Deep Reinforcement Learning

4 - Dynamic Programming

Prof. Dr. Malte Schilling

Autonomous Intelligent Systems Group

Recap – Markov Decision Process

MDP Definition (Sutton und Barto 2018)

A Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, consisting of

- a set of states $\mathcal{S}$
- a set of actions $\mathcal{A}$
- a joint probability $p(r, s'|s, a)$ describing the dynamics of the environment as
  - transition probabilities for switching states

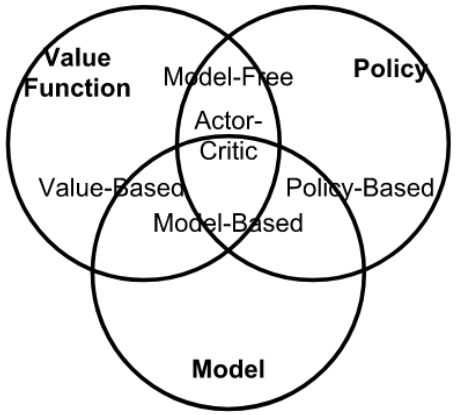
$$p(s'|s, a) = \sum_r p(r, s'|s, a)$$

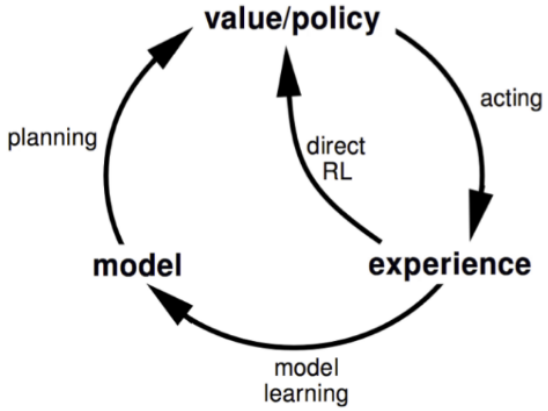
  - and expected reward $\mathbb{E}(R|s, a) = \sum_r r \sum_{s'} p(r, s'|s, a)$
- the discount factor $\gamma \in [0, 1]$

Recap – Approaches to Reinforcement Learning

The reaction of the environment to certain actions can be represented by a **model** which the agent may or may not know.

The model defines the reward function and transition probabilities.





- Model-based: Rely on the model of the environment; either the model is known or the algorithm learns it explicitly. Use planning on learned or given model.
- Model-free: No dependency on the model during learning. Learning with imperfect information.

(Weng 2018)

Distinction of problems in Reinforcement Learning

Prediction

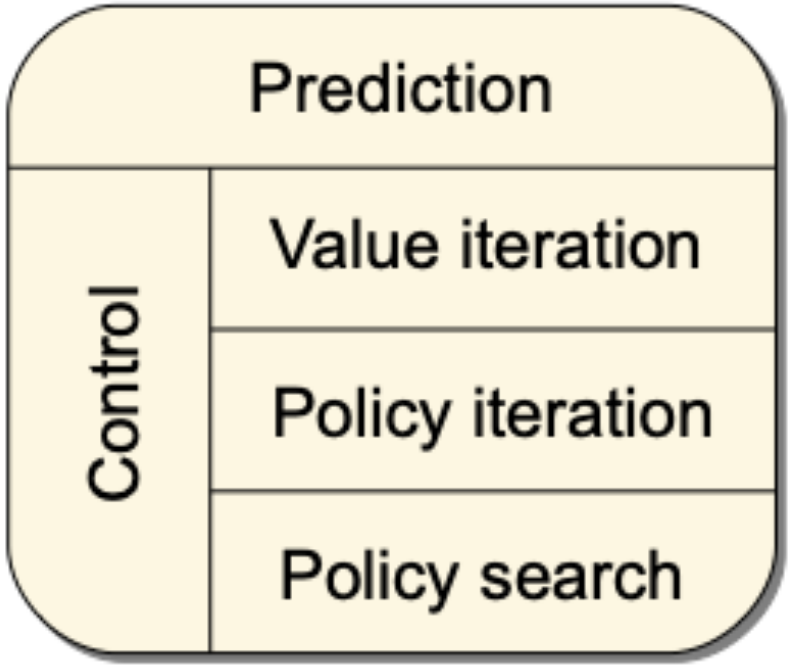
Estimating value functions for a given policy: this is called policy evaluation.

- Given a policy, what is my expected return?
- Using a strategy, what is my expected return?

Control

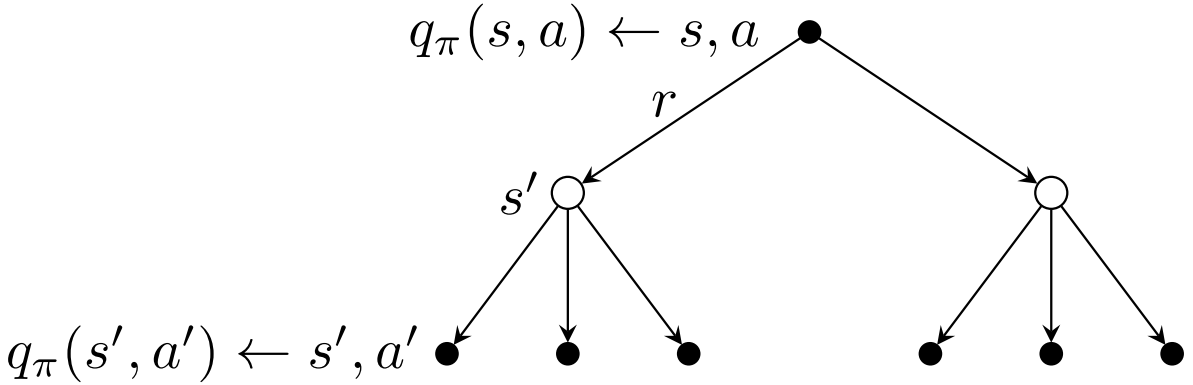
On the other hand, exploiting a value function (ideally an optimal one) is used for optimisation of a policy.

- What is the optimal way of behaving?
- What is the optimal control policy to maximise a reward or minimise time, fuel consumption, etc.?



(Szepesvári 2010)

Recap – Bellman Expectation for Q_π



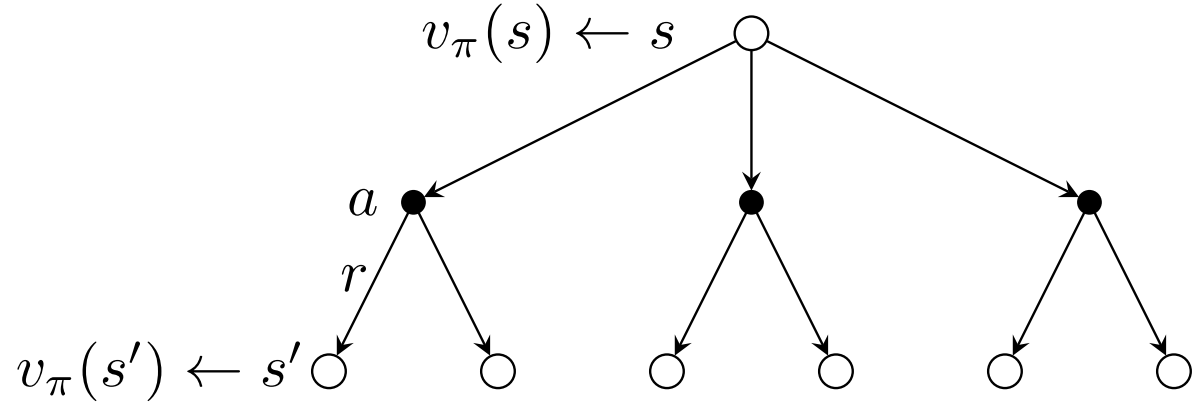
Action-Value Function using Bellman Expectation

$$q_\pi(s,a) = \sum_r \sum_{s' \in \mathcal{S}} p(r,s'|s,a)\left(r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')\right)$$

With reward independent of s':

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Recap – Bellman Expectation for V_π



Recursive Formulation for State-Value Function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_r \sum_{s' \in \mathcal{S}} p(r, s'|s, a) \Big(r + \gamma v_\pi(s') \Big)$$

With reward independent of s':

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \Big(R^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v_\pi(s') \Big)$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s') \right)$$

Overview Lecture

Until now:

- Assumed a policy as given
- Computing of a Value Function

Now find

- Optimal Policies and Value Function
- through Dynamic Programming when MDP is fully known (transition and reward policies)

Optimal Policies

Optimal Policy

There is a partial ordering over policies which means:

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

For any Markov Decision process ...

- There is an optimal policy π_* which is better (or equal) than all other policies.
- Every optimal policy achieves the optimal value function and the optimal action-value function.

Recap Example – Calculation of Value Function

Discounting Future Returns

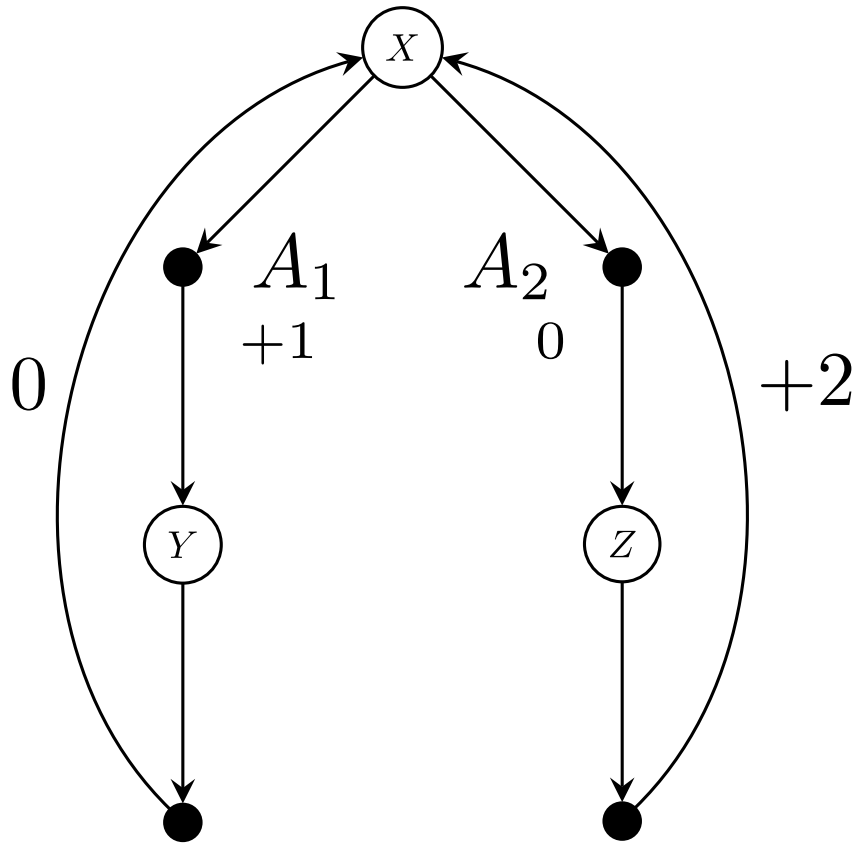
For $\gamma = 0.9$:

$$v_{\pi_1}(X) = \sum_{k=0}^{\infty} 0.9^{2k} = \frac{1}{1 - 0.9^2} \approx 5.3$$

$$v_{\pi_2}(X) = \sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 = \frac{0.9}{1 - 0.9^2} * 2 \approx 9.5$$

with π_1 always selecting A_1 and π_2 always selecting A_2.

