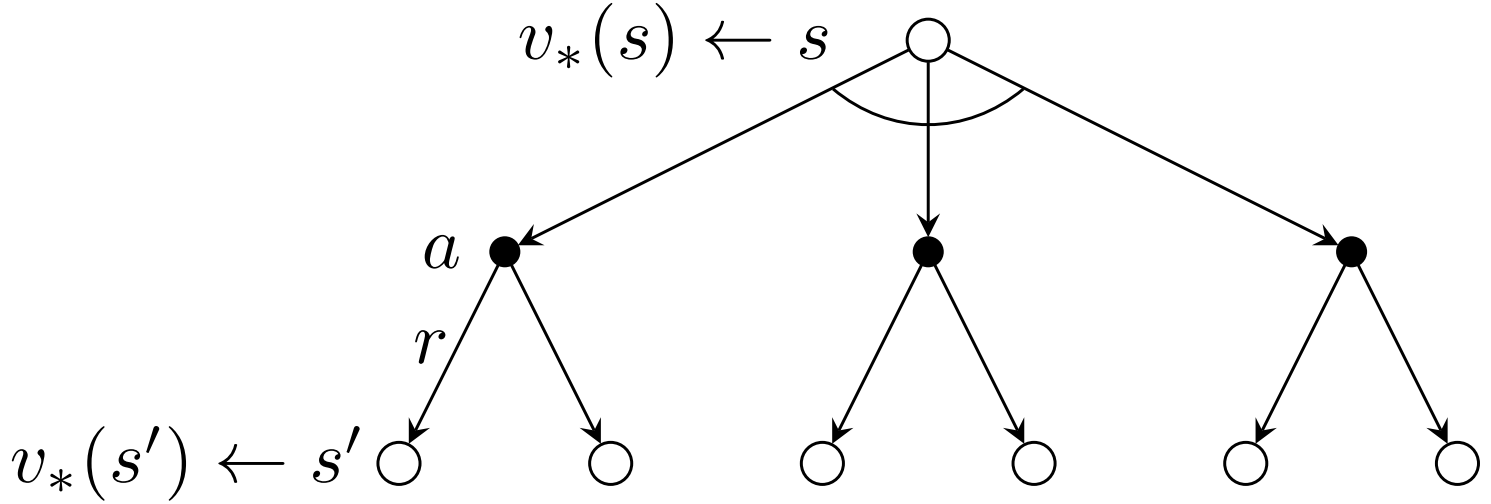
Deep Reinforcement Learning

5 - Model-Free Prediction – Monte-Carlo Methods

Prof. Dr. Malte Schilling

Autonomous Intelligent Systems Group

Recap – Bellman Optimality Equation for V_*



1) How to calculate optimal value function?

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

Recap – 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.

Recap – 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:

$\xrightarrow{\text{E}}$ How do we proceed?

Policy improvement:

$\xrightarrow{\text{I}}$ How do we adjust the policy?

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

Overview Lecture

Last week: Dynamic Programming

- for a known MDP
- solve using planning

General Policy Improvement Approach:

- Policy Evaluation
- Policy Improvement

Next: Model-Free Approaches

- Prediction (this week): Estimate the value function of an unknown MDP
  - Monte-Carlo Method
  - Temporal Difference Learning
- Model-free control (next week):
  - Optimise the policy

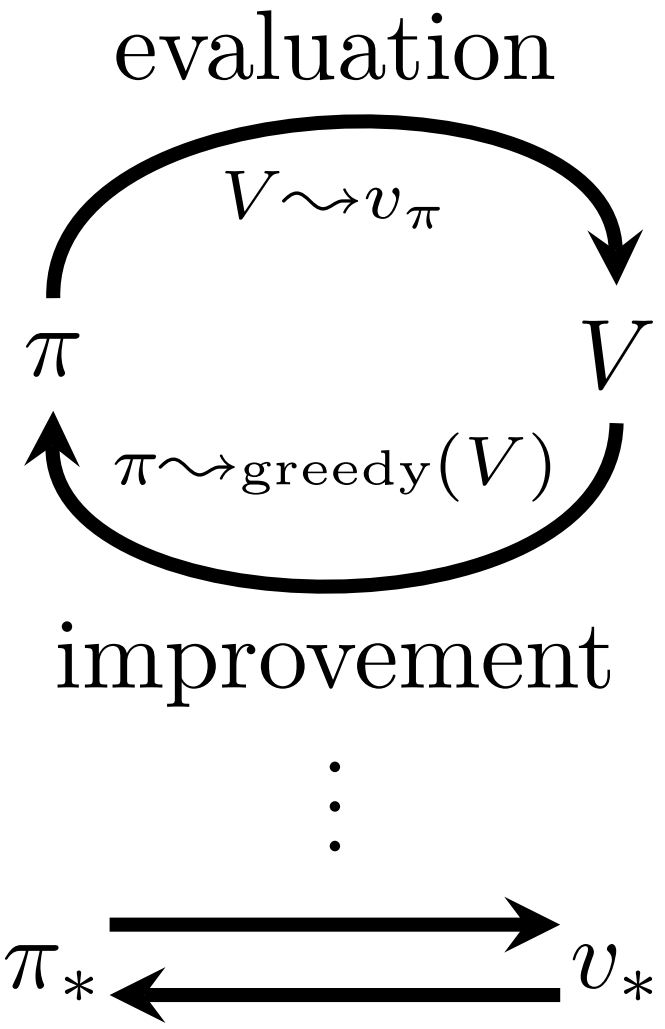
Dynamic Programming (ctd.)

- 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.



Recap – Policy Improvement Example

When stepping the environment: What do we observe?

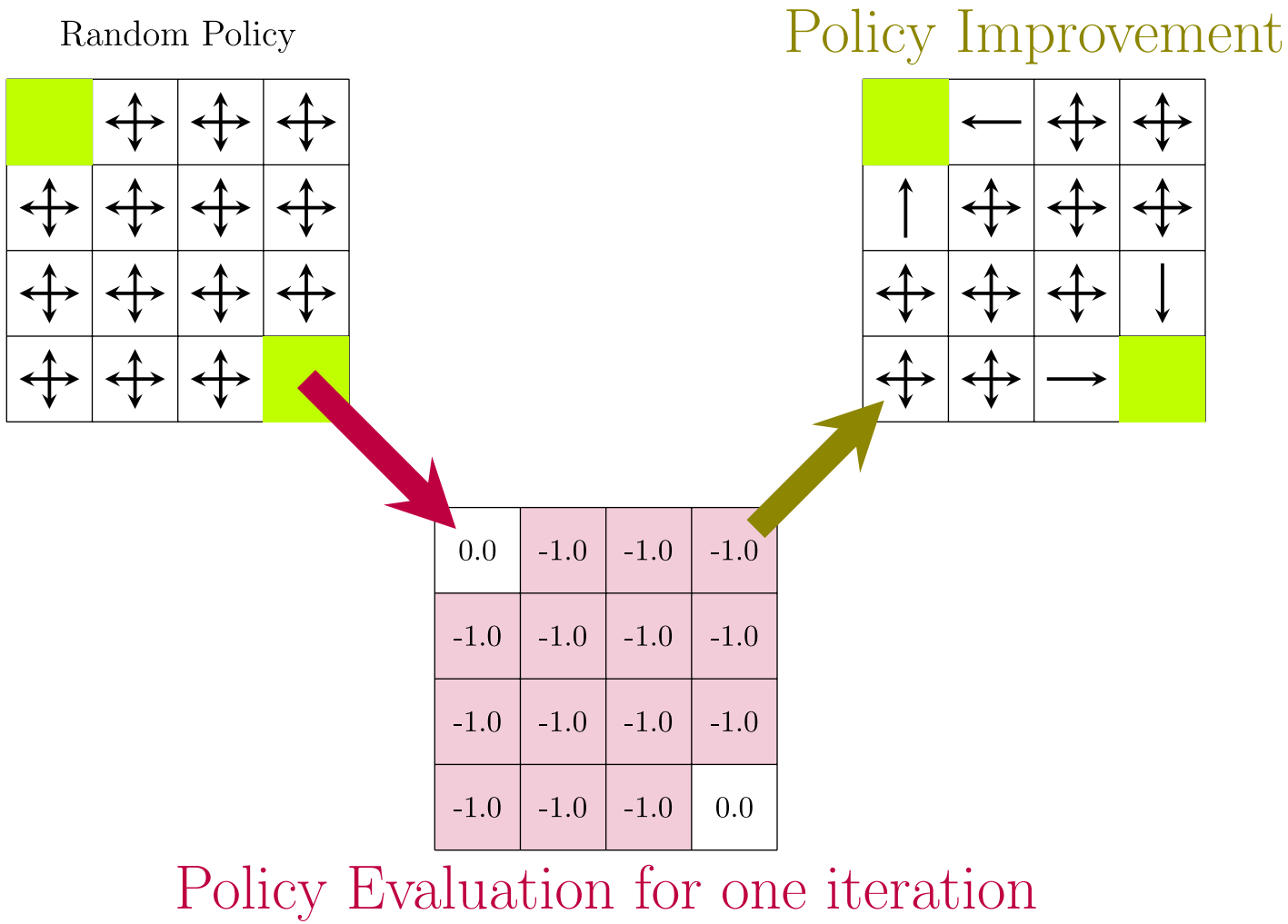
In policy evaluation the value function converges quite quickly and a stable pattern emerges even sooner.

Value Iteration

Make only a single policy evaluation step before immediatle updating policy.

