior. (d) Game with masked objects and distractor objects to ablate the concept of object. (e) Game with background textures to ablate affordance prior. (f) Game with background textures and different colors for all platforms to ablate similarity prior. (g) Game with modified ladder to hinder participant's prior about ladder interactions. (h) Rotated game to change participant's prior about gravity. Readers are encouraged to play all these games online[2].

Generalization

Significant differences in performance for game modifications

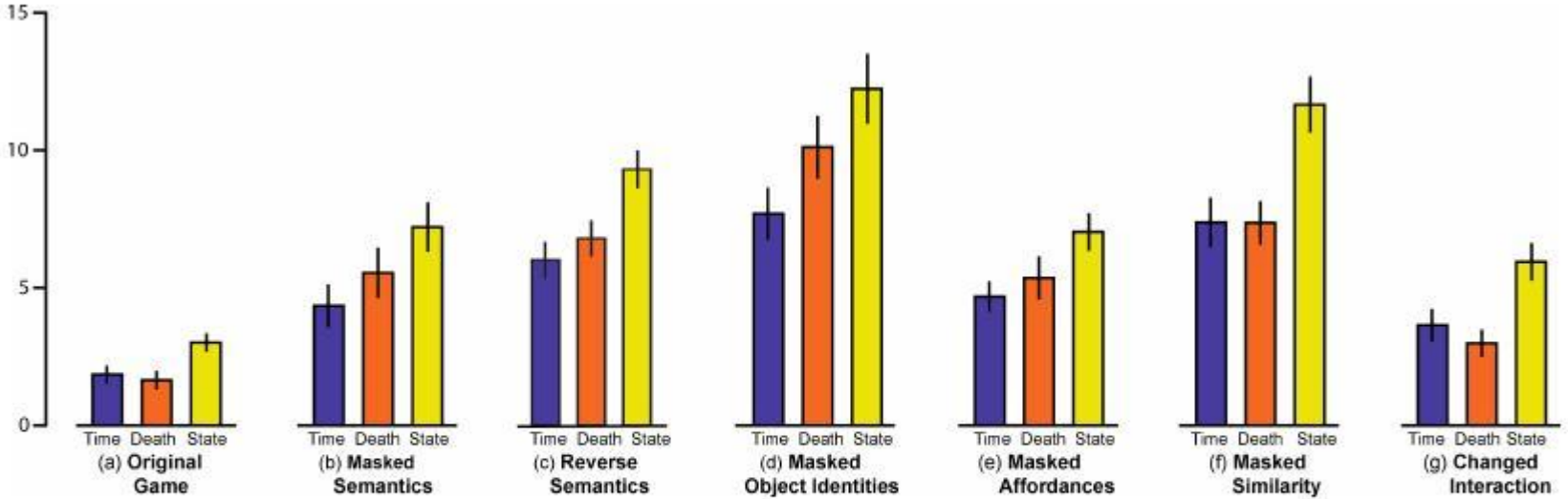


Figure 3. **Quantifying the influence of various object priors.** The blue bar shows average time taken by humans (in minutes), orange bar shows the average number of deaths, and yellow bar shows the number of unique states visited by players to solve the various games. For visualization purposes, the number of deaths is divided by 2, and the number of states is divided by 1000 respectively.

Generalization

RL agent (A3C + curiosity bonus) learns regardless of modifications to environment

