Introduzione al Reinforcement Learning

Maurizio Parton, Università di Chieti-Pescara

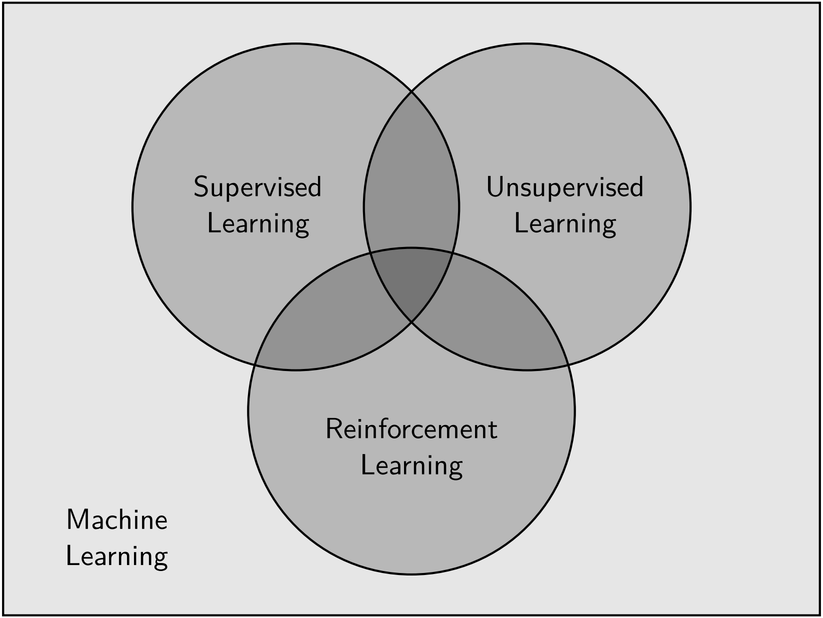
23 maggio 2019

Acknowledgements, resources and links

- Reinforcement Learning: An Introduction. Richard S. Sutton and Andrew G. Barto, second edition, 2018.
- UCL Course on RL, videos and slides. David Silver, 2015.
- Tutorial: Introduction to Reinforcement Learning with Function Approximation. Richard S. Sutton, 2016.
- Implementation of Reinforcement Learning algorithms. Denny Britz, GitHub project, 2016 (updated in 2018).

Both the organization and the content of the slides are extracted from David Silver's course and Richard S. Sutton tutorial.

Branches of Machine Learning



RL characteristics

What is RL?

- Agent-oriented learning: an agent learns by interacting with an environment to achieve a goal.
- The agent learns by trial and error, evaluating a (delayed) feedback.
- The kind of machine learning most like natural learning.
- Learning that can tell for itself when it is right or wrong.

RL vs SL and UL

- RL is not completely supervised: only reward.
- RL is not completely unsupervised: there is reward.
- Time matters: sequential data.
- Time matters: actions change possible future.

Let's play a game!

You are the learner

You live in a world where you can only do two things, called "1" and "2", and receiving a reward. . .

Examples

Real world applications of RL (original article)

- Resources management in computer clusters.
- Traffic light control.
- Robotics.
- Web system configuration.
- Chemistry.
- Personalized recommendations.
- Bidding and advertising.

Examples

Games

- AlphaGo's family.
- StarCraft II. Very recent achievement, 19 Dec 2018.
- Atari games. Very recent achievement, 28 Sep 2018.
- TD-Gammon.

Enjoy few minutes of video

- Atari:
  https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en
- AlphaGo:
  https://www.youtube.com/watch?v=8dMFJpEGNLQ
- StarCraft: https://youtu.be/UuhECwm31dM

1. Introduction
2. The RL setup: problem, actors, MDP framework
3. Prediction and control via Bellman equations
4. Putting things together: Monte Carlo learning
5. Turning tables to approximation

The RL problem

RL main task

Decision problem: choose actions that maximize the *return*, i.e. the total future reward.

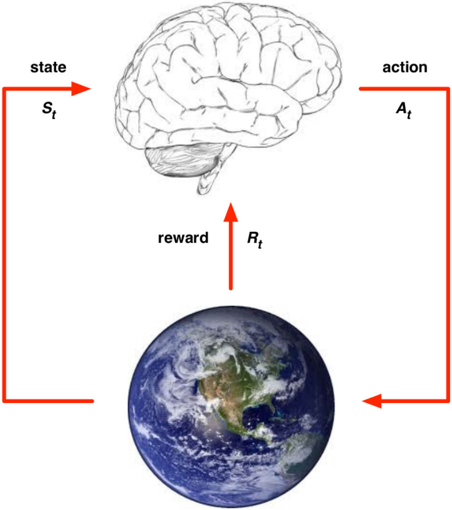
Sequential decision making

- Actions may have long term consequences.

To be *greedy* can be wrong

- A financial investment (may take months to mature).
- Refuelling a helicopter (might prevent a crash in several hours).
- Blocking opponent moves (might help winning chances many moves from now).

