NTNU – Trondheim
Norwegian University of
Science and Technology

SUBPRO
SUBSEA PRODUCTION AND PROCESSING

Reinforcement Learning in Optimal Control

Dinesh Krishnamoorthy

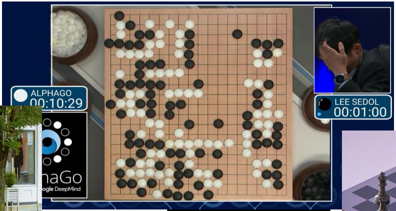
Department of Chemical Engineering
Norwegian University of Science and Technology (NTNU)
dinesh.krishnamoorthy@ntnu.no

07 November 2019

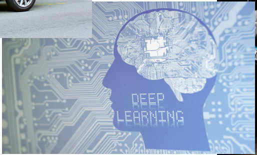
"We consider all of the work in optimal control also to be, in a sense, work in RL"

-Sutton & Barto (2018)

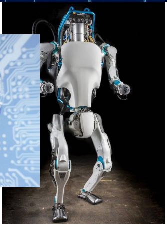
Everyone talks about it...

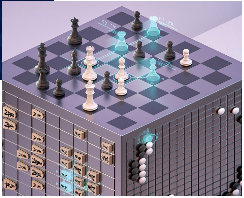


Source: Uber



Source: Forbes


Source: Boston Dynamics


Source: DeepMind

... but what exactly is it?

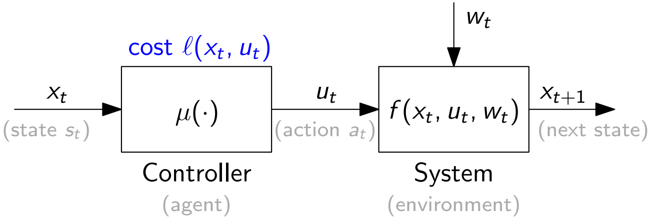
Source: jungle.princeton.edu

Aim of the talk

Provide a basic understanding of RL in optimal control

1. **What** is Optimal Decision-making? - Dynamic Programming
2. **Where** does Machine Learning come into the picture ?
3. **Why** do we need them?
4. **How** to use them?

Introduction - Decision making



Objective

Take suitable actions u_t, based on the current state x_t, to control a dynamic (stochastic) system, such that the overall cost is minimized.

These kind of problems are studied under the context of Dynamic Programming (DP).

Dynamic programming

Dynamic programming (DP) is a mathematical framework for solving multistage decision-making problems

.

Richard E. Bellman
(1920-1984)

Any optimization (deterministic/stochastic, discrete/continuous variables etc.) that involves a sequence of decisions fits the framework

- Operations (Inventory management, routing,...)
- Control (process control, robotics, path planning,....)
- Finance (portfolio management ....)
- Manufacturing (planning, scheduling,...)
- Games (Chess, Go, ...)

Notations

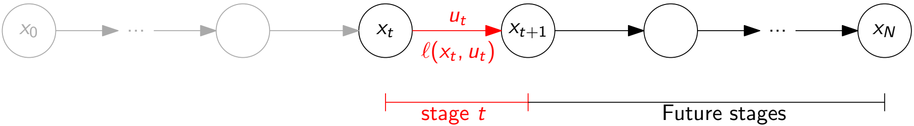
Control community	AI community	
x_k	S_t	state
x_{k+1}	S_{t+1}	successor state
u_k	A_t	action
$\ell(x_k, u_k, w_k)$	R_{t+1}	transition cost/Reward
$J_k(x_k)$	$v(S_t)$	cost of being in a state
$Q_k(x_k, u_k)$	$Q(S_t, A_t)$	Q-factor
$\mathbb{E}\{\sum_{k=0}^{N-1} \ell(x_k, u_k, w_k)\}$	$G_t = R_{t+1} + \cdots + R_N$	Return/Cost-to-go
$\min(\cdot)$	$\max -(\cdot)$	Objective
$\mu(x)$	$\pi(a\|s)$	policy/control law
$\mathbb{E}\{\ell(\cdot) + J_{k+1}(x_{k+1})\|x_k, u_k\}$	$\sum p(s', r\|s, a)[R_{t+1} + V(s')]$	

I will predominantly use the notation from Bertsekas (2019)

Sometimes I also include the notation from Sutton & Barto (2018) in gray!