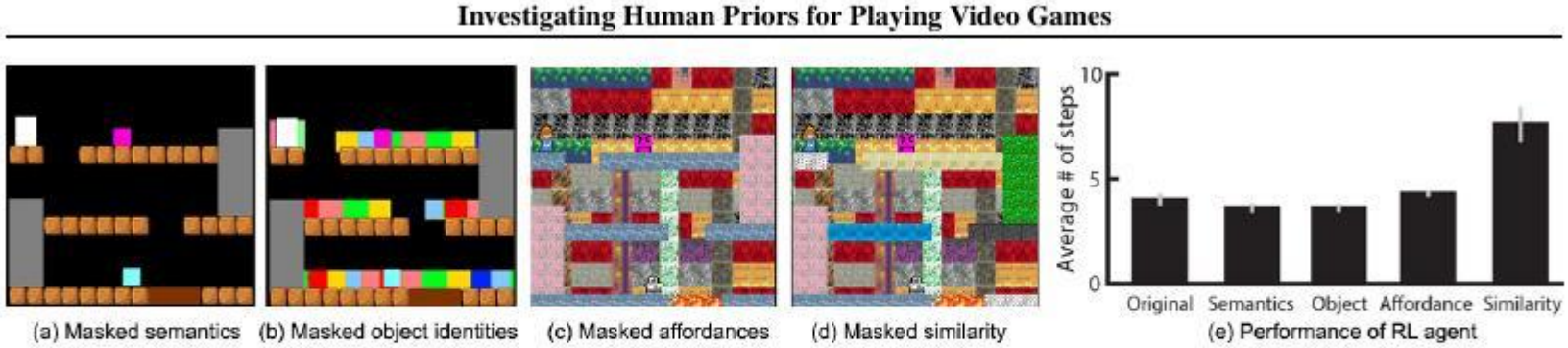
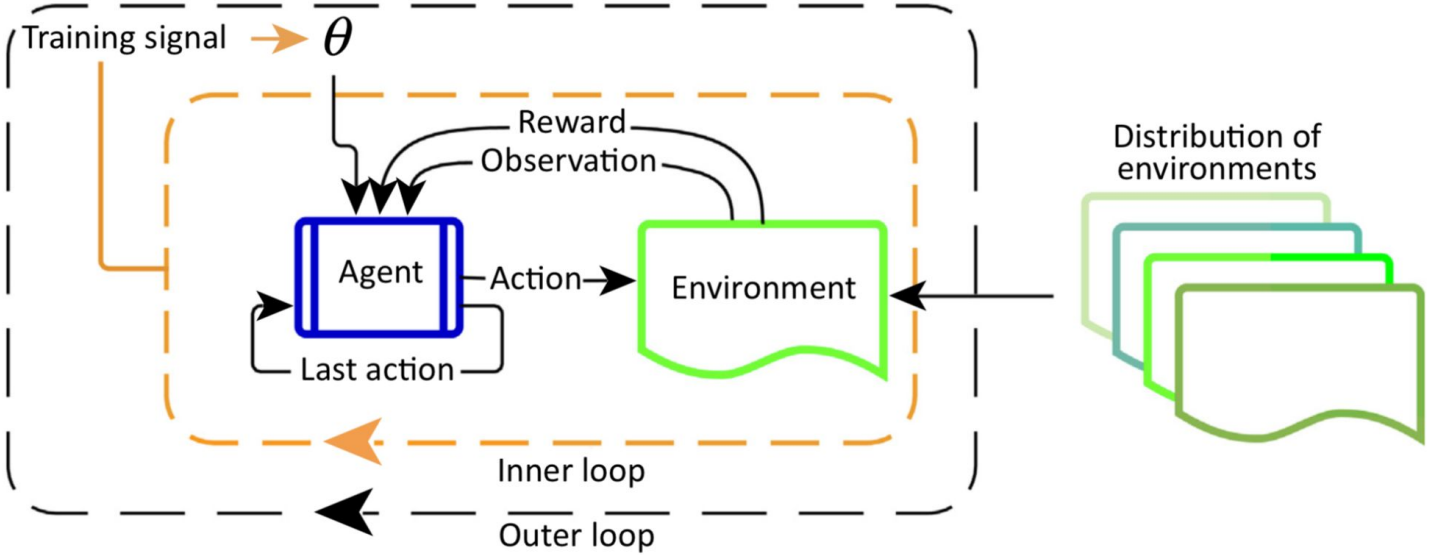


Figure 8. **Quantifying the performance of RL agent.** (a) Game without semantic information. (b) Game with masked and distractor objects to ablate concept of objects. (c) Game without affordance information. (d) Game without similarity information. (e) Performance of RL agent on various game manipulations (steps shown in order of million). Error bars indicate standard error of mean for the 5 random seeds. The RL agent performs similarly on all games except for the one without visual similarity.

Generalization

Learning to learn (meta-learning): learn a prior that allows fast learning in a new environment



Botvinick 2019

Generalization

- Model-agnostic meta learning (MAML)
- Developed at Berkeley AI research
- Objective function:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta).$$

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

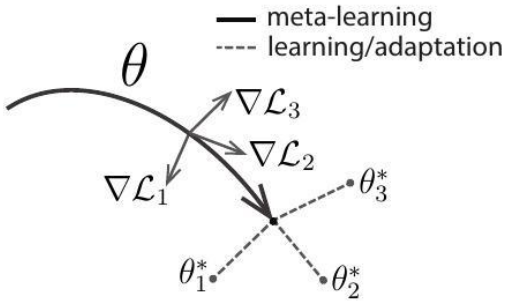


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta_i'}$ in $\mathcal{T}_i$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

