Sapienza University of Rome

Master in Engineering in Computer Science

Artificial Intelligence & Machine Learning

A.Y. 2024/2025

Prof. Fabio Patrizi

17. Markov Decision Processes and Reinforcement Learning

Fabio Patrizi

Overview

- Markov Decision Processes
- Deterministic Q-Learning
- MDP Non-deterministic case
- Non-deterministic Q-Learning
- Temporal Difference
- SARSA
- Policy Gradient Algorithms

References:

- [AIMA] 17.1, 17.2
- T. Mitchell. Machine Learning. Ch. 13.
- R.S. Sutton, A.G Barto. Reinforcement Learning 2nd Edition. (Available at: `incompleteideas.net/book/RLbook2020.pdf`)

Markov Decision Processes (MDP)

Markov Decision Process (MDP): $M = (S, A, \delta, r)$

- S: finite set of states
- A: finite set of actions
- $\delta(s'|s, a)$: probability distribution over transitions
- $r : S \times A \times S \to \mathbb{R}$: reward function

Model of dynamic systems with

- *Markov Property*: next transition depends on current state and action, not on history
- *Full Observability*: current state is completely known

Markov Decision Processes (MDP)

$$M = (S, A, \delta, r)$$

Variants:

- Deterministic:
  - $\delta : S \times A \to S$ (transition function)
  - $r : S \times A \to \mathbb{R}$
- Nondeterministic (non-stochastic):
  - $\delta \subseteq S \times A \times S$ (transition relation)
  - $r : S \times A \times S \to \mathbb{R}$

Markov property

Equivalent formulations:

- Once current state is known, system evolution is independent of history of states and actions (and observations)
- Current state contains all information needed to predict the future
- Future states are conditionally independent of past states (and past observations), given current state
- Knowledge of current state makes past, present and future observations statistically independent

Markov process: a process with Markov property

Rewards

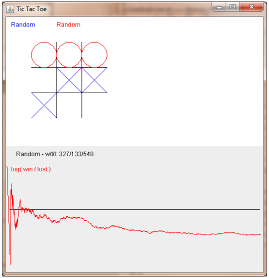
MDPs model also the desired task the agent has to carry out in the system

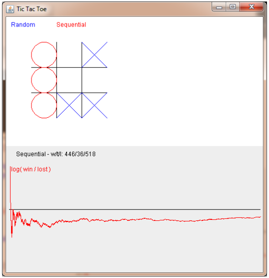
The *better* the agent behaves (wrt task) the *higher* the offered reward

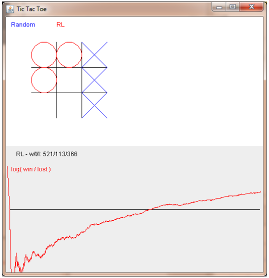
Reward function r models reward offering. Depends (in general) on:

- current state
- executed action
- next state

Example: Tic Tac Toe





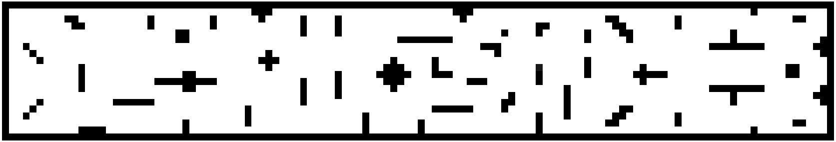


RL Example: Humanoid Walk

RL Example: Controlling an Helicopter

Example: deterministic grid controller

Reaching the right-most side of the environment from any initial state.

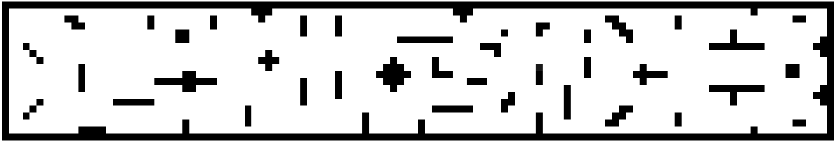


MDP $\langle S, A, \delta, r \rangle$

- $S = \{(r, c) | \text{coordinates in the grid}\}$
- $A = \{Left, Right, Up, Down\}$
- δ: cardinal movements with no effects (i.e., the agent remains in the current state) if destination state is a black square
- r: 1000 for reaching the right-most column, -10 for hitting obstacle, 0 otherwise

Example: non-deterministic grid controller

Reaching the right-most side of the environment from any initial state.



