Sapienza University of Rome, Italy
Master in Computer Engineering

Markov Decision Processes and Reinforcement Learning

Luca Iocchi

Summary

Part I - Problem definition

- Motivating examples
- Markov Decision Processes (MDP)
- Solution concept
- One-state MDP
- Exercise: Multi-armed bandit

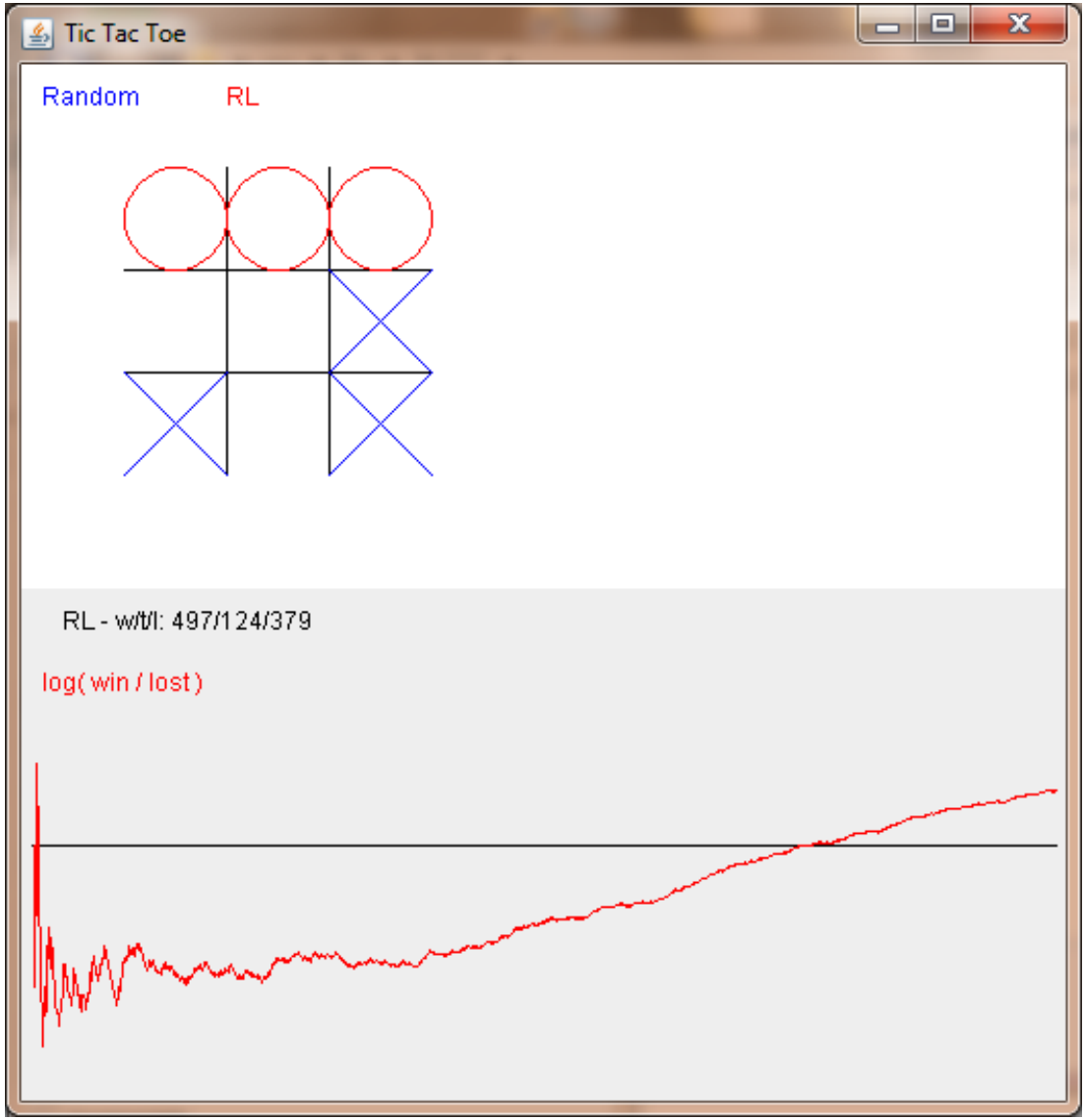
Part II - Algorithms

- Value iteration and policy iteration
- Q-Learning
- Sarsa
- Exercises: Grid world, Breakout

