Reinforcement learning for legged robots

Stéphane Caron

November 16, 2023

Inria, École normale supérieure

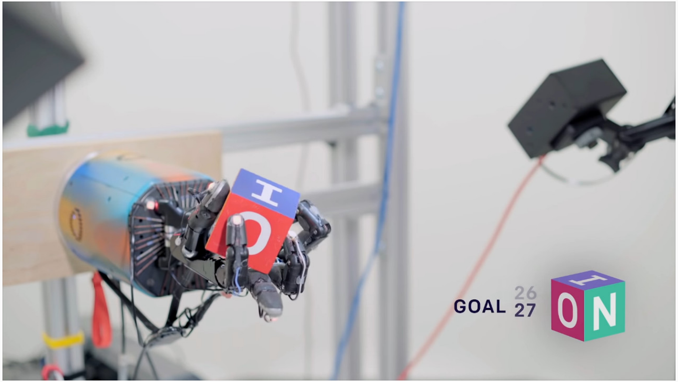
RL in robotics

2020: Quadrupedal locomotion

Teacher-student residual reinforcement learning [Lee+20]

Video: `https://youtu.be/oPNkeoGMvAE`

2018: In-hand reorientation



LSTM policy with domain randomization [And+20]

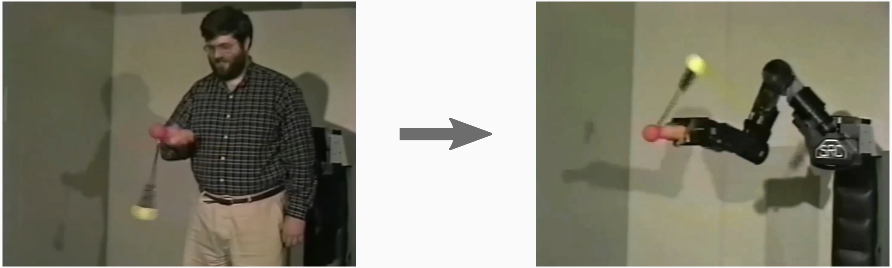
Video: `https://youtu.be/jwSbzNHGflM`

2010: Helicopter stunts

Helicopter aerobatics through apprenticeship learning [ACN10]

Video: `https://youtu.be/M-QUkgk3HyE`

1997: Pendulum swing up

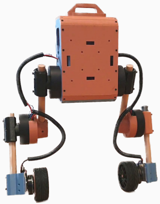


Swinging up an inverted pendulum from human demonstrations [AS97]

Video: `https://youtu.be/g3I2VjeSQUM?t=294`

Basics of reinforcement learning

Scope



Agent

Environment

action $a \in \mathcal{A}$

observation $o \in \mathcal{O}$

reward $r \in \mathbb{R}$

Rewards

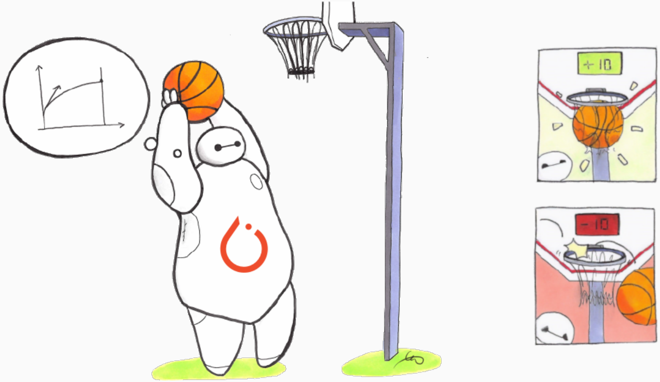
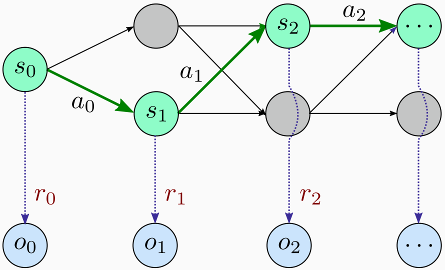


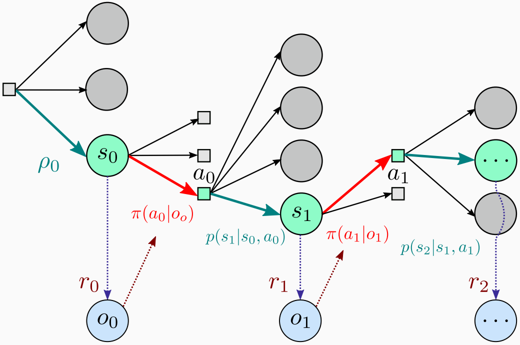
Image credit: L. M. Tenkes, source: `https://araffin.github.io/post/sb3/`

Partially observable Markov decision process (1/2)



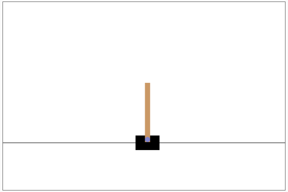
- **State:** s_t, ground truth of the environment
- **Action:** a_t, decision of the agent (discrete or continuous)
- **Observation:** o_t, *partial* estimation of the state from sensors
- **Reward:** $r_t \in \mathbb{R}$, scalar feedback, often $r_t = r(s_t, a_t)$ or $r(s_t, a_t, s_{t+1})$

Partially observable Markov decision process (2/2)