(Karpathy 2015)

Value Iteration Example

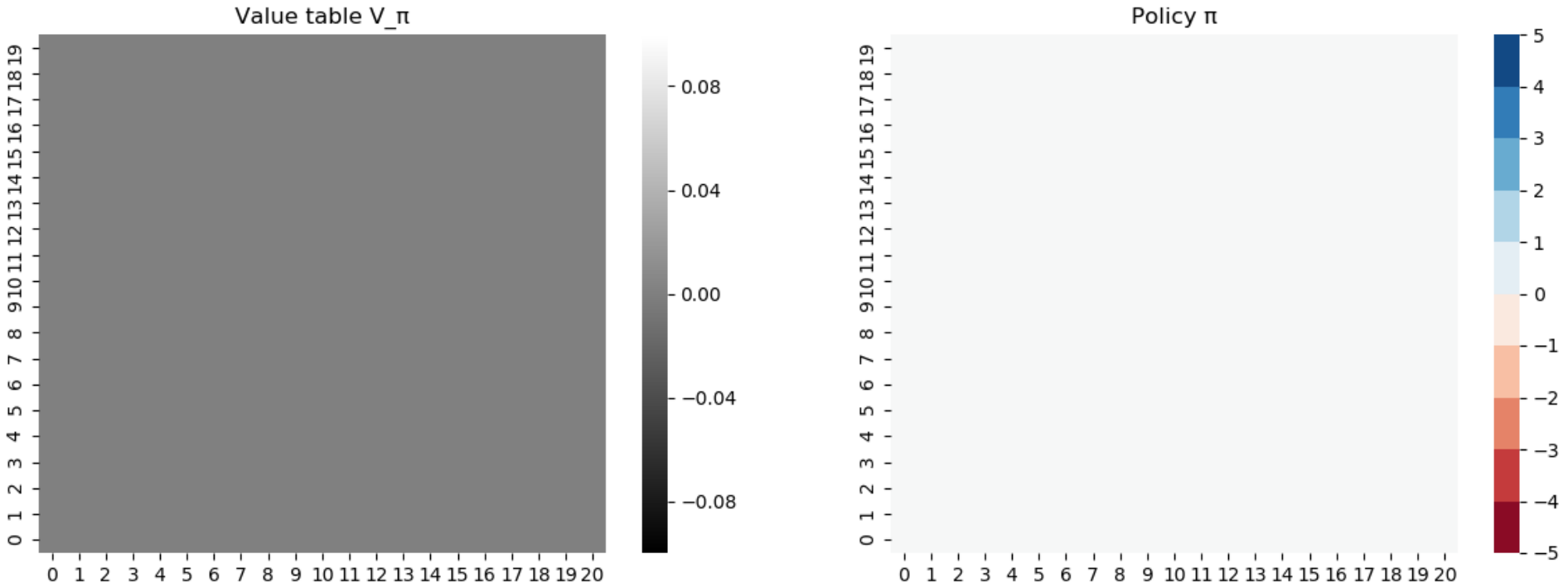


Jack's Car Rental

- State Space: For two locations, each maximum of 20 cars
- Actions: Move up to 5 cars overnight (-2 $)
- 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)

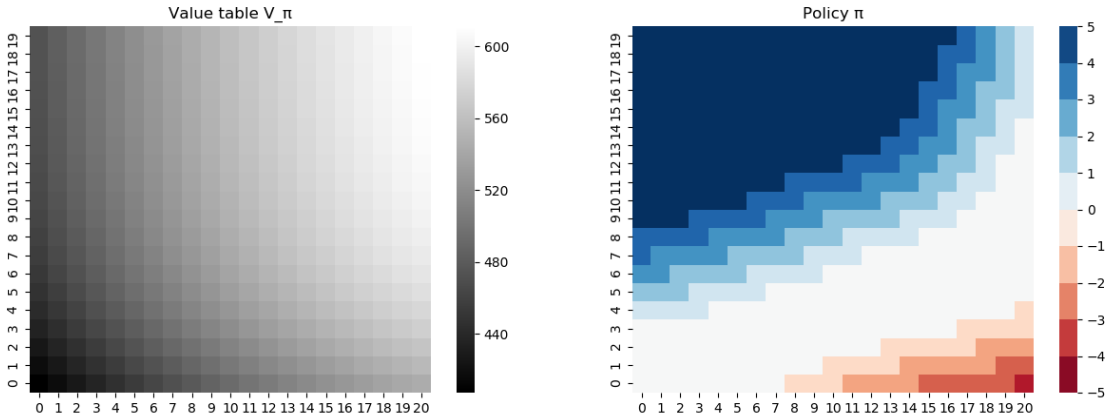
Jack's Car Rental – Value Iteration, iteration 0



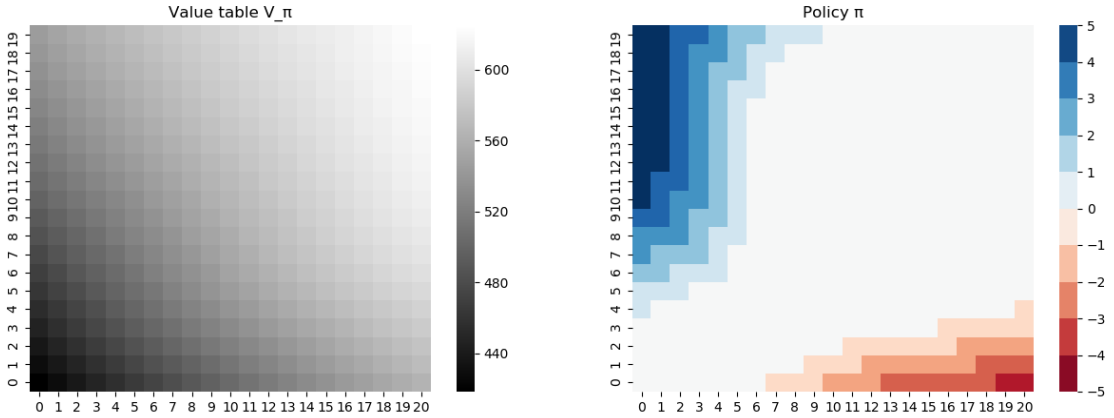
x-axis: number cars at second location, y-axis: number cars at first location

Visualization from (Pignatelli 2022)