MDP $\langle S, A, \delta, r \rangle$

- $S = \{(r, c) | \text{coordinates in the grid}\}$
- $A = \{Left, Right\}$
- δ: cardinal movements with non-deterministic effects (0.1 probability of moving diagonally)
- r: 1000 for reaching the right-most column, -10 for hitting obstacle, $+1$ for any *Right* action, -1 for any *Left* action.

Policy and Rewards

Policy $\pi : S \rightarrow A$

- Function returning action for every state
- $\pi(s)$: action to execute in state s

(Expected) cumulative discounted reward (ECDR) wrt π and s

- $V^{\pi}(s) \equiv E[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots] = E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t]$
- with:
  - $r_t = r(s_t, \pi(s_t), s_{t+1})$
  - $s_1 = s$
  - $\gamma \in [0, 1)$ *discount factor* for future rewards

Observe:

- if $r(s, a, s') \leq R_{max}$ (for all s, a, s') and $\gamma \in [0, 1)$, then:

$$\sum_{t=1}^{\infty} \gamma^{t-1} r_t \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma} \text{ (sum of geometric series)}$$

Value function

Can assign a value to every $s \in S$, based on ECDR induced by policy π:

- General case:

$$V^{\pi}(s) \equiv E[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots]$$

- Deterministic case:

$$V^{\pi}(s) \equiv r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$$

V: *(state-)value* function (or V-function)

Optimal Policy

π^* is an *optimal* policy if, for every state $s \in S$ and every policy π:

$$V^{\pi^*}(s) \geq V^{\pi}(s)$$

MDP Solution Definition

Problem:

- Given: MDP $M = (S, A, \delta, r)$
- Find *optimal* policy: $\pi^* : S \to A$

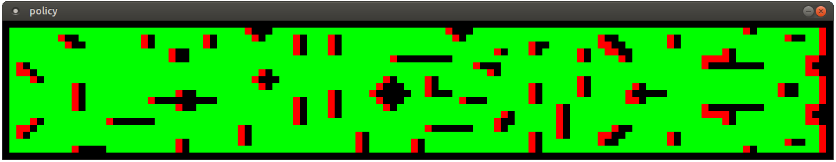
$$\pi^* \equiv \operatorname*{argmax}_{\pi} V^{\pi}(s) \, , \; \forall s \in S$$

For infinite-horizon, an MDP has always an optimal *stationary* policy

- *stationary*: optimal action depends on current state (not time-step)

Example: non-deterministic grid controller

Optimal policy



green: action *Right* red: action *Left*

Reasoning and Learning in MDP

Problem: MDP $M = (S, A, \delta, r)$

Solution: Policy $\pi : S \rightarrow A$

If the MDP $M = (S, A, \delta, r)$ is completely known $\rightarrow$ reasoning or planning

If the MDP $M = (S, A, \delta, r)$ is not completely known $\rightarrow$ learning

Observe:

- Simple examples of reasoning in MDP can be modeled as search problems and solved with standard algorithms (e.g., A*)

We focus on *Learning*

One-state Markov Decision Processes (MDP)

$$M = (\{s_0\}, A, \delta, r)$$

- $S = \{s_0\}$
- $A = \{a_1, \dots, a_n\}$ finite set of actions
- $\delta(s_0, a) = s_0, \ \forall a \in A$ transition function
- $r(s_0, a, s_0) = r(a)$ reward function

Optimal policy: $\pi^*(s_0) = a_i$, for some $a_i \in A$

Deterministic One-state MDP

If r is *deterministic* and *known*: pick action that yields highest reward:

$$\pi^*(s_0) = \operatorname*{argmax}_{a \in A} r(a)$$

What if reward function r is unknown?