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

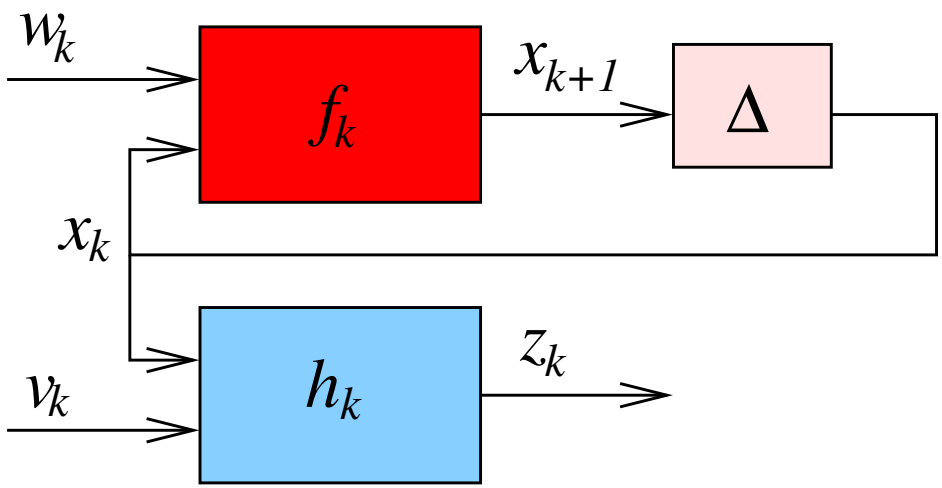
PART I - Problem definition

Motivating example: Tic-Tac-Toe



Dynamic System

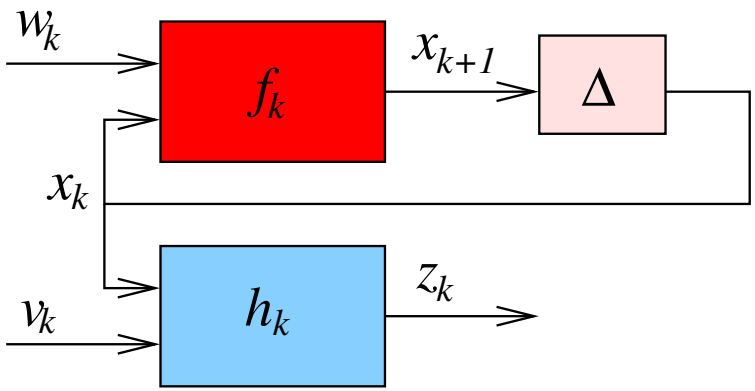
The classical view of a dynamic system



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:

- knowledge needed to predict the future
- knowledge gathered through operation
- 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*.
Let $\mathbf{X}$ be the set of all the possible states of our system.

Even when actions have non-deterministic effects (not possible to predict the outcome before their execution), full observability allows the agent to always know the current state (after the action has been executed).

Example: when playing chess, the agent cannot predict opponent's move, but it can observe it after it has been executed.

Solution concept

Given a finite set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of all the possible states of our system and a finite set $A = \{a_1, \ldots, a_m\}$ of all actions available to our agent, the goal of the agent (solution concept) is 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

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

Supervised vs. Reinforcement Learning

Collecting a data set for RL

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

is much easier, since a_i must not be the best action to be executed in x_i and rewards can be sporadic and given in the future.

Example: $r_i = 0, \forall i = 0, \ldots, n-1$, and $r_n \neq 0$, only for final states $\mathbf{x}_n$.

RL algorithms effectively solve the credit assignment problem (assignment of future rewards to sequences of actions).

