

Lecture 5: Solving MDPs and Reinforcement Learning

Viliam Lisý & Branislav Božanský

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

viliam.lisy@fel.cvut.cz

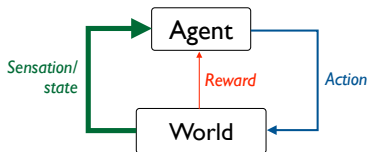
March, 2023

- 1 Value functions and Bellman equations
- 2 Basic iterative solution techniques for **known MDP**

Next lecture

- 1 RL algorithms in tabular representation for **unknown MDP**
- 2 Scaling up with Neural Networks
- 3 DQN algorithm and its application to Atari games

Reinforcement learning is *more autonomous learning*

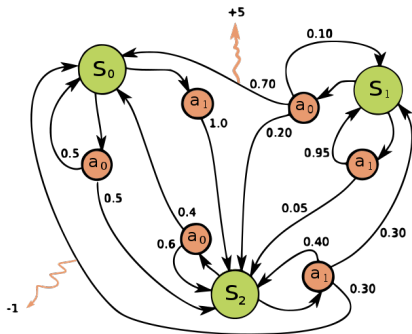


- Learning that requires less input from people
- AI that can learn for itself, during its normal operation

Taken from R. Sutton's slides (and many following are adaptations as well).

Standard model for Reinforcement Learning problems

- S – states
- R – rewards
- A – actions
- Discrete steps $t = 0, 1, 2, \dots$
- Environment *dynamics*



Source: Waldoalvarez @ wikimedia

$$p(s', r | s, a) \leftarrow Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

Policy at step t , denoted π_t , maps from states to actions.

$$\pi_t(a|s) = \text{probability that } A_t = a \text{ when } S_t = s$$

Special case are **deterministic** policies.

$$\pi_t(s) = \text{the action taken with } \textit{prob} = 1 \text{ when } S_t = s$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience
- Roughly, the agents goal is to get as much reward as it can **over the long run.**

One of the most fundamental concepts of RL!

A **value function** for an MDP and a policy π

$$v_{\pi} : \mathcal{S} \rightarrow \mathbb{R}$$

is a function assigning each state s the expected return $v_{\pi}(s) = \mathbb{E}_{\pi} G_0$ obtained by following policy π from state s .

- For finite MDPs, policies can be **partially ordered**:

$$\pi \leq \pi' \text{ if and only if } v_{\pi}(s) \leq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π_* .
- Optimal policies share the same **optimal state-value function**:

$$v_*(s) = \max_{\pi} v_{\pi}(s) \text{ for all } s \in \mathcal{S}$$

- Optimal policies also share the same **optimal action-value function**:

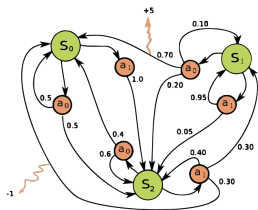
$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a) \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}.$$

This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why Are Optimal (Action-) Value Functions Useful

Any policy that is greedy with respect to v_* is an optimal policy.

$$\pi_*(s) = \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')]$$



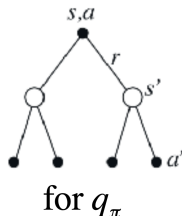
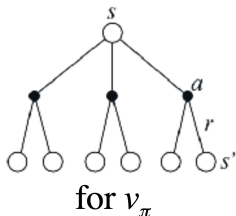
Given q_* , the agent does not even have to do a one-step-ahead search:

$$\pi_*(s) = \arg \max_a q_*(s, a)$$

Bellman Equation for a Policy

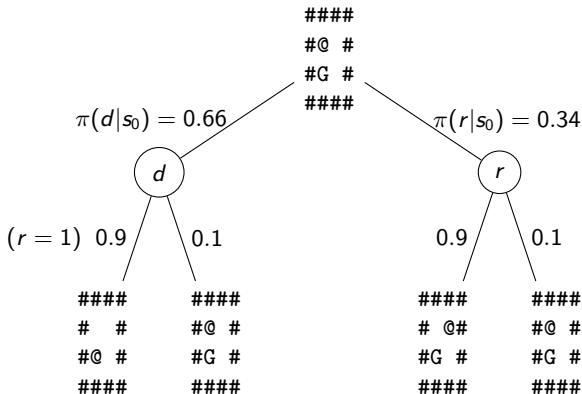
$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_{\pi}(s')]$$

This is a set of equations (in fact, linear), one for each state. The value function for π is its unique solution.



Allows simple computation of values for a policy.
Can we also compute policy for values?

A robot on a slippery floor successfully moves with probability 0.9.

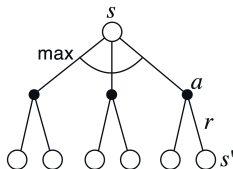


Bellman Optimality Equation for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned}v_*(s) &= \max_a q_{\pi_*}(s, a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')].\end{aligned}$$

The relevant backup diagram:

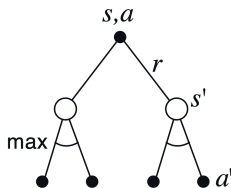


v_* is the **unique** solution of this system of nonlinear equations.

Bellman Optimality Equation for q_*

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right]$$
$$= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right].$$

The relevant backup diagram:



q_* is the unique solution of this system of nonlinear equations.

Is there any relation between v, q used in RL and f, g, h we defined for deterministic MDPs for algorithm A?

$$v_*(s) \approx -h^*(s)$$

Solving the Bellman Optimality Equation

Finding an optimal policy by solving the Bellman Optimality Equation **requires** the following:

- accurate knowledge of environment dynamics;
- we have enough space and time to do the computation;
- the Markov Property.