Deterministic One-state MDP

If r *deterministic* and *unknown*:

- Learn r
- Pick action that yields highest reward

Algorithm:

1. for each $a \in A$
    - **execute** a and **collect** reward $r(a)$
2. $\pi^*(s_0) = \mathrm{argmax}_{a \in A}\, r(a)$

Note: $|A|$ iterations sufficient

Non-Deterministic One-state MDP

If r *non-deterministic* and *known*:

$$\pi^*(s_0) = \operatorname*{argmax}_{a \in A} E[r(a)]$$

Example:

For $r(a) = \mathcal{N}(\mu_a, \sigma_a)$:

$$\pi^*(s_0) = \operatorname*{argmax}_{a \in A} \mu_a$$

Non-Deterministic One-state MDP

If r *non-deterministic* and *unknown*

- Estimate r (by sampling)
- Pick action that yields (estimated) highest expected reward

Algorithm:

1. Initialize a data structure Θ (to estimate r)
2. For $t = 1, \ldots, T$ (until termination condition)
   - **choose** an action $a_{(t)} \in A$
   - **execute** $a_{(t)}$ and **collect** reward $r_{(t)}$
   - Update Θ
3. $\pi^*(s_0) = \operatorname{argmax}_{a \in A} \tilde{r}(a)$ (according to Θ)

Note: **many** iterations ($T >> |A|$) needed (the more, the better)

Non-Deterministic One-state MDP

Example:

r is *non-deterministic* and *unknown*, e.g.:

- $r(a) = \mathcal{N}(\mu_a, \sigma_a)$

Algorithm:

1. Θ, c: action-indexed arrays (with $\Theta[a]$ average reward for a)
2. for all $a \in A$: $\Theta[a] \leftarrow 0$ and $c[a] \leftarrow 0$
3. For $t = 1, \ldots, T$ (until termination condition)
   - **choose** $a \in A$
   - **execute** a and **collect** reward r
   - increment $c[a]$ by 1
   - update $\Theta[a] \leftarrow \frac{1}{c[a]}(r + (c[a] - 1)\Theta[a])$
4. $\pi^*(s_0) = \mathrm{argmax}_{a \in A} \Theta[a]$

Reinforcement Learning

Problem:

- Given MDP $M = (S, A, \delta, r)$, with *unknown* δ and r
- Determine: optimal policy $\pi^* : S \to A$

Reinforcement Learning $\neq$ Supervised Learning!

- Target function $\pi : S \to A$
- Training examples have form: $(s_1, a_1, r_1) \cdots (s_t, a_t, r_t)$
- Supervised learning examples have form: $(s_i, \pi(s_i))$
- Target function (optimal policy) samples not available!

Agent's Learning Task

Observations:

- Since δ and r unknown, agent cannot predict action effects (or rewards)
- However, agent can execute actions and observe outcomes

The learning task can be performed by repeating these steps:

- **choose** an action
- **execute** the chosen action
- **observe** the resulting new state
- **collect** the reward

Approaches to Learning with MDP

Two main approaches:

- Value iteration
  - Estimate Value function, then compute π
- Policy iteration
  - Estimate directly π

Learning through value iteration

The agent could learn the value function $V^{\pi^*}(s)$ (written as $V^*(s)$)

From which it could determine the optimal policy:

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

However, π^* cannot be computed in this way as δ and r are unknown

Q-Function (deterministic case)

$Q^{\pi}(s,a)$: expected utility of executing a in s, then acting according to π

$$Q^{\pi}(s,a) \equiv r(s,a) + \gamma V^{\pi}(\delta(s,a))$$

$$Q(s,a) \equiv r(s,a) + \gamma V^{*}(\delta(s,a))$$

Learning Q is sufficient to learn optimal policy (no need to know δ and r):

$$\pi^{*}(s) = \operatorname*{argmax}_{a \in A} Q(s,a)$$

Q-Function (deterministic case)

Observe:

$$V^*(s) = \max_{a \in A}\{r(s,a) + \gamma V^*(\delta(s,a))\} = \max_{a \in A} Q(s,a)$$

Thus, we can rewrite

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

as

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(\delta(s,a), a')$$

Training Rule to Learn Q (deterministic case)

Deterministic case:

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(\delta(s,a), a')$$

Let $\hat{Q}$ denote learner's current approximation of Q.