$$\pi_2 \geq \pi_1$$



Optimal Value Function

The optimal state-value function $v_*(s)$ is the maximum value function over all policies:

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

The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

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

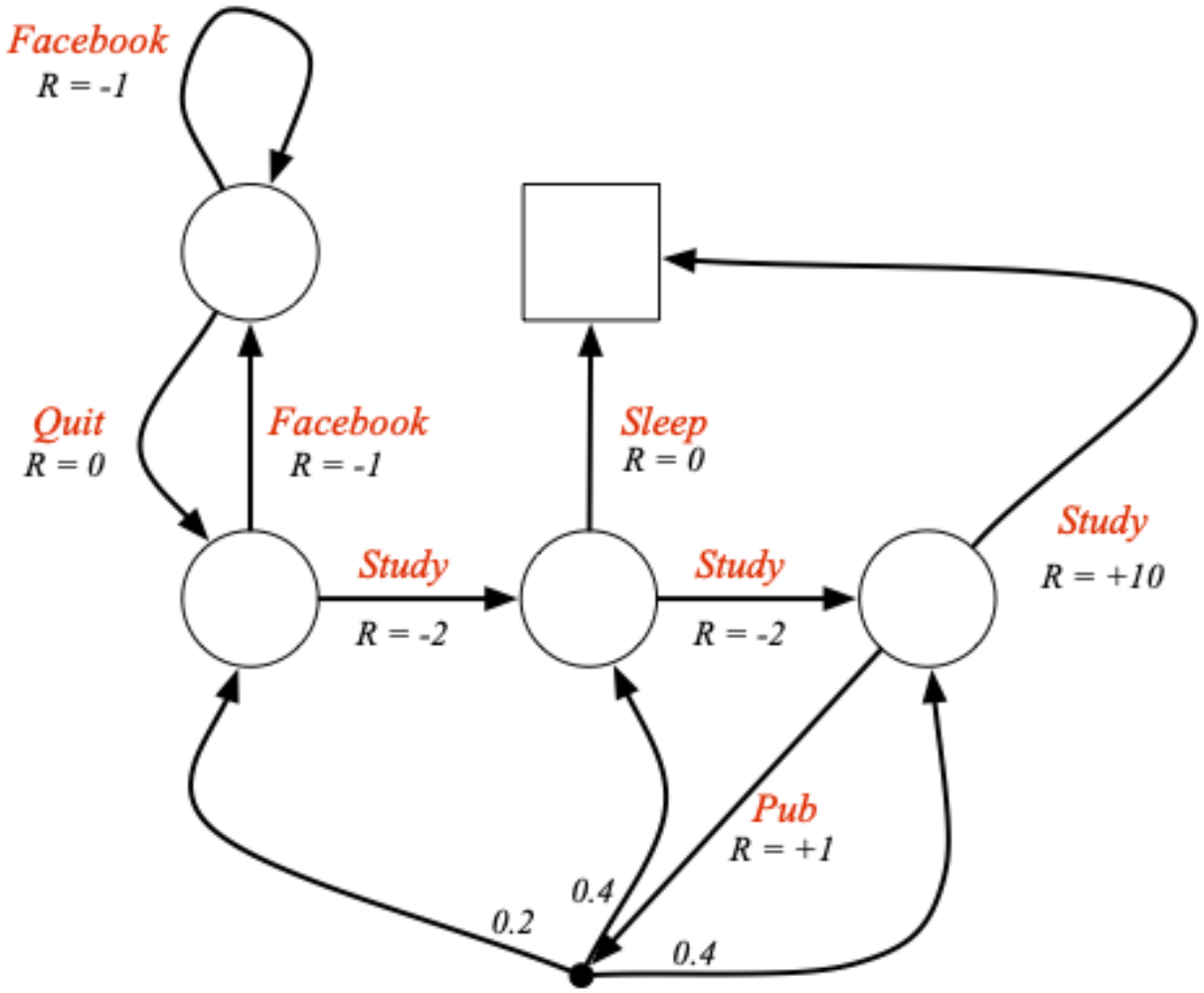
- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value function.

(Silver 2015)

Example: Optimal Value Function

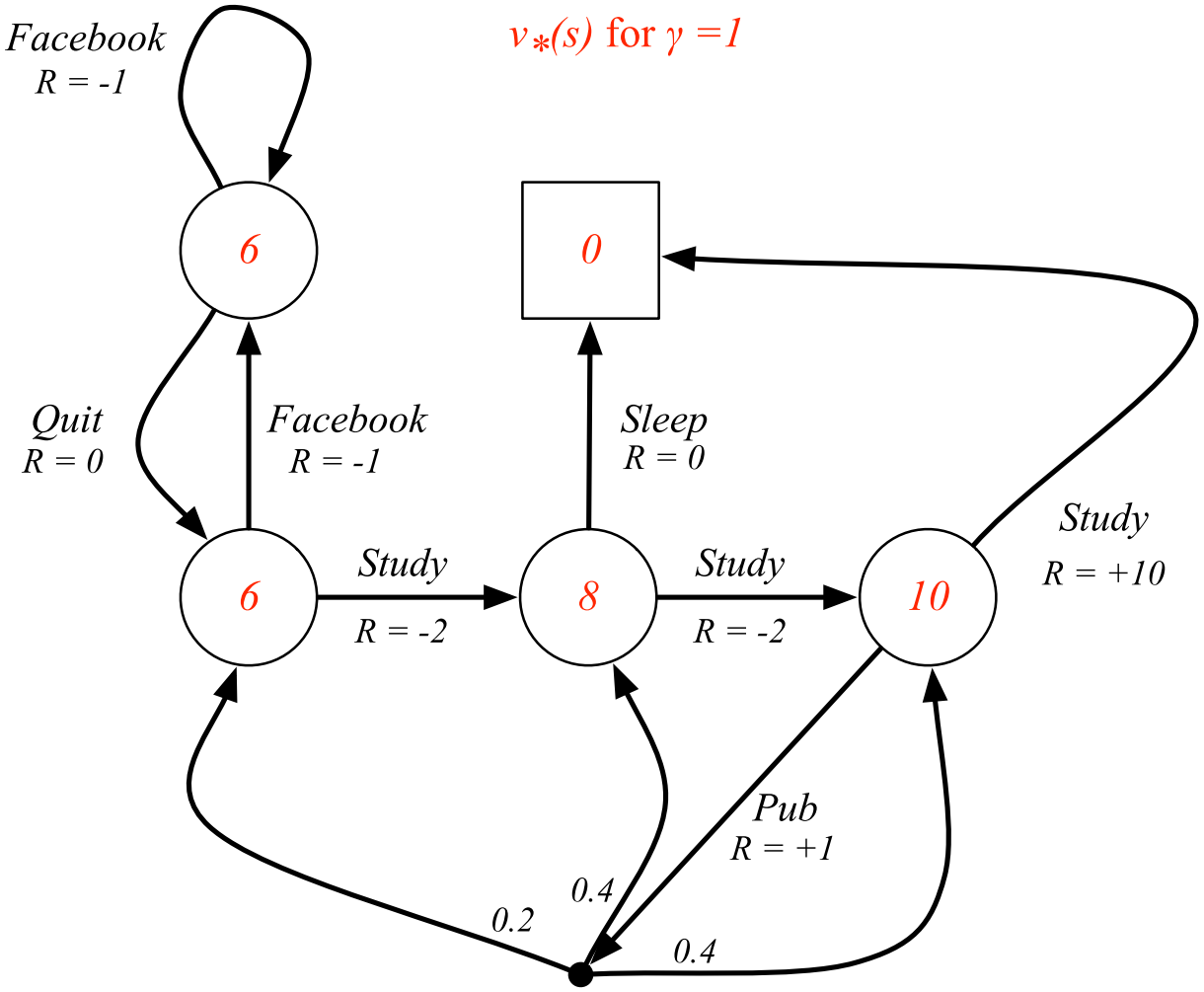
For an undiscounted MDP ($\gamma = 1$),

- What is the optimal value for each state?
- How do you compute the optimal value?