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

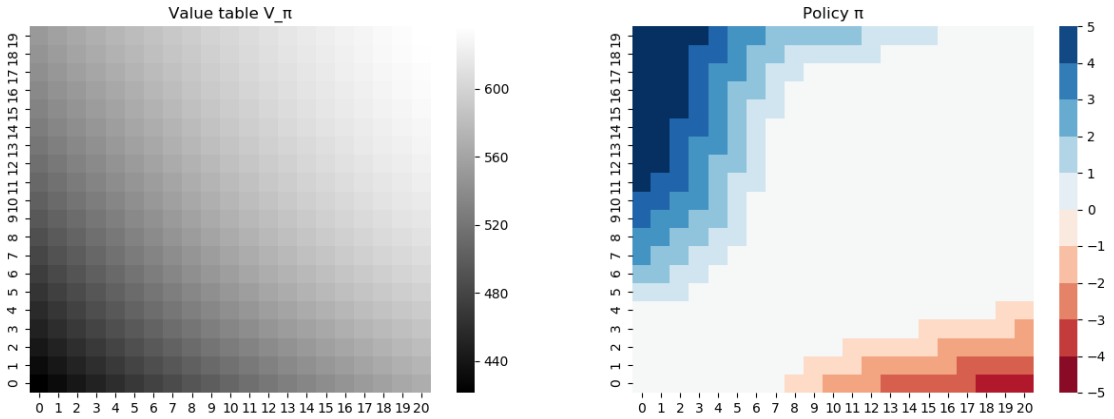


Policy after first iteration

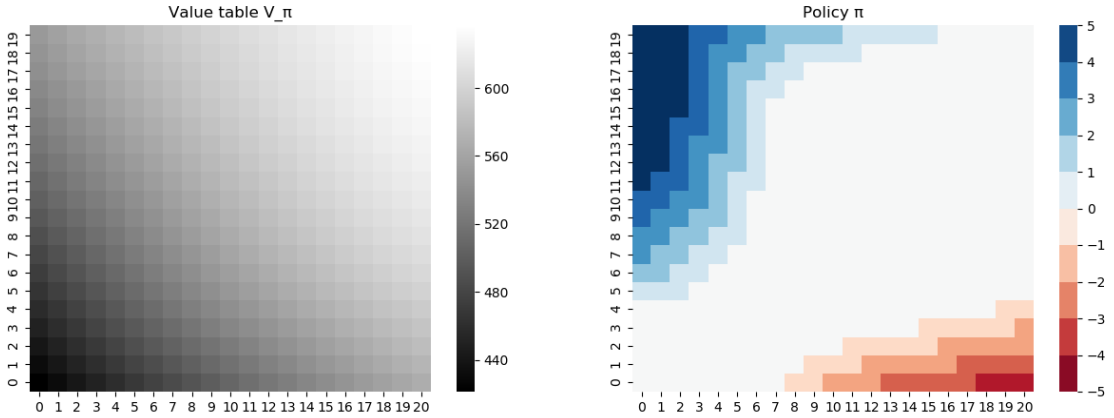


Policy after second iteration

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

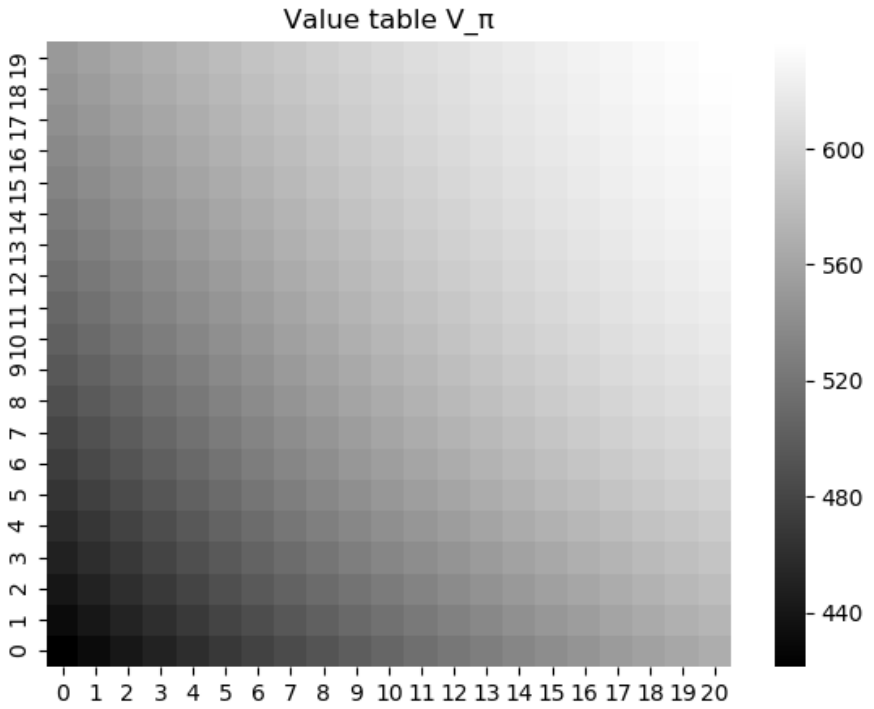


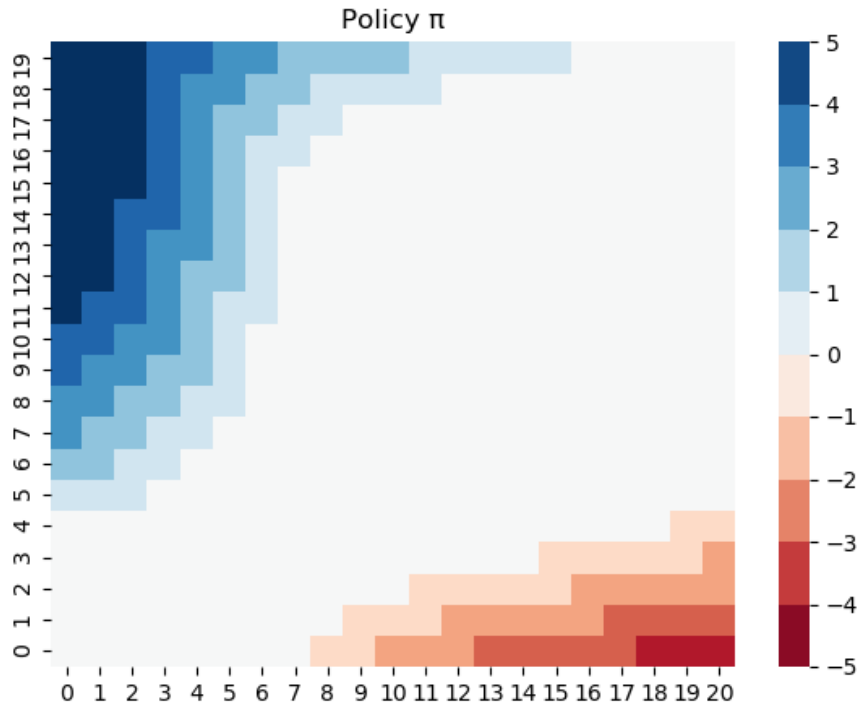
Policy after third iteration



Policy after fourth iteration

Jack's Car Rental – Value Iteration, iteration 4

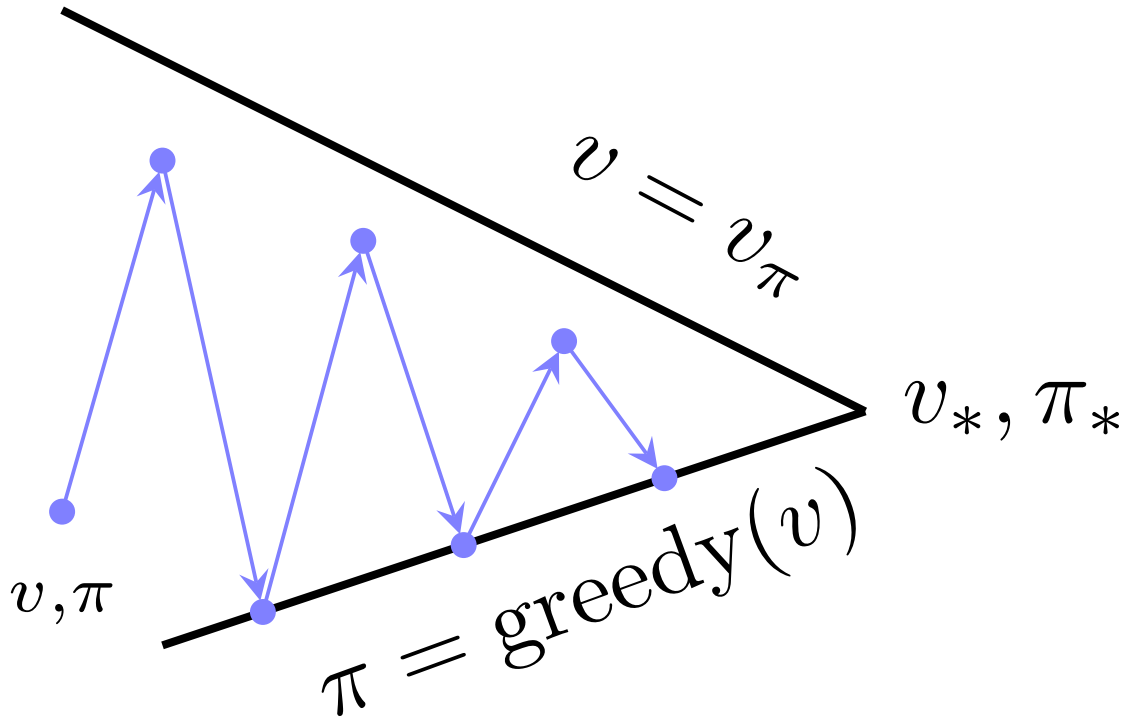




x-axis: number cars at second location, y-axis: number cars at first location

Visualization from (Pignatelli 2022)

Value Iteration



Policy evaluation = Estimate v_π

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

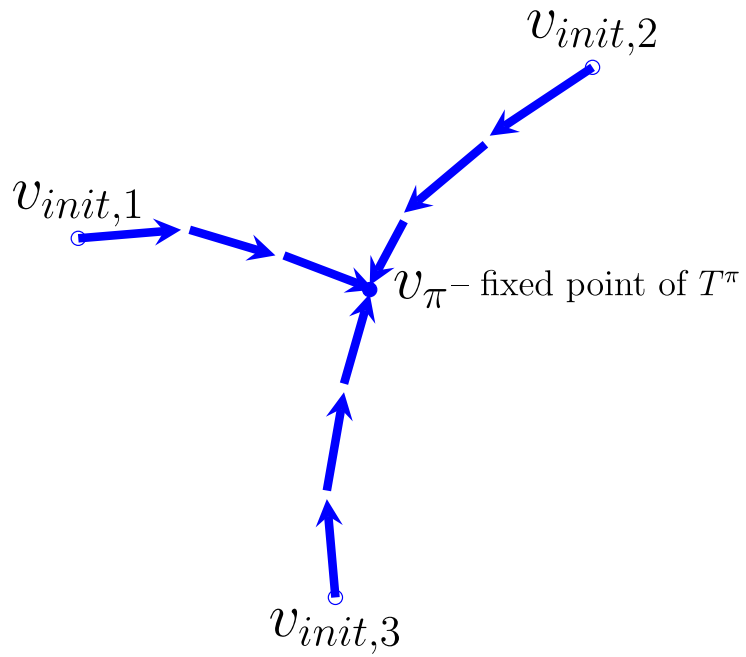
Considering Convergence of Bellman Backup

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 ($\gamma < 1$).



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) $T^\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$:

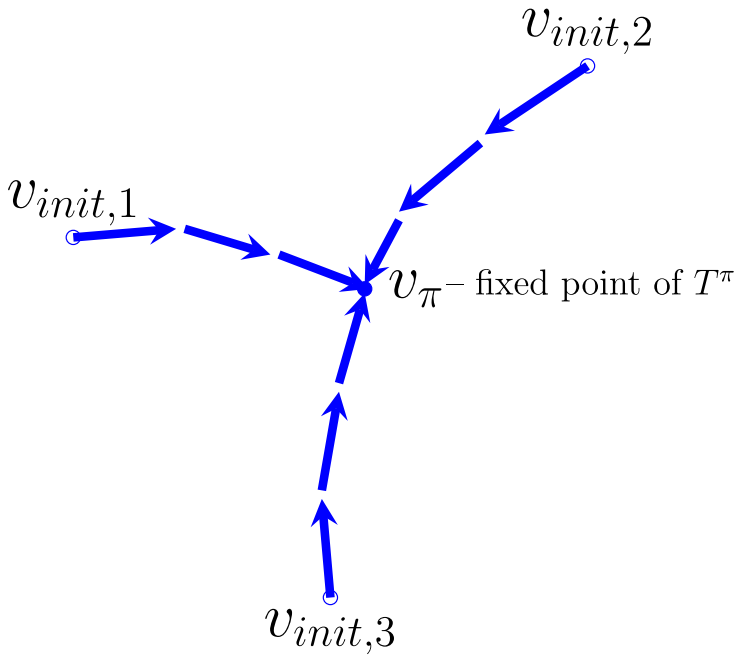
$$(T^\pi v)(s) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s')$$

(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)

Big Picture: Overview Approaches

Big picture: Solving an MDP

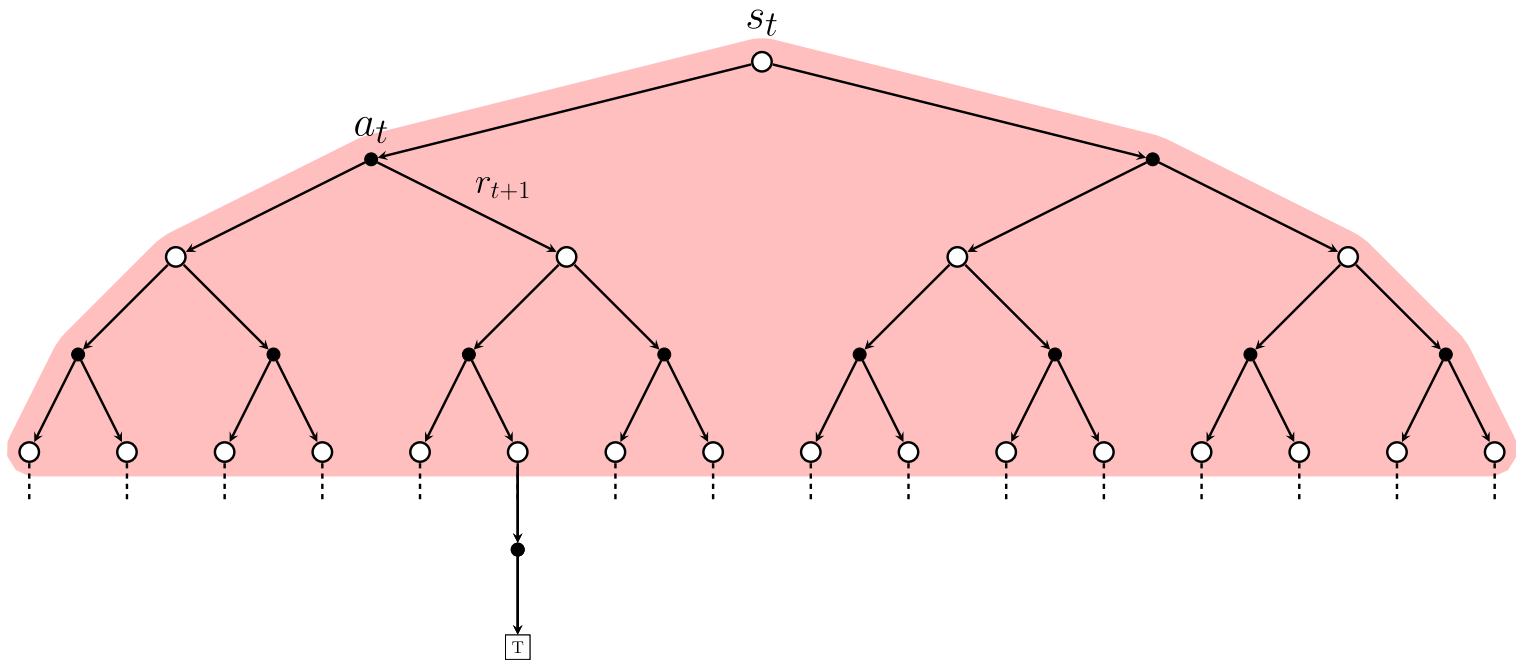
Goal: Maximize return

$$G_t = R_{t+1} + R_{t+2} + \dots$$

When MDP is fully known, joint probability $p(s', r|s, a)$:

- How environment develops and
- how reward is given.

