Deep Reinforcement Learning

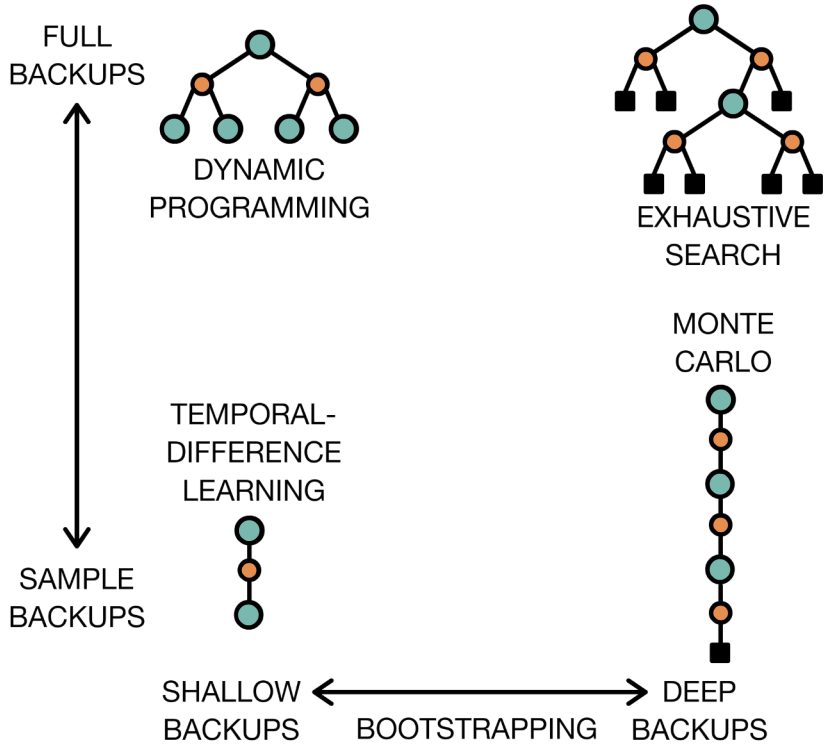
7 - Model Free Control

Prof. Dr. Malte Schilling

Autonomous Intelligent Systems Group

Recap Overview – Bootstrapping and Sampling

- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples



(Arulkumaran u. a. 2017)

Overview Lecture

- Prediction: Estimate the value function of an unknown MDP
  - Monte-Carlo Method
  - Temporal Difference Learning
- Model-free control (today):
  - **Optimise the policy**: General Policy Improvement
    - On-Policy Approach
    - Off-Policy Approach

Recap - 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:

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

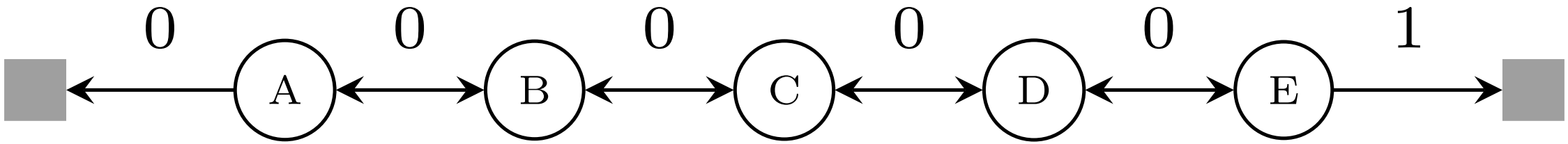
Simple temporal-difference learning algorithm TD(0):

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

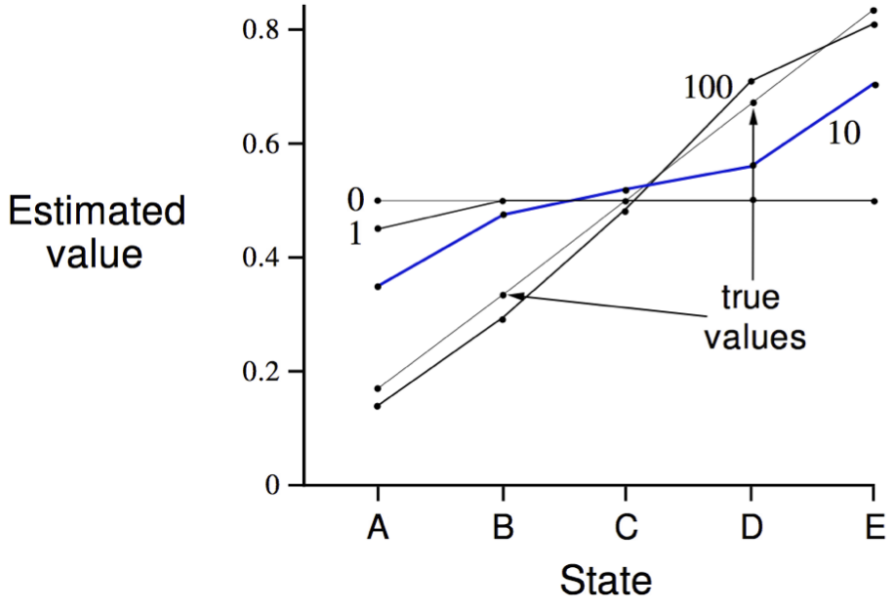
$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_t + 1) - v(S_t))$$

(Silver 2015)

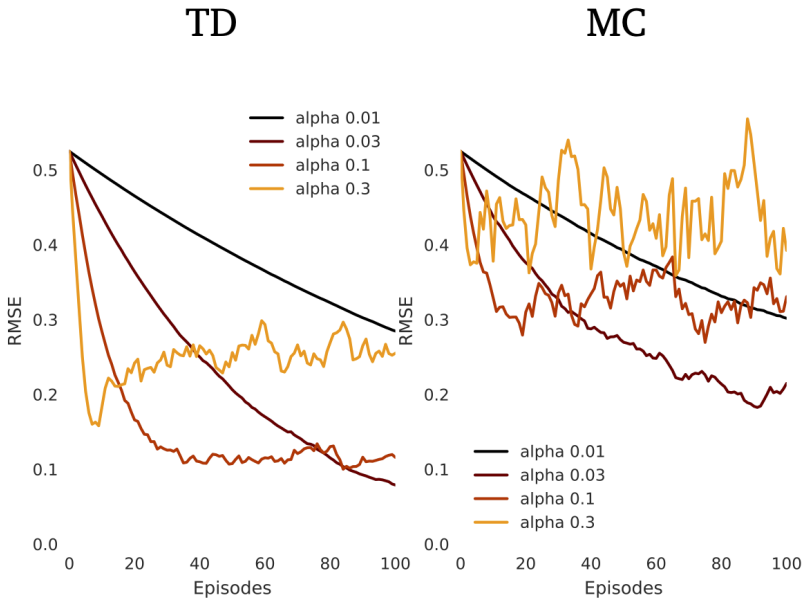
Last week – Example: Compare MC and TD empirically