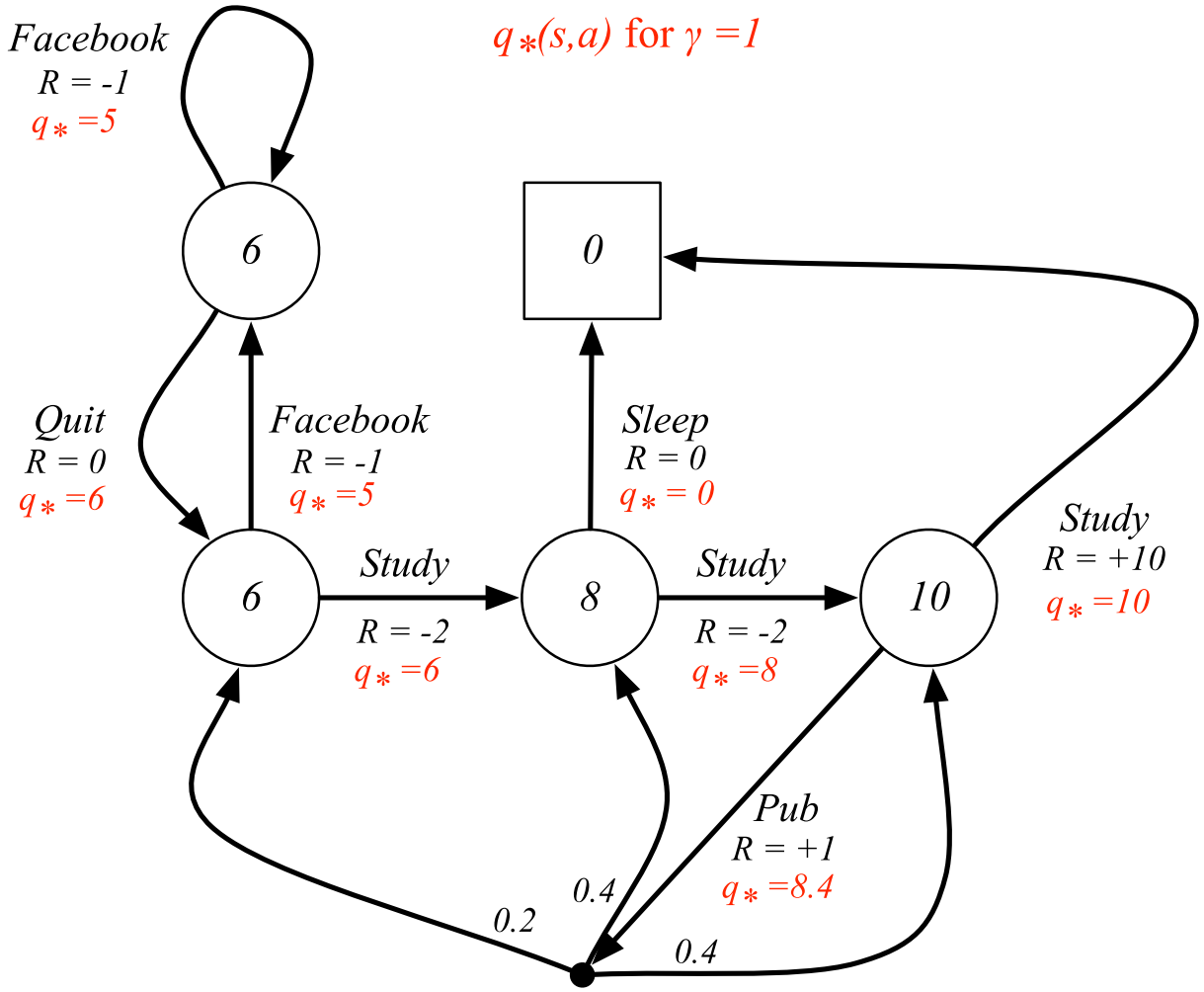
(Silver 2015)

Example: Optimal Value Function



(Silver 2015)

Example: Optimal Action-Value Function



(Silver 2015)

Finding an Optimal Policy

We can find an optimal policy by directly acting greedy on the optimal action-state value function $q_*(s, a)$:

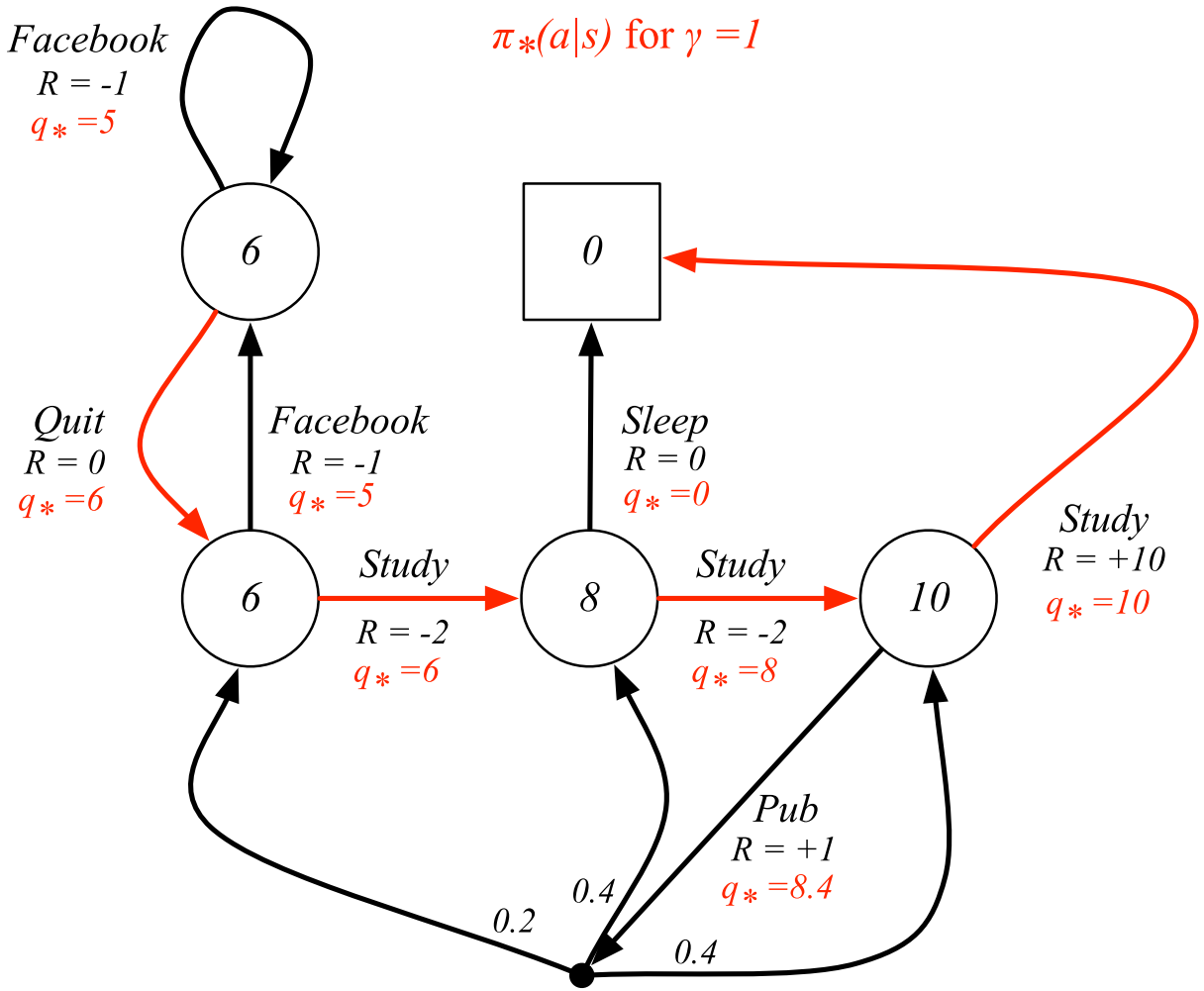
$$\pi_*(s, a) = \begin{cases} 1, & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0, & \text{otherwise} \end{cases}$$

Note:

- There is always a deterministic optimal policy for any MDP: $\pi_* \geq \pi, \forall \pi$
- There can be multiple optimal policies.
- If multiple actions maximize q_* in a state, we can simply pick any of these (including stochastically)

(Hasselt und Borsa 2021)

Example: Optimal Policy



(Silver 2015)

Recap – Summary Bellman Expectation Equations

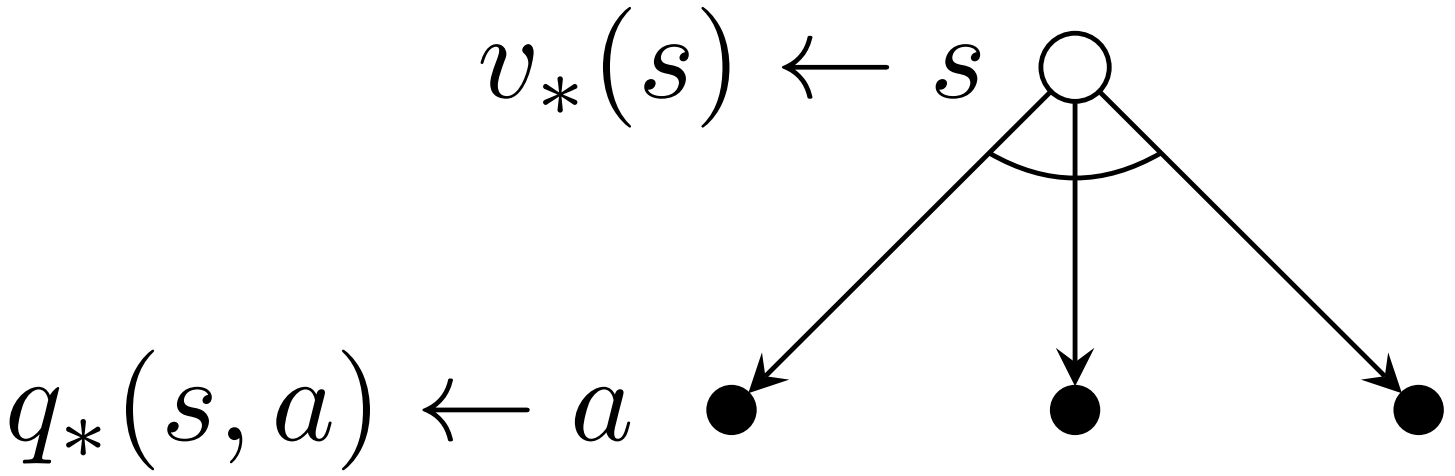
Bellman Expectation Equation

For a given MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ for any policy π, the value functions obey these expectation equations:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s') \right)$$

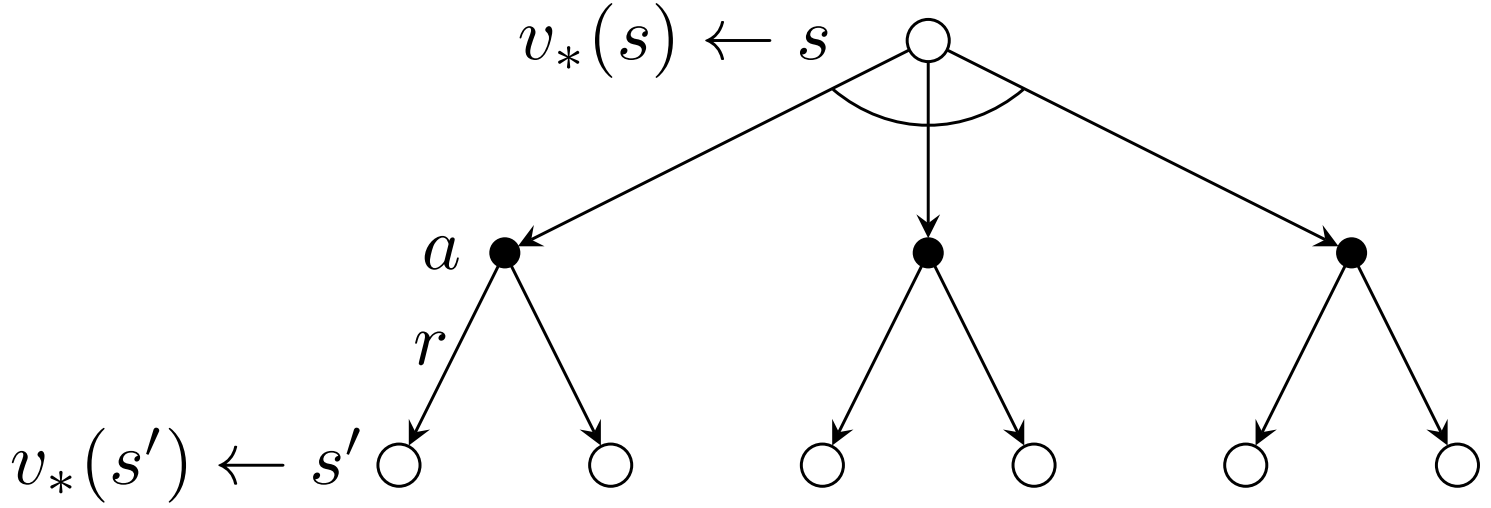
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Bellman Optimality Equation for V_*