Introduction - Dynamic programming

Finite Horizon Deterministic Problem



Decision-making: What do I need to take into account?

$\min_u$ (cost now + future costs)

Cost function:

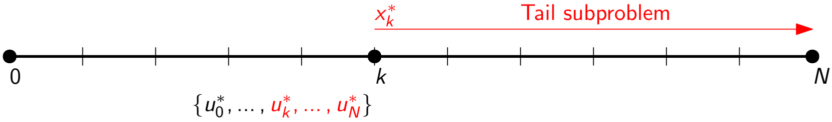
$$J(x_t) := \ell_t(x_t, u_t) + \sum_{k=t+1}^{N-1} \ell_k(x_k, u_k) + \ell_N(x_N)$$

Optimal cost function

$$J^*(x_t) = \min_{u_k \in U_k(x_k)} \ell_t(x_t, u_t) + \sum_{k=t+1}^{N-1} \ell_k(x_k, u_k) + \ell_N(x_N)$$

Solve using Dynamic Programming (DP)

Principle of optimality



- Let $\{u_0^*, \ldots u_{N-1}^*\}$ denote the optimal control sequence, with the corresponding optimal state sequence, $\{x_1^*, \ldots x_N^*\}$.
- Consider the tail subproblem at time k, starting at x_k^*, and minimizes over $\{u_k, \ldots u_{N-1}\}$, the "cost-to-go" from k to N

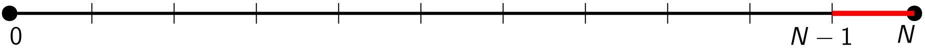
$$\ell_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} \ell_m(x_m, u_m) + \ell_N(x_N)$$

- Then the tail optimal control sequence $\{u_k^*, \ldots u_{N-1}^*\}$ is optimal for the tail subproblem.

Principle of Optimality - Every optimal policy consists only of optimal sub policies.

Exact Dynamic Programming (DP)

Idea of Exact DP - Make optimal decision in stages



DP recursion - Produces the optimal costs $J_k^*(x_k)$ of the x_k-tail subproblems

Start with

$$J_N^*(x_N) = \ell_N(x_N), \qquad \text{for all } x_N$$

and for all $k = N-1, N-2, \ldots, 0$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \underbrace{\ell_k(x_k, u_k)}_{\text{cost now}} + \underbrace{J_{k+1}^*(f(x_k, u_k))}_{\text{cost-to-go}}, \qquad \text{for all } x_N$$

Then the optimal cost $J_0^*(x_0)$ is obtained at the last step

Exact DP algorithm

Go backwards to compute the optimal costs $J_k^*(x_k)$

Start with

$$J_N^*(x_N) = \ell_N(x_N), \qquad \text{for all } x_N$$

and for all $k = N-1, N-2, \dots, 0$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \underbrace{\ell_k(x_k, u_k)}_{\text{cost now}} + \underbrace{J_{k+1}^*(f(x_k, u_k))}_{\text{opt. cost-to-go}}, \qquad \text{for all } x_N$$

Then the optimal cost $J^*(x_0)$ is obtained at the last step

Go forwards to construct the optimal sequence $\{u_0^*, \dots u_{N-1}^*\}$

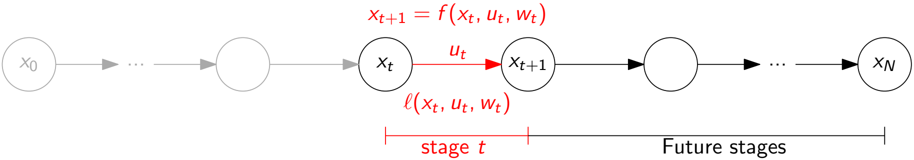
Given x_0, start with

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \left[\ell_0(x_0, u_0) + J_1^*(f_0(x_0, u_0))\right], \qquad x_1^* = f_0(x_0, u_0^*).$$

Going forward sequentially $k = 1, 2, \dots, N-1$, we get

$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} \left[\ell_k(x_k^*, u_k) + J_{k+1}^*(f_k(x_k^*, u_k))\right], \qquad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Finite Horizon Stochastic Problem



