F.L. Lewis
Moncrief-O'Donnell Endowed Chair
Head, Controls & Sensors Group

Supported by :
NSF - Paul Werbos
ARO- Sam Stanton
AFOSR- Fariba Fahroo

Automation & Robotics Research Institute (ARRI)
The University of Texas at Arlington

Adaptive Dynamic Programming (ADP) For Feedback Control Systems

The University of Texas ARLINGTON

Talk available online at
http://ARRI.uta.edu/acs

ARRI

It is man's obligation to explore the most difficult questions in the clearest possible way and use reason and intellect to arrive at the best answer.

Man's task is to understand patterns in nature and society.

The first task is to understand the individual problem, then to analyze symptoms and causes, and only then to design treatment and controls.

Ibn Sina 1002-1042
(Avicenna)

Importance of Feedback Control

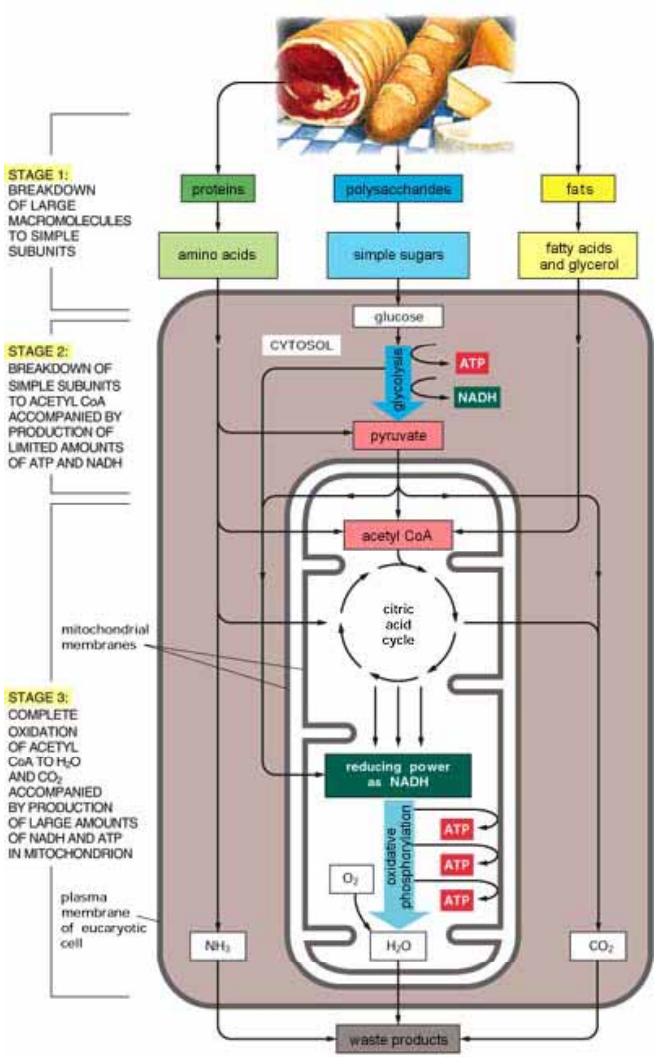
Darwin- FB and natural selection
Volterra- FB and fish population balance
Adam Smith- FB and international economy
James Watt- FB and the steam engine
FB and cell homeostasis

The resources available to most species for their survival are meager and limited

Nature uses Optimal control

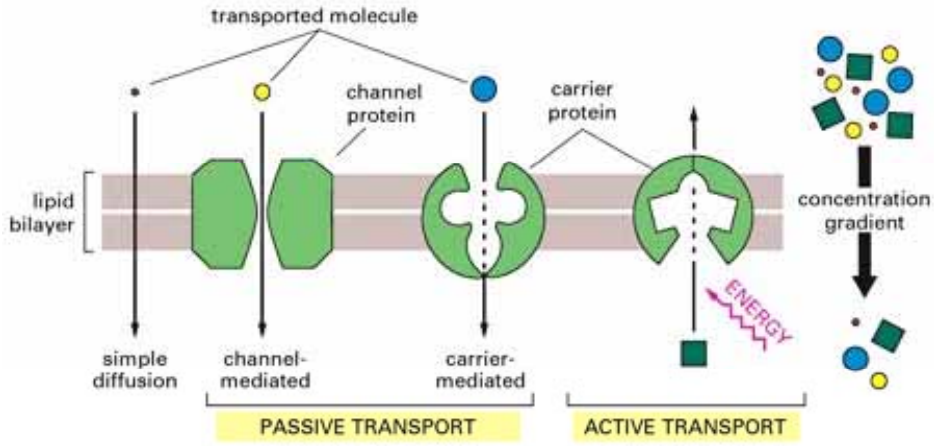
Optimality in Biological Systems

Cell Homeostasis



Cellular Metabolism

The individual cell is a complex feedback control system. It pumps ions across the cell membrane to maintain homeostatis, and has only limited energy to do so.



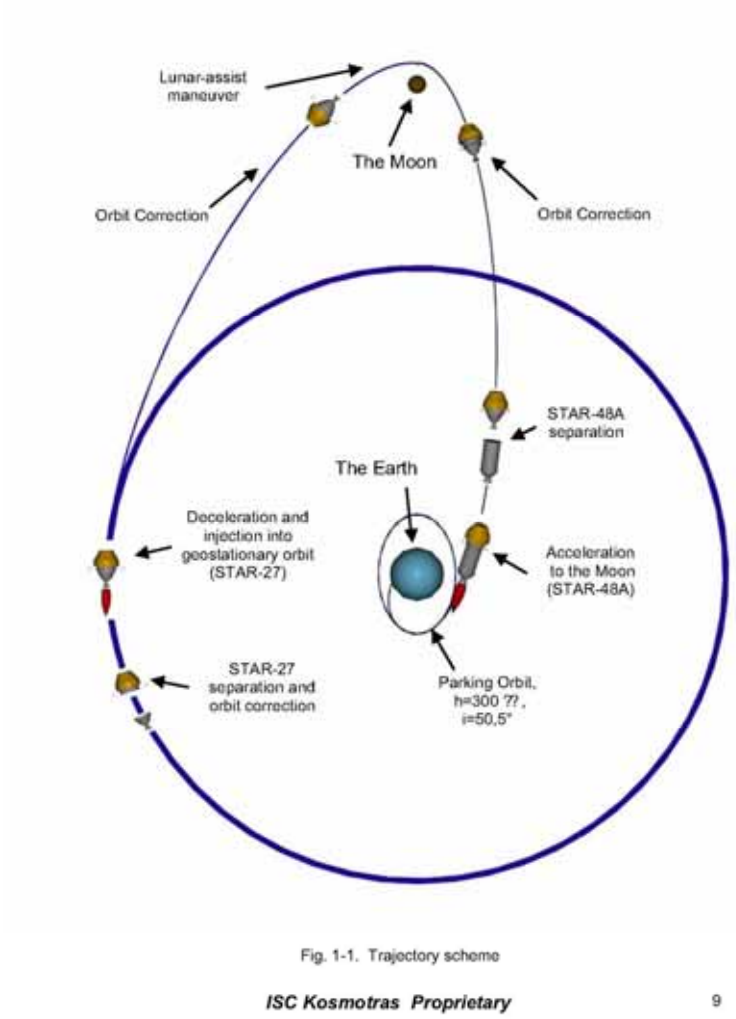
Permeability control of the cell membrane

http://www.accessexcellence.org/RC/VL/GG/index.html

R. Kalman 1960

Optimality in Control Systems Design

Rocket Orbit Injection



http://microsat.sm.bmstu.ru/e-library/Launch/Dnepr_GEO.pdf

Dynamics

$$\dot{r} = w$$

$$\dot{w} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{F}{m}\sin\phi$$

$$\dot{v} = \frac{-wv}{r} + \frac{F}{m}\cos\phi$$

$$\dot{m} = -Fm$$

Objectives

- Get to orbit in minimum time
- Use minimum fuel

Adaptive Control is generally not Optimal

Optimal Control is off-line,
and needs to know the system dynamics to solve design eqs.

We want ONLINE DIRECT ADAPTIVE OPTIMAL Control
For any performance cost of our own choosing

Reinforcement Learning turns out to be the key to this!

F. Lewis, Draguna Vrabie, and Vassilis Syrmos, "Optimal Control," 3rd edition, 2012. Chapter 11

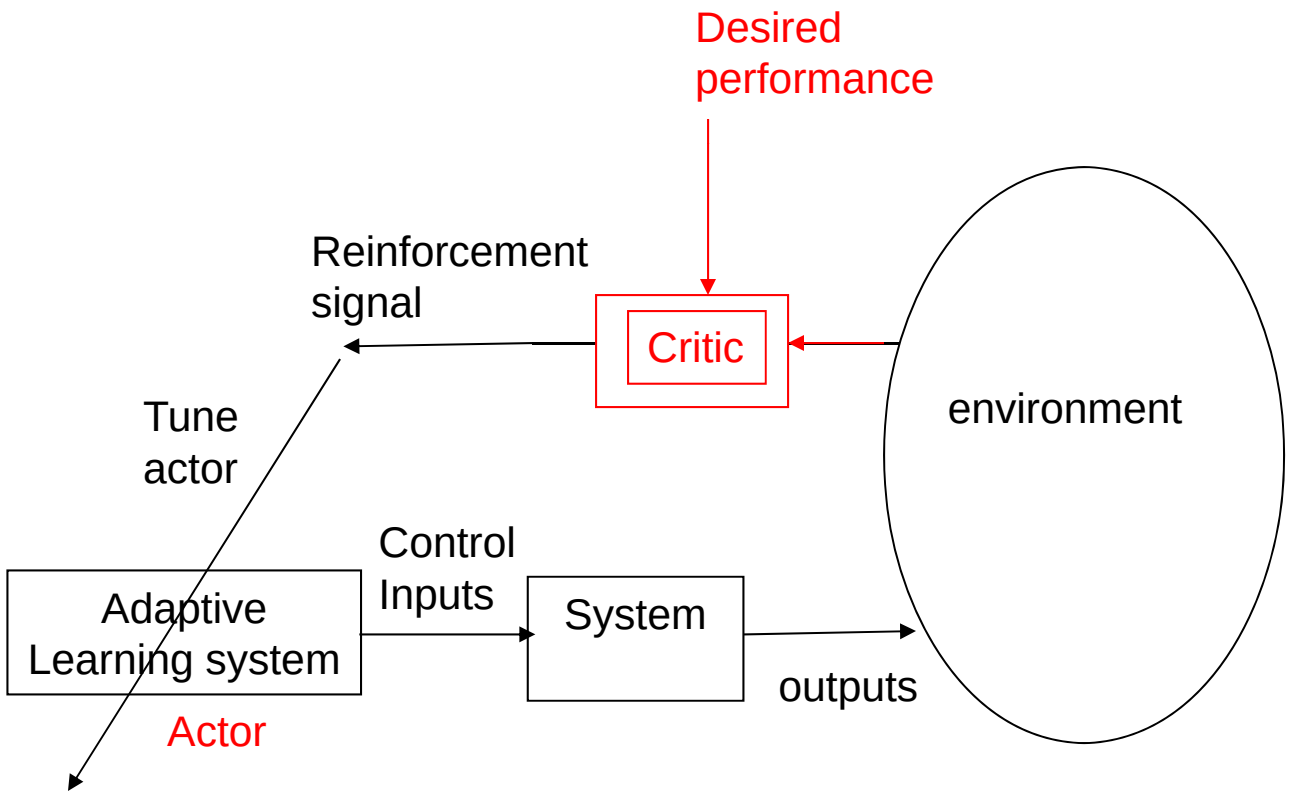
F.L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," IEEE Circuits & Systems Magazine, Invited Feature Article, pp. 32-50, Third Quarter 2009.

Different methods of learning

Reinforcement learning
Ivan Pavlov 1890s

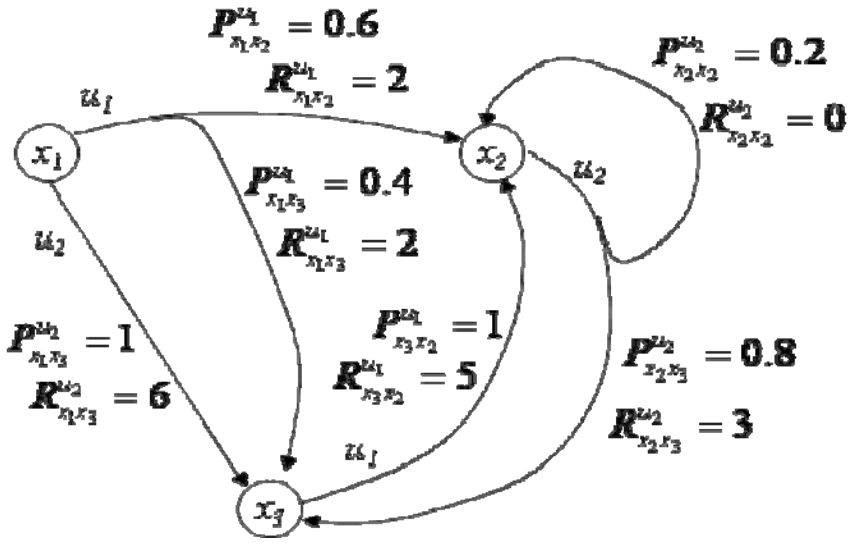
We want OPTIMAL performance
- ADP- Approximate Dynamic Programming

Actor-Critic Learning



Markov Decision Process

$$(X, U, P, R)$$



X= states
U= controls

$$P^{u}_{x,x'} = \Pr\{x' \mid x, u\}$$

Probability of going to state x' from state x given that the control is u

$$R^{u}_{xx'}$$

Expected reward on going to state x' from state x given that the control is u

---

R.S. Sutton and A.G. Barto, Reinforcement Learning– An Introduction, MIT Press, Cambridge, Massachusetts, 1998.

