L5: Relaxed dynamic programming and Q-learning

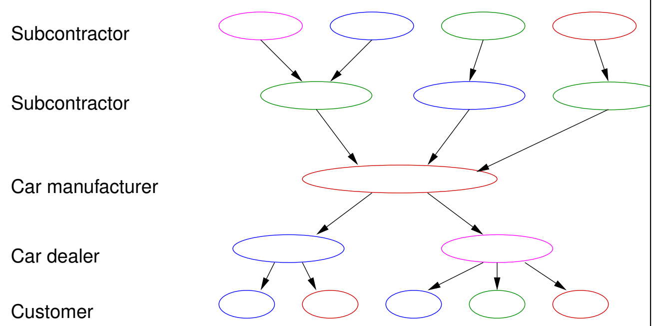
- **Relaxed Dynamic Programming**
  - Application to switching systems
  - Application to Model Predictive Control

Literature:

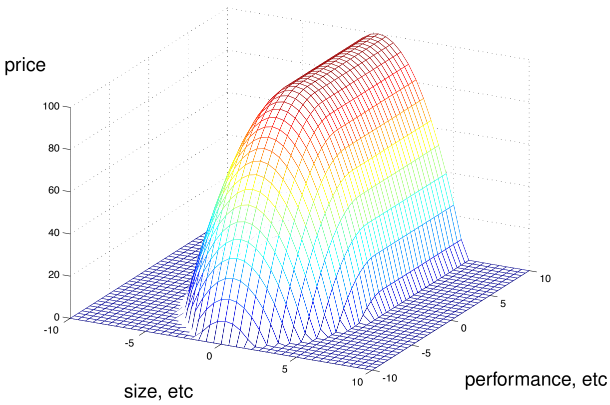
[Lincoln and Rantzer, *Relaxing Dynamic Programming*, TAC 51:8, 2006]

[Rantzer, *Relaxing Dynamic Programming in Switching Systems*, IEE Proceeding on Control Theory and Applications, 153:5, 2006]

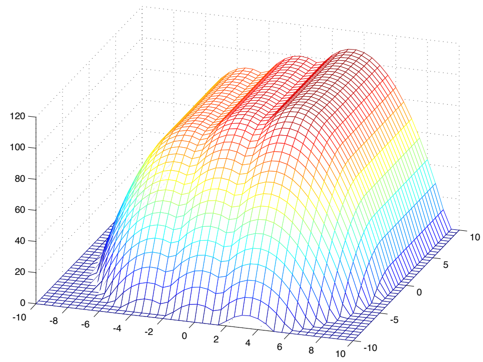
Who decides the price of a Volvo?



Valuation by the customer



Valuation by the car dealer



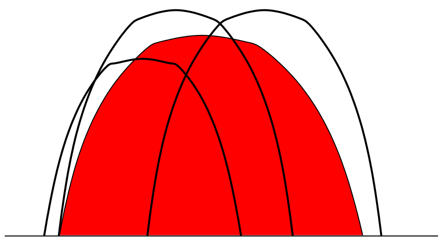
Customers: Andersson, Pettersson and Lundström

The key: Simplified valuation

Exact value-iteration gives absurd complexity.

Every subcontractor of Volvo would have to modify his prices when Andersson expands his garage.

Of course, pricing is not done like that.
Approximations are done in every step.



Dynamic Programming in Discrete Time

$$\text{Minimize} \quad \sum_{t=0}^{\infty} g(x(t), u(t))$$

$$\text{subject to} \quad x(t+1) = f(x(t), u(t)) \qquad x(0) = x_0$$

Let $J^*(x_0)$ denote the minimal value. The value function J^* satisfies the *Bellman equation*

$$J^*(x) = \min_u \left[g(x,u) + J^*(f(x,u)) \right]$$

If $J(x) \le \min_u \left[J(f(x,u)) + g(x,u) \right]$, then J is a lower bound on the optimal cost.

Conversely, if $\min_u \left[J(f(x,u)) + g(x,u) \right] \le J(x)$ then J is an upper bound on the optimal cost.

Relaxed Value Iteration

Replace the Bellman equation by an inequality:

$$\min_u \left[J(f(x,u)) + g(x,u)/\alpha \right] \le J(x) \le \min_u \left[J(f(x,u)) + \alpha g(x,u) \right]$$

where $\alpha > 1$.

