Sapienza University of Rome

Master in Artificial Intelligence and Robotics

Machine Learning

A.Y. 2024/2025

Prof. Luca Iocchi

Sapienza University of Rome, Italy - Machine Learning (2024/2025)

17. Markov Decision Processes and Reinforcement Learning

Luca Iocchi

Overview

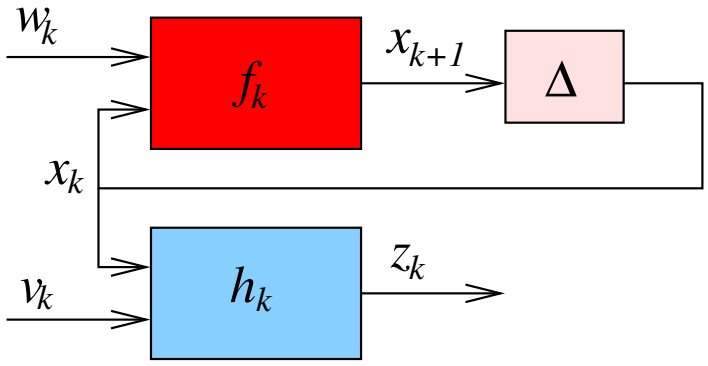
- Dynamic systems
- Markov Decision Processes
- Q-Learning
- Experimentation strategies
- Evaluation
- MDP Non-deterministic case
- Non-deterministic Q-Learning
- Temporal Difference
- SARSA
- Policy gradient algorithms

References

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. http://incompleteideas.net/book/the-book.html

Dynamic System

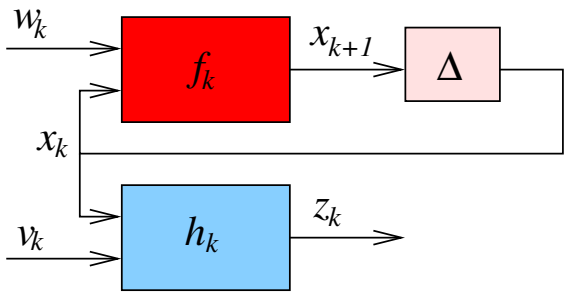
The classical view of a dynamic system



x: state
z: observations
w, v: noise

f: state transition model
h: observation model

Reasoning vs. Learning in Dynamic Systems



Reasoning: given the model (f, h) and the current state x_k, predict the future (x_{k+T}, z_{k+T}).

Learning: given past experience ($z_{0:k}$), determine the model (f, h).

State of a Dynamic System

The state x encodes:

- all the past knowledge needed to predict the future
- the knowledge gathered through operation
- the knowledge needed to pursue the goal

Examples:

- configuration of a board game
- configuration of robot devices
- screenshot of a video-game

Observability of the state

When the state is fully observable, the decision making problem for an agent is to decide which *action* must be executed in a given *state*.

The agent has to compute the function

$$\pi : \mathbf{X} \mapsto A$$

When the model of the system is not known, the agent has to *learn* the function π.

Supervised vs. Reinforcement Learning

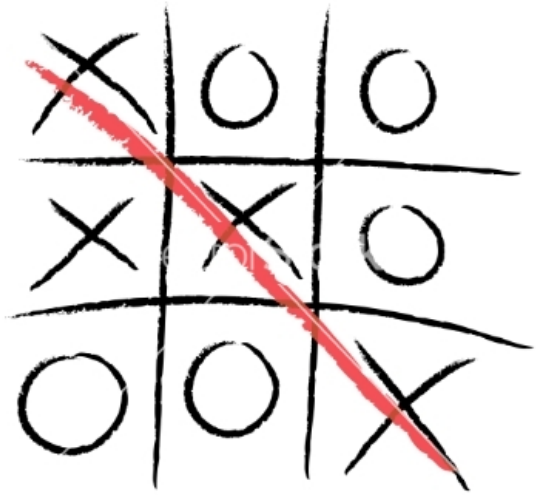
Supervised Learning
Learning a function $f : X \rightarrow Y$, given

$$D = \{\langle x_i, y_i \rangle\}$$

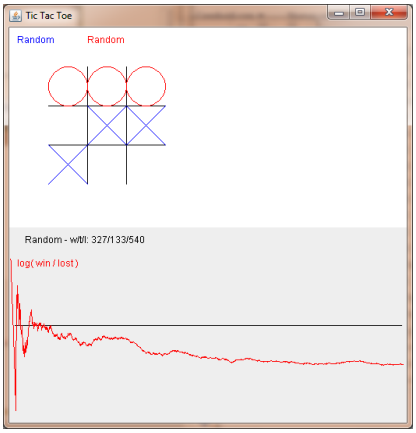
Reinforcement Learning
Learning a behavior function $\pi : \mathbf{X} \rightarrow A$, given

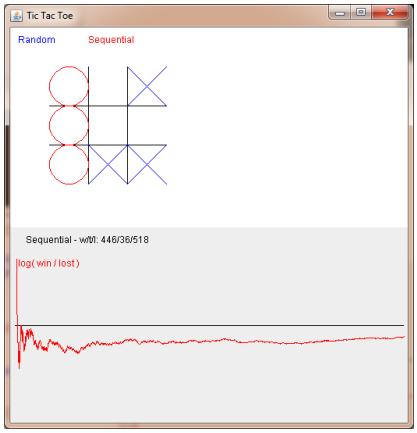
$$D = \{\langle \mathbf{x}_1, a_1, r_1, \ldots, \mathbf{x}_n, a_n, r_n \rangle^{(i)}\}$$