Training rule:

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

with:

- r: immediate reward (from executing a in s)
- s': state resulting from executing a in s

Q Learning Algorithm for Deterministic MDPs

1. for each s, a initialize table entry $\hat{Q}[s, a] \leftarrow 0$
2. observe current state s
3. for $t = 1, \ldots, T$ (until termination condition)
   - **choose** an action a
   - **execute** action a
   - **observe** new state s'
   - **collect** immediate reward r
   - update the table entry $\hat{Q}[s, a]$ as follows:

$$\hat{Q}[s, a] \leftarrow r + \gamma \max_{a' \in A} \hat{Q}[s', a']$$

   - $s \leftarrow s'$
4. return $\pi^*(s) = \mathrm{argmax}_{a \in A} \hat{Q}(s, a)$ (for all $s \in S$)

Observe: δ and r not used, just observing new state s' and r, resulting from executing a

Convergence in deterministic MDP

- $\hat{Q}_n(s, a)$ underestimates $Q(s, a)$
- For $r(a, s) \geq 0$, we have: $0 \leq \hat{Q}_n(s, a) \leq \hat{Q}_{n+1}(s, a) \leq Q(s, a)$
- Convergence guranteed if:
  - $|r(a, s)| \leq R_{max}$, for some $R_{max} \in \mathbb{R}$
  - all state-action pairs visited infinitely often

Experimentation Strategies

How does agent chooses next action?

- *Exploitation*: select action a that maximizes $\hat{Q}(s, a)$
- *Exploration*: select random action a (possibly lower $\hat{Q}(s, a)$)

ϵ-*greedy strategy*:

- Fix $\epsilon \in [0, 1]$
- select a random action with probability ϵ
- select best action with probability $1 - \epsilon$
- ϵ can decrease over time (prefer exploration first, then exploitation)

Experimentation Strategies

soft-max strategy:

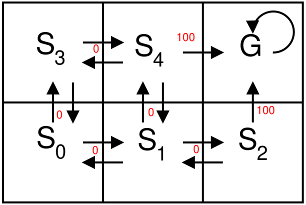
- actions with higher $\hat{Q}$ values are assigned higher probabilities
- every action is assigned non-null probability

$$P(a|s) = \frac{k^{\hat{Q}(s,a)}}{\sum_{a' \in A} k^{\hat{Q}(s,a')}}$$

- $k > 0$ determines how much actions with higher $\hat{Q}$ values are preferred
- k may increase over time (first exploration, then exploitation)

Example: grid world

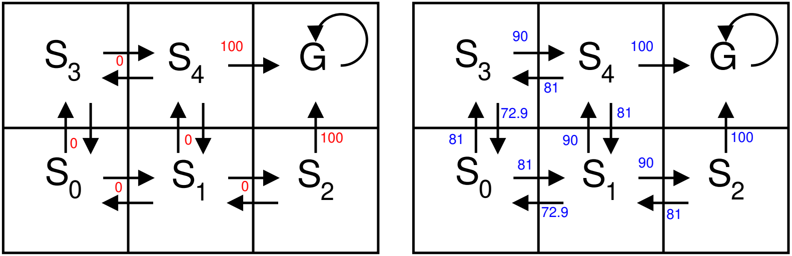
Reaching goal state G from s_0



$M = (S, A, \delta, r)$

- $S = \{s_0, s_1, s_2, s_3, s_4, G\}$
- $A = \{L, R, U, D\}$
- δ represented as arrows (e.g., $\delta(s_0, R) = s_1$)
- $r(s, a)$ represented as red values on arrows (e.g., $r(s_0, R) = 0$)

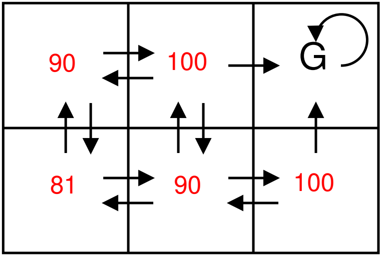
Example: grid world



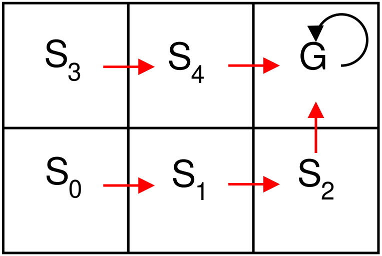
M $\hat{Q}$

(discount factor $\gamma = 0.9$)

