



**Machine Learning Applications
for Particle Accelerators**

[HOME](#)

[CONFERENCES](#)

[CONTACT US](#)

Advanced concepts for Reinforcement Learning

tcsc on ML 2024, Split, V. Kain, 13-19 Oct 2024

Recap

State-action value function:

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(s_{t'}, a_{t'}) \mid s_t, a_t]: \text{total reward from taking } a_t \text{ in } s_t$$

The value function:

$$V^\pi(s_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(s_{t'}, a_{t'}) \mid s_t]: \text{total reward from } s_t$$

$$V^\pi(s_t) = E_{a_t \sim \pi(a_t \mid s_t)} [Q^\pi(s_t, a_t)]$$

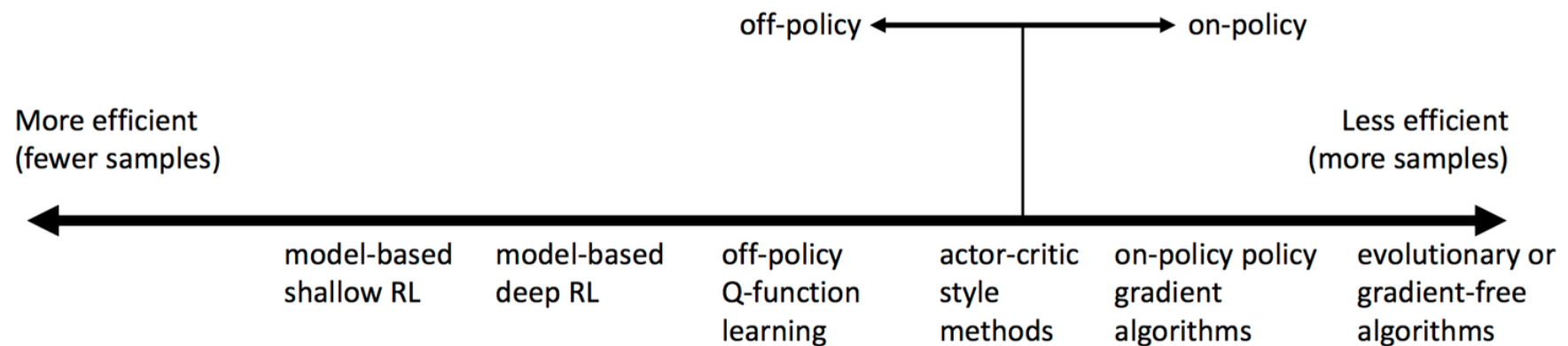
The advantage function: (advantage over taking a certain action over the average action)

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Sample efficiency

How many interactions does RL algorithm need until it has learned optimal policy/ Q -function/...?

Machine time is expensive. Some algorithms are excluded to train online!!

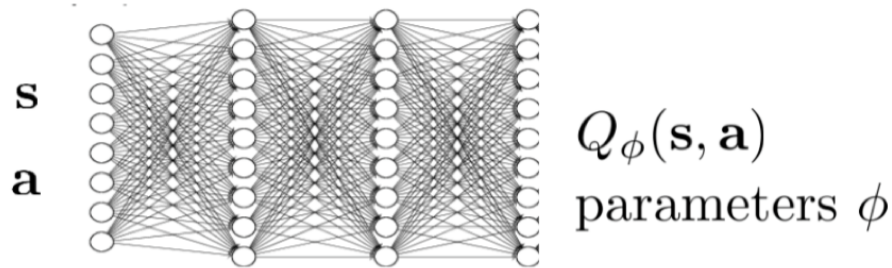


→ off-policy algorithms: Q -learning, actor-critic methods, model-based RL...

Continuous action space for accelerators

Sample-efficiency + continuous action space?

Basic Q -learning algorithm



1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$
- $K \times$

Issue for continuous action: $\max_a Q(s, a)$ in update rule and $\pi(s) = \arg \max_a Q(s, a)$; maximisation might not be straight forward for non-trivial Q

Q-learning with NAF

Various ways to overcome Q maximisation issue for continuous action space.

Most successful: actor-critic (e.g. DDPG)

For convex problems, can use a trick:

- Q -function assumed to belong to function class that is easy to optimise

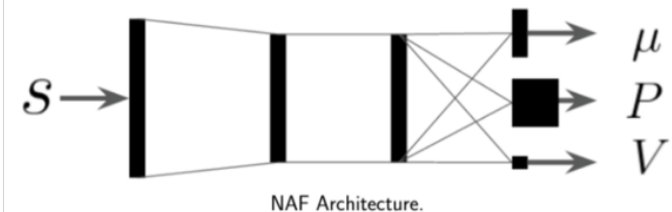
- * E.g. as a quadratic function of a

- E.g. NAF (Normalised Advantage Function) algorithm

- * $Q_\phi(s, a) = -\frac{1}{2}(a - \mu_\phi(s))^T P_\phi(s)(a - \mu_\phi(s)) + V_\phi(s)$

- * $\rightarrow \arg \max_a Q_\phi(s, a) = \mu_\phi(s)$

- * $\rightarrow \max_a Q_\phi(s, a) = V_\phi(s)$



Gu, Lillicrap, Sutskever, Levine, PMLR, 2016

RL “datasets” → problems → environments



API standard for reinforcement learning

OpenAI Gym → Gymnasium (Farama Foundation)



Most important methods and attributes:

gymnasium.Env

Env

Methods

`step()` **Updates an environment with actions, returning the next state/observation, the reward,...**

`reset()` **Resets the environment to an initial state. Is called at the beginning of episode, returns first state/observation.**