	Deterministic	*Stochastic*	
Model:	$s_{t+1} = f(s_t, a_t)$	$s_{t+1} \sim p(\cdot\|s_t, a_t)$	how the environment evolves
Initial state:	s_0	$s_0 \sim \rho_0(\cdot)$	where we start from
Observation:	$o_t = h(s_t)$	$o_t \sim z(\cdot\|s_t)$	how sensors measure the world
Policy:	$a_t = g(s_t)$	$a_t \sim \pi(\cdot\|o_t)$	what the agent decides

Example: The Gymnasium API



```python
import gymnasium as gym

with gym.make("CartPole-v1", render_mode="human") as env:
    env.reset()
    action = env.action_space.sample()
    for step in range(1000):
        observation, reward, terminated, truncated, _ = env.step(action)
        if terminated or truncated:
            observation, _ = env.reset()
        cart_position = observation[0]
        action = 0 if cart_position > 0.0 else 1
```

Same API for simulation and real robots

```python
import gymnasium as gym

with gym.make("UpkieGroundVelocity-v1", frequency=200.0) as env:
    env.reset()
    action = 0.0 * env.action_space.sample()
    for step in range(1_000_000):
        observation, reward, terminated, truncated, _ = env.step(action)
        if terminated or truncated:
            observation, _ = env.reset()
        pitch = observation[0]
        action[0] = 10.0 * pitch  # action is [ground_velocity]
```

Goal of reinforcement learning

Two last missing pieces:

- **Episode:** $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ truncated or infinite[1]
- **Return:** $R(\tau) = \sum_{t \in \tau} r_t$ or with discount $\gamma \in]0, 1[$: $R(\tau) = \sum_{t \in \tau} \gamma^t r_t$

We can now state what reinforcement learning is about:

Goal of reinforcement learning

The goal of reinforcement learning is to *find a policy that maximizes returns.*

[1] In practice episodes contain o_t rather than s_t. In RL, we implicitly assume that observations contain enough information to be in bijection with their corresponding states. See also *Augmenting observations* thereafter.

Stochastic reinforcement learning

In the stochastic setting, the goal of reinforcement learning is:

$$\begin{aligned}
\max_{\pi}\ & \mathbb{E}_{\tau}[R(\tau)] \\
\text{s.t. }\ & \tau = (s_0, a_0, s_1, a_1, \ldots) \\
& s_0 \sim \rho_0(\cdot) \\
& o_0 \sim z(\cdot | s_0) \\
& a_0 \sim \pi(\cdot | o_0) \\
& s_1 \sim p(\cdot | s_0, a_0) \\
& \vdots
\end{aligned}$$

Value functions (1/2)

State value functions V:

- **On-policy:** expected return from a given policy: $V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi}(R(\tau)|s_0 = s)$
- **Optimal:** best return we can expect from a state: $V^{*}(s) = \max_{\pi} \mathbb{E}_{\tau \sim \pi}(R(\tau)|s_0 = s)$

State-action value functions Q:

- **On-policy:** expected return from following policy:
  $Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim \pi}(R(\tau)|s_0 = s, a_0 = a)$
- **Optimal:** best return we can expect: $Q^{*}(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim \pi}(R(\tau)|s_0 = s, a_0 = a)$

Value functions (2/2)

Value functions satisfy the Bellman equation:

Bellman equation

$$V^*(s) = \max_\pi \mathbb{E}_{a \sim \pi(\cdot|s), (r, s') \sim p(s'|s,a)} [r + \gamma V^*(s')]$$

This is a connection to optimal control (*e.g.* differential dynamic programming) and Q-learning, but not our topic today.

Components of an RL algorithm

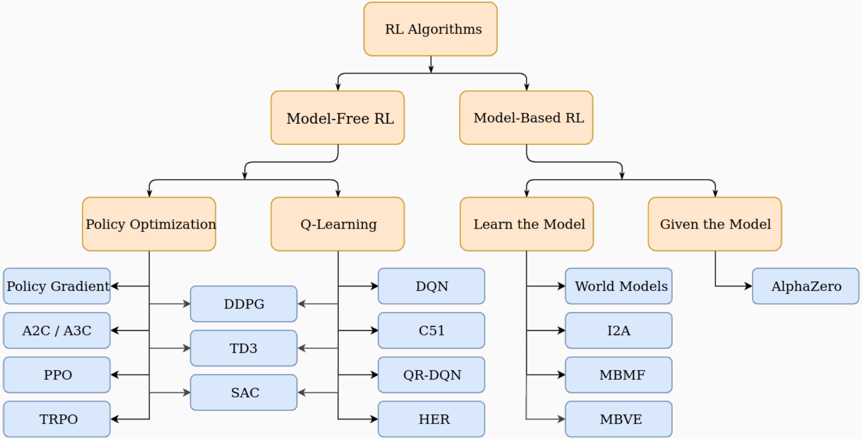
A reinforcement-learning algorithm may include any of the following:

- **Policy:** function approximator for the agent's behavior
- **Value function:** function approximator for the value of states
- **Model:** representation of the environment

An algorithm with a policy (actor) and a value function (critic) is called *actor-critic*.