Plus we fixed a policy how to act.



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

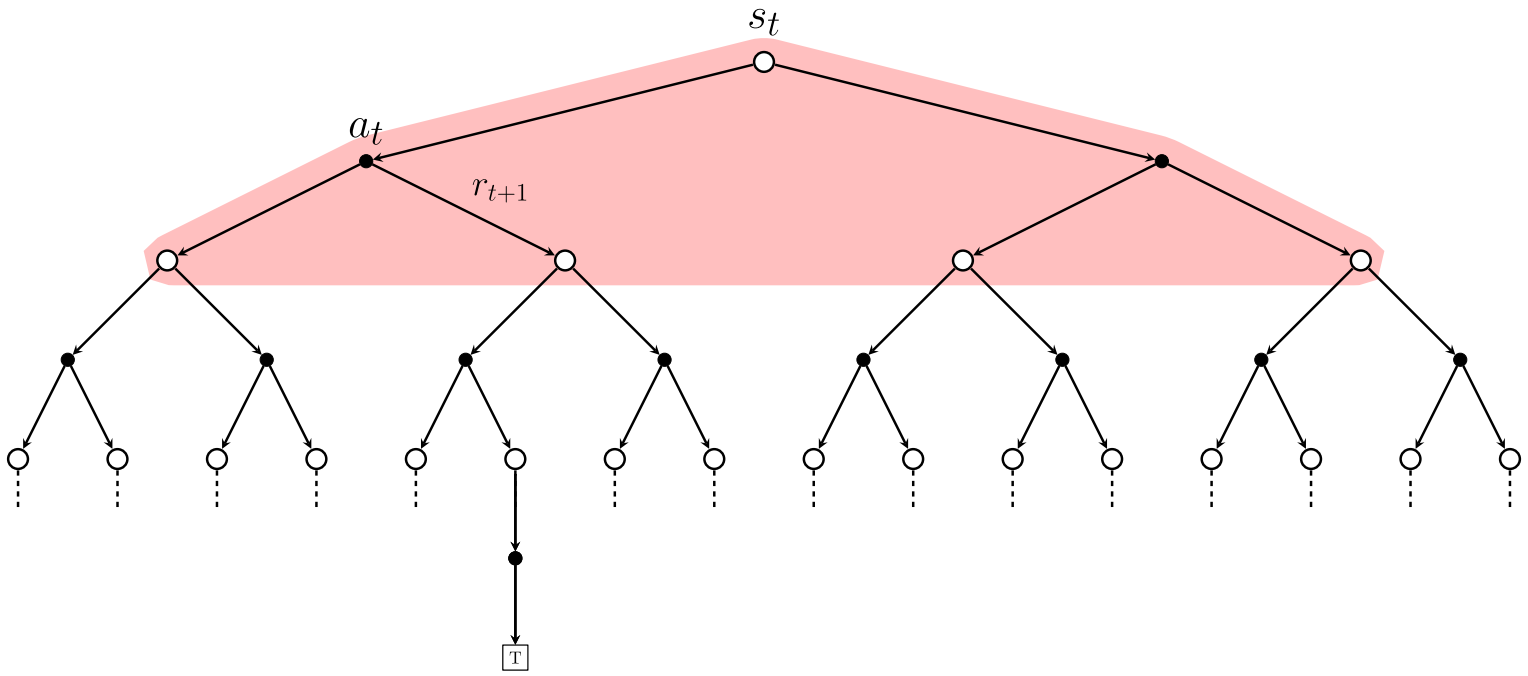
Big picture: Solving an MDP, Dynamic Programming

Goal: Maximize return

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots$$

- Use Bellman Equation for (recursive) backup.
- Model-Based Approach.
- Value function based Approach.

Called **bootstrapping**.



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

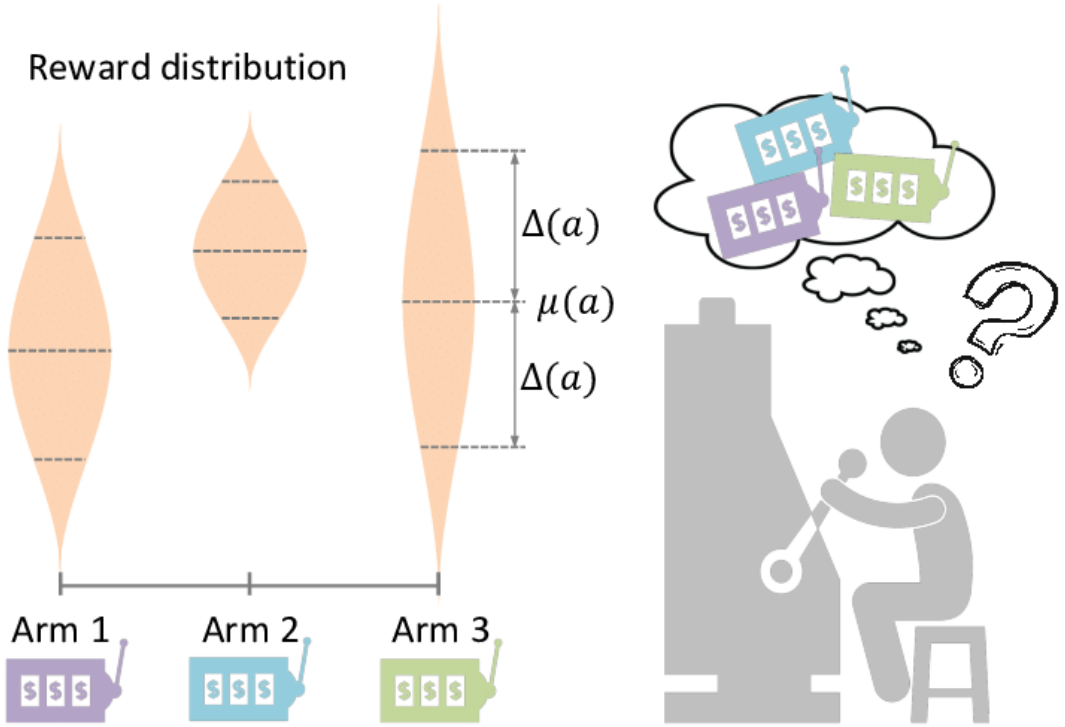
Example: Robot Playing Soccer – Hierarchical RL

See Video of robots playing soccer

Details see (Huang u. a. 2022)

Big picture: How to proceed when MDP is not known?

Recap – Multiarmed Bandit



Estimate Action Values

The action value for action a is the expected reward

$$q(a) = \mathbb{E}\Big[R_t \mid A_t = a\Big]$$

A simple estimate is the average of the sampled rewards:

$$Q_t(a) = \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

(Gao u. a. 2021)

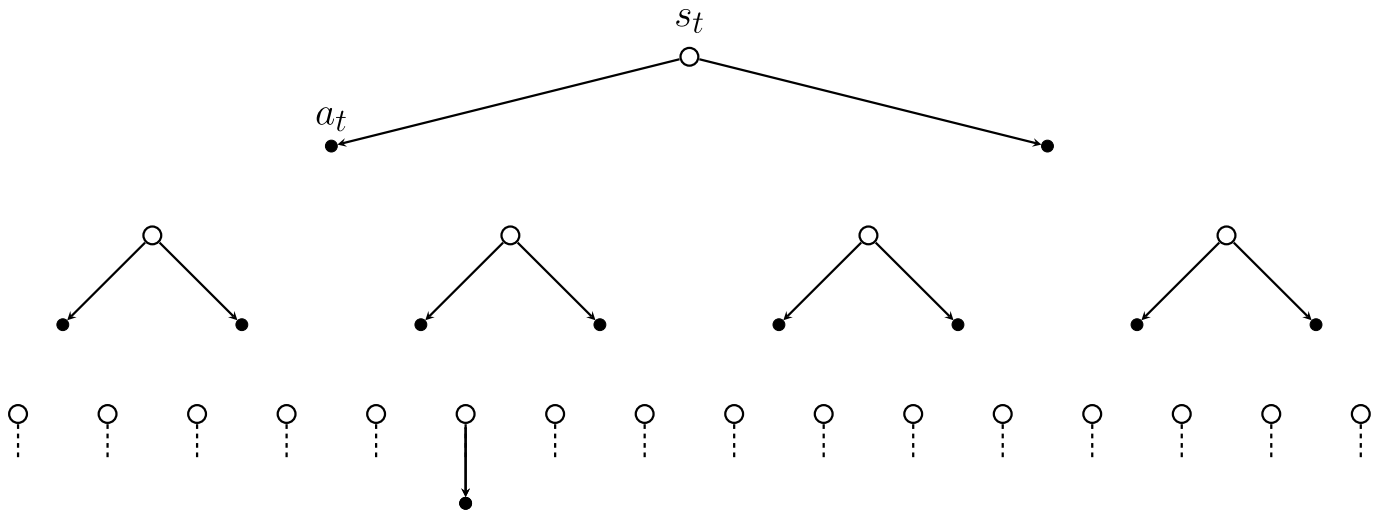
Bandits with States – Contextual bandits

How can we extend the multiarmed bandit approach to multiple states? Consider bandits with different states

- but episodes are still one step
- actions do not affect state transitions
- no long-term consequences

Then, we want to estimate:

$$q(s, a) = \mathbb{E}(R_{t+1}|S_t = s, A_t = a)$$



(Hasselt und Borsa 2021)