Attributes

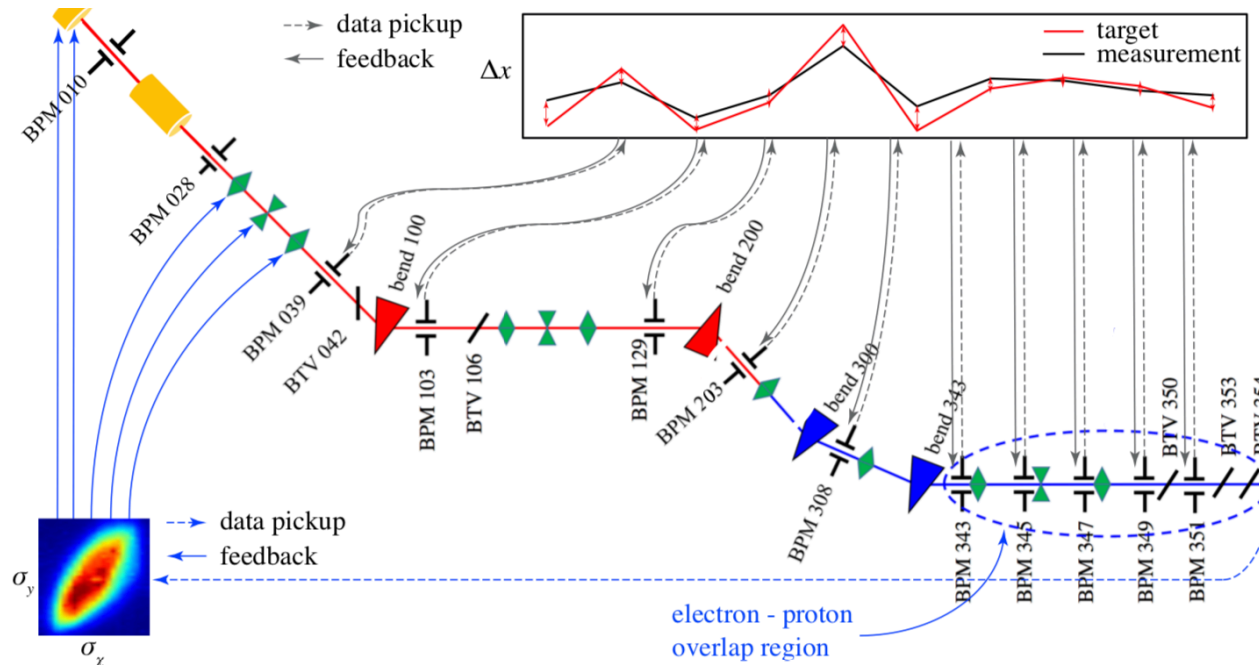
`Env.action_space`

`Env.observation_space`

Our MNIST-like problem for RL@ CERN: steering AWAKE e^- - line

The **A**dvanced **P**roton **D**riven **P**lasma **W**akefield **A**cceleration (AWAKE) Experiment investigates the use of plasma wakefields driven by a proton bunch to accelerate e^- .

Trajectory steering **environment** - fully observable: 10 beam position monitors (BPMs) for s , 10 dipole correction actors for a



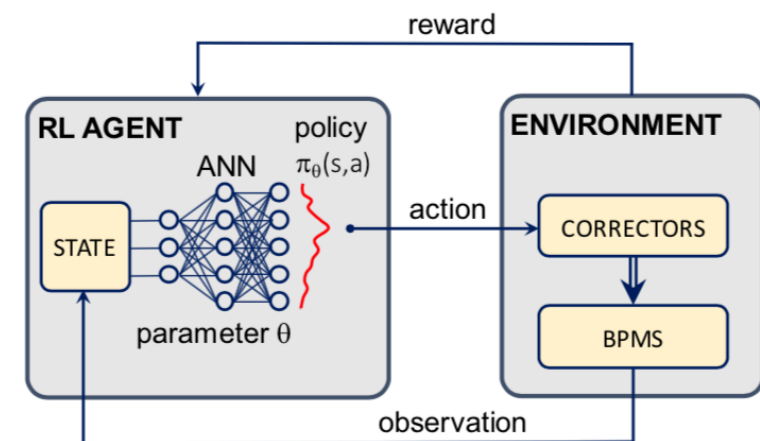
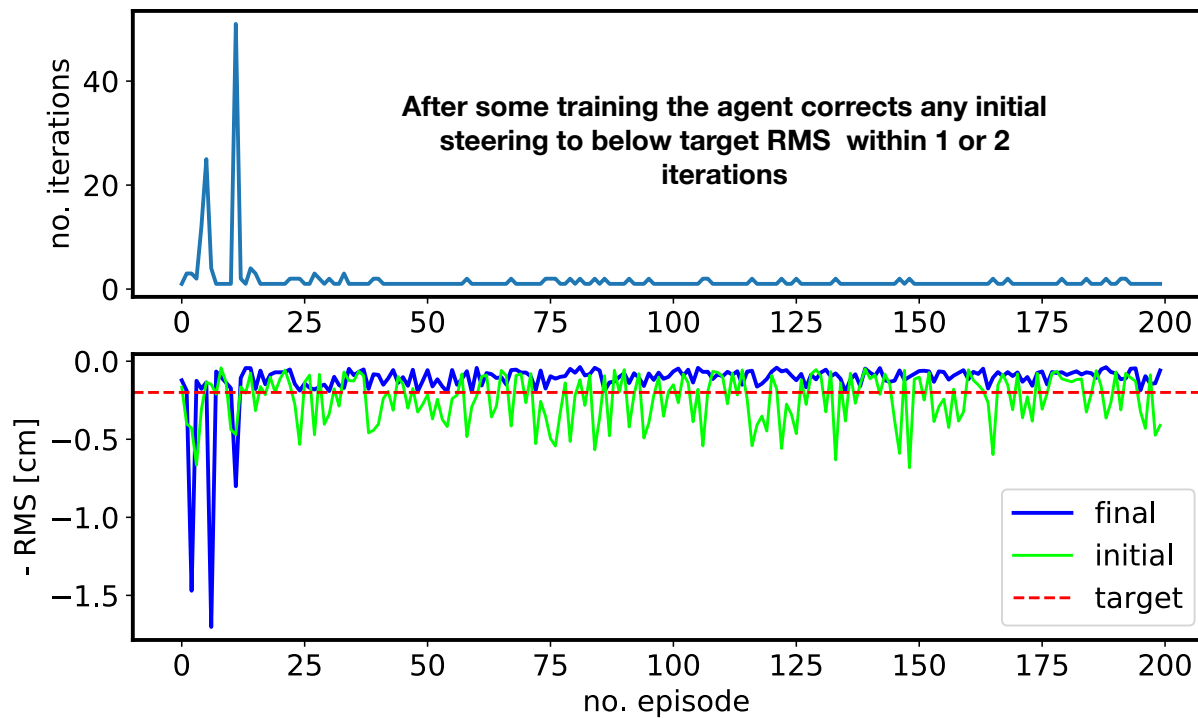
Courtesy A. Scheinker

