Computational Control

Reinforcement learning

Saverio Bolognani

Automatic Control Laboratory (IfA)
ETH Zurich

Can we find the optimal policy **without model information**?

Two different settings:

- Based on **collected data** (typically in the form of repeated **episodes**)

$$(x_0, u_0, r_0),\ (x_1, u_1, r_1),\ \ldots (x_T, u_T, r_T)$$

  - repetition of the control task in a numerical simulator
  - repetition of the control task in a controlled environment

- Online during the control task with no prior training ⚠
  → adaptive control
    - extremely challenging **dual** control task (learn AND control)
    - few guarantees
    - an open problem for decades!

Policy iteration and **value iteration** methods require either

- a full (and sufficiently rich) trajectory or
- a model.

Is it possible to learn the system as data is collected along a single trajectory?

Remember the difficult step in policy iteration: for a given policy π, estimate

$$Q^\pi(x,u) = R_x^u + \gamma \sum_{x'} P_{xx'}^u V^\pi(x')$$

where

$$V^\pi(x) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_k | x_0 = x\right]$$

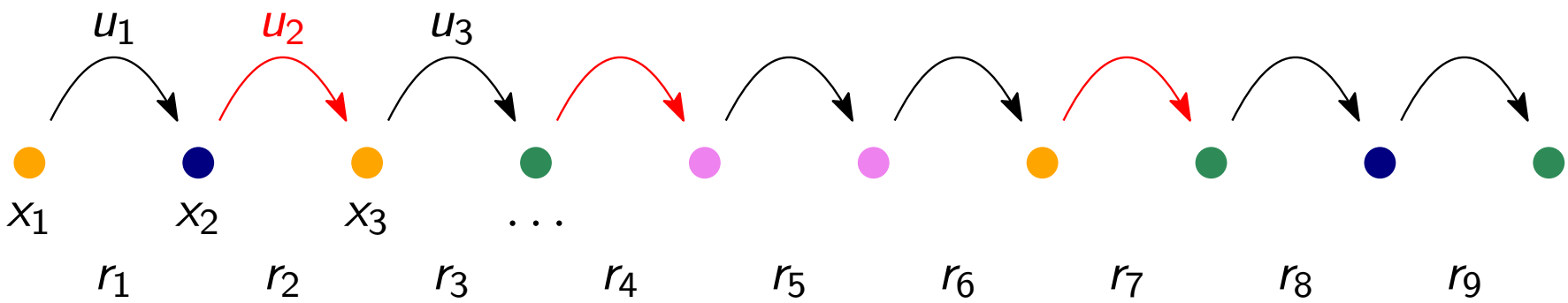
In **Monte carlo** learning we did it by simply evaluating it based on the episode data

$$Q^\pi(x_k,u_k) \approx g_k = \sum_{i=k}^{T} \gamma^{i-k} r_i$$

⚠ known as **Monte Carlo policy evaluation** in the literature

Monte Carlo policy evaluation

Choose the input u (red or black) for an entire episode based on a policy π. Then:



● ► $g_1 = r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$

● ► $g_2 = r_2 + \gamma r_3 + \gamma^2 r_4 + \ldots$

● ► $g_3 = r_3 + \gamma r_4 + \gamma^2 r_5 + \ldots$

$Q(●,►)$ $Q(●,►)$ $Q(●,►)$ $Q(●,►)$ $\ldots$

We havent't used Bellman equation for Q

$$Q(x,u) = R_x^u + \gamma \sum_{x'} P_{xx'}^u \sum_{u'} \pi(x',u')Q(x',u')$$

(although of course the estimate of Q will satisfy that, with infinite data)

Temporal Difference error

Given an individual data point

$$x_k, u_k, \quad x_{k+1}, u_{k+1}, \quad r_k$$

define the TD error as

$$e_k = r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)$$

According to Bellman equation, we have

$$\mathbb{E}[e_k] = 0$$

Stochastic approximation

Suppose that we have a random variable

$$e(q)$$

and we want to find q that solves $\mathbb{E}[e(q)] = 0$.

