Introduction to Reinforcement Learning

Rémi Munos

SequeL project: Sequential Learning
http://researchers.lille.inria.fr/∼munos/

INRIA Lille - Nord Europe

Machine Learning Summer School, September 2011, Bordeaux

Outline of the course

- Part 1: Introduction to Reinforcement Learning and Dynamic Programming
  - Settting, examples
  - Dynamic programming: value iteration, policy iteration
  - RL algorithms: TD(λ), Q-learning.
- Part 2: Approximate dynamic programming
  - L_∞- performance bounds
  - Sample-based algorithms: Least Squares TD, Bellman Residual, Fitted-VI
- Part 3: Exploration-Exploitation tradeoffs
  - The stochastic bandit: UCB
  - The adversarial bandit: EXP3
  - Populations of bandits: Tree search, Nash equilibrium.
  - Applications to games (Go, Poker) and planning.

Part 1: Introduction to Reinforcement Learning and Dynamic Programming

- Settting, examples
- Dynamic programming: value iteration, policy iteration
- RL algorithms: TD(λ), Q-learning.

General references:

- *Neuro Dynamic Programming*, Bertsekas et Tsitsiklis, 1996.
- *Introduction to Reinforcement Learning*, Sutton and Barto, 1998.
- *Markov Decision Problems*, Puterman, 1994.
- *Markov Decision Processes in Artificial Intelligence*, Sigaud and Buffet ed., 2008.
- *Algorithms for Reinforcement Learning*, Szepesvári, 2009.

Introduction to Reinforcement Learning (RL)

- Acquire skills for sequencial decision making in complex, stochastic, partially observable, possibly adversarial, environments.
- Learning from experience a behavior policy (what to do in each situation) from past success or failures
- **Examples**: Hot and Cold game, chess game, learning to ride a bicycle, autonomous robotics, operation research, decision making in stochastic market, ... any adaptive sequential decision making task.

Birth of the domain

Meeting in the end of the 70s:

- **Computational Neurosciences**. Reinforcement of synaptic weights in neuronal transmissions (Hebbs rules, Rescorla-Wagner models).
  Reinforcement = correlations in neuronal activity.
- **Experimental Psychology**. Animal conditioning: reinforcement of behaviors that lead to a satisfaction (behaviorism initiated by Pavlov, Skinner).
  Reinforcement = satisfaction or discomfort.

Independently, a mathematical formalism appeared in the 50s, 60s: **Dynamic Programming** by R. Bellman, in optimal control theory.
Reinforcement = criterion to be optimized.

Experimental Psychology

About animal conditioning, Thorndike (1911) says:

> *"Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond"*

"The Law of Effect".

History of the computational RL

- Shannon 1950: Programming a computer for playing chess.
- Minsky 1954: Neural-Analog Reinforcement Systems.
- Samuel 1959: Learning for the game of checkers.
- Michie 1961: Trial and error for tic-tac-toe game.
- Michie and Chambers 1968: Inverted pendulum.
- Widrow, Gupta, Maitra 1973: Punish/reward: learning with a critic in adaptive threshold systems. Neuronal rules.
- Barto, Sutton, Anderson 1983: Actor-Critic neuronal rules.
- Sutton 1984: Temporal Credit Assignment in RL.
- Sutton 1988: Learning from temporal differences.
- Watkins 1989: Q-learning.
- Tesauro 1992: TD-Gammon

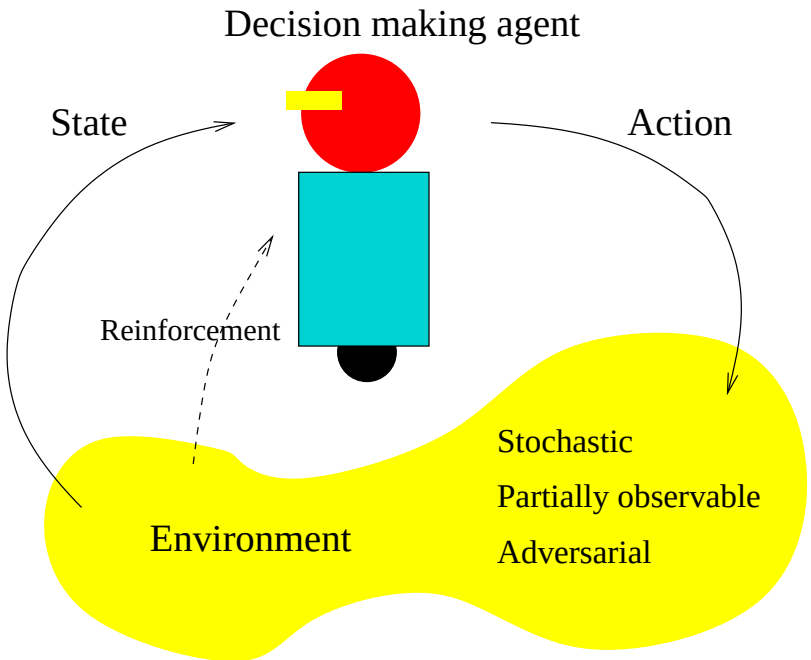
A few applications

- TD-Gammon. [Tesauro 1992-1995]: Backgammon.
- KnightCap [Baxter et al. 1998]: chess ($\simeq$2500 ELO)
- Robotics: juggling, acrobots [Schaal and Atkeson, 1994]
- Mobile robot navigation [Thrun et al., 1999]
- Elevator controller [Crites et Barto, 1996],
- Packet Routing [Boyan et Littman, 1993],
- Job-Shop Scheduling [Zhang et Dietterich, 1995],
- Production manufacturing optimization[Mahadevan et al., 1998],
- Game of poker (Bandit algo for Nash computation)
- Game of go (hierarchical bandits, UCT)