First RL test at CERN...

Learn how to steer AWAKE e^- beam in horizontal plane.

Q -learning with very sample-efficient NAF algorithm

Data from 2019 online training at 10 Hz: *PhysRevAccelBeams.23.1248*



Reliable Reinforcement Learning Implementations

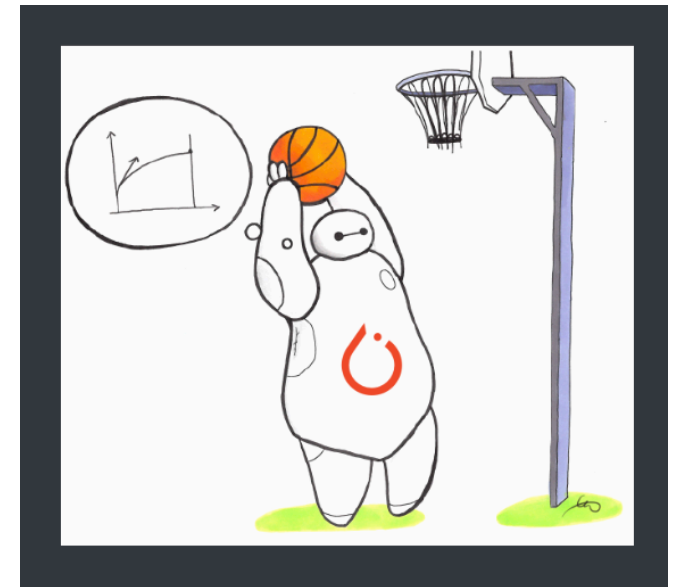
For deep RL, recommend to use Stable Baselines3 (SB3).

Solid implementations and lots of examples.

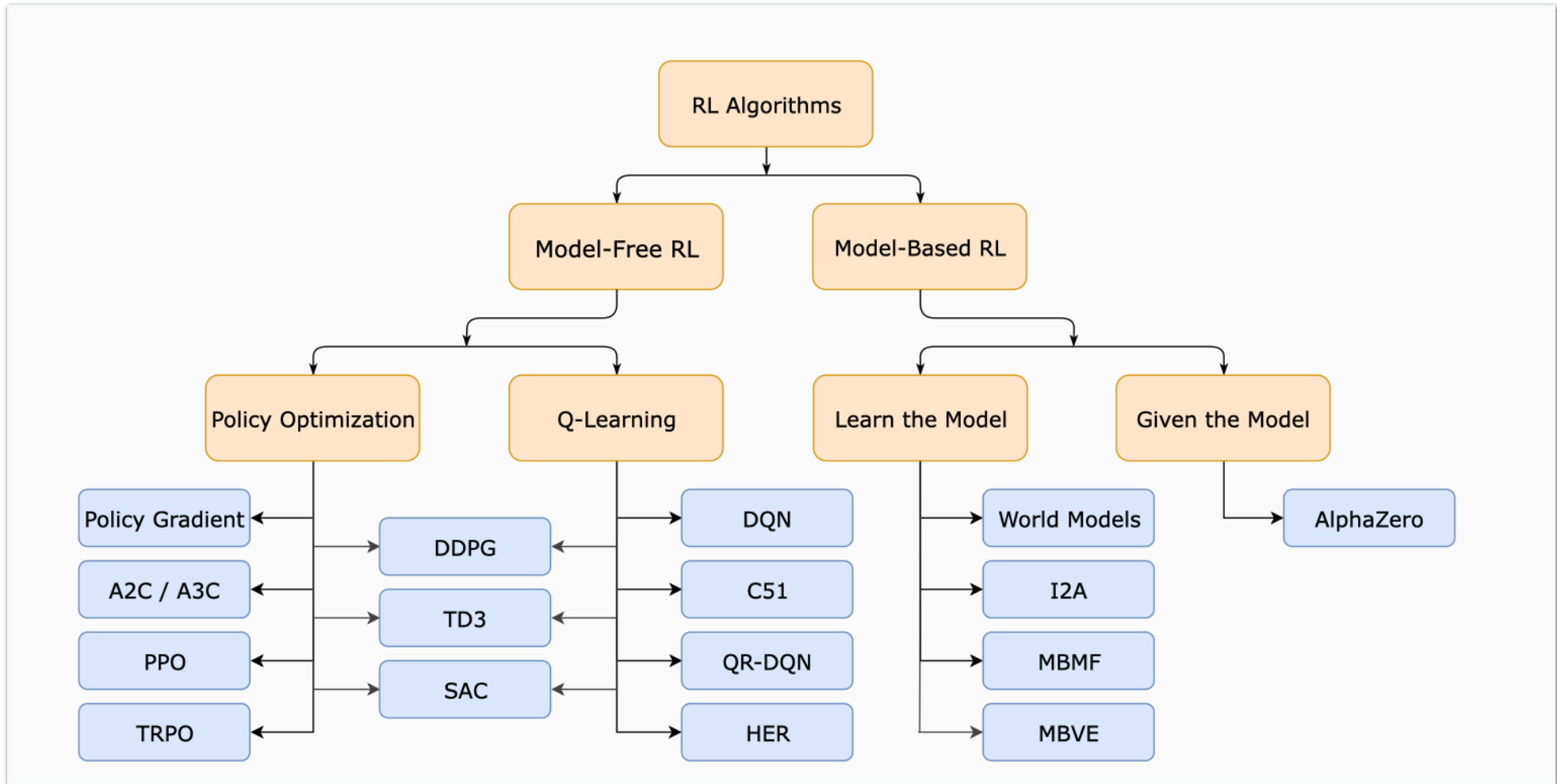
- New: first implementations towards SBX SB in Jax

Many state-of-the-art algorithms.

