

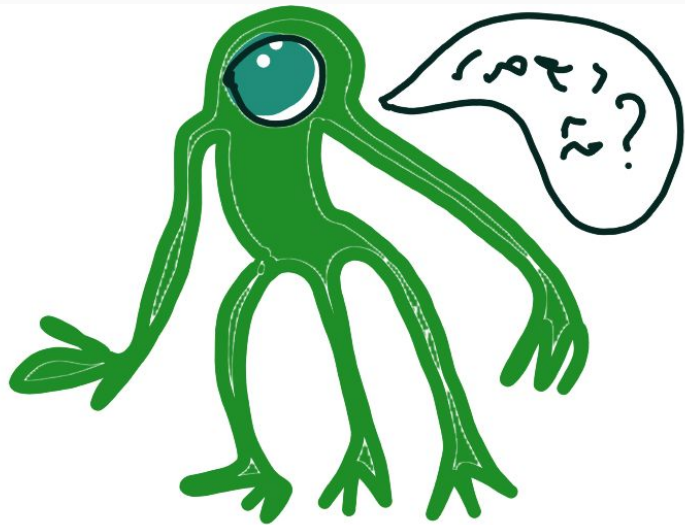
Reinforcement Learning

A crash course on the Deterministic Policy Gradient



Emanuele Panizon @ AreaSciencePark 31/5/2024

A “strange” tutorial: Deep Deterministic Policy Gradient



The shortest course.

We study reinforcement learning and control problems in which an agent acts in a stochastic environment by sequentially choosing actions over a sequence of time steps, in order to maximise a cumulative reward. We model the problem as a *Markov decision process* (MDP) which comprises: a *state space* \mathcal{S} , an *action space* \mathcal{A} , an *initial state distribution* with density $p_1(s_1)$, a stationary *transition dynamics distribution* with conditional density $p(s_{t+1}|s_t, a_t)$ satisfying the Markov property $p(s_{t+1}|s_1, a_1, \dots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$, for any trajectory $s_1, a_1, s_2, a_2, \dots, s_T, a_T$ in state-action space, and a *reward function* $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

A *policy* is used to select actions in the MDP. In general the policy is stochastic and denoted by $\pi_\theta : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on \mathcal{A} and $\theta \in \mathbb{R}^n$ is a vector of n parameters, and $\pi_\theta(a_t|s_t)$ is the conditional probability density at a_t associated with the policy. The agent uses its policy to interact with the MDP to give a trajectory of states, actions and rewards, $h_{1:T} = s_1, a_1, r_1, \dots, s_T, a_T, r_T$ over $\mathcal{S} \times \mathcal{A} \times \mathbb{R}$. The return r_t^γ is the total discounted reward from time-step t onwards, $r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$ where $0 < \gamma < 1$. Value functions are defined to be the expected total discounted reward, $V^\pi(s) = \mathbb{E}[r_t^\gamma | S_1 = s; \pi]$ and $Q^\pi(s, a) = \mathbb{E}[r_t^\gamma | S_1 = s, A_1 = a; \pi]$.¹ The agent's goal is to obtain a policy which maximises the cumulative discounted reward from the start state, denoted by the performance objective $J(\pi) = \mathbb{E}[r_1^\gamma | \pi]$.

We denote the density at state s' after transitioning for t time steps from state s by $p(s \rightarrow s', t, \pi)$. We also denote the (improper) discounted state distribution by $\rho^\pi(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \rightarrow s', t, \pi) ds$. We can then write the performance objective as an expectation,

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s, a) r(s, a) da ds = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [r(s, a)] \quad (1)$$

where $\mathbb{E}_{s \sim \rho} [\cdot]$ denotes the (improper) expected value with respect to discounted state distribution $\rho(s)$.² In the remainder of the paper we suppose for simplicity that $\mathcal{A} = \mathbb{R}^m$ and that \mathcal{S} is a compact subset of \mathbb{R}^d .

What is RL?

What is RL trying to learn?

How do I define an environment?

How to define the optimization problem?

Policy gradient algorithms are perhaps the most popular class of continuous action reinforcement learning algorithms. The basic idea behind these algorithms is to adjust the parameters θ of the policy in the direction of the performance gradient $\nabla_\theta J(\pi_\theta)$. The fundamental result underlying these algorithms is the *policy gradient theorem* (Sutton et al., 1999),

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)] \quad (2) \end{aligned}$$

The policy gradient is surprisingly simple. In particular, despite the fact that the state distribution $\rho^\pi(s)$ depends on the policy parameters, the policy gradient does not depend on the gradient of the state distribution.