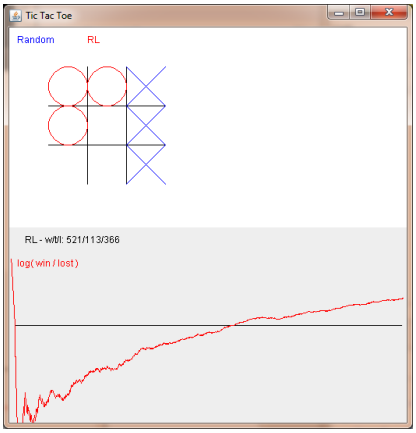
Example: Tic-Tac-Toe



Example: Tic Tac Toe





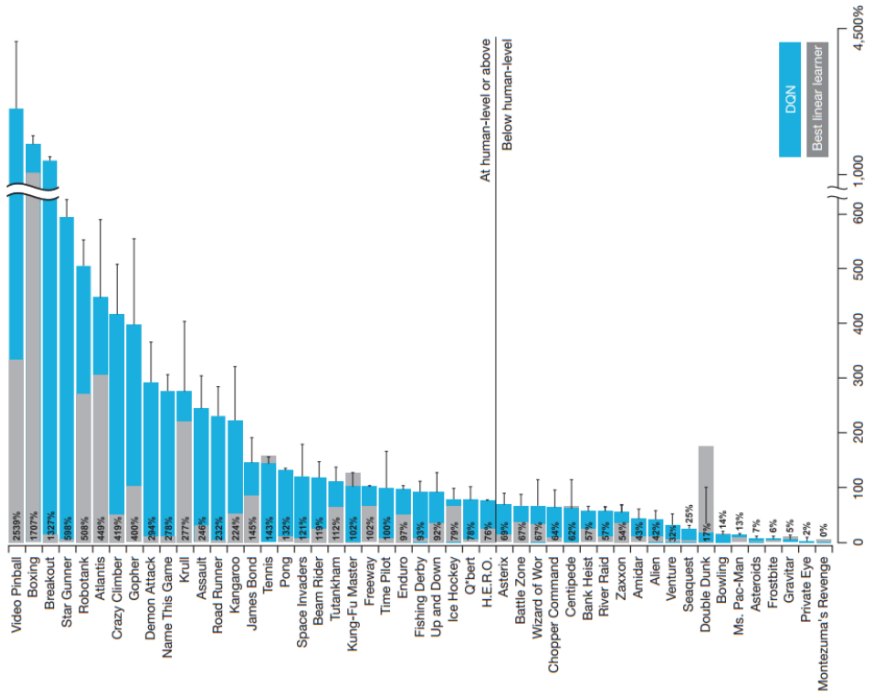


RL Example: Humanoid Walk

RL Example: Controlling an Helicopter

Deep Reinforcement Learning: ATARI games

Deep Q-Network (DQN) by Deep Mind (Google)



Human-level control through Deep Reinforcement Learning, Nature (2015)

Dynamic System Representation

X: set of states

- explicit discrete and finite representation $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
- continuous representation $\mathbf{X} = F(\ldots)$ (state function)
- probabilistic representation $P(\mathbf{X})$ (probabilistic state function)

A: set of actions

- explicit discrete and finite representation $\mathbf{A} = \{a_1, \ldots, a_m\}$
- continuous representation $\mathbf{A} = U(\ldots)$ (control function)

δ: transition function

- deterministic / non-deterministic / probabilistic

Z: set of observations

- explicit discrete and finite representation $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$
- continuous representation $\mathbf{Z} = Z(\ldots)$ (observation function)
- probabilistic representation $P(\mathbf{Z})$ (probabilistic observation function)

Markov property

Markov property

- Once the current state is known, the evolution of the dynamic system does not depend on the history of states, actions and observations.
- The current state contains all the information needed to predict the future.
- Future states are conditionally independent of past states and past observations given the current state.
- The knowledge about the current state makes past, present and future observations statistically independent.

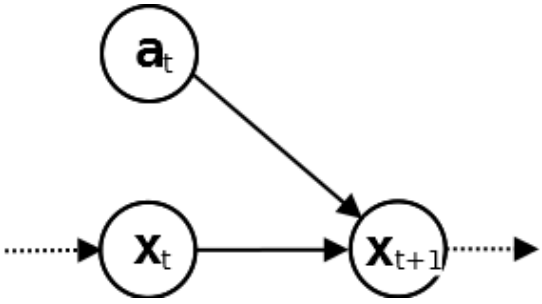
Markov process is a process that has the Markov property.

Markov Decision Processes (MDP)

Markov processes for decision making.

States are fully observable, no need of observations.

Graphical model



Markov Decision Processes (MDP)

Deterministic transitions

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

- $\mathbf{X}$ is a finite set of states
- $\mathbf{A}$ is a finite set of actions
- $\delta : \mathbf{X} \times \mathbf{A} \rightarrow \mathbf{X}$ is a transition function
- $r : \mathbf{X} \times \mathbf{A} \rightarrow \Re$ is a reward function

Markov property: $\mathbf{x}_{t+1} = \delta(\mathbf{x}_t, a_t)$ and $r_t = r(\mathbf{x}_t, a_t)$
Sometimes, the reward function is defined as $r : \mathbf{X} \rightarrow \Re$

Markov Decision Processes (MDP)

Non-deterministic transitions

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

- $\mathbf{X}$ is a finite set of states
- $\mathbf{A}$ is a finite set of actions
- $\delta : \mathbf{X} \times \mathbf{A} \rightarrow 2^{\mathbf{X}}$ is a transition function
- $r : \mathbf{X} \times \mathbf{A} \times \mathbf{X} \rightarrow \Re$ is a reward function

Markov Decision Processes (MDP)