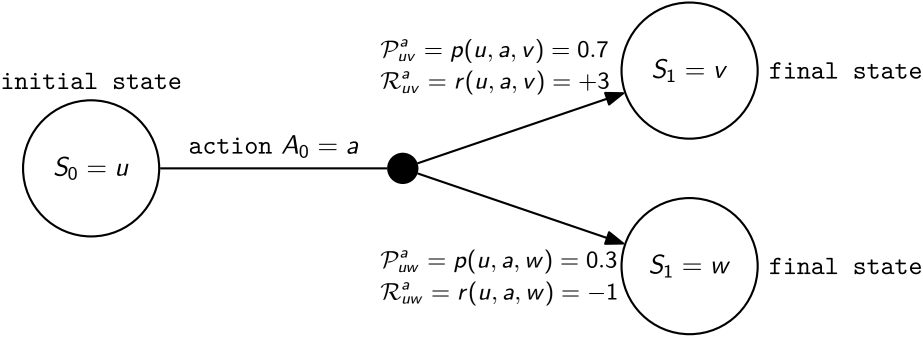
The big picture: environment and agent



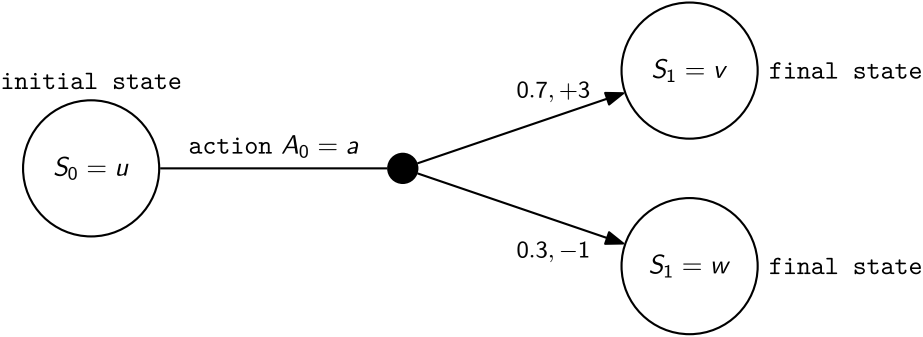
A never-ending loop

- ...we (the agent) receive R_t and observe S_t...
- ...and thus we decide to do action A_t...
- ...and because of our action A_t, the environment send us a reward R_{t+1} and a new state, that we observe as S_{t+1}...

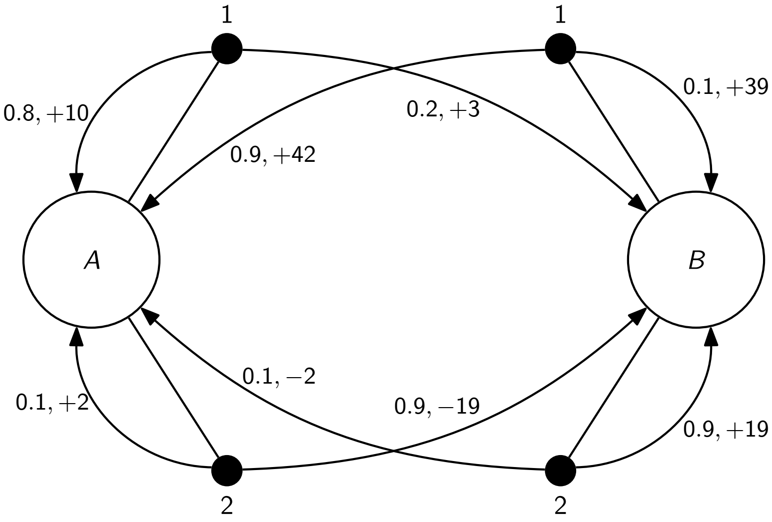
The building block: state, action, probability, reward



The building block: state, action, probability, reward



The MDP originating our game



What can we do?

We control only the actions! We are not in control of the environment probabilities and rewards (the model)!

Markov Decision Process: MDP

Markov decision process data

- A set of *states* $\mathcal{S}$ and a set of *actions* $\mathcal{A}$.
- For each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, a probability distribution $p(\cdot|s, a)$ over $\mathcal{S} \times \mathbb{R}$.
- A discount factor γ.

Distribution model

The probability p is called the *distribution model* of the MDP.

From now on, assume that $\mathcal{S}$ and $\mathcal{A}$ are finite, and $\gamma = 1$.

Markov Decision Process: MDP

Distribution model

- The probability distribution p of the MDP gives the next state and reward:

$$p(s', r|s, a) = \Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a).$$

- Given a state s, an action $a \in \mathcal{A}$ will take to a state s' with probability:

$$\mathcal{P}^a_{ss'} = p(s'|s, a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a).$$

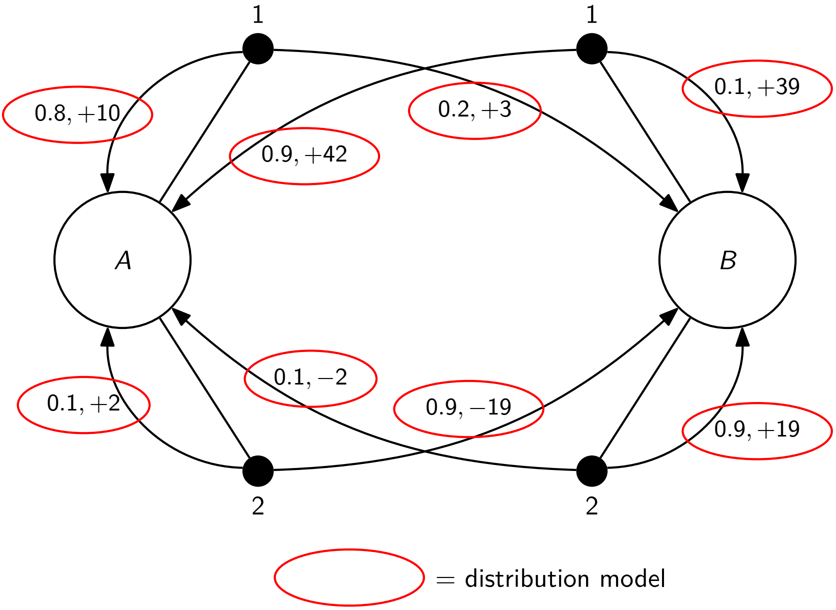
  Thus, we have a transition matrix $\mathcal{P}^a$ for each action a.

- Given a state s, an action $a \in \mathcal{A}$ will give an average reward:

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a].$$

  Thus, we have an average reward vector $\mathcal{R}^a$ for each action a.

Example



1
1
0.8, +10
0.2, +3
0.1, +39
0.9, +42
A
B
0.1, −2
0.1, +2
0.9, −19
0.9, +19
2
2
= distribution model

Decisions: $\mathcal{S} \to \mathcal{A}$

Where are the decisions?