A way to solve it!

Deterministic Policy Gradient Algorithms

David Silver
DeepMind Technologies, London, UK
Guy Lever
University College London, UK
Nicolas Heess, Thomas Degris, Dan Wierstra, Martin Riedmiller
DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM
GUY.LEVER@UCL.AC.UK
*@DEEPMIND.COM

Reinforcement Learning: the 3rd wheel.

Supervised

*Data with label.
“Extend” information.*

- image recognition
- force fields
- LLMs
- ...

Unsupervised

*Data without label.
“Extract” information.*

- clustering
- ID extraction
- ...

Reinforcement Learning

*Sequences of causal data.
Optimize “policy”.*

- games
- resource management
- “reverse engineering”
to understand behavior
- ...

Reinforcement Learning: the 3rd wheel.

Q: What is the right choice of “**action**”?

≠ **supervised:**

there is no “list of good actions” to look-up.

≠ **unsupervised:**

the task/objective/reward is given.



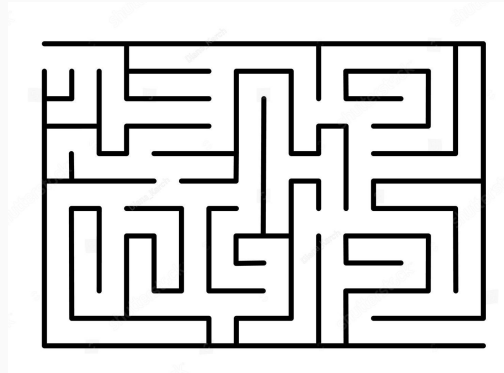
The environment: part I.

The fundamental building block is the concept of Markov Decision Process:
an agent interacting with a (rewarding) environment.

“**State**” $s \in \mathbf{S}$: is the well defined state of the environment. *countable/uncountable*

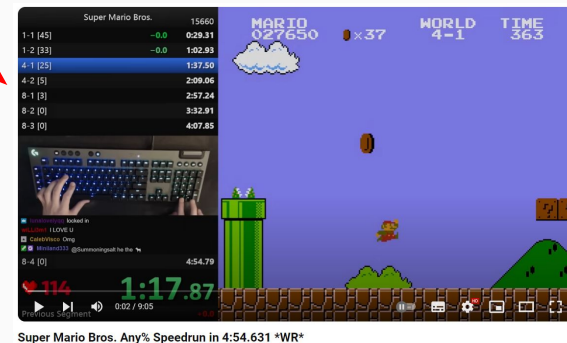
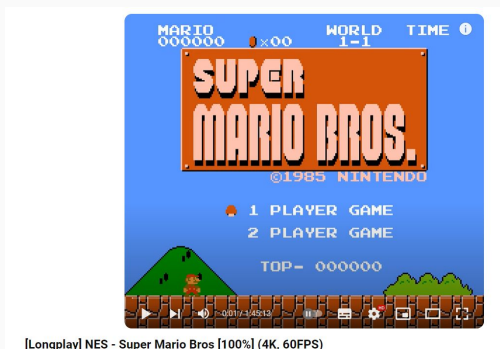
“**Action**” $a \in \mathbf{A}$: is what the agent can do to change the state. *discrete/continuous*

“**Transition**” $t(s'|s,a)$: encodes the dynamics of the environment. *stochastic/deterministic*



The environment: part II.

“**Reward**” $r(s,a,s')$: is the immediate gain the agent has being in a state/doing an action: reward at time t is r_t



“**Discount**” γ : is the discount for future reward ($1/\gamma$ is the time horizon)

“**Return**” : is the accumulated discounted reward from one time-step onwards.

$$R = \sum_k \gamma^k r_{t+k}$$

The agent.

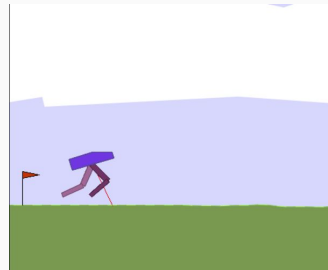
“**Policy**” $\pi(\mathbf{a}|\mathbf{s})$: is what tells the agent what action to take next, given the “**state**”.
($\pi(\mathbf{a}|\mathbf{s})$ = probability to take action \mathbf{a} , from state \mathbf{s})