From the inequalities, it follows that

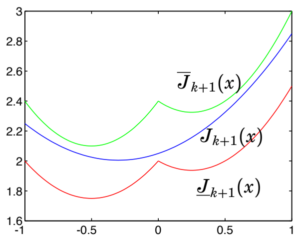
$$J^*(x)/\alpha \le J(x) \le \alpha J^*(x)$$

The recursive conditions become

$$\min_u \left[J_k(f(x,u)) + g(x,u)/\alpha \right] \quad \le J_{k+1}(x) \le \min_u \left[J_k(f(x,u)) + \alpha g(x,u) \right]$$

The interval for $J_{k+1}(x)$ makes it possible to work with a simplified parameterization of J_k.

Relaxed Dynamic Programming



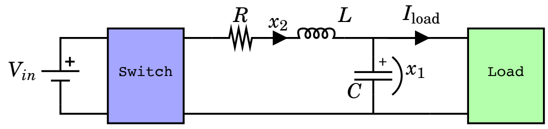
$$\underbrace{\min_u \Big\{ J_k(f(x,u)) + g(x,u)/\alpha \Big\}}_{\underline{J}_{k+1}(x)} \le J_{k+1}(x) \le \underbrace{\min_u \Big\{ J_k(f(x,u)) + \alpha g(x,u) \Big\}}_{\overline{J}_{k+1}(x)}$$

L5: Relaxed dynamic programming and Q-learning

- Relaxed Dynamic Programming
- **Application to switching systems**
- Application to Model Predictive Control

Example: Switched voltage converter

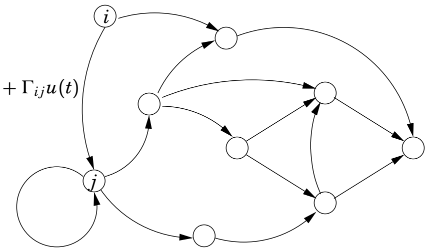
A step-down DC/DC converter.



- A linear system except for the switching actuator
- Objective: Keep output voltage constant.

Optimize switches for continuous dynamics

$x(t+1) = \Phi_{ij}x(t) + \Gamma_{ij}u(t)$

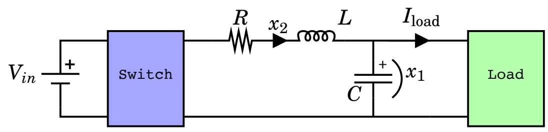


$x(t+1) = \Phi_{jj}x(t) + \Gamma_{jj}u(t)$

Minimize $\sum_t x(t)^T Q_{i(t)j(t)} x(t) + u(t)^T R_{i(t)j(t)} u(t)$

Two types of inputs, both affect the penalty

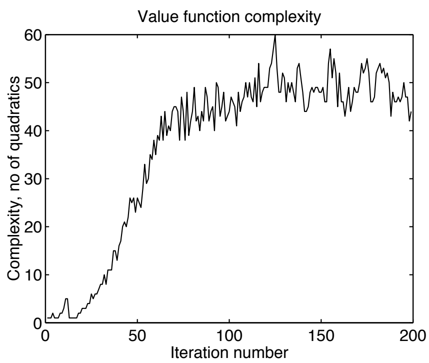
Example: Switched voltage converter



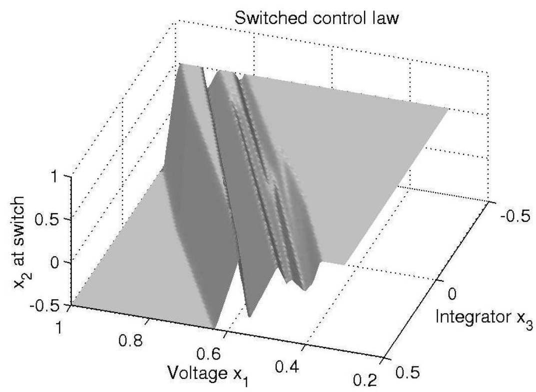
$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{C}\left(x_2 - I_{\text{load}}\right) \\ -\frac{1}{L}x_1 - \frac{R}{L}x_2 + \frac{1}{L}s(t)V_{\text{in}} \\ V_{\text{ref}} - x_1 \end{bmatrix}$$

$$g(x) = q_P(x_1 - V_{\text{ref}})^2 + q_I x_3^2 + q_D(x_2 - I_{\text{load}})^2$$