- In any state s, *the agent must choose* between available actions a.
- When choosing a from s, the environment answers s' with probability $\mathcal{P}^a_{ss'}$. Environment decision.
- The agent behaviour is given by probabilities $\pi(a|s)$: "how likely I'm going to choose a from s?". Agent decision.

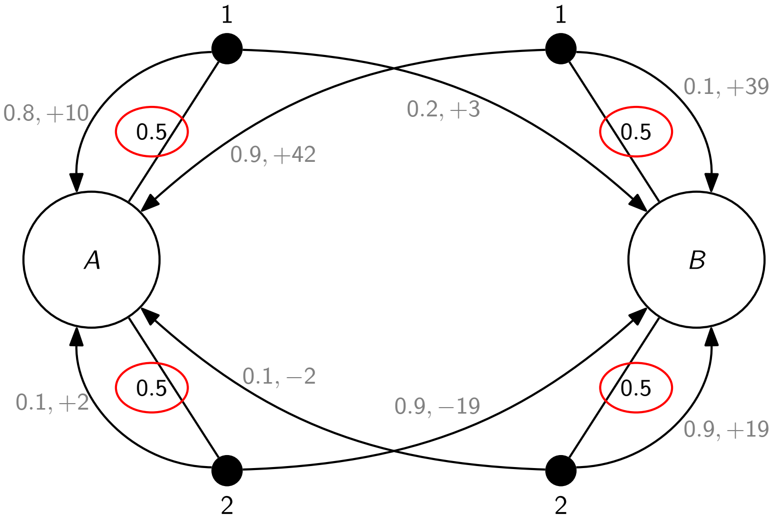
Definition

A *policy* π is a probability distribution over actions given states:

$$\pi(a|s) = \Pr(A_t = a | S_t = s)$$

inserire immagini di policy deterministiche (tabelle?)
fare esempio morra cinese per policy stocastica

A uniform stochastic policy



What can we do?

At every step, we choose the action according to the probability.

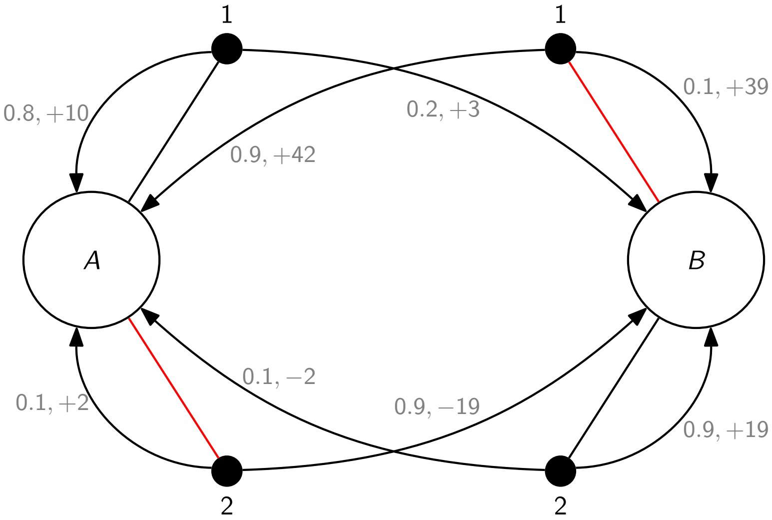
A uniform stochastic policy, tabular representation

A	[0.5,0.5]
B	[0.5,0.5]

Tabular representation

Every line in the table corresponds to a state.

A deterministic policy



Question

What can you say about this policy?

A deterministic policy, tabular representation

A	1
B	2

Tabular representation

Every line in the table corresponds to a state.

How much are states and actions worth?

Definition

The *total return* G_t at time t is

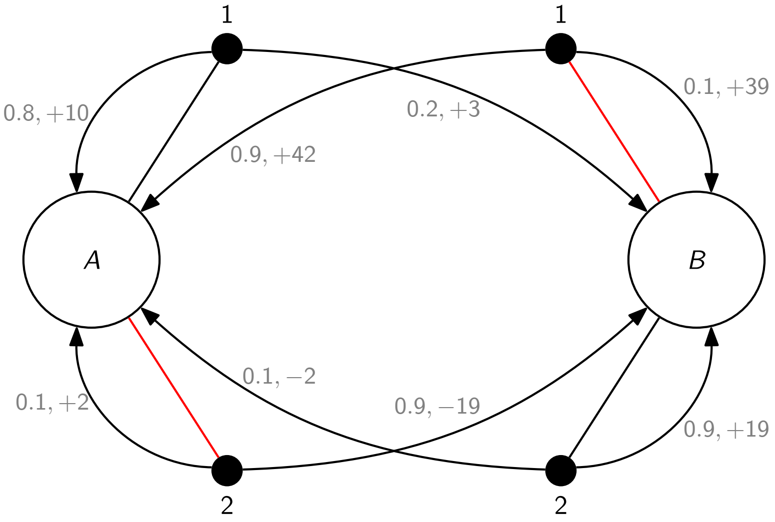
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{+\infty} \gamma^k R_{t+1+k}$$

Definition: state-value function

The *state-value function* $v_\pi(s)$ for a MDP is the return we can expect to accumulate starting from state s, *following the policy* π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