*policy can be
stochastic &
discrete*



$\pi(\mathbf{a}|\mathbf{s})$ means $p(\leftarrow | \text{img alt="Small screenshot of Super Mario Bros. showing Mario on a brick platform." data-bbox="735 608 782 682})$

*policy can be
deterministic
& continuous*

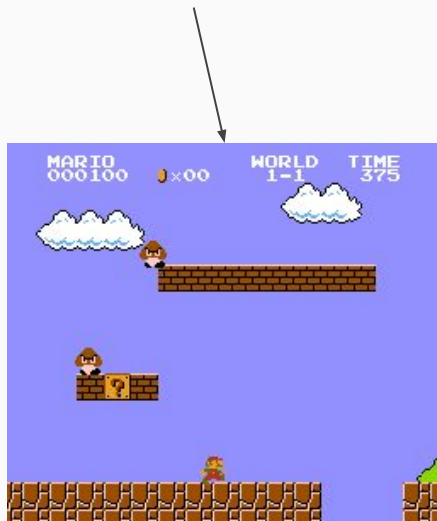


$$\mu = \pi(\mathbf{s})$$

Value Functions

“Value” $V^\pi(\mathbf{s})$: is some estimated of all future rewards, starting from a **state** \mathbf{s} and following the **policy** π .

$$V^\pi(\mathbf{s}) = E^\pi[\sum_k \gamma^k r_{t+k} \mid \mathbf{s} = \mathbf{s}^t]$$



$$V^\pi(s) = E^\pi[r_t + \gamma V^\pi(s^{t+1}) \mid s = s^t]$$

“Value is what I expect to get from this state following this policy:
the immediate reward + all I will get afterwards”

$$Q^\pi(s,a) = E^\pi[r_t + \gamma V^\pi(s^{t+1}) \mid s = s^t, a = a^t]$$

“Quality is what I expect to get from this state doing this first
action and only then following this policy:

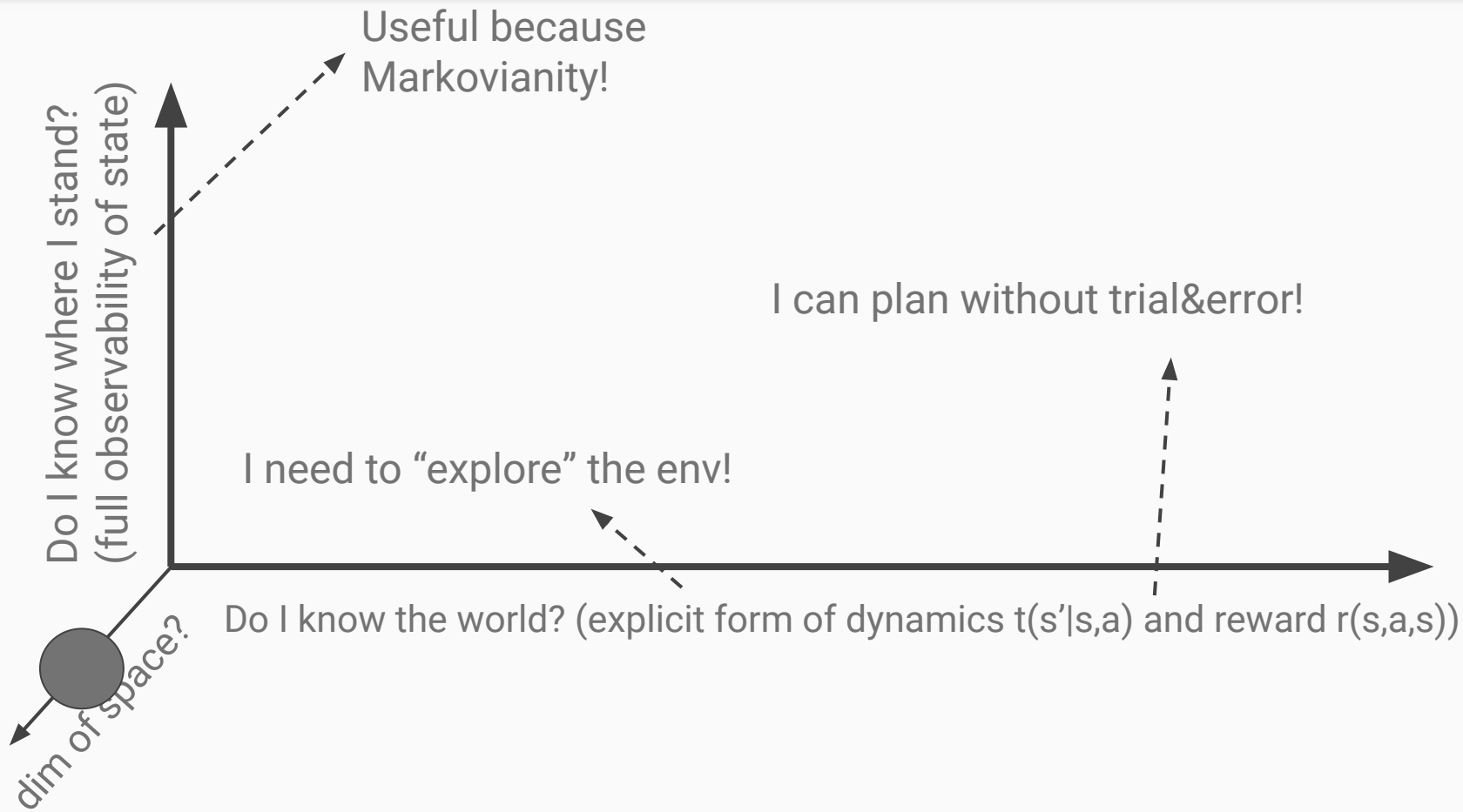
the immediate reward + all I will get afterwards”