- DDPG, PPO, SAC, DQN, TD3,...



Zoo of algorithms...



From OpenAI spinning up

Recap

Reinforcement learning goal:

Find optimal policy π_θ that gives optimal trajectory p_θ

$$\underbrace{p_\theta(s_1, a_1, \dots, s_T, a_T)}_{p_\theta(\tau)} = p(s_1) \prod_{t=1}^T \pi_\theta(a_t | s_t) p(s_{t+1} | a_t, s_t)$$

meaning finding θ^* by optimising the objective $J(\theta)$

$$\theta^* = \arg \max_{\theta} \underbrace{\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]}_{J(\theta)}$$

$$J(\theta) \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \quad \dots \text{trajectory samples } i$$



Policy update: $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

$$J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)}[r(\tau)] = \int p_{\theta}(\tau) r(\tau) d\tau$$

$$\nabla_{\theta} J = \int \nabla_{\theta} p_{\theta}(\tau) r(\tau) d\tau = \int p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) r(\tau) d\tau = \mathbb{E}_{\tau \sim p_{\theta}(\tau)}[\nabla_{\theta} \log p_{\theta}(\tau) r(\tau)]$$

$$g = \nabla_{\theta} J = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{t=1}^T r(s_t, a_t) \right) \right]$$

$$p_{\theta}(\tau) \nabla_{\theta} \log p_{\theta}(\tau) = p_{\theta}(\tau) \frac{\nabla_{\theta} p_{\theta}(\tau)}{p_{\theta}(\tau)} = \nabla_{\theta} p_{\theta}(\tau)$$

$$\underbrace{p_{\theta}(s_1, a_1, \dots, s_T, a_T)}_{p_{\theta}(\tau)} = p(s_1) \prod_{t=1}^T \pi_{\theta}(a_t | s_t) p(s_{t+1} | a_t, s_t)$$

$$\log p_{\theta}(\tau) = \log p(s_1) + \sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) + \log p(s_{t+1} | a_t, s_t)$$

General form of policy gradient

the following:

$$g = \nabla_{\theta} J = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \underbrace{\left(\sum_{t=1}^T r(s_t, a_t) \right)}_{\Psi_t} \right] \text{ and } \Psi_t \text{ may be one of}$$

- | | |
|--|---|
| 1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory. | 4. $Q^{\pi}(s_t, a_t)$: state-action value function. |
| 2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t . | 5. $A^{\pi}(s_t, a_t)$: advantage function. |
| 3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula. | 6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual. |

This is the main theoretical foundation behind policy gradient algorithms.

Vanilla policy gradient algorithms have the issue of **high variance** for gradient estimation because of $\sum_t r_t = R_{\tau}$ term \rightarrow introducing a bias reduces variance.

Why does bias reduce variance?

$$\text{Var}(X) := \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

The expectation is unbiased (otherwise introducing bias would not be an option)...

$$\text{Var} \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t(\tau) - b(s_t)) \right) \approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau} \left[\left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t(\tau) - b(s_t)) \right)^2 \right]$$

Assume independence...

$$\approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau} \left[\left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right)^2 \right] \mathbb{E}_{\tau} \left[\left(R_t(\tau) - b(s_t) \right)^2 \right]$$

...a least square problem for $\mathbb{E}_{\tau} \left[\left(R_t(\tau) - b(s_t) \right)^2 \right]$...want $b(s_t) \approx \mathbb{E}[R_t(\tau)]$

REINFORCE algorithm

Algorithm

1. Initialise the policy parameters θ at random
2. Generate one trajectory with policy π_θ : $s_0, a_0, s_1, r_1, a_1, \dots, s_T$
3. for $t = 1, 2, \dots, T$:
 1. Estimate the return R_t
 2. Update policy parameters: $\theta \leftarrow \theta + \alpha \gamma^t R_t \nabla_\theta \ln \pi_\theta(a_t | s_t)$

Commonly used variation: subtract baseline value from R_t to reduce variance of gradient estimation.

With $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_t | s_t, a_t]$ use baseline corrected: $A(s, a) = Q(s, a) - V(s)$

Proximal Policy Optimisation (PPO) (J. Schulman et al., 2017)



To improve training stability, avoid parameter updates that change policy too much at one step!

Evolution of Trust Region Policy Optimisation (TRPO): $L^{TRPO}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{old}}(s, a)]$,
arXiv:1502.05477

Idea: introduce ratio $r(\theta) = \frac{\pi_{\theta}(a | s)}{\pi_{\theta_{old}}(a | s)}$ and clipped objective function for automatic differentiation

$L^{CLIP}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{old}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_{\theta_{old}}(s, a))]$... a lower bound of the L^{TRPO} .

Shared weights between policy and value functions, S is an entropy term to guide exploration:

$$L(\theta) = \mathbb{E}_t[L_t^{CLIP} - c_1 L_t^V(\theta) + c_2 S[\pi_{\theta}](s_t)]$$

PPO - Pseudocode

Pseudocode

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **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:

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

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**
-



Actor-Critic

....from the perspective of policy gradient.

Policy gradient methods have two ingredients: a policy model π_θ and the value function.

Knowing the value function can reduce variance during policy gradient updates.