TD(0) Estimates for v_π



Learning Curves



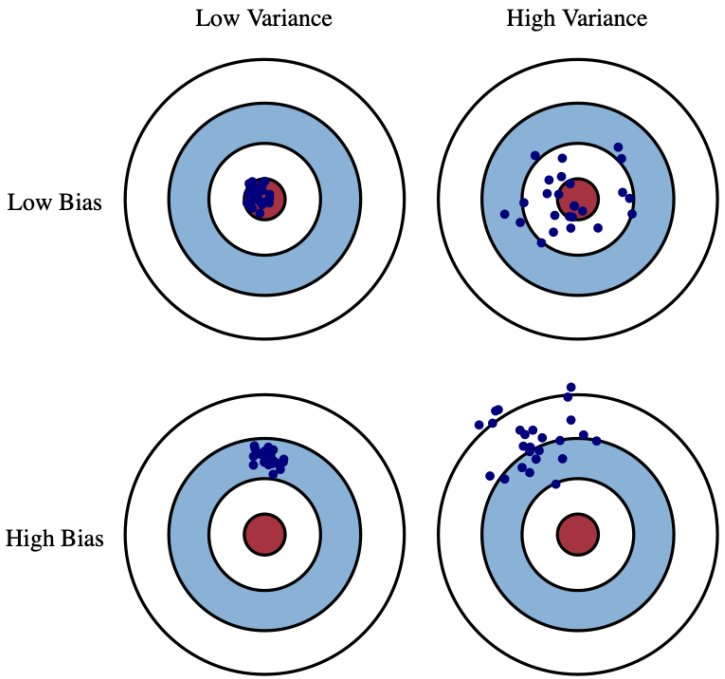
(Sutton und Barto 2018; Hasselt und Borsa 2021)

Recap – Bias-Variance Trade-Off

MC has high variance, zero bias

- Good convergence properties
- Even with function approximation
- Not very sensitive to initial value
- Very simple to understand and use

Bias and Variance



TD has low variance, some bias

- Usually more efficient than MC
- TD(0) converges to $v_\pi(s)$
- More sensitive to initial value

(Satakhutdinov 2018), (Fortmann-Roe 2012)

Recap – Temporal difference learning – q-function

- We can apply the same idea to action values – when dynamics are unknown, this is much more important
- Temporal-difference learning for action values:
  - Update value $q_t(S_t, A_t)$ towards estimated return $R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha \Big(\underbrace{R_{t+1} + \gamma q(S_{t+1}, A_{t+1} - q_t(S_t, A_t)}_{\text{TD Error}} \Big)$$

This algorithm is known as SARSA, because it uses $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

(Hasselt und Borsa 2021)

Advantages and Disadvantages of MC vs. TD

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

Advantages and Disadvantages of MC vs. TD (2)

- TD is independent of the temporal span of the prediction
  - TD can learn from single transitions
  - MC must store all predictions (or states) to update at the end of an episode
- TD needs reasonable value estimates
- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property
  - Usually more effective in non-Markov environments
- With finite data (or with function approximation) the solutions may differ

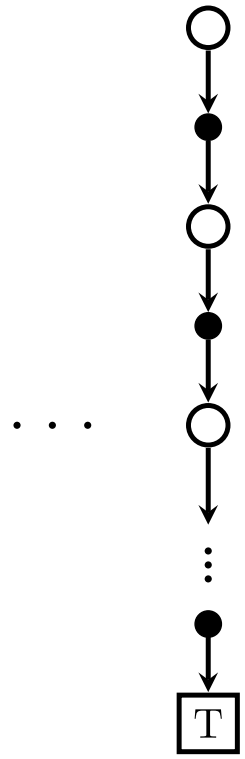
(Silver 2015)

Combining the two approaches – a unifying perspective

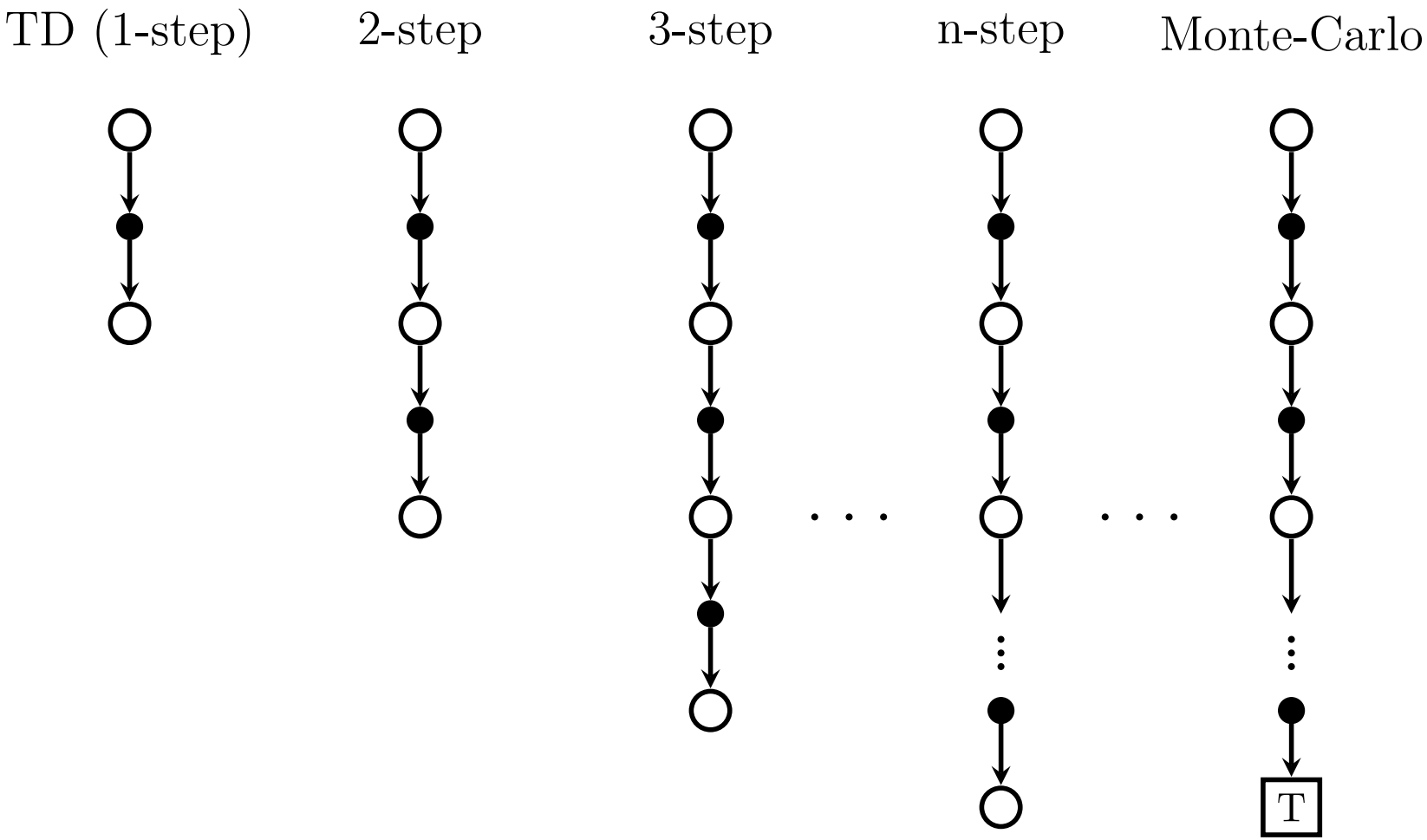
TD (1-step)

- TD uses value estimates which might be inaccurate
- In addition, information can propagate back quite slowly (possible bias)
- In MC information propagates faster, but the updates are noisier (high variance)
- We can go in between TD and MC

Monte-Carlo



Multi-step Predictions



Make n steps and then use TD target for prediction.

n-step Return

Consider the following n-step returns for $n = 1, 2, \infty$:

steps	Approach	Return
$n = 1$	TD	$G_t^{(1)} = R_{t+1} + \gamma v(S_{t+1})$
$n = 2$		$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2})$
$\vdots$		$\vdots$
$n = \infty$	MC	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$