Stochastic approximation

The iteration

$$q_{k+1} \leftarrow q_k - \alpha_k e(q_k)$$

converges to the solution q^* of $\mathbb{E}[e(q)] = 0$ if

- $e(q)$ is bounded
- $\mathbb{E}[e(q)]$ is non-decreasing in q (and increasing at q^*)
- the sequence α_k satisfies

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

TD-learning (SARSA)

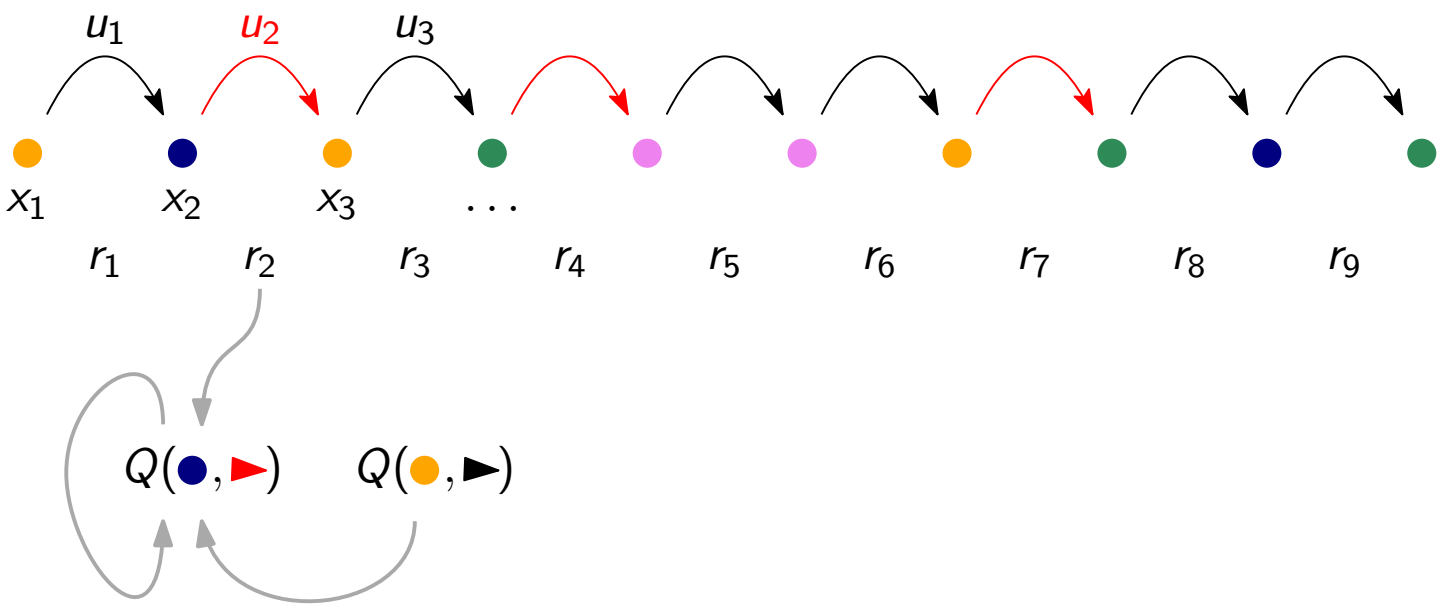
Key idea: Use the empirical evaluations of the TD error