Example: Grid World

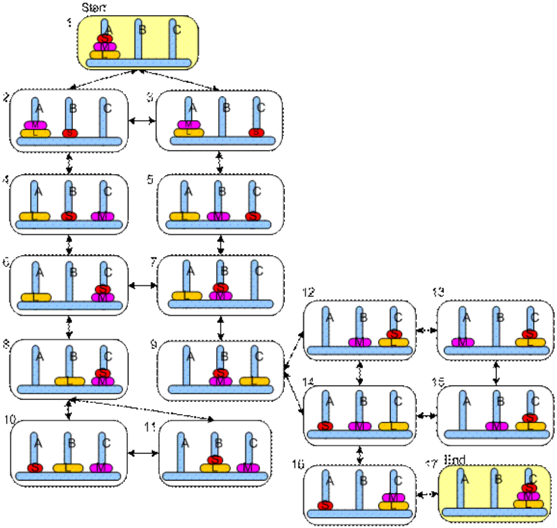


$V^*(s)$ values



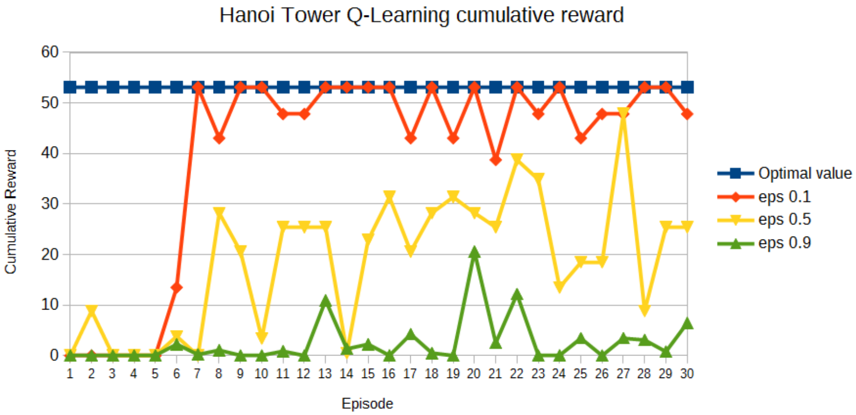
One optimal policy

Example: Hanoi Tower



Evaluating Reinforcement Learning Agents

RL agent evaluation usually measured by cumulative reward over time



Evaluating Reinforcement Learning Agents

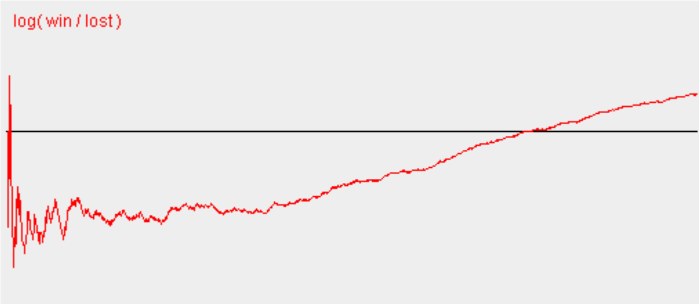
Cumulative reward plot may be very noisy, due to exploration phases

Possible solution:
Repeat until termination condition

1. Execute k steps of learning
2. Evaluate current policy π_k
   (μ, σ of cumulative reward obtained in d runs with no exploration)

Evaluating Reinforcement Learning Agents

Domain-specific performance metrics can also be used.



Average of all the results obtained during the learning process.

Non-deterministic Case

Transition and reward functions are non-deterministic.

V and Q defined using expected values

$$V^{\pi}(s) \equiv E[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots] = E[\sum_{t=1}^{\infty} \gamma^{t-1} r_t]$$

Optimal policy

$$\pi^* \equiv \operatorname*{argmax}_{\pi} V^{\pi}(s), \forall s \in S$$

Non-deterministic Case

Definition of Q

$$\begin{aligned} Q(s,a) &\equiv E[r(s,a,s') + \gamma V^*(s')](\text{with } s' \sim \delta(s'|s,a)) \\ &= \sum_{s'} \delta(s'|s,a)(r(s,a,s') + \gamma V^*(s')) \end{aligned}$$

Optimal policy

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} Q(s,a)$$

Example: k-Armed Bandit

One-state MDP with k actions: $a_1, \ldots, a_k$.

