Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2019/2020

Prof. L. Iocchi, F. Patrizi, V. Ntouskos

18. Reinforcement Learning (non-deterministic)

L. Iocchi, F. Patrizi, V. Ntouskos

Overview

- MDP Non-deterministic case
- Non-deterministic Q-Learning
- Temporal Difference
- SARSA
- Policy gradient algorithms

References
Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. (2nd edition). On-line:
http://incompleteideas.net/book/the-book.html

Markov Decision Processes (MDP)

$$MDP = \langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$$

- $\mathbf{X}$ is a finite set of states
- $\mathbf{A}$ is a finite set of actions
- $P(\mathbf{x}'|\mathbf{x}, a')$ is a probability distribution over transitions
- $r(\mathbf{x}, a, \mathbf{x}')$ is a reward function

Non-deterministic Case

Transition and reward functions are non-deterministic.

We define V, Q by taking expected values

$$\begin{aligned} V^{\pi}(\mathbf{x}) & \equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \\ & \equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] \end{aligned}$$

Optimal policy

$$\pi^* \equiv \operatorname*{argmax}_{\pi} V^{\pi}(\mathbf{x}), (\forall \mathbf{x})$$

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

Non-deterministic Case

Definition of Q

$$\begin{aligned} Q(\mathbf{x}, a) & \equiv & E[r(\mathbf{x}, a) + \gamma V^*(\delta(\mathbf{x}, a))] \\ & = & E[r(\mathbf{x}, a)] + \gamma E[V^*(\delta(\mathbf{x}, a))] \\ & = & E[r(\mathbf{x}, a)] + \gamma \sum_{x'} P(\mathbf{x}'|\mathbf{x}, a) V^*(\mathbf{x}') \\ & = & E[r(\mathbf{x}, a)] + \gamma \sum_{x'} P(\mathbf{x}'|\mathbf{x}, a) \max_{a'} Q(\mathbf{x}', a') \end{aligned}$$

Optimal policy

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

Example: k-Armed Bandit

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

Stochastic case: $r(a_i) = \mathcal{N}(\mu_i, \sigma_i)$ Gaussian distribution

Choice ϵ-greedy:
uniform random choice with prob. ϵ (exploration),
best choice with probability $1 - \epsilon$ (expoitation).

Training rule:

$$Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha[r_{t+1}(a_i) - Q_{n-1}(a_i)]$$

$$\alpha = 1/v_{n-1}(a_i)$$

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

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

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. $\mu_i \pm 10\%$ at unknown instants of time (with much lower frequency with respect to trials) ?

$$Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha[r_{t+1}(a_i) - Q_{n-1}(a_i)]$$

$\alpha =$ constant

Non-deterministic Q-learning

Q learning generalizes to non-deterministic worlds with training rule

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

which is equivalent to

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

where

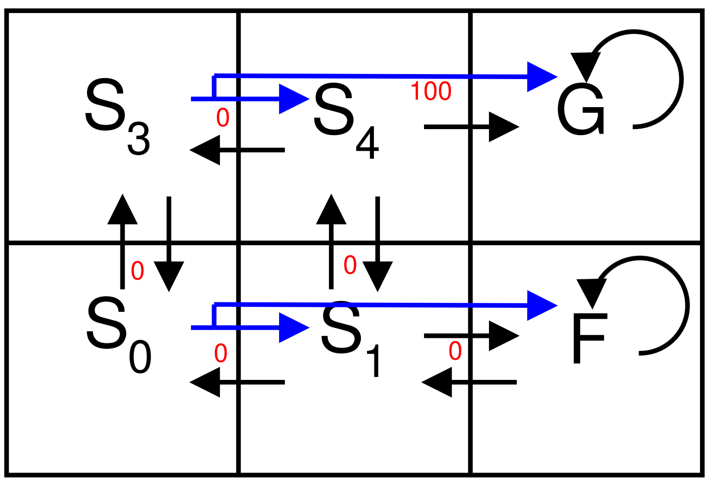
$$\alpha = \alpha_{n-1}(\mathbf{x}, a) = \frac{1}{1 + visits_{n-1}(\mathbf{x}, a)}$$

$visits_n(\mathbf{x}, a)$: total number of times state-action pair $(\mathbf{x}, a)$ has been visited up to n-th iteration

Convergence in non-deterministic MDP

- Deterministic Q-learning does not converge in non-deterministic worlds! $\hat{Q}_{n+1}(\mathbf{x}, a) \geq \hat{Q}_n(\mathbf{x}, a)$ is not valid anymore.
- Non-deterministic Q-learning also converges when every pair state, action is visited infinitely often [Watkins and Dayan, 1992].