→ learn the value function as well.

Critic updates the value function parameters: $Q_w(a|s)$ or $V_w(s)$

Actor updates policy parameters in direction suggested by critic: π_θ

Vanilla Actor-Critic

Algorithm:

1. Initialise s, θ, w at random; sample $a \sim \pi_{\theta}(a | s)$
2. For $t = 1 \dots T$:
 1. Sample reward $r_t \sim R(s, a)$ and $s' \sim P(s' | s, a)$
 2. Then sample the next action $a' \sim \pi_{\theta}(a' | s')$
 3. Update policy parameters: $\theta \leftarrow \theta + \alpha_{\theta} Q_w(s, a) \nabla_{\theta} \ln \pi_{\theta}(a | s)$
 4. Compute correction for critic (TD error): $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
 5. Update the weights of critic (check this line): $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
 6. Update $a \leftarrow a'$ and $s \leftarrow s'$

This algorithm is “on-policy”: training samples are collected according to the target policy (= the one we are optimising at the same time)



DDPG (Lillicrap et al, 2015)

Deep Deterministic Policy Gradient: model-free, off-policy, actor-critic

Q-function stabilised by experience replay and a slowly updating target networks

Deterministic policy $\mu(s)$; for exploration add noise \mathcal{N} : $\mu'(s) = \mu_\theta(s) + \mathcal{N}$

"soft updates" → conservative policy iteration: $\tau \ll 1$: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ (θ' ...weights of target networks)

DDPG pseudocode

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



TD3 (S. Fujimoto et al, 2018)

Q-learning is known for overestimation of the value function → Double Q-learning etc.

→ Twin Delayed Deep Deterministic (=TD3) = DDPG plus tricks

- 2 Q networks, Q_1, Q_2 : clipped double Q learning + target policy smoothing (with ϵ):

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \pi_{\phi}(s') + \epsilon) \dots \text{clipped random noise } \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$$

- Delayed updates of target and policy networks

TD3 pseudocode

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 $a \sim \pi(s) + \epsilon$,
 $\epsilon \sim \mathcal{N}(0, \sigma)$ 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}

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

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

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

if $t \bmod d$ **then** **Delayed update of target and policy networks**

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:

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

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

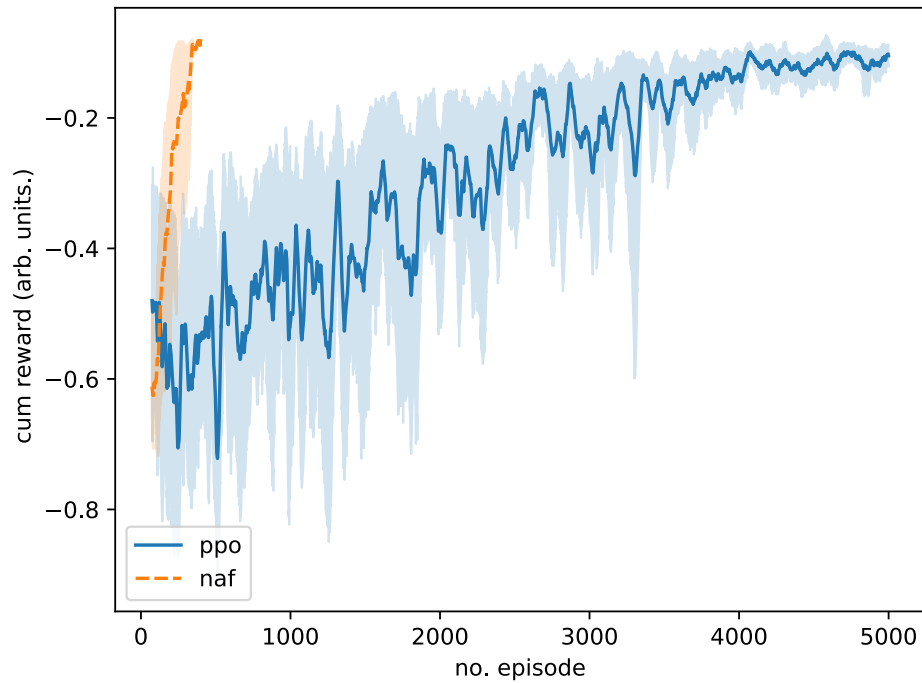
end if

end for

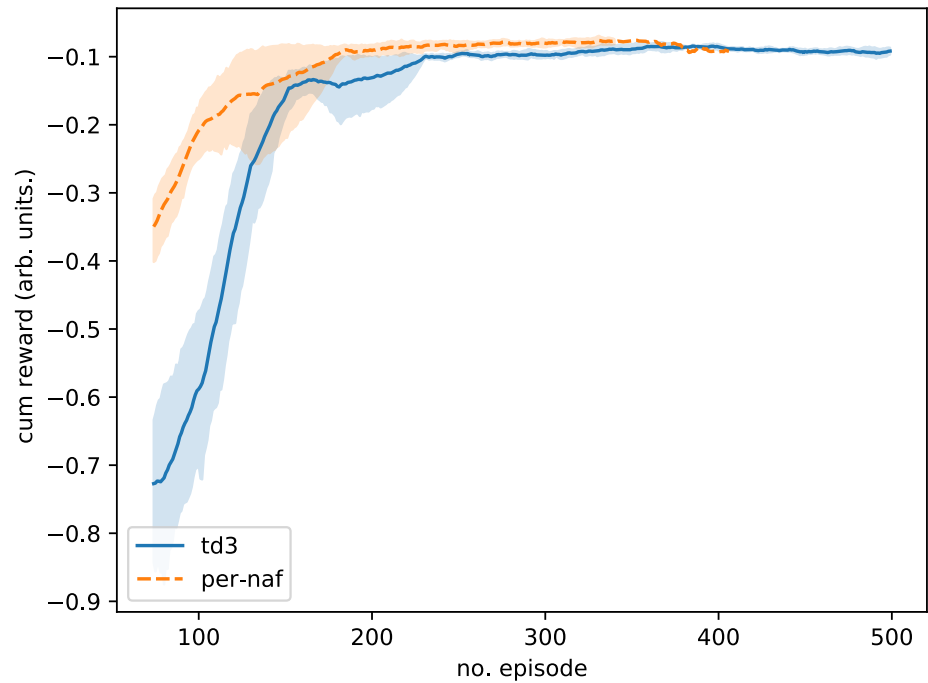
Comparison of algorithms - AWAKE steering



Policy-gradient algorithm PPO versus NAF for AWAKE steering problem in simulation:



TD3 versus NAF for AWAKE steering problem in simulation: similar performance





Model-based RL

Model-based RL

What if we knew the transition model $f(s_t, a_t) = s_{t+1}$?