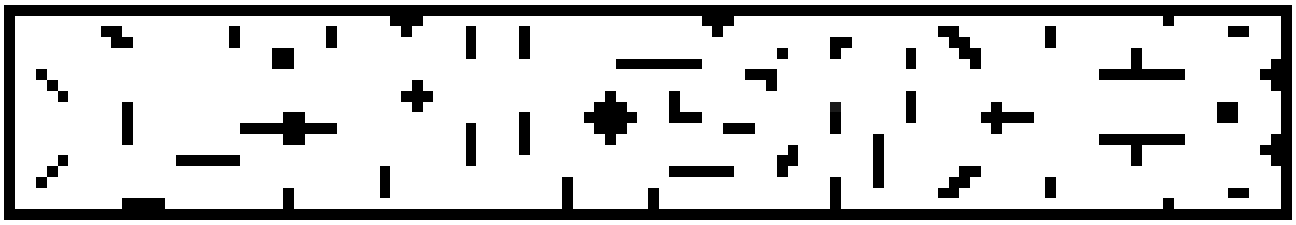
Stochastic transitions

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

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

Example: deterministic grid controller

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

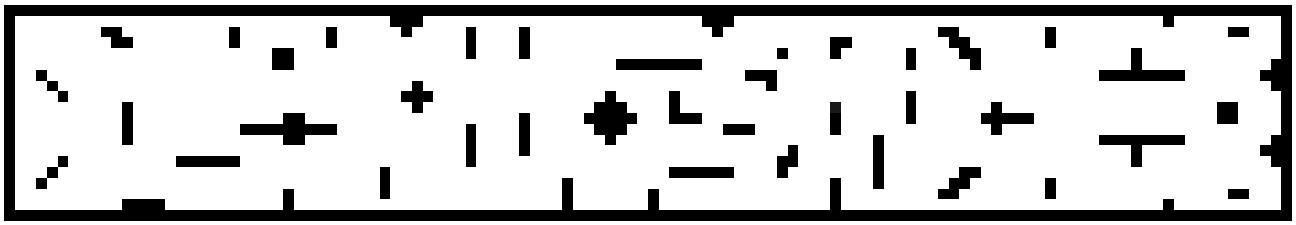


MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$

- $\mathbf{X} = \{(r, c)|\text{coordinates in the grid}\}$
- $\mathbf{A} = \{Left, Right, Up, Down\}$
- δ: cardinal movements with deterministic effects (if destination state is a black square, the agent remains in the current state)
- r: 1000 for reaching the right-most column, -10 for hitting any obstacle, 0 otherwise

Example: stochastic grid controller

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



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

- $\mathbf{X} = \{(r, c) | \text{coordinates in the grid}\}$
- $\mathbf{A} = \{\mathit{Left}, \mathit{Right}\}$
- δ: cardinal movements with non-deterministic effects (if destination state is a black square, the agent remains in the current state) (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.

Full Observability in MDP

States are fully observable.

Fully observe current state, may not predict future states.

In presence of non-deterministic or stochastic actions, the state resulting from the execution of an action is **not known before** the execution of the action, but it can be **fully observed after** its execution.

MDP Solution Concept

Given an MDP, we want to find an *optimal policy*.

Policy is a function

$$\pi : \mathbf{X} \to \mathbf{A}$$

For each state $\mathbf{x} \in \mathbf{X}$, $\pi(\mathbf{x}) \in \mathbf{A}$ is the *optimal* action to be executed in such state.

optimality = maximize the cumulative reward.

MDP Solution Concept

Optimality is defined with respect to maximizing the (expected value of the) *cumulative discounted reward*.

$$V^{\pi}(\mathbf{x}_1) \equiv E[\bar{r}_1 + \gamma \bar{r}_2 + \gamma^2 \bar{r}_3 + \ldots]$$

where $\bar{r}_t = r(\mathbf{x}_t, a_t, \mathbf{x}_{t+1})$, $a_t = \pi(\mathbf{x}_t)$, and $\gamma \in [0, 1]$ is the discount factor for future rewards.

Optimal policy: $\pi^* \equiv \text{argmax}_{\pi}\, V^{\pi}(\mathbf{x})\,,\ \forall \mathbf{x} \in \mathbf{X}$

Value function

Deterministic case

$$V^{\pi}(\mathbf{x}) \equiv r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

Non-deterministic/stochastic case:

$$V^{\pi}(\mathbf{x}) \equiv E[r_1 + \gamma r_2 + \gamma^2 r_3 + \dots]$$

Optimal policy

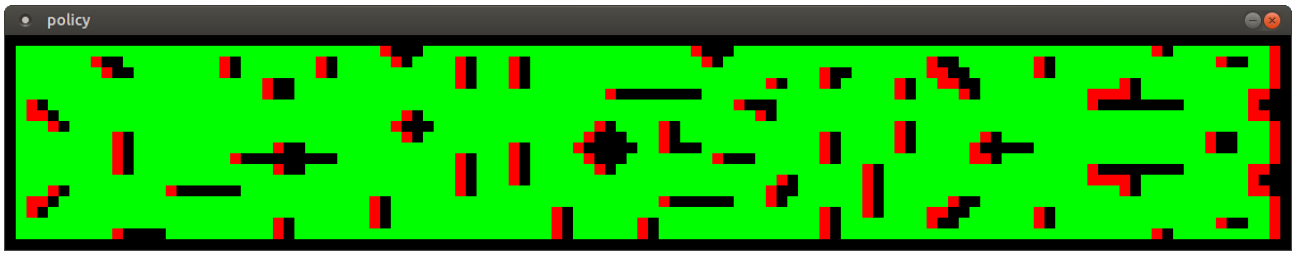
π^* is an **optimal policy** iff for any other policy π

$$V^{\pi^*}(\mathbf{x}) \geq V^{\pi}(\mathbf{x}), \forall \mathbf{x}$$

For infinite horizon problems, a stationary MDP always has an optimal stationary policy.

