Deep Reinforcement Learning

A brief introduction

Skander Moalla 24.07.2024, Deep Learning Bootcamp - LauzHack

whomai

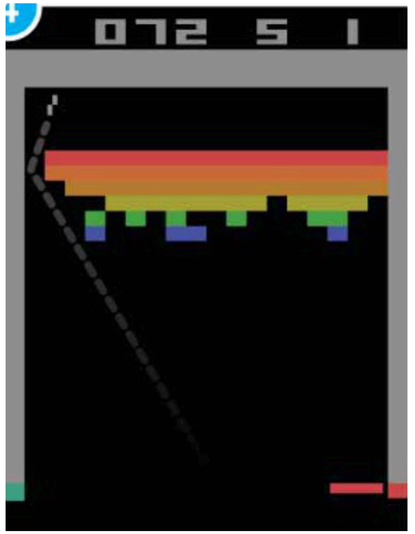
Skander Moalla

- PhD at CLAIRE (Prof. Caglar Gulcehre)
- Deep Reinforcement Learning
  - Policy gradient methods and RL finetuning (RLHF, etc.)
- Previously
  - MSc thesis multi-agent RL @WhiRL Oxford (Prof. Shimon Whiteson)
  - Applied research scientist (DeepRL for middle-mile logistics

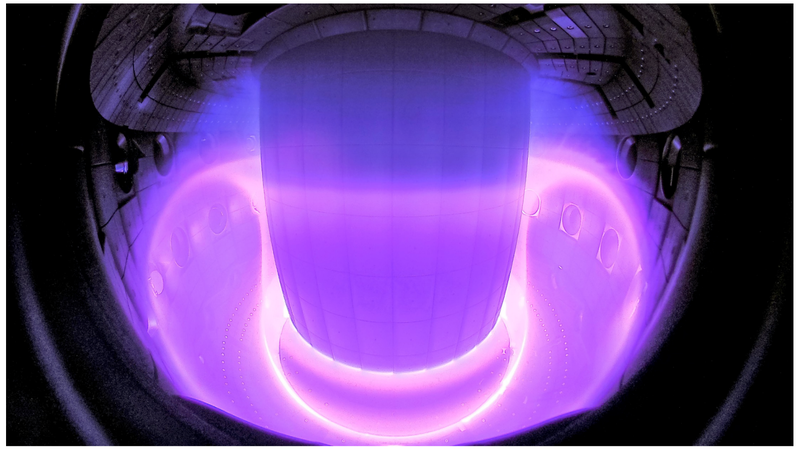
Successes in deep RL

nature

Human-level control through deep reinforcement learning



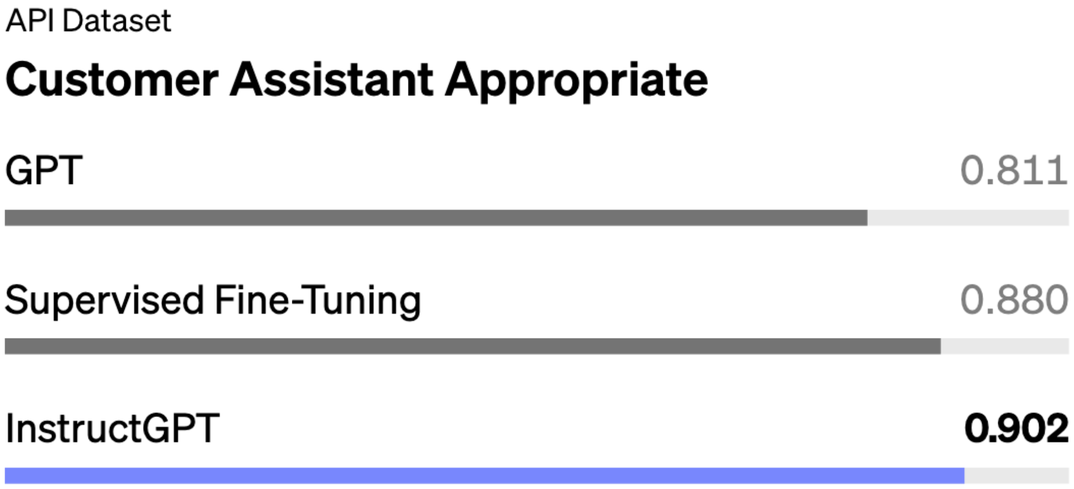
Outracing champion Gran Turismo drivers with deep reinforcement learning