Stochastic case: $r(a) = \mathcal{N}(\mu_a, \sigma_a)$ Gaussian distribution

Choose a with ϵ-greedy strategy:

- uniform random choice with prob. ϵ (exploration)
- best choice with probability $1 - \epsilon$ (exploitation)

Training rule:

$$Q_n(a) \leftarrow Q_{n-1}(a) + \alpha[r - Q_{n-1}(a)]$$

$$\alpha = \frac{1}{1 + v_{n-1}(a)}$$

$v_{n-1}(a) =$ number of executions of action a up to time $n - 1$.

Exercise: k-Armed Bandit

Compare the following two strategies for the stochastic k-Armed Bandit problem (with Gaussian distributions), by plotting the reward over time.

1. For each of the k actions, perform 30 trials and compute the mean reward; then always play the action with the highest estimated mean.
2. ϵ-greedy strategy (with different values of ϵ) and training rule from previous slide.

Note: realize a parametric software with respect to k and the parameters of the Gaussian distributions and use the following values for the experiments: $k = 4$, $r(a_1) = \mathcal{N}(100, 50)$, $r(a_2) = \mathcal{N}(90, 20)$, $r(a_3) = \mathcal{N}(70, 50)$, $r(a_4) = \mathcal{N}(50, 50)$.

Example: k-Armed Bandit

What happens if parameters of Gaussian distributions slightly varies over time, e.g. $\mu_a \pm 10\%$ at unknown instants of time (with much lower frequency with respect to trials) ?

$$Q_n(a) \leftarrow Q_{n-1}(a) + \alpha[r - Q_{n-1}(a)]$$

$\alpha =$ constant

Non-deterministic Q-learning

Q learning generalizes to non-deterministic worlds with training rule:

$$\hat{Q}_n \leftarrow \hat{Q}_{n-1}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a') - \hat{Q}_{n-1}(s,a)]$$

which is equivalent to

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha)\hat{Q}_{n-1}(s,a) + \alpha[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')]$$

where

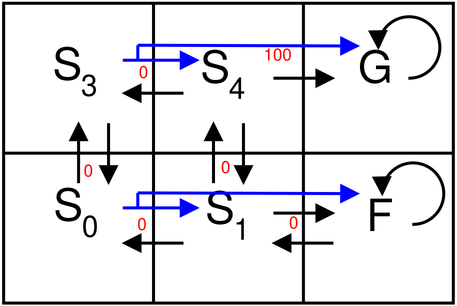
$$\alpha = \frac{1}{1 + visits_{n-1}(s,a)}$$

$visits_n(s,a)$: total number of times state-action pair (s,a) has been visited up to n-th iteration

Convergence in non-deterministic MDP

- Deterministic Q-learning does not converge in non-deterministic worlds!
- Non-deterministic Q-learning converges when every state-action pair visited infinitely often [Watkins and Dayan, 1992].
- Convergence guaranteed also for other $\alpha \in [0, 1)$

Example: non-deterministic Grid World



Other algorithms for non-deterministic learning

- Temporal Difference
- SARSA

Temporal Difference Learning

Q learning: reduce discrepancy between successive Q estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Two steps time difference:

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

n steps time difference:

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these ($0 \leq \lambda \leq 1$):

$$Q^{\lambda}(s_t, a_t) \equiv (1 - \lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \cdots \right]$$

Temporal Difference Learning

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda)\left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \cdots\right]$$

Equivalent expression:

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda)\max_{a} \hat{Q}(s_t, a_t) + \lambda \; Q^{\lambda}(s_{t+1}, a_{t+1})]$$

- $\lambda = 0$: $Q^{(1)}$ learning as seen before
- $\lambda > 0$: algorithm increases emphasis on discrepancies based on more distant look-aheads
- $\lambda = 1$: only observed r_{t+i} are considered.

Temporal Difference Learning

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda)\max_a \hat{Q}(s_t, a_t) + \lambda\ Q^{\lambda}(s_{t+1}, a_{t+1})]$$

TD(λ) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$ [Dayan, 1992]
- TD-Gammon [Tesauro, 1995] uses this algorithm (approximately equal to best human backgammon player).