Example: non-deterministic grid controller

Optimal policy



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

Reasoning and Learning in MDP

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

Solution: Policy $\pi : \mathbf{X} \rightarrow \mathbf{A}$

If the MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$ is completely known $\rightarrow$ reasoning or planning

If the MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$ is not completely known $\rightarrow$ learning

Note: Planning in MDP can be solved with Value Iteration or Policy Iteration algorithms.

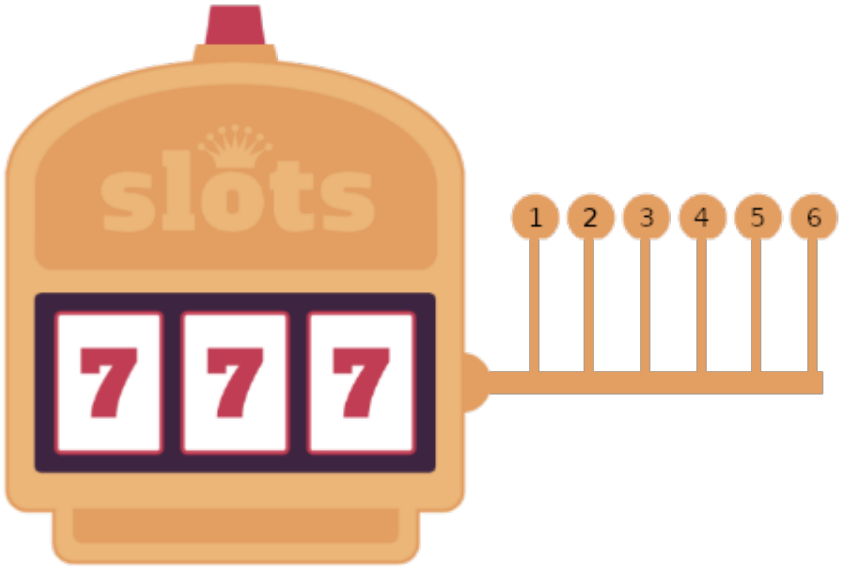
One-state Markov Decision Processes (MDP)

$$MDP = \langle \{\mathbf{x}_0\}, \mathbf{A}, \delta, r \rangle$$

- $\mathbf{x}_0$ unique state
- $\mathbf{A}$ finite set of actions
- $\delta(\mathbf{x}_0, a_i) = \mathbf{x}_0, \ \forall a_i \in \mathbf{A}$ transition function
- $r(\mathbf{x}_0, a_i, \mathbf{x}_0) = r(a_i)$ reward function

Optimal policy: $\pi^*(\mathbf{x}_0) = a_i$

One-state MDP



$$MDP = \langle \{\mathbf{x}_0\}, \{a_1, \ldots, a_n\}, \delta, r \rangle$$

$r(a_i)$: money won when executing action a_i

Deterministic One-state MDP

If $r(a_i)$ is **deterministic** and **known**, then

Optimal policy: $\pi^*(\mathbf{x}_0) = \text{argmax}_{a_i \in \mathbf{A}} r(a_i)$

What if reward function r is unknown?

Deterministic One-state MDP

If $r(a_i)$ is **deterministic** and **unknown**, then

Algorithm:

1. for each $a_i \in \mathbf{A}$
   - **execute** a_i and **collect** reward $r_{(i)}$
2. Optimal policy: $\pi^*(\mathbf{x}_0) = a_i$, with $i = \text{argmax}_{i=1...|\mathbf{A}|} r_{(i)}$

Note: exactly $|\mathbf{A}|$ iterations are needed.

Non-Deterministic One-state MDP

If $r(a_i)$ is **non-deterministic** and **known**, then

Optimal policy: $\pi^*(\mathbf{x}_0) = \text{argmax}_{a_i \in \mathbf{A}} E[r(a_i)]$

Example:

If $r(a_i) = \mathcal{N}(\mu_i, \sigma_i)$, then

$\pi^*(\mathbf{x}_0) = a_i$, with $i = \text{argmax}_{i=1\ldots|\mathbf{A}|} \mu_i$

Non-Deterministic One-state MDP

If $r(a_i)$ is **non-deterministic** and **unknown**, then

Algorithm:

1. Initialize a data structure Θ
2. For each time $t = 1, \ldots, T$ (until termination condition)
   - **choose** an action $a_{(t)} \in \mathbf{A}$
   - **execute** $a_{(t)}$ and **collect** reward $r_{(t)}$
   - Update the data structure Θ
3. Optimal policy: $\pi^*(\mathbf{x}_0) = \ldots$, according to the data structure Θ

Note: **many** iterations ($T >> |\mathbf{A}|$) are needed.

Non-Deterministic One-state MDP

Example:

If $r(a_i)$ is **non-deterministic** and **unknown** and $r(a_i) = \mathcal{N}(\mu_i, \sigma_i)$, then

Algorithm:

1. Initialize $\Theta_{(0)}[i] \leftarrow 0$ and $c[i] \leftarrow 0$, $i = 1...|\mathbf{A}|$
2. For each time $t = 1, \ldots, T$ (until termination condition)
   - **choose** an index $\hat{\imath}$ for action $a_{(t)} = a_{\hat{\imath}} \in \mathbf{A}$
   - **execute** $a_{(t)}$ and **collect** reward $r_{(t)}$
   - increment $c[\hat{\imath}]$
   - update $\Theta_{(t)}[\hat{\imath}] \leftarrow \frac{1}{c[\hat{\imath}]} \left(r_{(t)} + (c[\hat{\imath}] - 1)\Theta_{(t-1)}[\hat{\imath}] \right)$
3. Optimal policy: $\pi^*(\mathbf{x}_0) = a_i$, with $i = \operatorname{argmax}_{i=1...|\mathbf{A}|} \Theta_{(T)}[i]$

Experimentation Strategies

How actions are chosen by the agents?
Exploitation: select action a that maximizes $\hat{Q}(\mathbf{x}, a)$
Exploration: select random action a (with low value of $\hat{Q}(\mathbf{x}, a)$)

ϵ-greedy strategy

Given, $0 \leq \epsilon \leq 1$,
select a random action with probability ϵ
select the best action with probability $1 - \epsilon$

ϵ can decrease over time (first exploration, then exploitation).