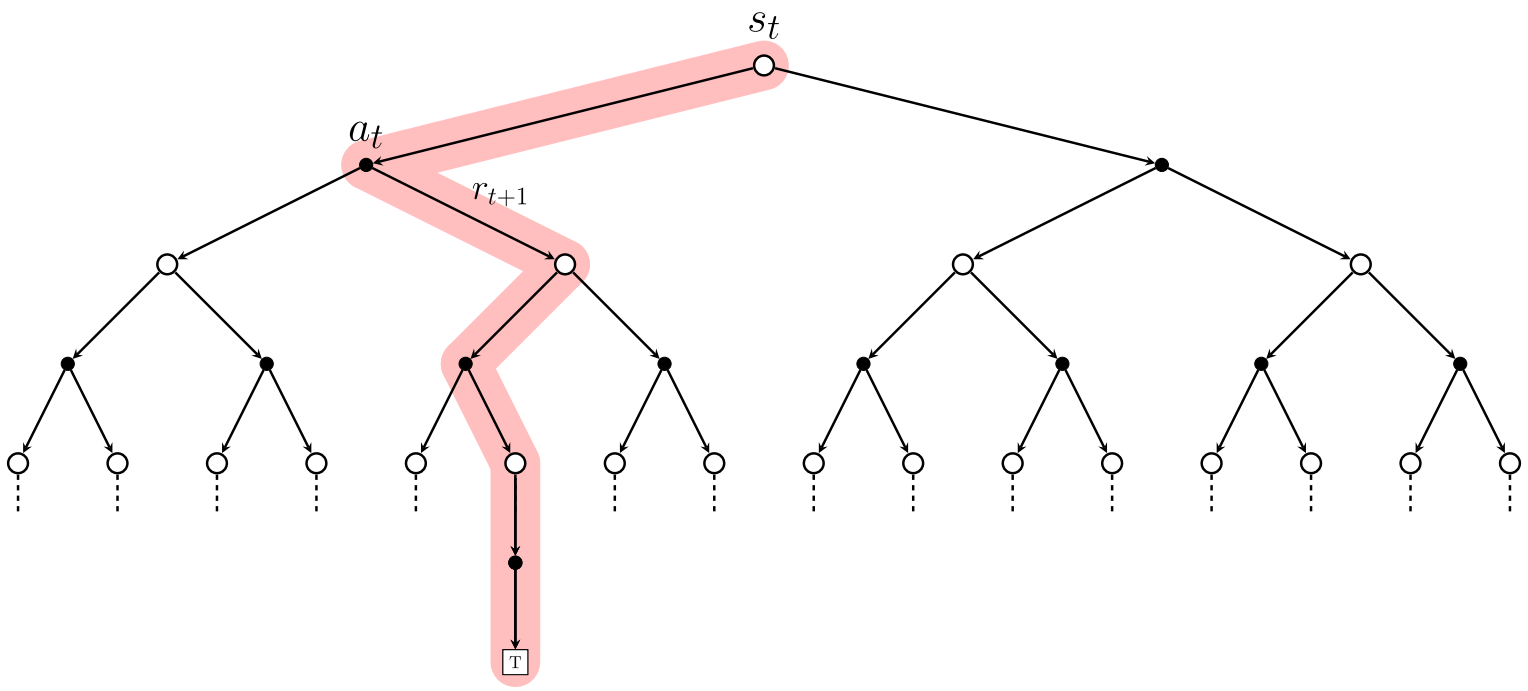
Monte Carlo Sampling

Goal: Maximize return

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots$$

- Model-free Approach.
- Sample from experience.
- Exploit sequential data.

Sample along path (Monte Carlo)



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

Recap – 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)

Monte-Carlo Sampling Methods

Using Monte-Carlo Method for Reinforcement Learning

Monte-Carlo:

- is a model-free approach that does not use knowledge of MDP (neither transition probabilities nor reward distribution),
- learns directly from full episodes of experience and the obtained return in these
- as it uses the straightforward idea to estimate the mean return for a state from these episodes.

One drawback: requires episodic MDPs as the episodes must terminate.

Monte-Carlo Policy Evaluation

Reminder:

- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$
- Value function describes expected return $v_\pi(s) = \mathbb{E}_\pi(G_t | S_t = s)$

Goal in Policy Evaluation: Learn v_π from episodes of experience when following policy π.

Monte-Carlo policy evaluation:

Simply observe the **empirical mean return** obtained from a state as an approximation for the expected return

Simple implementation: Every-Visit MC Policy Evaluation

To evaluate a state s, keep track of visits $N(s)$ and sum of returns $S(s)$:

- Every time t that state s is visited in an episode: Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $v(s) = S(s)/N(s)$.

Convergence

Problem: Individual datapoints are not independent.

Still, this converges towards the value function of the policy: $v(s) \rightarrow v_\pi(s)$ for $N(s) \rightarrow \infty$

Simple implementation: First-Visit MC Policy Evaluation

To evaluate a state s, keep track of visits $N(s)$ and sum of returns $S(s)$:

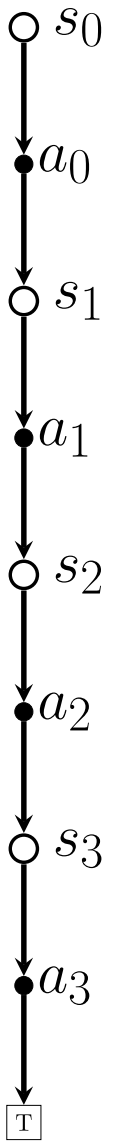
- The first time-step t that state s is visited in an episode: Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $v(s) = S(s)/N(s)$.

This converges towards the value function of the policy as each state would be visited infinite number of times: $v(s) \rightarrow v_\pi(s)$ for $N(s) \rightarrow \infty$

Monte Carlo methods imply estimates for each state are independent.

- Each return is included in the equation for writing the previous time step's return.
- We avoid duplicating computations by starting at the terminal state and working our way backwards.

$$G_0 = R_1 + \gamma G_1$$
$$G_1 = R_2 + \gamma G_2$$
$$G_2 = R_3 + \gamma G_3$$
$$G_3 = R_4 + \gamma G_4$$
$$G_4 = 0$$



First-visit MC prediction, for estimating $v \approx v_\pi$

Input: A policy π to be evaluated
`// Initialization.`
$v(s) \in \mathbb{R}$ arbitrarily, for all $s \in \mathcal{S}$
`Returns`$(s) \leftarrow$ an empty list, 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 *not in* $S_0, S_1, \ldots, S_{t-1}$ **then**
      Append G to `Returns`(S_t)
      $v(S_t) \leftarrow$ `average`(`Returns`(S_t))
    end
  end
end

Example: Blackjack

Game: Play only against a dealer.

Goal: sum of cards is as great as possible without exceeding 21.

Counting:

- Number cards equal their number,
- all face cards count as 10,
- an ace can count as either 1 or 11.

Example: Gameplay Blackjack

Initially:

- Player gets two cards.
- Dealer gets two cards, one is visible.

Player starts

- hit = take another card (and possibly another) until
- stick = don’t take another card (or goes bust).

Dealer has a fixed strategy: stick on any sum 17 or greater, and hit otherwise.

This problem can naturally be formulated as an episodic MDP, where each game is an episode.

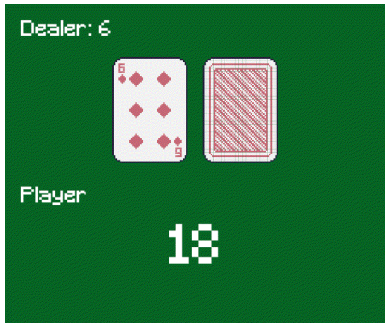
Task – Blackjack as a Reinforcement Learning problem

Our goal: Find a value function (that we later can exploit for good decision making).

How can we describe Blackjack as a MDP problem?

- What is the state space?
- What are the actions?
- What is an episode?
- What makes this difficult to approach using DP?

Blackjack



This environment is part of the Toy Text environments. Please read that page first for general information.

Action Space	Discrete(2)
Observation Space	Tuple(Discrete(32), Discrete(11), Discrete(2))
Import	`gym.make("Blackjack-v1")`

Blackjack is a card game where the goal is to beat the dealer by obtaining cards that sum to closer to 21 (without going over 21) than the dealers cards.

Description

Example: Blackjack as an MDP

State space is a 3-tuple of ($32 \times 10 \times 2$):

- the player's current sum $\in \{0, 1, \ldots, 31\}$,
- the dealer's face up card $\in \{1, \ldots, 10\}$, and
- whether or not the player has a usable ace ($\texttt{no} = 0$, $\texttt{yes} = 1$).

Actions:

- hit: take another card
- stick: don't take another card

Reward for stick:

- $+1$ if sum cards > sum dealer cards
- 0 if sum cards = sum dealer cards
- -1 if sum cards < sum dealer cards

Reward for twist:

- -1 if sum of cards > 21 (and terminate)
- 0 otherwise

Problems for Dynamic Programming in Blackjack

There is complete knowledge of the environment in the blackjack task. Therefore, we could use DP.

But this is not easy as DP relies on

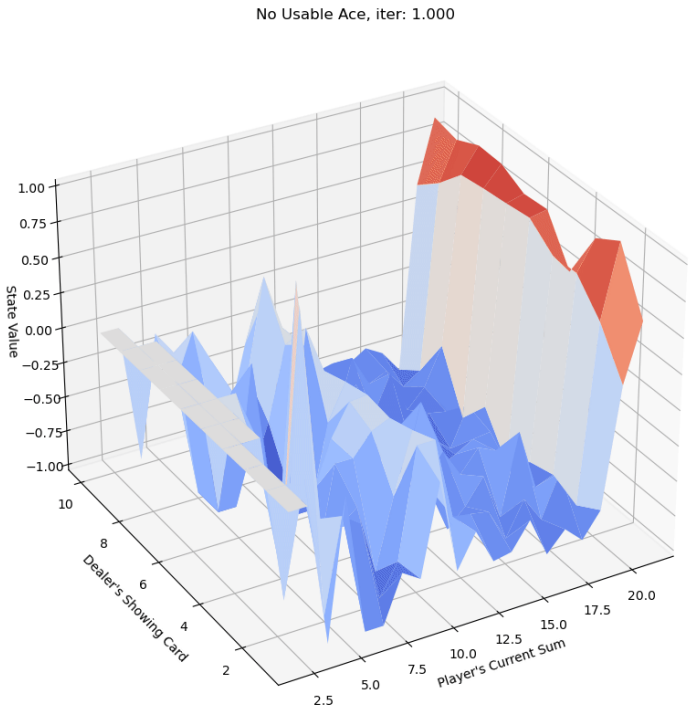
- the distribution of next events (for bootstrapping)
- in particular, the environments dynamics as given by the joint probability $p(s', r|s, a)$

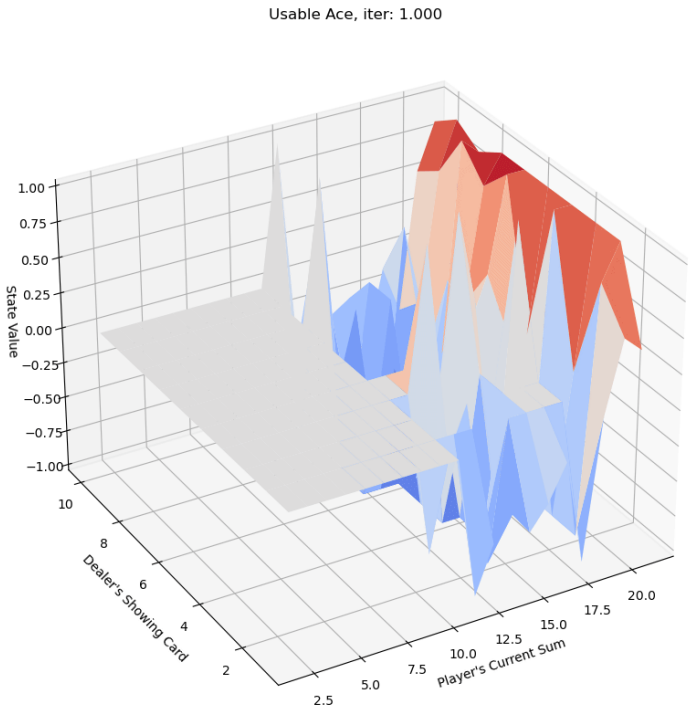
This is difficult to determine for blackjack.