SARSA

SARSA is based on the tuple (s, a, r, s', a').

$$\hat{Q}_n(s, a) \leftarrow \hat{Q}_{n-1}(s, a) + \alpha[r + \gamma \hat{Q}_{n-1}(s', a') - \hat{Q}_{n-1}(s, a)]$$

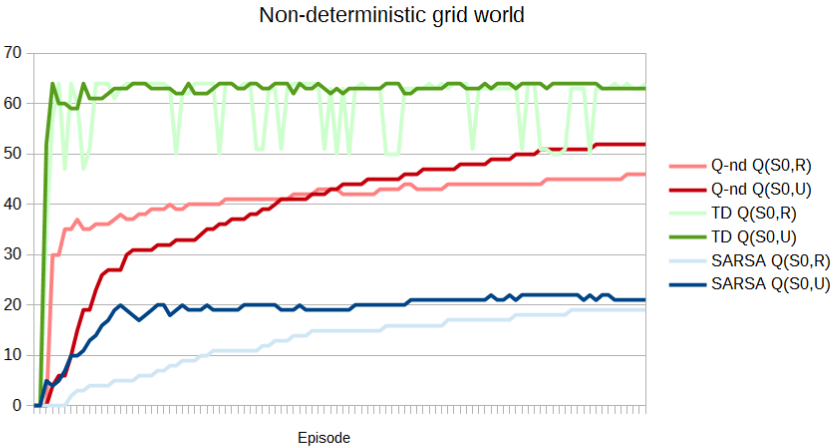
a' is chosen according to a policy based on current estimate of Q.

On-policy method: it evaluates the current policy

Convergence of non-deterministic algorithms

Fast convergence does not imply better solution in the optimal policy.

Example: comparison among Q-learning, TD, and SARSA.



Remarks on explicit representation of Q

- Explicit representation of $\hat{Q}$ table may not be feasible for large models.
- Algorithms perform a kind of rote learning. No generalization on unseen state-action pairs.
- Convergence is guaranteed only if every possible state-action pair is visited infinitely often.

Remarks on explicit representation of Q

Use function approximation:

$$Q_\theta(s, a) = \theta_0 + \theta_1 F_1(s, a) + \ldots + \theta_n F_n(s, a)$$

Use linear regression to learn $Q_\theta(s, a)$.

Remarks on explicit representation of Q

Use a neural network as function approximation and learn function Q

Implementation options:

- Train a network, using (s, a) as input and $\hat{Q}(s, a)$ as output
- Train a separate network for each action a, using s as input and $\hat{Q}(s, a)$ as output
- Train a network, using s as input and one output $\hat{Q}(s, a)$ for each action

TD-Gammon [Tesauro, 1995] uses a neural network together with TD(λ)

Reinforcement Learning with Policy Iteration

Use directly π instead of $V(s)$ or $Q(s, a)$

Parametric representation of π: $\pi_\theta(s) = \max_{a \in A} \hat{Q}_\theta(s, a)$

Policy value: $\rho(\theta) =$ expected value of executing π_θ

Policy gradient: $\nabla_\theta \rho(\theta)$

Policy Gradient Algorithm

Policy gradient algorithm for parameterized policy $\pi_\theta(s)$

choose θ
while *termination condition* **do**
    estimate $\nabla_\theta \rho(\theta)$ *(through experiments)*
    $\theta \leftarrow \theta + \eta \nabla_\theta \rho(\theta)$
end while

Policy Gradient Algorithm

Policy Gradient Algorithm for robot learning [Kohl and Stone, 2004]

Estimate optimal parameters of a controller $\pi_\theta = \{\theta_1, ..., \theta_N\}$, given an objective function F.

Method is based on iterating the following steps:

1) generating perturbations of π_θ by modifying the parameters
2) evaluate these perturbations
3) generate a new policy from "best scoring" perturbations

Policy Gradient Algorithm

General method

$\pi \leftarrow$ *InitialPolicy*

while *termination condition* **do**

compute $\{R_1, ..., R_t\}$, *random perturbations of* π

evaluate $\{R_1, ..., R_t\}$

$\pi \leftarrow$ *getBestCombinationOf*($\{R_1, ..., R_t\}$)

end while

Note: in the last step we can simply set $\pi \leftarrow \mathrm{argmax}_{R_j} F(R_j)$ (i.e., hill climbing).