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.

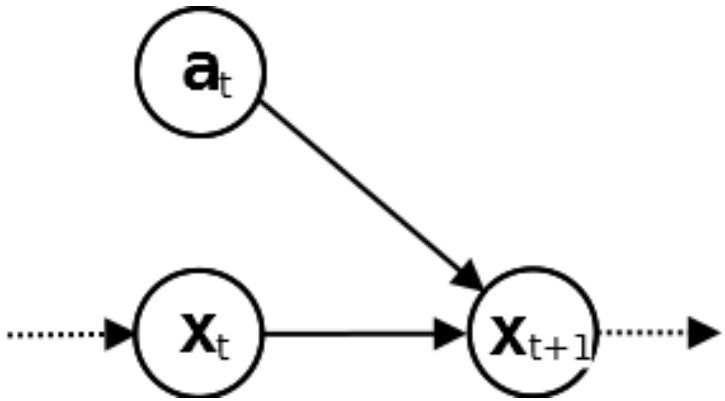
Markov process is a process that has the Markov property.

Markov Decision Processes (MDP)

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

- $\mathbf{X}$ 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

Graphical model



One-state Markov Decision Processes (MDP)

Multi-armed bandit

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

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

Deterministic One-state MDP

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

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

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
   - **collect** reward $r_{(i)}$ and store it
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 $i = \text{argmax}_{i=1...|\mathbf{A}|} \mu_i$

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** $a_{(t)}$
   - **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 $\Theta_{(t)}[\hat{\imath}] \leftarrow \Theta_{(t-1)}[\hat{\imath}] + \frac{1}{c[\hat{\imath}]} \left(r_{(t)} - \Theta_{(t-1)}[\hat{\imath}] \right)$
3. Optimal policy: $\pi^*(\mathbf{x}_0) = a_i$, with $i = \operatorname{argmax}_{i=1...|\mathbf{A}|} \Theta_{(T)}[i]$

Exploitation vs Exploration

Exploitation: choose an action that is believed to be the best one in the current state

Exploration: choose a random action

A proper balance between exploration and exploitation is needed in order to realize an effective RL system.

Exercise

Multi-armed bandit

PART II - Algorithms

MDP Solution Concept

Given an MDP $\langle \mathbf{X}, A, \delta, r \rangle$, find an *optimal policy*.

Policy is a function

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

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

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

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.

Planning and Learning in MDP

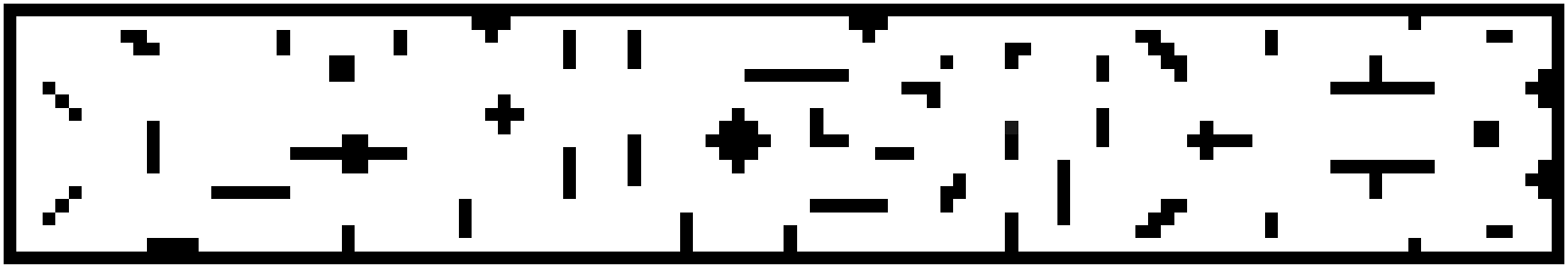
Problem: MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$