$$e_k = r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)$$

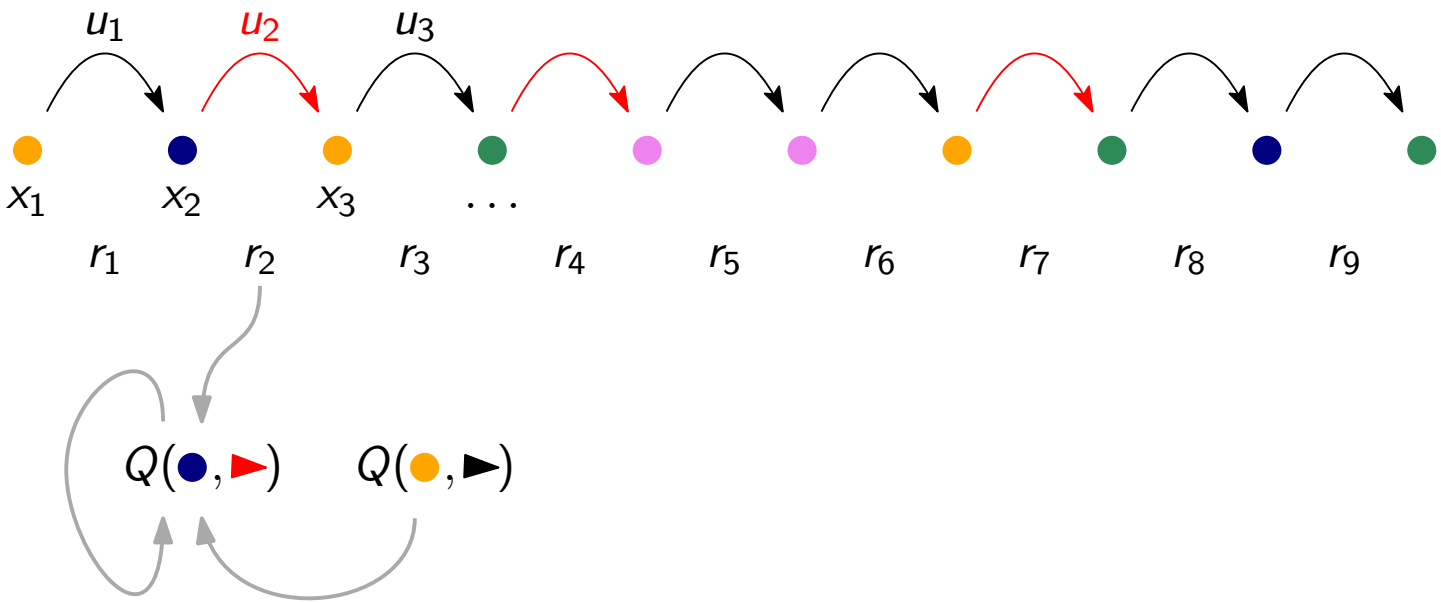
to update the Q function via the (negative) **stochastic approximation** iteration

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \Big(r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k) \Big)$$

Usually with a constant α_k $\Rightarrow$ steady-state non-zero variance



TD-learning (SARSA)



First obvious use: more efficient use of episode data!

We are enforcing the Bellman equation (a consistency principle) instead of assuming that it will asymptotically emerge from the data.

TD-learning (SARSA)

Moreover, instead of alternating between

- complete policy evaluation (full episode)
- policy improvement

we can **interleave** the two operations.

TD-learning (policy iteration)

At every time k

1. Select u_k via an ϵ-greedy policy based on the **latest estimate of** $Q(x_k, u_k)$

$$\pi(x,u) \leftarrow \begin{cases} \text{argmax}_u\, Q(x,u) & \text{with probability } 1-\epsilon \\ \text{Uniform}(\mathcal{U}) & \text{with probability } \epsilon \end{cases}$$

2. Perform an **iterative update** of the Q function

$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha\Big(r_k + \gamma Q(x_{k+1},u_{k+1}) - Q(x_k,u_k)\Big)$$

Meanwhile, allow $\epsilon \to 0$ as $k \to \infty$.

No more need for episodes! (in theory)

Q-learning

Instead of using the temporal difference error to perform policy evaluation (via stochastic approximation), we can directly use it to do a **stochastic approximation** of the **Bellman optimality principle**.

Bellman optimality principle (in Q)

$$Q^*(x,u) = R_x^u + \gamma \sum_{x'} P_{xx'}^u \left(\max_{u'} Q^*(x',u') \right)$$

Individual realization

$$Q^*(x_k,u_k) = r_k + \gamma \max_{u'} Q^*(x_{k+1},u')$$

Notice that the expectation of the individual realization is equal to the Bellman optimality principle.

(Can we do the same with the Bellman optimality principle on the **value function**?)