- Often we do know the dynamics → Monte Carlo Search Trees, LQR, ...
 - * Games (chess,..)
 - * Easily modelled systems (e.g. navigating a car)
 - * Simulated environments (AWAKE steering environment,...)
- Often we learn the dynamics → will focus on this!
 - * System identification - fit unknown parameters of a known model
 - * Learning - fit a general-purpose model to observed transition data

Does this make things easier? Often, yes!

Let's learn the model...

Learn $f(s_t, a_t)$ from data and then plan through it

Algorithm:


1. run base policy $\pi_0(a_t | s_t)$ (e.g. random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2. learn dynamics $f(s, a)$ to minimise $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. plan through $f(s, a)$ to choose actions

...works particularly well if can go physics-based and just fit a few parameters of model.

Potential issue distribution mismatch, model bias from data collection process.

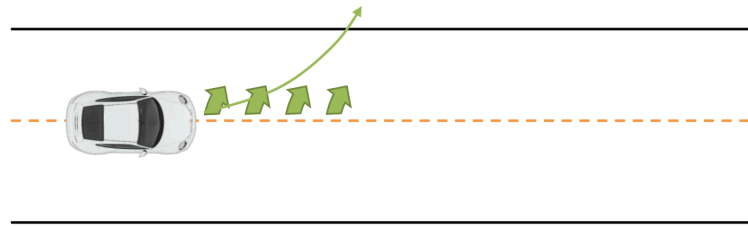
Improved algorithm 0.1

Collect data according to representative policy...

1. run base policy $\pi_0(a_t | s_t)$ (e.g. random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
 2. learn dynamics $f(s, a)$ to minimise $\sum_i \|f(s_i, a_i) - s'_i\|^2$
 3. plan through $f(s, a)$ to choose actions
 4. execute those actions add the resulting data $\{(s, a, s')_j\}$ to \mathcal{D}
- 

Improved algorithm 1.0

How to deal with model errors... long-term accumulation of errors during planning



We “replan”!

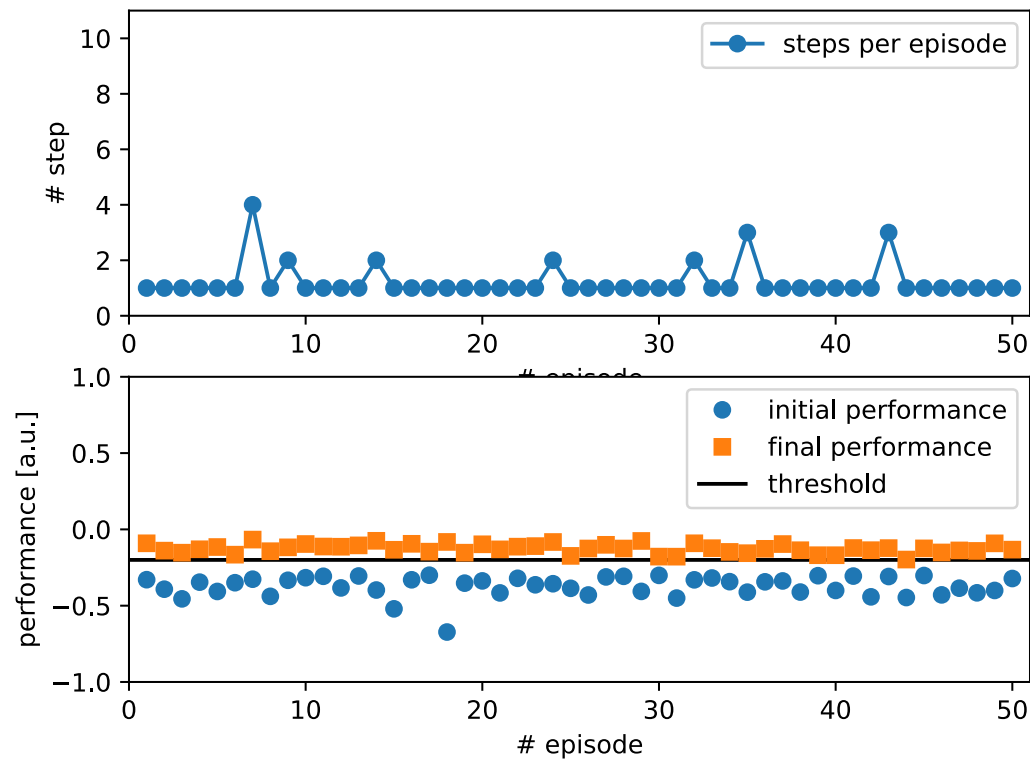
Every N steps

1. run base policy $\pi_0(a_t | s_t)$ (e.g. random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2. learn dynamics $f(s, a)$ to minimise $\sum_i \|f(s_i, a_i) - s'_i\|^2$
3. plan through $f(s, a)$ to choose actions
4. execute the **first** planned action, observe resulting state s' (MPC)
5. append $\{(s, a, s')_j\}$ to \mathcal{D}

Example: AWAKE steering

Number of training steps $N = 1$, i.e. learn dynamics only once. MPC algorithm: iLQR

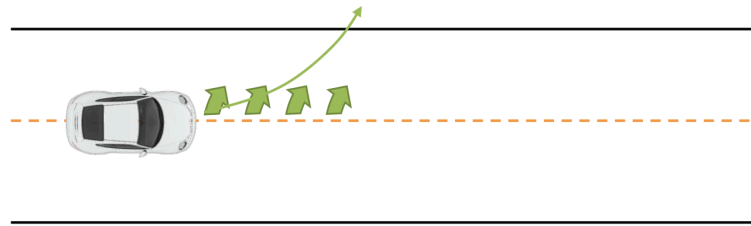
Dynamics model trained on 200 datapoints collected with random policy.