Magnetic control of tokamak plasmas through deep reinforcement learning

OpenAI

Aligning language models to follow instructions

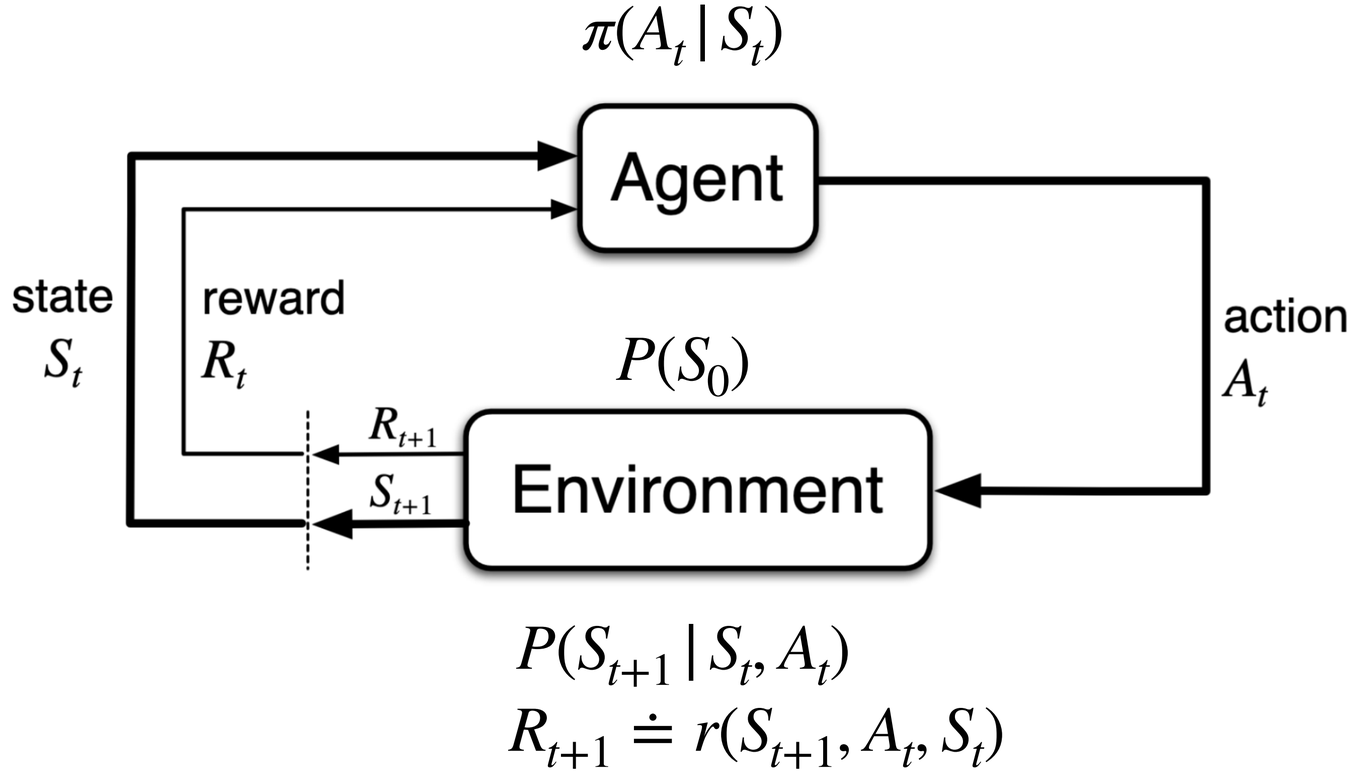


What is reinforcement learning?