Decision-making: What do I need to take into account?

$$\min \mathbb{E}\{\text{cost now } + \gamma \text{future cost}\}$$

Policies $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, sequence of control law that specifies what u_k to apply, when at x_k, i.e. $u_k = \mu_k(x_k)$. The cost is then,

$$J_\pi(x_t) := \mathbb{E}\left\{\ell_t(x_t, \mu_t(x_t), w_t) + \sum_{k=t+1}^{N-1} \gamma^{k-t}\ell_k(x_k, \mu_k(x_k)) + \gamma^{N-t}\ell_N(x_N)\right\}$$

Optimal cost

$$J^*(x_t) := \min_\pi J_\pi(x_t)$$

$0 < \gamma \leq 1$ - Discount factor

Exact DP solution

Go backwards to compute the optimal costs $J_k^*(x_k)$

Start with $J_N^*(x_N) = \ell_N(x_N)$, for all x_N and for all $k = N-1, N-2, \ldots, 0$

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \mathbb{E}\left\{\ell_k(x_k, u_k, w_k) + J_{k+1}^*(f(x_k, u_k))\right\}, \qquad \text{for all } x_N$$

Then the optimal cost $J^*(x_0)$ is obtained at the last step, and the optimal control law μ_k^* is constructed alongside J_k^*

Go forwards to construct the optimal sequence $\{u_0^*, \ldots u_{N-1}^*\}$

Going forward sequentially $k = 0, \ldots, N-1$, observe x_k and apply

$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} \left[\mathbb{E}\left\{\ell_k(x_k^*, u_k, w_k) + J_{k+1}^*(f_k(x_k^*, u_k))\right\}\right], \qquad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Bellman Equation

$$J^*(x_k) = \min_{u \in U_k(x_k)} \mathbb{E}\left\{\ell(x_k, u_k, w_k) + J^*(x_{t+1}) | x_t, u_k\right\}$$

$$v^*(s) = \max_a \mathbb{E}\{R_{t+1} + v^*(S_{t+1}) | S_t = s, A_t = a\}$$

Exact DP - Illustrative Example

- Consider a system

$$x_{k+1} = x_k + u_k - w_k$$

with $x_k \in \mathbb{R}$, $u_k \in \mathbb{R}$ and $w_k \sim \mathcal{N}(0, \sigma^2)$

- OCP:

$$\min \sum_{k=0}^{N-1} (x_k^2 + u_k^2) + x_N^2$$

with $N = 3$

Rawlings & Mayne (2009)

Exact DP - Illustrative Example

- Consider a system $x_{k+1} = x_k + u_k - w_k$ with $x_k \in \mathbb{R}$, $u_k \in \mathbb{R}$ and $w_k \sim \mathcal{N}(0, \sigma^2)$
- OCP: $\min \sum_{k=0}^{N-1} (x_k^2 + u_k^2) + x_N^2$, with $N = 3$

$$J_k^*(x_k) = min_{u_k} \mathbb{E}\{\ell(x_k, u_k, w_k) + J_{k+1}^*(x_{k+1})\}$$

at stage $k = 2$

$$\Rightarrow u_2^* = -0.5x_2, \qquad J_2^*(x_2) = 1.5x_2^2$$

at stage $k = 1$

$$\Rightarrow u_1^* = -3/5x_1, \qquad J_1^*(x_1) = 8/5x_1^2$$

at stage $k = 0$

$$\Rightarrow u_0^* = -8/13x_0, \qquad J_0^*(x_0) = 21/26x_0^2$$

Go backward to compute the optimal cost, go forward to construct the optimal sequence

---

Rawlings & Mayne (2009)

Issues with Exact DP

- For linear quadratic (LQ) problems optimal control policy: $u_k = -K_k(x_k)$
- As $N \to \infty$, $K = (R + B^T PB)^{-1} B^T PA$
- P is a solution to the Discrete Algebraic Riccati Equation.

But in general, it is difficult to provide such closed-form representations

Curse of dimensionality

$$\min_{u_k} \mathbb{E}\{\ell(x_k, u_k, w_k) + J^*_{k+1}(x_{k+1})\}$$

- Need to compute (and store) $J^*_{k+1}(x_{k+1})$
- compute expectation for each u_k
- minimize over all u_k !