“Objective $J(\pi)$ ”: is the “overall goodness” of the policy.



$$\begin{aligned} J(\pi) &= E^\pi[\sum_t \gamma^t r_t \mid s_0 \sim \rho_0] \\ &= E^\pi[V(s_0) \mid s_0 \sim \rho_0] \end{aligned}$$

A conceptual map of RL



Model - free Evaluation

One needs to have an approximated evaluation of the value, generally based on some parameters W :

$$Q^w_{\text{approx}}{}^\pi(s, a)$$

Model- free: gather trajectories from interactions.

collect (lots of) $\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \dots, r_T\}$

each time I visit a state I have some new “experimental” info

$$Q^\pi(s, a) = E^\pi[r_t + \gamma Q^\pi(s^{t+1}, a^{t+1}) \mid s = s^t, a = a^t]$$

$$Q_{\text{exp}}{}^\pi(s_0, a_0) = r_1 + \gamma Q^w_{\text{approx}}{}^\pi(s_1, a_1)$$

Model - free: function approximation

how do I update $Q_{\text{approx}}^w \pi(s,a)$?

1) choose:

$$\sum_t \gamma^t r_{t+1}$$

MonteCarlo

$$Q_{\text{exp}} \pi(s,a) =$$

$$r + \gamma Q_{\text{approx}}^w \pi(s',a')$$

SARSA

$$r + \gamma \max_a Q_{\text{approx}}^w \pi(s',a)$$

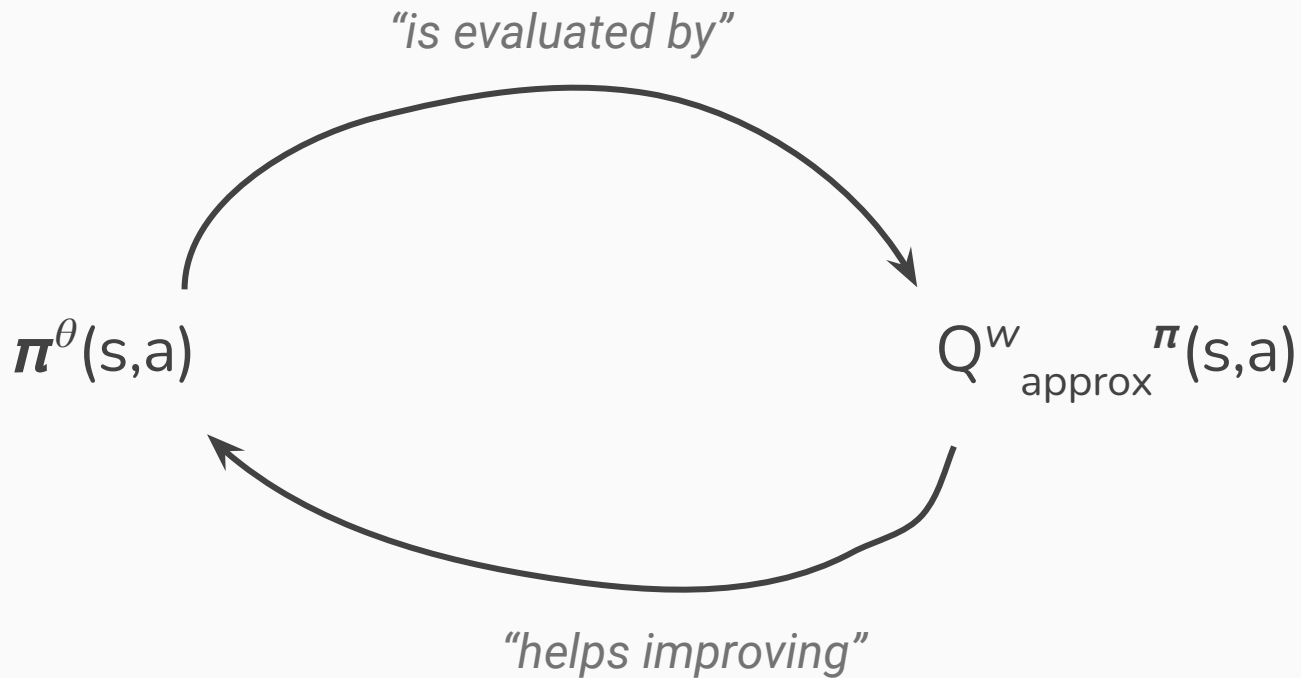
Q-Learning

2) update w :

$$\text{loss} = (Q_{\text{approx}}^w \pi(s,a) - Q_{\text{exp}} \pi(s,a))^2 = \delta Q^2$$
$$\nabla_w \text{loss}$$

How to solve RL in practice?