Problem setting: sequential decision making

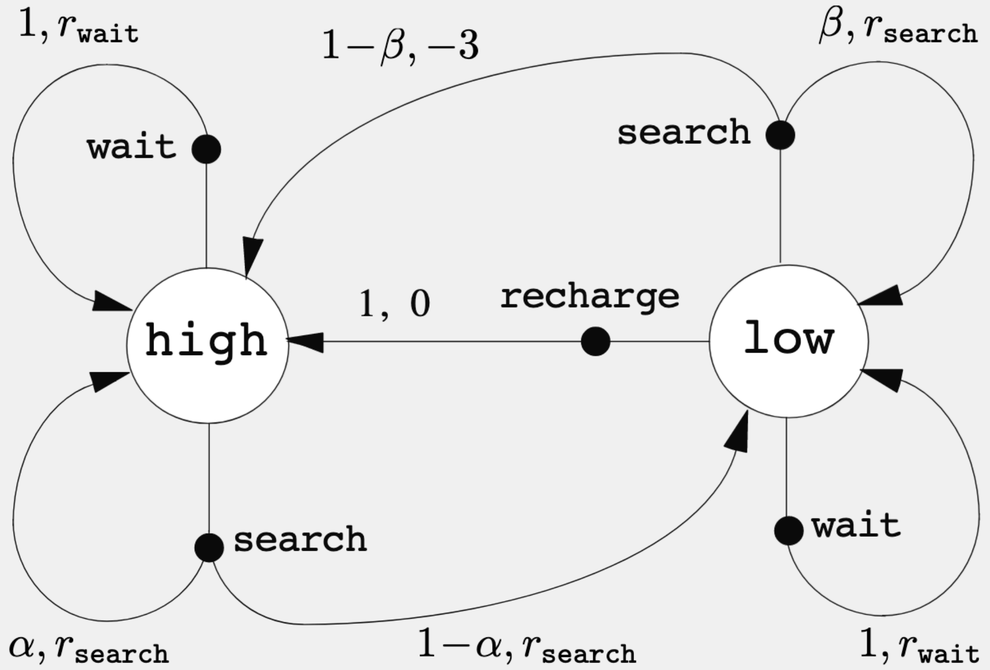
Markov Decision Process (MDP), aka an environment and an agent

- A set of states, with an initial state distribution.
- A set of actions, that the agent can take.
- The agent acts according to a (stochastic) policy: given a state picks an action
- A (stochastic) transition function determines the next state
- A reward function



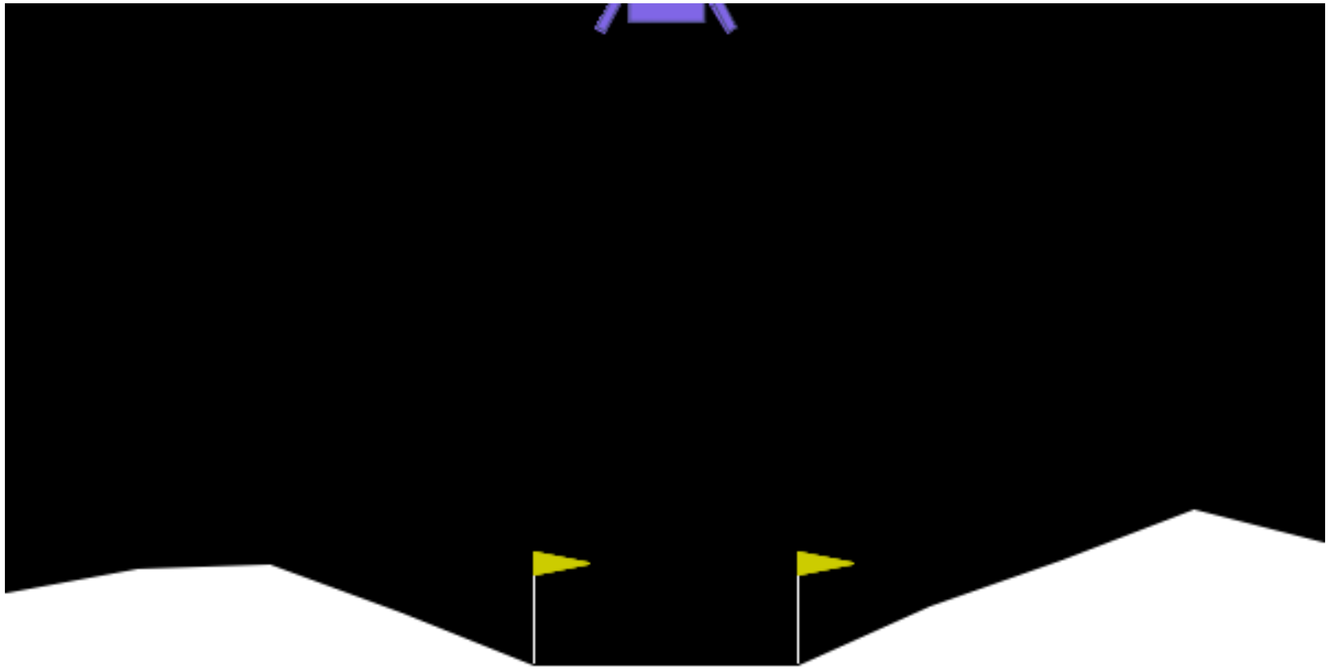
Example

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



(Sutton and Barto 2018)

Example



(https://gymnasium.farama.org/)