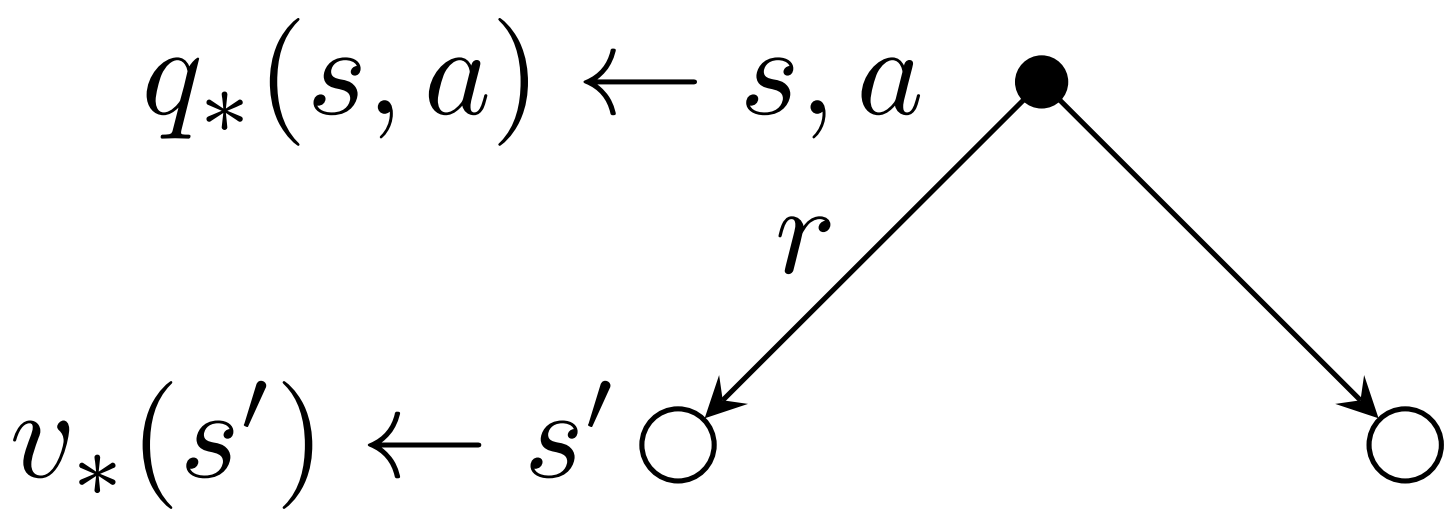
$$v_*(s) = \max_{a \in \mathcal{A}} q_*(s,a)$$

Bellman Optimality Equation for V_* (2)



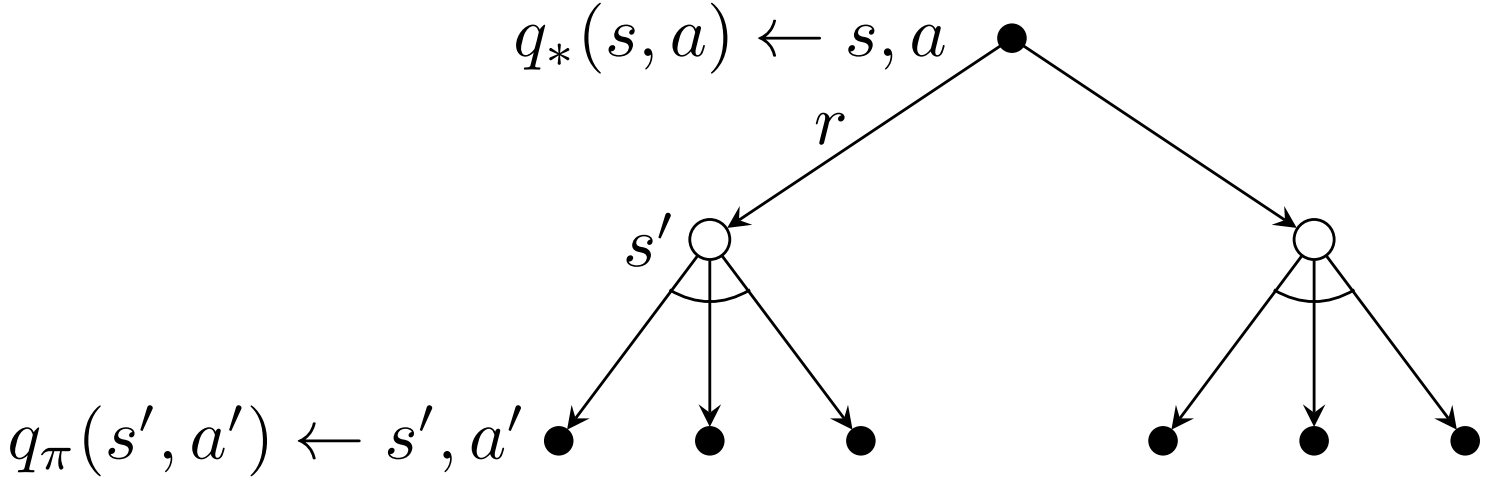
$$v_*(s) = \max_{a \in \mathcal{A}} \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_*(s') \right)$$

Bellman Expectation for Q_*



$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_*(s')$$

Bellman Expectation for Q_* (2)



$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q_*(s',a')$$

Solving the Bellman Optimality Equation

- The Bellman optimality equation is non-linear
- Multiple iterative solution methods:
  - Using models – dynamic programming
    - Value iteration
    - Policy iteration
  - Using samples
    - Monte Carlo Approaches
    - Q-learning
    - SARSA

(Hasselt und Borsa 2021)

Summary – Optimal Value Function and Policy

The goal in RL is to act optimally – this is possible through learning an optimal value function or directly an optimal policy.

The optimal value function produces the maximum return:

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

The optimal policy achieves optimal value functions:

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

These are directly related as

$$v_{\pi_*}(s) = v_*(s), q_{\pi_*}(s,a) = q_*(s,a)$$

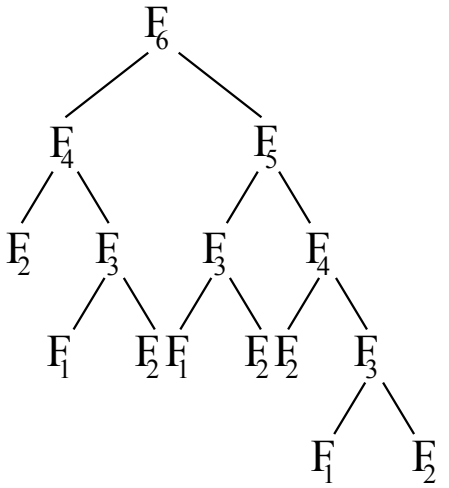
(Weng 2018)

Dynamic Programming

Dynamic Programming

In general, dynamic programming relies on two parts:

- Optimal substructure: The problem can be broken down into subproblems that provide partial solutions that can be used to solve the overall problem.
- Overlapping sub-problems: Sub-problems occur many times so that they can be cached for later reuse.



Example for Fibonacci Numbers shown as a recursive (inefficient) tree

$$F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$$

History of the Name Dynamic Programming

From Multistage Decision Making …

> *— The 1950s were not good years for mathematical research … [the] Secretary of Defense … had a pathological fear … of the word, research… His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical… I felt I had to … shield … Air Force from the fact that I was … doing mathematics.*

… to Dynamic Programming

> *— What title, what name, could I choose? … I was interested in planning, in decision making, in thinking … I decided therefore to use the word, `programming.' I wanted to get across the idea that this was dynamic, … [which] has an absolutely precise meaning, … It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. … It was something not even a Congressman could object to. So I used it as an umbrella for my activities.*

(Bellman 1984, p. 159)

The essence of reinforcement learning is memorized (context-sensitive) search.

(Sutton und Barto 2018)

Dynamic Programming for MDPs

In general, dynamic programming relies on two parts:

- Optimal substructure: The problem can be broken down into subproblems.
- Overlapping sub-problems: Sub-problems occur many times.

Dynamic programming algorithms allow to deal with planning problems: This means, the **complete model and environment is known (the MDP)**.

Markov Decision Processes satisfy both properties

- The Bellman Equation gives recursive decomposition: we have the optimal behaviour of the next step and then the estimated optimal value of the remaining steps.
- The Value function stores solutions for reuses: It caches the optimal achievable reward from a state.

Principle of Optimality for a Policy

As we are searching an optimal policy, this optimal policy can be divided into two parts:

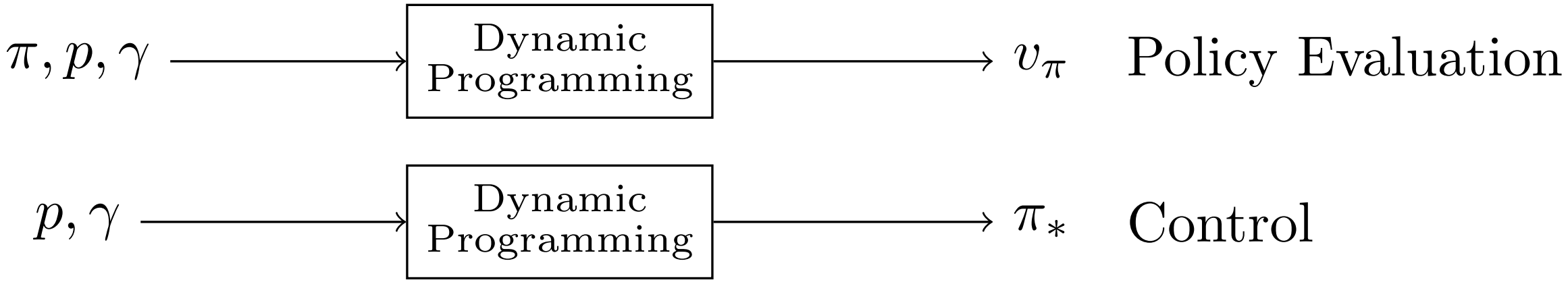
- An optimal immediate action a,
- and following the optimal policy afterwards from new state s'.