Experimentation Strategies

soft-max strategy

actions with higher $\hat{Q}$ values are assigned higher probabilities, but every action is assigned a non-zero probability.

$$P(a_i|\mathbf{x}) = \frac{k^{\hat{Q}(\mathbf{x},a_i)}}{\sum_j k^{\hat{Q}(\mathbf{x},a_j)}}$$

$k > 0$ determines how strongly the selection favors actions with high $\hat{Q}$ values.
k may increase over time (first exploration, then exploitation).

Learning with Markov Decision Processes

Given an agent accomplishing a task according to an MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$, for which functions δ and r are **unknown** to the agent,

determine the optimal policy π^*

Note: This is not a supervised learning approach!

- Target function is $\pi : \mathbf{X} \rightarrow \mathbf{A}$
- but we do not have training examples $\{(\mathbf{x}_{(i)}, \pi(\mathbf{x}_{(i)}))\}$
- training examples are in the form $\langle (\mathbf{x}_{(1)}, a_{(1)}, r_{(1)}), \ldots, (\mathbf{x}_{(t)}, a_{(t)}, r_{(t)}) \rangle$

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

Choose a_i with ϵ-greedy strategy:
uniform random choice with prob. ϵ (exploration),
best choice with probability $1 - \epsilon$ (exploitation).

Training rule:

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

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

with $v_{n-1}(a_i)$ = number of executions of action a_i 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. $\mu_i \pm 10\%$ at unknown instants of time (with much lower frequency with respect to trials) ?

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

$\alpha =$ constant

MDP Learning Task

Let's now consider also the set of states **X** and the state evolution (unknown to the agent).

Since δ and r are not known, the agent cannot predict the effect of its actions. But it can execute them and then observe the outcome.

The learning task is thus 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

- Value iteration
  (estimate the Value function and then compute π)
- Policy iteration
  (estimate directly π)

Learning through value iteration

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

From which it could determine the optimal policy:

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

However, this policy cannot be computed in this way because δ and r are not known.

Q Function (deterministic case)

$Q^\pi(\mathbf{x}, a)$: expected value when executing a in the state $\mathbf{x}$ and then act according to π.

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

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

If the agent learns Q, then it can determine the optimal policy without knowing δ and r.

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

Q Function (deterministic case)

Observe that

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

Thus, we can rewrite

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

as

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

Training Rule to Learn Q (deterministic case)

Deterministic case:

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

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

Training rule:

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

where $\bar{r}$ is the immediate reward and $\mathbf{x}'$ is the state resulting from applying action a in state $\mathbf{x}$.

Q Learning Algorithm for Deterministic MDPs

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

$$\hat{Q}_{(t)}(\mathbf{x}, a) \leftarrow \bar{r} + \gamma \max_{a' \in \mathbf{A}} \hat{Q}_{(t-1)}(\mathbf{x}', a')$$

   - $\mathbf{x} \leftarrow \mathbf{x}'$
4. Optimal policy: $\pi^*(\mathbf{x}) = \operatorname{argmax}_{a \in \mathbf{A}} \hat{Q}_{(T)}(\mathbf{x}, a)$

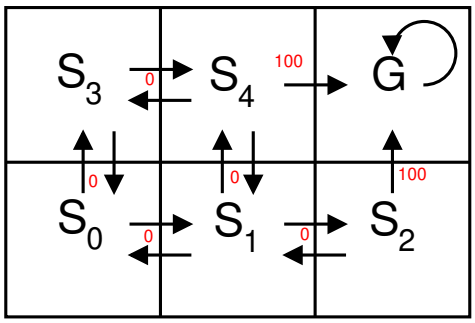
Note: not using δ and r, but just observing new state $\mathbf{x}'$ and immediate reward $\bar{r}$, after the execution of the chosen action.

Convergence in deterministic MDP

- $\hat{Q}_n(\mathbf{x}, a)$ underestimates $Q(\mathbf{x}, a)$
- We have: $0 \leq \hat{Q}_n(\mathbf{x}, a) \leq \hat{Q}_{n+1}(\mathbf{x}, a) \leq Q(\mathbf{x}, a)$
- Convergence guranteed if all state-action pairs visited infinitely often

Example: grid world

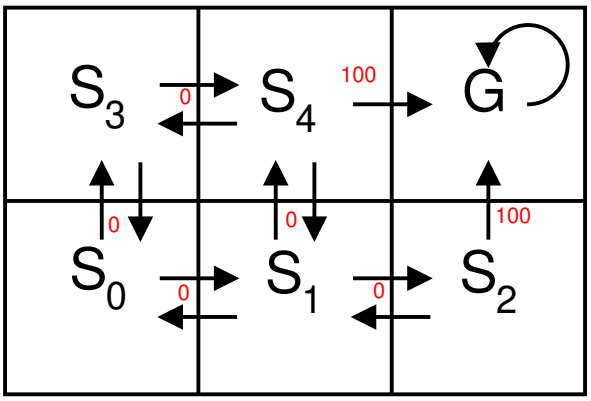
Reaching the goal state G from initial state S_0.