Q-learning

The **stochastic approximation** update for the Bellman optimality principle is

$$Q^*(x_k, u_k) \leftarrow Q^*(x_k, u_k) + \alpha_k \left(r_k + \gamma \max_u Q^*(x_{k+1}, u) - Q^*(x_k, u_k) \right)$$

Important things to notice:

- The input u_{k+1} is irrelevant (**off-policy algorithm**), it doesn't need to be the optimal one.
- **Convergence** to the optimal Q is guaranteed under
  - non-summability/square-summability assumption on α_k
  - full state-action space exploration
- The policy is typically updated as ϵ**-greedy**.
  In theory, as the learning happens off-policy, the policy could be even just exploration! For performance reason, it is usually a good idea to explore high-reward actions.
- Same **scalability** issue, and same solution: function approximations
  - linear approximants
  - neural networks
- **Forward dynamic programming!**

Q-learning or SARSA

The main difference is the fact that SARSA learns **on policy**, while Q-learning learns **off policy**.

In fact, SARSA does a policy evaluation, and only by improving the policy along the way we eventually converge to the optimal one (of in the vicinity of it)

Q-Learning learns the optimal Q function regardless of the policy that is employed. To obtain the optimal policy, one has to "extract" it by minimizing the optimal Q function in u.

SARSA is more conservative because it learns a policy that has a residual exploration term, and therefore it hedge against the effect of future exploration. Q-learning is very "aggressive" in suggesting the optimal policy assuming optimal future play.

COCO Expert

For this kind of questions and comparison, feel free to ask the COCO Expert at `https://bsaver.io/COCO-Expert` (and think critically about its answers!).

Q-learning as an optimization problem

The Q-learning update can be interpreted as a **stochastic gradient step** for an optimization problem of minimizing the **error of the Bellman optimality principle**.

$$\text{Loss function } L\text{:} \quad \frac{1}{2}\left(R_x^u + \gamma \sum_{x'} P_{xx'}^u \max_{u'} Q^*(x', u') - Q^*(x, u)\right)^2$$

in the decision variable $Q^*(x, u)$ (the current best estimate is used for $Q^*(x', u')$, which is known as **bootstrapping**).

Gradient:

$$\nabla_{Q^*(x,u)} L = -\left(R_x^u + \gamma \sum_{x'} P_{xx'}^u \max_{u'} Q^*(x', u') - Q^*(x, u)\right)$$

Unbiased gradient estimate:

$$-\left(r_k + \gamma \max_{u'} Q^*(x_{k+1}, u') - Q^*(x, u)\right)$$

Q learning is stochastic gradient descent!

Q-learning with linear approximant

Consider the linear parametrization

$$Q_\theta(x,u) = \sum_{\ell=1}^{d} \phi_\ell(x,u)\theta_\ell = \phi^\top(x,u)\theta$$

How do we update the parameter θ in order to satisfy the Bellman optimality principle

$$Q^*(x,u) = R_x^u + \gamma \sum_{x'} P_{xx'}^u \left(\max_{u'} Q^*(x',u') \right)$$

that it

$$\phi^\top(x,u)\theta^* = R_x^u + \gamma \sum_{x'} P_{xx'}^u \left(\max_{u'} \phi^\top(x',u')\theta^* \right)$$

iteratively as samples become available?

$$\phi^\top(x,u)\theta^* = \underbrace{R_x^u + \gamma \sum_{x'} P_{xx'}^u \left(\max_{u'} \phi^\top(x',u')\theta^* \right)}_{Q^+}$$

First, notice that this equation (most likely) **does not have a solution**.

Loss function

$$\min_\theta \underbrace{\frac{1}{2}\left(\phi^\top(x,u)\theta - Q^+\right)^2}_{L(\theta)}$$

A simple iteration that converges to this minimum is the **gradient descent**:

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha \nabla L(\theta_k) \\ &= \theta_k - \alpha \left(\phi^\top(x,u)\theta_k - Q^+\right)\phi(x,u)\end{aligned}$$

Q-learning as stochastic gradient descent

$$\theta_{k+1} = \theta_k - \alpha \left(\phi^\top(x,u)\theta_k - Q^+ \right) \phi(x,u)$$

Similarly to **stochastic approximation**, we can

- replace the real gradient with samples whose expectation is the gradient
- use a non-summable/square-summable step size