As a though experiment: consider the player's sum is 14 and he chooses to stick. Computing of the win probability (based on the dealer's showing card) requires computation of all of the probabilities which is quite complex (and error-prone).

Example: Convergence of Value Function over time

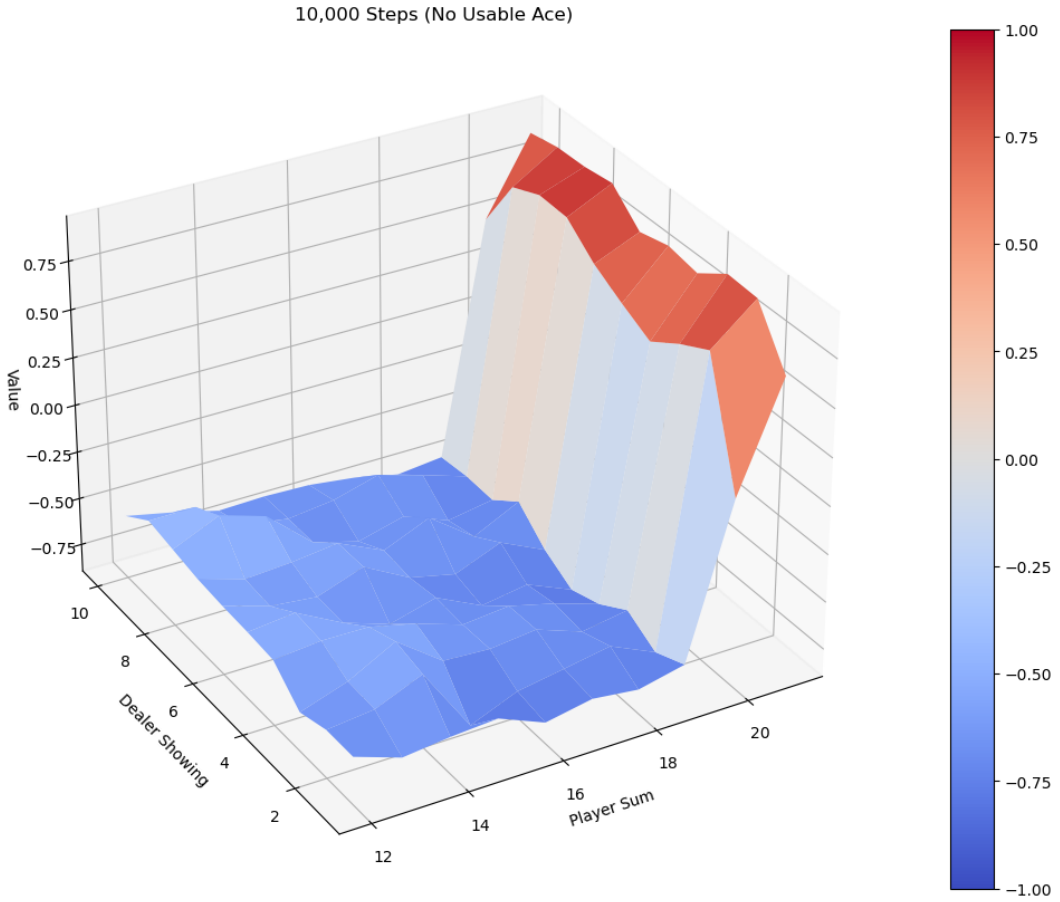
Policy: take cards when count is lower 20.

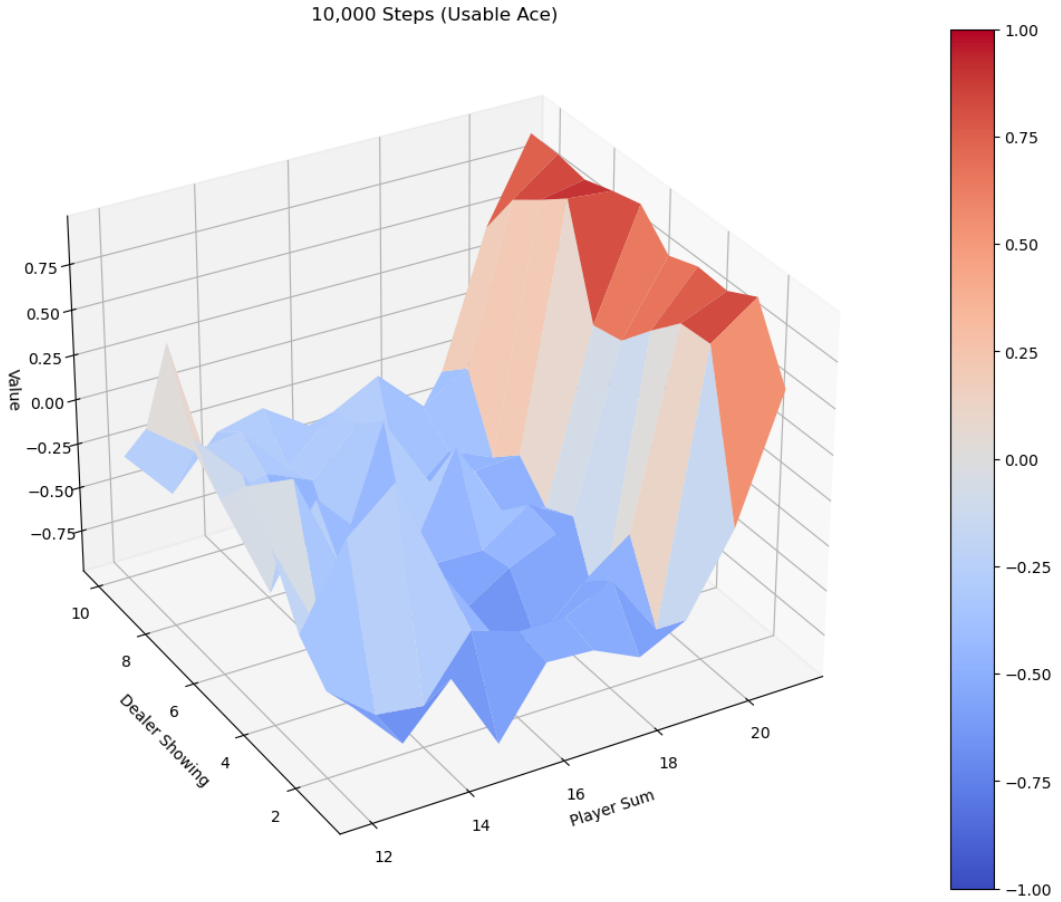




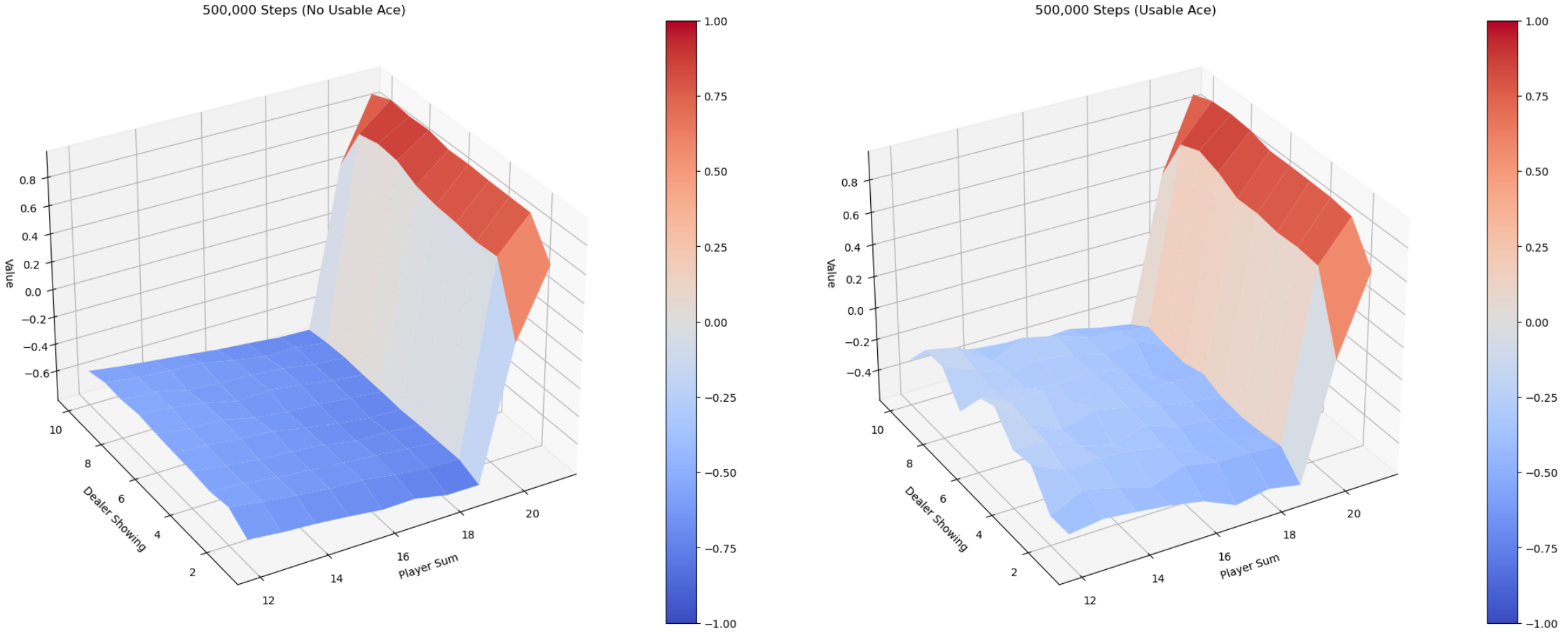
Example: Overview Results after 10k episode

Policy: take cards when count is lower 20.