Define the n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

n-step Return

n-step return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

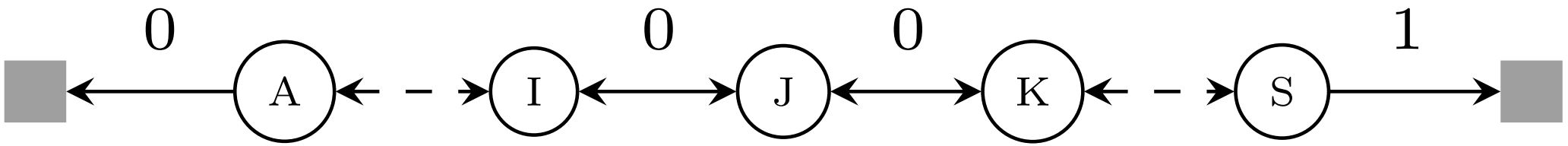
n-step Temporal Difference learning

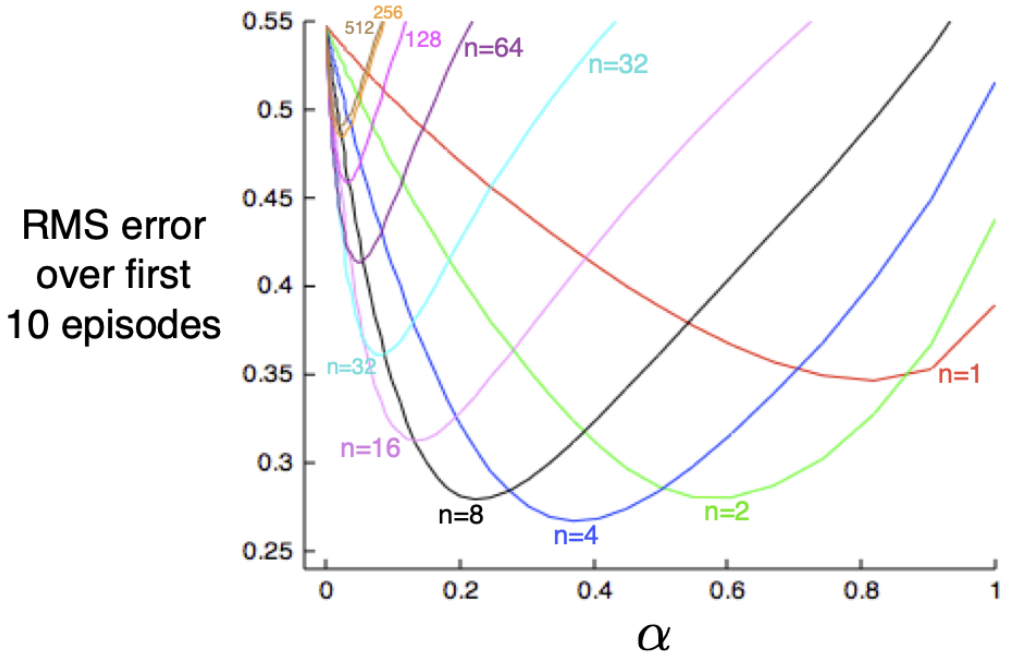
$$v(S_t) \leftarrow v(S_t) + \alpha\big(G_t^{(n)} - v(S_t)\big)$$

(Silver 2015)

(Large) Random Walk Example – Error for different n-steps

MDP: Chain of 19 Nodes; Start in J; random policy; $r = 0$ unless terminating right: $r = 1$





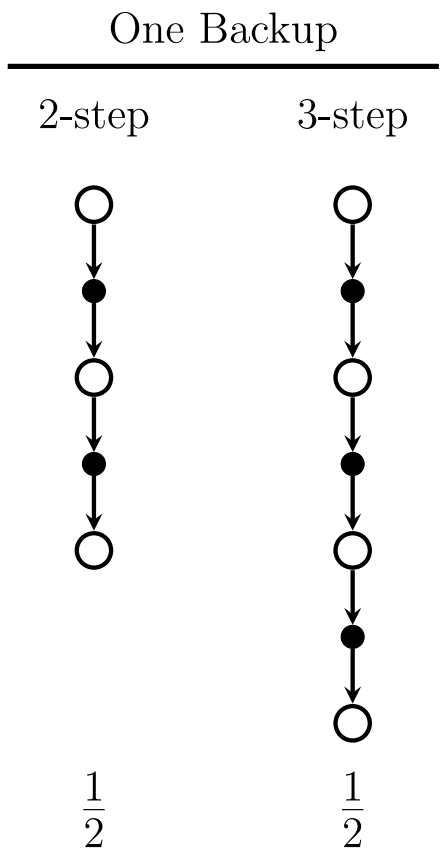
(Hasselt und Borsa 2021) following (Sutton und Barto 2018)

Averaging n-step Return

- We can average n-step returns over different n.
- For example: average the 2-step and 3-step returns as

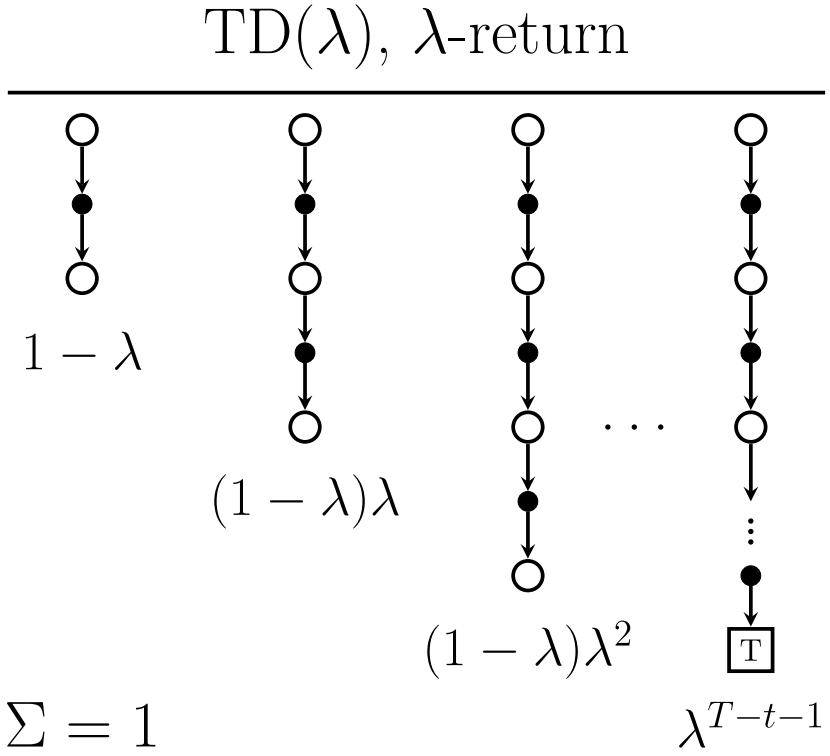
$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(3)}$$

- Combines information from two different time-steps



(Silver 2015)

λ-Return



The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$ using as a weight $(1-\lambda)\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

Forward-view TD(λ)

$$v(S_t) \leftarrow v(S_t) + \alpha\Big(G_t^\lambda - v(S_t)\Big)$$

(Silver 2015)

Computation of TD(λ)

Forward-view TD(λ)

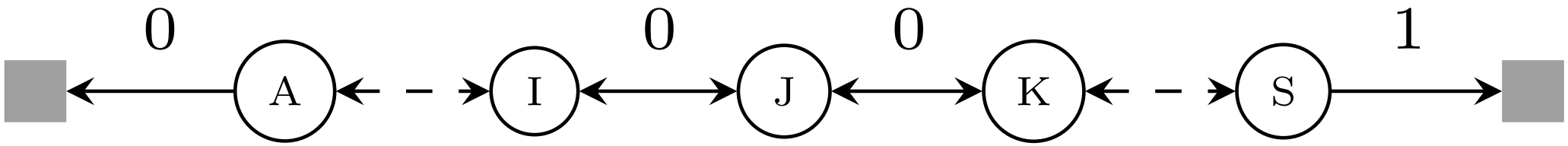
- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^λ
- Drawback: Like Monte-Carlo, can only be computed from complete episodes

Backward View TD(λ)