Finn et al 2017

Generalization

- Can adapt to new tasks with only a few gradient updates

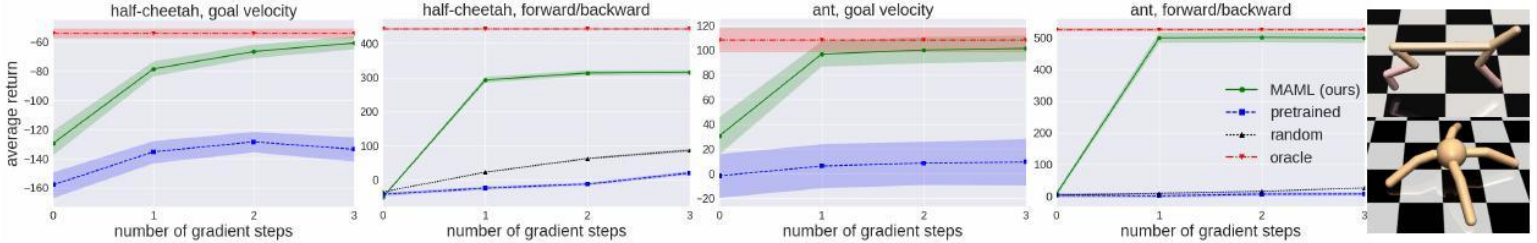


Figure 5. Reinforcement learning results for the half-cheetah and ant locomotion tasks, with the tasks shown on the far right. Each gradient step requires additional samples from the environment, unlike the supervised learning tasks. The results show that MAML can adapt to new goal velocities and directions substantially faster than conventional pretraining or random initialization, achieving good performs in just two or three gradient steps. We exclude the goal velocity, random baseline curves, since the returns are much worse (< -200 for cheetah and < -25 for ant).

Other issues: reproducibility

Tools to increase reproducibility:

- Standardized software tools
  - OpenAI gym (in next week's recitation)
  - Reference implementations of common algorithms
- Online teaching resources:
  - 'Spinning up' course in deep RL (https://spinningup.openai.com/en/latest/)
- Major CS conferences now encourage code with submissions

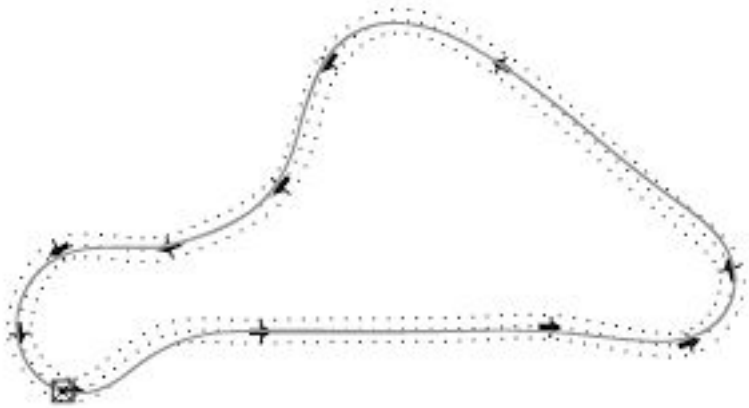
Other topics in learning agents research

Beyond trying to maximize a reward signal, how else can we learn a new task?

Imitation learning

Imitation learning

- No reward, instead observe actions of an expert we want to copy
- Use when rewards are hard to define
- Common approaches
  - Behavior cloning
  - Inverse reinforcement learning
- E.g. race car trajectories:



Imitation learning

Types of Imitation Learning

Behavioral Cloning

$$\text{argmin}_\theta \, E_{(s,a^*)\sim P^*} L(a^*, \pi_\theta(s))$$

Works well when P^* close to P_θ

Inverse RL

Learn r such that:

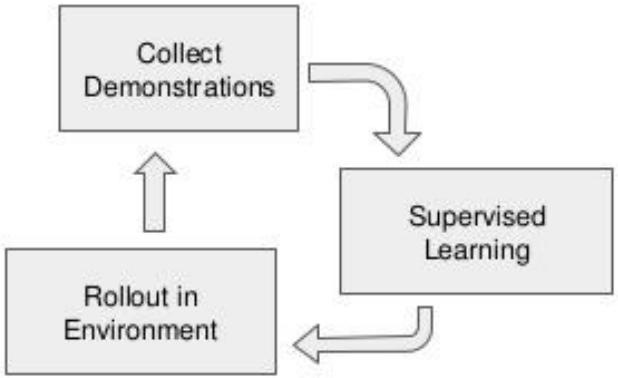
$$\pi^* = \text{argmax}_\theta \, E_{s\sim P(s|\theta)} r(s, \pi_\theta(s))$$