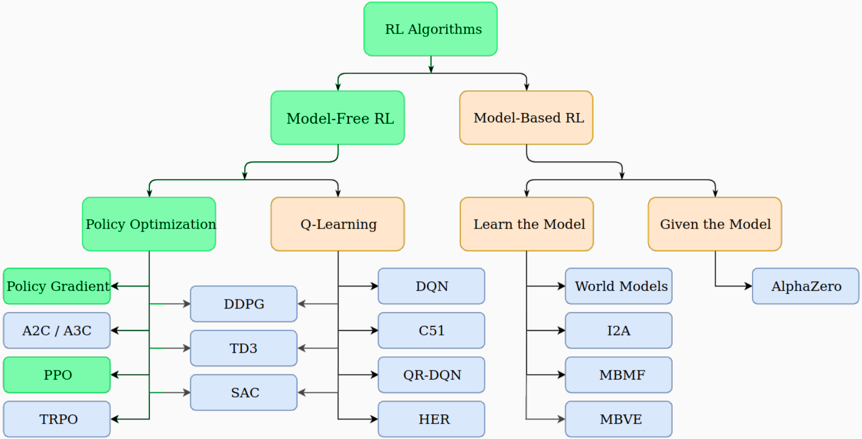
An algorithm with/without an explicit model is called *model-based*/*free*.

A taxonomy of RL algorithms



There are several taxonomies, none of them fully works. This one is from [Ach18].

A taxonomy of RL algorithms



Our focus in what follows.

Policy optimization

Policy-based algorithms

Policy-based algorithms update policy parameters θ iteratively. At each iteration k:

- Collect episodes $\mathcal{D}_k = \{\tau\}$
- Update the policy $\pi_{\theta_{k+1}} = update(\pi_{\theta_k}, \mathcal{D}_k)$

Two ways to collect episodes:

- **On-policy:** episodes $\mathcal{D}_k$ are collected with the latest policy π_k
- **Off-policy:** episodes $\mathcal{D}_k$ are collected with any policy

Policy optimization

The goal of RL is to find a policy that maximizes the expected return. In terms of θ:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

In policy optimization, we seek an optimum by gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

The gradient $\nabla_\theta J$ with respect to policy parameters θ is called the *policy gradient*.

Policy gradient theorem

Policy gradient theorem

The policy gradient can be computed from returns and the log-policy gradient $\nabla_\theta \log \pi_\theta$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left(R(\tau) \sum_{s_t, a_t \in \tau} \nabla_\theta \log \pi_\theta(a_t | s_t) \right)$$

LHS: the graal. RHS: things we observe ($R(\tau)$) or know by design ($\nabla_\theta \log \pi_\theta$).

Policy gradient theorem: proof sketch

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}(R(\tau)) && \text{definition} \\
&= \nabla_\theta \int_\tau R(\tau)\mathbb{P}(\tau|\theta)\mathrm{d}\tau && \text{expectation as integral} \\
&= \int_\tau R(\tau)\nabla_\theta \mathbb{P}(\tau|\theta)\mathrm{d}\tau && \text{Leibniz integral rule} \\
&= \int_\tau R(\tau)\mathbb{P}(\tau|\theta)\nabla_\theta \log \mathbb{P}(\tau|\theta)\mathrm{d}\tau && \text{log-derivative trick} \\
&= \int_\tau R(\tau) \sum_{s_t, a_t \in \tau} \nabla_\theta \log \pi_\theta(a_t|s_t)\mathbb{P}(\tau|\theta)\mathrm{d}\tau && \text{expand } \mathbb{P}(\tau|\theta) \text{ as product} \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left(R(\tau) \sum_{s_t, a_t \in \tau} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) && \text{integral as expectation}
\end{aligned}
$$

REINFORCE (1/2)

REINFORCE algorithm [SB18]

Data: initial policy parameters θ_0, learning rate α
Initialize policy parameters θ (e.g. to 0);
for $k = 0, 1, 2, \ldots$ **do**
  Roll out an episode $\tau = (o_0, a_0, \ldots, o_N, a_N)$ following π_{θ_k};
  for *each step* $t \in \tau$ **do**
    $R \leftarrow \sum_{t'=t+1}^{N} \gamma^{t'-t-1} r_{t'}$;
    $\theta \leftarrow \theta + \alpha \gamma^t R \nabla_\theta \log \pi_\theta(a_t | s_t)$
  end
end

REINFORCE (2/2)

Gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

From the policy gradient theorem, this is equivalent to:

$$\theta_{k+1} = \theta_k + \alpha \mathbb{E}_{\tau \sim \pi_\theta} \left(R(\tau) \sum_{s_t, a_t \in \tau} \nabla_\theta \log \pi_\theta(a_t | s_t) \right)$$

REINFORCE drops the expectation:

$$\theta_{k+1} = \theta_k + \alpha R(\tau_k) \sum_{s_t, a_t \in \tau_k} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Vanilla policy gradient [Ach18]

Data: initial policy parameters θ_0, initial value function parameters ϕ_0, learning rate α

for $k = 0, 1, 2, \dots$ **do**

Collect episodes $\mathcal{D}_k = \{\tau_i\}$ by running $\pi_\theta = \pi(\theta_k)$;

Compute returns $\hat{R}_t$ and advantage estimates $\hat{A}_t$ based on V_{ϕ_k};

Estimate the policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t$$

Update policy parameters by *e.g.* gradient ascent, $\theta_{k+1} = \theta_k + \alpha \hat{g}_k$;

Fit value function by regression on mean-square error:

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

end

Proximal policy optimization [Sch+17]

Data: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \dots$ **do**

Collect episodes $\mathcal{D}_k = \{\tau_i\}$ by running $\pi_\theta = \pi(\theta_k)$;

Compute returns $\hat{R}_t$ and advantage estimates $\hat{A}_t$ based on V_{ϕ_k};

Clipping: Update policy parameters by maximizing the clipping objective:

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

where $g(\epsilon, A) = (1+\epsilon)A$ if $A \geq 0$ else $(1-\epsilon)A$

Fit value function by regression on mean-square error:

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

end

Intuition behind clipping

When the advantage is positive:

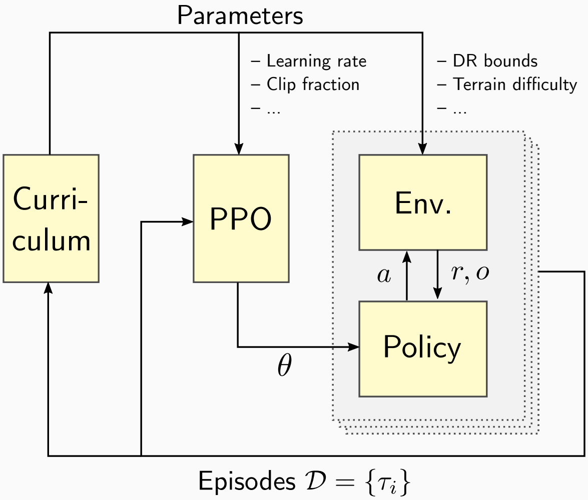
$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

The objective increases if the action becomes more likely $\pi_\theta(a|s) > \pi_{\theta_k}(a|s)$, but no extra benefit as soon as $\pi_\theta(a|s) > (1+\epsilon)\pi_{\theta_k}(a|s)$.

When the advantage is negative: *idem mutatis mutandis*.

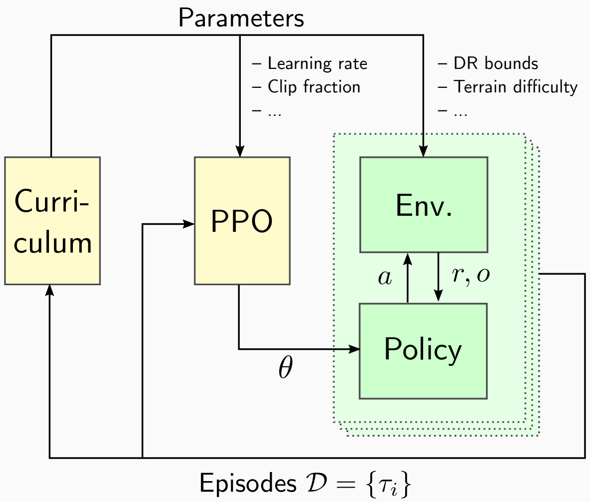
Training with PPO

Training



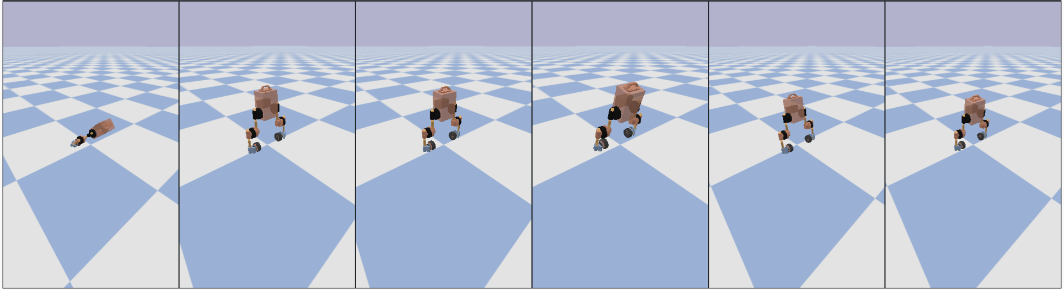
Parameters
– Learning rate
– Clip fraction
– ...
– DR bounds
– Terrain difficulty
– ...
Curri-
culum
PPO
Env.
Policy
a
r, o
θ
Episodes $\mathcal{D} = \{\tau_i\}$