D.P. Bertsekas and J. N. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, MA, 1996.

W.B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality, Wiley, New York, 2009.

Optimal Sequential Decision Problems

control or action *strategy* or *policy*.

$$\pi : X \times U \to [0,1] \qquad \pi(x,u) = \Pr\{u \mid x\}$$

performance index

$$J_{k,T} = \sum_{i=0}^{T} \gamma^i r_{k+i} = \sum_{i=k}^{k+T} \gamma^{i-k} r_i$$

stage cost at time *k*

$$r_k = r_k(x_k, u_k, x_{k+1})$$

discount factor

$$0 \le \gamma < 1$$

stochastic or *mixed* policy

$$\pi_k(x_k, u_k)$$

probability distribution vectors

stationary policies

$$\pi_k(x,u) = \pi(x,u) = \Pr\{u \mid x\}, \quad for\ all\ k$$

deterministic policies = functions from X to U

$$\mu_k(x) : X \to U; k = 0,1,...$$

Stationary deterministic policies

$$\pi = \{\mu, \mu, \cdots\}$$

Value and Optimality

value of a policy $\pi(x,u)$

$$V_k^{\pi}(x) = E_{\pi}\{J_{k,T} \mid x_k = x\} = E_{\pi}\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\}.$$

Optimal control problem

determine a policy $\pi(x,u)$ to minimize the expected future cost

optimal policy

$$\pi^*(x,u) = \arg\min_{\pi} V_k^{\pi}(s) = \arg\min_{\pi} E_{\pi}\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\}.$$

optimal value

$$V_k^*(x) = \min_{\pi} V_k^{\pi}(x) = \min_{\pi} E_{\pi}\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\}.$$

Under the assumption that the Markov chain corresponding to each policy is ergodic, it can be shown that every MDP has a stationary deterministic optimal policy (Wheeler and Narendra 1986, Bertsekas and Tsitsiklis 1996).

A Backward Recursion for the Value

Value

$$V_k^{\pi}(x) = E_{\pi}\{J_k \mid x_k = x\} = E_{\pi}\{\sum_{i=k}^{k+T} \gamma^{i-k} r_i \mid x_k = x\}$$

$$V_k^{\pi}(x) = E_{\pi}\{r_k + \gamma \sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i \mid x_k = x\}$$

$$V_k^{\pi}(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma E_{\pi}\{\sum_{i=k+1}^{k+T} \gamma^{i-(k+1)} r_i \mid x_{k+1} = x'\} \right].$$

using the Chapman-Kolmogorov identity
and the Markov property

Backwards Recursion

$$V_k^{\pi}(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x') \right].$$

Dynamic Programming

The Backward Recursion

$$V_k^{\pi}(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x') \right].$$

Optimal Value

$$V_k^*(x) = \min_{\pi} V_k^{\pi}(x) = \min_{\pi} \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x') \right].$$

Bellman optimality principle

$$V_k^*(x) = \min_{\pi} \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x') \right].$$

$$u_k^* = \arg\min_{\pi} \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x') \right]$$

Under the assumption that the Markov chain corresponding to each policy is ergodic, every MDP has a stationary deterministic optimal policy:

$$V_k^*(x) = \min_u \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x') \right],$$

$$u_k^* = \arg\min_u \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^*(x') \right].$$

offline methods for working *backwards in time* to determine optimal policies

Bellman Equation

To derive forward-in-time methods for finding optimal values and optimal policies, set the time horizon *T* to infinity

$$J_k = \sum_{i=0}^{\infty} \gamma^i r_{k+i} = \sum_{i=k}^{\infty} \gamma^{i-k} r_i \ .$$

$$V^{\pi}(x) = E_{\pi}\{J_k \mid x_k = x\} = E_{\pi}\{\sum_{i=k}^{\infty} \gamma^{i-k} r_i \mid x_k = x\} .$$

The Backward Recursion for the Value

$$V_k^{\pi}(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^{\pi}(x') \right].$$

becomes the Bellman equation

$$V^{\pi}(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V^{\pi}(x') \right].$$

N simultaneous linear equations

Same value function appears on both sides!

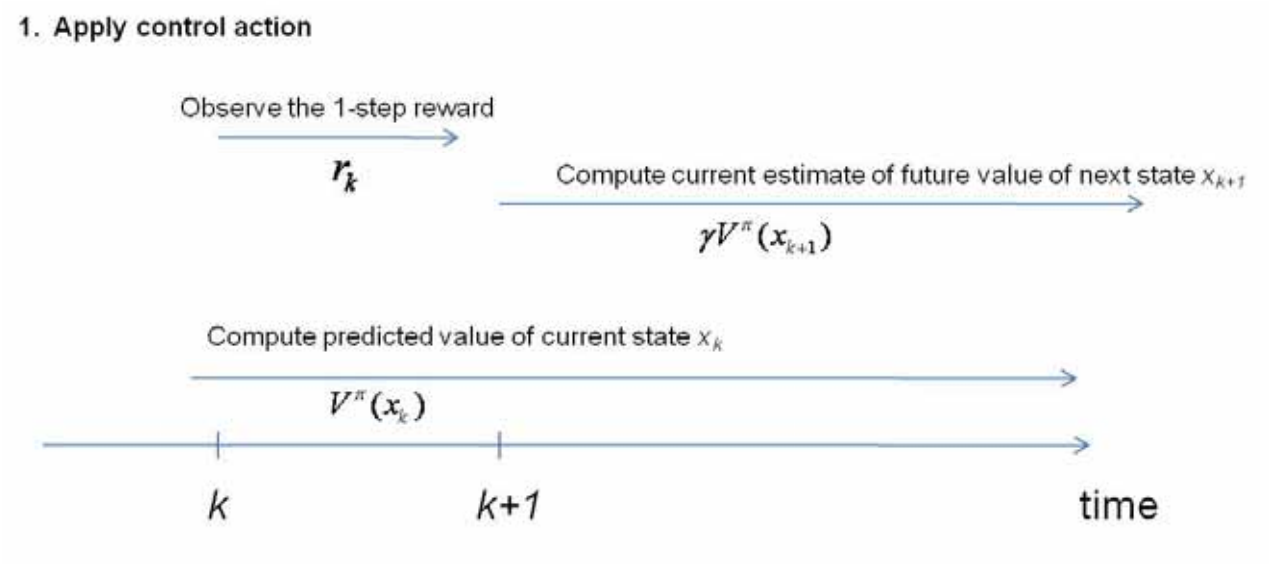
Bellman Equation

$$V^{\pi}(x) = \sum_{u} \pi(x,u) \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right].$$

Can be interpreted as a consistency equation that must be satisfied by the value function at each time stage.

Expresses a relation between the current value of being in state *x* and the value(s) of being in next state *x'* given that policy

Temporal Difference Idea - Later



2. Update predicted value to satisfy the Bellman equation

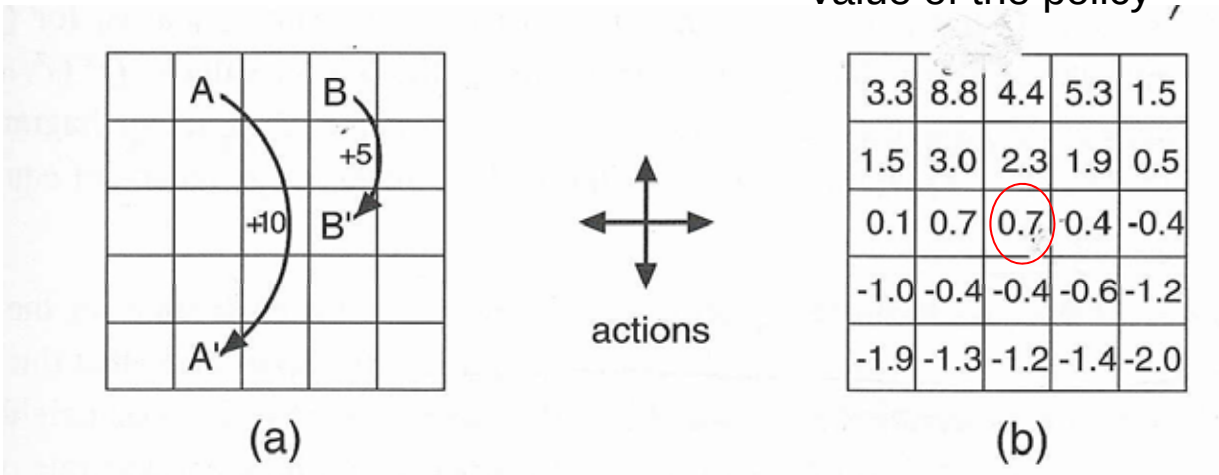
$$V^{\pi}(x_k) = r_k + \gamma V^{\pi}(x_{k+1})$$

3. Improve control action

Captures the action, observation, evaluation, and improvement mechanisms of reinforcement learning.

Example 1 : Gridworld

Value of the policy



At each cell the four actions are equally probable $\pi(s,a) = \frac{1}{4}$

Set discount rate $\gamma = 0.9$

Reward is 0 for all states except for A and B
except that moves which take the agent off the grid are valued at -1

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left\{ R^{a}_{ss'} + \gamma V^{\pi}(s') \right\}$$ Bellman eq must hold at each cell (state) *s*

Verify at red cell:

$$\tfrac{1}{4}\times 1\times\{0.9\times 0.4\} + \tfrac{1}{4}\times 1\times\{0.9\times 2.3\} + \tfrac{1}{4}\times 1\times\{0.9\times 0.7\} + \tfrac{1}{4}\times 1\times\{0.9\times(-0.4)\} = 0.7$$

To find the value of a policy with *N* states, must solve the *n* simultaneous linear equations

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left\{ R^{a}_{ss'} + \gamma V^{\pi}(s') \right\}$$ Bellman eq

Bellman Optimality Equation (HJB)

optimal value satisfies

$$V^*(x) = \min_\pi V^\pi(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^\pi(x') \right].$$

Bellman's optimality principle yields the *Bellman Optimality Equation*

$$V^*(x) = \min_\pi V^\pi(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^*(x') \right].$$

under the ergodicity assumption on the Markov chains corresponding to each policy, one has

$$V^*(x) = \min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^*(x') \right].$$

$$u^* = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^*(x') \right].$$

This consists of *N simultaneous* ***nonlinear*** eqs that can be solved for $V^*(s)$

Hamilton-Jacobi-Bellman (HJB) Equation

Compare to Bellman Equation- N linear equations

$$V^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^\pi(x') \right].$$

Example 1 : Gridworld

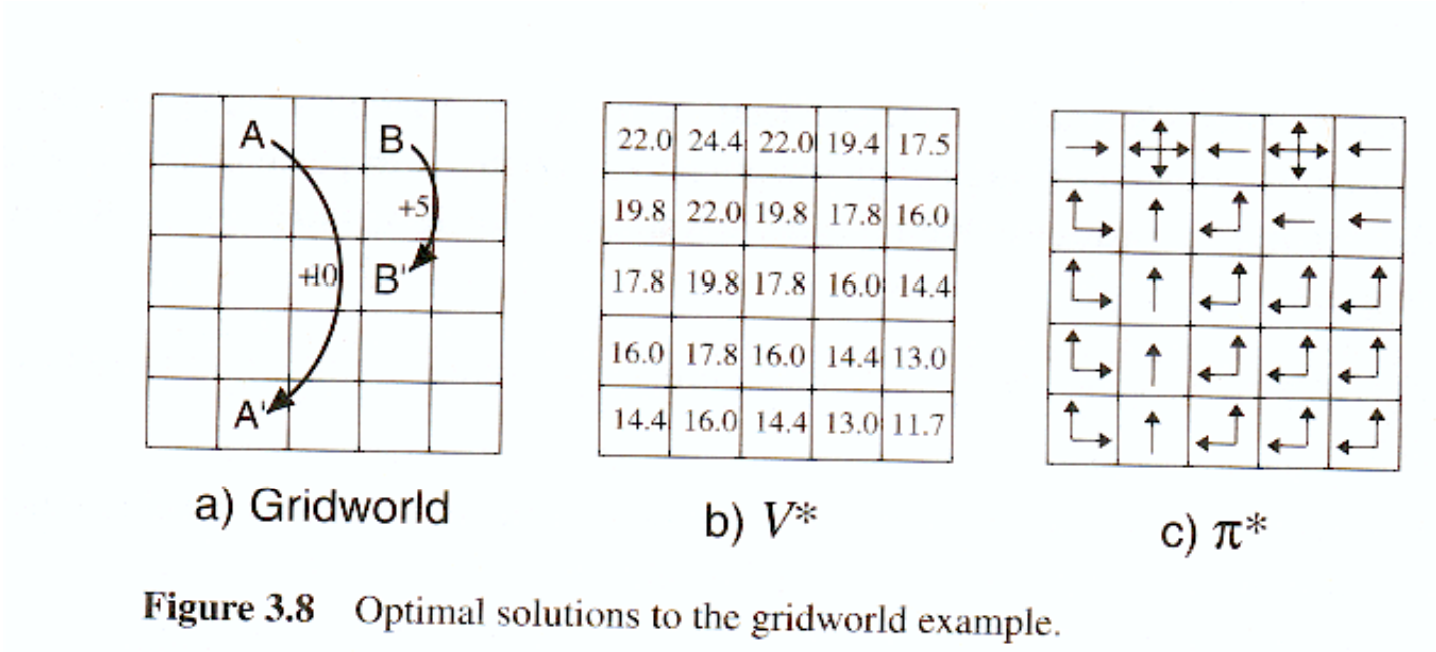


Figure 3.8 Optimal solutions to the gridworld example.