Replace

$$Q^+ = R_x^u + \gamma \sum_{x'} P_{xx'}^u \left(\max_{u'} \phi^\top(x',u')\theta^* \right)$$

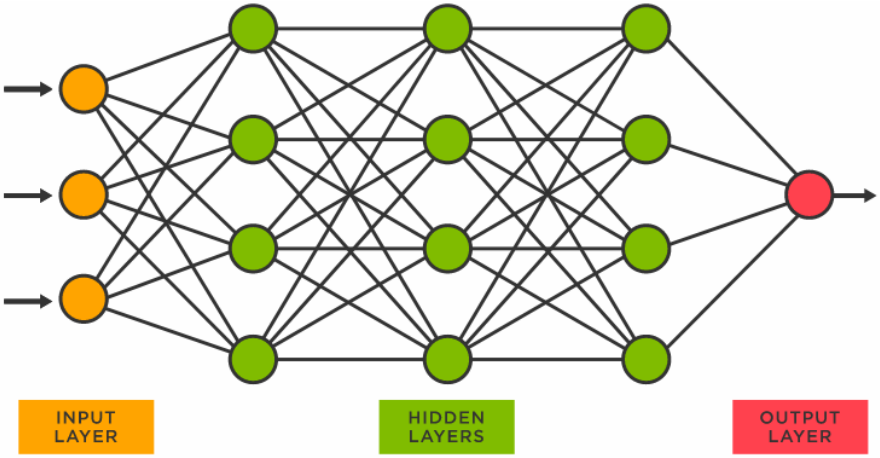
with

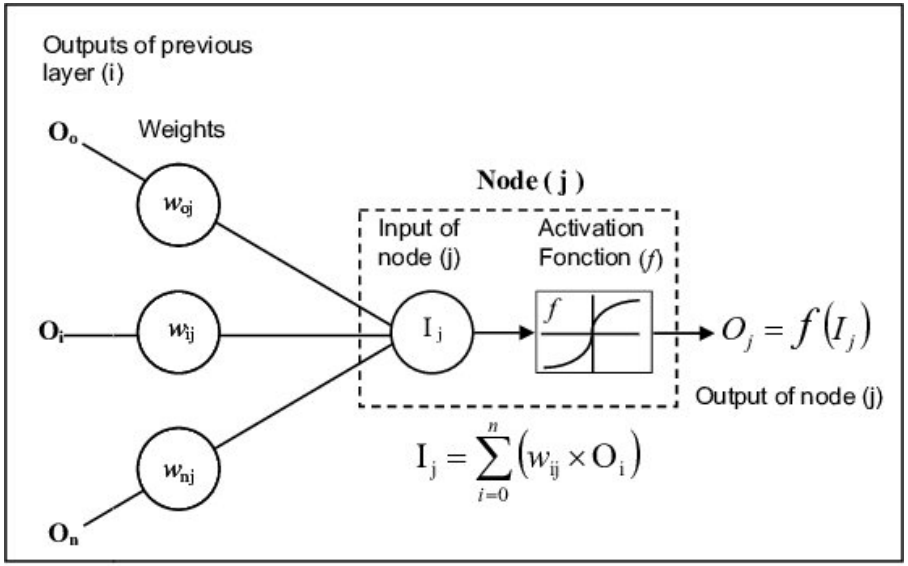
$$r_k + \gamma \max_u \phi^\top(x_{k+1}, u)\theta_k$$

and obtain the iterative update

$$\theta_{k+1} = \theta_k - \alpha_k \left(\phi^\top(x_k,u_k)\theta_k - \left(r_k + \gamma \max_u \phi^\top(x_{k+1},u)\theta_k \right) \right) \phi(x_k,u_k)$$