Intractable and high dimensional? $\to$ Approximate !

Bertsekas (2019)

Recht, *Annual Review of Control, Robotics, and Autonomous Systems* (2019)

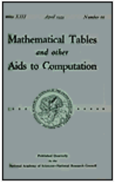
Function Approximations

Polynomial Approximation—A New Computational Technique in Dynamic Programming: Allocation Processes

By Richard Bellman, Robert Kalaba, and Bella Kotkin

Functional Approximations and Dynamic Programming

Richard Bellman and Stuart Dreyfus



Mathematical Tables and Other Aids to Computation
Vol. 13, No. 68 (Oct., 1959), pp. 247-251 (5 pages)

Published by: American Mathematical Society

Approximate a complicated/unknown function $f(\cdot)$ with something simpler!

- Assume access to noisy values of $\beta^s := f(x^s)$, $s = 1, 2, \dots$
- Introduce a parametric architechture - a desirable functional form $\tilde{f}(x, \theta)$
- find $\hat{\theta}$ such that $\tilde{f}(x, \theta) \approx f(x)$, for all (or most) x

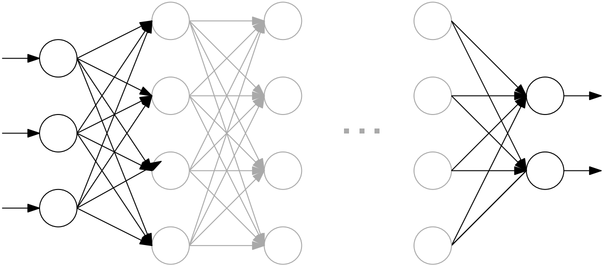
$$\hat{\theta} = \arg\min_{\theta} \sum_{s} \tilde{f}(x^s, \theta) - \beta^s$$

Supervised Learning

- Linear feature-based architecture $\tilde{f}(x, \theta) = \theta^T \phi(x)$
- Nonlinear architecture, e.g. Neural networks[a]

[a] Neuro DP, Bertsekas & Tsitsiklis (1996)

Neural networks



- Universal function approximators (with a sufficiently rich parameterization)
- Deep NN a key factor in recent RL success stories
- First several layers extract features, and last layers engage in correlating the features

Silver et al, *Nature* (2016)
Shin et al, *Comput. & Chem. Eng* (2019)

Reinforcement learning - High level mind map

1. **Value space approximation**
   - Approximate optimal cost-to-go $\tilde{J}_{k+1}(x_{k+1})$
   - Alternatively, approximate the "Q-function" $\tilde{Q}_{k+1}(x_{k+1}, u_{k+1})$
2. **Policy space approximation**
   - optimal policy is complicated
   - Use a parametric form for the policies $\tilde{\mu}(x, \theta)$
3. **Actor-critic**
   - Given $\tilde{\mu}$, learn $\tilde{J}$
   - Using $\tilde{J}$, improve $\tilde{\mu}$

Approximate DP	$\Leftrightarrow$	Reinforcement learning	$\Leftrightarrow$	Neuro DP
Powell (2007)		Barto & Sutton (1998)		Bertsekas & Tsitsikilis (1996)

Approximation in value space (one-step look ahead)

Consider the DP problem

Approximate min

Discretization

Now — cost-to-go

$$\min_{u_k} \mathbb{E}\left\{\ell(x_k, u_k, w_k) + \tilde{J}_{k+1}(x_{k+1})\right\}$$

Approximate $\mathbb{E}\{\cdot\}$

Certainty equivalence

Monte Carlo tree search

⋮

Approximate cost-to-go

Parametric approximations

Rollout / MPC

Aggregation

⋮

Use $\tilde{J}_{k+1}$ instead of J^*_{k+1} and one-step look ahead minimization to construct a suboptimal control law $\tilde{\mu}_k$

Bertsekas (2019)

Sequential DP approximation - Fitted Value Iteration

How do we use NN in finite horizon DP?