Solve 5x5=25 simultaneous nonlinear eqs

$$V^*(s) = \max_\pi \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left\{ R^a_{ss'} + \gamma V^*(s') \right\}$$

Then

$$\pi^*(s) = \arg\max_a \sum_{s'} P^a_{ss'} \left\{ R^a_{ss'} + \gamma V^*(s') \right\}$$

Comp. Intelligence guys MAXIMIZE the reward function

Bellman Equation for Discrete-Time Linear Quadratic Regulator (DT LQR)

Deterministic state transition equation

$$x_{k+1} = Ax_k + Bu_k$$

Deterministic infinite horizon performance index

$$J_k = \tfrac{1}{2}\sum_{i=k}^{\infty} r_i = \tfrac{1}{2}\sum_{i=k}^{\infty}(x_i^T Q x_i + u_i^T R u_i)\,.$$

Value function for a fixed policy

$$V(x_k) = \tfrac{1}{2}\sum_{i=k}^{\infty} r_i = \tfrac{1}{2}\sum_{i=k}^{\infty}(x_i^T Q x_i + u_i^T R u_i)\,.$$

Bellman Equation- a difference equation equivalent

$$V(x_k) = \tfrac{1}{2}\left(x_k^T Q x_k + u_k^T R u_k\right) + \tfrac{1}{2}\sum_{i=k+1}^{\infty}(x_i^T Q x_i + u_i^T R u_i) = \tfrac{1}{2}\left(x_k^T Q x_k + u_k^T R u_k\right) + V(x_{k+1})\,.$$

Does not contain A,B

value is quadratic in the state $V_k(x_k) = \tfrac{1}{2}x_k^T P x_k$,

$$2V(x_k) = x_k^T P x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P x_{k+1}\,,$$

SVFB policies $u_k = \mu(x_k) = -Kx_k$

$$2V(x_k) = x_k^T P x_k = x_k^T Q x_k + x_k^T K^T R K x_k + x_k^T (A-BK)^T P (A-BK) x_k\,.$$

Contains A, B

Bellman eq is a Lyapunov equation

$$(A-BK)^T P(A-BK) - P + Q + K^T R\, K = 0\,.$$

SVFB must be stabilizing

Bellman equation

$$V^{\pi}(x) = \sum_u \pi(x,u)\sum_{x'} P^u_{xx'}\left[R^u_{xx'} + \gamma V^{\pi}(x')\right].$$

Bellman Optimality Equation for DT LQR: the Algebraic Riccati Equation

Bellman Optimality equation (HJB)

$$V^*(x) = \min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^*(x') \right].$$

$$u^* = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^*(x') \right].$$

Bellman equation

$$V^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^\pi(x') \right].$$

Hamiltonian function

$$H(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P (Ax_k + Bu_k) - x_k^T P x_k \ .$$

For optimality, the Hamiltonian is equal to zero. Bellman eq is $H(x_k, u_k) = 0$

Stationarity condition

$$\partial H(x_k, u_k) / \partial u_k = 0$$

Optimal control

$$u_k = -Kx_k = -(B^T P B + R)^{-1} B^T P A x_k \ .$$

Bellman Optimality Equation = Algebraic Riccati Equation

$$A^T P A - P + Q - A^T P B (B^T P B + R)^{-1} B^T P A = 0 \, .$$

Closed-Loop Markov Chain

Policy Evaluation and Policy Improvement

consider algorithms that repeatedly interleave the two procedures:

Policy Evaluation by Bellman Equation:

$$V^{\pi}(x) = \sum_{u} \pi(x,u) \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right] \qquad \textit{for all } x \in S \subseteq X \ .$$

Policy Improvement:

$$\pi'(x,u) = \arg\min_{u} \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right] \qquad \textit{for all } x \in S \subseteq X \ .$$

S is a suitably selected subspace of the state-space

Policy Improvement makes $V^{\pi'}(x) \leq V^{\pi}(x)$

(Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998).

the policy $\pi'(x,u)$ is said to be *greedy* with respect to value function $V^{\pi}(x)$

At each step, one obtains a policy that is no worse than the previous policy.
Can prove convergence under fairly mild conditions to the optimal value and optimal policy.
Most such proofs are based on the Banach Fixed Point Theorem.
One step is a contraction map.

There is a large family of algorithms that implement the
policy evaluation and policy improvement procedures in various ways

Policy Iteration - An iterative solution algorithm for the optimal value and policy

Start with admissible initial policy $\pi_0(x,u)$

Policy evaluation by Bellman eq.

$$V_j(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right] \qquad \textit{for all } x \in X \,. \qquad N \text{ simultaneous linear equations}$$

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right] \qquad \textit{for all } x \in X \,.$$

Note that *j* is not the time or stage index *k*, but a PI step iteration index.

Policy Evaluation equation is a system of *N* simultaneous linear equations, one for each state.

Initial policy must be admissible- selected so that $V_1 \le V_0$

Then, for finite Markov chains with *N* states, PI converges to solution to HJB eq in a finite number of steps (less than or equal to *N*) because there are only a finite number of policies.

Solves the nonlinear HJB eq using successive solutions of LINEAR equations

---

It will be seen how to implement PI for dynamical systems **online in real-time** by observing data measured along the system trajectories.
Generally, data for multiple times *k* is needed to solve the Bellman equation at each step *j*

Iterative Policy Evaluation

The Bellman equation

$$V_j(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

A system of N simultaneous equations.

Instead of directly solving the Bellman equation, one can solve it by an iterative policy evaluation procedure. Note that Bellman eq is a fixed point equation for $V_j(.)$

It defines the *iterative policy evaluation* map

$$V_j^{i+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j^i(x') \right], \quad i = 1, 2, \ldots$$

Can be shown to be a contraction map under rather mild conditions

By the Banach fixed point theorem the iteration can be initialized at any nonnegative value of $V_j^1(.)$ and it will converge to the solution of Bellman eq.

Index *j* in refers to the step number of the PI Algorithm. By contrast *i* is an iteration index.

Compare to Dynamic programming

$$V_k^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_{k+1}^\pi(x') \right].$$

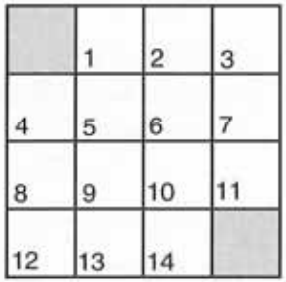
backwards recursion for the value at time *k* in terms of the value at time *k+1*.

Example 2: Gridworld

R.S. Sutton and A.G. Barto, Reinforcement Learning– 1998

Iterative Policy Evaluation

Example 4.1 Consider the 4 × 4 gridworld shown below.



$r = -1$ on all transitions

At each cell the four actions are equally probable

$$\pi(s,a) = \frac{1}{4}$$

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+1+k}$$

with all rewards -1 until final states (grey)

$$\gamma = 1$$

Sweep through the states

$$V_{i+1}^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left\{ R_{ss'}^a + \gamma V_i^{\pi}(s') \right\}$$

$$V_{i+1}^{\pi}(s) = \tfrac{1}{4}\times 1\times\{r_{sR} + s_i^R\} + \tfrac{1}{4}\times 1\times\{r_{sA} + s_i^A\} + \tfrac{1}{4}\times 1\times\{r_{sL} + s_i^L\} + \tfrac{1}{4}\times 1\times\{r_{sB} + s_i^B\}$$

V_k for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Solution to Bellman eq for the equi-probable control

Example 11.3-1: Solution of Partial Differential Equations: Relaxation Algorithms

Consider Poisson's equation in two dimensions

$$\Delta V(x,y) = \nabla^2 V(x,y) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) V(x,y) = f(x,y), \tag{11.3.7}$$

where $\Delta = \nabla^2$ is the Laplacian operator and ∇ the gradient. Function $f(x,y)$ is a forcing function, often specified on the boundary of a region.

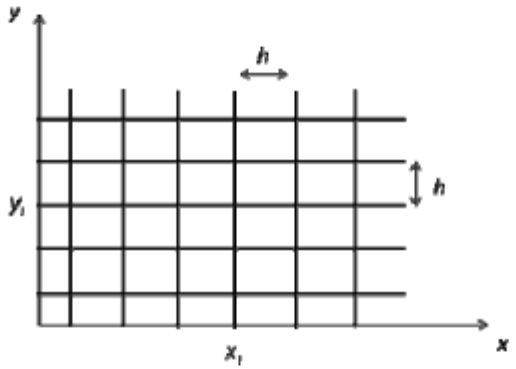


Figure 11.3-1. Shows sampling Poisson's equation in the *(x,y)* plane with a uniform mesh of size *h*.

Discretizing the equation on a uniform mesh as shown in Figure 11.3-1 with grid size h in the *(x,y)* plane one writes in terms of the forward difference

$$\frac{\partial V(x,y)}{\partial x} = \frac{1}{h}\left(V(x+h,y) - V(x,y)\right) + O(h),$$

$$\frac{\partial^2 V(x,y)}{\partial x^2} = \frac{1}{h^2}\left(V(x+h,y) + V(x-h,y) - 2V(x,y)\right) + O(h^2),$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial x^2}\right) V(x,y) = \frac{1}{h^2}\left(V(x+h,y) + V(x-h,y) + V(x,y+h) + V(x,y-h) - 4V(x,y)\right) \approx f(x,y).$$

Indexing the *(x,y)* positions with an index i and denoting $X_i = (x_i, y_i)$ one has approximately to order h^2

$$\frac{1}{h^2}\left(V(X_{i,R}) + V(X_{i,L}) + V(X_{i,u}) + V(X_{i,D}) - 4V(X_i)\right) = f(X_i), \tag{11.3.8}$$

for states not on the boundary and a similar equation for boundary states. $X_{i,R}$ denotes the *(x,y)* location of the state to the right of X_i, and similarly for states to the left, up, and down relative to X_i. This is a set of N simultaneous equations.

$$\frac{1}{h^2}\left(V(X_{i,R})+V(X_{i,L})+V(X_{i,u})+V(X_{i,D})-4V(X_i)\right)=f(X_i)\,, \qquad (11.3.9)$$

One can interpret (11.3.8) as a Bellman equation (11.2.17) for a properly defined underlying MDP. Specifically, define an MDP with stage costs equal to zero and $\gamma=1$. Then, one may interpret the state transitions as deterministic and the control as the equi-probable control with probabilities of ¼ (for non-boundary nodes) for moving up, right, down, and left. Alternatively, one may interpret the control as deterministic and the state transition probabilities as equi-probable with probabilities of ¼ for moving up, right, down, and left. In either case, the Bellman equation for the MDP is (11.3.8). Functions $f(X_i)$ may be interpreted as stage costs.

Now the iterative policy evaluation method of solution (11.3.6) performs the iterations

$$V^{m+1}(X_i)=\tfrac{1}{4}V^m(X_{i,U})+\tfrac{1}{4}V^m(X_{i,R})+\tfrac{1}{4}V^m(X_{i,D})+\tfrac{1}{4}V^m(X_{i,L})-\tfrac{h^2}{4}f(X_i)\,. \qquad (11.3.10)$$

The theory of MDP guarantees that this algorithm will converge to the solution of (11.3.8). These updates may be done for all nodes or states simultaneously. Then, this is nothing but the *relaxation method* for numerical solution of Poisson's equation. The relaxation algorithm converges, but may do so slowly. Variants have been developed to speed it up.