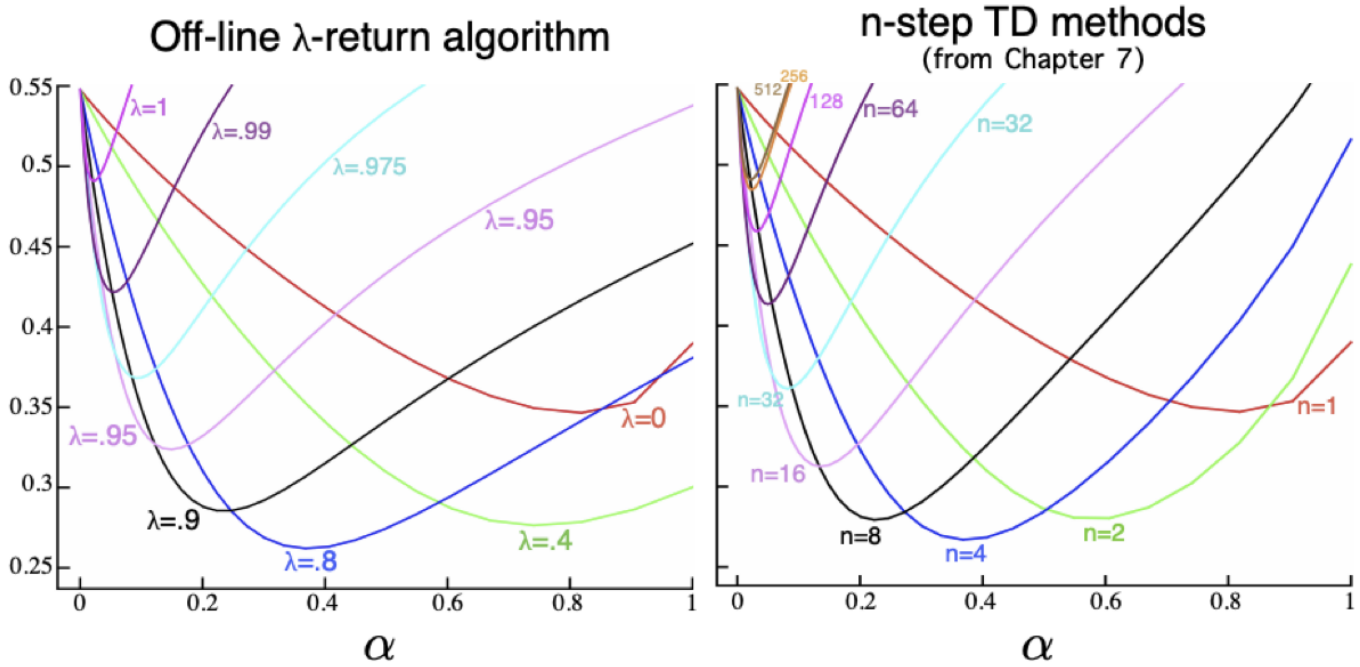
- While Forward view provides theory
- Backward view provides mechanism
- Update online, every step, from incomplete sequences – and update for previous steps going backwards into the past

(Silver 2015)

(Large) Random Walk Example – Error for TD(λ)



RMS error at the end of the episode over the first 10 episodes



Intuition: $\frac{1}{1-\lambda}$ is the `horizon', e.g., $\lambda = 0.9 \rightsquigarrow n = 10$ steps.

(Hasselt und Borsa 2021) following (Sutton und Barto 2018)

Benefits of multi-step returns

Multi-step returns have benefits from both TD and MC

- Bootstrapping can have issues with bias
- Monte-Carlo can have issues with variance
- Typically, intermediate values of n or λ are good (e.g., $n = 10, \lambda = 0.9$)

(Hasselt und Borsa 2021)

Model-free Control – Monte-Carlo

Example: Learning Walking on a Robot

See Video of robot learning walking from scratch (exploration directly on robot)

Details see (Haarnoja u. a. 2018)

Uses of Model-Free Control

Example problems that can be modelled as MDPs

- Elevator control
- Parallel Parking
- Helicopter
- Robocup Soccer
- Portfolio management
- Robot walking

For most of these problems, either:

- MDP model is unknown, but experience can be sampled
- MDP model is known, but is too big to use, except by samples

Model-free control can solve these problems

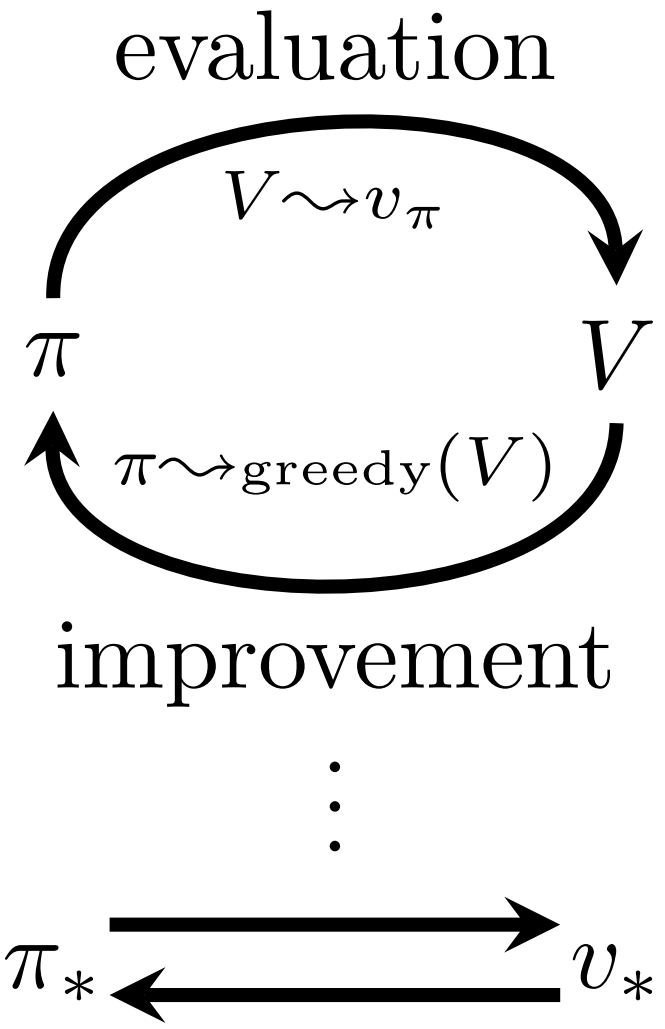
(Silver 2015)

- policy improvement: derive new policy π', $v_{\pi'}(s) \geq v_{\pi}(s) \forall s$

in interaction.

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.



Summary: Model-Free Policy Evaluation Approaches

Iterative approximation of value function for given policy π:

$$v_{n+1}(S_t) = v_n(S_t) + \alpha\Big(G_t - v_n(S_t)\Big)$$

Different Methods:

Approach	Target computation
Monte-Carlo	$G_t^{\text{MC}} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$
TD(0)	$G_t^{(1)} = R_{t+1} + \gamma v_t(S_{t+1})$
n-step TD	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n v_t(S_{t+n})$
TD(λ)	$G_t^{(\lambda)} = R_{t+1} + \lambda\Big((1-\gamma) v_t(S_{t+1}) + \lambda G_{t+1}^{\lambda}\Big)$

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

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.