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:

$$Q^{(1)}(\mathbf{x}_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(\mathbf{x}_{t+1}, a)$$

Two steps time difference:

$$Q^{(2)}(\mathbf{x}_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(\mathbf{x}_{t+2}, a)$$

n steps time difference:

$$Q^{(n)}(\mathbf{x}_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(\mathbf{x}_{t+n}, a)$$

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

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

Temporal Difference Learning

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

Equivalent expression:

$$Q^\lambda(\mathbf{x}_t, a_t) = r_t + \gamma[(1-\lambda)\max_a \hat{Q}(\mathbf{x}_t, a_t) + \lambda\ Q^\lambda(\mathbf{x}_{t+1}, a_{t+1})]$$

- $\lambda = 0$: $Q^{(1)}$ 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

$$Q^\lambda(\mathbf{x}_t, a_t) = r_t + \gamma[(1-\lambda)\max_a \hat{Q}(\mathbf{x}_t, a_t) + \lambda\ Q^\lambda(\mathbf{x}_{t+1}, a_{t+1})]$$

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 $< s, a, r, s', a' >$ ($< \mathbf{x}, a, r, \mathbf{x}', a' >$ in our notation).

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

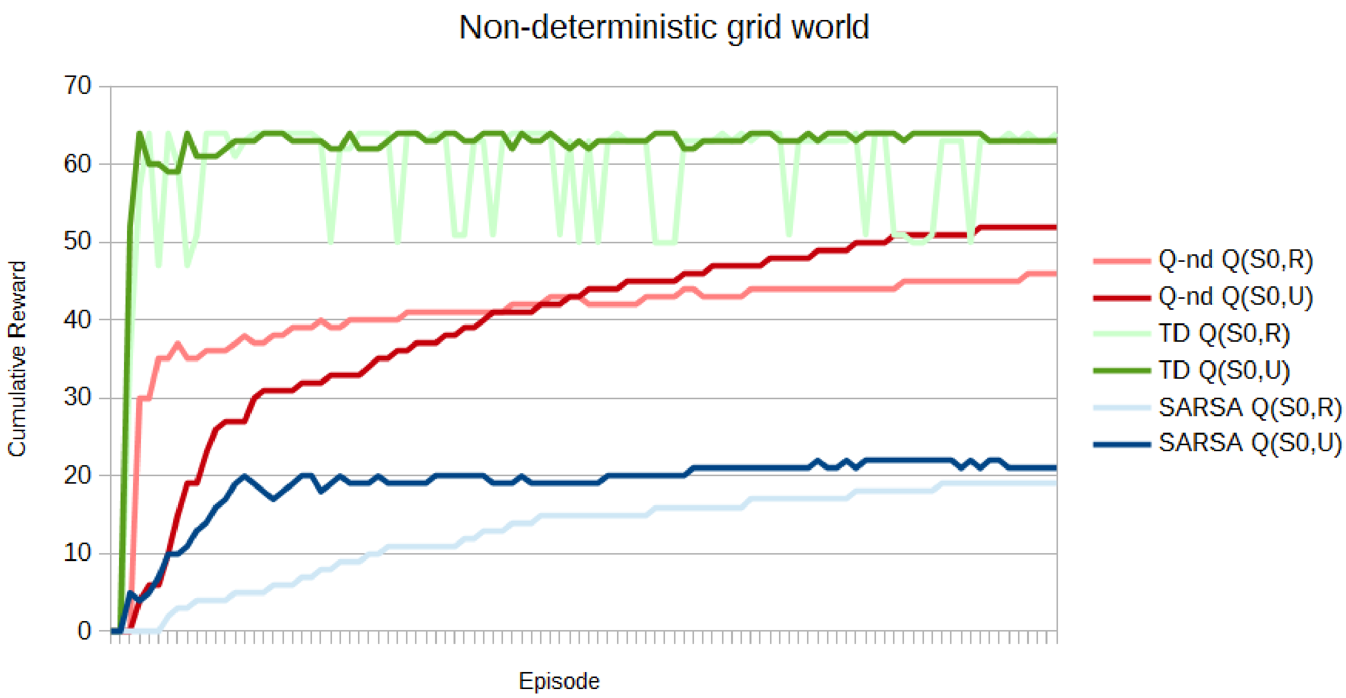
a' is chosen according to a policy based on current estimate of Q.

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.

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

Remarks on explicit representation of Q

Use function approximation:

$$Q_\theta(\mathbf{x}, a) = \theta_0 + \theta_1 F_1(\mathbf{x}, a) + \ldots + \theta_n F_n(\mathbf{x}, a)$$

Use linear regression to learn $Q_\theta(\mathbf{x}, a)$.

Remarks on explicit representation of Q

Use a neural network as function approximation and learn function Q with Backpropagation.

Implementation options:

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

TD-Gammon [Tesauro, 1995] uses a neural network and the Backpropagation algorithm together with TD(λ).