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

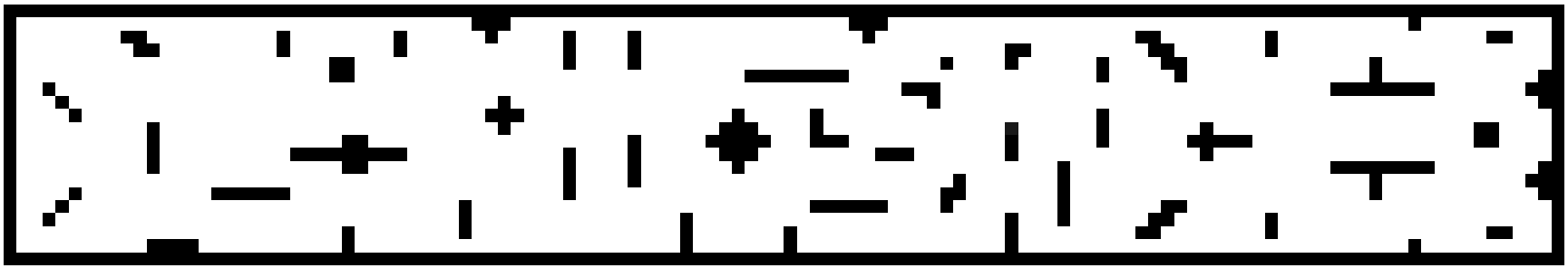
Example



An agent can move in this environment by just going left and right with non-deterministic effects of the actions and noisy sensors.

The goal is to reach the right side starting from the left side, avoiding hitting the obstacles.

Example



Markov Decision Process (MPD) with:

- states $\mathbf{x}_t = (r, c)$ coordinates in the world,
- observations (emissions) $\mathbf{z}_t = (\mathbf{z}_t^U, \mathbf{z}_t^D, \mathbf{z}_t^L, \mathbf{z}_t^R)$ noisy sensor of obstacles in the U,D,L,R directions,
- controls/actions $\mathbf{A} = \{Left, Right\}$,

Example: model of the problem

Transition probability:

$$P(\mathbf{x}_{t+1} = (r', c') | \mathbf{x}_t = (r, c), a = Right) =$$

$$= \begin{cases} 0, \textit{if outside the map} \\ 0, \textit{if} \, |r' - r| > 1 \textit{or} |c' - c| > 1 \\ \gamma_F, \textit{if} \, \neg \textit{obstacle}(r, c+1) \wedge c' = c+1 \wedge r' = r \\ \frac{1-\gamma_F}{2}, \textit{if} \, \neg \textit{obstacle}(r, c+1) \wedge c' = c+1 \wedge |r' - r| = 1 \\ \gamma_B, \textit{if obstacle}(r, c+1) \wedge c' = c-1 \wedge r' = r \\ \frac{1-\gamma_B}{2}, \textit{if obstacle}(r, c+1) \wedge c' = c-1 \wedge |r' - r| = 1 \end{cases}$$

Example: model of the problem

Observation probability:

$$\begin{aligned} P(\mathbf{z}_t|\mathbf{x}_t = (r,c)) &= P((\mathbf{z}_t^U, \mathbf{z}_t^D, \mathbf{z}_t^L, \mathbf{z}_t^R)|\mathbf{x}_t = (r,c)) \\ &= \prod_{\lambda} P(\mathbf{z}_t^\lambda|\mathbf{x}_t = (r,c)) \end{aligned}$$

$$P(\mathbf{z}_t^R|\mathbf{x}_t = (r,c)) = \begin{cases} \delta, \textit{if obstacle}(r, c+1) \\ 1-\delta, \textit{otherwise} \end{cases}$$

With similar terms for the other directions.