RL problem

Assumes learning r is statistically easier than directly learning π^*

Direct Policy Learning

via Interactive Demonstrator

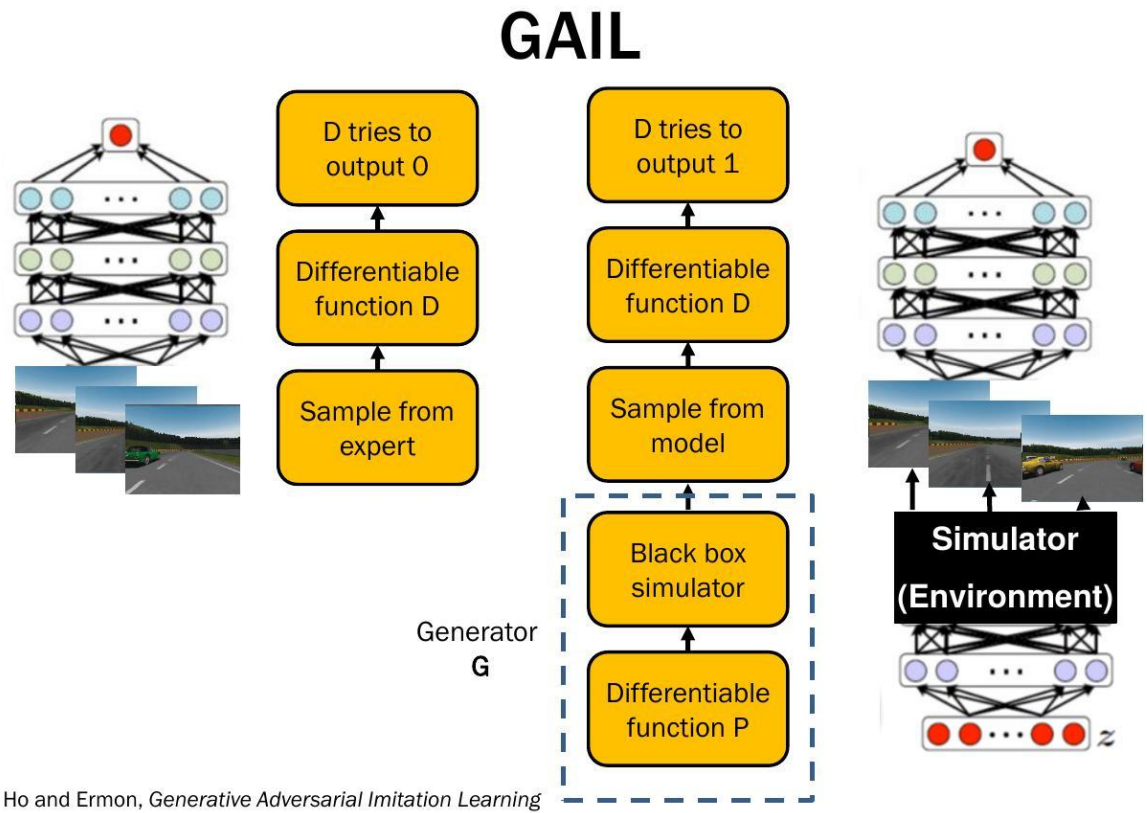


Requires Interactive Demonstrator (BC is 1-step special case)

Yue and Le 2018 ICML Imitation learning tutorial

Imitation learning

Generative adversarial imitation learning



Ho and Ermon, *Generative Adversarial Imitation Learning*

Imitation learning

Adversarial imitation learning:

- Permits learning from state sequences only (learning purely from observing s_t, without having labeled actions) (e.g. Torabi et al 2019)
  - Learn from video data only
- Can be combined with reward signal (e.g. Zolna et al 2019):
  - Copy expert when don't know what else to do
  - Come up with own solution when think can do better

Summary

- Markov decision processes formalize the reinforcement learning problem
- Policies can be learnt with model-free, model-based or policy-gradient methods
- Most recent progress has come from:
  - Combining these approaches
  - Approximating states/value functions/policies with neural networks (DQN, A3C, etc)
- Can 'solve'
  - Many* Atari games
  - Go, chess
  - Rubik's cube manipulation
- Just focusing on maximizing reward may be limiting
  - Future progress likely to come from combining RL and imitation learning

*All? Read up on DeepMind's Agent57

Take-home messages

Thoughts on RL (and other learning agents):

- A very fast-moving field
- Computational resources required to reach good performance are prohibitive for many researchers and practitioners
- Further advances in efficiency, generalization and sim2real transfer required before useful in many real-world settings
- Safe AI: what are appropriate reward functions?
- Exciting to see where the field moves after its emphasis on games

How could this lecture be better?