General approximators





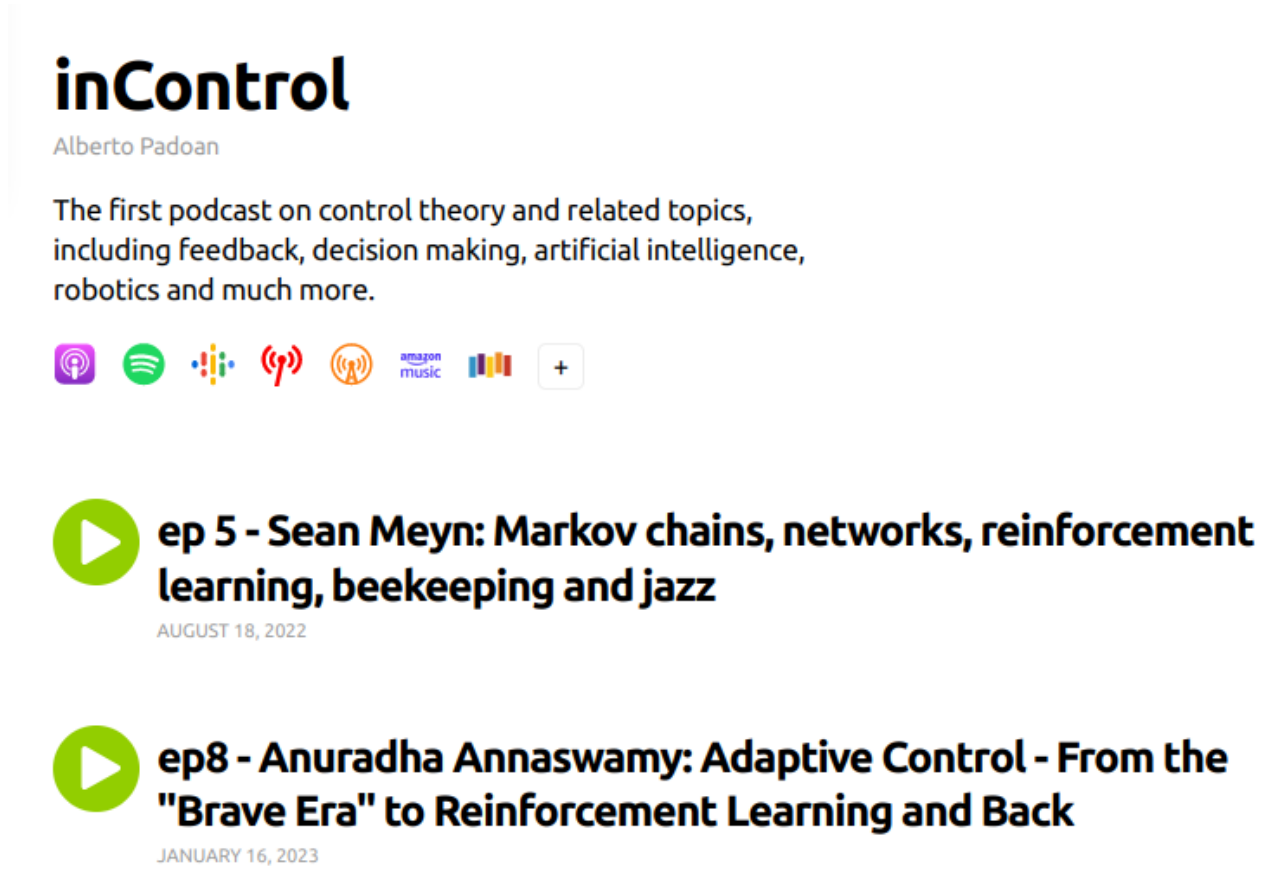
Much more complex approximators can be used (**neural networks**) as long as:

- they provide a **parametrized approximation** $Q_\theta(x, u)$
- they allow to minimize $Q_\theta(x, u)$ over u
- they allow to perform a **stochastic gradient step** in θ to minimize a **loss function** (quality of the approximation)

`https://arxiv.org/abs/1312.5602`
Playing Atari with Deep Reinforcement Learning (2013)

inControl

www.incontrolpodcast.com



Listen to Anu telling us about the challenge of adaptation

Really model free?

Monte Carlo and **Temporal Difference** methods seem to do **without a model of the system**.

Is that really true? **What model information are we assuming**?