Example: reward function

$$R(\mathbf{x} = (r, c), a, \mathbf{x}' = (r', c')) = \begin{cases} 0, \textit{if } a = \textit{Right} \wedge \textit{obstacle}(r, c+1) \\ 0, \textit{if } a = \textit{Left} \wedge \textit{obstacle}(r, c-1) \\ c', \textit{if } a = \textit{Left} \\ c'^2, \textit{if } a = \textit{Right} \end{cases}$$

Example: solutions

Computing the policy $\pi : \mathbf{X} \mapsto \mathbf{A}$

MDP known
value iteration algorithm

MDP unknown
Reinforcement learning algorithm: SARSA

Planning with MDP

Given MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$ is completely known, compute optimal polici $\pi : \mathbf{x} \mapsto A$

Value iteration algorithms
Policy iteration algorithms

Value function (V)

Deterministic case

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

$$V^{\pi}_{(t)}(\mathbf{x}) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$V^{\pi}_{(t)}(\mathbf{x}) = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) = r_t + \gamma V^{\pi}_{(t+1)}(\mathbf{x}')$$

Non-deterministic/stochastic case:

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

$$\dots = \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, \pi(\mathbf{x}))(r_t + \gamma V^{\pi}(\mathbf{x}'))$$

$r_t = r(\mathbf{x}, \pi(\mathbf{x}), \mathbf{x}')$

Action-value function (Q)

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

$$Q^\pi(\mathbf{x}, a) \equiv \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a)(r(\mathbf{x}, a, \mathbf{x}') + \gamma V^\pi(\mathbf{x}'))$$

Thus,

$$V^\pi(\mathbf{x}) = Q^\pi(\mathbf{x}, \pi(\mathbf{x}))$$

V and Q for the optimal policy

For an optimal policy π^*

$$V^*(\mathbf{x}) \equiv V^{\pi^*}(\mathbf{x})$$

$$Q^*(\mathbf{x}, a) \equiv \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a)(r(\mathbf{x}, a, \mathbf{x}') + \gamma V^*(\mathbf{x}'))$$

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

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

Dynamic programming

Input: known MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$
Output: optimal policy π^*

Initialize $V_{(0)}(\mathbf{x})$ and $\pi_{(0)}(\mathbf{x})$ randomly
for $t = 1, \ldots, T$ // until a termination condition

for each $\mathbf{x} \in \mathbf{X}$*:*

1. $\hat{a} = \pi_{(t-1)}(\mathbf{x})$
2. $V_{(t)}(\mathbf{x}) \leftarrow \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, \hat{a}) \left(r(\mathbf{x}, \hat{a}, \mathbf{x}') + \gamma V_{(t-1)}(\mathbf{x}') \right)$
3. $\pi_{(t)}(\mathbf{x}) \leftarrow$
   $\operatorname{argmax}_{a' \in \mathbf{A}} \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a') \left(r(\mathbf{x}, a', \mathbf{x}') + \gamma V_{(t)}(\mathbf{x}') \right)$

return $\pi^*(\mathbf{x}) = \pi_{(T)}(\mathbf{x})$

Termination condition: no changes in π

Value Iteration

Input: known MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$
Output: optimal policy π^*

Initialize $V_{(0)}(\mathbf{x})$, $Q_{(0)}(\mathbf{x}, a)$ randomly (or to zero)
for $t = 1, \ldots, T$ // until a termination condition

1. $Q_{(t)}(\mathbf{x}, a) \leftarrow \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) \left(r(\mathbf{x}, a, \mathbf{x}') + \gamma V_{(t-1)}(\mathbf{x}')\right)$
2. $V_{(t)}(\mathbf{x}) \leftarrow \max_{a \in \mathbf{A}} Q_{(t)}(\mathbf{x}, a)$

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

Termination condition: $\forall \mathbf{x}\,,\ V_{(t)}(\mathbf{x}) - V_{(t-1)}(\mathbf{x}) < \delta$

Policy Iteration

Input: known MDP $\langle \mathbf{X}, \mathbf{A}, \delta, r \rangle$
Output: optimal policy π^*

Initialize the policy $\pi_0(\mathbf{x})$ randomly
for $t = 1, \ldots, T$ // until a termination condition