N. Bruchon et al., 2020

Improved algorithm 1.1

How to deal with model errors... long-term accumulation of errors during planning



We “replan”! **And use uncertainty-aware models**

Every N steps

1. run base policy $\pi_0(a_t | s_t)$ (e.g. random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2. learn dynamics $f(s, a)$ to minimise $\sum_i \|f(s_i, a_i) - s'_i\|^2$ **with uncertainty**
3. plan through $f(s, a)$ to choose actions, **take action with high reward in expectation**
4. execute the **first** planned action, observe resulting state s' (MPC)
5. append $\{(s, a, s')_j\}$ to \mathcal{D}

An example: GP-MPC

→ **GP-MPC**...very sample-efficient, shallow model-based RL

- Learn dynamics as GP, optimise for cumulative reward in virtual rollouts

The idea in more detail:

- Learn transition model: $s_{t+1}, r_t = f(s_t, a_t)$ where f is GP
- Define cumulative reward $J = \sum_{k=1}^h r_k$ over horizon h in virtual rollouts
- Find sequence of actions $a_{1:h} = \arg \max J$, making use by defining an acquisition function and derivatives wrt $a_{1:h}$
- → optimise at each iteration; allows to treat time-varying systems, constraints
- See also: [arXiv:1706.06491](https://arxiv.org/abs/1706.06491)

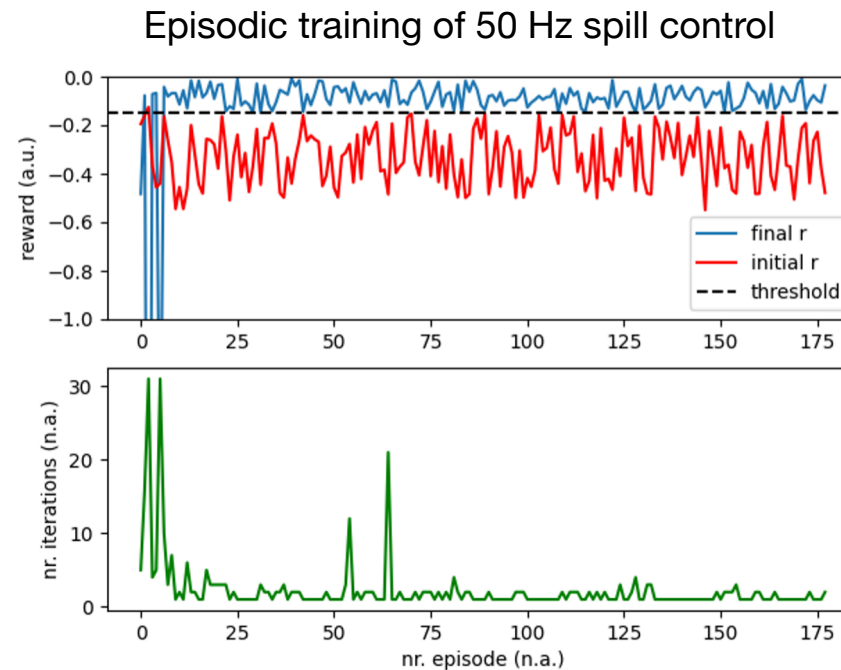
Example: Combine BO with MPC - GP-MPC

Tested idea on 50 Hz control for the SPS slow extracted spill in simulation

Objective: correct norm. amplitude of 50 Hz component A_{spill} such that $A_{spill} < 0.15$

$$\vec{s} = [A_{spill}, \phi_{spill}, A_{corr}, \phi_{corr}], \vec{a} = [\Delta A_{corr}, \Delta \phi_{corr}]$$

$$r = -\sqrt{A_{noise}^2 + A_{corr}^2 + 2A_{noise}A_{corr} \cos \Delta \phi}$$

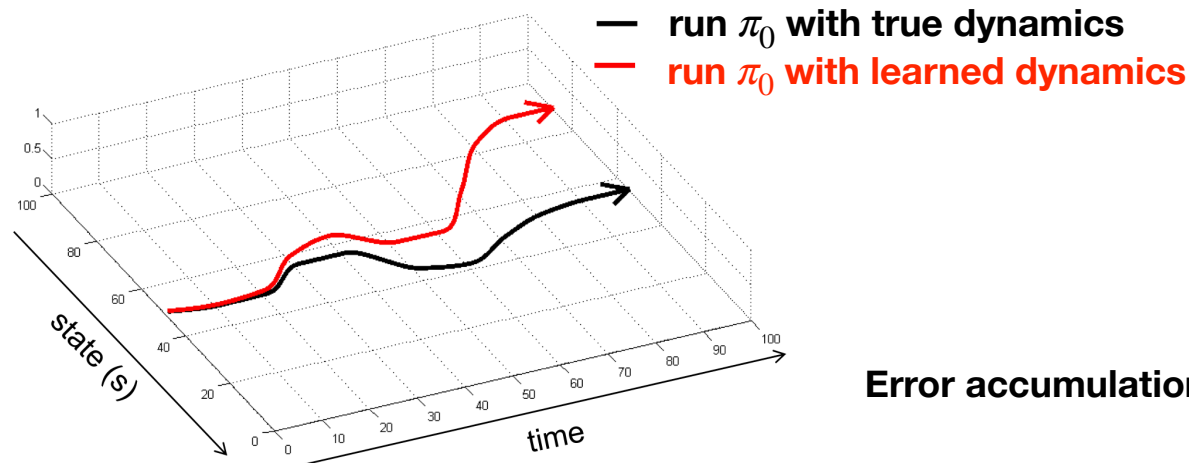


Towards algorithm 2.0

Use “model-free” RL algorithms together with the dynamics model to generate synthetic samples

→ “model-based acceleration” for model-free RL

To avoid issue with error accumulation along long horizons, go **off-policy** and **short rollouts**.