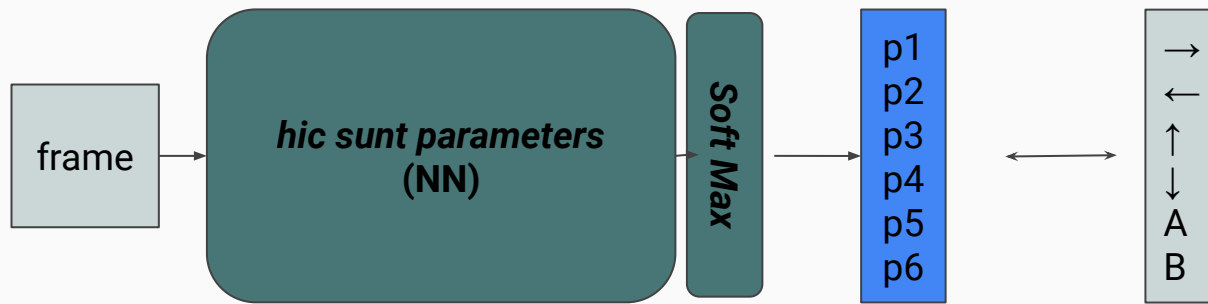
Policy iteration cycle (not exactly, but...)



Introducing the Deep into the Deep Reinforcement Learning

let's fix a parameterized policy:

$$\pi^{\theta}(s,a)$$



it is associated to a score:

$$J(\pi^{\theta}(s,a))$$

Policy Gradient Theorem

$$\nabla_{\theta} J(\pi) \sim E^{\pi} [\nabla_{\theta} \log \pi^{\theta}(\mathbf{a} | \mathbf{s}) \times (Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}))]$$

change of prob to take the
action I *really* took

expected value
of that action

**Neural Network weights are changed not to minimize loss,
but to maximise objective!**

General scheme of actor/critic algorithm

Initialize $\boldsymbol{\pi}^\theta(s,a)$, $Q^w_{\text{approx}}{}^\pi(s,a)$

loop over episodes

- select $\mathbf{a} \sim \boldsymbol{\pi}^\theta(s,a)$
- perform action and get \mathbf{s}', r'
- compute $Q^{\text{exp}}(s,a)$ e.g. $r' + \gamma \max_{a'} Q(s',a')$
- evaluate gradient for *critic* weights w
$$\nabla_w (Q^w_{\text{approx}}{}^\pi(s,a) - Q^{\text{exp}}(s,a))^2$$
- evaluate gradient for *actor* weights θ
$$\nabla_\theta J \sim \nabla_\theta \log \boldsymbol{\pi}^\theta(\mathbf{a} | \mathbf{s}) \times (Q^\pi(s,a) - V^\pi(s))$$
- update weights

What about continuous actions?