■

Value Iteration - An iterative solution algorithm for the optimal value and policy

PI= $V_j(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$

Start with initial policy $\pi_0(x,u)$

Value Update

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right] \quad \textit{for all } x \in S_j \subseteq X \,.$$

One step of iterative PI

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right] \qquad \textit{for all } x \in S_j \subseteq X \,.$$

Note that *j* is not the time or stage index *k*, but a VI step iteration index.

combine the value update and policy improvement into one equation

$$V_{j+1}(x) = \min_\pi \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

equivalently under the ergodicity assumption

$$V_{j+1}(x) = \min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

Value Update is a simple iteration, not N simultaneous linear equations
It uses a single iteration of *iterative policy evaluation*

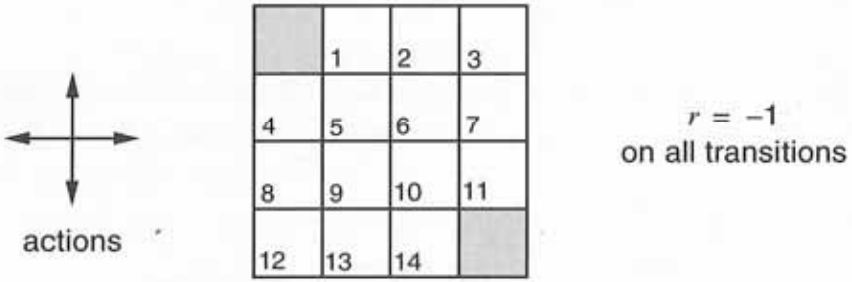
VI- Arbitrary Initial policy
PI- stabilizing initial policy

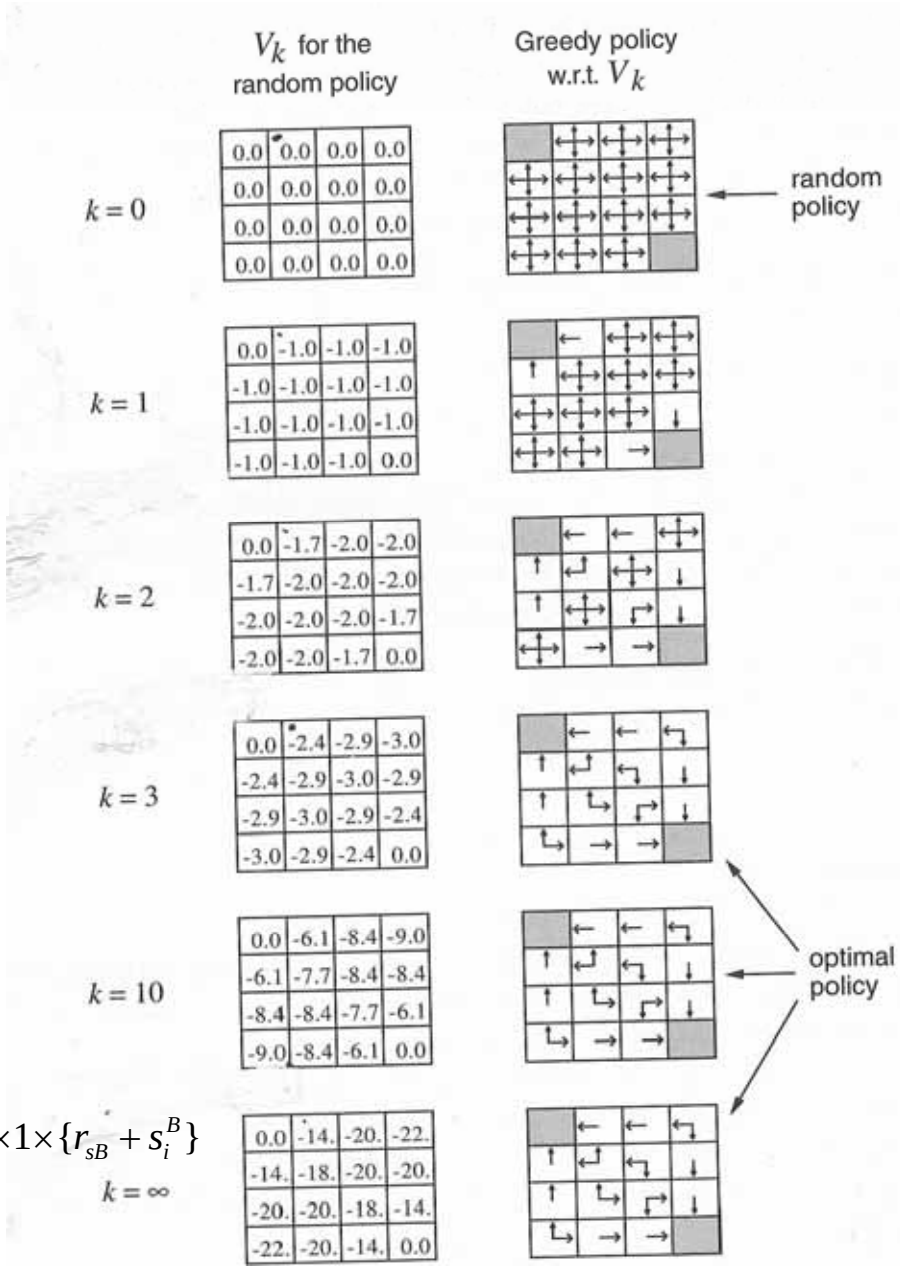
PI converges in a finite number of steps, less than or equal to *N*
VI converges slowly

Example 2: Gridworld

Value Iteration

Example 4.1 Consider the 4 × 4 gridworld shown below.





Sweep through the states or cells

Update value

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

Update control policy

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

First step uses the equi-probable control:

$$V^{\pi}_{i+1}(s) = \tfrac{1}{4}\times 1\times\{r_{sR} + s^R_i\} + \tfrac{1}{4}\times 1\times\{r_{sA} + s^A_i\} + \tfrac{1}{4}\times 1\times\{r_{sL} + s^L_i\} + \tfrac{1}{4}\times 1\times\{r_{sB} + s^B_i\}$$

Only the optimal policy is greedy wrt its own value

Asynchronous Value Iteration.

Value Iteration (VI) Algorithm

Initialize. Select an initial policy $\pi_0(x,u)$. Do for $j=0$ until convergence:

Value Update:

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right] \quad , \text{ for all } x \in S_j \subseteq X . \qquad (11.3.15)$$

Policy Improvement:

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_{j+1}(x') \right] \quad , \text{ for all } x \in S_j \subseteq X . \qquad (11.3.16)$$

Asynchronous Value Iteration. Standard VI takes the update set as $S_j = X$, *for all* j. That is, the value and policy are updated for all states simultaneously. *Asynchronous* VI methods perform the updates on only a subset of the states at each step. In the extreme case, one may perform the updates on only one state at each step.

Like coop ctrl gossip algorithms

It is shown in (Bertsekas and Tsitsiklis 1996) that standard VI ($S_j = X$, *for all* j) converges for finite MDP for any initial conditions when the discount factor satisfies $0 < \gamma < 1$. When $S_j = X$, *for all* j and $\gamma = 1$ an absorbing state is added and a 'properness' assumption is needed to guarantee convergence to the optimal value. When a single state is selected for value and policy updates at each step, the algorithm converges, for any choice of initial value, to the optimal cost and policy if each state is selected for update infinitely often. More general algorithms result if value update (11.3.11) is performed multiple times for various choices of S_j prior to a policy improvement. Then, it is required that updates (11.3.11) and (11.3.12) be performed infinitely often for each state, and a monotonicity assumption must be satisfied by the initial starting value.

Considering (11.2.20) as a fixed point equation, VI is based on the associated iterative map (11.3.11), (11.3.12), which can be shown under certain conditions to be a contraction map. In contrast to PI which converges under certain conditions in a finite number of steps, VI generally takes an infinite number of steps to converge (Bertsekas and Tsitsiklis 1996). Consider finite MDP and consider the transition probability graph having probabilities (11.2.2) for the Markov chain corresponding to an optimal policy $\pi^*(x,u)$. If this graph is acyclic for some $\pi^*(x,u)$, then VI converges in at most N steps when initialized with a large value.

Estimate for the future stage cost-to-go

Compare Value Iteration

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V_j(x') \right]$$

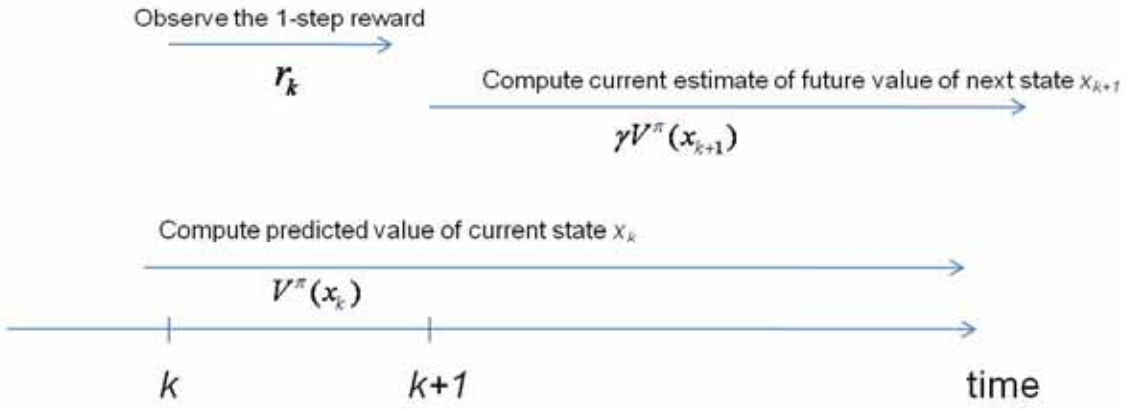
To Dynamic Programming

$$V^\pi_k(x) = \sum_u \pi(x,u) \sum_{x'} P^u_{xx'} \left[R^u_{xx'} + \gamma V^\pi_{k+1}(x') \right].$$

one can interpret $V_j(x')$

as an approximation or estimate for the future stage cost-to-go from the future state x'

1. **Apply control action**



2. **Update predicted value to satisfy the Bellman equation**

$$V^\pi(x_k) = r_k + \gamma V^\pi(x_{k+1})$$

3. **Improve control action**

Example 11.3-2: Deterministic Shortest Path Problems: the Bellman Ford Algorithm

A special case of finite MDP are the *shortest path problems* (Bertsekas and Tsitsiklis 1996) which are undiscounted $\gamma = 1$, and have a cost-free termination or absorbing state. These effectively have the time horizon T finite since the number of states is finite. The setup is shown in Figure 11.3-2.

Consider a directed graph $G = (V, E)$ with a nonempty finite set of N nodes $V = \{v_1, \cdots, v_N\}$ and a set of edges or arcs $E \subseteq V \times V$. We assume the graph is simple, that is, it has no repeated edges and $(v_i, v_i) \notin E, \forall i$ no self loops. Let the edges have weights $e_{ij} \geq 0$, with $e_{ij} > 0$ *if* $(v_i, v_j) \in E$ and $e_{ij} = 0$ otherwise. Note $e_{ii} = 0$. The set of neighbors of a node v_i is $N_i = \{v_j : (v_i, v_j) \in E\}$, that is, the set of nodes with arcs coming out from v_i.

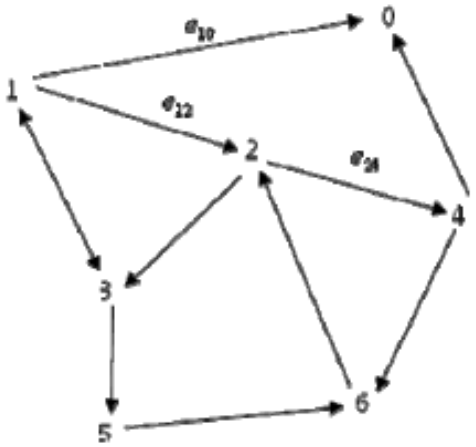


Figure 11.3-2. Sample graph for shortest path routing problem. The absorbing state *0* has only incoming edges. The objective is to find the shortest path from all nodes to node *0* using only local neighborhood computations.

Consider an agent moving through the graph along edges between nodes. Interpret the control at each node as the decision on which edge to follow leading out of that node. Assume deterministic controls and state transitions. Interpret the edge weights e_{ij} as deterministic costs incurred by moving along that link. Then, the Value Iteration Algorithm (11.3.14), (11.3.12) in this general digraph is

$$V_i(k+1) = \min_{j \in N_i}(e_{ij} + V_j(k)), \tag{11.3.17}$$

$$u_i(k+1) = \arg\min_{j \in N_i}(e_{ij} + V_j(k)), \tag{11.3.18}$$

with k an iteration index. One has changed the notation to conform to existing practice in cooperative control theory. Endow the graph with one node, v_0, which uses as its update law $V_0(k+1) = V_0(k) = 0$. This corresponds to an absorbing state in MDP parlance. It is interpreted as a node having only incoming edges, none outgoing.

The VI Algorithm in this scenario is nothing but the Bellman-Ford Algorithm, which finds the shortest path from any node in the graph to the absorbing node v_0.

In the terminology of cooperative systems, each node v_i is endowed with two state variables. The node state variable $V_i(k)$ keeps track of the value, that is, the shortest path length to node v_0, while node state variable $u_i(k)$ keeps track of which direction to follow while leaving the *i*-th node in order to follow the shortest path.