Model-Free Policy Iteration Using Action-Value Function

Using Value Function

Using Value Function for improvement:
We still need a model (which state do we arrive when using an action) for greedy policy improvement:

$$\pi'(s) = \arg\max_a \mathbb{E}(R_{t+1} + \gamma v(S_{t+1}) | S_t = s, A_t = a)$$

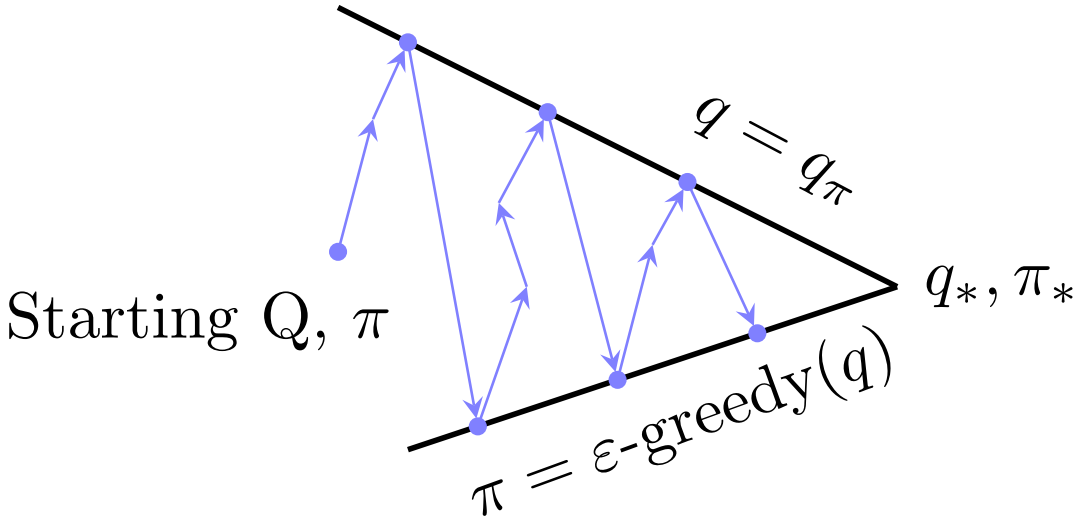
Using Action-Value Function

In contrast: Greedy policy improvement over $q(s, a)$ is model-free which makes it directly applicable

$$\pi'(s) = \arg\max_a q(s, a)$$

Monte-Carlo Policy Improvement

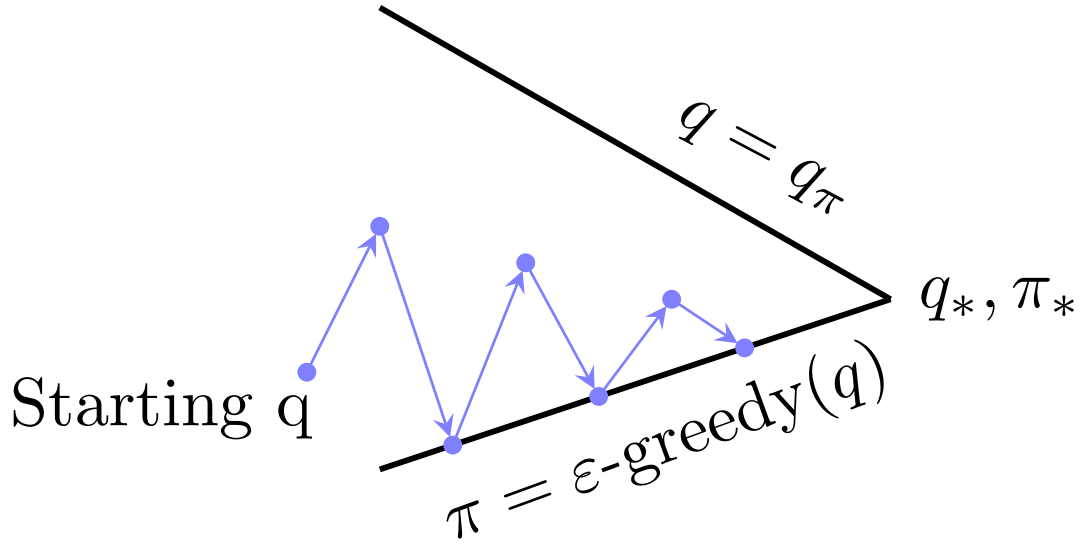
Monte-Carlo Policy Iteration



Policy evaluation: MC policy evaluation, $q = q_\pi$

Policy improvement: ε-greedy policy improvement

Monte-Carlo Control



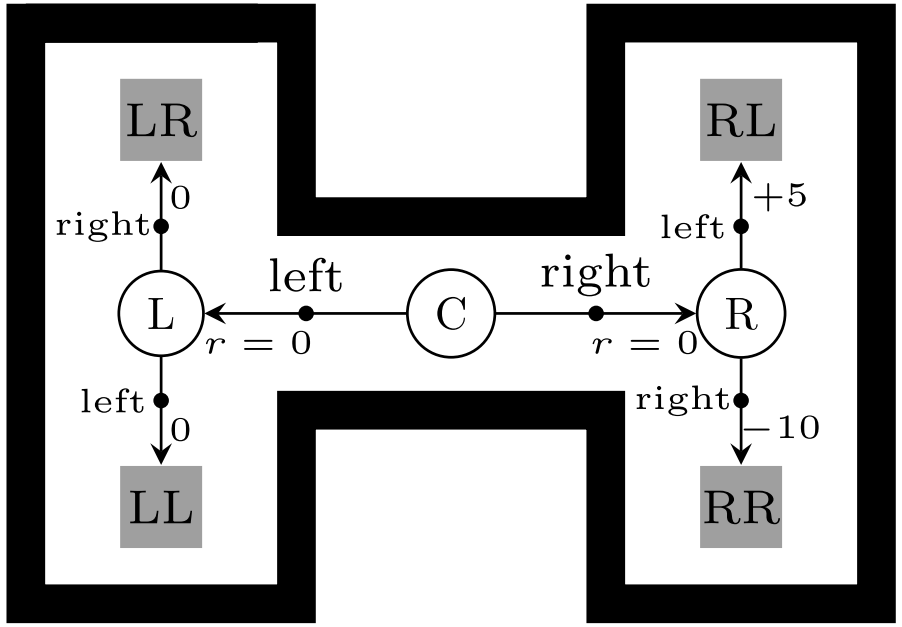
Every episode:

MC policy evaluation, $q \approx q_\pi$

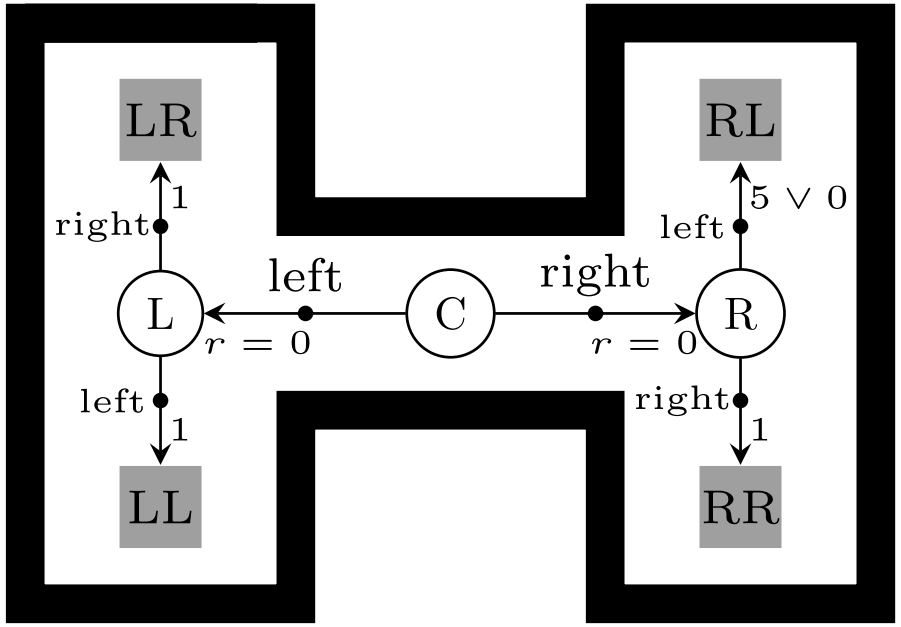
ε-greedy policy improvement

Explore a maze – but structure and rewards are unknown (visualized here as a MDP). Use MC to find an action-value function for a random policy, and improve the policy. What are your observations and takeaways for GPI?