Error accumulation as function of time

Algorithm 2.0 → Dyna-Style Algorithms

Model-based RL with **short** rollouts

1. run base policy $\pi_0(a_t | s_t)$ (e.g. random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$

2. learn dynamics $f(s, a)$ to minimise $\sum_i \|f(s_i, a_i) - s'_i\|^2$

3. pick states s_i from \mathcal{D} use $f(s, a)$ to make **short** rollouts from them

4. use **both** real and model data to improve $\pi_\theta(a | s)$ with **off-policy** RL

5. run $\pi_0(a_t | s_t)$, append the visited $\{(s, a, s')_j\}$ to \mathcal{D}

See: Algorithm Dyna by R. S. Sutton: Integrated architectures for learning, planning and reacting based on approximating dynamic programming. (1990)



General “Dyna-style” model-based RL recipe

1. collect some data, consisting of transitions (s, a, s', r)
2. learn model $\hat{p}(s' | s, a)$ (and optionally, $\hat{r}(s, a)$)
3. repeat K times:
 1. sample $s \sim \mathcal{B}$ from buffer
 2. choose action a (from \mathcal{B} , from π , or random)
 3. simulate $s' \sim \hat{p}(s' | s, a)$ (and $r = \hat{r}(s, a)$) → **only requires short (as few as one step) rollouts from model**
 4. train on (s, a, s', r) with model-free RL
 5. (optional) take N more model-based steps

Our DYNA code (2020)

Use model-free RL agent as provided by stable-baselines3

Simple MLP for dynamics and reward models: wrapped in “surrogate environment”

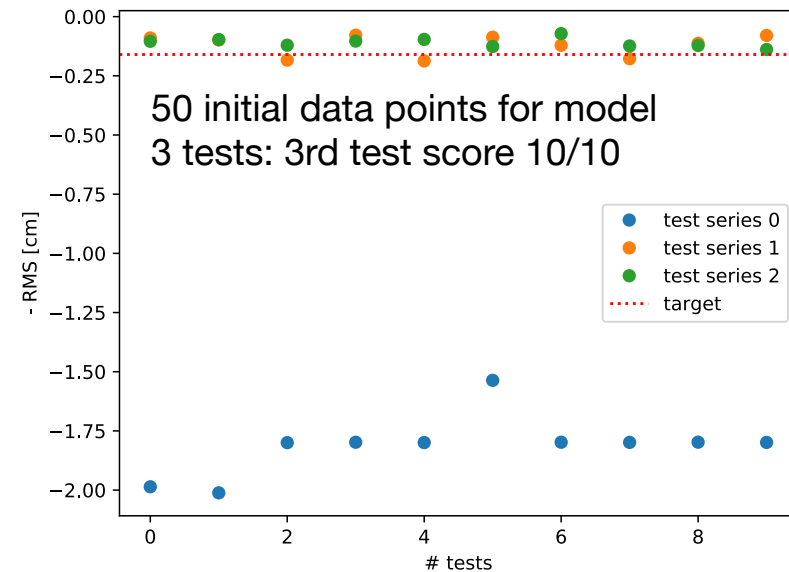
```

env_surrogate = env_surrogate(env_real, s_0);
td3 = TD3(env_surrogate);
fill initial buffer  $\mathcal{D}$  with  $N_{init}$  samples;
env_surrogate.train_model();
for  $N_{dyna}$  do
    td3.learn( $N_{td3}$ );
    test on env_real with td3.predict( $s$ );
    add validation data to  $\mathcal{D}$ ;
    if test OK then
        | break;
    end
    env_surrogate.train_model()
end

```



Example: AWAKE H steering





References

Reinforcement Learning - An Introduction (second edition), R. S. Sutton and Andrew G. Barto, 2018

Deep Reinforcement Learning - CS285 Berkeley University, Sergey Levine, <https://rail.eecs.berkeley.edu/deeprlcourse/>

Trust Region Policy Optimization, J. Schulman et al, arXiv:502.05477

Proximal Policy Optimization Algorithms, J. Schulman et al, arXiv:1707.06347

Addressing Function Approximation Error in Actor-Critic Methods, S. Fujimoto et al, arXiv:1802.09477



EXTRA

tCSC on ML 2024, Split, V. Kain, 13-19 Oct 2024

Towards state-of-the-art Policy Gradient

Use idea of introducing bias:

$$g = \nabla_{\theta} J = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \underbrace{\left(\sum_{t=1}^T r(s_t, a_t) \right)}_{\Psi_t} \right] \approx \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \bar{Q}_{\pi}(s_t, a_t) \right]$$

$$\text{with } A_{\pi}(s_t, a_t) = Q_{\pi}(s_t, a_t) - V_{\pi}(s_t) \approx \mathbb{E}_{\tau \sim p_{\theta}(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \bar{A}_{\pi}(s_t, a_t) \right]$$

Importance sampling...e.g. if collecting data with different policy than target policy.

$$\mathbb{E}_{x \sim p(x)} [f(x)] = \int p(x) f(x) dx = \int \frac{q(x)}{q(x)} p(x) f(x) dx = \int q(x) \frac{p(x)}{q(x)} f(x) dx = \mathbb{E}_{x \sim q(x)} \left[\frac{p(x)}{q(x)} f(x) \right]$$