`http://www.ualberta.ca/~szepesva/RESEARCH/RLApplications.html`


INFINITE
MARIO BROS
(NOT Sequential Learning)
http://www

Reinforcement Learning

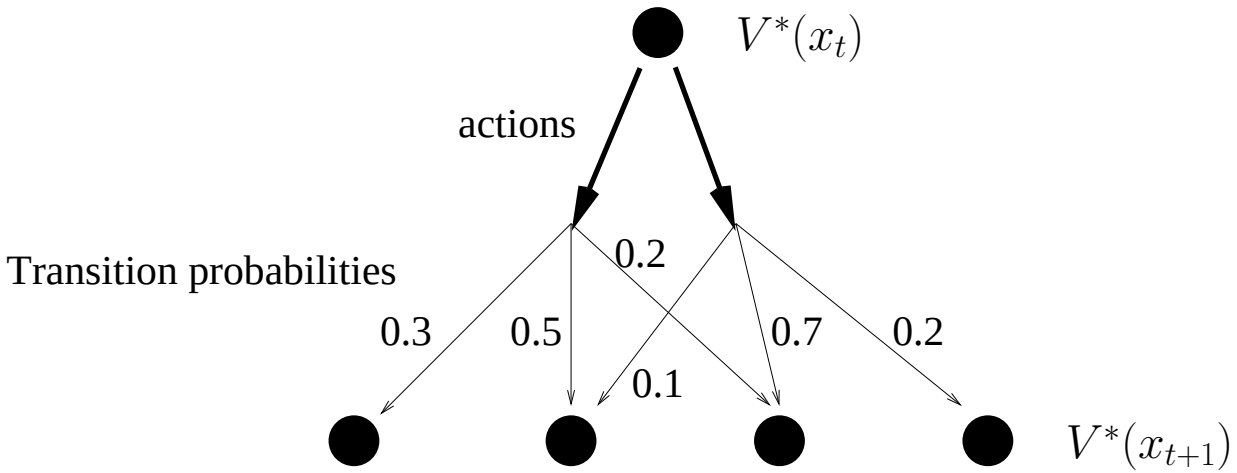


- **Environment:** can be stochastic (Tetris), adversarial (Chess), partially unknown (bicycle), partially observable (robot)
- **Available information:** the reinforcement (may be delayed)
- **Goal:** maximize the expected sum of future rewards.

Problem: How to sacrify a short term small reward to priviledge larger rewards in the long term?

Optimal value function

- Gives an evaluation of each state if the agent plays optimally.
- Ex: in a stochastic environment:



- Bellman eq.:
  $V^*(x_t) = \max_{a \in A} \left[r(x_t, a) + \sum_y p(y|x_t, a) V^*(y) \right]$
- Temporal difference: $\delta_t(V^*) = V^*(x_{t+1}) + r(x_t, a_t) - V^*(x_t)$
- If V^* is known, then when choosing the optimal action a_t, $\mathbb{E}[\delta_t(V^*)] = 0$ (i.e., in average there is no surprise)

Learning the value function

- How can we learn the value function?
  Thanks to the surprise! (i.e., the temporal difference)
  If V is an approximation of V^* and we observe

$$\delta_t(V) = V(x_{t+1}) + r(x_t, a_t) - V(x_t) > 0$$

  then $V(x_t)$ should be increased.
- From V^* we deduce the optimal action in x_t:

$$\arg\max_{a \in A} \left[r(x_t, a) + \sum_y p(y|x_t, a) V^*(y) \right]$$

  locally maximizing the value function $\iff$ maximizing the sum of future rewards.

Note that RL is really different from supervised learning!

Challenges of RL

- **The state-dynamics and reward functions are unknown**. Two approaches:
  - Model-based: learn first a model, then use dynamic programming
  - Model-free: use samples to directly learn a good value function
- **The curse of dimensionality**: In high-dimensional problems, the computational complexity may be prohibitively large. Need to learn an *approximation* of the value function / policy.
- **Exploration issue:** Where should one explore the state space in order to build a good representation of the unknown function where (and only where) it is useful.

Introduction to Dynamic Programming

References:

- *Neuro Dynamic Programming*, Bertsekas et Tsitsiklis, 1996.
- *Markov Decision Problems*, Puterman, 1994.
- *Markov Decision Processes in Artificial Intelligence*, Sigaud and Buffet ed., 2008.

Markov Decision Process

[Bellman 1957, Howard 1960, Dubins et Savage 1965]
We consider a discrete-time process $(x_t) \in X$ where:

- X: **state space** (assumed to be finite)
- A: **action space** (or decisions) (assumed to be finite)
- **State dynamics**: All relevant information about future is included in the current state and action

$$\mathbb{P}(x_{t+1} \,|\, x_t, x_{t-1}, \ldots, x_0, a_t, a_{t-1}, \ldots, a_0) = \mathbb{P}(x_{t+1} \,|\, x_t, a_t)$$

(**Markov property**). Thus we can define **transition probabilities** $p(y|x,a) = \mathbb{P}(x_{t+1} = y | x_t = x, a_t = a)$.

- **Reinforcement** (or **reward**): $r(x,a)$ is obtained when choosing action a in state x.

An MDP is defined by (X, A, p, r).

Definition of policy

Policy $\pi = (\pi_1, \pi_2, \dots)$, where $\pi_t : X \to A$ defines the action $\pi_t(x)$ choosen in x at time t.

If $\pi = (\pi, \pi, \dots, \pi)$, the policy is *stationary* ou *Markovian*.

When following a stationary policy π the process $(x_t)_{t \geq 0}$ is a Markov chain with transition probabilities

$$p(x_{t+1}|x_t) = p(x_{t+1}|x_t, \pi_t(x_t)).$$

Our goal is to learn a policy that maximizes the sum of rewards.