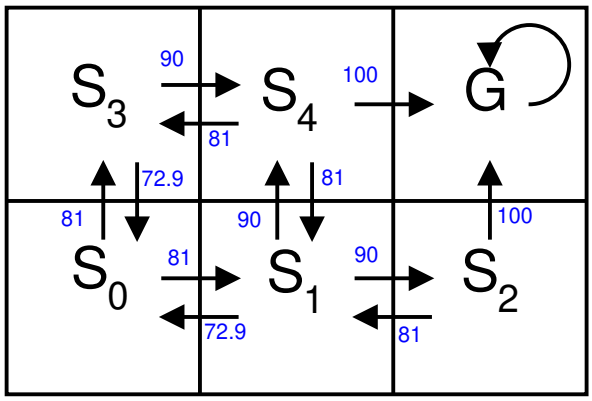
MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$

- $\mathbf{X} = \{S_0, S_1, S_2, S_3, S_4, G\}$
- $\mathbf{A} = \{L, R, U, D\}$
- δ represented as arrows in the figure (e.g., $\delta(S_0, R) = S_1$)
- $r(\mathbf{x}, a)$ represented as red values on the arrows in the figure (e.g., $r(S_0, R) = 0$)

Example: grid world

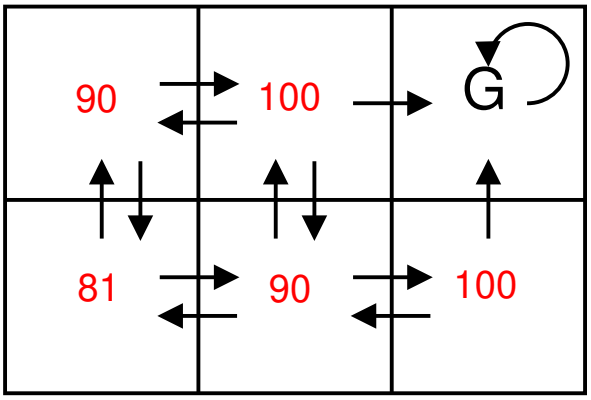


MDP

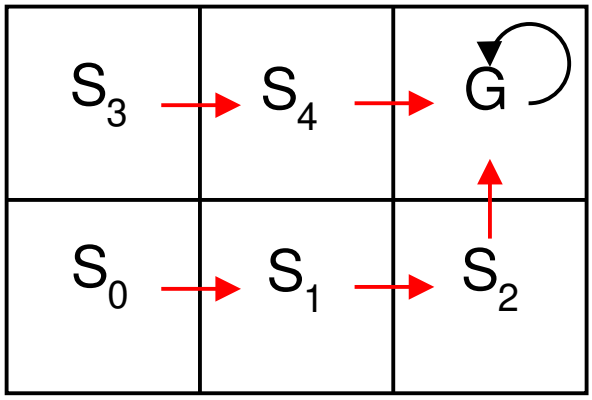


$\hat{Q}$

Example: Grid World

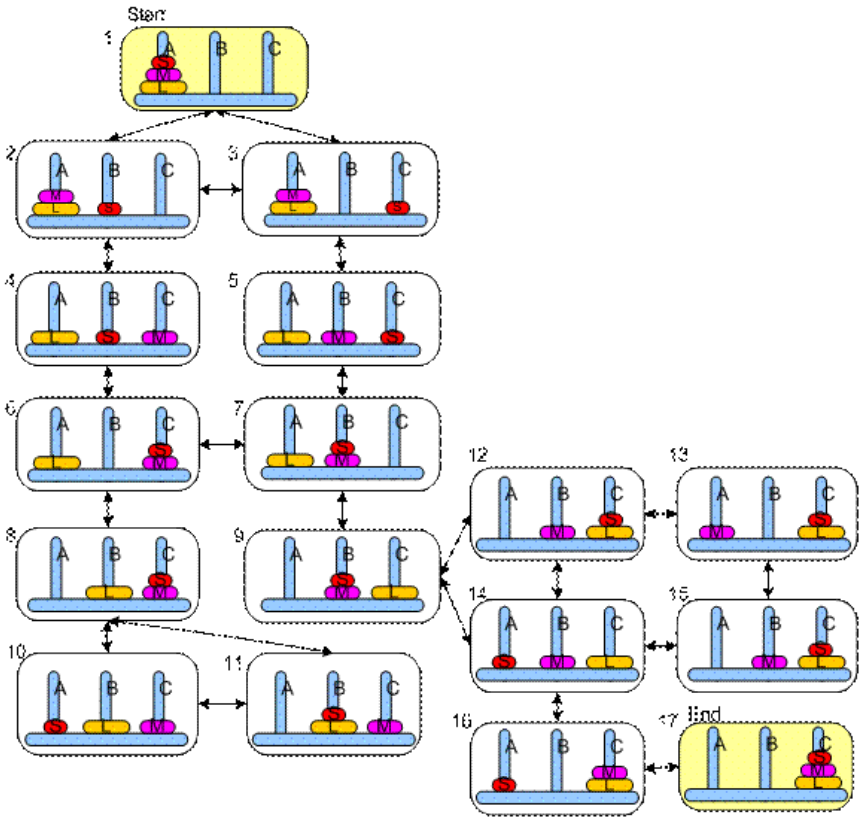


$V^*(\mathbf{x})$ values



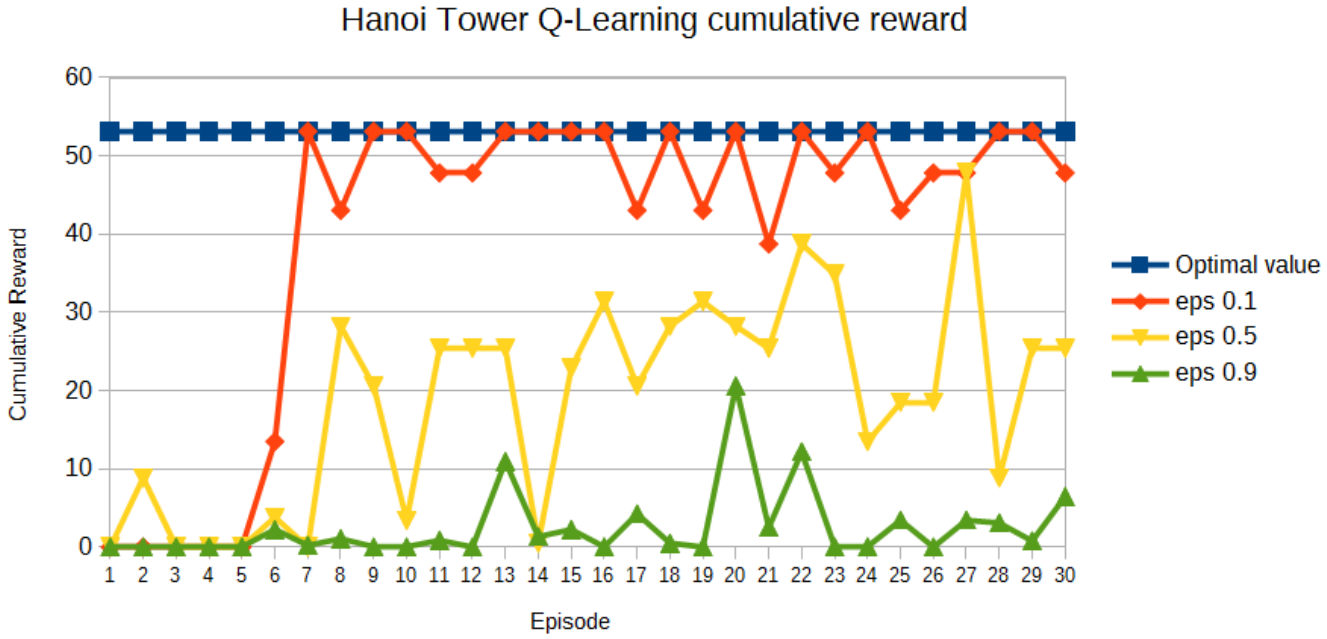
One optimal policy

Example: Hanoi Tower



Evaluating Reinforcement Learning Agents

Evaluation of RL agents is usually performed through the cumulative reward gained over time.