Principle of Optimality

A policy $\pi(a|s)$ achieves the optimal value from a state s as $v_\pi(s) = v_*(s)$, iff:

- $\forall s'$ reachable from s
- π achieves the optimal value from state $s', v_\pi(s') = v_*(s')$

Policy Evaluation and Control



DP uses the various Bellman equations along with **full** knowledge of the joint probability to work out value functions and optimal policies.

Classical DP does not involve interaction with the environment at all.

Summary Bellman Optimality Equations

Bellman Optimality Equation

For a given MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ the optimal value functions obey these expectation equations:

$$v_*(s) = \max_{a \in \mathcal{A}} \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_*(s') \right)$$

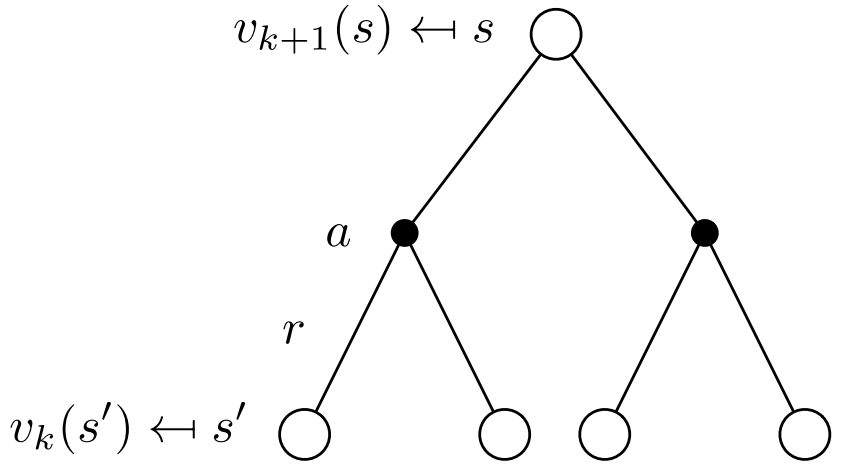
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q_*(s',a')$$

DP: Policy Evaluation

Policy Evaluation is to compute the state-value v_π for a given policy π:

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}_\pi[r + \gamma v_k(s') | S = s] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)(r + \gamma v_k(s'))
\end{aligned}
$$

It iteratively applies the Bellman expectation backup and converges.



(Weng 2018)

Policy Evaluation: Iterative Convergence of Value Function

Given is a policy π for an MDP and we want to find the corresponding value function.

Approach: Iteratively apply Bellman expectation: $v_1 \longrightarrow v_2 \longrightarrow \cdots \longrightarrow v_\pi$

Input: π, the policy to be evaluated; algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$ for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$.

while $\Delta \geq \theta$ **do**

  $\Delta \leftarrow 0$

  for $s \in \mathcal{S}$ **do**

    $v \leftarrow V(s)$

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

    $\Delta \leftarrow \max\left(\Delta, |v - V(s)|\right)$ `// Track largest update to state-value function.`

  end

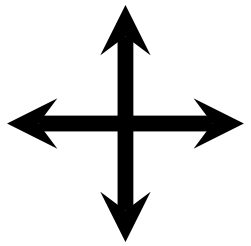
end

Grid world example

Given a grid word (MDP)

- $\mathcal{S}$: 16 states as places, two terminal states
- $\mathcal{A}$: 4 actions into main directions
- R: -1 for each step
- $\gamma = 1$ (undiscounted)

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



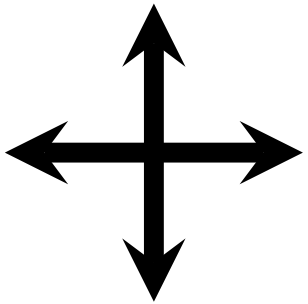
random actions

Reward: -1 for each step

(Sutton und Barto 2018)

Policy Evaluation $k = 0$

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

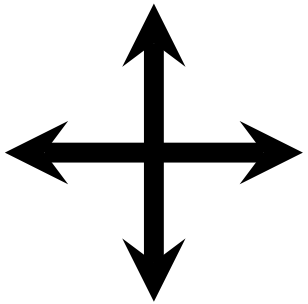


random actions

Reward: -1 for each step

Policy Evaluation $k = 1$

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

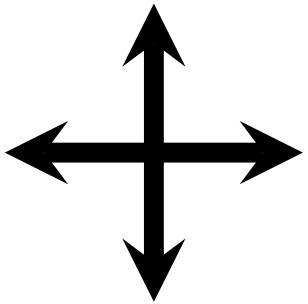


random actions

Reward: -1 for each step

Policy Evaluation $k = 2$

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

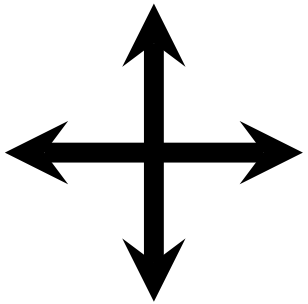


random actions

Reward: -1 for each step

Policy Evaluation $k = 3$

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

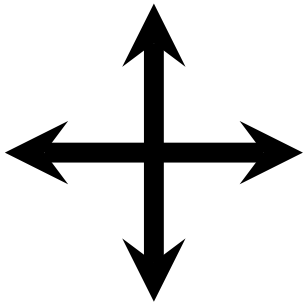


random actions

Reward: -1 for each step

Policy Evaluation $k = 10$

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

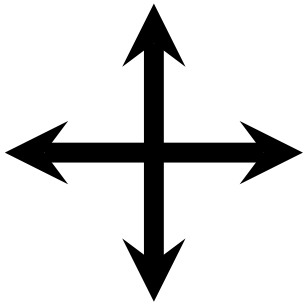


random actions

Reward: -1 for each step

Policy Evaluation $k = \infty$

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



random actions

Reward: -1 for each step

Policy Improvement

We can use v_π to change π. We are interested if for some state s we can pick a better action a then proposed by π.

$v_\pi(s)$ tells us how valuable it is to follow π from this current state. Selecting a different action a from s and only afterwards following π is given as

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

If we find a better action, we can improve on our current policy π when encountering state s and define this as an improved policy $\pi' \geq \pi$.

Convergence of Policy Improvement

Consider we start with a deterministic policy $a = \pi(s)$.

We can improve this policy by acting greedy on the action-value function

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$$

which improves the value from any state s over one step

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

Convergence of Policy Improvement (2)

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

improves the value function as well, $v_{\pi'} \geq v_\pi$:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s\right] = v_{\pi'}(s)
\end{aligned}$$

Policy Improvement towards optimal policy

If a new greedy policy π' is as good as, but not better than, the old policy π. Then $v_\pi = v_{\pi'}$ (and if we choose π' acting greedy on v_π): $v_{\pi'} = v_* $.

So, if the improvment of our policy stops:

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s),$$

which directly satisfies the Bellman equation

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a).$$

Policy improvement therefore must give a strictly better policy except when the original policy is already optimal.

Policy Improvement Theorem

Policy Improvement Theorem

For any deterministic policies π and π' with for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

it holds that policy π' is as good as or better than π; i.e. it must obtain greater than or equal expected return for all states $s \in \mathcal{S}$: $v_{\pi'}(s) \geq v_\pi(s)$.

Finding an optimal policy

Policy improvement converges towards optimal policies.

Policy Iteration (Control)

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

Policy evaluation: $\xrightarrow{\text{E}}$

Policy improvement $\xrightarrow{\text{I}}$

For deterministic policies: each policy is guaranteed to be strictly better until we reach the optimal policy.

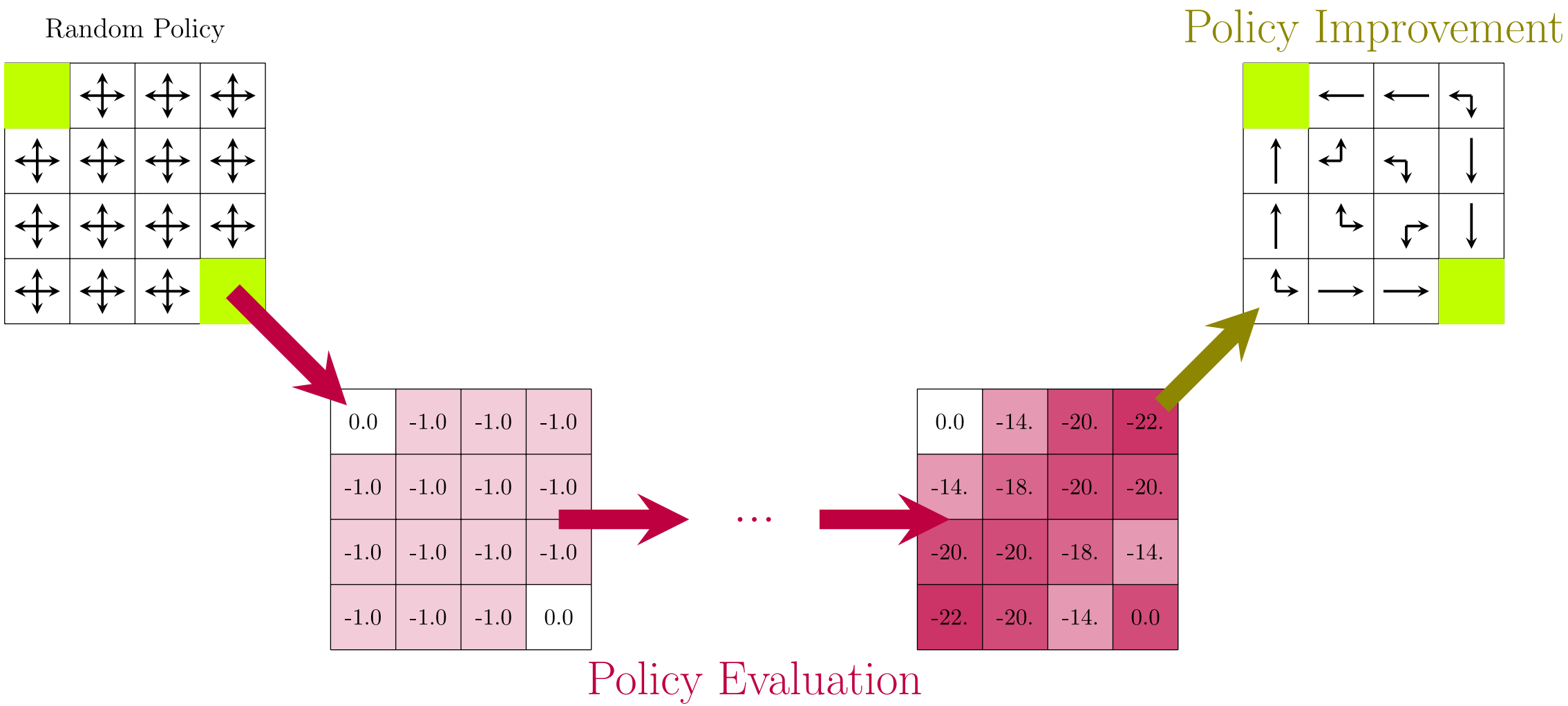
For finite MDP: $\exists$ only a finite number of deterministic policies; therefore this converges to an optimal policy and an optimal value function in a finite number of iterations.