1. Solve the linear system in $V(\mathbf{x})$:
   $V(\mathbf{x}) = \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, \pi_{(t-1)}(\mathbf{x})) \left(r(\mathbf{x}, \pi_{(t-1)}(\mathbf{x}), \mathbf{x}') + \gamma V(\mathbf{x}')\right)$
2. $\pi_{(t)}(\mathbf{x}) \leftarrow \operatorname{argmax}_{a \in \mathbf{A}} \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) \left(r(\mathbf{x}, a, \mathbf{x}') + \gamma V(\mathbf{x}')\right)$

return $\pi^*(\mathbf{x}) = \pi_{(T)}(\mathbf{x})$

Termination condition: no changes in π

Learning with MDP

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 $\langle (\mathbf{x}_{(1)}, a_{(1)}, r_{(1)}), \ldots, (\mathbf{x}_{(t)}, a_{(t)}, r_{(t)}) \rangle$

Agent's Learning Task

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

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

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

$$Q^\pi(\mathbf{x}, a) \equiv \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a)(r(\mathbf{x}, a, \mathbf{x}') + \gamma V^\pi(\mathbf{x}'))$$

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

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

Training Rule to Learn Q

Deterministic case:

$$Q(\mathbf{x}_t, a_t) = r(\mathbf{x}_t, a_t) + \gamma \max_{a' \in \mathbf{A}} Q(\mathbf{x}_{t+1}, a')$$

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

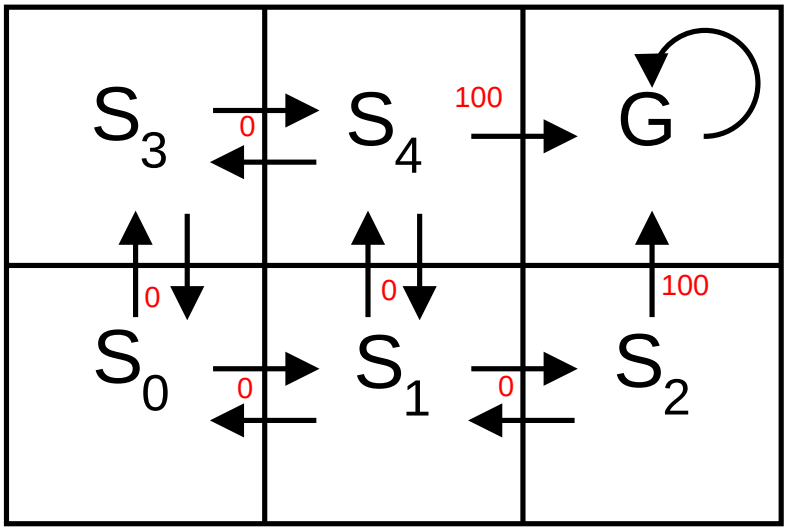
1. for each $\mathbf{x}, a$ initialize table entry $\hat{Q}_{(0)}(\mathbf{x}, a) \leftarrow 0$
2. observe current state $\mathbf{x}$
3. for each time $t = 1, \ldots, T$ (until termination condition)
   - **choose** an action a
   - **execute** the action a
   - **observe** the new state $\mathbf{x}'$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

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

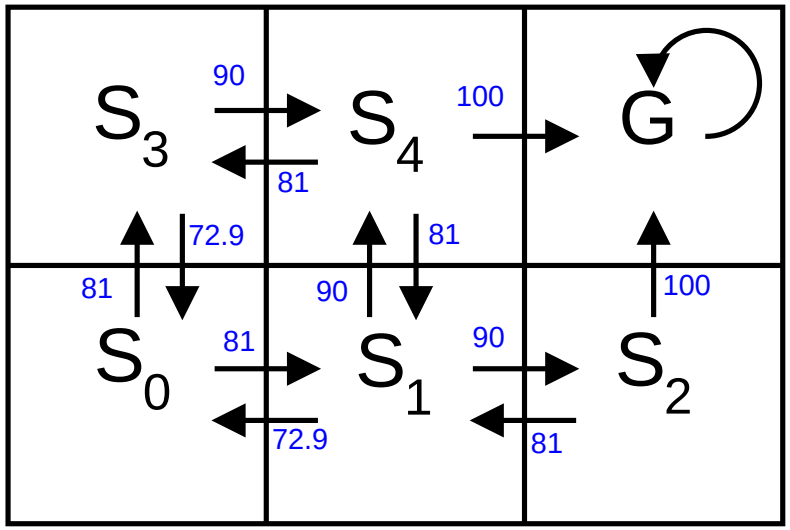
   - $\mathbf{x} \leftarrow \mathbf{x}'$
4. Optimal policy: $\pi^*(\mathbf{x}) = \mathrm{argmax}_{a \in \mathbf{A}}\, Q_{(T)}(\mathbf{x}, a)$