Reinforcement Learning with Policy Iteration

Use directly π instead of $V(\mathbf{x})$ or $Q(\mathbf{x}, a)$.

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

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

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

Policy Gradient Algorithm

Policy gradient algorithm for a parametric representation of the policy $\pi_\theta(\mathbf{x})$

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

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

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 $\pi \leftarrow \text{argmax}_{R_j} F(R_j)$ (i.e., hill climbing).

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

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 $\{-\epsilon_j, 0, +\epsilon_j\}$, and ϵ_j is a small fixed value relative to 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

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

Sapienza University of Rome, Italy - Machine Learning (2019/2020)

Policy Gradient Algorithm

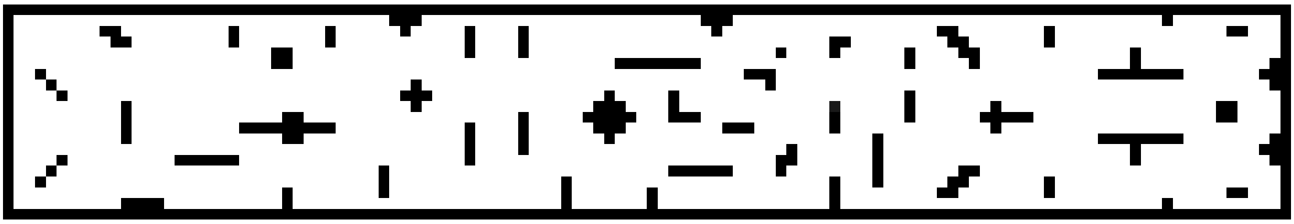
Task: optimize AIBO gait for fast and stable locomotion [Saggar et al., 2006]
Objective function F

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

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: non-deterministic grid controller

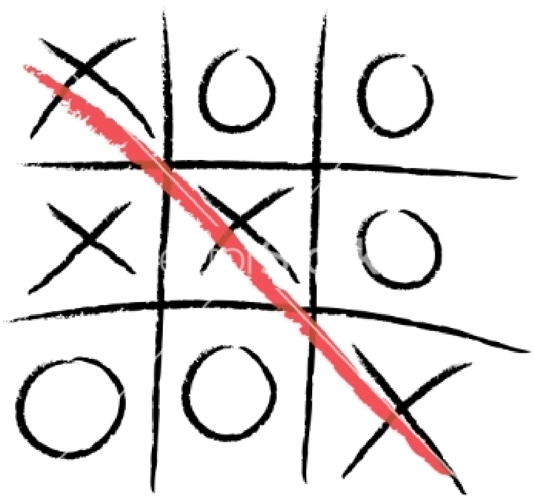
Reaching the right-most side of the environment from any initial state.



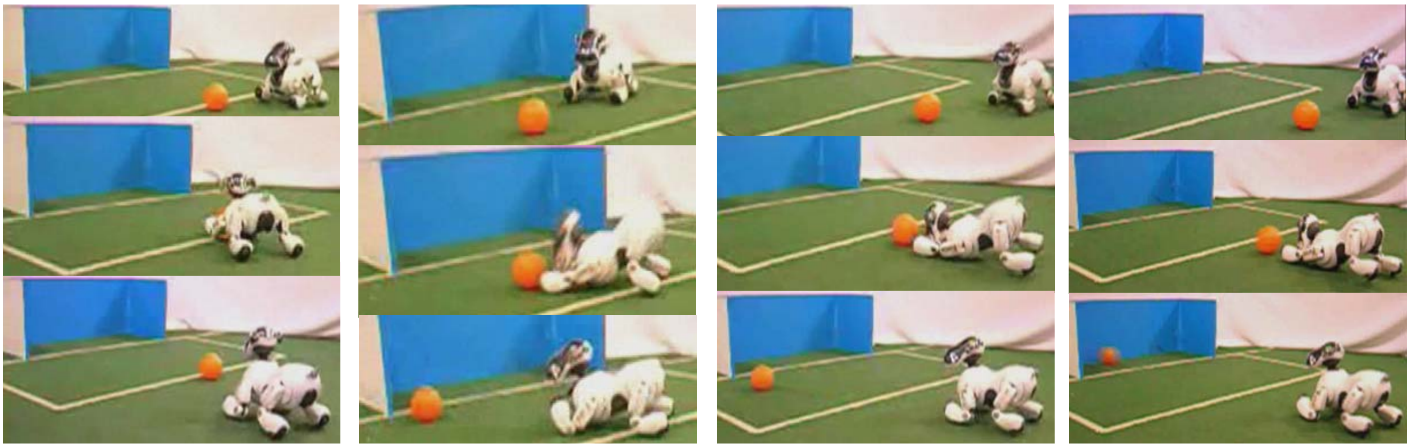
MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$

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

Example: Tic-Tac-Toe



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. 98109, Springer Verlag, 2007.