It is possible to learn directly the **optimal policy** $\pi(x, u)$ without also learning the value function or Q function (which implies knowledge of a Markovian state x)

Policy gradient methods

Policy gradient

- Consider a trajectory τ of the system

$$\tau = (x_0, u_0, x_1, u_1, \ldots, x_T, u_T)$$

- Assume for simplicity that x_0 is fixed over multiple episodes/experiments.

For a fixed x_0, let $\mathbb{P}(\tau)$ be the **probability of trajectory τ happening**.

Let the associated reward be

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

Parametrized policy

Let $\pi_\theta(x, u)$ be a stochastic policy parametrized in θ

- based on a linear combination of basis functions $\phi^\top \theta$
- based on some parametric form (Gaussian)
- neural network with weights θ

Goal: maximize the expected reward

$$J(\theta) := \mathbb{E}_{\tau \sim \pi_\theta} R(\tau) = \sum_{\tau} \mathbb{P}_\theta(\tau) R(\tau)$$

where $\mathbb{P}_\theta(\tau)$ is the probability that trajectory τ happens when the policy π_θ is used.

Two sources of **complexity**

1. It's an expectation, therefore a sum over all possible trajectories
2. $\mathbb{P}_\theta(\tau)$ is unknown: it depends on both
   - the transition probabilities of the system
   - the policy π_θ

Trajectory probability

$$\mathbb{P}_\theta(\tau) = \Pi_{t=0}^{T} \underbrace{\mathbb{P}(x_{t+1}|x_t, u_t)}_{\text{Transition probabilities}} \underbrace{\pi_\theta(x_t, u_t)}_{\text{Policy}}$$

As we are trying to minimize $J(\theta)$, let's try to derive the gradient $\nabla J(\theta)$.

Proposition

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \Big[\nabla \log \mathbb{P}_\theta(\tau)\, R(\tau) \Big]$$

where $\mathbb{P}_\theta(\tau)$ is the probability of τ when π_θ is used.

Good news! We knew that $J(\theta)$ is an expectation, but also $\nabla J(\theta)$ is an expectation. Do you remember stochastic gradient iterations?

Proof:

$$\begin{aligned}
\nabla J(\theta) = \nabla \big[\mathbb{E}_{\tau \sim \pi_\theta} R(\tau) \big] &= \nabla \left[\sum_\tau P_\theta(\tau) R(\tau) \right] \\
&= \sum_\tau \nabla P_\theta(\tau) R(\tau) \\
&= \sum_\tau \frac{P_\theta(\tau)}{P_\theta(\tau)} \nabla P_\theta(\tau) R(\tau) = \sum_\tau P_\theta(\tau) \nabla \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \Big[\nabla \log \mathbb{P}_\theta(\tau)\, R(\tau) \Big]$$

What is $\nabla \log \mathbb{P}_\theta(\tau)$?

Proposition

$$\nabla \log \mathbb{P}_\theta(\tau) = \sum_{t=0}^{T} \nabla \log \pi_\theta(x_t, u_t)$$

Amazing result!

- no more dependance on the transition probabilities
- depending on the parametrization of π_θ, $\nabla \log \pi_\theta$ is known
- we need a trajectory τ to compute it

Proof:

$$\nabla \log P_\theta(\tau) = \nabla \log \left(\Pi_{t=0}^{T} \mathbb{P}(x_{t+1}|x_t, u_t) \pi_\theta(x_t, u_t) \right)$$

$$= \nabla \left(\sum_{t=0}^{T} \log \mathbb{P}(x_{t+1}|x_t, u_t) + \sum_{t=0}^{T} \log \pi_\theta(u_t, x_y) \right) = \sum_{t=0}^{T} \nabla \log \pi_\theta(u_t, x_t)$$

Putting things together

Instead of computing

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \Big[\nabla \log \mathbb{P}_\theta(\tau)\, R(\tau) \Big]$$

we use a sample

$$\nabla \log \mathbb{P}_\theta(\hat{\tau})\, R(\hat{\tau})$$

where $\hat{\tau}$ comes from the distribution $\tau \sim \pi_\theta$.
How do we sample this distribution?

