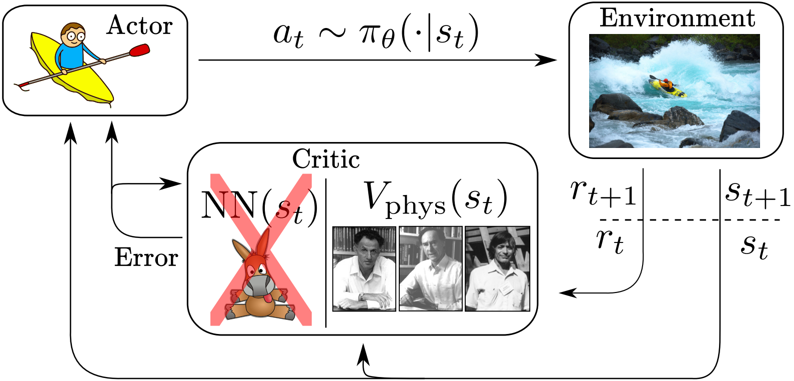
A Brief Intro to Reinforcement Learning

Laurent Pagnier

September, 2024

Destination



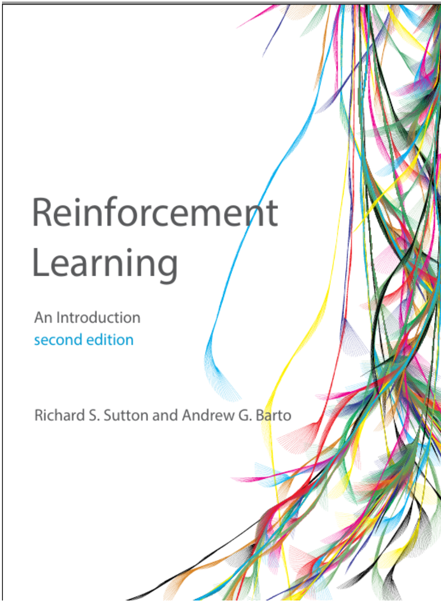
Physics-Informed Critic in an Actor-Critic Reinforcement Learning for Swimming in Turbulence (2024), soon on arXiv.

Disclaimer

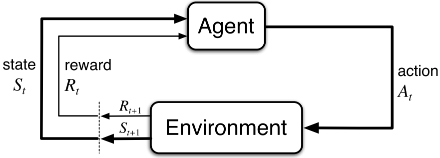
This talk was haphazardly crafted (mostly today).

Reference

Most of the material in this talk is from:



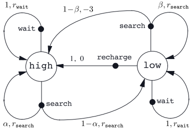
Framework



Markov Decision Process:

$$P(X_{n+1} = x_{n+1} \mid X_n = x_n, \ldots, X_1 = x_1) = P(X_{n+1} = x_{n+1} \mid X_n = x_n) \text{ for all } n \in \mathbb{N}.$$

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	-



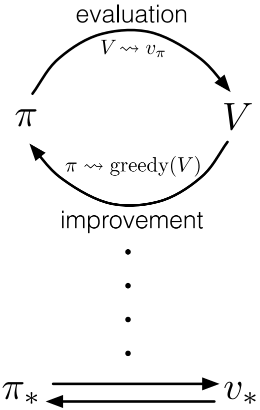
reward hypothesis:

That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)

Dynamic Programming

Bellman optimality equations:

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_*(s') \Big], \text{ or} \\
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a \Big] \\
&= \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma \max_{a'} q_*(s', a') \Big],
\end{aligned}
$$



TD Learning

Idea:

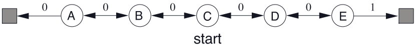
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big],$$

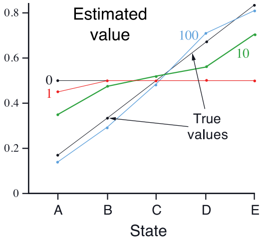
1 step approx:

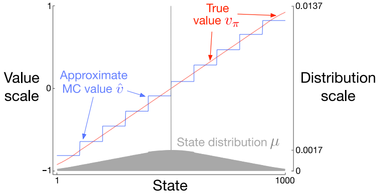
$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

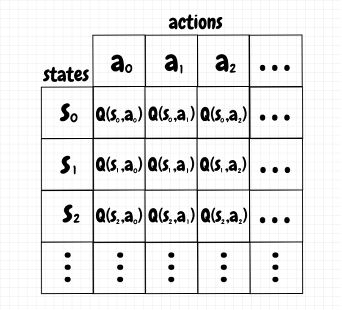
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t \leq T-n,$$

Approximate of Q and V



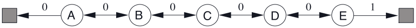


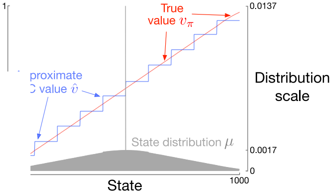


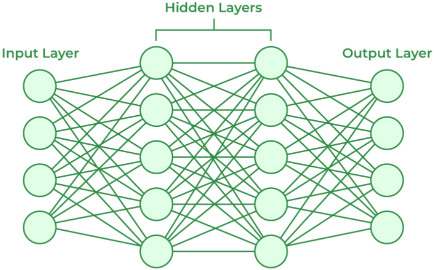


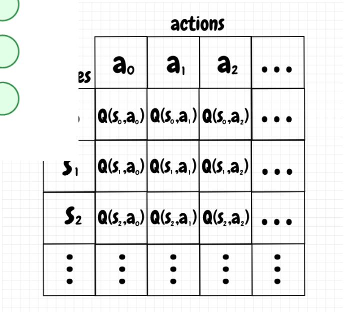
states	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

Approximate of Q and V









Policy Gradient

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathsf{S})$$

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \right],$$

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

Algorithm 1 Vanilla Policy Gradient Algorithm

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

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

    Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

    or via another gradient ascent algorithm like Adam.
    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.
end for

https://spinningup.openai.com/en/latest/algorithms/vpg.html