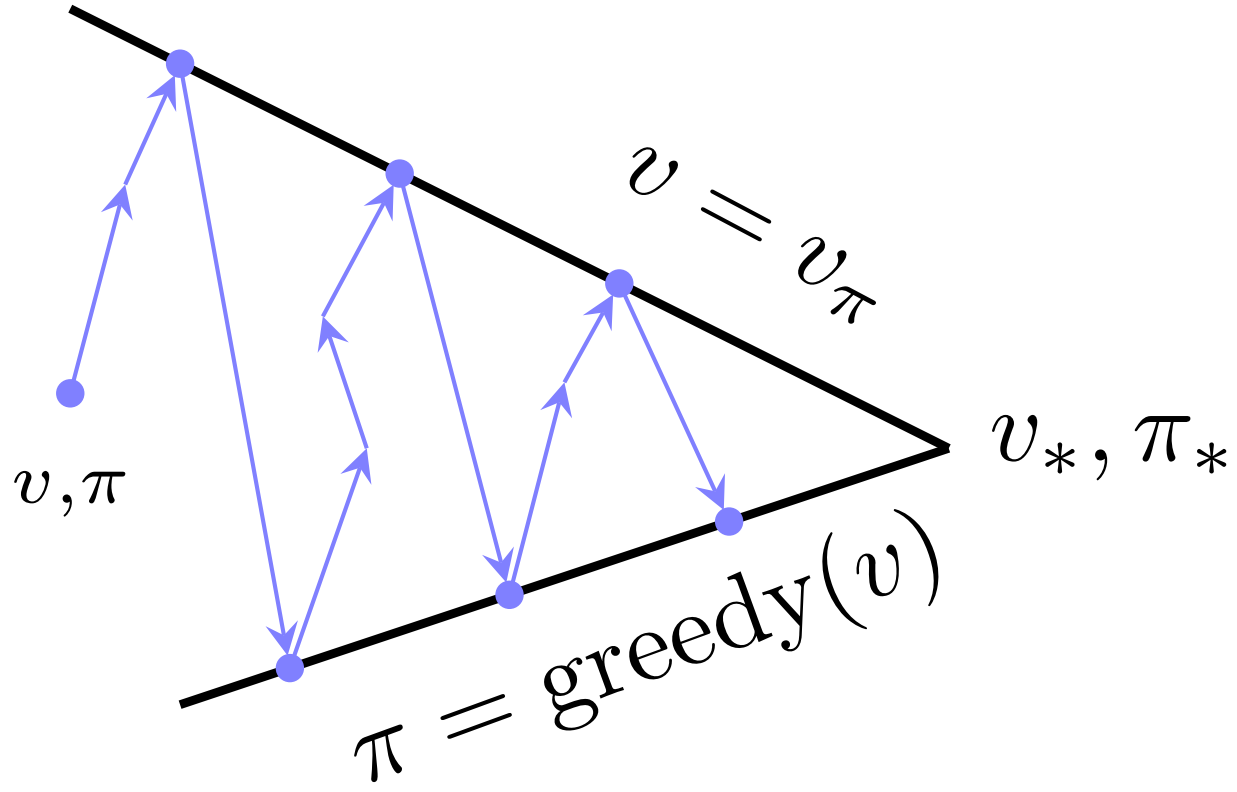
Stochastic optimal policies

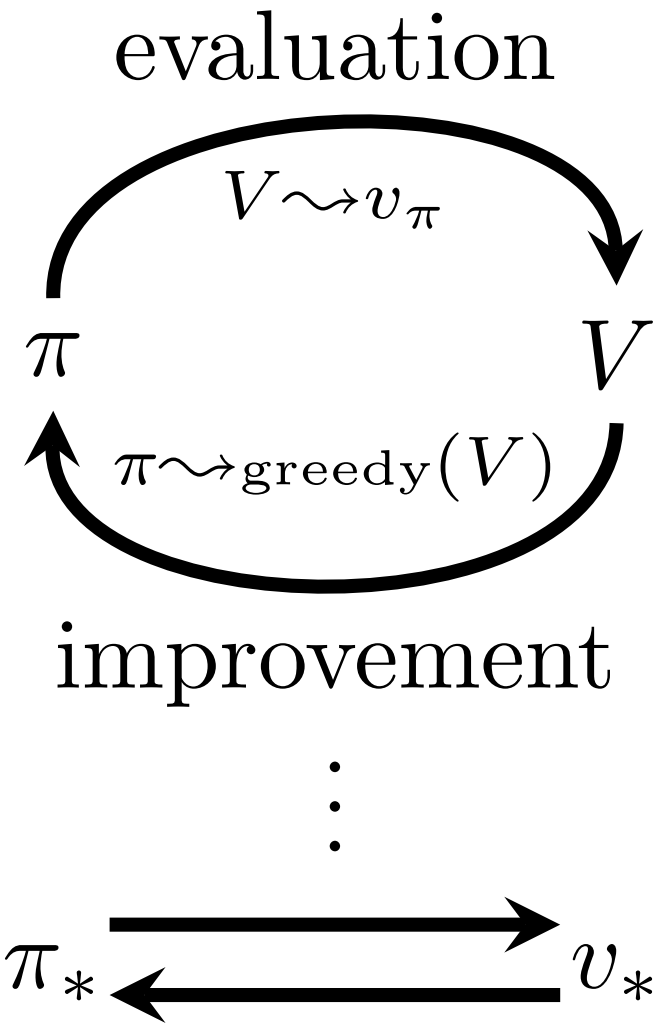
For a policy π action selection can be defined in a probabilistic way through $\pi(a|s)$ for taking each action a in each state s.

When there are ties in the policy improvement steps – there are several actions with maximum value – then we can select any of these maximal actions or can distribute the probability among these in any way (as long as all submaximal actions are given zero probability.

Policy Iteration Example







Policy evaluation = Estimate v_π

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

Policy Improvement Example

You can step **policy evaluation** and **policy improvement** in this interactive grid environment.

- What is worst case complexity for finding an optimal policy (n states, m actions)?
- Observe convergence towards the optimal policy using a form of policy improvement!

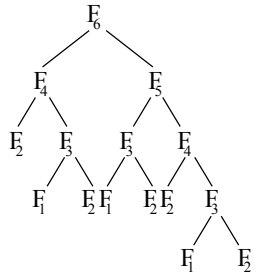
- How do you proceed?

Efficiency of Dynamic Programming

MDP with n the number of states and k for action.

Worst Case Scenarios

Considering Fibonacci Numbers: 2^n steps when not caching solutions (without DP).



For **MDP**: Number of possible deterministic policies is in k^n.

Dynamic Programming

Exploiting structure and reusing solutions leads to $O(n)$ for Fibonacci numbers.

For MDP: worst case work is **only** polynomial in the number of states and actions.

MDPs of growing state space are considered difficult and problematic – but this is inherent and mostly not stemming from using DP.

Required Space

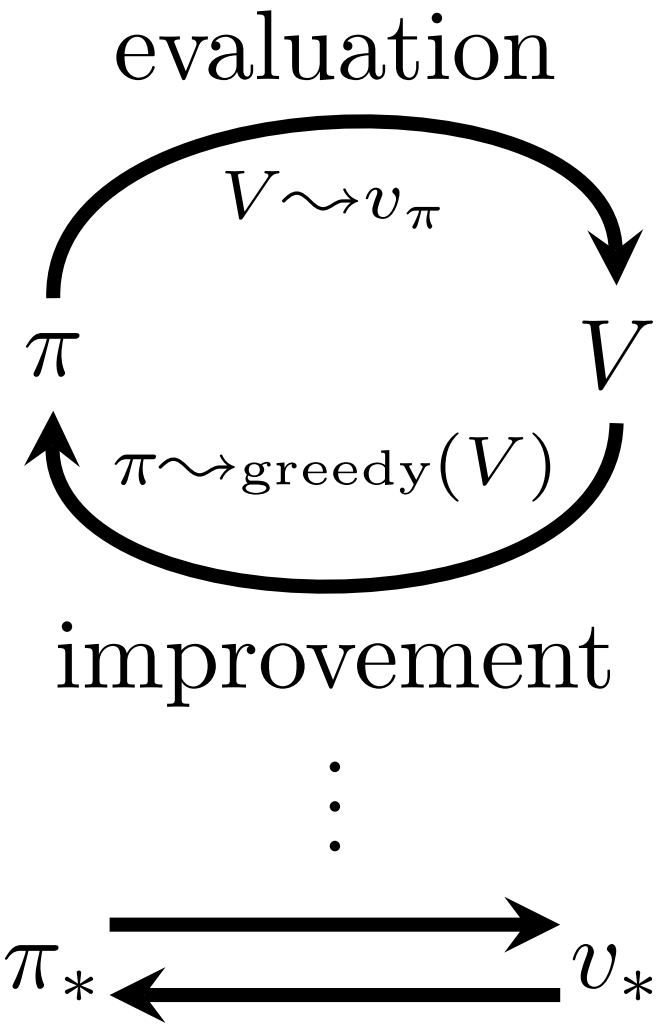
For high dimensional state spaces: asynchronous methods are preferred