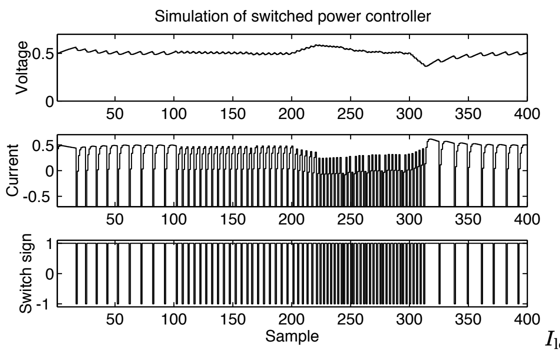
Example: Switched voltage converter



Example: Switched voltage converter



Example: Switched voltage converter

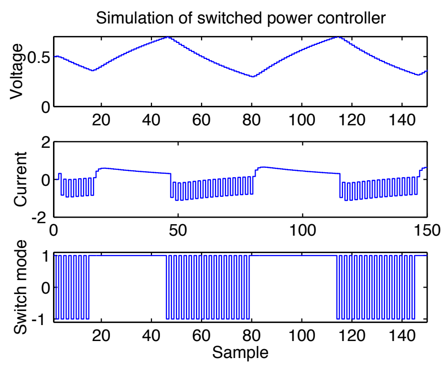


$I_{\text{load}} =$

$\{0.3A \quad 0.1A \quad -0.2A \quad 0.3A\}$

Example: Switched voltage converter

Frequency weights in the cost function can be used to suppress undesired harmonics. This increases state dimension, but has no significant effect on computational complexity.



More on Control of DC-DC Converters

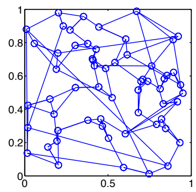
1126 IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, VOL. 18, NO. 5, SEPTEMBER 2010

Comparison of Hybrid Control Techniques for Buck and Boost DC-DC Converters

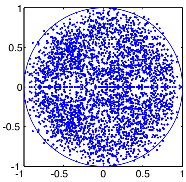
Sébastien Mariéthoz, *Member, IEEE*, Stefan Almér, Mihai Bâja, Andrea Giovanni Beccuti, Diego Patino, Andreas Wernrud, Jean Buisson, Hervé Cormerais, Tobias Geyer, *Member, IEEE*, Hisaya Fujioka, *Member, IEEE*, Ulf T. Jönsson, *Member, IEEE*, Chung-Yao Kao, *Member, IEEE*, Manfred Morari, *Fellow, IEEE*, Georgios Papafotiou, *Member, IEEE*, Anders Rantzer, *Fellow, IEEE*, and Pierre Riedinger

Optimal control: 60 discrete states, 30 continuous

120 edges



120 × 30 eigenvalues



Minimize

$\sum_t z(t)^T Q_{i(t)u(t)} z(t)$

Continuous dynamics: $z(t+1) = A_{i(t)u(t)} z(t) \quad z(0) = z_0 \in \mathbb{R}^{30}$

Discrete jumps: $i(t+1) = u(t) \quad i(0) = i_0$

Four iterations give $P^1, \dots, P^{120} \in \mathbf{R}^{30\times 30}$ such that the following switch rule is within a factor 3.81 from optimality:

From node i, jump to node $n = \arg\min_n \left(z^T [A_{in}^T P^n A_{in} + Q_{in}] z\right)$

Two versions of relaxed value iteration

$$\min_u [J_k(f(x,u)) + g(x,u)/\alpha] \quad \le J_{k+1}(x) \le \min_u [J_k(f(x,u)) + g(x,u)]$$

Decentralized computations!

$$\min_u [J_k(f(x,u)) + g(x,u)/\alpha] \quad \le J_{k+1}(x) \le \min_u [J_{k+1}(f(x,u)) + g(x,u)]$$

Global convergence!

If simple approximation exists, we will find one!