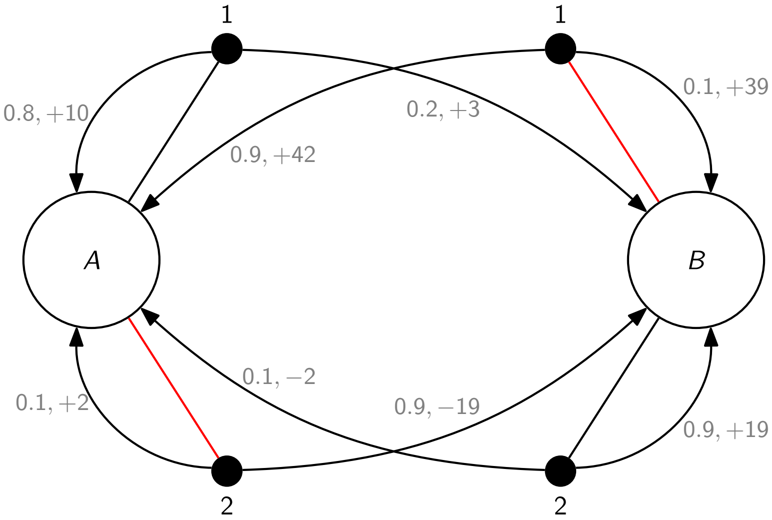
A deterministic policy



Example: value for the optimal policy π_*

$$0.1 \cdot 2v_*(A) + 0.9 \cdot (-19)v_*(B)$$

A deterministic policy



Iterative, infinite computation for v_* – can you spot a problem?

$$0.1 \cdot 2[0.1 \cdot 2v_*(A) + 0.9 \cdot (-19)v_*(B)] + 0.9 \cdot (-19)(0.9 \cdot 42v_*(A) + 0.1 \cdot 39v_*(B))$$

How much are states and actions worth?

Definition

The *total return* G_t at time t is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{+\infty} \gamma^k R_{t+1+k}$$

Definition: action-value function

The *action-value function* $q_\pi(s, a)$ for a MDP is the return we can expect to accumulate starting from a state s, choosing action a, and then *following the policy* π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

What is the best value for states and actions?

Definition

The *optimal state-value function* v_* is the maximum state-value over all policies:

$$v_*(s) = \max_\pi v_\pi(s)$$

Definition

The *optimal action-value function* q_* is the maximum action-value over all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Definition

Any policy obtaining optimal state-value or optimal action-value is a *optimal policy*: π_* is a optimal policy if

$$q_{\pi_*} = q_* \quad \text{or} \quad v_{\pi_*} = v_*$$

RL in short

Optimal policy

Our aim is to find a policy π that, for each state s, obtains the best $v_\pi(s)$. That is, our aim is to find the optimal policy π_*.

The *prediction* problem in RL

Forecast the future: can you say from each state how much will be your return? Policy *evaluation* step: $\pi \xrightarrow{E} v_\pi$ or $\pi \xrightarrow{E} q_\pi$.

The *control* problem in RL

Change the future: can you find a different policy that will give you a better return? Policy *improvement* step: $v_\pi \xrightarrow{I} \pi'$.

RL in short

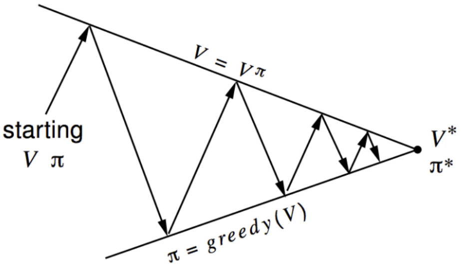
Finding the optimal policy: *policy iteration* step

Iteration of policy evaluation and policy improvement gives a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_{\pi_*}$$

finite MDP $\Rightarrow$ finite number of policies $\Rightarrow$ converge in finite steps

RL in short: policy iteration

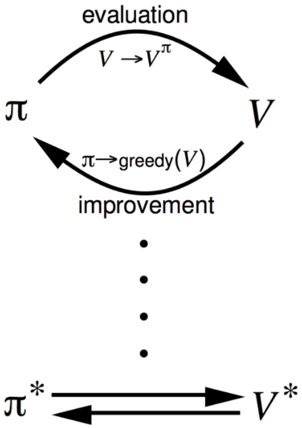


Policy evaluation Estimate v_π

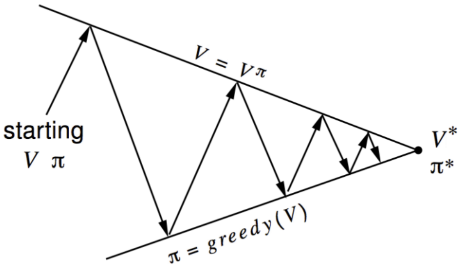
Iterative policy evaluation

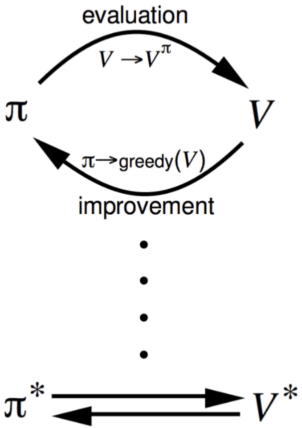
Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement



RL in short: generalized policy iteration





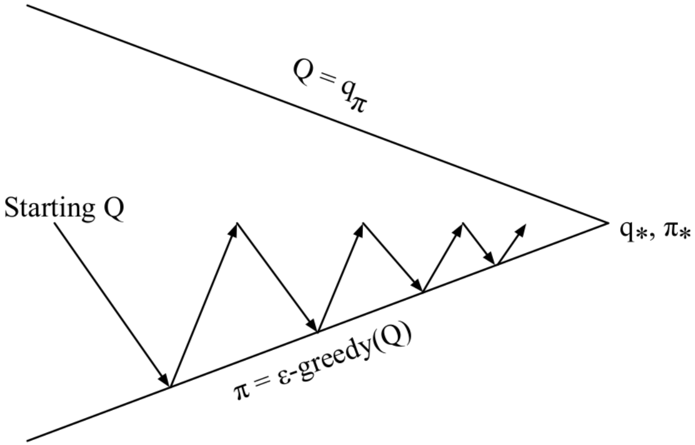
Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

RL in short: GPI with partial evaluation of q



- Partial policy evaluation: $Q \sim q_{\pi}$.
- Any policy improvement algorithm.