Example

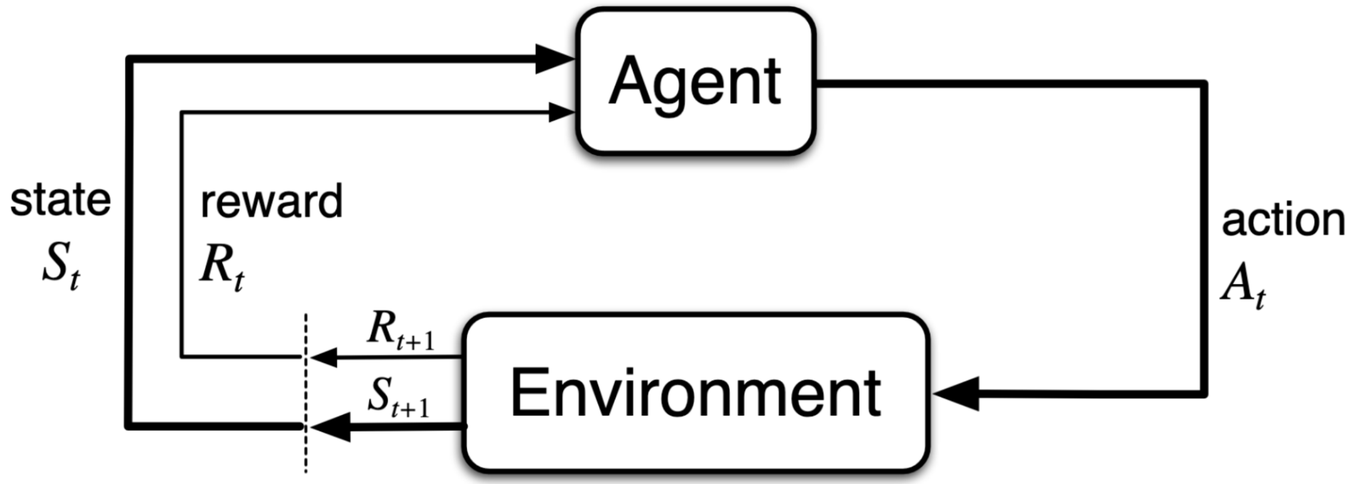
```python
import gymnasium as gym
env = gym.make("LunarLander-v2", render_mode="human")
observation, info = env.reset(seed=42)
for _ in range(1000):
   action = env.action_space.sample()  # this is where you would insert your policy
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

(https://gymnasium.farama.org/)

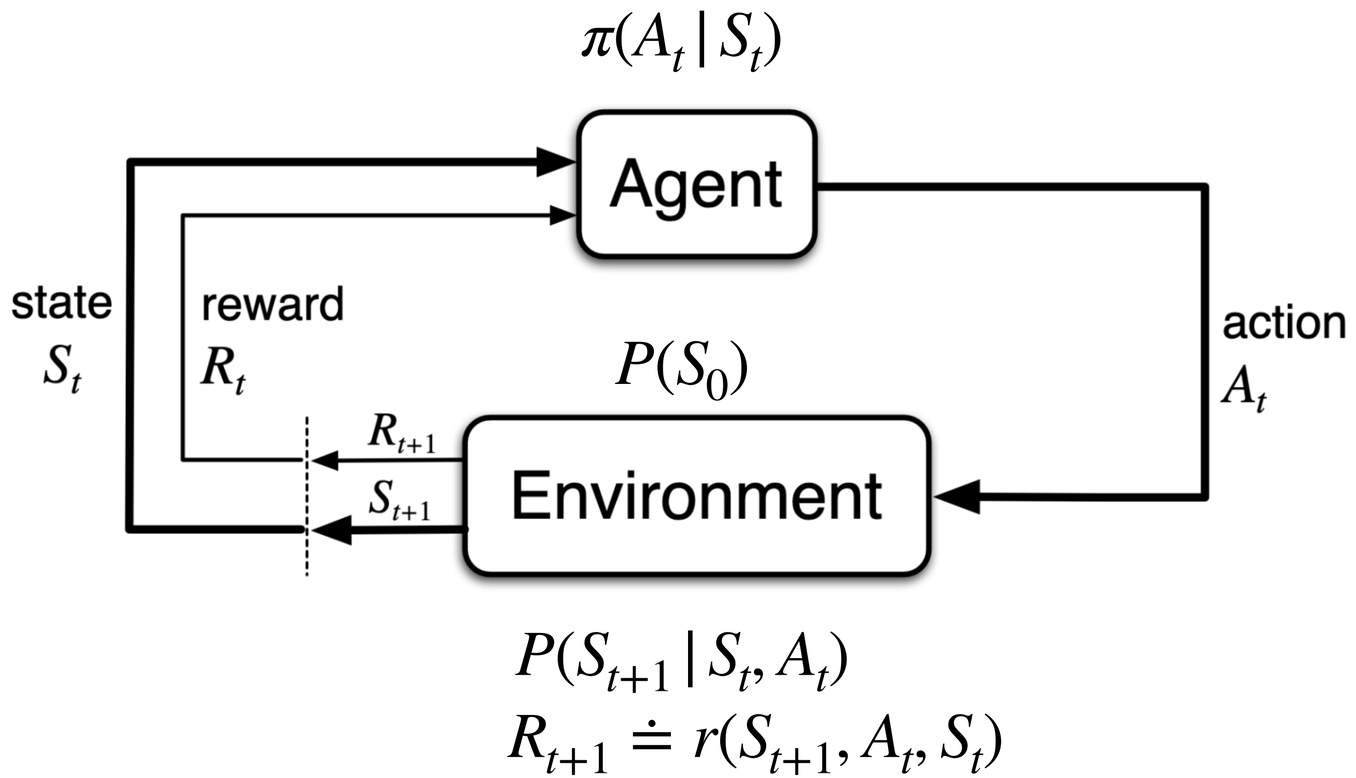
Problem setting

Markov Decision Process (MDP), aka an environment and an agent

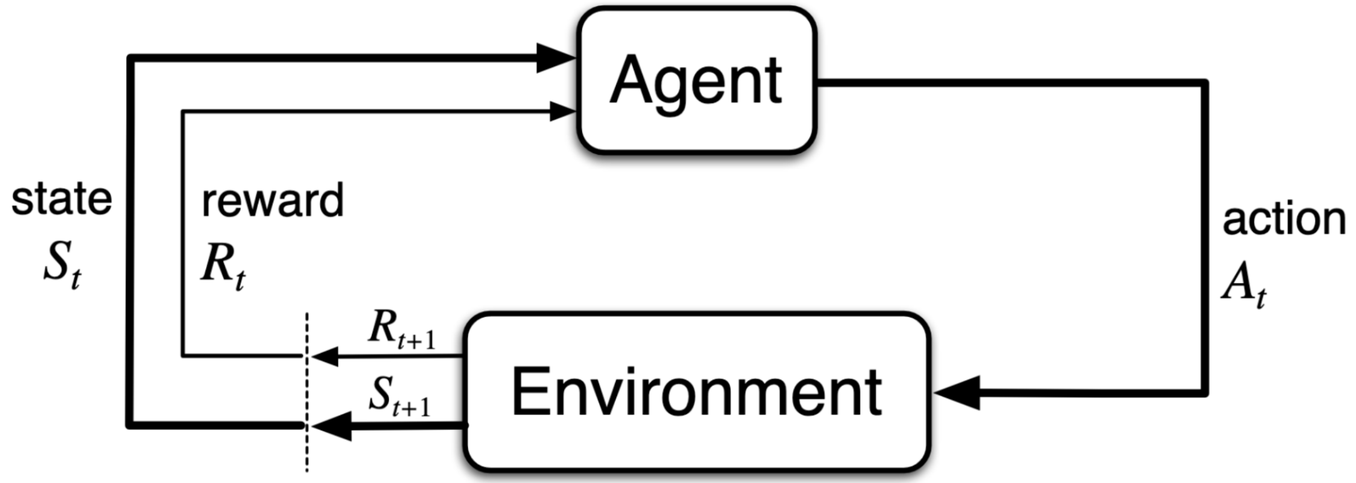
- The interaction defines random trajectories following an underlying distribution
- $S_0, A_0, S_1, R_1, A_1, S_2, R_2, \ldots, S_T$
- S_T is a terminal state. T is random and the trajectory can be infinite.



What is “Markov” in the MDP?



What is “Markov” in the MDP?



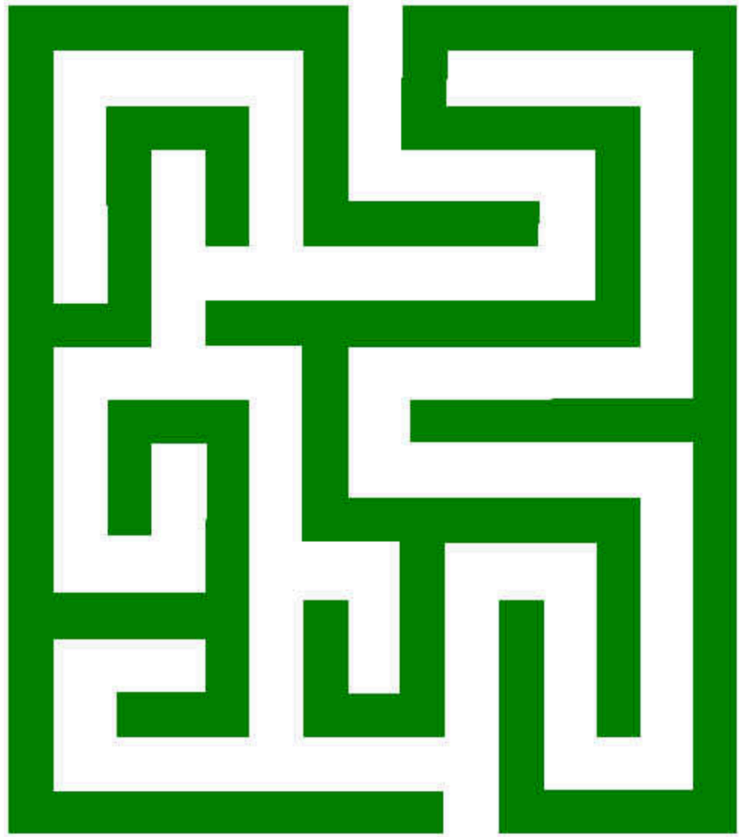
$$P(S_{t+1} \mid S_t, A_t) = P(S_{t+1} \mid S_t, A_t, S_{t-1}, \ldots, S_0)$$

“The future is independent of the past given the present”

Is it Markov?

A quick game

- A robot in a maze
- State: vector of 4 elements indicating wall/no wall on each side
- Actions: move up, down, left, right, unless a wall is in the way



(Shimon Whiteson's lecture notes)

Is it Markov?

A quick game

- A game of chess
- State: the board position
- Actions: all legal moves
- Opponent, part of the environment, has a fixed reactive policy

(Shimon Whiteson’s lecture notes)

The objective

Maximize the expected return

- Define the return of a trajectory start at time step t

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots$$

- Find the policy that **maximizes** the **expected** return

$$max_\pi \mathbb{E}[G_0] = max_\pi \mathbb{E}_\pi[R_0 + \gamma R_1 + \gamma^2 R_2 + \dots]$$

- The Markov property ensures that we can search over only reactive policies $\pi(a \mid s)$
- There is always at least one deterministic optimal policy.

Quick Quizz

RL vs supervised learning

- What’s the training dataset?
- What’s the model?
- What’s the loss function?

How do we find the optimal policy?

Value functions

We first need to figure out how well the current policy is doing