Assume J^{S} is "simple" and satisfies

$$\min_u [J^*(f(x,u)) + g(x,u)/\alpha] \quad \le J^{\mathrm{S}}(x) \le \min_u \left[J^{\mathrm{S}}(f(x,u)) + g(x,u)\right]$$

Then $J^*/\alpha < J^{\mathrm{S}} < J^*$ and the following relaxed value iteration with $J_0 = 0$ is feasible in every step:

$$\min_u [J_k(f(x,u)) + g(x,u)/\alpha] \quad \le J_{k+1}(x) \le \min_u [J_{k+1}(f(x,u)) + g(x,u)]$$

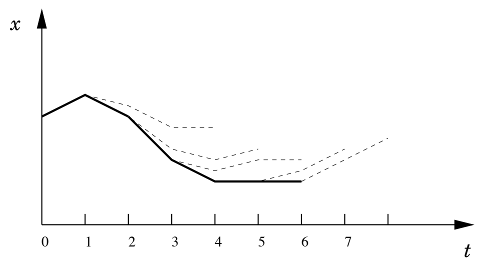
Moreover

$$J^*(x)/\alpha < \limsup_{k\to\infty} J_k(x) < J^*(x)$$

L5: Relaxed dynamic programming and Q-learning

- Relaxed Dynamic Programming
- Application to switching systems
- **Application to Model Predictive Control**

Model Predicitive Control (Receding Horizon Control)



At time t:

1. Measure the state $x(t)$
2. Use model to optimize trajectory for $t+1, \dots, t+N$
3. Apply the optimization result $u(t)$ to the system
4. After one sample, go to 1 to repeat the procedure

The MPC Control Law

Consider

$$J_N(x_0) = \inf_{u,x} \sum_{t=0}^{N-1} g(x(t), u(t))$$

where infimum is taken over $x(t) \in X$, $u(t) \in U$ satisfying $x(t+1) = f(x(t), u(t))$ and $x(0) = x_0$.

The MPC control law

$$\mu_N(x) := \arg\min_u \{J_{N-1}(f(x,u)) + g(x,u)\}$$

gives the cost

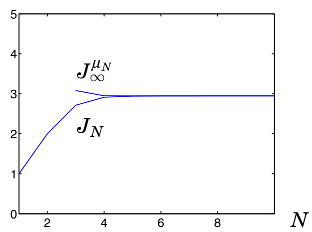
$$J_\infty^{\mu_N}(x_0) = \sum_{t=0}^{\infty} g(x_{\mu_N}(t), \mu_N(x_{\mu_N}(t)))$$

Notice that $J_1 \le J_2 \le \ldots \le J_N \le \ldots \le J_\infty \le J_\infty^{\mu_N}$

Example 1 — Double Integrator

$$J_N(x_0) = \inf_{u,x} \sum_{t=0}^{N-1} (|x(t)|^2 + u(t)^2)$$

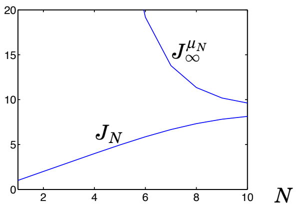
$$x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad x(0) = x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



Example 1 — Double Integrator

$$J_N(x_0) = \inf_{u,x} \sum_{t=0}^{N-1} (|x(t)|^2 + 1000u(t)^2)$$

$$x(t+1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad x(0) = x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



Longer horizon required. Why?

Long horizon need not help!

For the system

$$\begin{cases} x_1(t+1) = u(t) \\ x_2(t+1) = -2x_1(t) + u(t) \end{cases}$$

the cost function

$$\sum_{t=0}^{N-1} x_2(t)^2$$

is minimized by the control law $u(t) = 2x_1(t)$, which gives the unstable dynamcs

$$x_1(t+1) = 2x_1(t)$$

The transfer function from u to x_2 has an unstable zero at $z = 2$!

Major Issues of MPC Theory

- Can we guarantee stability?
- Can we guarantee performance?
- What prediction horizon is needed?

MPC with Terminal cost

Assume that

$$W(f(x, \mu(x)) + g(x, \mu(x)) \le W(x) \qquad \text{for all } x$$

Define the MPC control law μ_N using the minimization

$$\overline{J}_N(x_0) = \inf_{u,x} \left[\sum_{t=0}^{N-1} g(x(t), u(t)) \underbrace{+W(x(N))}_{\textit{terminal cost}} \right]$$

with $x(t) \in X$, $u(t) \in U$, $x(t+1) = f(x(t), u(t))$, $x(0) = x_0$.

Then μ_N is stabilizing and $J_\infty \le J_\infty^{\mu_N} \le \overline{J}_N \le \ldots \le \overline{J}_2 \le \overline{J}_1$.

Terminal cost and terminal constraint

Assume existence of a function $W(x) \ge 0$, a control law $u = \mu(x)$ and a number $\epsilon > 0$ such that $W(x) \le \epsilon \Rightarrow W(f(x, \mu(x)) + g(x, \mu(x)) \le W(x)$.