Bellman equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman equation for v_π

The state-value function satisfy the following recursive formula:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Theorem: Bellman equation for q_π

The action-value function satisfy the following recursive formula:

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s')(q_\pi(s', a'))$$

Bellman equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman equation for v_π

The state-value function satisfy a linear recursive formula:

$$v_\pi(s) = f(v_\pi(s'))$$

Theorem: Bellman equation for q_π

The action-value function satisfy a linear recursive formula:

$$q_\pi(s, a) = f(q_\pi(s', a'))$$

Bellman equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman equation for v_π

The state-value function satisfy a linear fixed-point formula:

$$v_\pi = f(v_\pi)$$

Theorem: Bellman equation for q_π

The action-value function satisfy a linear fixed-point formula:

$$q_\pi = f(q_\pi)$$

Bellman optimality equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman optimality equation for v_*

The optimal state-value function satisfy the following recursive formula:

$$v_*(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

Theorem: Bellman optimality equation for q_*

The optimal action-value function satisfy the following recursive formula:

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'}(q_*(s',a'))$$

Bellman optimality equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman optimality equation for v_*

The optimal state-value function satisfy a non linear recursive formula:

$$v_*(s) = f(v_*(s'))$$

Theorem: Bellman optimality equation for q_*

The optimal action-value function satisfy a non linear recursive formula:

$$q_*(s, a) = f(q_*(s', a'))$$

Bellman optimality equations

Recursive formula for return

The total return satisfies $G_t = R_{t+1} + \gamma G_{t+1}$.

Theorem: Bellman optimality equation for v_*

The optimal state-value function satisfy a non linear fixed-point formula:

$$v_* = f(v_*)$$

Theorem: Bellman optimality equation for q_*

The optimal action-value function satisfy a non linear fixed-point formula:

$$q_* = f(q_*)$$

Prediction: policy evaluation

Problem: evaluate a given policy π

Solution: iterative application of Bellman (expectation) equation.

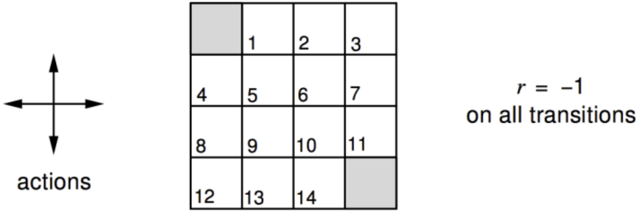
How to do it

- Start from any v_0.
- Given v_k, use Bellman equation as a definition for v_{k+1}.
- Stop when you like it.

Iterative policy evaluation for estimating $V \sim v_\pi$

Input: Policy π to be evaluated.
Parameter: *Threshold* $\theta > 0$ *determining accuracy of estimation.*
Output: Estimate V of v_π.
Initialize $V(s)$, for all $s \in \mathcal{S}$, arbitrarily except $V(\text{final}) = 0$.
do
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s')\right)$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
while $\Delta > \theta$

Policy evaluation example: gridworld



- Undiscounted episodic MDP: 14 nonterminal states $1, \ldots, 14$, one terminal state (shown twice as □), four actions $\rightarrow, \leftarrow, \uparrow, \downarrow$.
- Actions leading out of the grid leave state unchanged.
- Reward is -1 until the terminal state is reached.
- Agent follows uniform random policy: $\pi(\cdot|\cdot) = 0.25$.

Exercise

Compute the first step of iterative evaluation of v_π.

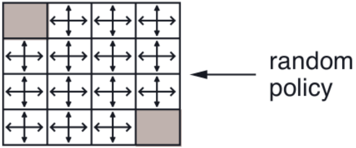
Policy evaluation example: gridworld

v_k for the Random Policy

Greedy Policy w.r.t. v_k

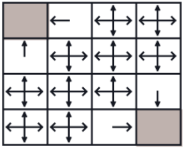
$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



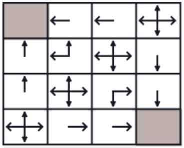
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy evaluation example: gridworld

$k = 3$

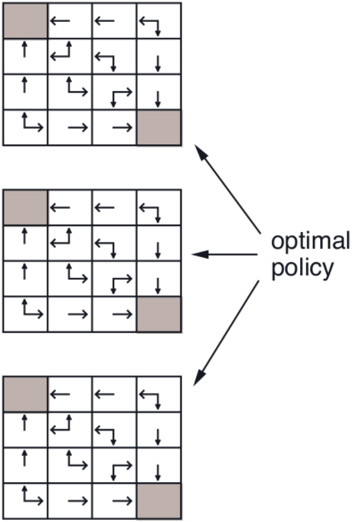
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Control: policy improvement

We have (we know how to compute) the value v_π. Then?

Improve the policy by acting greedily with respect to v_π:

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

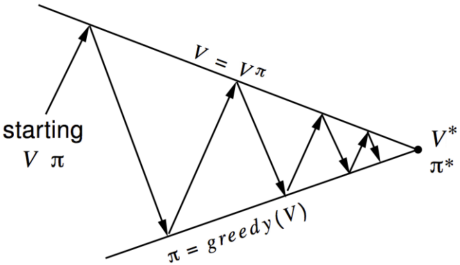
Rationale

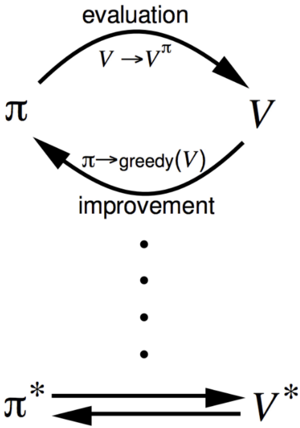
No need to follow the policy if we know that a certain action is better than the others.

Definition

We say that π' is the *greedy policy with respect to* π.

Putting things together: policy iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Modified policy iteration

Exercise

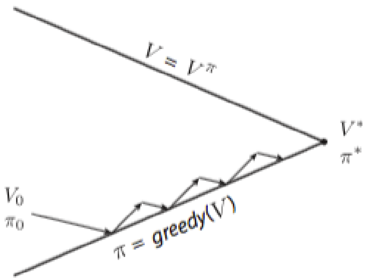
Can policy iteration be improved? Hint: look what happens in the gridworld example.