Performance of a policy

For any policy π, define the **value function** V^π:
Infinite horizon:

- Discounted: $V^\pi(x) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \,|\, x_0 = x; \pi\big]$,

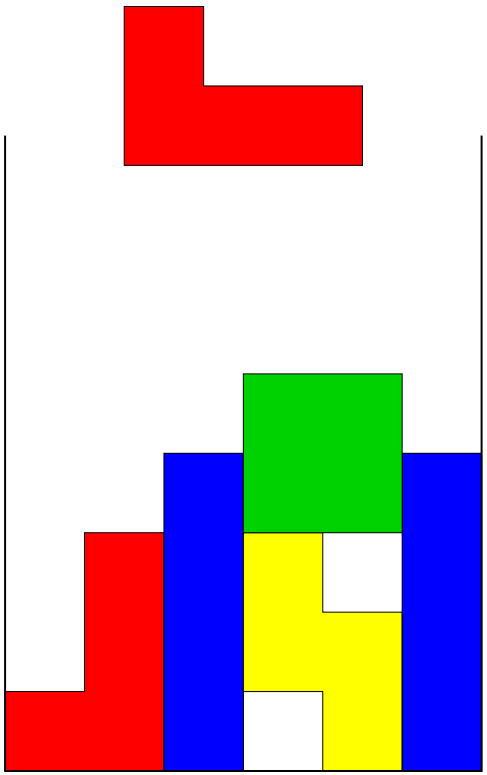
  where $0 \leq \gamma < 1$ is the discount factor
- Undiscounted: $V^\pi(x) = \mathbb{E}\big[\sum_{t=0}^{\infty} r(x_t, a_t) \,|\, x_0 = x; \pi\big]$
- Average: $V^\pi(x) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\big[\sum_{t=0}^{T-1} r(x_t, a_t) \,|\, x_0 = x; \pi\big]$

Finite horizon: $V^\pi(x, t) = \mathbb{E}\big[\sum_{s=t}^{T-1} r(x_s, a_s) + R(x_T) \,|\, x_t = x; \pi\big]$

Goal of the MDP: Find an **optimal policy** π^*, i.e.

$$V^{\pi^*} = \sup_\pi V^\pi$$

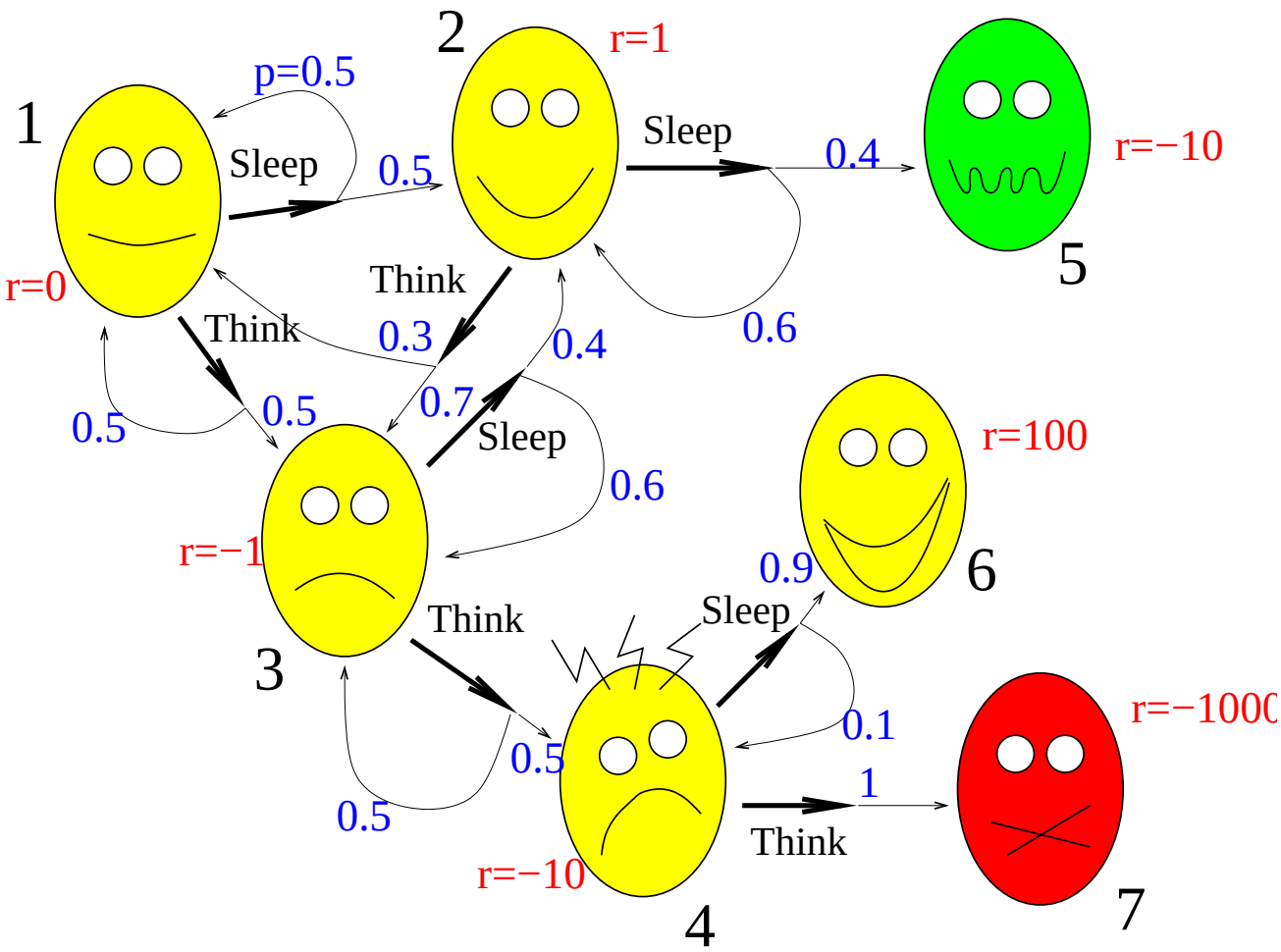
Example: Tetris



- **State**: wall configuration + new piece
- **Action**: posible positions of the new piece on the wall,
- **Reward**: number of lines removed
- **Next state**: Resulting configuration of the wall + random new piece.