Define the MPC control law μ_N using the minimization

$$\overline{J}_N(x_0) = \inf_{u,x} \left[\sum_{t=0}^{N-1} g(x(t), u(t)) \underbrace{+W(x(N))}_{\textit{terminal cost}} \right]$$

subject to $x(t) \in X$, $u(t) \in U$, $x(t+1) = f(x(t), u(t))$, $x(0) = x_0$ and the *terminal constraint* $W(x) \le \epsilon$.

Then μ_N is stabilizing and $J_\infty \le J_\infty^{\mu_N} \le \overline{J}_N \le \ldots \le \overline{J}_2 \le \overline{J}_1$.

When is MPC Stabilizing Without Terminal Cost?

Consider

$$J_N(x_0) = \inf_{u,x} \sum_{t=0}^{N-1} g(x(t), u(t))$$

where infimum is taken over $x(t) \in X$, $u(t) \in U$ satisfying $x(t+1) = f(x(t), u(t))$ and $x(0) = x_0$. The MPC control law

$$\mu_N(x) := \arg\min_u \{J_{N-1}(f(x, u)) + g(x, u)\}$$

gives

$$J_N(x) = g(x, \mu_N(x)) + J_{N-1}(f(x, \mu_N(x)))$$

so J_N is a Lyapunov function provided that the right hand side is bigger than $J_N(f(x, \mu_N(x)))$!

Exponential stabilizability

Suppose there exist numbers $C > 0$ and $\sigma \in (0, 1)$ such that for every $x_0 \in X$ there exists a sequence $u(0), u(1), \ldots \in U$ with

$$g(x(t), u(t)) \le C\sigma^t g^*(x_0) \qquad \text{for all } t \ge 0$$

where $g^*(x_0) = \min_v g(x_0, v)$. This can be viewed as a condition of exponential stabilizability.

Then the MPC control law $\mu_N(x)$ is stabilizing provided that

$$N \ge 2\gamma \ln \gamma$$

where $\gamma = \frac{C}{1-\sigma}$.

[Grüne and Rantzer, TAC 53:9, 2009, Proposition 4.7]

Dynamic Programming versus MPC

- Dynamic Programming (Explicit MPC)
  - Corresponds to MPC with $N = 2$ and accurate terminal cost
  - Heavy off-line computations and memory requirements
  - Extremely fast on-line
- Model Predictive Control
  - No off-line computations
  - Heavy on-line computations
  - Wide range of industrial applications exist

Reinforcment Learning

Q-learning, SARSA, Dual Control

Bo Bernhardsson

based on Gabriel Ingesson's presentation in DL course and "A Tutorial on Linear Function Approximators for DP and RL", Geramifard et al (MIT)

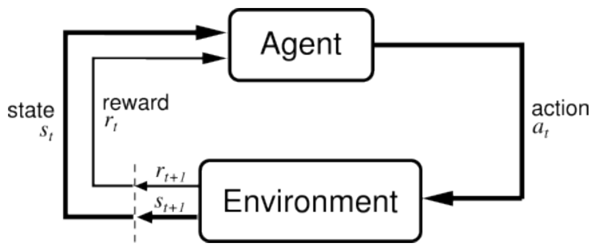
Notation - Markov Decision Process (MDP)

A finite Markov Decision Process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ where

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{P}$ is a transition probability matrix: $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}^a_{ss'} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s']$
- γ is a discount factor $\gamma \in [0, 1)$.

The core problem of MDPs is to find a policy for the agent, that maximizes return given the MDP.

Reinforcement Learning



- Initially, the agent does not have to know anything about the environment.
- The agent recieves a reward signal and the environment state.
- Adjusts its actions in order to maximize the cumulative reward.

Examples

- Pancake Robot
- Atari Game

State-Value Function, $V^\pi(s)$

Evaluates a state, given a policy π.

The state-value function $V^\pi(s)$, is a prediction of the discounted return given a policy and the current state S_t:

$$V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s]$$

is used to evaluate a state and helps to select actions.

Action-Value Function, $Q^\pi(s,a)$

Evaluate an alternative action, given a policy π.

The action-value function $Q^\pi(s,a)$ is the expected return starting from state s, taking a, and then following policy π

$$Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, A_0 = a].$$

is used to evaluate actions and helps to update the policy.