Start with $\tilde{J}_N = \ell_N$ and sequentially train going backwards until $k = 0$

- Using the cost-to-go approximation from the preceding stage $\tilde{J}_{k+1}(x_{k+1}, \theta_k)$, and one-step look ahead,
- Construct a large number of sample state-cost pairs (x_k^s, β_k^s), $s = 1, 2, \cdots, M$

$$\beta_k^s = \min_{u \in U_k(x_k)} \mathbb{E}\left\{\ell_k(x_k^s, u, w_k) + \tilde{J}_{k+1}(f(x_k^s, u, w_k), \theta_{k+1})\right\} \qquad s = 1, \dots, M$$

- Train a parametric architecture $\tilde{J}_k(x_k, \theta_k)$ on the training set (x_k^s, β_k^s), $s = 1, 2, \cdots, M$

$$\hat{\theta} = \arg\min_{\theta_k} \sum_{s=1}^{M} (\tilde{J}_k(x_k^s, \theta_k) - \beta_k^s)$$

- One neural network at each stage !
- Generate data using the NN trained at the preceding stage (NB! Bias)

But requires a lot of computation! $\rightarrow$ Use Q-factors!

Bertsekas (2019)

Q-factors

Cost functions of state-action pairs

Optimal Q-factors are given by,

$$Q_k^*(x_k, u_k) = \mathbb{E}\left\{\ell_k(x_k, u_k, w_k) + J_{k+1}^*(x_{k+1})\right\}$$

which defines the optimal policy and cost-to-go functions as

$$\mu_k^*(x_k) \in \arg\min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k), \qquad J_k^*(x_k) = \min_{u_k \in U_k(x_k)} Q_k^*(x_k, u_k)$$

DP algorithm for Q-factors

$$Q_k^*(x_k, u_k) = \mathbb{E}\left\{\ell_k(x_k, u_k, w_k) + \min_{u_{k+1}} Q_{k+1}^*(f(x_k, u_k, w_k), u_{k+1})\right\}$$

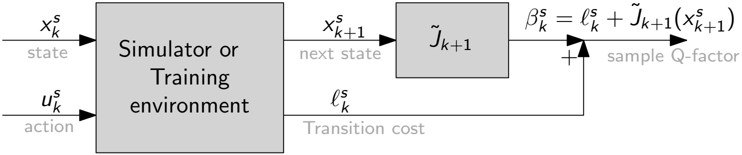
- NB! Order of $\mathbb{E}\{\cdot\}$ and min has been reversed!
- R.H.S can be approximated by sampling and simulation

Approximate optimal Q-factors $Q_k^*(x_k, u_k)$ with $\tilde{Q}_k(x_k, u_k)$

Sequential Q-factor approximation

$$\tilde{Q}_k^*(x_k, u_k) = \mathbb{E}\left\{\ell_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(x_{k+1})\right\}$$

Assuming $\tilde{J}_{k+1}$ is available, how to compute the Q-factors "model-free"?



- Train a parametric architecture $\tilde{Q}_k(x_k, u_k, \theta_k)$ on the training set $((x_k^s, u_k^s), \beta_k^s)$, $s = 1, 2, \cdots, M$

$$\hat{\theta} = \arg\min_{\theta_k} \sum_{s=1}^{M} (\tilde{Q}_k(x_k^s, u_k^s, \theta_k) - \beta_k^s)$$

After tuning θ_k, the one-step lookahead control can be obtained online as

$$\tilde{\mu}_k(x_k) \in \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u_k, \hat{\theta}_k)$$

....all this is done model-free

Bertsekas (2019)

Q-learning

On policy (SARSA)

$$\tilde{Q}_k(x_k, u_k) = \ell_k(x_k, u_k, w_k) + \tilde{Q}_{k+1}(x_{k+1}, u_{k+1})$$

u_k and u_{k+1} derived from the same policy (on-policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Off policy (Q-learning)

$$\tilde{Q}_k(x_k, u_k) = \ell_k(x_k, u_k, w_k) + \min_{u_{k+1}} \tilde{Q}_{k+1}(x_{k+1}, u_{k+1})$$