Discrete Stochastic
Policy

$$\pi^\theta(a | s)$$



Deterministic Policy

$$a = \mu^\theta(s)$$

$$\nabla_\theta \log \pi^\theta(a_t | s_t)$$



?

$$\nabla_\theta J \sim \nabla_\theta \log \pi^\theta(a | s) \\ \times (Q^\pi(s,a) - V^\pi(s))$$

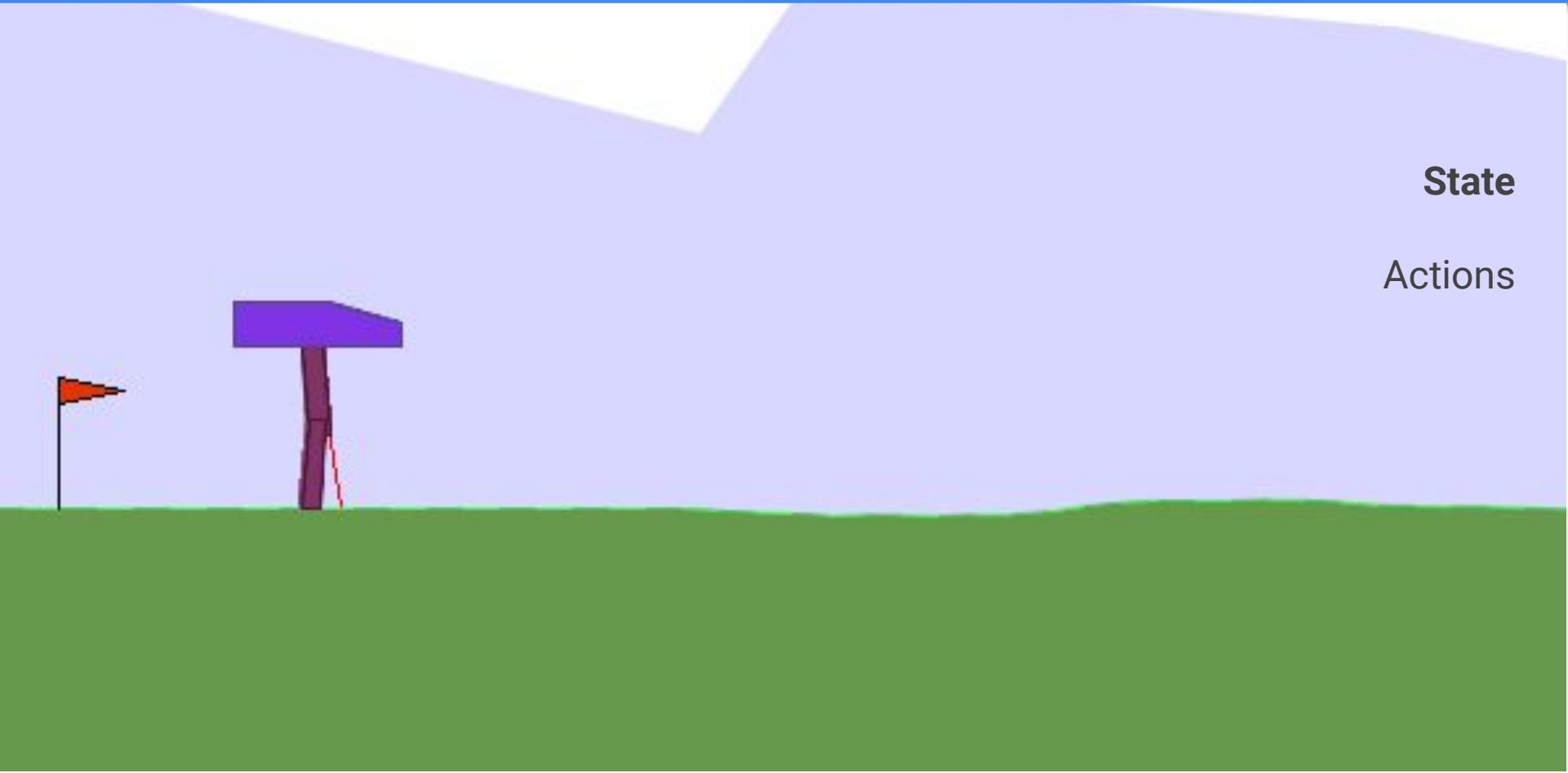


???!?!?

! "semi"-deterministic
policies (PPO)

$$a \sim \mu^\theta(s) + N(0, \sigma) \\ \pi^{\theta, \sigma}(a|s) \\ \text{etc. etc.}$$

Our task today.



State

Actions

A new Policy Gradient.