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.

Example: Grid World

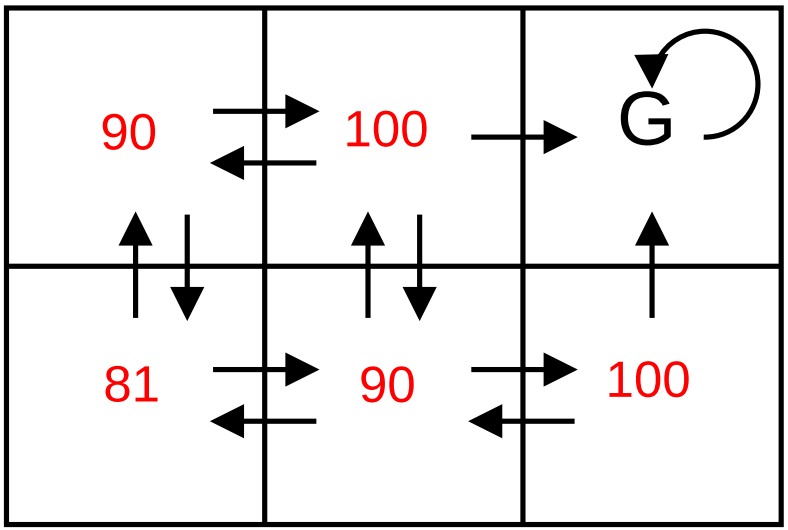


MDP

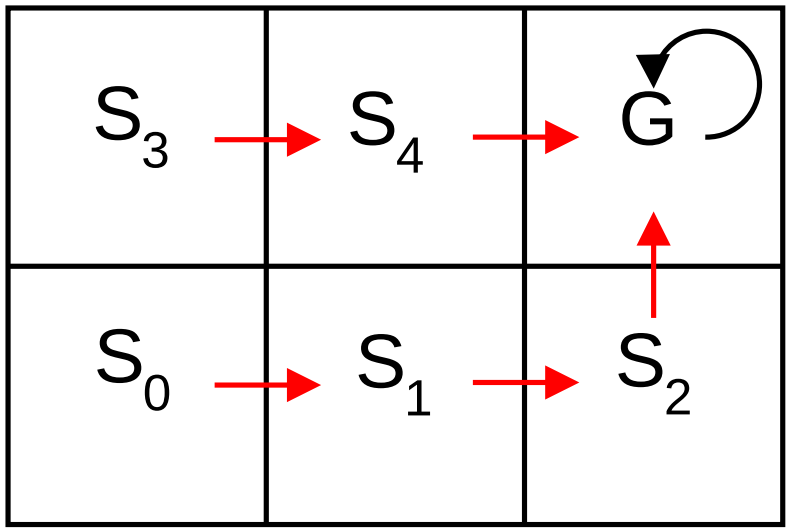


$\hat{Q}$

Example: Grid World



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



One optimal policy

Non-deterministic Q-learning

Q learning generalizes to non-deterministic worlds with training rule

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

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

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.

Experimentation Strategies

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

Convergence

RL algorithms find the optimal policy when every pair state, action is visited infinitely often.

The choice of the action to execute at each step is critical to find the optimal solution.

Sometimes sub-optimal solutions or average-good solutions are acceptable.

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)

Exercise

Algorithm comparison on grid world.