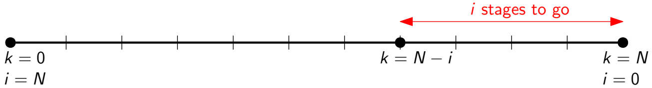
u_k derived from the current policy , but u_{k+1} is from a different policy (off-policy)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- α - learning rate for incremental learning

Sutton & Barto (2018)

Value Iteration - Infinite horizon



Fix horizon N and let terminal cost $= 0$

At $k = N - i$, we have i stages to-go and

$$J_{N-i}(x) = \min_{u \in U(x)} \mathbb{E}\left\{\ell(x, u, w) + \gamma J_{N-i+1}(f(x, u, w))\right\}$$

Reverse the time index and define $V_i(x) = J_{N-i}(x)$

$$V_i(x) = \min_{u \in U(x)} \mathbb{E}\left\{\ell(x, u, w) + \gamma V_{i-1}(f(x, u, w))\right\}$$

$$v_{k+1}(s) = \max_a \mathbb{E}\left\{R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right\}$$

VI algorithm : Start at some V_0 and iterate until convergence!

Convergence of VI

$$V^*(x) = \lim_{N \to \infty} V_N(x)$$ - under some conditions, see Bertsekas (2019)

Exact Policy Iteration (PI)

1. Policy Evaluation

- Evaluate the cost for a given policy $\mu(x)$

$$V_i(x) = \mathbb{E}\left\{\ell(x, \mu(x), w) + \gamma V_{i-1}(f(x, \mu(x), w))\right\}$$

$$v_{k+1}(s) = \mathbb{E}\left\{R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = \pi(s)\right\}$$

2. Policy Improvement

$$\mu(x) = \arg \min_{u \in U(x)} \mathbb{E}\left\{\ell(x, u, w) + \gamma V_{i-1}(f(x, \mu(x), w))\right\}$$

$$v_{k+1}(s) = \arg\max_a \mathbb{E}\left\{R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right\}$$

$$\mu \xrightarrow{E} V \xrightarrow{I} \mu' \xrightarrow{E} V' \xrightarrow{I} \mu'' \xrightarrow{E} \cdots \xrightarrow{I} \mu^* \xrightarrow{E} V^*$$

Monotonically decreasing (Policy improvement theorem)

Sutton & Barto (2018)

Approximate Policy Iteration (API)

$$\mu \xrightarrow{\text{E}} \tilde{V} \xrightarrow{\text{I}} \mu' \xrightarrow{\text{E}} \tilde{V}' \xrightarrow{\text{I}} \mu'' \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \mu^* \xrightarrow{\text{E}} \tilde{V}^*$$

- Run the policy for different initial states x^s for some number of stages
- Accumulate the corresponding discounted cost β^s
- Train a parametric architecture $\tilde{V}(\mu(x^s), \theta)$ using state-cost pairs (x^s, β^s)
- Policy improvement

$$\mu'(x) = \arg\min_{u \in U(x)} \mathbb{E}\left\{\ell(x, u, w) + \gamma \tilde{V}_{i-1}(f(x, \mu(x), w), \theta)\right\}$$

Bertsekas (2019)

Approximation in policy space

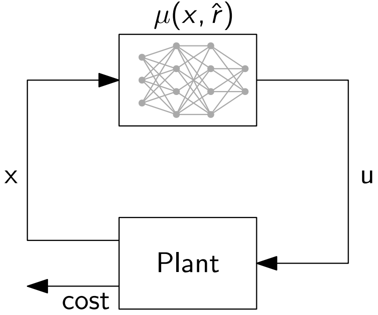


Figure:

- Parameterize the control law $u = \tilde{\mu}(x, r)$
- tune the parameters r to approximate the optimal policy
- E.g. PID controller with three control parameters !
- Also similar to extremum seeking control
- Expert supervisory learning (Surrogate Optimizer)

Bertekas (2018)

Policy gradient

Direct policy search

$$\min_z F(z)$$

$F(z) := \sum_t \gamma \ell(x_t, u_t)$
$z = (x_1, u_1, x_2, u_2, \dots)$

Express as an approx. stochastic optimization problem

$$\min_r \mathbb{E}_{p(z,r)}\{F(z)\}$$

$$r^{i+1} = r^i - \alpha \nabla \left(\mathbb{E}_{p(z,r^i)}\{F(z)\} \right)$$

log-likelihood trick

$$\nabla \left(\sum p(z, r^i) F(z) \right)$$
$$= \sum \nabla p(z, r^i) F(z)$$
$$= \sum p(z, r^i) \frac{\nabla p(z, r^i)}{p(z, r^i)} F(z)$$
$$= \sum p(z, r^i) \nabla \log_e p(z, r^i) F(z)$$