DP - Bellman Equation

The value functions can be decomposed into immediate reward plus discounted value of successor state

A recursive relationship ($(')$ denotes subsequent state/action):

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \mathcal{P}^a_{ss'} \sum_{s' \in \mathcal{S}} (\mathcal{R}^a_{ss'} + \gamma V^\pi(s'))$$
$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \sum_{a' \in \mathcal{A}} (\mathcal{R}^a_{ss'} + \gamma Q^\pi(s',a'))$$

Note that if $\mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'}$ and π are known, these are linear equations systems.

Often too large though.

Optimal state-value function

The optimal value functions is the maximum value function over all policies:

- $V^*(s) = \max_\pi V^\pi(s)$
- $Q^*(s,a) = \max_\pi Q^\pi(s,a)$

Bellman Optimality Equation

$$V^*(s) = \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}(\mathcal{R}^a_{ss'} + \gamma V^*(s')) = \max_a Q^*(s,a)$$

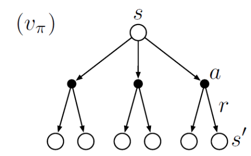
$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a'))$$

- Non-linear system of equations, high computational cost
- Approximate solutions
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Iterative Policy Evaluation & Improvement

1. Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.

2. Policy Evaluation:
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta \leq \epsilon$



3. Policy Improvement:
policy-stable $\leftarrow$ true
For each $s \in \mathcal{S}$:
    old action $\leftarrow$ argmax$_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    If old action $\neq \pi(s)$, then policy-stable $\leftarrow$ false
If policy-stable, then stop and return $V \approx V^*$ and $\pi \approx \pi_*$ else go to 2

Monte-Carlo (”Trajectory Based”) Reinforcement Learning

- Model free, no prior knowledge about the environment.
- Monte Carlo methods require only experience, i.e. sample sequences of states, actions, and rewards from interaction with the environment.
- Learns from complete episodes, updates policy from computed return

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve large problems, e.g.

- Backgammon 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

So far we have represented the value functions as lookup table, this becomes slow and memory expensive for large problems.

A solution is to estimate the value function with function approximation:

$$V^\pi(s) \approx \tilde{V}(s,\theta)$$
$$Q^\pi(s,a) \approx \tilde{Q}(s,a,\theta)$$

Update the parameter θ from trajectories

Function Approximators

Examples of approximators:

- Linear, $\tilde{V}(s,\theta) = \sum_i \theta_i \phi(s), \quad \tilde{Q}(s,a,\theta) = \sum_i \theta_i \phi(s,a),$
- Nonlinear, neural networks

Objectives to minimize:

$$C_V(\theta) = \sum_s \text{weight}(s)[V^\pi(s) - \tilde{V}(s,\theta)]^2$$
$$C_Q(\theta) = \sum_{s,a} \text{weight}(s,a)[Q^\pi(s,a) - \tilde{Q}(s,a,\theta)]^2$$

Learn from experience

Back-propagation, stochastic gradient descent

Exploitation vs. Exploration

How do we get experience? What (s,a) to visit ?

Example

$$\pi_Q^\epsilon(s) = \begin{cases} \text{argmax}_{a\in A} Q(s,a) & \text{w.p. } 1-\epsilon \\ \text{random } a & \text{w.p. } \epsilon \end{cases}$$

Q-learning

Use some fixed policy π (i.e. $\pi^\epsilon(s)$) to generate samples (s,a,r,s')

Initialize θ, s and $a := \pi(s)$

While time left repeat

In state s take action a, receive reward r and next state s'

$Q^+(s,a) := r + \gamma \max_{a'} Q(s',a')$

$\theta := \theta - \alpha \frac{\partial [Q^+(s,a) - \tilde{Q}(s,a,\theta)]^2}{\partial \theta}$

$(s,a) := (s', \pi(s'))$

Return π^{new} greedy w.r.t. $\tilde{Q}$

$Q^+(s,a) - \tilde{Q}(s,a,\theta)$ is called the **temporal difference (TD)** error

Sometimes works well. Examples with convergence issues

SARSA

Update the policy π when you learn $\tilde{Q}$

While time left repeat

In state s take action a, receive reward r and next state s'

$a' := \pi^\epsilon(s')$

$Q^+(s,a) := r + \gamma Q(s',a')$

$\theta := \theta - \alpha \frac{\partial [Q^+(s,a) - \tilde{Q}(s,a,\theta)]^2}{\partial \theta}$

$(s,a) := (s',a')$

Return π^{new} greedy w.r.t. $\tilde{Q}$

Less convergence issues.