Deterministic Rewards

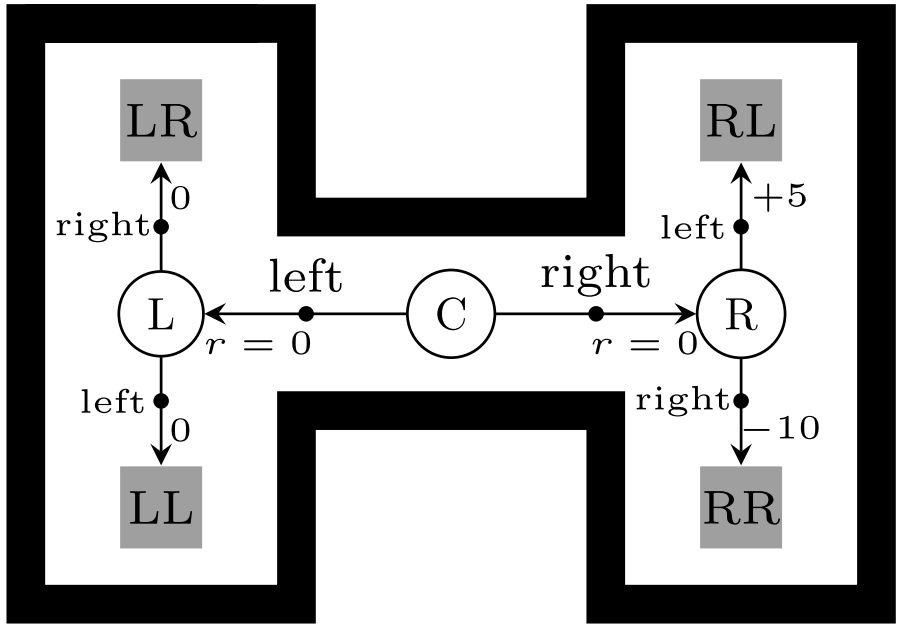


Probabilistic rewards



Left Task – Value Function Considerations

Deterministic Rewards



Action-Value Function

s	a	$q(s,a)$
C	left	0
C	right	-5

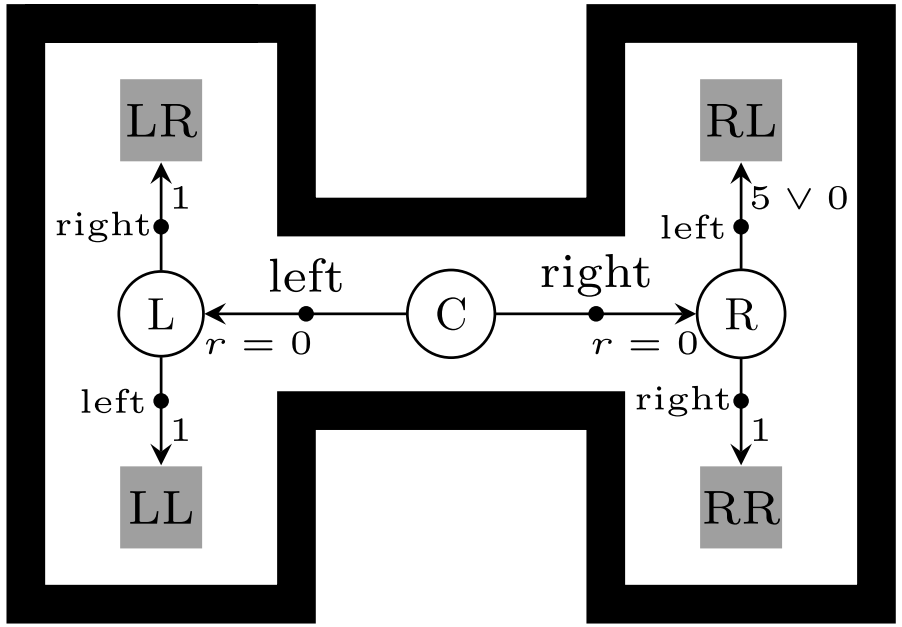
with discount factor $\gamma = 1$

Observation: After first iteration, action-value-function for moving right – $q(C, right)$ – is negative and an improved policy will pick the (overall) suboptimal policy.

Important: Action-values are always with respect to a given policy. As this is still

Right Task – Probabilistic Rewards

Probabilistic rewards



Action-Value Function

s	a	$q(s,a)$	single episode
C	left	1	1
C	right	3	$0 \vee 1 \vee 5$

with discount factor $\gamma = 1$, but converges only for enough episodes.

Observation: As reward is probabilistic, we have to sample enough to get a good estimate. If we only sample once we might improve our policy in a way that excludes the optimal action and never recovers.

Important: We still have to guarantee sufficient exploration in order to (guaranteed!)

Convergence of Policy Improvement with Greedy Selection

Policy improvement assures us that π_{k+1} is uniformly better than π_k – unless π already converged and is then (guaranteed) the optimal policy.

But: this requires, that our estimates converges for each action, state pair for which we have to test these an infinite number of times.

Approach: Exploring Starts

In, e.g., simulated settings we can enforce this using Exploring Starts – and can assure starting from each possible state and selecting in that state each possible action.

Example: Blackjack scenario for which we can initialize states and select the actions.

We use a single policy for evaluation and directly improve this. This is called **on-policy**.

Problem: When exploring starts is not possible

Monte-Carlo Exploring Starts, for estimating $\pi \approx \pi_*$

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $sin\mathcal{S}, a \in \mathcal{A}(s)$
`Returns`$(s,a) \leftarrow$ empty list, for all $sin\mathcal{S}, a \in \mathcal{A}(s)$
for *each episode (without termination)* **do**
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0.
  Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
  $G \leftarrow 0$.
  for *each step of the episode* $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t, A_t *does not appear in* $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ **then**
      Append G to `Returns`(S_t, A_t)
      $Q(S_t, A_t) \leftarrow$ `average(Returns`(S_t, A_t)`)` `// Policy Evaluat.`
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ `// Policy Improvement`
    end
  end
end

Agent part of the algorithm; Environment interaction

Recap Example: Blackjack

Game: Play only against a dealer.