$\Rightarrow \mathbb{E}_{p(z,r)} \left\{ \nabla \log_e p(z, r^i) F(z) \right\}$

Policy gradient algorithm

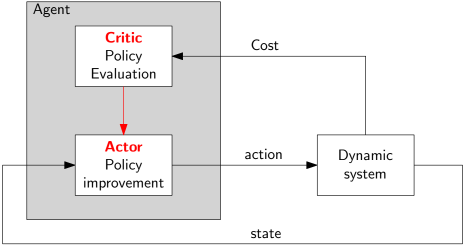
- At r^i obtain a sample z^i according to the distribution $p(z, r^i)$
- Compute $\mathbb{E}_{p(z,r^i)} \left\{ \nabla \log_e p(z^i, r^i) F(z^i) \right\}$
- Iterate $r^{i+1} = r^i - \alpha \nabla \left(\mathbb{E}_{p(z^i,r^i)}\{F(z^i)\} \right)$

Note! There are also other (gradient-free) random search approaches, e.g. cross entropy.

Bertsekas (2019)

Actor-critic

Approximation in value space and approximation in policy space in PI



Actor-critic

Critic

- Learn the approximate policy evaluation $\tilde{J}$

Actor

- Given $\tilde{J}$, improve the approximate policy $\tilde{\mu}$

Some thoughts

Training Environment

- Need a high-fidelity simulator - Usually not a problem for games...
- Robotics and autonomous driving - laboratory training ($$)...
- What about process & manufacturing industries !?

Training

- Tolerate failure - Learn from mistakes!
- Are we ready to trust it?

PSE - Where in the decision-making hierarchy?

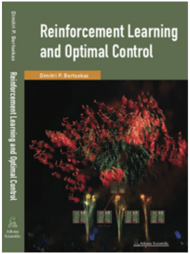
- Does not make sense to replace PID control.
- MPC (a specific case of Rollout approximation!)
- Perhaps more useful in planning & scheduling - integrated decision-making?
- Can assist with optimal tuning? - need to tune hyper-parameters instead !

Look ahead...

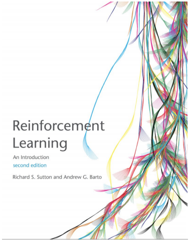
- At present, RL is an art.
- DeepMind, Google Brain, facebook, Uber, ...
- Is academic research following/competing with them?

Thank you !

References



Bertsekas (2019)

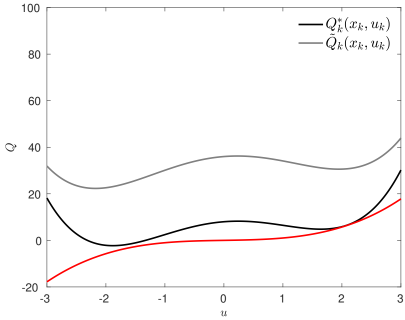


Sutton & Barto (2018)

- Shin, J., Badgwell, T.A., Liu, K.H. and Lee, J.H., 2019. Reinforcement Learning–Overview of recent progress and implications for process control. Computers & Chemical Engineering, 127, pp.282-294.
- Lee, J.H., Shin, J. and Realff, M.J., 2018. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Computers & Chemical Engineering, 114, pp.111-121.
- Recht, B., 2019. A tour of reinforcement learning: The view from continuous control. Annual Review of Control, Robotics, and Autonomous Systems, 2, pp.253-279.
- Bertsekas, D. 1976. Dynamic Programming and Stochastic Control, Academic Press Inc.
- Rawlings, J., Mayne, D. and Diehl, M. 2017. Model Predictive Control: Theory, Computation, and Design, 2nd Edition, Nob Hill Publishing.
- jungle.princeton.edu

What is a “good” approximation?

Good approximation



Poor approximation