- Policy evaluation

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_\pi$$

  can be stopped before v_π is reached.
- Stopping condition (for instance, when the max error is below a threeshold), or stop after k iterations.
- In gridworld $k = 3$ gives optimal policy.
- Extreme case: stop evaluation after one iteration (called *value iteration*).

Value iteration: partial evaluation of v_π



- Partial policy evaluation: $V \sim v_\pi$.
- Any policy improvement algorithm.

Control via Bellman optimality equations

Question

Assuming you know the optimal state-value function v_* or the optimal action-value function q_*, how do you find an optimal policy?

Answer for v_*

In a state s, choose the best a:

$$\pi_*(s) = \underset{a}{\text{argmax}}(\mathcal{R}_s^a + \sum_{s'} \mathcal{P}_{ss'}^a v_*(s'))$$

Question

Do you see a problem in using v_* to find π_*?

Control via Bellman optimality equations

Question

Assuming you know the optimal state-value function v_* or the optimal action-value function q_*, how do you find an optimal policy?

Answer for v_*

In a state s, choose the best a:

$$\pi_*(s) = \underset{a}{\operatorname{argmax}}(\mathcal{R}_s^a + \sum_{s'} \mathcal{P}_{ss'}^a v_*(s'))$$

Question

Do you see a problem in using v_* to find π_*? We need a distribution model! And what happens if we have q_* instead?

Control via Bellman optimality equations

Question

Assuming you know the optimal state-value function v_* or the optimal action-value function q_*, how do you find an optimal policy?

Answer for v_*

In a state s, choose the best a:

$$\pi_*(s) = \operatorname*{argmax}_a (\mathcal{R}_s^a + \sum_{s'} \mathcal{P}_{ss'}^a v_*(s'))$$

Question

Do you see a problem in using v_* to find π_*? We need a distribution model! And what happens if we have q_* instead?

Answer for q_*: model-free solution

In a state s, choose the best a: $\pi_*(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_*(s, a)$. If we know q_*, we are done!

Prediction with Monte Carlo

State-value function

Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Without the model, how do you compute the expected return?

Law of large numbers

Monte Carlo: empirical mean instead of expected return. To learn $v_\pi(s)$, run *episodes of experience* from s under policy π:

$$S_1 = s, A_1, R_2, S_2, A_2, R_3, \ldots, R_T, S_T \sim \pi$$

and then compute the empirical mean of all the total returns $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$ obtained.

Prediction with Monte Carlo

Monte Carlo: learning from samples

- MC learns from samples: knowledge of transitions $\mathcal{P}^a_{ss'}$ and rewards not needed.
- MC learns from *complete* episodes: no bootstrapping.
- MC estimates of s are independent on estimates of other states s'.
- MC uses the law of large numbers: state-value=expected value=empirical mean.

MC Prediction: towards a target through error

Incremental mean formula

The empirical mean V_k of a sequence $G_1, G_2, \ldots, G_k$ can be computed incrementally:

$$V_k = V_{k-1} + \frac{1}{k}(G_k - V_{k-1})$$

Rewording the incremental mean formula

V_k is obtained going from V_{k-1} towards a *target* G_k. The quantity "target − previous value" is called *error*:

$$V_k = V_{k-1} + \alpha_k \cdot \text{error} = V_{k-1} + \alpha_k \cdot \Delta$$

MC prediction, different versions

Incremental updates MC algorithm

The incremental formula can be used to update $V(s)$ incrementally after episode $S_1, A_1, R_2, \ldots, S_T$. For each state S_t with return G_t:

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

Constant-α MC algorithm

If the problem is non-stationary, we can use a *running mean*, giving less and less importance to old episodes:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

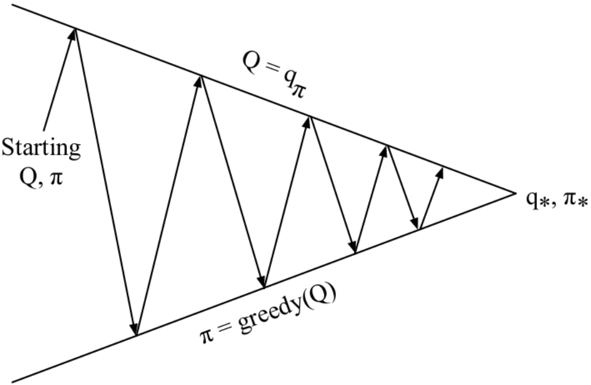
First-visit MC, incremental updates, for estimating $V \sim v_\pi$

Input: Policy π to be evaluated.
Initialize: $V(s) \in \mathbb{R}$ *arbitrarily;* $N(s) \leftarrow 0$, *for all* $s \in \mathcal{S}$.
while *True* **do**
    Generate an episode following π:
     $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t \in \{S_0, S_1, \ldots, S_{t-1}\}$ **then**
            next t
        else
            $N(S_t) \leftarrow N(S_t) + 1$
            $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G - V(S_t))$
        end
    end
end

First-visit *constant* α MC prediction, for estimating $V \sim v_\pi$

Input: Policy π to be evaluated.
Parameter: *Learning rate* $\alpha > 0$.
Initialize: $V(s) \in \mathbb{R}$ *arbitrarily.*
while *True* **do**
    Generate an episode following π:
    $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t \in \{S_0, S_1, \ldots, S_{t-1}\}$ **then**
            next t
        else
            $V(S_t) \leftarrow V(S_t) + \alpha(G - V(S_t))$
        end
    end
end

MC control: General Policy Iteration with Q-value



- MC policy evaluation: $Q = q_\pi$.
- Policy improvement: greedy policy improvement, does it work?

Greedy is not always good

Which bandit?

You played 2 times each. Reward(left)=0, reward(center)=7, reward(right)=10. Which one next? Is the greedy policy correct?