Environment

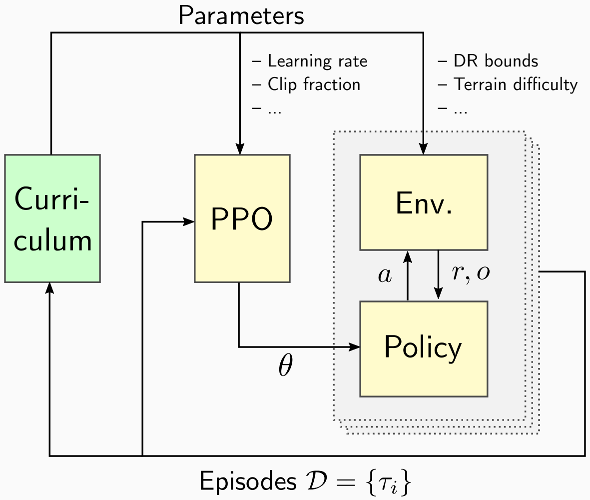


Rolling out episodes with a simulator

```
~/src/upkie > playground > ./tools/bazel run //agents/ppo_balancer:train -- --nb-envs 6 --show
INFO: Analyzed target //agents/ppo_balancer:train (108 packages loaded, 17832 targets configured).
INFO: Found 1 target...
Target //agents/ppo_balancer:train up-to-date:
  bazel-bin/agents/ppo_balancer/train
INFO: Elapsed time: 4.220s, Critical Path: 0.16s
INFO: 1 process: 1 internal.
INFO: Build completed successfully, 1 total action
INFO: Running command line: bazel-bin/agents/ppo_balancer/train --nb-envs 6 --show
[2023-11-14 11:31:04,519] [info] Logging training data in /home/scaron/src/upkie/training/2023-11-14 (train.py:365)
[2023-11-14 11:31:04,519] [info] To track in TensorBoard, run `tensorboard --logdir /home/scaron/src/upkie/training/2023-11-14` (train.py:366)
[2023-11-14 11:31:04,524] [info] New policy name is "marshiest" (train.py:236)
[2023-11-14 11:31:04,524] [info] Training data will be logged to /home/scaron/src/upkie/training/2023-11-14/marshiest_1 (train.py:237)
[2023-11-14 11:31:04,550] [info] Waiting for spine /monogamous to start (trial 1 / 10)... (spine_interface.py:46)
[2023-11-14 11:31:04,552] [info] Waiting for spine /rundown to start (trial 1 / 10)... (spine_interface.py:46)
[2023-11-14 11:31:04.554] [info] Command line: shm_name = /monogamous
[2023-11-14 11:31:04.554] [info] Command line: nb_substeps = 5
[2023-11-14 11:31:04.554] [info] Command line: spine_frequency = 1000 Hz
[2023-11-14 11:31:04.554] [warning] [Joystick] Observer disabled: no joystick found at /dev/input/js0
startThreads creating 1 threads.
starting thread 0
started thread 0
argc=2
argv[0] = --unused
argv[1] = --start_demo_name=Physics_Server
```

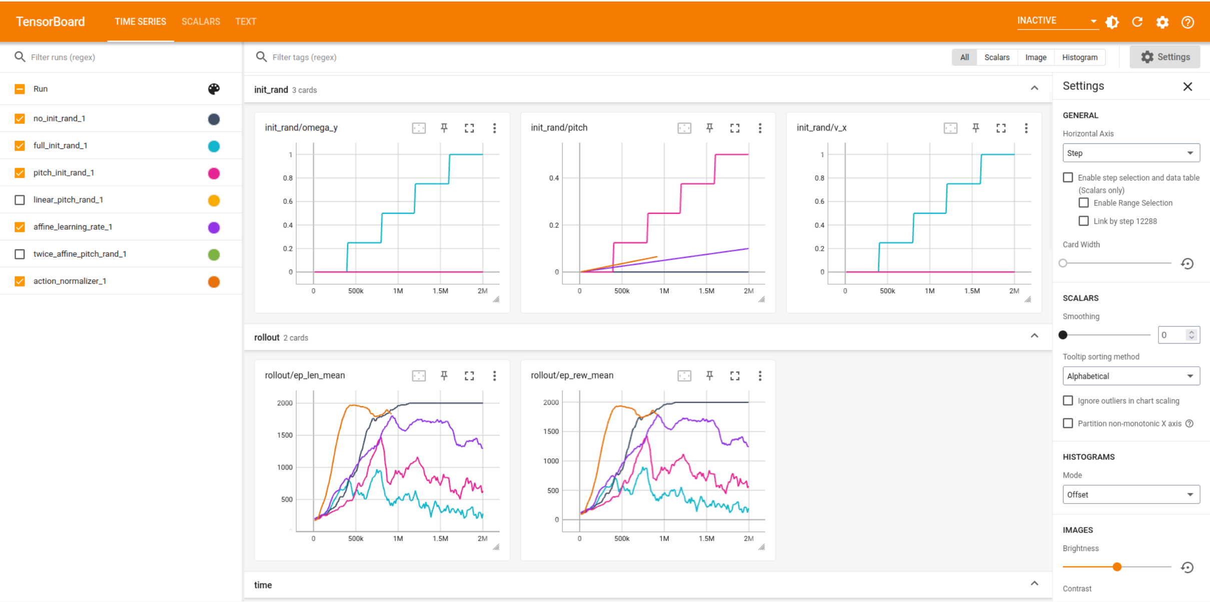


Curriculum



Parameters
– Learning rate
– Clip fraction
– ...
– DR bounds
– Terrain difficulty
– ...
Curri-
culum
PPO
Env.
Policy
a
r, o
θ
Episodes $\mathcal{D} = \{\tau_i\}$

Monitoring training



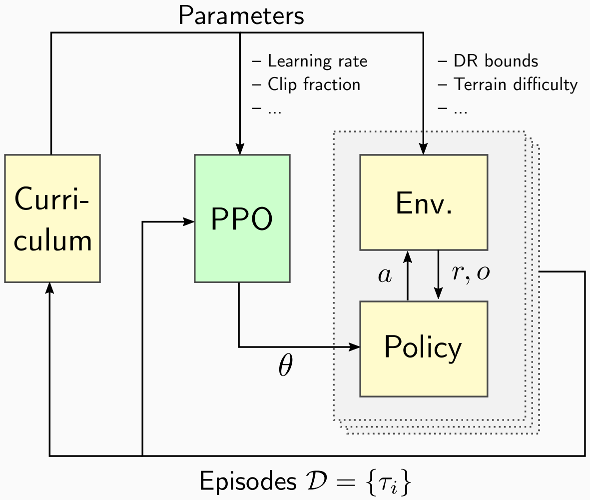
Episode metrics

Two main episode metrics:

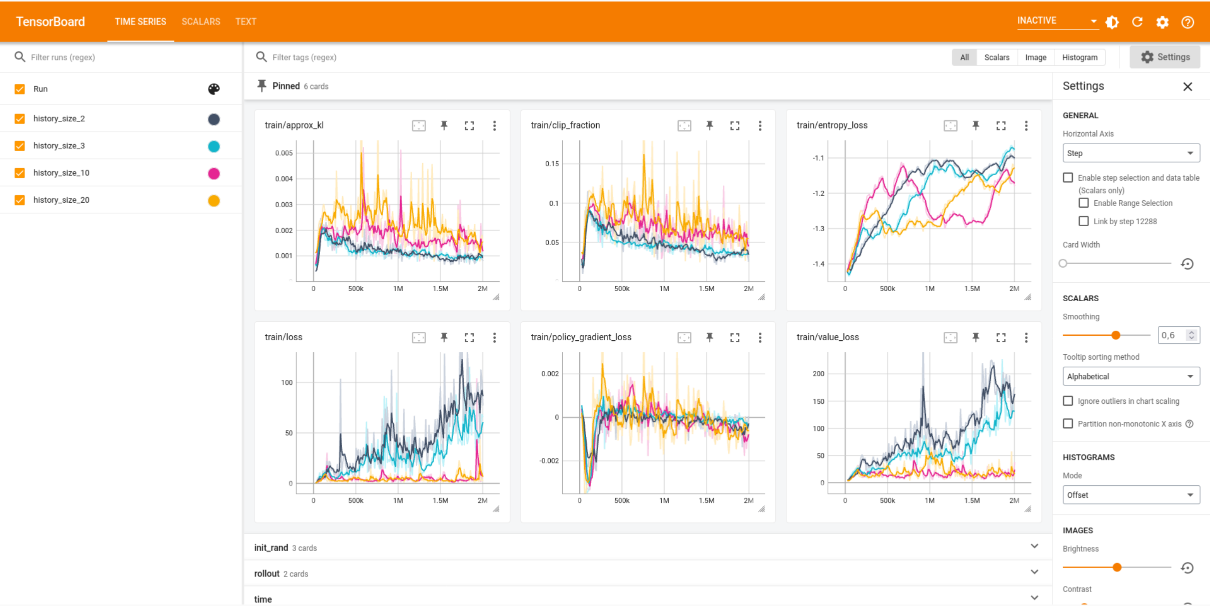
- `ep_len_mean` : average length of an episode, in number of environment steps.
- `ep_rew_mean` : average return of an episode.

If training goes well, both eventually plateau at their maximum values.

Training with PPO



Monitoring PPO



PPO loss function

Surrogate loss of PPO

```
loss = policy_gradient_loss + ent_coef * entropy_loss + vf_coef * value_loss
```

- `policy_gradient_loss` : regular loss resulting from episode returns.
- `entropy_loss` : negative of the average policy entropy. It should increase to zero over training as the policy becomes more deterministic.
- `value_loss` : value function estimation loss, *i.e.* error between the output of the function estimator and Monte-Carlo or TD(GAE lambda) estimates.

PPO hyperparameters

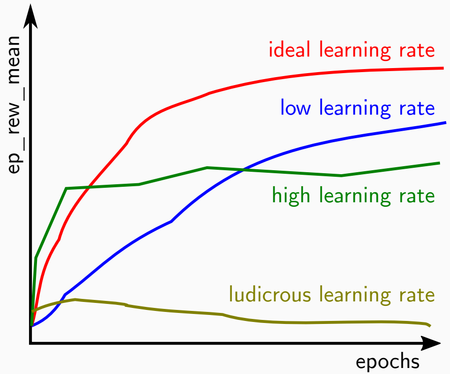
The PPO implementation in Stable Baselines3 has > 25 parameters, including:

- `clip_range`: clipping factor in policy loss.
- `ent_coef`: weight of entropy term in the surrogate loss.
- `gae_lambda`: parameter of Generalized Advantage Estimation.
- `net_arch_pi`: policy network architecture.
- `net_arch_vf`: value network architecture.
- `normalize_advantage`: use advantage normalization?
- `vf_coef`: weight of value-function term in the surrogate loss.

Optimizer parameters: steps, epochs, mini-batching

The optimizer behind PPO, usually Adam [KB14], comes with parameters:

- `learning_rate` : step size parameter, typically decreasing with a linear schedule.
- `n_steps` : number of environment steps to collect per rollout buffer.
- `n_epochs` : number of uses of the rollout buffer while optimizing the surrogate loss.
- `batch_size` : mini-batch size, same as in Stochastic Gradient Descent.



PPO health metrics

Finally, some metrics indicate whether training is going well:

- `approx_kl` : approximate KL divergence between the old policy and the new one.
- `clip_fraction` : mean fraction of policy ratios that were clipped.
- `clip_range` : value of the clipping factor for the policy ratios.
- `explained_variance` : ≈ 1 when the value function is a good predictor for returns.

Application to robotics

Training a policy

General things to do when training a policy:

- Augment observations with history
- Observation-action normalization
- Curriculum learning

Augmenting observations with history

(Welcome to the worst slide of this deck!)

- **Lag of a system:** number of observations required to estimate its state
- We assume the Markov property: $\text{lag} = 1$
- Delays in control and physical systems increase their lag
- Counter-measure: augment observations with history, restore the Markov property

RL-trained policies can be remarkable state estimators!

Observation-action normalization

Specific to deep RL, normalize the:

- **Observation space:** bound physical quantities, when possible.
- **Action space:** rescaling to $[-1, 1]$ is common practice.