- State-value function

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[\sum_{k=0} \gamma^k R_{t+k+1} \,|\, S_t = s \right]$$

- Action-value function

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{k=0} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a \right]$$

Estimating value functions

Monte Carlo estimation

- Simply sample lots of trajectories and estimate the function
- In deep RL you would have a function $\hat{v}(\,\cdot\,;\mathbf{w})$ and would minimize

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=0}^{N}(\hat{v}(s_i;\mathbf{w}) - y_i)^2$$

- Where $y_i = r_{t+1} + \gamma r_{t+2} + \ldots$ of a trajectory starting at s_i
- The same would apply for a $\hat{q}(\,\cdot\,,\cdot\,;\mathbf{w})$

Value functions

Bellman equations

- Value functions can be defined recursively

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s \right]$$

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a \right]$$

Estimating value functions

Bootrapping

- Can now reuse $\hat{v}(\,\cdot\,;\mathbf{w})$ itself to learn faster

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=0}^{N}(\hat{v}(s_i;\mathbf{w}) - y_i)^2$$

- Where $y_i = r_{t+1} + \gamma\hat{v}(s_{t+1};\mathbf{w})$ of a trajectory starting at s_i

Estimating value functions

Bootrapping

- Can now reuse $\hat{q}(\cdot, \cdot; \mathbf{w})$ itself to learn faster

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=0}^{N} (\hat{q}(s_i, a_i; \mathbf{w}) - (r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}; \mathbf{w}))^2$$

- Any issues?

Estimating value functions

Bootrapping

- Can now reuse $\hat{q}(\cdot, \cdot; \mathbf{w})$ itself to learn faster

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=0}^{N} (\hat{q}(s_i, a_i; \mathbf{w}) - (r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}; \mathbf{w}))^2$$

- Any issues?
  - Must take a semi-gradient. The value function in the target should be taken out of the computation graph of autograd!

Estimating value functions

Bias vs variance tradeoff

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=0}^{N} (\hat{q}(s_i, a_i; \mathbf{w}) - y_i)^2$$