Also called *on-policy* learning

Sarsa: TD control algorithm

State-Action-Reward-State-Action (SARSA)

$Q(S,A)$ depends on (S,A,R',S',A')

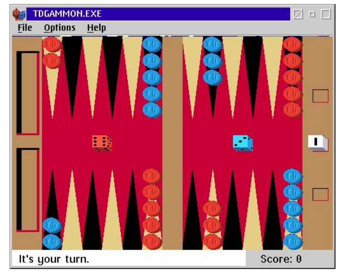
Policy evaluation, $Q \approx q_\pi$:

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R' + \gamma Q(S',A') - Q(S,A))$$

Policy improvement is then chosen ϵ- greedy w.r.t. $Q(S,A)$.

TD-Gammon

- Used a multi-layer artificial neural network trained by TD(λ) to evaluate each possible move.
- Achieved a level of play just slightly below that of the top human backgammon player in 1992.
- Found new strategies.



Playground: OpenAI Gym

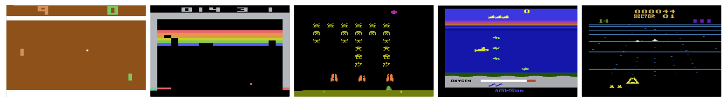
OpenAI Gym:

- A toolkit for developing and comparing reinforcement-learning algorithms.
- From simulated robots to Atari games.
- A site for comparing and reproducing results.

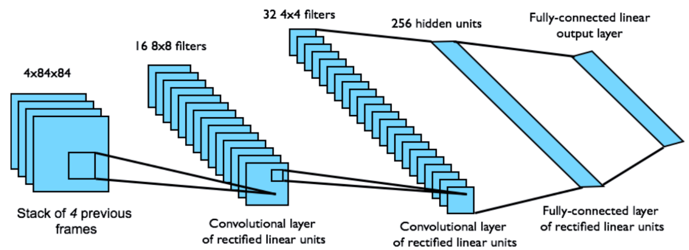
miniproject, if interested in neural networks

- Run a RL algorithm on one of the OpenAI-gym examples
- Start with trying an already working implementation:
  - Pong
  - Breakout
  - Space Invaders

Google Deepmind's Deep Q-Network (DQN)



Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." (2013)



David Silver's presentation on function approximation

Dual control and a research challenge

Efficient exploration and learning of uncertain control systems

Åström, Bohlin, Sternby, . . .

Example: "Dual Control of a first order system with two possible gains", Bernhardsson (1989), Int. Jour. of Adaptive Control and Signal Processing

$$x_{t+1} = ax_t + bu_t + e_t, \qquad e_t \in N(0, \sigma)$$

Say a and σ are known but b is unknown, either $b = 1$ or $b = -1$

$\mathrm{Prob}(b = 1) = p_0, \quad \mathrm{Prob}(b = -1) = 1 - p_0,$

Find policy $u_t = \pi(x_{[0,t]})$ that minimizes expected loss (horizon N)

$$E_\pi \sum_{t=1}^{N} |x(t)|^2$$

Dual Control - example continued

$N = 1$: If $p = 0.5$ then $u^*(x) = 0$ is unique optimal solution

Cautious control, never learns

$N > 1$: Probing occurs: $u^*(x) \neq 0$ even if $x = 0$ (if $p \approx 0.5$)

Dual controller that probes the system to actively learn b.

Dual Control - example continued

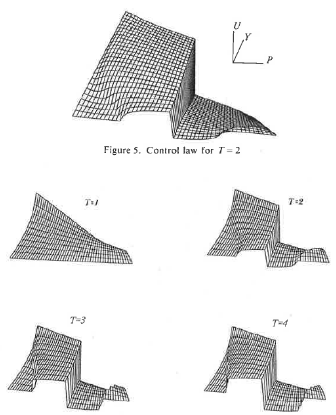


Figure 5. Control law for $T = 2$

$N > 1$: Probing occurs: $u^*(x) \neq 0$ even if $x = 0$ (if $p \approx 0.5$)

Dual controller that probes the system to actively learn b.

Dual Control - Learning Systems

Represent also the uncertainty about the system

Hyperstate - a probability function of state and parameters

Can we represent the hyperstate efficiently using recent progress in ML, DL, MCMC . . .