The VI update iterations may be performed on all nodes simultaneously. Alternatively, updates may be performed on one state at a time, as in asynchronous VI. With simultaneous updates at all nodes, according to results about VI (Bertsekas and Tsitsiklis 1996) it is known that this algorithm converges to the solution to the shortest path problem in a finite number of steps (less than or equal to N) if there is a path from every node to node v_0 and the graph is acyclic. It is known further that the algorithm converges, possibly in an infinite number of iterations, if there are no cycles with net negative gain (that is, the product of gains around the cycle is not negative). With asynchronous updates, the algorithm converges under these connectivity conditions if each node is selected for update infinitely often.

What is a state?

Note that in the terminology of MDP, the nodes are termed states, so that the state space is finite. On the other hand, in the terminology of feedback control systems theory, the state (variable) of each node is $V_i(m)$, which is a real number so that the state-space is continuous.

■

Example 11.3-3: **Cooperative Control Systems**

Consider the directed graph $G=(V,E)$ with N nodes $V=\{v_1,\cdots,v_N\}$ and edge weights $e_{ij}\geq 0$, with $e_{ij}>0$ *if* $(v_i,v_j)\in E$ and $e_{ij}=0$ otherwise. The set of neighbors of a node v_i is $N_i=\{v_j:(v_i,v_j)\in E\}$, that is, the set of nodes with arcs coming out from v_i. Define the out-degree of node *i* in graph *G* as $d_i=\sum_{j\in N_i} e_{ij}$. Figure 11.3-3 shows a representative graph topology.

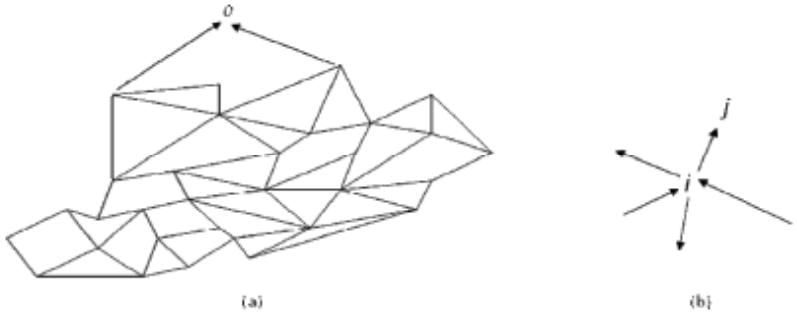


Figure 11.3-3. Cooperative control of multiple systems linked by a communication graph structure. The objective is for all nodes to reach consensus to the value of control node *0* by using only local neighborhood communication. (a) Representative graph structure. (b) Local neighborhood of node *i*. Each node has edges incoming and outgoing. The outgoing edges show the states reached from node *i* in one step.

On the graph *G*, define a MDP that has a stochastic policy $\pi(v_i,u_{ij})=\Pr\{u_{ij}\,|\,v_i\}$ where u_{ij} means the control action that takes the MDP from node v_i to node v_j. Let the transition probabilities be deterministic. Endow the graph with one absorbing node v_0 that is connected to a few of the existing nodes. If there is an edge from node v_i to node v_0, define $b_i\equiv e_{i0}$ as its edge weight. Let actions u_{ij} have probabilities $u_{ii}=1/(1+d_i+b_i)$, $u_{ij}=e_{ij}/(1+d_i+b_i)$, that is, the MDP may return to the same state in one step. Let the stage costs all be zero. Then, the Bellman equation (11.2.17) is

$$V_i=\frac{1}{1+d_i+b_i}\left[V_i+\sum_{j\in N_i} e_{ij}V_j \quad + \quad b_iV_0\right], \tag{11.3.19}$$

where V_i is the value of node v_i. This is a set of N simultaneous equations in the values $V_i, i=1,N$ of the nodes.

Iterative policy evaluation (11.3.6) can be used to solve this set of equations. The iterative policy evaluation algorithm is written here as

$$V_i(k+1)=\frac{1}{1+d_i+b_i}\left[V_i(k)+\sum_{j\in N_i} e_{ij}V_j(k) \quad + \quad b_iV_0\right], \tag{11.3.20}$$

with k the iteration index, which can be thought of here as a time index. Node v_0 keeps its value constant at V_0.

The updates in (11.3.20) may be done simultaneously at all nodes. Then the theory of MDP shows that it is guaranteed to converge to the solution of the set of equations (11.3.19). Alternatively, one may use the theory of asynchronous VI or generalized PI to motivate other update schemes. For instance, if only one node is updated at each value of k, the theory of MDP shows that the algorithm still converges as long as each node is selected for update infinitely often.

Algorithm (11.3.20) can be written as the local control protocol

$$V_i(k+1) = V_i(k) + \frac{1}{1+d_i+b_i}\left[\sum_{j\in N_i} e_{ij}\left(V_j(k)-V_i(k)\right) \quad + \quad b_i\left(V_0 - V_i(k)\right)\right]. \qquad (11.3.21)$$

On convergence $V_i(k+1) = V_i(k)$ so that the term in square brackets converges to zero. Assuming all nodes have a path to the absorbing node v_0, it is easy to show that this guarantees that $\|V_j(k)-V_i(k)\| \to 0, \|V_0 - V_i(k)\| \to 0$, *for all* i, j, that is, all nodes reach the same consensus value, namely the value of the absorbing node (Jadbabaie et al. 2003, Olfati-Saber and Murray 2004).

The term in square brackets in (11.3.21) is known as the *temporal difference error* for this MDP.

Routing Graph vs. Control Graph

The graphs used in routing problems have edges from node i to node j if the transition probabilities (11.2.2) in the MDP using a fixed policy $\pi(x_i,u)$, namely $p_{ij} \equiv P^{\pi}_{x_i,x_j} = \sum_u \pi(x_i,u)P^u_{x_i,x_j}$, are nonzero. The edge weights are taken as $e_{ij} = p_{ij}$. Then, e_{ij} is nonzero if there is an edge coming *out* of node i to node j. By contrast, the edge weights a_{ij} for graphs in cooperative control problems are generally taken as nonzero if there is an edge coming *in* from node j to node i, that is $a_{ij} > 0$ *iff* $(v_j, v_i) \in E$ This is interpreted to mean that information from node j is available to node i for its decision process in computing its control input.

It is convenient to think of the former as *motion or routing graphs*, and the latter as *information flow* or *decision & control graphs*. In fact, the routing graph is the reverse of the decision graph. The *reverse graph* of a given graph $G = (V,E)$ is the graph with the same node set, but all edges reversed. Note that the Bellman-Ford protocols (11.3.17), (11.3.18) assume that node i gets information from its neighbor node j, but that the shortest path problem is in fact solved for motion in the reverse graph having edges e_{ij}.

Define the matrix of transition probabilities $P = [p_{ij}]$ and the adjacency matrix $A = [a_{ij}]$. Then $A = P^T$.

■

Example 11.3-4: Policy Iteration and Value Iteration for the DT LQR

1. Policy Iteration: Hewer's Algorithm

Value Update

$$V^{j+1}(x_k) = \tfrac{1}{2}\left(x_k^T Q x_k + u_k^T R u_k\right) + V^{j+1}(x_{k+1}).$$

$$x_k^T P^{j+1} x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P^{j+1} x_{k+1},$$

$$0 = (A - BK^j)^T P^{j+1}(A - BK^j) - P^{j+1} + Q + (K^j)^T R\, K^j. \qquad \text{Lyapunov equation}$$

Policy Improvement

$$\mu^{j+1}(x_k) = K^{j+1} x_k = \arg\min(x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P^{j+1} x_{k+1}),$$

$$K^{j+1} = -(B^T P^{j+1} B + R)^{-1} B^T P^{j+1} A.$$

2. Value Iteration: Lyapunov recursions

$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j(x')\right]$$

$$x_k^T P^{j+1} x_k = x_k^T Q x_k + u_k^T R u_k + x_{k+1}^T P^j x_{k+1},$$

$$P^{j+1} = (A - BK^j)^T P^j (A - BK^j) + Q + (K^j)^T R\, K^j. \qquad \text{Lyapunov recursion}$$

3. Iterative Policy Evaluation

$$P^{j+1} = (A - BK)^T P^j (A - BK) + Q + K^T R\, K.$$

this recursion converges to the solution of the Lyapunov equation

All algorithms have a model-free scalar form based on measuring states, and a model-based matrix form based on cancelling states

Generalized Policy Iteration

Start with initial policy $\pi_0(x,u)$

Policy update iterations. Set $V_j^0(x) = V_{j-1}(x)$

$$V_j^{i+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j^i(x') \right], \quad i = 1,2,... \qquad \textit{for all } x \in X \ .$$

Policy Improvement

$$\pi_{j+1}(x,u) = \arg\min_u \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j(x') \right] \qquad \textit{for all } x \in X \ .$$

Policy iteration value update

$$V_j(x) = \sum_u \pi_j(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j(x') \right] \quad \textit{for all } x \in X \ .$$

N simultaneous linear equations

GPI

If *i=1*, one policy update iteration. Same as value iteration

As *i* gets large, GPI is same as policy iteration

Methods for Implementing PI and VI

Exact Computation. **Model-Based.**

All transition probabilities are available to solve the Bellman equation

$$V^{\pi}(x) = \sum_{u} \pi(x,u) \sum_{x'} P^{u}_{xx'} \left[R^{u}_{xx'} + \gamma V^{\pi}(x') \right].$$

Monte Carlo Learning. **Model-Free. Episodic tasks.**

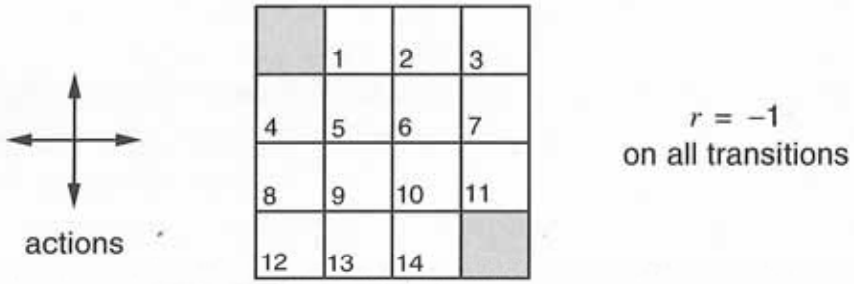
Approximate the value function $V^{\pi}(x) = E_{\pi}\{J_k \mid x_k = x\} = E_{\pi}\{\sum_{i=k}^{\infty} \gamma^{i-k} r_i \mid x_k = x\}$.

by repeated trials along sample paths until termination.
At termination, update all the states through which the path passed.

For finite MDP, Montecarlo converges to the true value function if all states are visited infinitely often.

Problem of maintaining *Exploration*. Method of *Exploring Starts*.

Example 4.1 Consider the 4×4 gridworld shown below.



Related to Iterative Learning Control

Temporal Difference Learning (TDL) Along State Trajectories

Real-Time online

Can be model-free or partially model-free

Stochastic approximation techniques

Can write Bellman equation as

$$V^{\pi}(x_k) = E_{\pi}\{r_k \mid x_k\} + \gamma E_{\pi}\{V^{\pi}(x_{k+1}) \mid x_k\}.$$

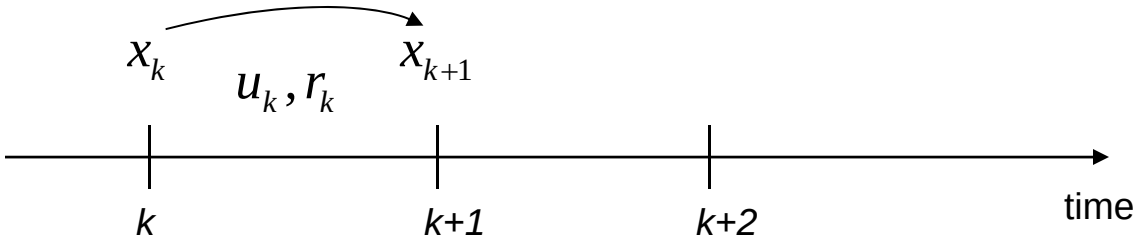
Evaluate along a sample path.
Deterministic Bellman eq

$$V^{\pi}(x_k) = r_k + \gamma V^{\pi}(x_{k+1}),$$

Temporal Difference Error

$$e_k = -V^{\pi}(x_k) + r_k + \gamma V^{\pi}(x_{k+1}),$$

Measure the data set (x_k, u_k, r_k, x_{k+1})



Discrete-Time Optimal Adaptive Control

system $x_{k+1} = f(x_k) + g(x_k)u_k$

cost $V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)$

This is like:

$P^u_{x,x'} = \Pr\{x' \mid x, u\}$

$R^u_{xx'}$

Example $r(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k$

Difference eq equivalent $V_h(x_k) = r(x_k, u_k) + \gamma \sum_{i=k+1}^{\infty} \gamma^{i-(k+1)} r(x_i, u_i)$