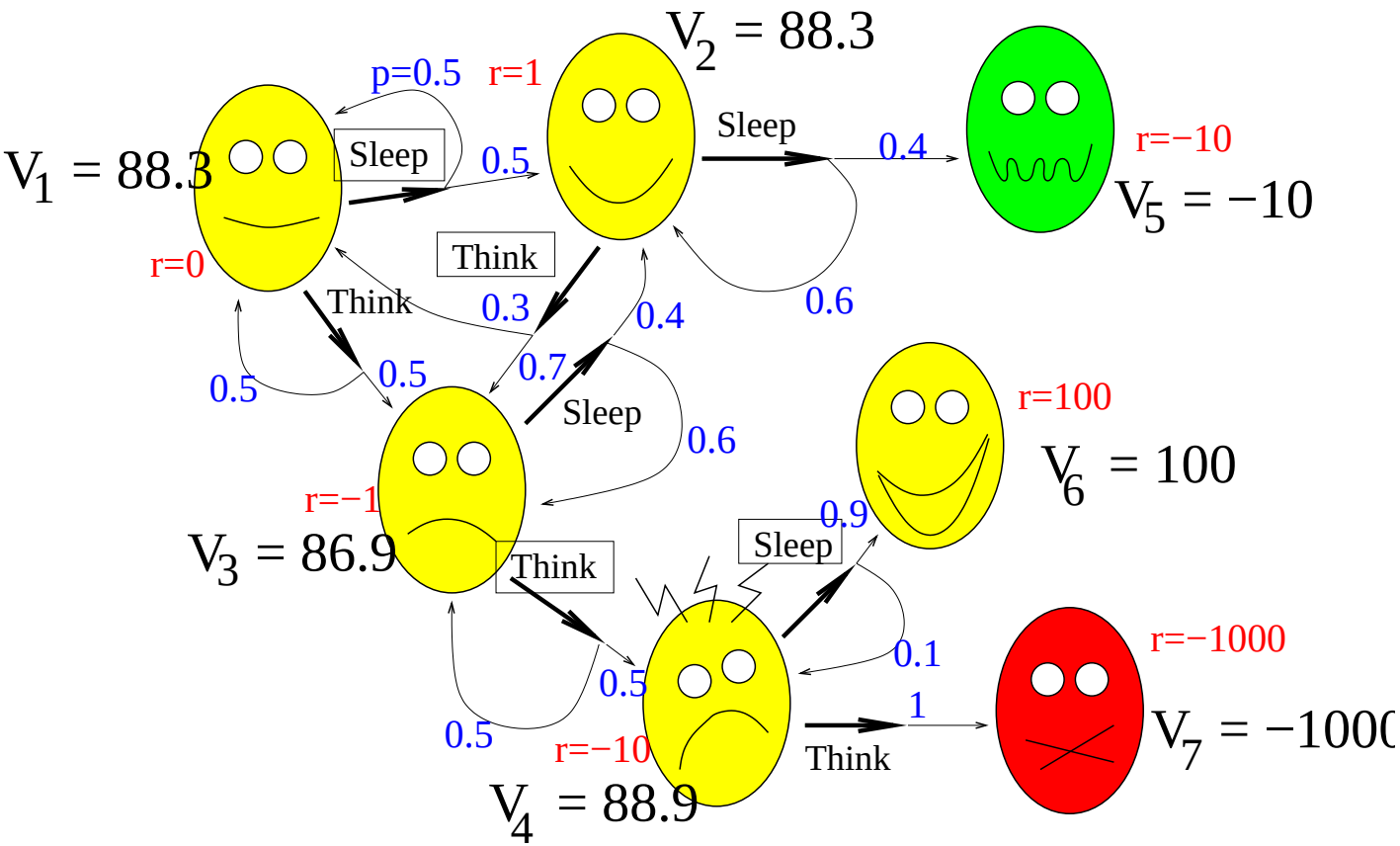
(we can prove that for any policy, the game ends in finite time a.s.)
State space 10^{61} states for regular Tetris

The dilemma of the MLSS student



You try to maximize the sum of rewards!

Solution of the MLSS student



$V_5 = -10,\ V_6 = 100,\ V_7 = -1000,$

$V_4 = -10 + 0.9V_6 + 0.1V_4 \simeq 88.9.$

$V_3 = -1 + 0.5V_4 + 0.5V_3 \simeq 86.9.\ V_2 = 1 + 0.7V_3 + 0.3V_1$ and

$V_1 = \max\{0.5V_2 + 0.5V_1, 0.5V_3 + 0.5V_1\}$, thus: $V_1 = V_2 = 88.3.$

Infinite horizon, discounted problems

Let π be a stationary deterministic policy.
Define the **value function** V^π as:

$$V^\pi(x) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(x_t)) \,|\, x_0 = x; \pi\Big],$$

where $0 \leq \gamma < 1$ a discount factor (which relates rewards in the future compared to current rewards).
And define the **optimal value function**: $V^* = \sup_\pi V^\pi$.

Proposition 1 (Bellman equations).

For any policy π, V^π satisfies:

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{y \in X} p(y|x, \pi(x)) V^\pi(y),$$

and V^* satisfies:

$$V^*(x) = \max_{a \in A} \Big[r(x, a) + \gamma \sum_{y \in X} p(y|x, a) V^*(y)\Big].$$

Proof of Proposition 1

$$
\begin{aligned}
V^\pi(x) &= \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t r(x_t,\pi(x_t))\,|\,x_0=x;\pi\Big] \\
&= r(x,\pi(x)) + \mathbb{E}\Big[\sum_{t\geq 1}\gamma^t r(x_t,\pi(x_t))\,|\,x_0=x;\pi\Big] \\
&= r(x,\pi(x)) + \gamma\sum_y P(x_1=y\,|\,x_0=x;\pi) \\
&\qquad\qquad \mathbb{E}\Big[\sum_{t\geq 1}\gamma^{t-1} r(x_t,\pi(x_t))\,|\,x_1=y;\pi\Big] \\
&= r(x,\pi(x)) + \gamma\sum_y p(y|x,\pi(x))V^\pi(y).
\end{aligned}
$$