Example: Overview Results after 500k episode



Incremental Update of a Mean

In general, the mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be incrementally computed:

$$
\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})
\end{aligned}
$$

(Silver 2015)

Incremental Monte-Carlo Updates

Update $v(s)$ incrementally after epise $S_1, A_1, R_2, \ldots, S_T$

For each state S_t with return G_t

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

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

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes:

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

(Silver 2015)

Monte-Carlo Learning Characteristics - Advantages

Monte-Carlo algorithms can learn value predictions from experience without knowing the underlying MDP.

Compared to Dynamic Programming MC has as advantages

- Learn directly from interaction with the environment;
- Can be used with simulation or sample models (in many cases: while sampling episodes is easy, it is often difficult to construct the full model of transition probabilities required for DP);
- Efficient (and easy to apply) on small subset of the states (estimates are independent);
- (may be less harmed by violations of Markov property because they do not bootstrap)

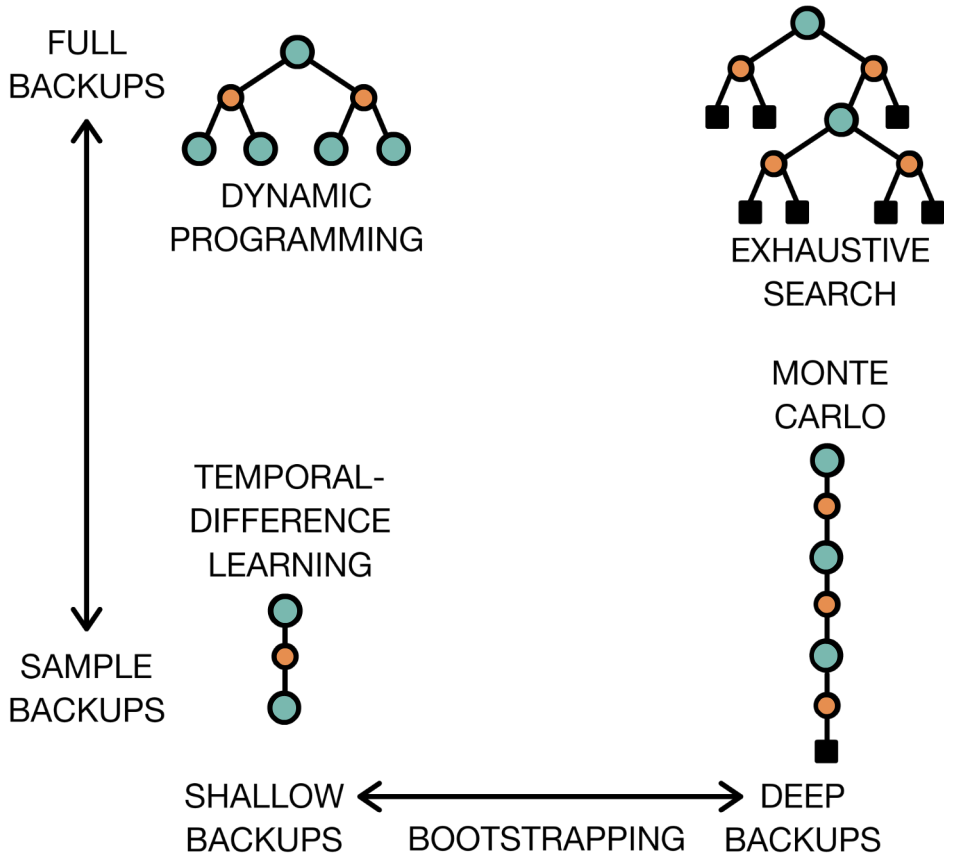
Monte-Carlo Learning Characteristics - Disdvantages

But when episodes are long, learning can be slow, as

- we have to wait until an episode ends before we can learn (we are in an episodic setting and need termination),
- returns can have high variance.

Temporal Difference Learning

Reinforcement Learning Algorithms Overview



(Arulkumaran u. a. 2017)

Temporal Difference Learning

Temporal Difference …

- Learning is model-free: no knowledge on MDP transition or reward probabilities is needed
- methods also learn directly from episodes of experience

Bootstrapping

TD learning methods update targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns as in MC methods.

The key idea in TD learning is to update the value function $v(S_t)$ towards an estimated return $R_{t+1} + \gamma(S_{t+1})$ (known as "TD target").

(Weng 2018)

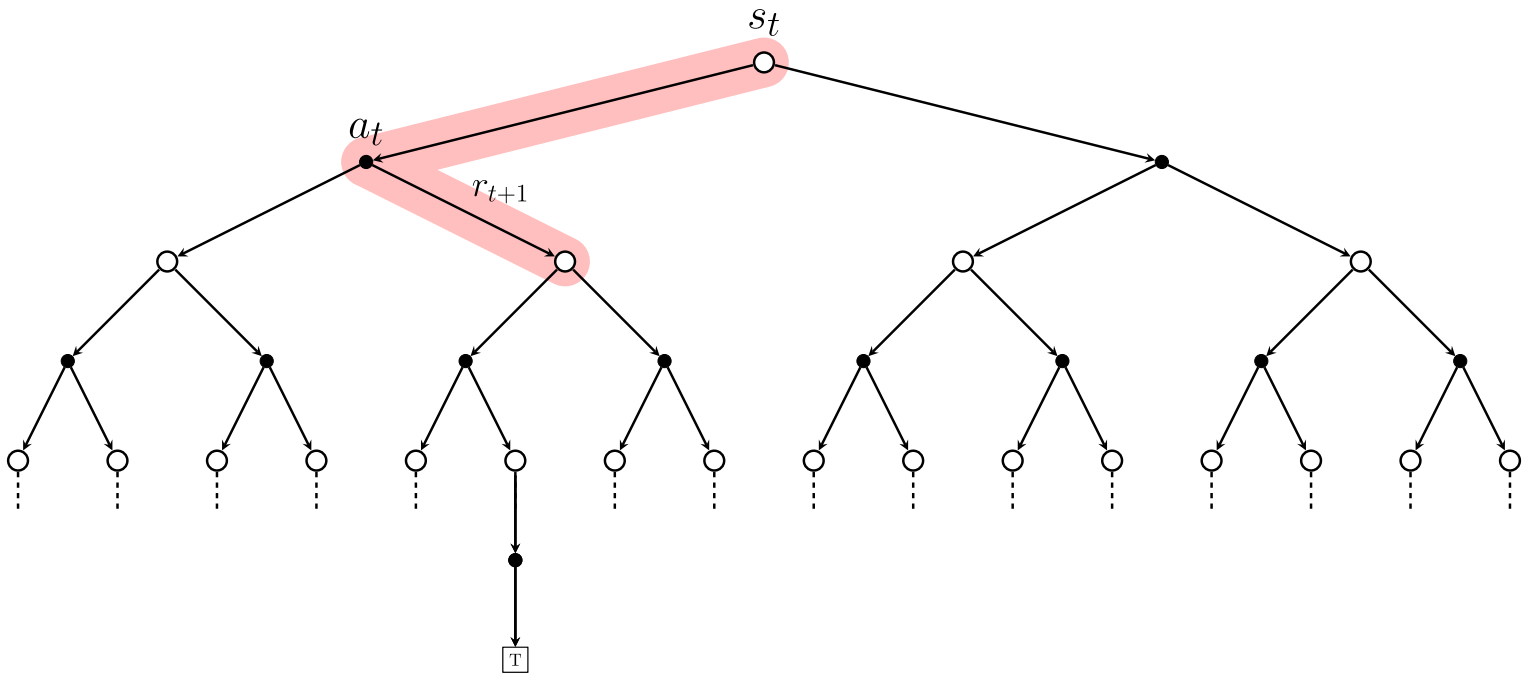
Temporal Difference Learning – Backup Diagram

Goal: Maximize return

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots$$

- Model-free Approach.
- Sample from experience (Monte Carlo.
- Exploit sequential data.

But: Bootstrap as in DP (use Bellman equation).



$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_t + 1) - v(S_t))$$

From MC to Temporal Difference Learning

In both: Goal is to learn v_π online from experience under policy π

Incremental every-visit Monte-Carlo:

- Update value $v(S_t)$ towards **actual** return G_t:

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

Simple temporal-difference learning algorithm TD(0):

- Update value $v(S_t)$ towards **estimated** return $R_{t+1} + \gamma v(S_{t+1})$:

$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_t + 1) - v(S_t))$$

(Silver 2015)

Temporal Difference Learning

$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_t + 1) - v(S_t))$$

TD Target

$R_{t+1} + \gamma v(S_t + 1)$ is called the TD target.

TD Error

$\delta_t = R_{t+1} + \gamma v(S_t + 1) - v(S_t)$ is called the TD error.

TD: Value Estimation

Update of the value function is regulated by the learning rate α.

In brief: TD means update a guess (of the value function) towards a guess (experiencing a single step and a guess of what follows):

$$
\begin{aligned}
v(S_t) &\leftarrow (1-\alpha)v(S_t) + \alpha G_t \\
v(S_t) &\leftarrow v(S_t) + \alpha(G_t - v(S_t)) \\
v(S_t) &\leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))
\end{aligned}
$$

Similarly for the Q-function:

$$
q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha(R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t))
$$

(Weng 2018)

Tabular TD(0) for estimating v_π

Input: The policy π to be evaluated; step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\texttt{terminal}) = 0$
for *each episode* **do**
  Initialize S
  for *each step in the episode, until state S is terminal* **do**
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha \left[R + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  end
end

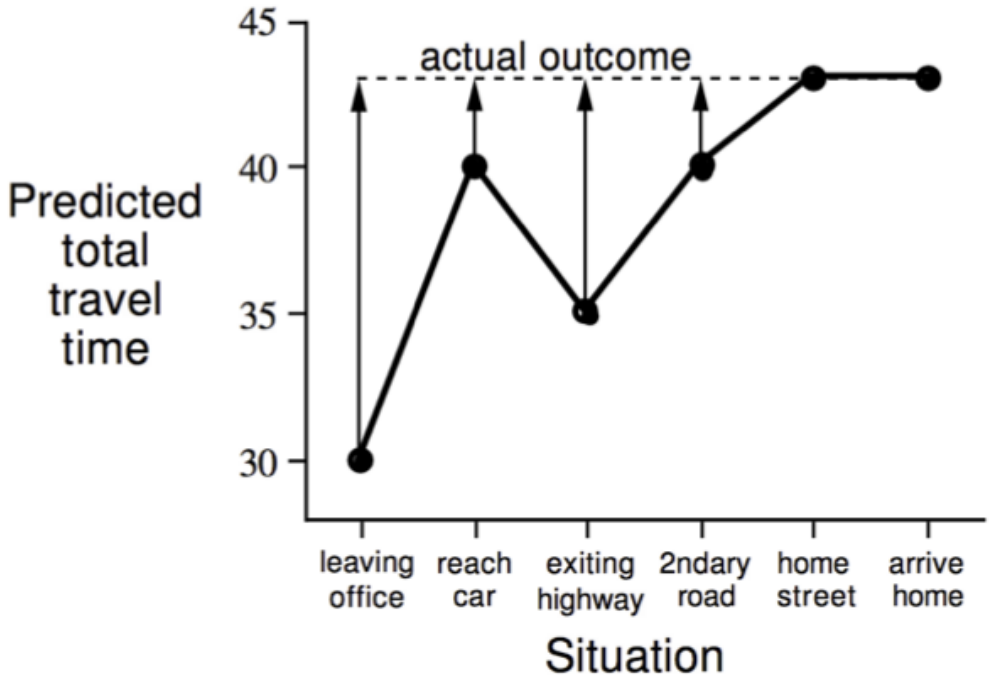
Example: Driving Home

State	Elapsed time	Predicted time to Go	Predicted total time
Leaving office, Friday at 6	0	30	30
Reach car, raining	5	35	40
Exiting highway	20	15	35
small road, behind truck	30	10	40
Entering home street	40	3	43
Arrive home	43	0	43