Evaluating Reinforcement Learning Agents

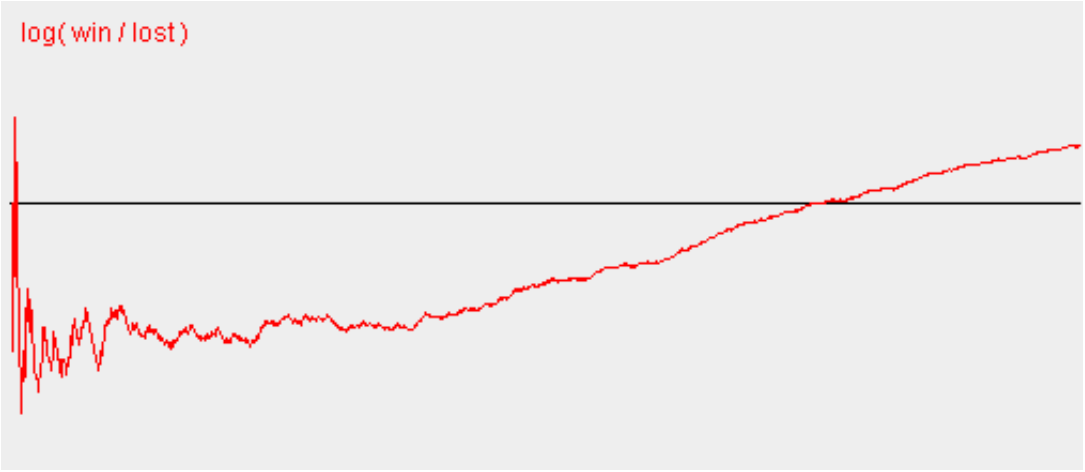
Cumulative reward plot may be very noisy (due to exploration phases).
A better approach could be:

Repeat until termination condition:

1. Execute k steps of learning
2. Evaluate the current policy π_k (average and stddev 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.

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

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

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

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

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

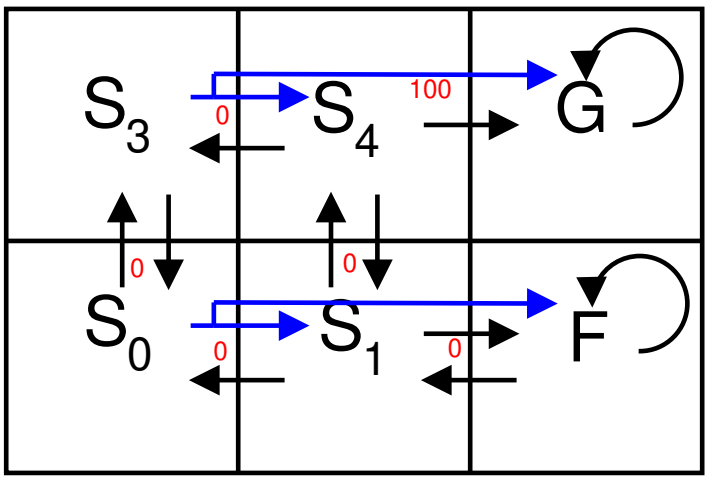
$$\alpha = \alpha_{n-1}(\mathbf{x}, a) = \frac{1}{1 + visits_{n-1}(\mathbf{x}, a)}$$