Policy gradient for deterministic policies.

$$\begin{aligned}\nabla_{\theta} J(\boldsymbol{\mu}) &= \mathbb{E}^{\pi} [\nabla_{\theta} Q^{\pi}(s, \boldsymbol{\mu}^{\theta}(s))] \\ &= \mathbb{E}^{\pi} [\nabla_{\theta} \boldsymbol{\mu}^{\theta}(s) \nabla_a Q^{\pi}(s, a)|_{a=\boldsymbol{\mu}^{\theta}(s)}]\end{aligned}$$

Magically simple.

The gradient for the overall score is the (weighted) gradient for the quality function.

$$\nabla_a Q^{\pi}(s, a)|_{a=\boldsymbol{\mu}^{\theta}(s)} \times \nabla_{\theta} \boldsymbol{\mu}^{\theta}(s)$$

“How much the quality changes around the chosen action”

“How much the action changes wrt the parameters”

What about learning the critic?

stochastic

$$\text{loss} = (r + \gamma \max_a Q^w_{\text{approx}} \pi(s', a) - Q^w_{\text{approx}} \pi(s, a))^2$$



deterministic (in a sense, $\mu^\theta(s')$ is implicitly supposed to maximize Q)

$$\text{loss} = (r + \gamma Q^w_{\text{approx}} \pi(s', \mu^\theta(s')) - Q^w_{\text{approx}} \pi(s, a))^2$$

And now: the technical problems.

1) How to explore with a deterministic policy?

Add noise! Trajectories are constructed with an exploration policy

$$\boldsymbol{\mu}'(s) = \boldsymbol{\mu}^\theta(s) + \mathcal{N}$$


Gaussian

Ornstein-Uhlenbeck

[Why is it not a problem to be off-policy? Thanks to Q-Learning!]

Part II. Technical problems!

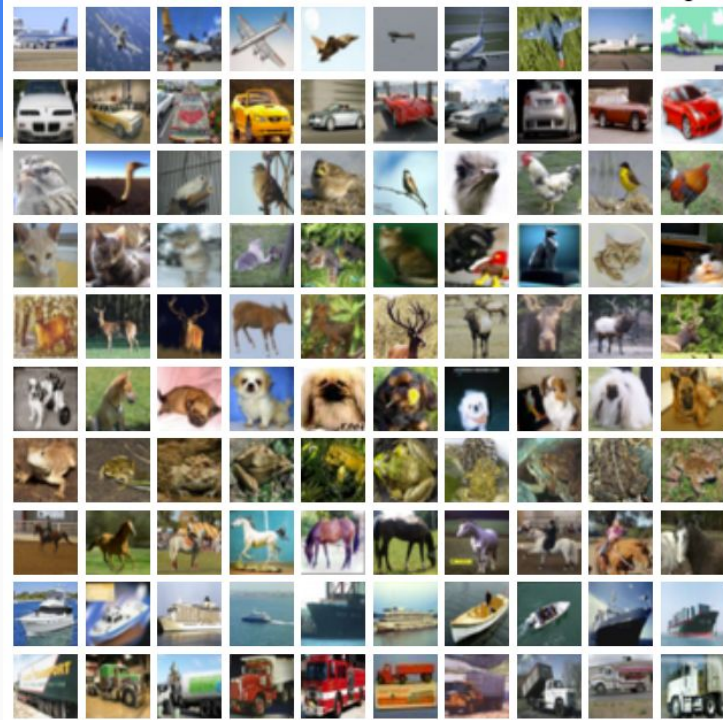
2) *Deeply unstable.*

2a) *Experience is very correlated.*

Solution: memory/replay buffer!

Experience is stored in a “database” and learning is done selecting random times in the (even long) past, not only last experience.

$exp = (s, a, r, s', done)$



Part II. Technical problems!


2b) NN are chasing their own tails.

$$\triangle_w \text{loss} = (r + \gamma Q_{\text{approx}}^w \pi(s', \mu^\theta(s')) - Q_{\text{approx}}^w \pi(s, a))^2$$


Solution: split critic Q into two:

critic Q^w , learns by stochastic gradient descent

and *target_critic* $Q^{w'}$, updates its parameters w' by (slowly) copying w does not enter in the gradient calculation.

$$\triangle_w \text{loss} = (r + \gamma Q_{\text{approx}}^{w'} \pi(s', \mu^\theta(s')) - Q_{\text{approx}}^w \pi(s, a))^2$$


Here at last!

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

for episode = 1, M **do**

Initialize a random process \mathcal{N} for action exploration
Receive initial observation state s_1

for t = 1, T **do**

Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R

Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for

end for

3) Q-Learning update suffers from maximization bias.

$$\text{loss} = (r + \gamma \max_a Q_{\text{approx}}^{w'} \pi(s', a) - Q_{\text{approx}}^w \pi(s, a))^2$$

the max over a noisy variable has always a positive bias!