Policy gradient algorithm

Iteratively, repeat

1. Generate τ via π_θ
2. Compute $R(\tau)$
3. Update $\theta \leftarrow \theta + \alpha\, R(\tau) \sum_{t=0}^{T} \nabla \log \pi_\theta(u_t, x_t)$

Why not learn all the time?

Consider a simple system

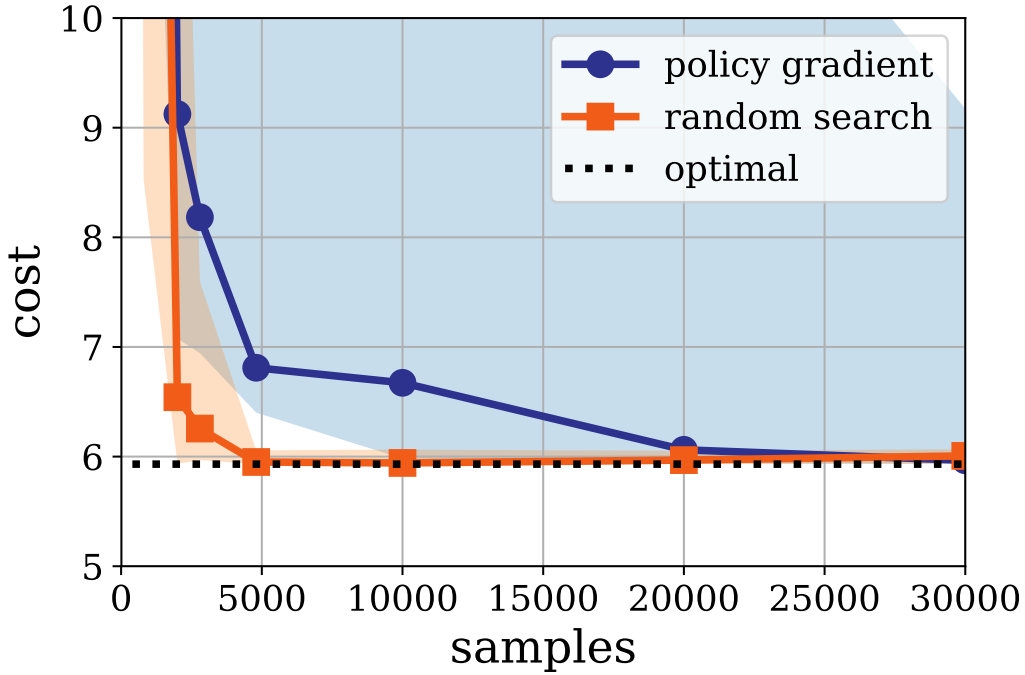
$$x_{t+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t$$

and an LQR-type cost with

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = r$$

Consider the alternatives

- Q-learning (what is the Q function in an LQR problem?)
  - prior information: Markovian state
  - good parametrization of the Q function
- policy gradient



A Tour of Reinforcement Learning: The View from Continuous Control

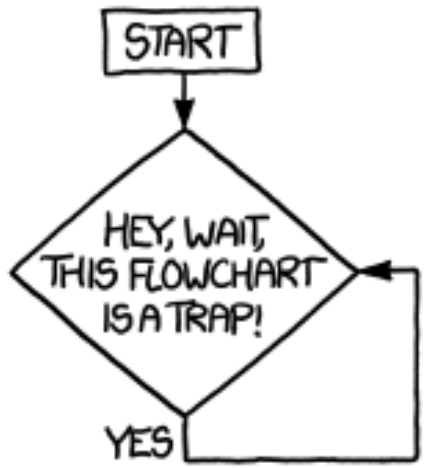
Annual Review of Control, Robotics, and Autonomous Systems
Vol. 2:253-279 (Volume publication date May 2019)
First published as a Review in Advance on December 14, 2018
https://doi.org/10.1146/annurev-control-053018-023825

Benjamin Recht
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720, USA; email: brecht@berkeley.edu

Final tasks

1. **Update your flowchart**

The control engineer flowchart



2. **Fill the course evaluation survey**



This work is licensed under a
Creative Commons Attribution-NonCommercial 4.0 International License

https://bsaver.io/COCO