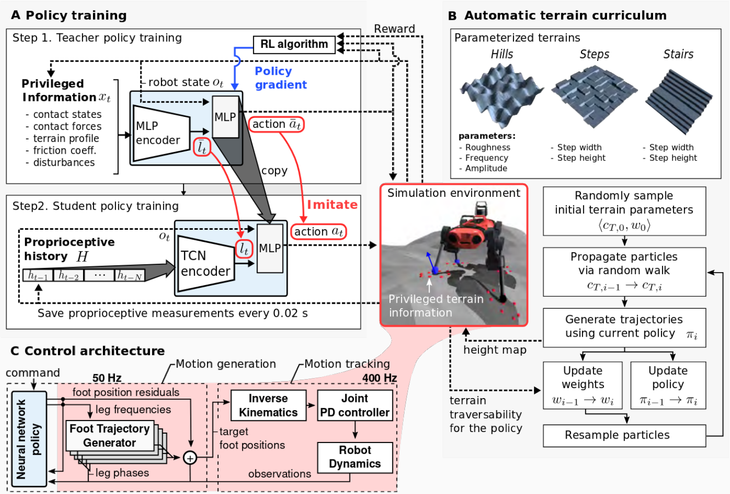
Unnormalized actions don't work well on actors with Gaussian outputs (as in PPO):

- Limits too large $\Rightarrow$ sampled actions cluster around zero.
- Limits too small $\Rightarrow$ sampled actions saturate all the time, *bang-bang* behavior.

Curriculum learning

Domain randomization and task difficulty vary based on policy performance.

Example: terrain curriculum for quadrupedal locomotion [Lee+20]:



Sim-to-real gap

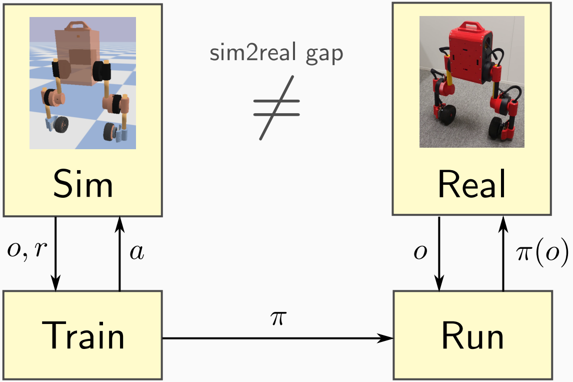


Figure 1: The “sim-to-real gap” is a metaphor for model mismatch.

Crossing the sim-to-real gap

To improve state observation:

- Teacher-student distillation

To learn a more robust[2] policy:

- Domain randomization
- Data-based simulation
- Reward shaping

[2] Robustness over model variations.

Domain randomization

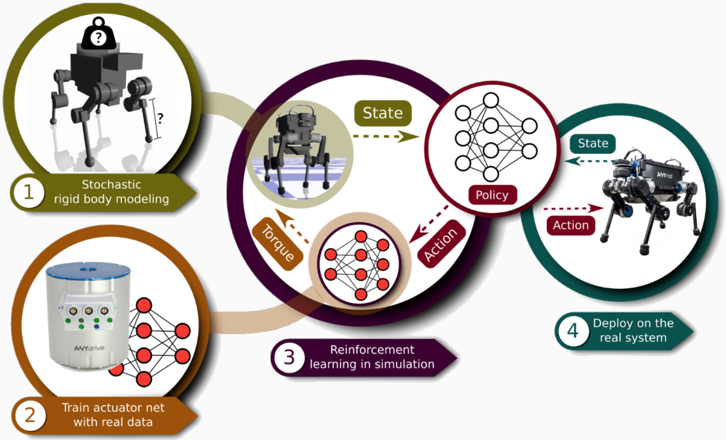
Randomize selected environment parameters:

- **Robot geometry:** limb lengths, wheel diameters, ...
- **Inertias:** masses, mass distributions
- **Initial state:** $s_0 \sim \rho_0(\cdot)$
- **Actuation models:** delays, bandwidth, ...
- **Perturbations:** send $(1 \pm \epsilon)\tau$ torques...

Domain randomization makes policies more conservative.

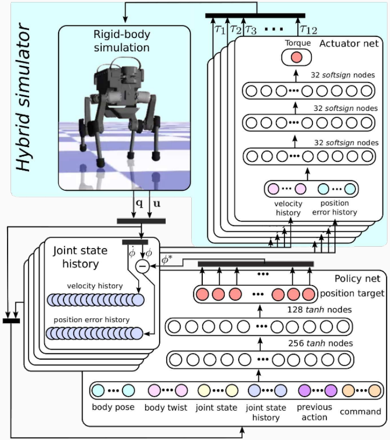
It slows down or completely hampers training.

Data-based actuation models (1/2)



[3] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019).

Data-based actuation models (2/2)



[4] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019).