- Which one is biased? Which one has more variance?
  - 1. $y_i = R_{t+1} + \gamma R_{t+2} + \dots$
  - 2. $y_i = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}; \mathbf{w})$

Value improvement

Local improvement to get global improvement

- The policy is updated with the best action estimated so far (greedy with respect to its q values)

$$\pi(\cdot \mid s) = \arg\max_{a} \hat{q}(s, a; \mathbf{w})$$

- Need to maintain some exploration otherwise will be stuck
  - ϵ-greedy scheme: pick the best action with probability $1 - \epsilon$, uniformly randomize the remaining ϵ

Q-learning

Directly estimating the optimal Q-value

- The Bellman optimality equation
  - The optimal q-value of the optimal policy is greedy with respect to itself

$$q^*(s,a) \doteq \mathbb{E}_\pi \left[R_{t+1} + \max_{a'} \gamma q^*(S_{t+1}, a') \,|\, S_t = s, A_t = a \right]$$

Deep Q-Network (DQN)

Directly estimating the optimal Q-value

- Optimize

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=0}^{N}(\hat{q}(s_i, a_i; \mathbf{w}) - (r_{t+1} + \max_{a'}\gamma\hat{q}(s_{t+1}, a'; \mathbf{w}^-))^2$$

- Using a ϵ-greedy policy to collect data
- The data is stored in a replay buffer and a minibatch is sampled after each data collection
- A target network (a lagging copy of the network) is used for the targets
- On Atari (Mnih et al. 2015) the network is a CNN.

Deep Q-Network (DQN)

The notion of off-policy

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=0}^{N}(\hat{q}(s_i, a_i; \mathbf{w}) - (r_{t+1} + \max_{a'}\gamma\hat{q}(s_{t+1}, a'; \mathbf{w}^-))^2$$

- The q-values being learned are not the ones of the policy collecting the data
- This is called off-policy learning

Quick Quizz

- What is Monte Carlo?
- What is bootstrapping?
- What’s the bias-variance tradeoff?
- What’s the difference between on-policy and off-policy?

Policy gradient methods

The policy as model

- We can directly parameterize the policy with a neural network $\pi_\theta(a \mid s) \doteq \pi(a \mid s; \theta)$