- policy improvement

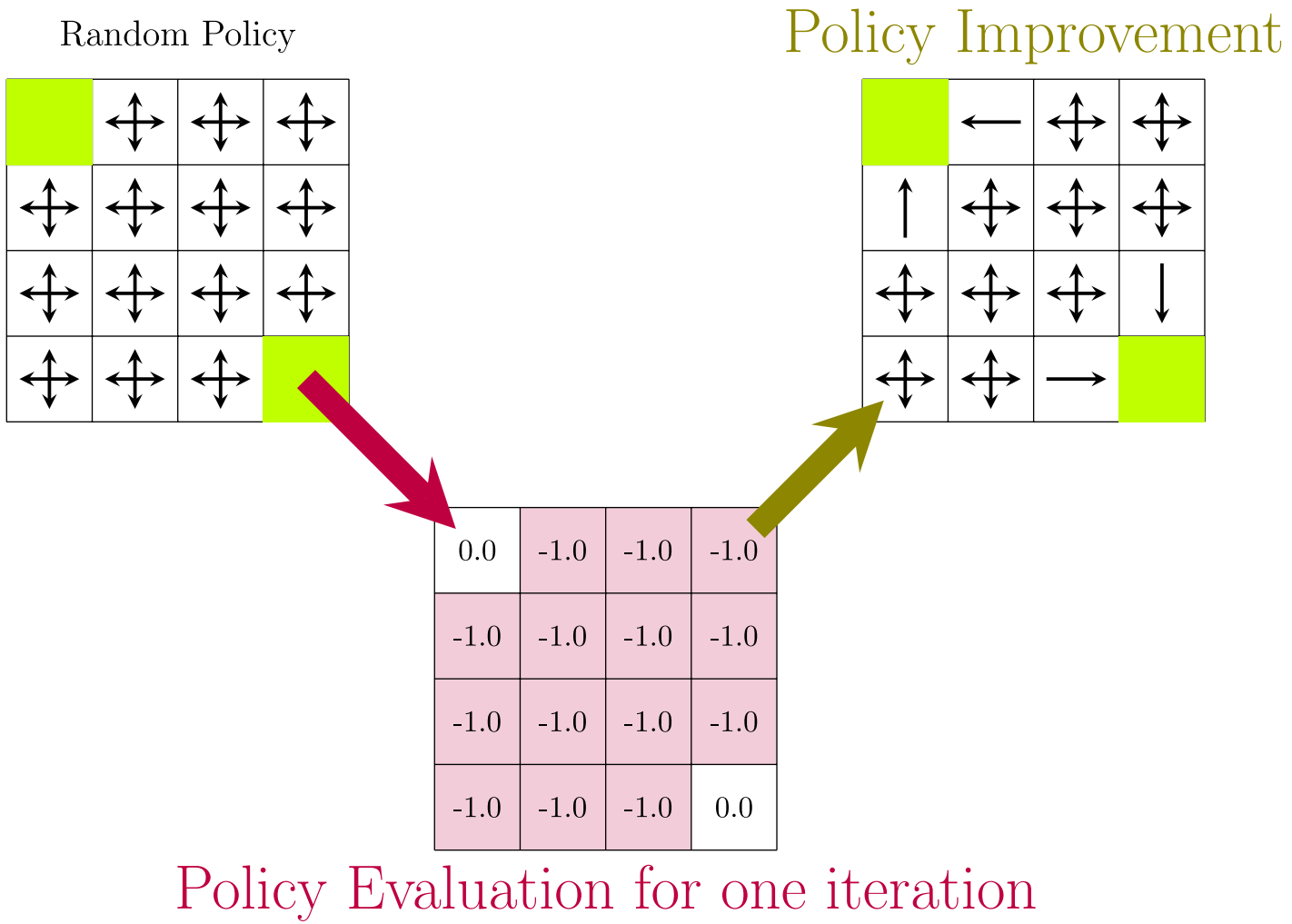
in interaction. Importantly, this could be as well done asynchronously.

Most RL learning methods can be described as GPI: they consist of a policy and value function.

When evaluation and improvement process each converge, then value function and policy are optimal – the policy is greedy wrt. the stable value function. This implies: the Bellman optimality equation holds.



Value Iteration Example

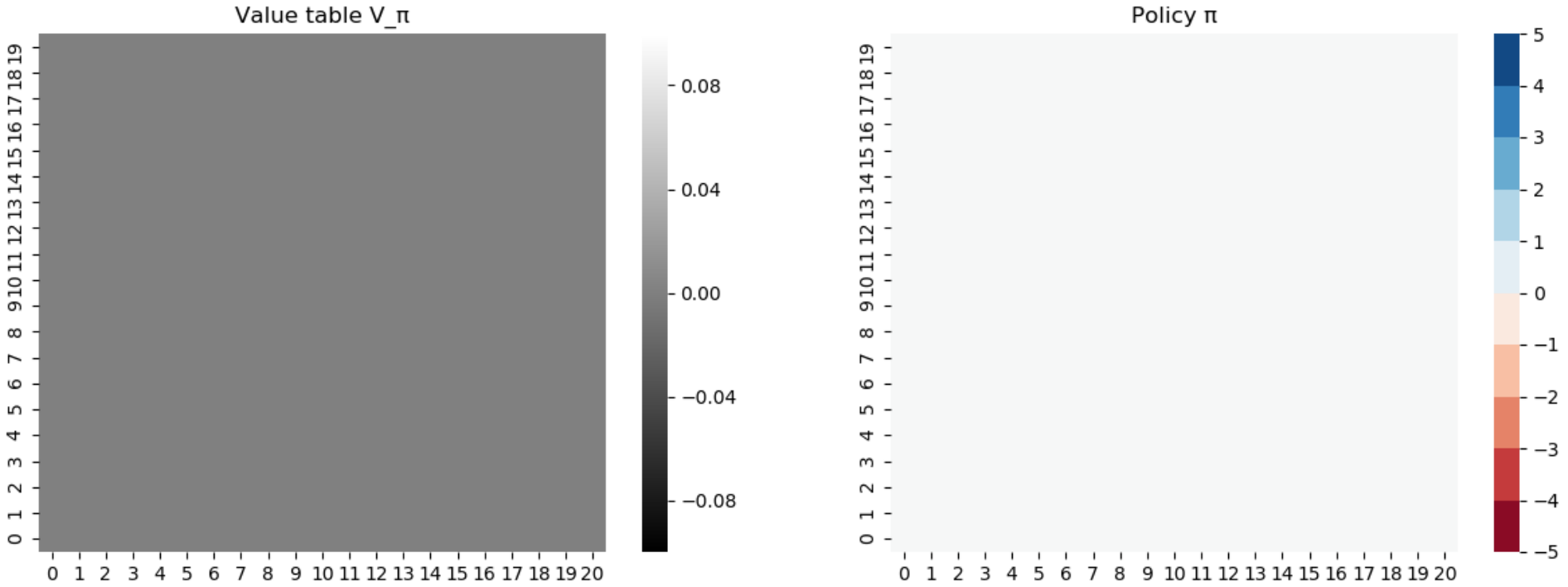


Jack's Car Rental

- State Space: For two locations, each maximum of 20 cars
- Actions: Move up to 5 cars overnight
- Reward: $10 for each rented car
- Transitions of Environment: Returns and Requests are probabilistic
  - following Poisson distribution, n returns (same for requests) with probability $\frac{\lambda^n}{n!}e^{-\lambda}$
  - location A: average requests is 3, returns also 3
  - location B: average request is 4, returns 2

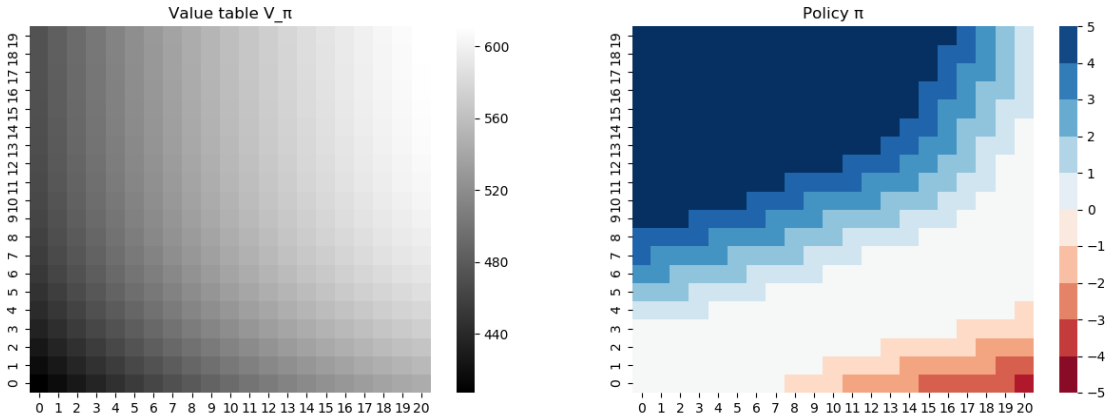
(Sutton und Barto 2018)

Jack's Car Rental – Value Iteration, iteration 0

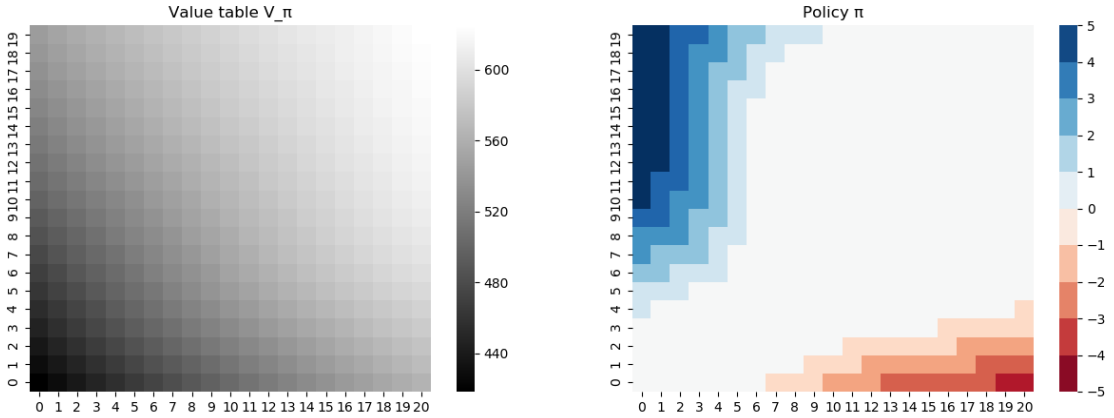


Visualization from (pignatelli_sb_python_2020?)