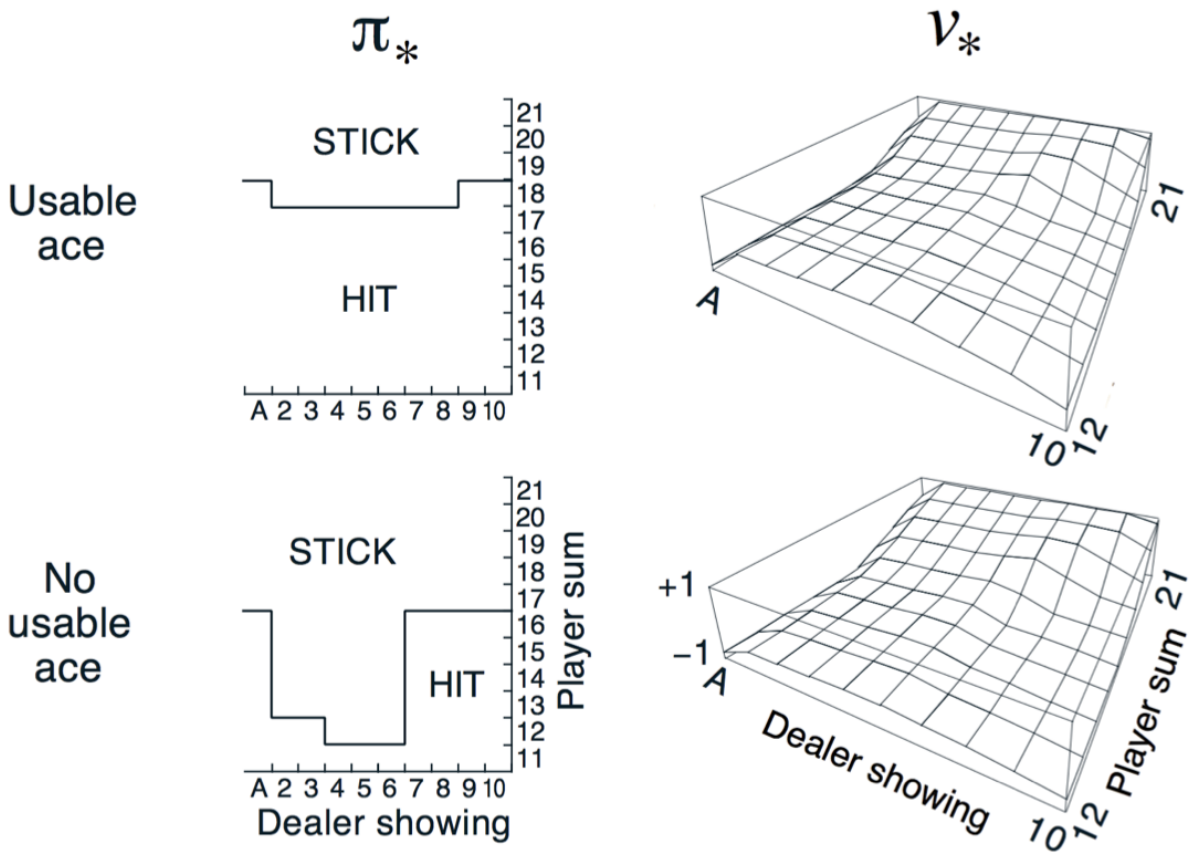
Goal: sum of cards is as great as possible without exceeding 21.

Counting:

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

MC Converged Policy

Optimal policy and state-value function for blackjack, found by Monte Carlo (using exploring starts) (Sutton und Barto 2018).
This direct improvement of a policy while running the environment interaction is called **on-policy**.



On and Off-Policy Learning

On-policy learning

- "Learn on the job"
- Learn about policy π from experience sampled from π
- e.g., using MC with exploring starts

Off-policy learning

- "Look over someone's shoulder"
- Learn about policy π from experience sampled from different policy b

(Silver 2015)

ε-Greedy Exploration

Simplest idea for ensuring continual exploration: Continue to sample randomly (for a small fraction).

- All m actions are tried with non-zero probability
- With probability $1 - \varepsilon$ choose the greedy action
- With probability ε choose an action at random

$$\pi(a|s) = \begin{cases} \frac{\varepsilon}{m+1-\varepsilon} & \text{if } a^* = \mathbf{arg\,max}_{a \in \mathcal{A}} q(s,a) \\ \frac{\varepsilon}{m} & \text{otherwise} \end{cases}$$

(Silver 2015)

On-Policy Characteristics

The policy ...

- is generally *soft*: $\pi(a|s) > 0 \forall s \in \mathcal{S}$ and $a \in \mathcal{A}$,
- gradually shifts closer and closer to a deterministic optimal policy.

We can use an ε-greedy policy.

> **ε-soft policy**
>
> A policy, for which
>
> $$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|} \forall \text{ states and actions for some } \varepsilon > 0$$
>
> Among ε-soft policies: ε-greedy policies are closest to greedy.

Overall: Idea of on-policy Monte Carlo methods is General Policy Improvement

Convergence: GLIE

Greedy in the Limit with Infinite Exploration (GLIE)

All state-action pairs are explored infinitely many times,

$$\lim_{t \to \infty} N_t(s, a) = \infty, \forall a, s$$

The policy converges on a greedy policy,

$$\lim_{t \to \infty} \pi_t(a|s) = \mathbb{1}\left(\arg\max_{a' \in \mathcal{A}} Q_t(s, a') \right)$$

For example, ε-greedy is GLIE if ε reduces to zero at $\varepsilon_t = \frac{1}{t}$

GLIE Monte-Carlo Control

Sample $k - th$ episode using π:

$$S_1, A_1, R_2, \ldots, S_T \sim \pi$$

For each state S_t and action A_t in the episode:

$$N(S_t, A_t) \rightarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \rightarrow Q(S_t, A_t) + \frac{1}{N(S_t. A_t)} \big(G_t - Q(S_t, A_t)\big)$$

Improve policy based on new action-value function

$$\varepsilon \rightarrow \frac{1}{k}, \pi \rightarrow \varepsilon\text{-greedy}(q)$$

(Silver 2015)

Convergence GLIE

GLIE Model-free control converges to the optimal action-value function, $q_t(s, a) \rightarrow q_*(s, a)$.

Model-free Control – Temporal Difference Learning

MC vs. TD Control

TD-learning has advantages over Monte-Carlo in prediction

- Lower variance
- Online
- Can use incomplete sequences

Natural Idea

Use TD instead of MC in our control loop

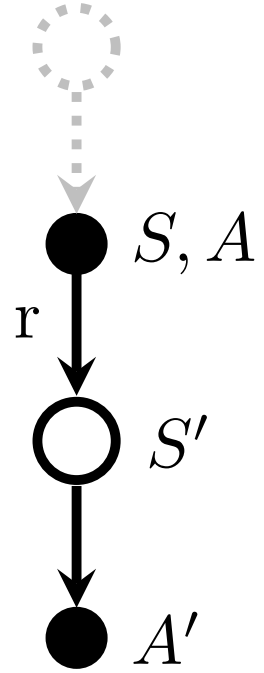
- Apply TD to $q(S, A)$
- Use ε-greedy policy improvement
- Update every time-step

(Hasselt und Borsa 2021)

SARSA - for update of the Action-Value Function

In every time-step:

- Policy evaluation: SARSA, $q(s, a) \approx q_\pi(s, a)$
- Policy improvement: ε-greedy policy improvement step



$$q'(s, a) \leftarrow q(s, a) + \alpha\Big(r + \gamma q(S', A') - q(S, A)\Big)$$

SARSA – On-Policy TD Control

Initialize $Q(s,a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\texttt{terminal},\cdot) = 0$.

for *each episode* **do**

  Initialize S

  Choose A from S using policy derived from Q (e.g. ϵ-greedy)

  for *each step in the episode, until state S is terminal* **do**

    Take action A, observe R, S'

    Choose A' from S' using policy derived from Q (e.g. ϵ-greedy)