- The objective $J(\theta) = \mathbb{E}_{\pi_\theta}[G_0]$ is differentiable and its derivative is given by the policy gradient theorem (Sutton et al. 1999)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_t q(S_t, A_t)\, \nabla_\theta \log \pi(A_t \mid S_t; \theta)\right]$$

Policy gradient methods
REINFORCE

- With autograd with can simply estimate the expectation with sampled trajectories and do backprop
- The q-value can also be estimated with the same samples
- This is the “REINFORCE” algorithm

$$\sum_t G_t \nabla_\theta \log \pi(A_t \mid S_t; \theta)$$

Policy gradient methods
REINFORCE

- The trajectories collected to estimate the policy gradient can have high variance and limited samples can cause a step to be in the wrong direction.
- Taking a bad step in RL is much more problematic than in supervised learning. Why?

Policy gradient methods
REINFORCE

- The trajectories collected to estimate the policy gradient can have high variance and limited samples can cause a step to be in the wrong direction.
- Taking a bad step in RL is much more problematic than in supervised learning. Because it will affect the policy and the next batch of data it collects
- We need to way to use the collected data as efficiently as possible

Proximal Policy Optimization (PPO)

And trust-region methods

- The notion of advantage $a_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$
- PPO maximizes the advantage of the new policy vs. the policy collecting the data

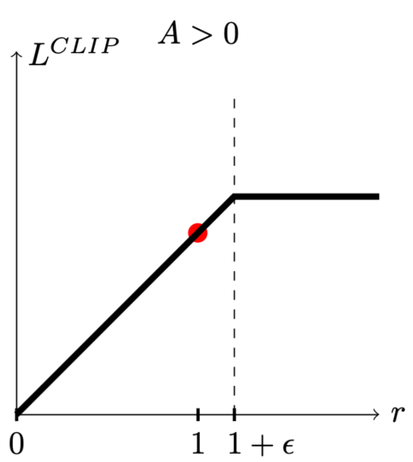
$$L^{PPO}(\theta) = \mathbb{E}_{\pi_{old}}\left[\sum_t a_{\pi_{old}}(S_t, A_t)\frac{\pi(A_t \mid S_t;\theta)}{\pi_{old}(A_t \mid S_t)}\right]$$

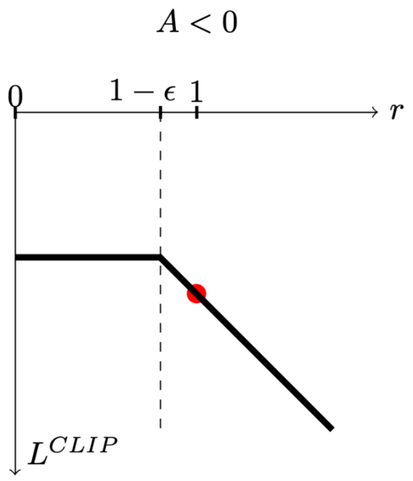
Proximal Policy Optimization (PPO)

And trust-region methods

- PPO maximizes the advantage of the new policy vs. the policy collecting the data while maintaining a trust region.

$$L^{CLIP}(\theta) = \mathbb{E}_{\pi_{old}}\left[\sum_t min(a_{\pi_{old}}(S_t, A_t)\frac{\pi(A_t \mid S_t; \theta)}{\pi_{old}(A_t \mid S_t)}, a_{\pi_{old}}(S_t, A_t)clip(\frac{\pi(A_t \mid S_t; \theta)}{\pi_{old}(A_t \mid S_t)}, 1+\epsilon, 1-\epsilon)\right]$$