Bellman equation $V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) \quad , \quad V_h(0) = 0$

$$V_h(x_k) = x_k^T Q x_k + u_k^T R u_k + \gamma V_h(x_{k+1})$$

Temporal difference (TD) error

$$e_k = x_k^T Q x_k + u_k^T R u_k + \gamma V_h(x_{k+1}) - V_h(x_k)$$

Control policy $u_k = h(x_k)$ = the prescribed control input function

$\pi(x,u) = \Pr\{u \mid x\}$

Example $u_k = -K x_k$ Linear state variable feedback

Discrete-Time Optimal Adaptive Control

value $V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)$

Bellman Equation $V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$

$u_k = h(x_k)$ = the prescribed control policy

Hamiltonian $H(x_k, \nabla V(x_k), h) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) - V_h(x_k)$

The Bellman eq is $H(x_k, \nabla V(x_k), h) = 0$

Optimal cost $V^*(x_k) = \min_h (r(x_k, h(x_k)) + \gamma V_h(x_{k+1}))$

Bellman's Principle gives Bellman opt. eq= DT HJB

$$V^*(x_k) = \min_{u_k} (r(x_k, u_k) + \gamma V^*(x_{k+1}))$$

Dynamic programming- Backwards in time solution

Optimal Control $h^*(x_k) = \arg\min_{u_k} (r(x_k, u_k) + \gamma V^*(x_{k+1}))$

$$u^*(x_k) = -\tfrac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}}$$

The Solution: Hamilton-Jacobi-Bellman Equation

System $$x_{k+1} = f(x_k) + g(x_k)u_k$$

$$V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u_i^T R u_i$$

Bellman opt eq= DT HJB equation

$$V^*(x_k) = \min_{u_k}\left[x_k^T Q x_k + u_k^T R u_k + V^*(x_{k+1})\right]$$

$$= \min_{u_k}\left[x_k^T Q x_k + u_k^T R u_k + V^*\left(f(x_k) + g(x_k)u_k \right)\right]$$

Minimize wrt u_k

$$2Ru_k + g(x_k)^T \frac{dV^*(x_{k+1})}{dx_{k+1}} = 0$$

$$u^*(x_k) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}}$$

Offline solution
Difficult to solve
Contains the dynamics

DT Optimal Control – Linear Systems Quadratic cost (LQR)

system

$$x_{k+1} = Ax_k + Bu_k$$

cost $V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u_i^T R u_i$

Fact. The cost is quadratic $V(x_k) = x_k^T P x_k$ for some symmetric matrix P

HJB = DT Riccati equation

$$0 = A^T PA - P + Q - A^T PB(R + B^T PB)^{-1} B^T PA$$

Optimal Control $u_k = -Lx_k$

$$L = (R + B^T PB)^{-1} B^T PA$$

Optimal Cost

$$V^*(x_k) = x_k^T P x_k$$

Off-line solution
Dynamics must be known

DT Policy Iteration

e.g. Control policy = SVFB

$$h(x_k) = -Lx_k$$

Cost for any given control policy $h(x_k)$ satisfies the recursion

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$

Bellman eq.

Recursive form
Consistency equation

Recursive solution - Actor/Critic Structure

Pick stabilizing initial control

Policy Evaluation

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

f(.) and g(.) do not appear

Policy Improvement

$$h_{j+1}(x_{k+1}) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Howard (1960) proved convergence for MDP

Temporal difference

$$e_k = -V_{j+1}(x_k) + r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

DT Policy Iteration – Linear Systems Quadratic Cost- LQR

$$x_{k+1} = Ax_k + Bu_k = (A - BL)x_k, \qquad u_k = -Lx_k$$

For any stabilizing policy, the cost is

$$V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u^T(x_i) R u(x_i)$$

LQR value is quadratic $V(x) = x^T P x$

Solves Lyapunov eq. without knowing A and B

DT Policy iterations

$$V_{j+1}(x_k) = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k) + V_{j+1}(x_{k+1})$$

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}}$$

Equivalent to an Underlying Problem- DT LQR:

$$(A - BL_j)^T P_{j+1} (A - BL_j) - P_{j+1} = -Q - L_j^T R L_j$$

DT Lyapunov eq.

$$L_{j+1} = (R + B^T P_{j+1} B)^{-1} B^T P_{j+1} A$$

Hewer proved convergence in 1971

Policy Iteration Solves Lyapunov equation WITHOUT knowing System Dynamics

DT Policy Iteration – Linear Systems Quadratic Cost- LQR

DT Policy iterations

$$V_{j+1}(x_k) = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k) + V_{j+1}(x_{k+1})$$

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}}$$

How to implement online?

DT Policy Iteration – How to implement online?
Linear Systems Quadratic Cost- LQR

$$x_{k+1} = Ax_k + Bu_k \qquad V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u(x_i) R u(x_i)$$

LQR cost is quadratic $V(x) = x^T P x$ for some matrix P

DT Policy iterations

Solves Lyapunov eq. without knowing A and B

$$V_{j+1}(x_k) = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k) + V_{j+1}(x_{k+1})$$

$$x_k^T P_{j+1} x_k - x_{k+1}^T P_{j+1} x_{k+1} = x_k^T Q x_k + u_j^T R u_j$$

$$\begin{bmatrix} x_k^1 & x_k^2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} - \begin{bmatrix} x_{k+1}^1 & x_{k+1}^2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix}$$

$$= \begin{bmatrix} p_{11} & p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} (x_k^1)^2 \\ 2x_k^1 x_k^2 \\ (x_k^2)^2 \end{bmatrix} - \begin{bmatrix} p_{11} & p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} (x_{k+1}^1)^2 \\ 2x_{k+1}^1 x_{k+1}^2 \\ (x_{k+1}^2)^2 \end{bmatrix}$$

← Quadratic basis set

$$W_{j+1}^T \left[\varphi(x_k) - \varphi(x_{k+1}) \right] = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k)$$

Then update control using

$$h_j(x_k) = L_j x_k = (R + B^T P_j B)^{-1} B^T P_j A x_k$$

Need to know A AND B for control update

Implementation- DT Policy Iteration Nonlinear Case

Value Function Approximation (VFA)

$$V(x) = W^T \varphi(x)$$

weights

basis functions

LQR case- V(x) is quadratic

$$V(x) = x^T P x = W^T \varphi(x)$$

$$\varphi(x) = \left[x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_n^2\right]'.$$

Quadratic basis functions

$$W^T = [p_{11} \quad p_{12} \quad \cdots]$$

Nonlinear system case- use Neural Network

Implementation- DT Policy Iteration

Value function update for given control

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

Assume measurements of x_k and x_{k+1} are available to compute u_{k+1}

VFA $\quad V_j(x_k) = W_j^T \varphi(x_k)$

Then

regression matrix

$$W_{j+1}^T \left[\varphi(x_k) - \gamma\varphi(x_{k+1})\right] = r(x_k, h_j(x_k))$$

Since x_{k+1} is measured, do not need knowledge of f(x) or g(x) for value fn. update

Indirect Adaptive control with identification of the optimal value

Solve for weights in real-time using RLS

or, batch LS- many trajectories with different initial conditions over a compact set

Then update control using

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}} = -\frac{1}{2} R^{-1} g(x_k)^T \nabla\varphi^T(x_{k+1}) W_{j+1}^T$$

Need to know $g(x_k)$ for control update

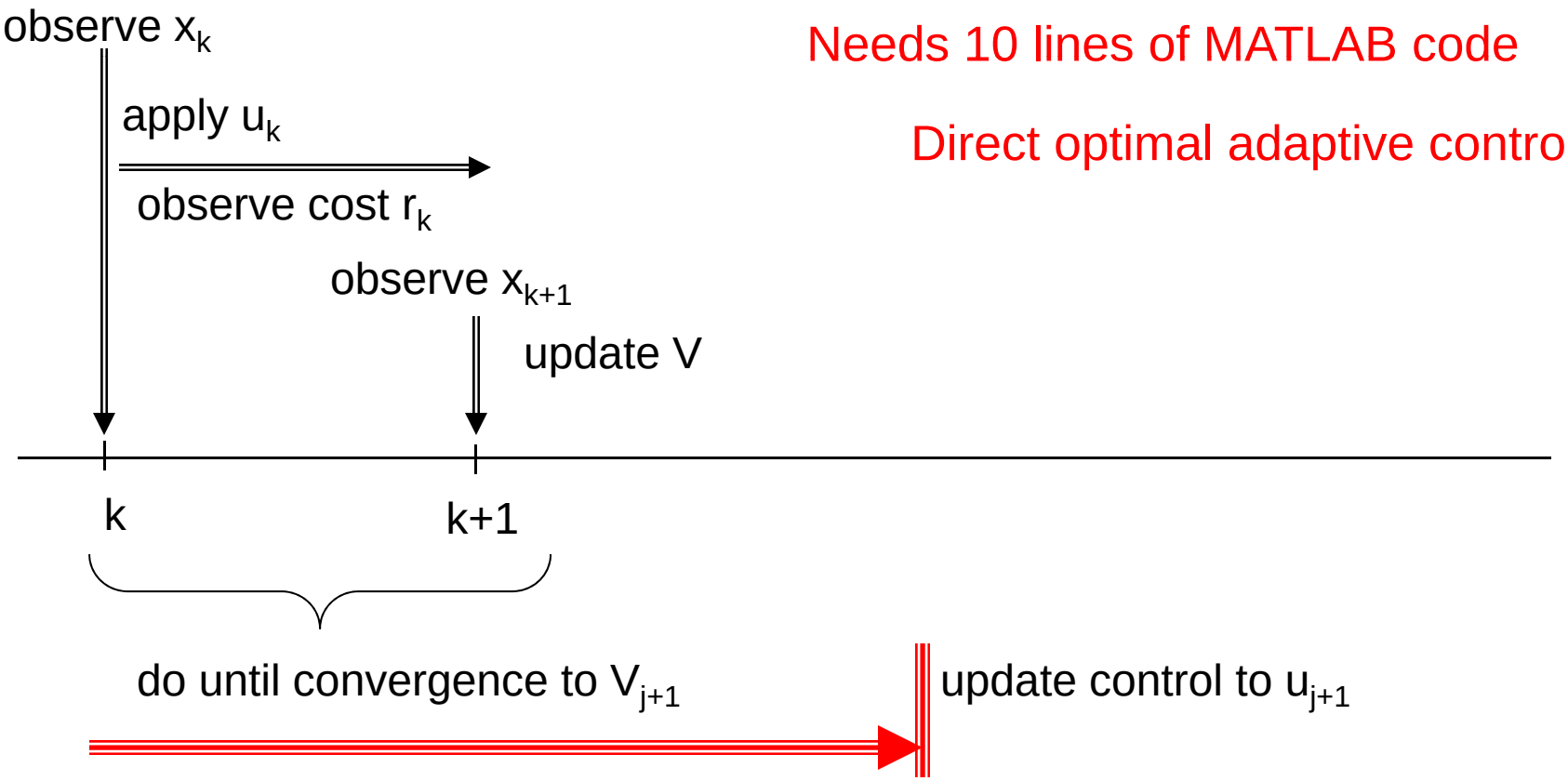
1. Select control policy

Solves Lyapunov eq. without knowing dynamics

2. Find associated cost $V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$

$$W_{j+1}^T[\varphi(x_k) - \gamma\varphi(x_{k+1})] = r(x_k, h_j(x_k))$$

3. Improve control $u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_j(x_{k+1})}{dx_{k+1}}$

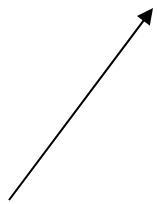


Needs 10 lines of MATLAB code

Direct optimal adaptive control

Persistence of Excitation

$$W_{j+1}^{T}\left[\varphi(x_k) - \gamma\varphi(x_{k+1})\right] = r(x_k, h_j(x_k))$$