How much space and time do we need?

- polynomial in number of states,
- BUT, number of states is often huge (e.g., backgammon has about 10^{20} states).

We usually have to settle for approximations.

Many RL methods can be understood as approximately solving the Bellman Optimality Equation.

Policy Evaluation (Prediction)

Policy Evaluation: for a given policy π , compute the state-value function v_π

Recall: **State-value function for policy π**

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$


Recall: **Bellman equation for v_π**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma v_\pi(s')]$$

—a system of $|S|$ simultaneous equations

Iterative Policy Evaluation (Prediction)

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

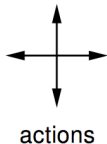
a “sweep” 

A sweep consists of applying a **backup operation** to each state.

A full policy-evaluation backup:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

A Small Gridworld Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$$\gamma = 1$$

- ❑ An undiscounted episodic task
- ❑ Nonterminal states: 1, 2, . . . , 14;
- ❑ One terminal state (shown twice as shaded squares)
- ❑ Actions that would take agent off the grid leave state unchanged
- ❑ Reward is -1 until the terminal state is reached

Iterative Policy Eval for the Small Gridworld

V_k for the
Random Policy

$\pi =$ equiprobable random action choices



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, ..., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$k = 0$

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

$k = 3$

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

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Iterative Policy Evaluation – One array version

Input π , the policy to be evaluated

Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

 For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Why Does Iterative Policy Evaluation Work?

Many other RL algorithms use the same proof technique.

Definition (γ -contraction)

Any function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a γ -contraction for $0 < \gamma < 1$ if and only if for some norm $\| \cdot \|$ and all $x, y \in \mathbb{R}^n$

$$\|F(x) - F(y)\| \leq \gamma \|x - y\|$$

Theorem (Contraction mapping)

For a γ -contraction F

- Iterative application of F converges to a **unique** fixed point independently of the starting point
- at a linear convergence rate determined by γ .

(Based on Tom Mitchell's [slides](#))

Suppose we have computed a v_π for policy π . Can we easily improve it?

If there is a state s and action a such that $q_\pi(s, a) > v_\pi(s)$ than setting $\pi(s) = a$ improves the strategy.

Can it break the strategy somewhere else?

No, because the value at s improves.

The values in other states that eventually lead to s improve.

There is **no** state for which the value can decrease.

Can anything break if we modify more states at once?

No, for a similar reason, the value in each state only increases.

Policy Iteration – One array version (+ policy)

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

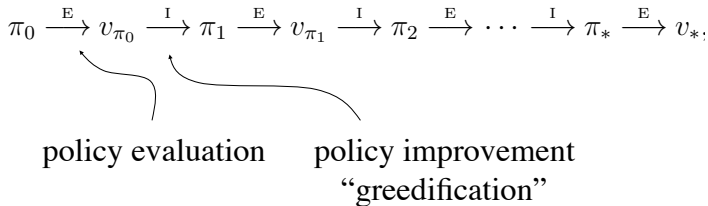
$$a \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$$

If $a \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return V and π ; else go to 2

Policy Iteration



Iterative Policy Eval for the Small Gridworld

$\pi =$ equiprobable random action choices



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

- An undiscounted episodic task
- Nonterminal states: 1, 2, ..., 14;
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

$$\pi'(s) \doteq \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')]$$

for all $s \in \mathcal{S}$

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

	↔	↔	↔
↔	↔	↔	↔
↔	↔	↔	↔
↔	↔	↔	

← random policy

$k = 1$

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

	←	←	↔
↑	↔	↔	↓
↑	↔	↔	↓
↔	→	→	

$k = 3$

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

	←	←	↔
↑	↔	↔	↓
↑	↔	↔	↓
↔	→	→	

← optimal policy

$k = \infty$

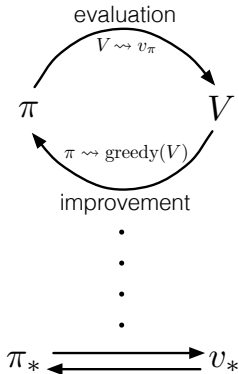
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

	←	←	↔
↑	↔	↔	↓
↑	↔	↔	↓
↔	→	→	

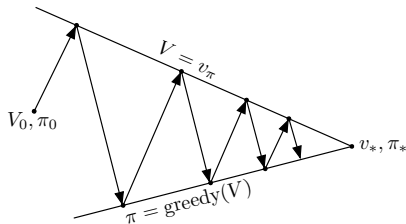
Generalized Policy Iteration

Generalized Policy Iteration (GPI):

any interaction of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



It is sufficient to combine **any** consistent improvement in value estimate with **any** consistent improvement of the policy based on the value.

- Subset of states (even one)
- Improvement only in expectation
- Policy improvement only based on one action
- Small value improvement in the right direction

Value Iteration

Recall the **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Here is the **full value-iteration backup**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_k(s') \right] \quad \forall s \in \mathcal{S}$$

Value Iteration – One array version

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

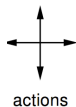
$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π , such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Example



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$R = -1$
on all transitions

$\gamma = 1$

$$V_0 =$$

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

$$V_1 =$$

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

$$V_2 =$$

0.0	-1.0	-2.0	-2.0
-1.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.0
-2.0	-2.0	-1.0	0.0

$$V_3 =$$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

State and action value functions are key concepts in RL

Their values in different states are tied by Bellman equations

Bellman equations used as operators are contractions and hence their iterative application converges to unique solutions

(Generalized) policy iteration and value iterations are simple algorithms to solve MDPs

However, these algorithms require full knowledge of MDP, which is not necessary in RL methods, which generally approximate the full Bellman operator