$visits_n(\mathbf{x}, a)$: 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

Estimate the Q function: $\hat{Q}(\mathbf{x}, a) \approx Q(\mathbf{x}, a)$

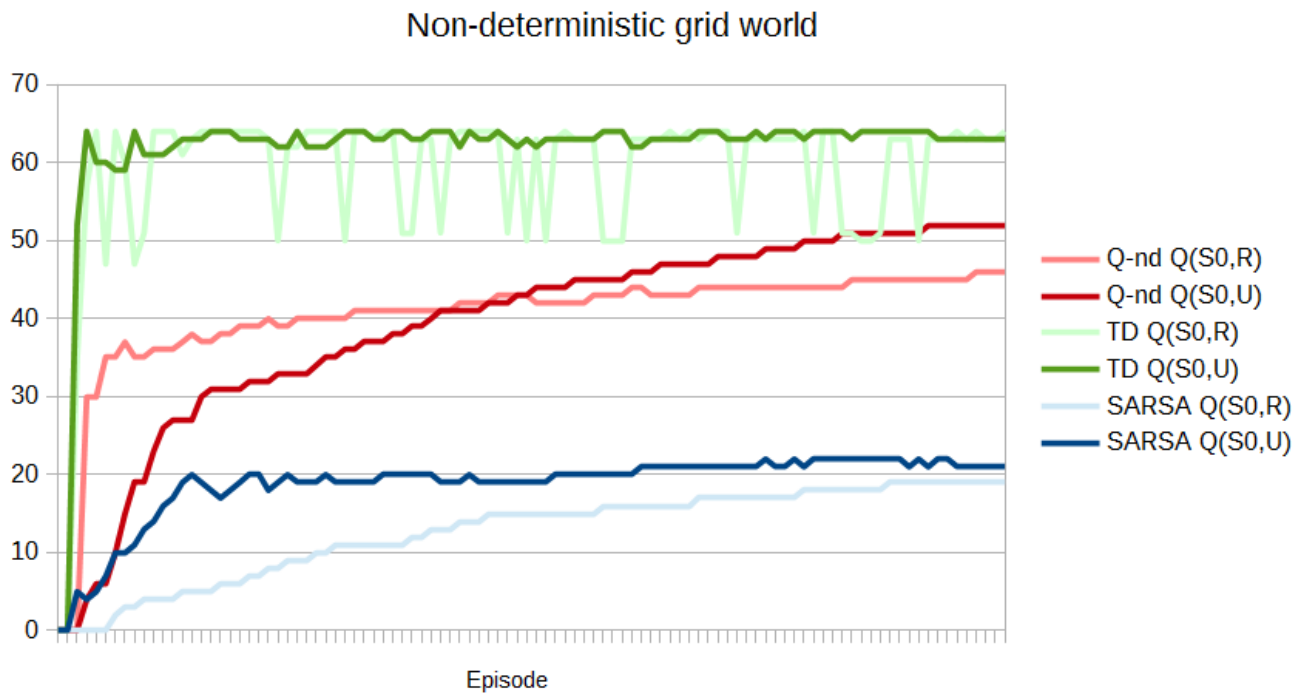
However, we just need

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

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:

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

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

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).

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

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

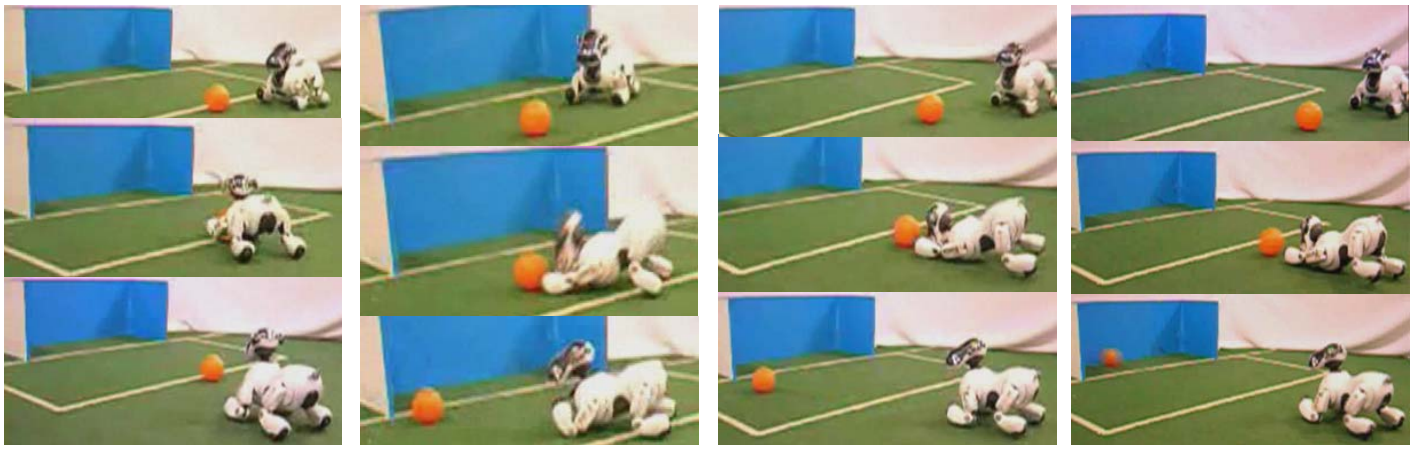
Policy Gradient Algorithm

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.

[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.