Regression vector must be PE

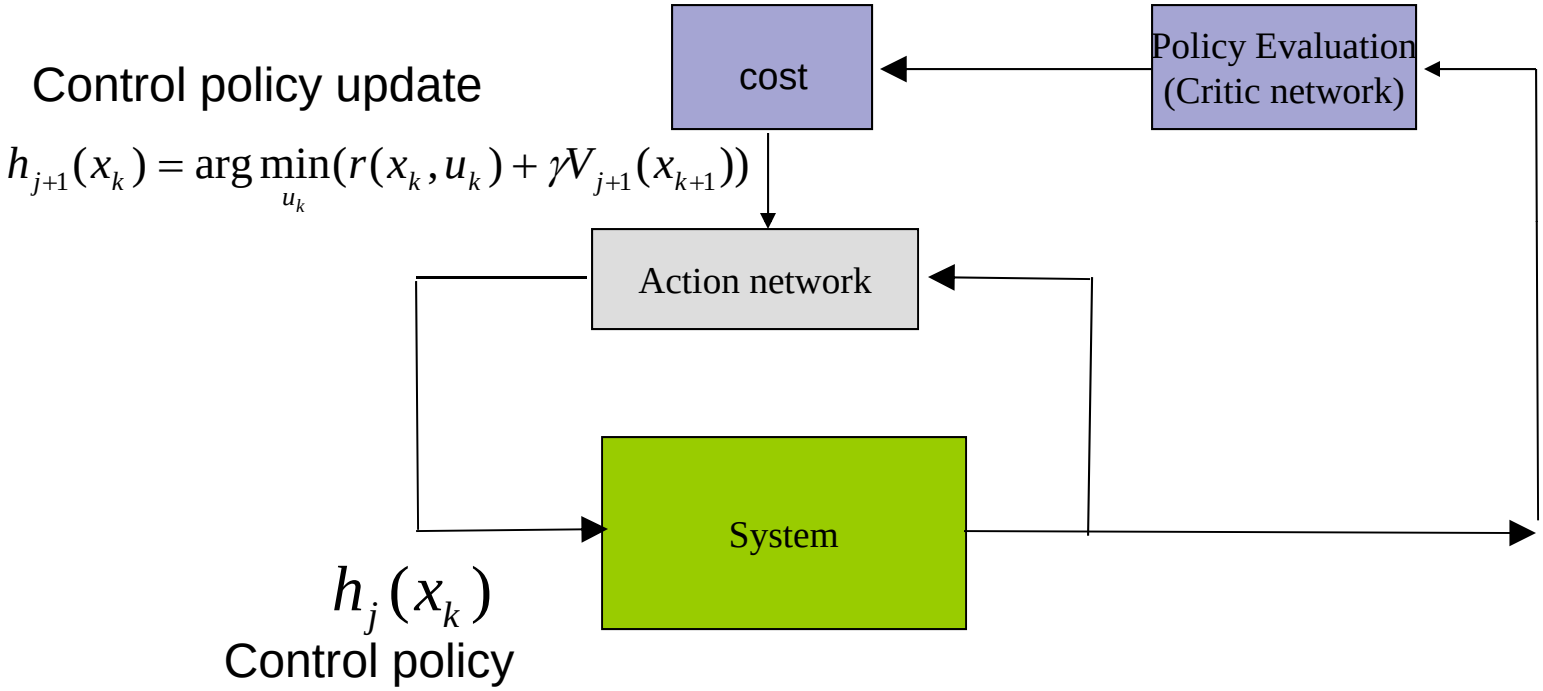
Adaptive Critics

Use RLS until convergence

The Adaptive Critic Architecture

Value update

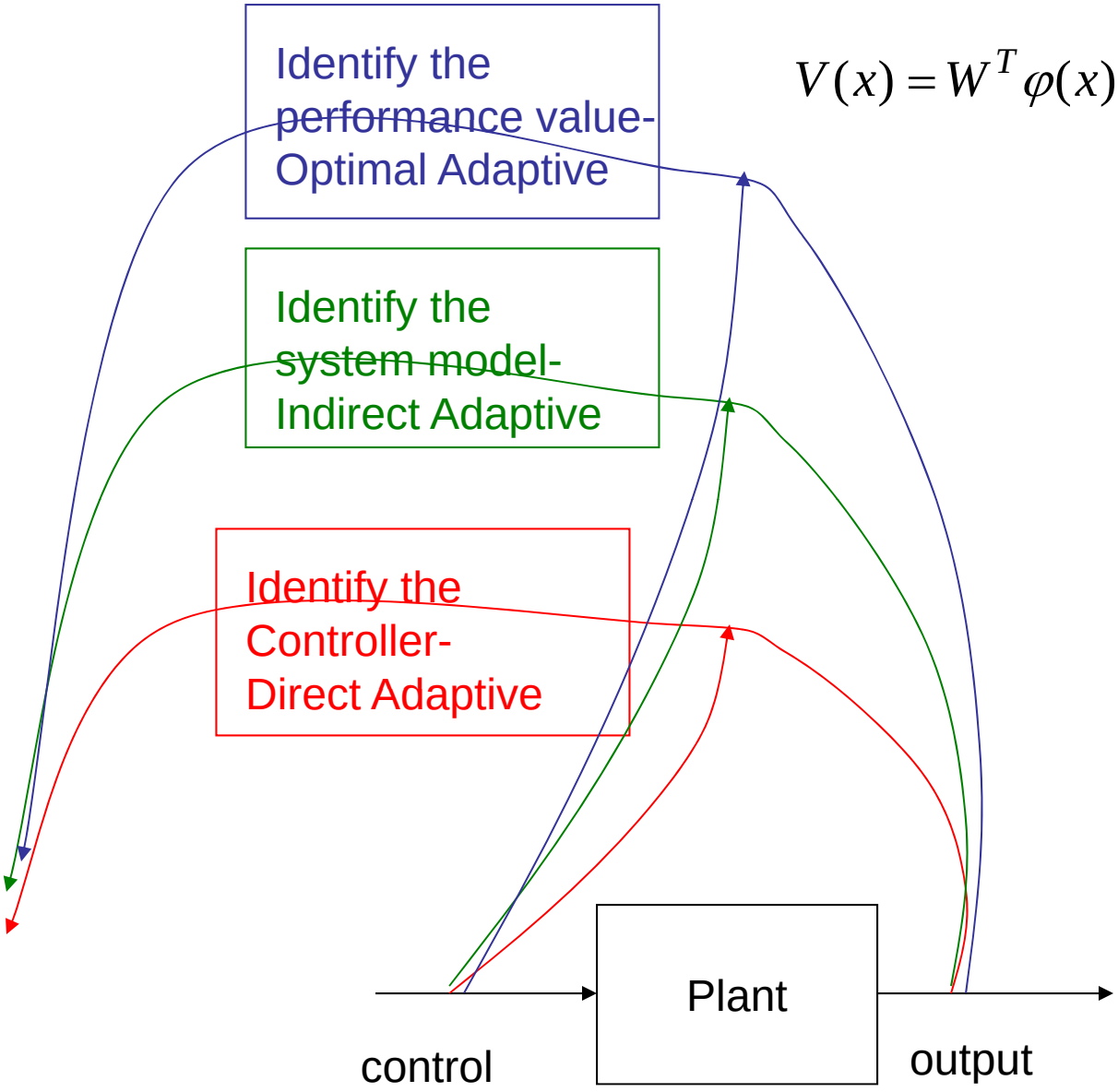
$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$



Leads to ONLINE FORWARD-IN-TIME implementation of optimal control

Optimal Adaptive Control

Adaptive Control



Greedy Value Fn. Update- Approximate Dynamic Programming
Value Iteration= Heuristic Dynamic Programming (HDP)

Paul Werbos

Policy Iteration

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Lyapunov eq.

For LQR Underlying RE

$$(A - BL_j)^T P_{j+1}(A - BL_j) - P_{j+1} = -Q - L_j^T R L_j$$

Hewer 1971

$$L_j = -(R + B^T P_j B)^{-1} B^T P_j A$$

Initial stabilizing control is needed

Value Iteration

Two occurrences of cost allows def. of greedy update

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1})$$

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Simple recursion

For LQR Underlying RE

$$P_{j+1} = (A - BL_j)^T P_j (A - BL_j) + Q + L_j^T R L_j$$

$$L_j = -(R + B^T P_j B)^{-1} B^T P_j A$$

Lancaster & Rodman proved convergence

Initial stabilizing control is NOT needed

Adaptive (Approximate) Dynamic Programming

Four ADP Methods proposed by Paul Werbos

Critic NN to approximate:

Heuristic dynamic programming

Value Iteration

Value $V(x_k)$

AD Heuristic dynamic programming
(Watkins Q Learning)

Q function $Q(x_k, u_k)$

Dual heuristic programming

Gradient $\frac{\partial V}{\partial x}$

AD Dual heuristic programming

Gradients $\frac{\partial Q}{\partial x}, \frac{\partial Q}{\partial u}$

Action NN to approximate the Control

Bertsekas- Neurodynamic Programming

Barto & Bradtke- Q-learning proof (Imposed a settling time)

A problem with DT Policy Iteration and VI

Policy Evaluation

Assume measurements of x_k and x_{k+1} are available to compute u_{k+1}

$$V_j(x_k) = W_j^T \varphi(x_k)$$

Then

$$W_{j+1}^T \left[\varphi(x_k) - \gamma\varphi(x_{k+1})\right] = r(x_k, h_j(x_k))$$

Since x_{k+1} is measured,
do not need knowledge of f(x)
or g(x) for value fn. update

Policy Improvement

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}}$$

LQR case

$$h_j(x_k) = L_j x_k = (R + B^T P_j B)^{-1} B^T P_j A x_k$$

Need to know $f(x_k)$ AND $g(x_k)$
for control update

Easy to fix – use 2 NN

Standard Neural Network VFA for On-Line Implementation

Asma Al-Tamimi

NN for Value - Critic

$$\hat{V}_i(x_k, W_{Vi}) = W_{Vi}^T \phi(x_k)$$

NN for control action

$$\hat{u}_i(x_k, W_{ui}) = W_{ui}^T \sigma(x_k)$$

(can use 2-layer NN)

HDP

$$V_{i+1}(x_k) = x_k^T Q x_k + u^T R u + V_i(x_{k+1})$$

$$x_{k+1} = f(x_k) + g(x_k)u(x_k)$$

$$u_i(x_k) = \arg\min_u (x_k^T Q x_k + u^T R u + V_i(x_{k+1}))$$

Define target cost function

$$d(\phi(x_k), W_{Vi}^T) = x_k^T Q x_k + \hat{u}_i^T(x_k) R \hat{u}_i(x_k) + \hat{V}_i(x_{k+1})$$
$$= x_k^T Q x_k + \hat{u}_i^T(x_k) R \hat{u}_i(x_k) + W_{Vi}^T \phi(x_{k+1})$$

Explicit equation for cost – use LS for Critic NN update or RLS

$$W_{Vi+1} = \arg\min_{W_{Vi+1}} \{\int_\Omega | W_{Vi+1}^T \phi(x_k) - d(\phi(x_k), W_{Vi}^T)|^2 \, dx_k\} \Longrightarrow W_{Vi+1} = \left(\int_\Omega \phi(x_k)\phi(x_k)^T dx\right)^{-1} \int_\Omega \phi(x_k) d^T(\phi(x_k), W_{Vi}^T, W_{ui}^T) dx$$

or $$W_{Vi+1}\big|_{m+1} = W_{Vi+1}\big|_m + \beta\phi^T(x_k)\left(-W_{Vi+1}^T\big|_m \phi(x_k) + r(x_k, u_k) + W_{Vi}^T \phi(x_{k+1})\right)$$

Implicit equation for DT control- use gradient descent for action update

$$W_{ui} = \arg\min_W \left(\begin{matrix} x_k^T Q x_k + \hat{u}^T(x_k, W) R \hat{u}(x_k, W) + \\ \hat{V}_i(f(x_k) + g(x_k)\hat{u}(x_k, W)) \end{matrix} \right)\Big|_\Omega \Longrightarrow W_{ui(j+1)} = W_{ui(j)} - \alpha \frac{\partial(x_k^T Q x_k + \hat{u}_{i(j)}^T R \hat{u}_{i(j)} + \hat{V}_i(x_{k+1})}{\partial W_{ui(j)}}$$

$$W_{ui}^{j+1} = W_{ui}^j - \alpha\sigma(x_k)(2R\hat{u}_{i(j)} + g(x_k)^T \frac{\partial \phi^T(x_{k+1})}{\partial x_{k+1}} W_{Vi})^T$$

Backpropagation- P. Werbos

Interesting Fact for HDP for Nonlinear systems

Linear Case $h_j(x_k) = L_j x_k = -(I + B^T P_j B)^{-1} B^T P_j A x_k$

must know system A and B matrices

NN for control action

$$\hat{u}_i(x_k, W_{ui}) = W_{ui}^T \sigma(x_k)$$

Information about A is stored in NN

Implicit equation for DT control- use gradient descent for action update

$$W_{ui} = \arg\min_{\alpha} \left(\begin{array}{l} x_k^T Q x_k + \hat{u}^T(x_k, \alpha) R \hat{u}(x_k, \alpha) + \\ \hat{V}_i(f(x_k) + g(x_k)\hat{u}(x_k, \alpha)) \end{array} \right)\Big|_{\Omega}$$

$$W_{ui(j+1)} = W_{ui(j)} - \alpha \frac{\partial (x_k^T Q x_k + \hat{u}_{i(j)}^T R \hat{u}_{i(j)} + \hat{V}_i(x_{k+1})}{\partial W_{ui(j)}}$$

$$W_{ui}^{j+1} = W_{ui}^{j} - \alpha \sigma(x_k)(2R\hat{u}_{i(j)} + g(x_k)^T \frac{\partial \phi(x_{k+1})}{\partial x_{k+1}} W_{Vi})^T$$

g(.) is needed

Note that state drift dynamics $f(x_k)$ is NOT needed since:

1. NN Approximation for action is used
2. x_{k+1} is measured in training phase

Discrete-time nonlinear HJB solution using Approximate dynamic programming : Convergence Proof

- Simulation Example 1
- Linear system – Aircraft longitudinal dynamics

$$A=\begin{bmatrix} 1.0722 & 0.0954 & 0 & -0.0541 & -0.0153 \\ 4.1534 & 1.1175 & 0 & -0.8000 & -0.1010 \\ 0.1359 & 0.0071 & 1.0 & 0.0039 & 0.0097 \\ 0 & 0 & 0 & 0.1353 & 0 \\ 0 & 0 & 0 & 0 & 0.1353 \end{bmatrix}$$

$$B=\begin{bmatrix} -0.0453 & -0.0175 \\ -1.0042 & -0.1131 \\ 0.0075 & 0.0134 \\ 0.8647 & 0 \\ 0 & 0.8647 \end{bmatrix}$$