Jack's Car Rental – Value Iteration, iteration 1 and 2

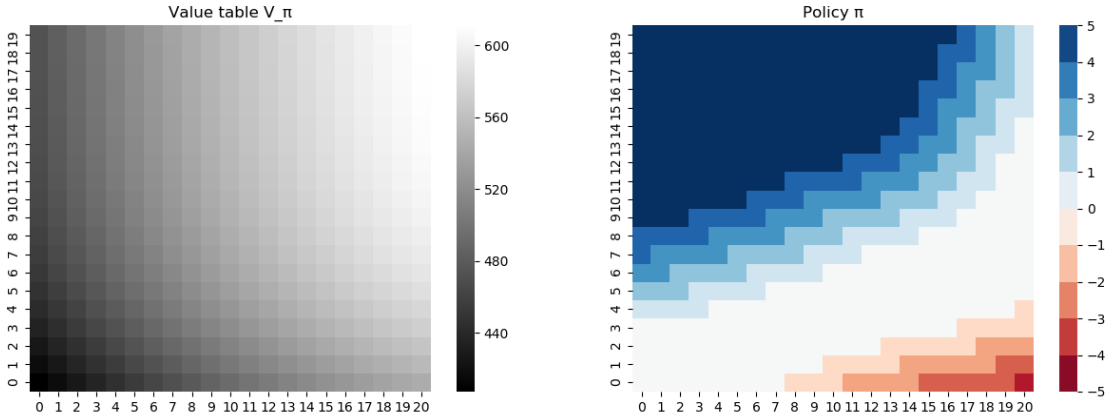


Policy after first iteration

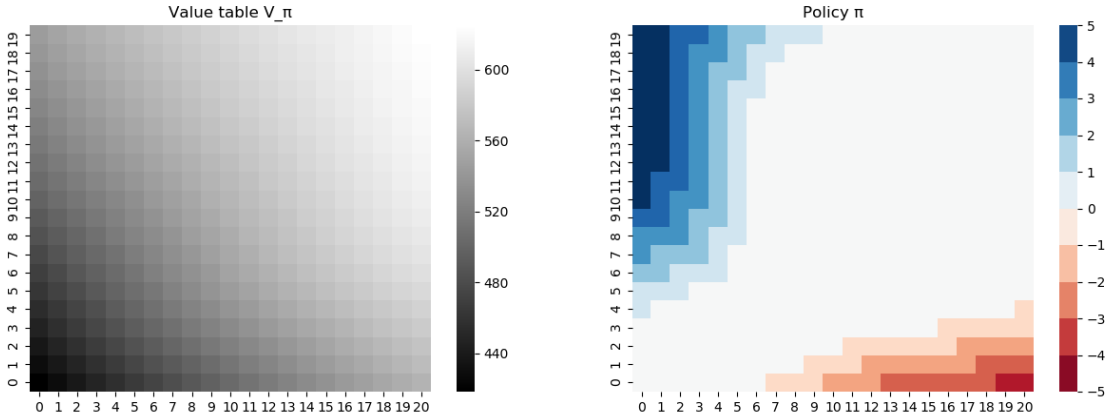


Policy after second iteration

Jack's Car Rental – Value Iteration, iteration 3 and 4

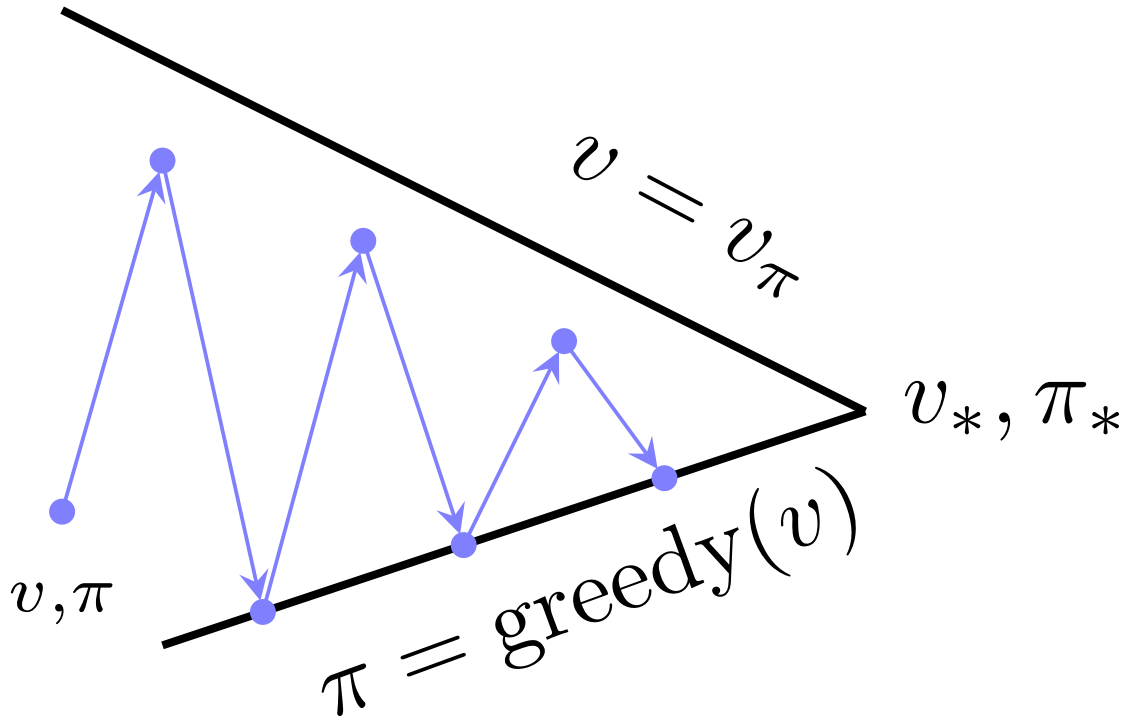


Policy after third iteration



fourth

Value Iteration



Policy evaluation = Estimate v_π

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

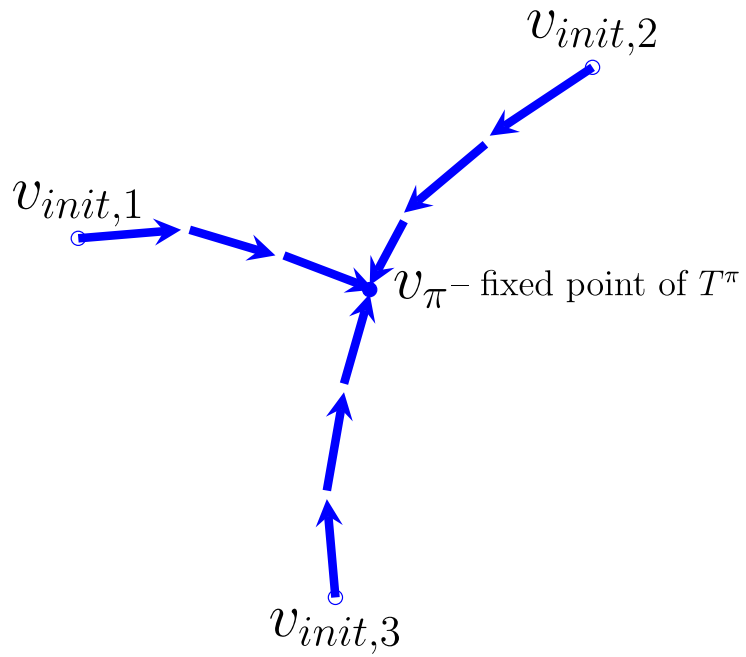
Considering Convergence of Bellman Backup

The Bellman equation for $s \in \mathcal{S}$ is given as

$$v_\pi(s) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v_\pi(s')$$

We can span a vector space $\mathcal{V}$ over all possible value functions of dimensionality $|\mathcal{S}|$ in which each point represents a specific value function.

The Bellman backup alters value functions in this space and we can show that it actually brings value functions between iterations closer together.



Considering Convergence of Bellman Backup (2)

We can measure the distance between two state-value functions u and v using the ∞-norm (the largest difference between state values):

$$|| u - v ||_\infty \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

Bellman operator

The Bellman operator underlying π is defined as a mapping (between value functions) $T^\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$:

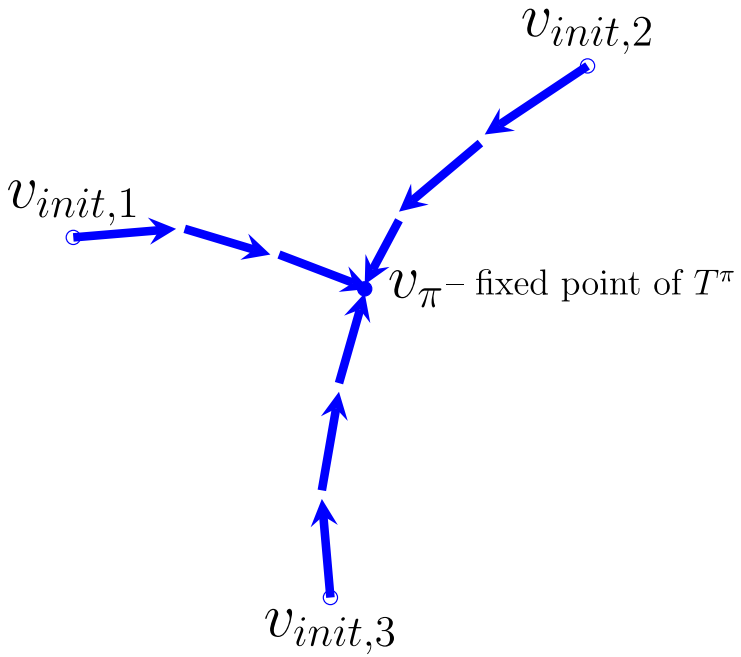
$$(T^\pi v)(x) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s')$$

(Silver 2015)

Considering Convergence of Bellman Backup (3)

If $0 < \gamma < 1$ then T^π is a maximum-norm contraction with a unique fixed point as a solution to $T^\pi v = v$.

The Bellman operator is a γ-contraction – it brings value functions closer together by at least factor γ.



For more details on convergence and derivation using contraction mapping theorem, see (Szepesvári 2010).

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^{π} has a unique fixed point
- which is the value function v_{π} to which it converges (through updates using the Bellman expectation equation).

As a consequence * Iterative policy evaluation converges on v_{π} * and over time policy iteration converges on v_{*}

(Silver 2015)

Example: Robot Playing Soccer – Hierarchical RL

See Video of robots playing soccer

Details see (Huang u. a. 2022)

References

Bellman, Richard. 1984. *Eye of the Hurricane*. World Scientific.

Hasselt, Hado van, und Diana Borsa. 2021. „Reinforcement Learning Lecture Series 2021“. https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021.

Huang, Xiaoyu, Zhongyu Li, Yanzhen Xiang, Yiming Ni, Yufeng Chi, Yunhao Li, Lizhi Yang, Xue Bin Peng, und Koushil Sreenath. 2022. „Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning“. arXiv. doi:10.48550/ARXIV.2210.04435.

Karpathy, Andrej. 2015. „REINFORCEjs“. https://github.com/karpathy/reinforcejs.

Silver, David. 2015. „UCL Course on RL UCL Course on RL UCL Course on Reinforcement Learning“. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Sutton, Richard S., und Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second. The MIT Press.

Szepesvári, Csaba. 2010. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. http://dx.doi.org/10.2200/S00268ED1V01Y201005AIM009.

Weng, Lilian. 2018. „A (Long) Peek into Reinforcement Learning“. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.