Proof of Proposition 1 (continued)

And for all policy $\pi = (a, \pi')$ (not necessarily stationary),

$$
\begin{aligned}
V^*(x) &= \max_{\pi} \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r(x_t, \pi(x_t)) \,|\, x_0 = x; \pi\Big] \\
&= \max_{(a,\pi')} \Big[r(x,a) + \gamma \sum_y p(y|x,a) V^{\pi'}(y)\Big] \\
&= \max_{a} \Big[r(x,a) + \gamma \sum_y p(y|x,a) \max_{\pi'} V^{\pi'}(y)\Big] \qquad (1) \\
&= \max_{a} \Big[r(x,a) + \gamma \sum_y p(y|x,a) V^*(y)\Big].
\end{aligned}
$$

where (1) holds since:

- $\max_{\pi'} \sum_y p(y|x,a) V^{\pi'}(y) \leq \sum_y p(y|x,a) \max_{\pi'} V^{\pi'}(y)$
- Let $\bar{\pi}$ be the policy defined by $\bar{\pi}(y) = \arg\max_{\pi'} V^{\pi'}(y)$. Thus $\sum_y p(y|x,a) \max_{\pi'} V^{\pi'}(y) = \sum_y p(y|x,a) V^{\bar{\pi}}(y) \leq \max_{\pi'} \sum_y p(y|x,a) V^{\pi'}(y)$.

Bellman operators

Since X is finite (say with N states), V^π can be considered as a vector of $\boldsymbol{R}^N$.
Write:

- $r^\pi \in \boldsymbol{R}^N$ the vector with components $r^\pi(x) = r(x, \pi(x))$
- $P^\pi \in \boldsymbol{R}^{N \times N}$ the stochastic matric with elements $P^\pi(x, y) = p(y|x, \pi(x))$.

Define the

- **Bellman operator** $\mathcal{T}^\pi : \boldsymbol{R}^N \to \boldsymbol{R}^N$: for any $W \in \boldsymbol{R}^N$,

$$\mathcal{T}^\pi W(x) = r(x, \pi(x)) + \gamma \sum_y p(y|x, \pi(x)) W(y)$$

- **Dynamic Programming operator** $\mathcal{T} : \boldsymbol{R}^N \to \boldsymbol{R}^N$:

$$\mathcal{T} W(x) = \max_{a \in A} \left[r(x, a) + \gamma \sum_y p(y|x, a) W(y) \right].$$

Properties of the value functions

Proposition 2.

1. *V^π is the unique fixed-point of $\mathcal{T}^\pi$*

$$V^\pi = \mathcal{T}^\pi V^\pi.$$

2. *V^* is the unique fixed-point of $\mathcal{T}$:*

$$V^* = \mathcal{T} V^*.$$

3. *For any policy π, we have $V^\pi = (I - \gamma P^\pi)^{-1} r^\pi$*
4. *The policy defined by*

$$\pi^*(x) \in \arg\max_{a \in A} \left[r(x,a) + \gamma \sum_y p(y|x,a) V^*(y) \right]$$

is optimal (and stationary)

Proof of Proposition 2 [part 1]

Basic properties of the Bellman operators:

- **Monotonicity**: If $W_1 \leq W_2$ (componentwise) then

$$\mathcal{T}^\pi W_1 \leq \mathcal{T}^\pi W_2, \text{ and } \mathcal{T} W_1 \leq \mathcal{T} W_2.$$

- **Contraction in max-norm**: For any vectors W_1 and W_2,

$$\begin{aligned} ||\mathcal{T}^\pi W_1 - \mathcal{T}^\pi W_2||_\infty &\leq \gamma ||W_1 - W_2||_\infty, \\ ||\mathcal{T} W_1 - \mathcal{T} W_2||_\infty &\leq \gamma ||W_1 - W_2||_\infty. \end{aligned}$$

Indeed, for all $x \in X$,

$$\begin{aligned} |\mathcal{T} W_1(x) - \mathcal{T} W_2(x)| &= \Big| \max_a \Big[r(x,a) + \gamma \sum_y p(y|x,a) W_1(y) \Big] \\ &\quad - \max_a \Big[r(x,a) + \gamma \sum_y p(y|x,a) W_2(y) \Big] \Big| \\ &\leq \gamma \max_a \sum_y p(y|x,a) |W_1(y) - W_2(y)| \\ &\leq \gamma ||W_1 - W_2||_\infty \end{aligned}$$

Proof of Proposition 2 [part 2]

1. From Proposition 1, V^π is a fixed point of $\mathcal{T}^\pi$. Uniqueness comes from the contraction property of $\mathcal{T}^\pi$.
2. Idem for V^*.
3. From Point 1, $V^\pi = r^\pi + \gamma P^\pi V^\pi$. Thus $(I - \gamma P^\pi)V^\pi = r^\pi$. Now P^π is a stochastic matrix (whose eingenvalues have a modulus ≤ 1), thus the eing. of $(I - \gamma P^\pi)$ have a modulus $\geq 1 - \gamma > 0$, thus is invertible.
4. From the definition of π^*, we have $\mathcal{T}^{\pi^*} V^* = \mathcal{T} V^* = V^*$. Thus V^* is the fixed-point of $\mathcal{T}^{\pi^*}$. But, by definition, V^{π^*} is the fixed-point of $\mathcal{T}^{\pi^*}$ and since there is uniqueness of the fixed-point, $V^{\pi^*} = V^*$ and π^* is optimal.

Value Iteration

Proposition 3.