Reward shaping

Let r_e denote the reward associated with an error function e:

Motivation:

- Exponential: $r_e = \exp(-e^2)$

Penalization:

- Absolute value $r_e = -|e|$
- Squared value: $r_e = -e^2$

RewArt

Making an RL pipeline work can lead to complex rewards, e.g. in [Lee+20]:

- Linear velocity tracking: $r_{lv} = \exp(-2.0(v_{pr} - 0.6)^2)$, or 1, or 0
- Angular velocity tracking: $r_{av} = \exp(-1.5(\omega_{pr} - 0.6)^2)$, or 1
- Base motion tracking: $r_b = \exp(-1.5v_o^2) + \exp(-1.5\|({}_{IB}^{B}\omega)_{xy}\|^2)$
- Foot clearance: $r_{fc} = \sum_{i \in I_{swing}} \mathbf{1}_{fclear}(i)/|I_{swing}|$
- Body-terrain collisions: $r_{bc} = -|I_{c,body} \setminus I_{c,foot}|$
- Foot acceleration smoothness: $r_s = -\|(r_{f,d})_t - 2(r_{f,d})_{t-1} + (r_{f,d})_{t-2}\|$
- Torque penalty: $r_\tau = -\sum_i |\tau_i|$

Final reward: $r = 0.05r_{lv} + 0.05r_{av} + 0.04r_b + 0.01r_{fc} + 0.02r_{bc} + 0.025r_s + 2 \cdot 10^{-5} r_\tau$

Example: Upkie ground velocity

Observation:

Index	Symbol	Description
0	θ	pitch from torso to inertial frame
1	p	ground position
2	$\dot{\theta}$	pitch angular velocity from torso to inertial frame
3	$\dot{p}$	ground velocity

Action:

Index	Symbol	Description
0	v	commanded ground velocity

Example: Rewards

RewArt is not a necessity: advocacy for simple rewards.

In the Upkie ground-velocity environment, we went for:

$$
\begin{aligned}
p_{tip} &= p + \ell \sin(\theta) \\
r(o) &:= \exp\left(-\left(\frac{p_{tip}}{\sigma}\right)^2\right)
\end{aligned}
$$

Including higher-order derivatives does not always help.

Example: environment wrappers

Composability of environments helps while prototyping. In the Upkie ground-velocity environment:

- Base environment: velocity commands
- (Training:) add action noise
- (Training:) add observation noise
- **(Training:) low-pass filter on wheel actuators**
- **Include last ten observation-action pairs in observation vector**
- Change action to ground acceleration (with bounds)
- **Rescale action between -1 and 1**

Keep in mind that we are in a stochastic world

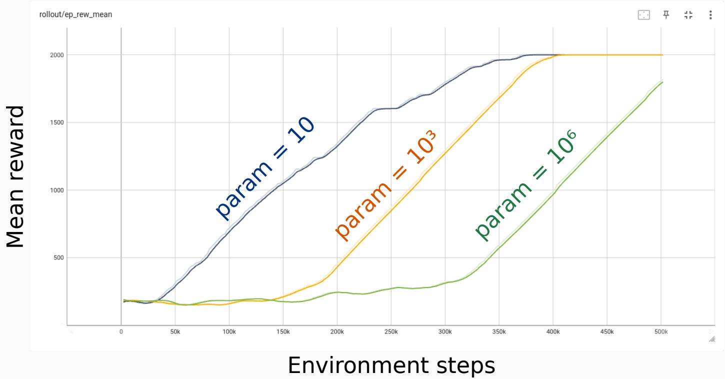


Figure 2: We may be observing the effect of our parameter.

Keep in mind that we are in a stochastic world

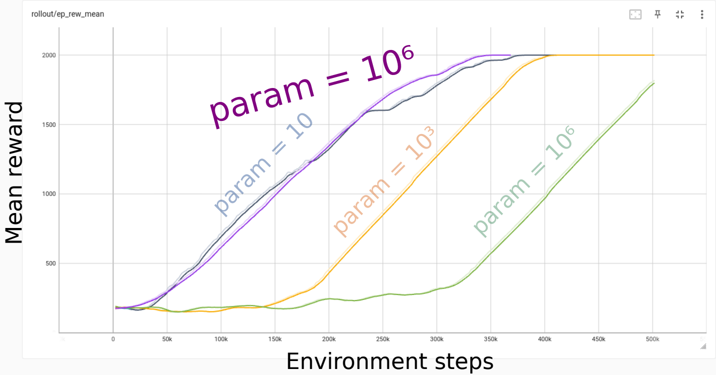


Figure 2: Or we may be observing the variance of the training process.

Challenges for RL

- **Sim-to-real:** no one-size-fits-all method, depends on robot and task
- **Jittering:** action noise, due to need for stochastic policies (PGT!)
- **Sample efficiency:** requires millions of samples (s_t, a_t, s_{t+1})
- **Reward shaping:** often helps, not a necessity, "RewArt"!
- **Curriculum learning:** humans monitoring computers that learn[5]

[5] Also known as SGD: "Student Grad Descent"

What did we see?

What we saw

Introduction to policy optimization:

- Partially-observable Markov decision process (POMDP)
- The goal of reinforcement learning
- Model, policy and value function
- Policy optimization: REINFORCE, policy gradient, PPO

Application to robotics:

- Training policies: add history, normalization, curriculum
- Sim-to-real gap: domain randomization, data-based sim, “RewArt”

RL is not magic: great results, possibly going to great lengths!

Thank you for your attention![6]

[6] Thanks to Elliot Chane-Sane, Thomas Flayols, Nicolas Perrin-Gilbert, Philippe Souères and the 2023 class at MVA for feedback on previous versions of these slides.

Bibliography

References i

[Ach18] Josh Achiam. *Spinning Up in Deep Reinforcement Learning*. `https://spinningup.openai.com/`. 2018.

[ACN10] Pieter Abbeel, Adam Coates, and Andrew Y Ng. "Autonomous helicopter aerobatics through apprenticeship learning". In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.

[And+20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[AS97] Christopher G Atkeson and Stefan Schaal. "Robot learning from demonstration". In: *ICML*. Vol. 97. 1997, pp. 12–20.

[Hwa+19] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019).

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[Lee+20] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47 (2020).

References ii

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).