(Schulman et al. 2017)

Proximal Policy Optimization (PPO)

And trust-region methods

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

(Schulman et al. 2017)

A bit about my work

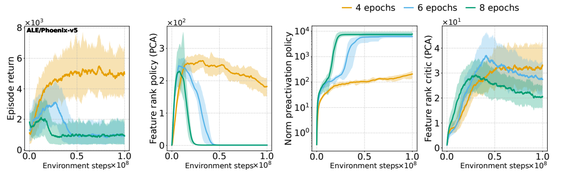
PPO suffers from a **deteriorating representation** that **breaks** its **trust region.**

No Representation, No Trust: Connecting Representation, Collapse, and Trust Issues in PPO

Skander Moalla, Andrea Miele, Razvan Pascanu, Caglar Gulcehre

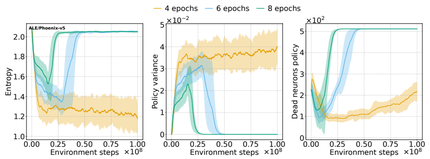
EPFL Google DeepMind

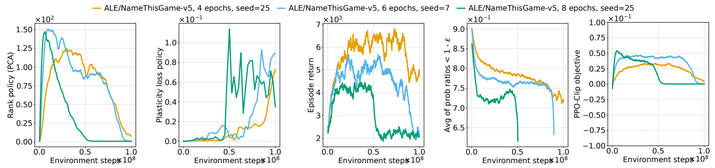
Fully reproducible and replicable



PPO agents on Atari and MuJoCo suffer from **representation collapse causing performance collapse**. The collapse is **faster with stronger non-stationarity,** achieved with more epochs.

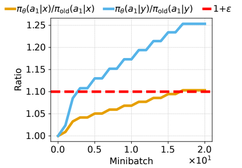
The collapsed policy has **high entropy, but zero variance across states**. All neurons are dead and the **model doesn't distinguish between states**, leading to trivial performance.

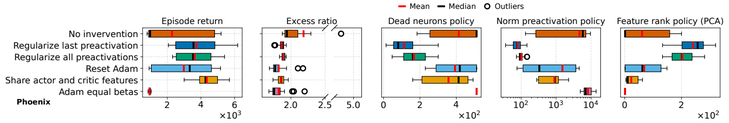




The **trust region cannot prevent this catastrophic change** as **it also breaks down with a poor representation.** The **policy's plasticity also becomes so poor** that the **agent cannot recover** by optimizing the surrogate objective.

We show that it's **not possible to maintain the trust region with a collapsed representation.**





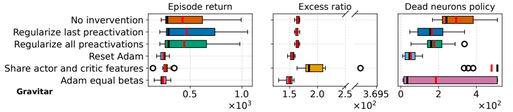
Regularizing representations and non-stationarity results in a better trust region and **mitigates performance collapse.**

Proximal Feature Optimization (PFO)

A simple auxiliary loss to motivate controlling the representation

$$L^{PFO}_{\pi_{old}}(\theta) = \mathbb{E}_{\pi_{old}}\left[\sum_{t=0}^{t_{max}-1}\left(\phi_\theta(S_t) - \phi_{old}(S_t)\right)^2\right]$$

Bonus: when should you share actor-critic features?
Probably not when rewards are sparse as it drives the critic to rank collapse, and the actor with it.



Extra topics in (deep) RL

- Offline RL
- Imitation learning
- Inverse RL
- Partially observable environments (POMDP)
- Multi-agent