- *For any $V_0 \in \mathbb{R}^N$, define $V_{k+1} = \mathcal{T} V_k$. Then $V_k \to V^*$.*
- *Idem for V^π: define $V_{k+1} = \mathcal{T}^\pi V_k$. Then $V_k \to V^\pi$.*

Proof.

$$||V_{k+1} - V^*|| = ||\mathcal{T} V_k - \mathcal{T} V^*|| \leq \gamma ||V_k - V^*|| \leq \gamma^{k+1} ||V_0 - V^*|| \to 0$$

(idem for V^π) □

Variant: asynchronous iterations

Policy Iteration

Choose any initial policy π_0. Iterate:

1. **Policy evaluation**: compute V^{π_k}.
2. **Policy improvement**: π_{k+1} greedy w.r.t. V^{π_k}:

$$\pi_{k+1}(x) \in \arg\max_{a \in A} \left[r(x,a) + \gamma \sum_y p(y|x,a) V^{\pi_k}(y) \right],$$

(i.e. $\pi_{k+1} \in \arg\max_\pi \mathcal{T}^\pi V^{\pi_k}$)

Stop when $V^{\pi_k} = V^{\pi_{k+1}}$.

Proposition 4.

Policy iteration generates a sequence of policies with increasing performance ($V^{\pi_{k+1}} \geq V^{\pi_k}$) and terminates in a finite number of steps with the optimal policy π^*.

Proof of Proposition 4

From the definition of the operators $\mathcal{T}$, $\mathcal{T}^{\pi_k}$, $\mathcal{T}^{\pi_{k+1}}$ and from π_{k+1},

$$V^{\pi_k} = \mathcal{T}^{\pi_k} V^{\pi_k} \leq \mathcal{T} V^{\pi_k} = \mathcal{T}^{\pi_{k+1}} V^{\pi_k}, \tag{2}$$

and from the monotonicity of $\mathcal{T}^{\pi_{k+1}}$, we have

$$V^{\pi_k} \leq \lim_{n \to \infty} (\mathcal{T}^{\pi_{k+1}})^n V^{\pi_k} = V^{\pi_{k+1}}.$$

Thus $(V^{\pi_k})_k$ is a non-decreasing sequence. Since there is a finite number of possible policies (finite state and action spaces), the stopping criterion holds for a finite k; We thus have equality in (2), thus

$$V^{\pi_k} = \mathcal{T} V^{\pi_k}$$

so $V^{\pi_k} = V^*$ and π_k is an optimal policy.

Back to Reinforcement Learning

What if the transition probabilities $p(y|x,a)$ and the reward functions $r(x,a)$ are unknown?
In DP, we used their knowledge

- in value iteration:

$$V_{k+1}(x) = \mathcal{T} V_k(x) = \max_a \left[r(x,a) + \gamma \sum_y p(y|x,a) V_k(y) \right].$$

- in policy iteration:
  - when computing V^{π_k} (which requires iterating $\mathcal{T}^{\pi_k}$)
  - when computing the greedy policy:

$$\pi_{k+1}(x) \in \arg\max_{a \in A} \left[r(x,a) + \gamma \sum_y p(y|x,a) V^{\pi_k}(y) \right],$$

RL = introduction of 2 ideas: **sampling** and **Q-functions**.

1st idea: Use sampling

Use the real system (or a simulator) to **generate trajectories**.
This means that from any state-action (x_t, a_t) we can obtain a next-state sample $x_{t+1} \sim p(\cdot|x_t, a_t)$ and a reward sample $r(x_t, a_t)$.
So we can estimate $V^\pi(x)$ by **Monte-Carlo**:
Run trajectories (x_t^k) starting from x and following π:

$$V_{k+1}(x) = (1 - \eta_k) V_k(x) + \eta_k \sum_{t \geq 0} \gamma^t r(x_t^k, \pi(x_t^k)),$$

where e.g. $\eta_k = 1/k$ (sample mean), or more generally use **Stochastic Approximation** with $\sum_k \eta_k = \infty$ and $\sum_k \eta_k^2 < \infty$.

Problem: this method should be repeated for all initial states x and is not sample efficient.

Temporal difference learning

The update rule rewrites: for $s \leq t$ (by writing $r_t = r(x_t, \pi(x_t))$)

$$\begin{aligned} V_{k+1}(x_s) &= (1-\eta_k)V_k(x_s) + \eta_k \sum_{t \geq s} \gamma^{t-s} r_t, \\ &= V_k(x_s) + \eta_k \sum_{t \geq s} \gamma^{t-s} \big[\underbrace{r_t + \gamma V_k(x_{t+1}) - V_k(x_t)}_{\delta_t(V_k)} \big] \end{aligned}$$

$\delta_t(V_k)$ is the **temporal difference** for the transition $x_t \to x_{t+1}$.

- $\delta_t(V_k)$ provides an indication of the direction towards which the estimate $V_k(x_s)$ should be updated.
- Note that $\mathbb{E}[\delta_t(V^\pi)] = 0$ (this is Bellman equation).

TD(λ) algorithm

[Sutton, 1988] Observe a trajectory (x_t) by following π. Update the value of each states (x_s):

$$V_{k+1}(x_s) = V_k(x_s) + \eta_k(x_s) \sum_{t \geq s} (\gamma\lambda)^{t-s} \delta_t(V_k),$$

where $0 \leq \lambda \leq 1$.
$V_k(x_s)$ is impacted by $\delta_t(V_k)$ at all times $t \geq s$

- For $\lambda = 1$, we recover Monte-Carlo:

$$V_{k+1}(x_s) = V_k(x_s) + \eta_k(x_s) \sum_{t \geq s} \gamma^{t-s} \delta_t(V_k)$$

- For $\lambda = 0$, we have TD(0):

$$V_{k+1}(x_s) = V_k(x_s) + \eta_k(x_s) \delta_s(V_k)$$

Convergence of TD(λ)

Proposition 5 (Jaakkola, Jordan et Singh, 1994).

Assume that all states $x \in X$ are visited infinitely often and that the learning steps $\eta_k(x)$ satisfy for all $x \in X$, $\sum_{k\geq 0} \eta_k(x) = \infty$, $\sum_{k\geq 0} \eta_k^2(x) < \infty$, then $V_k \xrightarrow{a.s.} V^\pi$.