ϵ-greedy policy improvement

Exploration-exploitation dilemma

Since we are using the law of large numbers, we need to be sure that every state is visited infinite times: we need to *explore* states that have not been visited enough. But we would also like to *exploit* states with high values!

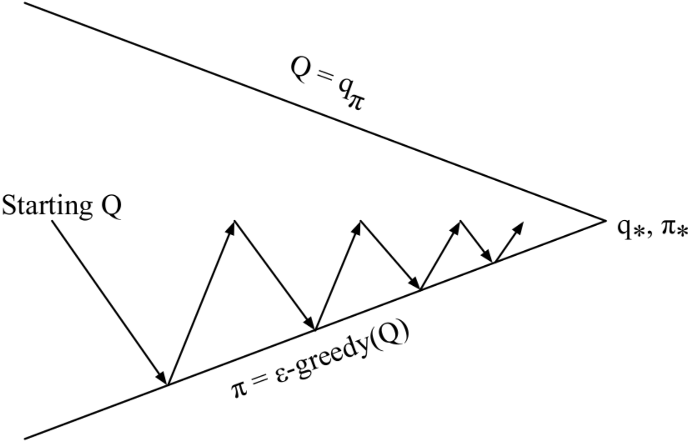
Solution: try all actions eventually

Choose the greedy action quite often:

$$\pi'(a_*|s) = \begin{cases} 1-\epsilon & \text{if } a_* = \text{argmax}_a\, Q_\pi(s,a) \\ \text{what is left} & \text{otherwise} \end{cases}$$

This is called *ϵ-greedy improvement of π*.

GPI with Q-value, ϵ-greedy improvement, episode based



- MC policy evaluation *episode based*: $Q \sim q_\pi$.
- Policy improvement: ϵ-greedy policy improvement.

Enjoyable videos

Playing Atari Breakout

https://www.youtube.com/watch?v=_LEthduIbtk

Learning to walk

https://www.youtube.com/watch?v=gn4nRCC9TwQ

Large-scale problems

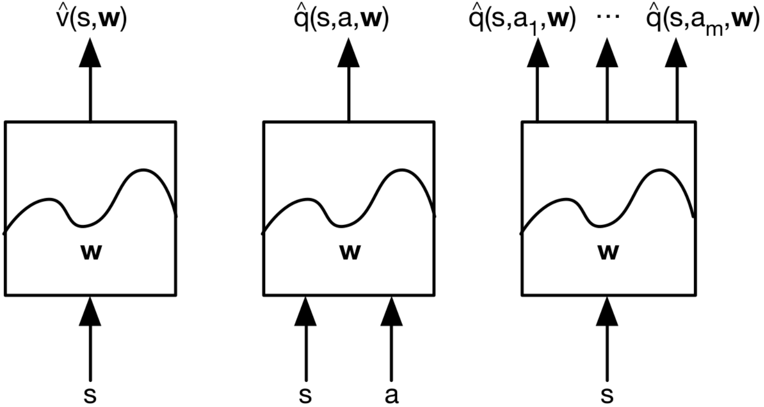
Real life problems can be very large

- Backgammon: 10^{20} states.
- Computer Go: 10^{170} states.
- Starcraft: more than 10^{1685}.
- Helicopter: continuous state space.
- Protein folding problem.

Tabular methods doesn't work

- With methods seen up to now, we need a *lookup table* storing $V(s)$ (dimension $|\mathcal{S}|$) or $Q(s, a)$ (dimension $|\mathcal{S}||\mathcal{A}|$) elements.
- There are too many states and/or actions to store in memory.
- Assuming you can store a large table, it is too slow to learn the value of each state individually.
- Need to scale up model-free RL technique.

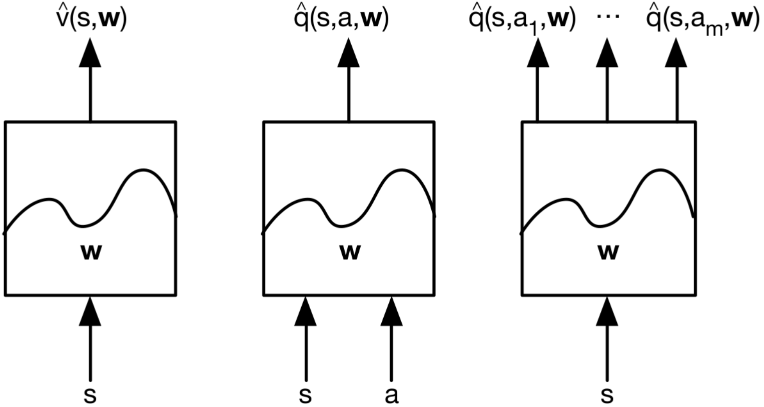
Large scale problems



Solution for large MDP

- Use an *approximation* $\hat{q}(s, a, \mathbf{w}) \sim q_\pi(s, a)$, where $\mathbf{w} \in \mathbb{R}^d$.
- Update **w** instead of the table: the dimension of the problem becomes $d << |\mathcal{S}|$. Use RL to update **w**.
- Try to make the approximation *generalize* to unseen states.

Large scale problems



Standard approximators

- Linear combination of features (Deep Blue, 8000 binary features).
- Neural network (AlphaGo family).

These are *differentiable* approximators: needed for gradient descent training.

RL as supervised learning

state $\rightarrow$ update

All prediction methods: estimated value q of pair s, a shifts toward an update target u:

$$q_{k+1}(s, a) = q_k(s, a) + \alpha(u - q_k(s, a)).$$

Idea

Use $s, a \mapsto u$ as training data for supervised learning! For instance, the MC update rule is $S_t, A_t \mapsto G_t$.