Policy Gradient Algorithm

Perturbations are generated from π by

$$R_i = \{\theta_1 + \delta_1, ..., \theta_N + \delta_N\}$$

with δ_j randomly chosen in $\{-\epsilon_j, 0, +\epsilon_j\}$, for ϵ_j small fixed value wrt θ_j

Policy Gradient Algorithm

Combination of $\{R_1, ..., R_t\}$ is obtained by computing for each parameter j:
- $Avg_{-\epsilon,j}$: average score of all R_i with a negative perturbations
- $Avg_{0,j}$: average score of all R_i with a zero perturbation
- $Avg_{+\epsilon,j}$: average score of all R_i with a positive perturbations

Then define a vector $A = \{A_1, ..., A_N\}$ as follows

$$A_j = \begin{cases} 0 & \text{if } Avg_{0,j} > Avg_{-\epsilon,j} \text{ and } Avg_{0,j} > Avg_{+\epsilon,j} \\ Avg_{+\epsilon,j} - Avg_{-\epsilon,j} & \text{otherwise} \end{cases}$$

and finally

$$\pi \leftarrow \pi + \frac{A}{|A|}\eta$$

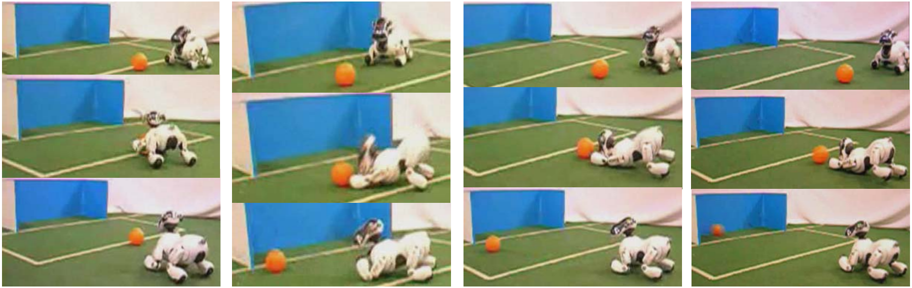
Policy Gradient Algorithm

Task: optimize AIBO gait for fast and stable locomotion [Saggar et al., 2006]
Objective function F

$$F = 1 - (W_t M_t + W_a M_a + W_d M_d + W_\theta M_\theta)$$

M_t: normalized time to walk between two landmarks
M_a: normalized standard deviation of AIBO's accelerometers
M_d: normalized distance of the centroid of landmark from the image center
M_θ: normalized difference between slope of the landmark and an ideal slope
W_t, W_a, W_d, W_θ: weights

Example: Robot Learning



References

[AI] S. Russell and P. Norvig. Artificial Intelligence: A modern approach, Chapter 21. Reinforcement Learning. 3rd Edition, Pearson, 2010.

[RL] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
On-line: `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/`

[ML] Tom Mitchell. Machine Learning. Chapter 13 Reinforcement Learning. McGraw-Hill, 1997.

[ArtInt] David Poole and Alan Mackworth. Artificial Intelligence: Foundations of Computational Agents, Chapter 11.3 Reinforcement Learning. Cambridge University Press, 2010.
On-line: `http://artint.info/`

[Watkins and Dayan, 1992] Watkins, C. and Dayan, P. Q-learning. Machine Learning, 8, 279-292. 1992.

References

[Dayan, 1992] Dayan, P.. The convergence of TD(λ) for general λ. Machine Learning, 8, 341-362. 1992.

[Tesauro, 1995] Gerald Tesauro. Temporal Difference Learning and TD-Gammon Communications of the ACM, Vol. 38, No. 3. 1995.

[Kohl and Stone, 2004] Nate Kohl and Peter Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2004.

[Saggar et al., 2006] Manish Saggar, Thomas D'Silva, Nate Kohl, and Peter Stone. Autonomous Learning of Stable Quadruped Locomotion. In RoboCup-2006: Robot Soccer World Cup X, pp. 98–109, Springer Verlag, 2007.