Proof.

TD(0) rewrites $V_{k+1}(x_s) = V_k(x_s) + \eta_k(x_s)\left[\widehat{\mathcal{T}^\pi} V_k(x_s) - V_k(x_s)\right]$, where

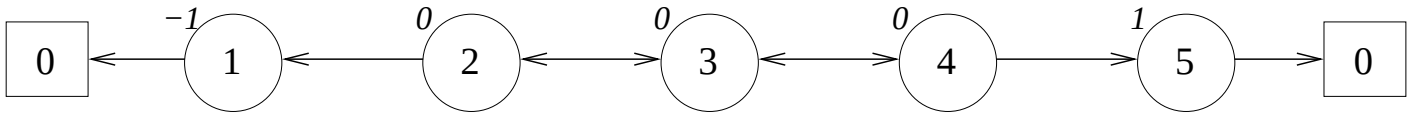
$$\begin{aligned} \widehat{\mathcal{T}^\pi} V_k(x_s) &= r_s + \gamma V_k(x_{s+1}) \text{ is a noisy estimate of} \\ \mathcal{T}^\pi V_k(x_s) &= r(x_s, a_s) + \gamma \sum_y p(y|x_s, a_s) V_k(y). \end{aligned}$$

This is a Stochastic Approximation algorithm for estimating the fixed-point of the contraction mapping $\mathcal{T}^\pi$.

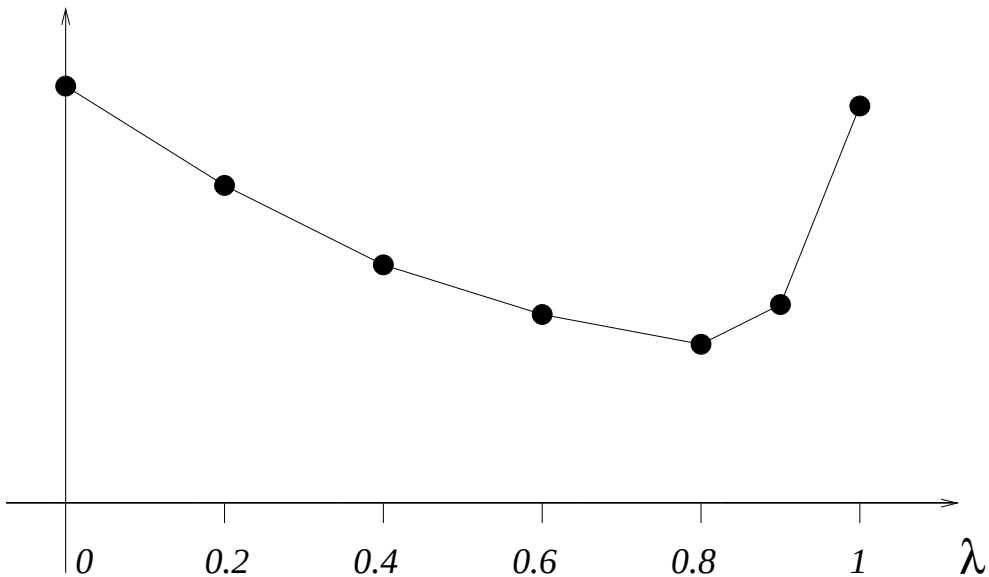
TD(λ) similar.

Tradeoff for λ

Example of the linear chain:



Expected quadratic error (for 100 trajectories):



- Small λ reduces variance
- Large λ propagates rewards faster

2nd Idea: use Q-value functions

Define the Q-value function $Q^\pi : X \times A \to \boldsymbol{R}$: for a policy π,

$$Q^\pi(x,a) = \mathbb{E}\Big[\sum_{t\geq 0} \gamma^t r(x_t,a_t) | x_0 = x, a_0 = a, a_t = \pi(x_t), t \geq 1\Big]$$

and the optimal Q-value function $Q^*(x,a) = \max_\pi Q^\pi(x,a)$.

Proposition 6.

Q^π and Q^* satisfy the Bellman equations:

$$\begin{aligned} Q^\pi(x,a) &= r(x,a) + \gamma \sum_{y\in X} p(y|x,a) Q^\pi(y,\pi(y)) \\ Q^*(x,a) &= r(x,a) + \gamma \sum_{y\in X} p(y|x,a) \max_{b\in A} Q^\pi(y,b) \end{aligned}$$

Idea: compute Q^* and then $\pi^*(x) \in \arg\max_a Q^*(x,a)$.

Q-learning algorithm

[Watkins, 1989] Whenever a transition $x_t, a_t \xrightarrow{r_t} x_{t+1}$ occurs, update the Q-value:

$$Q_{k+1}(x_t, a_t) = Q_k(x_t, a_t) + \eta_k(x_t, a_t)\left[r_t + \gamma \max_{b \in A} Q_k(x_{t+1}, b) - Q_k(x_t, a_t)\right].$$

Proposition 7 (Watkins et Dayan, 1992).