Unstable, Two-input system

- The HJB, i.e. ARE, Solution

$$P=\begin{bmatrix} 55.8348 & 7.6670 & 16.0470 & -4.6754 & -0.7265 \\ 7.6670 & 2.3168 & 1.4987 & -0.8309 & -0.1215 \\ 16.0470 & 1.4987 & 25.3586 & -0.6709 & 0.0464 \\ -4.6754 & -0.8309 & -0.6709 & 1.5394 & 0.0782 \\ -0.7265 & -0.1215 & 0.0464 & 0.0782 & 1.0240 \end{bmatrix}$$

$$L=\begin{bmatrix} -4.1136 & -0.7170 & -0.3847 & 0.5277 & 0.0707 \\ -0.6315 & -0.1003 & 0.1236 & 0.0653 & 0.0798 \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming : Convergence Proof

- Simulation
- The Cost function approximation – quadratic basis set

$$\hat{V}_{i+1}(x_k, W_{Vi+1}) = W_{Vi+1}^T \phi(x_k)$$

$$\phi^T(x) = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 & x_1x_4 & x_1x_5 & x_2^2 & x_2x_3 & x_4x_2 & x_2x_5 & x_3^2 & x_3x_4 & x_3x_5 & x_4^2 & x_4x_5 & x_5^2 \end{bmatrix}$$

$$W_V^T = \begin{bmatrix} w_{V1} & w_{V2} & w_{V3} & w_{V4} & w_{V5} & w_{V6} & w_{V7} & w_{V8} & w_{V9} & w_{V10} & w_{V11} & w_{V12} & w_{V13} & w_{V14} & w_{V15} \end{bmatrix}$$

- The Policy approximation – linear basis set

$$\hat{u}_i = W_{ui}^T \sigma(x_k)$$

$$\sigma^T(x) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$$

$$W_u^T = \begin{bmatrix} w_{u11} & w_{u12} & w_{u13} & w_{u14} & w_{u15} \\ w_{u21} & w_{u22} & w_{u23} & w_{u24} & w_{u25} \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming : Convergence Proof

- Simulation

The convergence of the cost

$$W_V^T = [55.5411 \quad 15.2789 \quad 31.3032 \quad -9.3255 \quad -1.4536 \quad 2.3142 \quad 2.9234 \quad -1.6594 \quad -0.2430$$

$$24.8262 \quad -1.3076 \quad 0.0920 \quad 1.5388 \quad 0.1564 \quad 1.0240]$$

$$\begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} \end{bmatrix} = \begin{bmatrix} w_{V1} & 0.5w_{V2} & 0.5w_{V3} & 0.5w_{V4} & 0.5w_{V5} \\ 0.5w_{V2} & w_{V6} & 0.5w_{V7} & 0.5w_{V8} & 0.5w_{V9} \\ 0.5w_{V3} & 0.5w_{V7} & w_{V10} & 0.5w_{V11} & 0.5w_{V12} \\ 0.5w_{V4} & 0.5w_{V8} & 0.5w_{V11} & w_{V13} & 0.5w_{V14} \\ 0.5w_{V5} & 0.5w_{V9} & 0.5w_{V12} & 0.5w_{V14} & w_{V15} \end{bmatrix}$$

Actual ARE soln:

$$P = \begin{bmatrix} 55.8348 & 7.6670 & 16.0470 & -4.6754 & -0.7265 \\ 7.6670 & 2.3168 & 1.4987 & -0.8309 & -0.1215 \\ 16.0470 & 1.4987 & 25.3586 & -0.6709 & 0.0464 \\ -4.6754 & -0.8309 & -0.6709 & 1.5394 & 0.0782 \\ -0.7265 & -0.1215 & 0.0464 & 0.0782 & 1.0240 \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming : Convergence Proof

- Simulation

  The convergence of the control policy

$$W_u = \begin{bmatrix} 4.1068 & 0.7164 & 0.3756 & -0.5274 & -0.0707 \\ 0.6330 & 0.1005 & -0.1216 & -0.0653 & -0.0798 \end{bmatrix}$$

$$\begin{bmatrix} L_{11} & L_{12} & L_{13} & L_{14} & L_{15} \\ L_{21} & L_{22} & L_{23} & L_{24} & L_{25} \end{bmatrix} = -\begin{bmatrix} w_{u11} & w_{u12} & w_{u13} & w_{u14} & w_{u15} \\ w_{u21} & w_{u22} & w_{u23} & w_{u24} & w_{u25} \end{bmatrix}$$

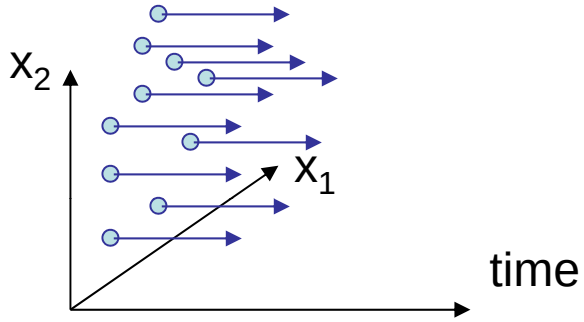
Actual optimal ctrl. $L = \begin{bmatrix} -4.1136 & -0.7170 & -0.3847 & 0.5277 & 0.0707 \\ -0.6315 & -0.1003 & 0.1236 & 0.0653 & 0.0798 \end{bmatrix}$

Note- In this example, drift dynamics matrix A is NOT Needed. Riccati equation solved online without knowing A matrix

Issues with Nonlinear ADP

LS solution for Critic NN update

$$W_{Vi+1} = \left(\int_{\Omega} \phi(x_k)\phi(x_k)^T dx \right)^{-1} \int_{\Omega} \phi(x_k) d^T(\phi(x_k), W_{Vi}^T, W_{ui}^T) dx$$

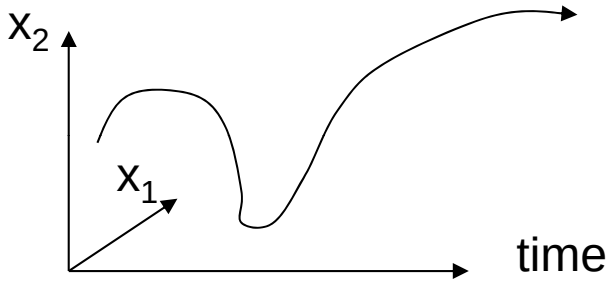


Integral over a region of state-space
Approximate using a set of points

Batch LS

Selection of NN Training Set

$$W_{Vi+1}\big|_{m+1} = W_{Vi+1}\big|_{m} + \beta\phi^T(x_k)\left(-W_{Vi+1}^T\big|_{m}\phi(x_k) + r(x_k, u_k) + W_{Vi}^T\phi(x_{k+1})\right)$$



Take sample points along a single trajectory

Recursive Least-Squares RLS

Set of points over a region vs. points along a trajectory

For Linear systems- these are the same under PE condition

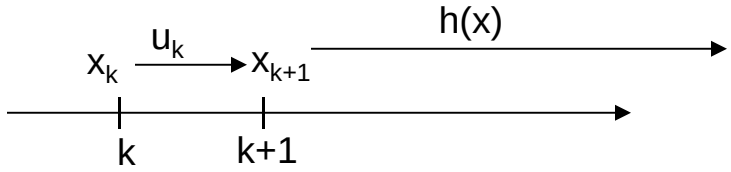
Exploitation (optimal regulation) vs Exploration

PE allows local smooth solution of Bellman eq.

Q Learning - Action Dependent ADP

Value function recursion for given policy $h(x_k)$

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$



Define Q function

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma V_h(x_{k+1})$$

u_k arbitrary

policy $h(.)$ used after time k

Note $Q_h(x_k, h(x_k)) = V_h(x_k)$

Recursion for Q $Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$

Simple expression of Bellman's principle

$$V^*(x_k) = \min_{u_k}(Q^*(x_k, u_k)) \qquad h^*(x_k) = \arg\min_{u_k}(Q^*(x_k, u_k))$$

Optimal Adaptive Control for completely unknown DT systems

Q Function Definition

Specify a control policy $u_j = h(x_j); \quad j = k, k+1, \ldots.$

Define Q function

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma V_h(x_{k+1})$$

- u_k arbitrary
- policy *h(.)* used after time k

Note $Q_h(x_k, h(x_k)) = V_h(x_k)$

Bellman equation for Q $Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$

Optimal Q function $Q^*(x_k, u_k) = r(x_k, u_k) + \gamma V^*(x_{k+1}))$

$$Q^*(x_k, u_k) = r(x_k, u_k) + \gamma Q^*(x_{k+1}, h^*(x_{k+1}))$$

Optimal control solution

$$V^*(x_k) = Q^*(x_k, h^*(x_k)) = \min_h(Q_h(x_k, h(x_k))) \qquad h*(x_k) = \arg\min_h(Q_h(x_k, h(x_k))$$

Simple expression of Bellman's principle

$$V^*(x_k) = \min_{u_k}(Q^*(x_k, u_k)) \qquad h*(x_k) = \arg\min_{u_k}(Q^*(x_k, u_k))$$

Q Function HDP – Action Dependent HDP

Q function for any given control policy $h(x_k)$ satisfies the Bellman equation

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$$

Recursive solution

Pick stabilizing initial control policy

Find Q function

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_j(x_{k+1}, h_j(x_{k+1}))$$

Update control

$$h_{j+1}(x_k) = \arg\min_{u_k}(Q_{j+1}(x_k, u_k))$$

Now $f(x_k, u_k)$ not needed

Bradtke & Barto (1994) proved convergence for LQR

Q Learning does not need to know $f(x_k)$ or $g(x_k)$

For LQR $V(x) = W^T \varphi(x) = x^T P x$ V is quadratic in x

---

$$Q_h(x_k, u_k) = r(x_k, u_k) + V_h(x_{k+1})$$
$$= x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P(Ax_k + Bu_k)$$

$$= \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q + A^T PA & A^T PB \\ B^T PA & R + B^T PB \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \equiv \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}$$

Q is quadratic in x and u

Control update is found by $0 = \frac{\partial Q}{\partial u_k} = 2[B^T PAx_k + (R + B^T PB)u_k] = 2[H_{ux}x_k + H_{uu}u_k]$

so $u_k = -(R + B^T PB)^{-1} B^T PAx_k = -H_{uu}^{-1} H_{ux} x_k = L_{j+1} x_k$

Control found only from Q function
A and B not needed

Implementation- DT Q Function Policy Iteration

Bradtke and Barto

For LQR

Q function update for control $u_k = L_j x_k$ is given by

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_{j+1}(x_{k+1}, L_j x_{k+1})$$

Assume measurements of u_k, x_k and x_{k+1} are available to compute u_{k+1}

QFA – Q Fn. Approximation

$$Q(x,u) = W^T \varphi(x,u)$$ Now u is an input to the NN- Werbos- Action dependent NN

Then

regression matrix

$$W_{j+1}^T \left[\varphi(x_k, u_k) - \gamma \varphi(x_{k+1}, L_j x_{k+1})\right] = r(x_k, L_j x_k)$$

Since x_{k+1} is measured in training phase, do not need knowledge of f(x) or g(x) for value fn. update

Solve for weights using RLS or backprop.

For LQR case

$$\varphi(x) = \left[x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_n^2\right]'.$$

Model-free policy iteration

Q Policy Iteration

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_{j+1}(x_{k+1}, L_j x_{k+1})$$

Bradtke, Ydstie, Barto

$$W_{j+1}^T \left[\varphi(x_k, u_k) - \gamma \varphi(x_{k+1}, L_j x_{k+1}) \right] = r(x_k, L_j x_k)$$

Control policy update

Stable initial control needed

$$h_{j+1}(x_k) = \arg\min_{u_k}(Q_{j+1}(x_k, u_k))$$

$$u_k = -H_{uu}^{-1} H_{ux} x_k = L_{j+1} x_k$$

Greedy Q Fn. Update - Approximate Dynamic Programming

ADP Method 3. Q Learning

Action-Dependent Heuristic Dynamic Programming (ADHDP)

Paul Werbos

Greedy Q Update

Model-free HDP

Stable initial control NOT needed

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_j(x_{k+1}, h_j(x_{k+1}))$$

$$W_{j+1}^T \varphi(x_k, u_k) = r(x_k, L_j x_k) + W_j^T \gamma \varphi(x_{k+1}, L_j x_{k+1}) \equiv \text{target}_{j+1}$$

Update weights by RLS or backprop.

Q learning actually solves the Riccati Equation
WITHOUT knowing the plant dynamics

Model-free ADP

Direct OPTIMAL ADAPTIVE CONTROL

Works for Nonlinear Systems

Proofs?
Robustness?
Comparison with adaptive control methods?