    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$

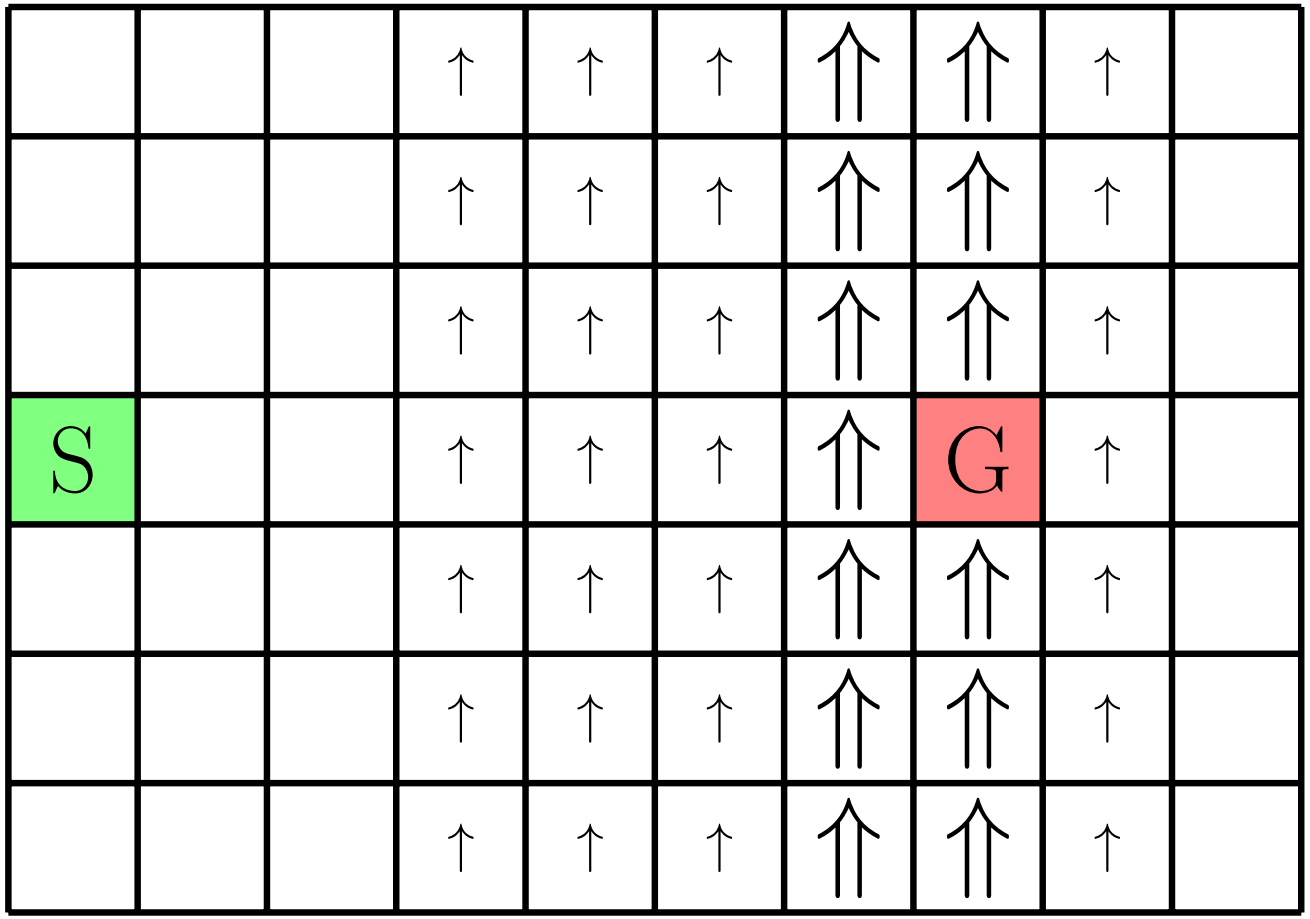
    $S \leftarrow S'$; $A \leftarrow A'$;

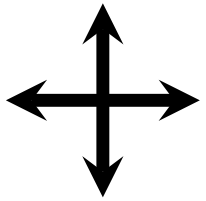
  end

end

Agent part of the algorithm; Environment interaction

Example for TD-Learning: Windy Grid World





actions

Reward:
−1 for each step

(Sutton und Barto 2018)

Example - Explanation: Windy Grid World

Goal: An agent should find a travel from the start point to the goal point.

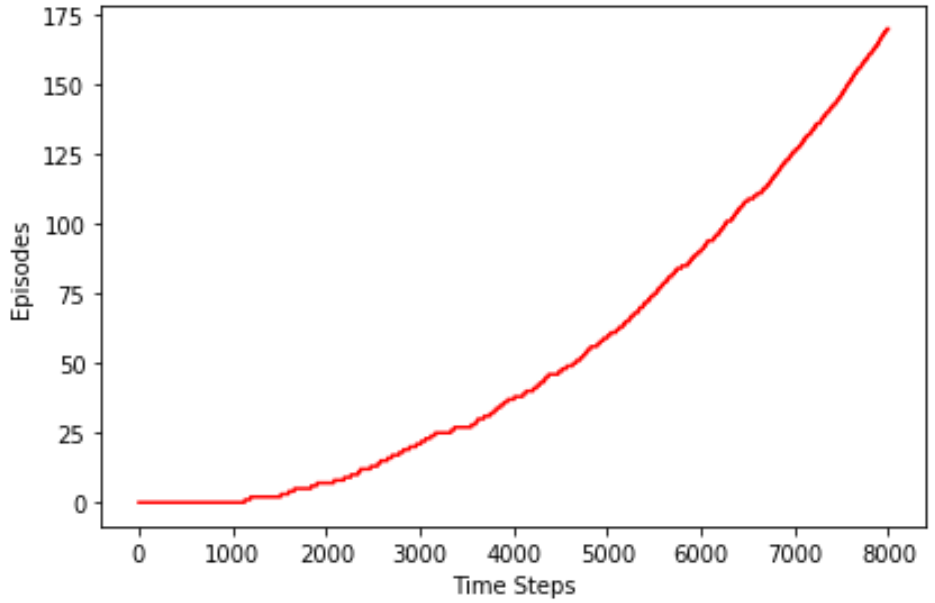
Environment:

- 10×7 grid environment
- actions in the four major directions
- crosswind (upwards) $\rightarrow$ shifts agent one grid further to the top

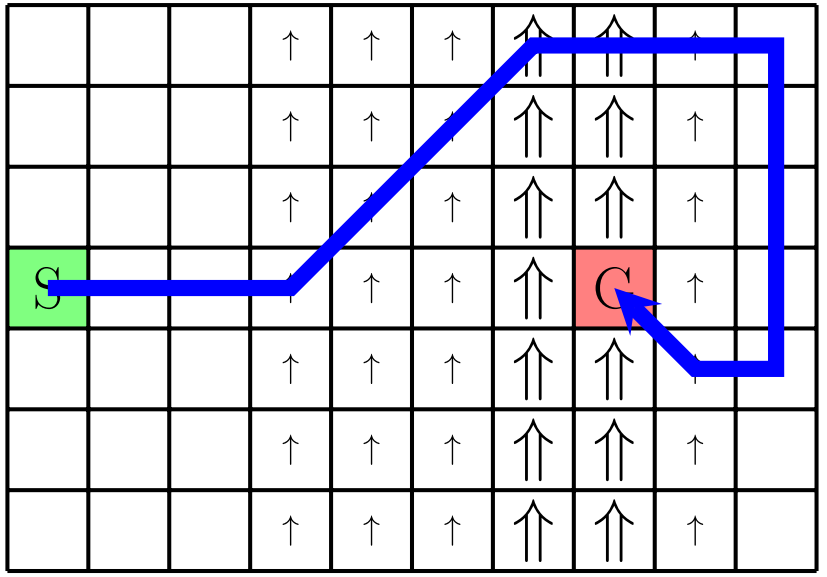
Reward: Aiming for reac hing goal as fast as possible, -1 for each timestep in environment.

Example Results for Windy Grid World via SARSA

Learning optimal path



Learned Policy



$\varepsilon = 0.1, \alpha = 0.5,$
initially $q(S, A) = 0 \forall S, A.$

Policy Improvement Example

You can use the known grid world environment and apply GPI (policy evaluation and policy improvement) in this interactive grid environment using TD learning instead of Dynamic Programming.

(Karpathy 2015)

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).

Fortmann-Roe, Scott. 2012. „Understanding the Bias-Variance Tradeoff“. http://scott.fortmann-roe.com/docs/BiasVariance.html.

Haarnoja, Tuomas, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, und Sergey Levine. 2018. „Learning to Walk via Deep Reinforcement Learning“. *CoRR* abs/1812.11103. http://arxiv.org/abs/1812.11103.

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

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

Satakhutdinov, Ruslan. 2018. „Deep RL and Control“. Course 10703, Carnegie Mellon University, School of Computer Science.

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.