RL as supervised learning

Loss function on single example (S_t, A_t)

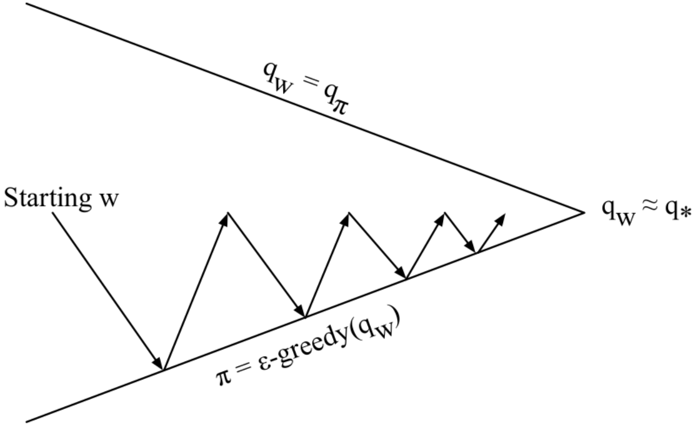
$$\frac{1}{2}(q(S_t, A_t) - \hat{q}(S_t, A_t, \mathbf{w}))^2$$

MC update rule for parameters $\mathbf{w}$ of $\hat{q}(s, a, \mathbf{w})$

We are estimating $q(S_t, A_t)$ with G_t, thus the gradient descent gives:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla(\hat{q}(S_t, A_t, \mathbf{w}))$$

Policy iteration with approximation



- *Approximate* policy evaluation time-step based: $\hat{q}(\cdot, \cdot, \mathbf{w}) \sim q_\pi$.
- Policy improvement: ϵ-greedy policy improvement. Poor convergence results!

Convergence issues: the deadly triad

Deadly triad

Instability and divergence can, and usually will, arise whenever we combine *all of the following* three elements:

Function approximation A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., neural networks).

Bootstrapping Update targets that include existing estimates (as in *Temporal Difference* methods), rather than relying exclusively on actual rewards and complete returns (as in MC methods).

Off-policy training Training on a distribution of transitions other than that produced by the target policy.

If any two elements of the deadly triad are present, but not all three, then instability can be avoided.

Example: Blackjack

- States: current sum (12-21), dealer's showing card (ace-10), usable ace (yes/no).
- Actions: stick (stop receiving cards and terminate), twist (take another card).
- Reward for stick: $+1, 0, -1$ if sum of cards $>, =, <$ sum of dealer cards.
- Reward for twist: -1 if sum of cards > 21, and terminate, 0 otherwise.
- Transitions (dealer's rule): automatically twist if sum of cards < 12.

Exercise

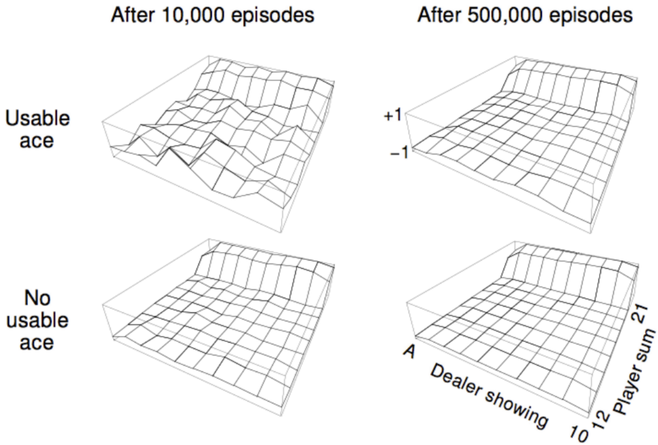
Consider the policy that sticks if sum of cards ≥ 20, twist otherwise. Compute its value function.

First-visit MC control episode based, for estimating $\pi \sim \pi_*$

Parameter: *Real number* $\epsilon > 0$.
Initialize: π = *any* ϵ*-greedy policy.*
$Q(s, a) \in \mathbb{R}$ *arbitrarily.*
returns(s, a)=*empty list.*
while *True* **do**
    Generate an episode following π:
      $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \dots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if $S_t \in \{S_0, S_1, \dots, S_{t-1}\}$ **then**
            next t
        else
            returns(S_t, A_t).*append*(G)
            $Q(S_t, A_t) \leftarrow$ *average*(*returns*(S_t, A_t))
            $\pi \leftarrow$ *greedy*(π, S_t) (make the policy greedy for S_t)
        end
    end
end

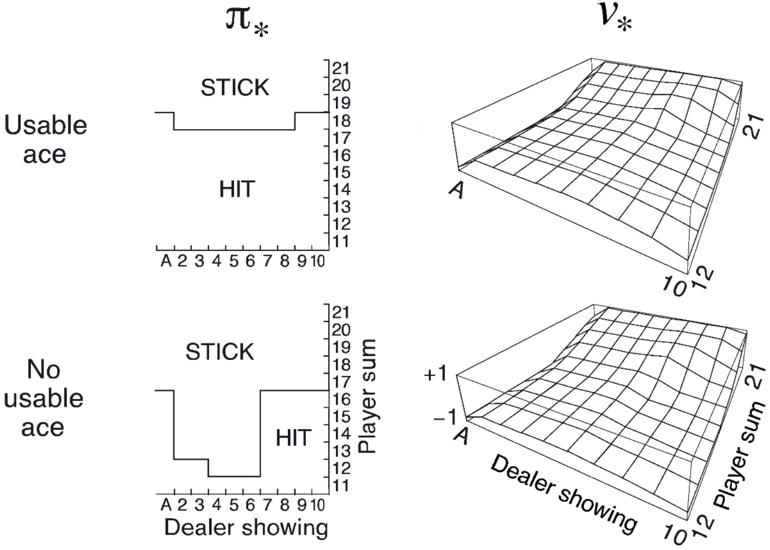
Blackjack value function after MC prediction

Policy: stick if sum of cards ≥ 20, otherwise twist.



After 500000 episodes, MC prediction gives the correct value function for this policy.

Blackjack optimal policy after MC learning



After 500000 episodes, MC learning computes Thorp's Blackjack strategy.


That's all Folks!

Licenza