Solution(s):

1) make two independent copies of the critic (and their targets) $Q_1^{w_1}, Q_2^{w_2}$

$$\text{new target} = r + \min_{1,2} [\gamma Q_i^{w_i} \pi(s', \mu^\theta(s'))]$$

2) evaluate the $Q_i^{w_i}$ not exactly at the “optimal” action, but with some error, so that artificial peaks get smoothed out

Last but not least!

Algorithm 1 TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ

Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$

Initialize replay buffer \mathcal{B}

for $t = 1$ **to** T **do**

Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
 $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
Store transition tuple (s, a, r, s') in \mathcal{B}

Sample mini-batch of N transitions (s, a, r, s') from \mathcal{B}

$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$

$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$

Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

if $t \bmod d$ **then**

Update ϕ by the deterministic policy gradient:

$$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

Update target networks:

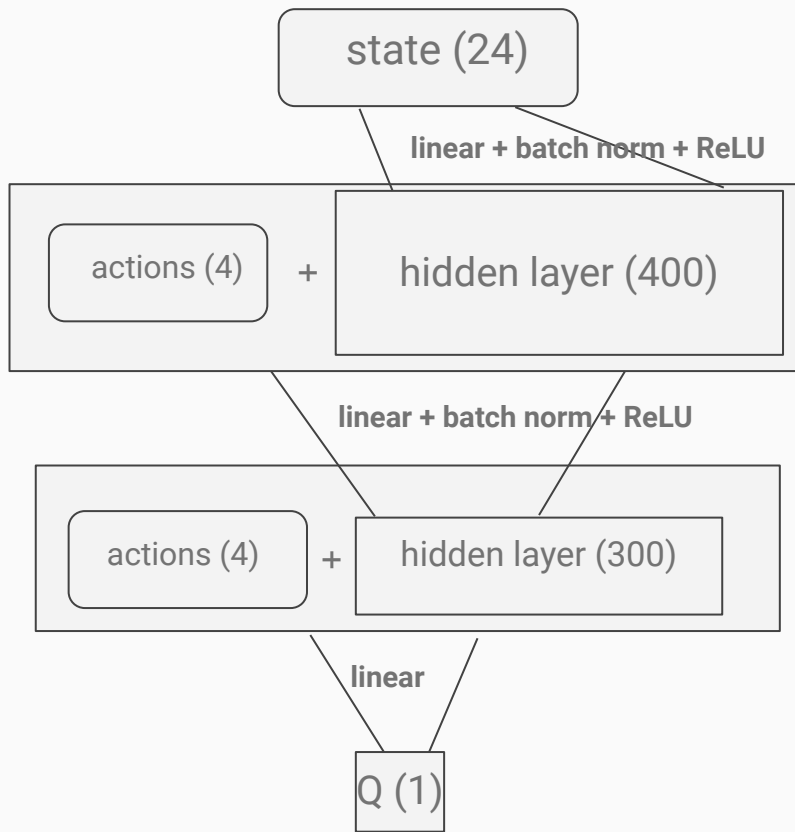
$$\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$$

$$\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$$

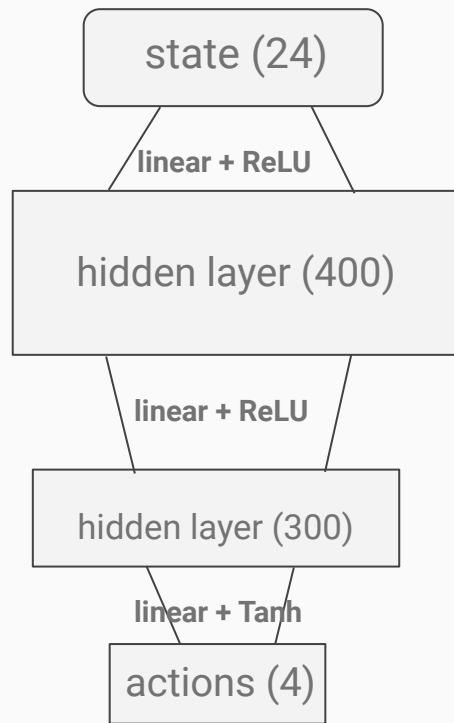
end if

end for

Critic - TD3/DDPG(same)



Actor TD3/DDPG(same)



The end.