Example: Comparison Monte-Carlo and TD Approach

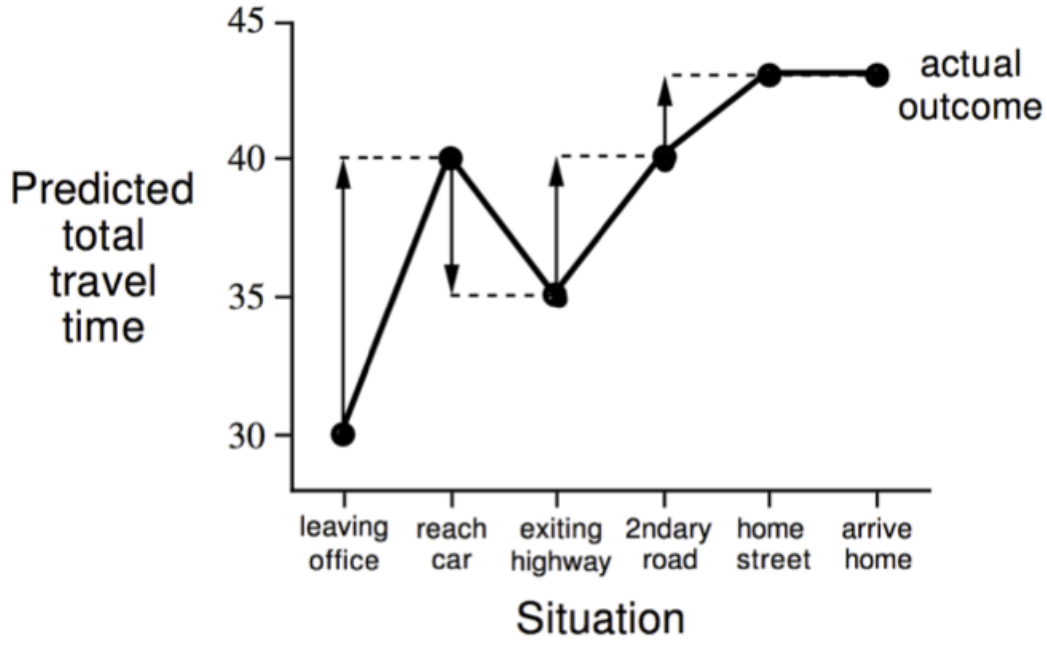
Monte-Carlo Approach

Changes recommended, MC ($\alpha = 1$):



Temporal Difference Method

Changes recommended in TD ($\alpha = 1$):

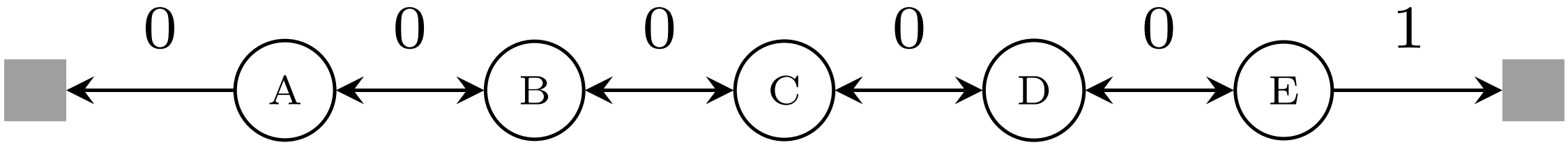


Comparison TD and MC

- TD can learn before knowing the final outcome
  - TD can learn online after every step MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

(Silver 2015)

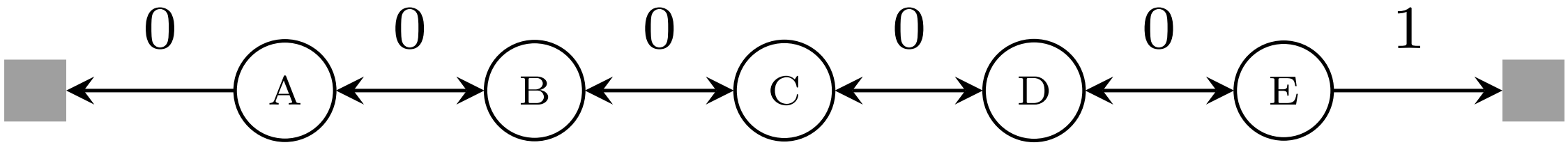
Example: Random Walk



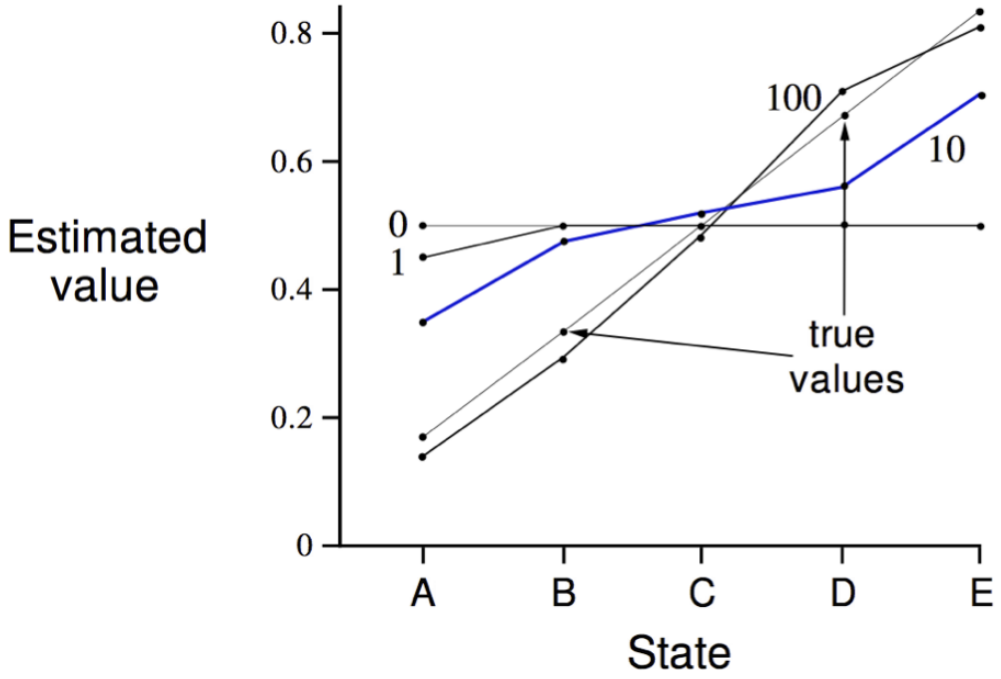
- all episodes start in the center state C,
- then proceed randomly left or right.
- Termination states are on the extreme left and extreme right.
- Reward: only given when terminating right ($+1$).

Value of a state is the probability of terminating in the right node.

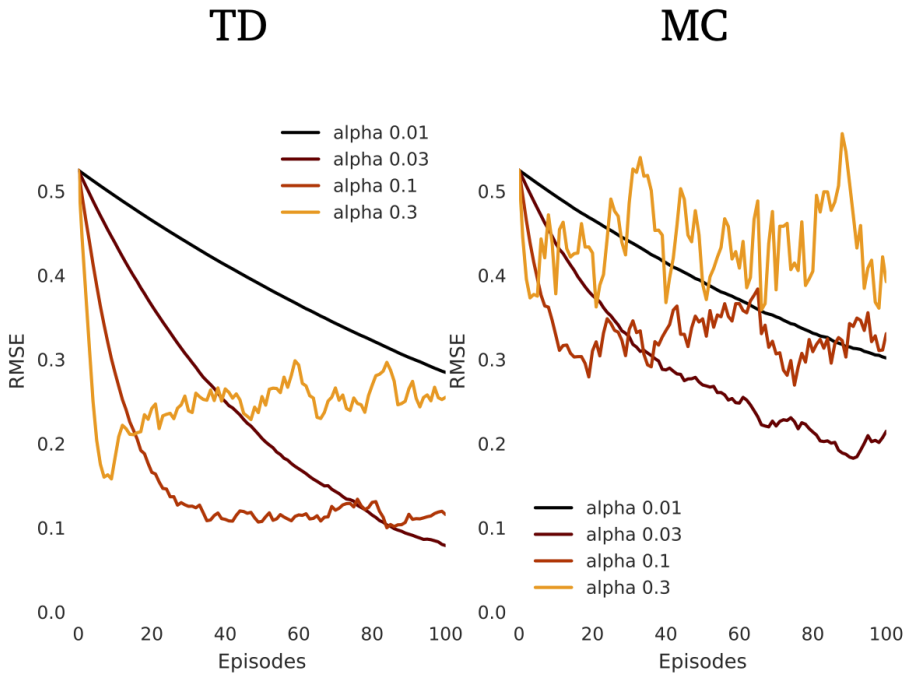
Therefore: $v_\pi(C) = \frac{1}{2}$, and for states A through E these are $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$.



TD(0) Estimates for v_π



Learning Curves



(Sutton und Barto 2018; Hasselt und Borsa 2021)

References

Arulkumaran, Kai, Marc P. Deisenroth, Miles Brundage, und Anil A. Bharath. 2017. „Deep Reinforcement Learning: A Brief Survey“. *IEEE Signal Processing Magazine* 34 (6).

Gao, Chongming, Wenqiang Lei, Xiangnan He, Maarten Rijke, und Tat-Seng Chua. 2021. „Advances and Challenges in Conversational Recommender Systems: A Survey“.

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.

Pignatelli, Eduardo. 2022. „Python Repository for visualization for Sutton and Barto's ‚Reinforcement Learning' book“. https://github.com/epignatelli/reinforcement-learning-an-introduction.

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.