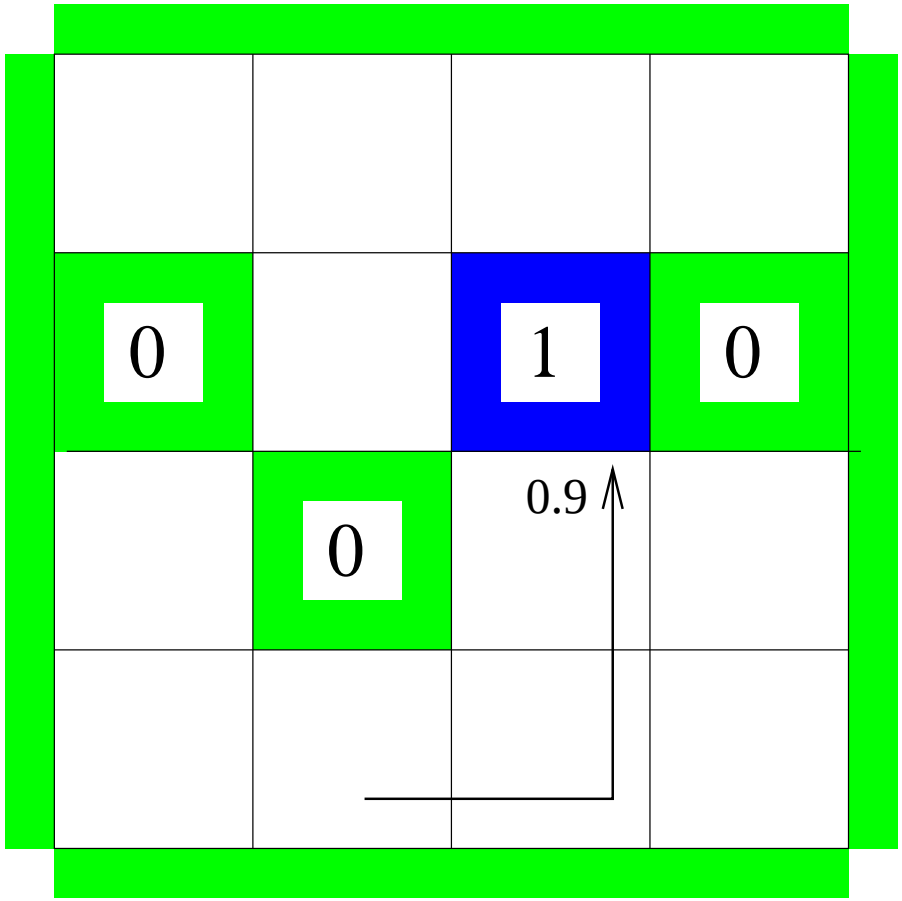
Assume that all state-action pairs (x, a) are visited infinitely often and that the learning steps satisfy for all x, a, $\sum_{k \geq 0} \eta_k(x, a) = \infty$, $\sum_{k \geq 0} \eta_k^2(x, a) < \infty$, *then* $Q_k \xrightarrow{a.s.} Q^*$.

Again the proof relies on Stochastic Approximation algorithm for estimating the fixed-point of a contraction mapping.

Remark: This does not say how to **explore** the space.

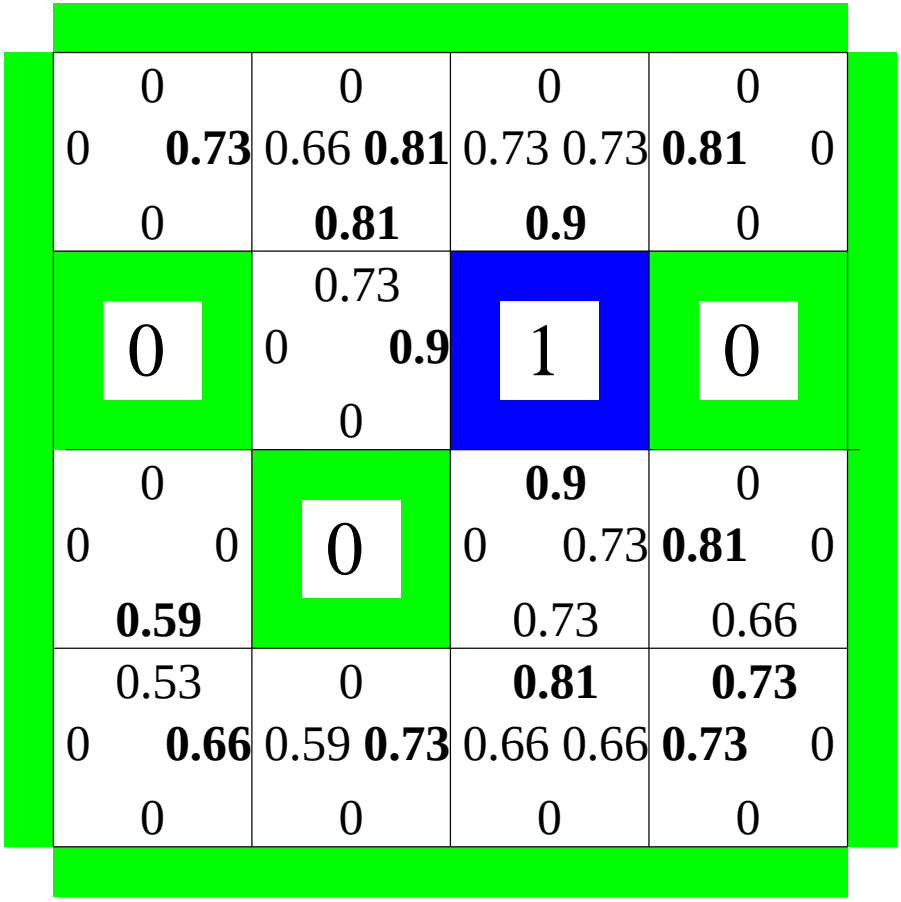
Q-learning algorithm

Deterministic case, discount factor $\gamma = 0.9$. Take steps $\eta = 1$.



After transition $x, a \xrightarrow{r} y$ update $Q_{k+1}(x, a) = r + \gamma \max_{b \in A} Q_k(y, b)$

Optimal Q-values



Bellman's equation: $Q^*(x, a) = \gamma \max_{b \in A} Q^*(\text{next-state}(x, a), b)$.

Conclusions of Part 1

When the state-space is finite and "small":

- If transition probabilities and rewards are known, then DP algorithms (value iteration, policy iteration) compute the optimal solution
- Otherwise, use sampling techniques and RL algorithms (TD(λ), Q-learning) apply

2 big problems:

- Usually state-space is HUGE! We face the **curse of dimensionality** and thus we need to find approximate solutions $\rightarrow$ Part 2.
- We need efficient **exploration** $\rightarrow$ Part 3.