First Steps in Reinforcement Learning

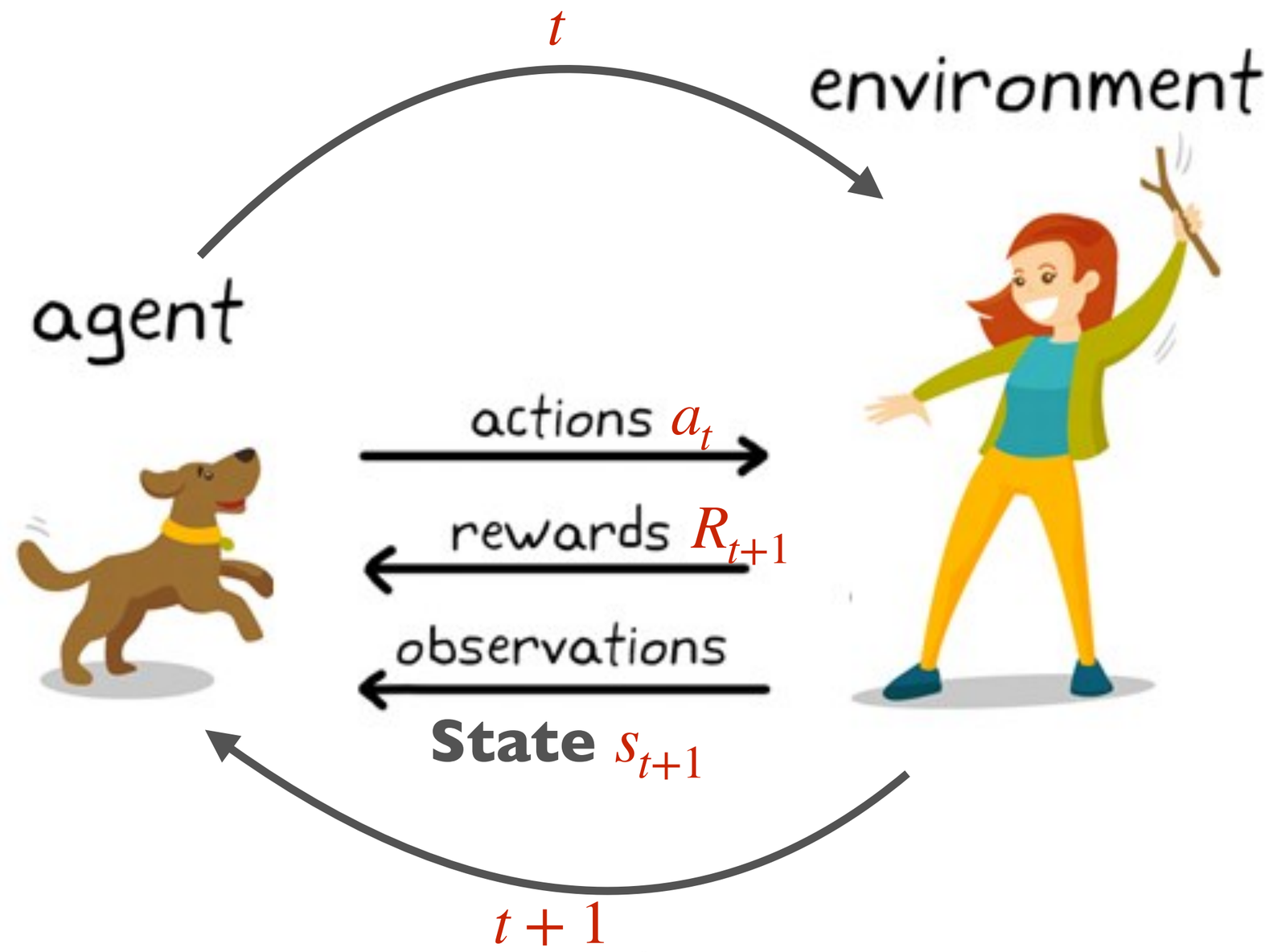
Policy Iteration, Monte Carlo Learning and SARSA

Volker Krüger

with slides stolen from David Silver's lecture and Figs. from the book by Sutton and Barto

LUND UNIVERSITY

Revisiting
Markov Decision Processes


t
environment
agent
actions a_t
rewards R_{t+1}
observations
State s_{t+1}
t + 1

Markov Decision Process

- A Markov Decision Process is a Markov Reward Process with decisions on actions

Definition: A **Markov Decision Process** is a tupel $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$

- $\mathbf{S}$ is a finite set of states
- $\mathbf{A}$ is a finite set of actions
- $\mathbf{P}$ is a state transition probability matrix $P_{ss'}^{a} = P\left(S_{t+1} = s' \,|\, S_t = s, A_t = a\right)$
- $\mathbf{R}$ is a reward function $R_s^a = E\left(R_{t+1} \,|\, S_t = s, A_t = a\right)$
- γ is a discount factor $\gamma \in [0,1]$

LUND UNIVERSITY

Value Functions

Definition: The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s and following policy π

$$v_\pi(s) = E_\pi\left\{G_t \,|\, S_t = s\right\} = E_\pi\left\{R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s\right\}$$

Definition: The *action-value function* $q_\pi(s, a)$ of an MDP is the expected return starting from state s by taking action a and following policy π

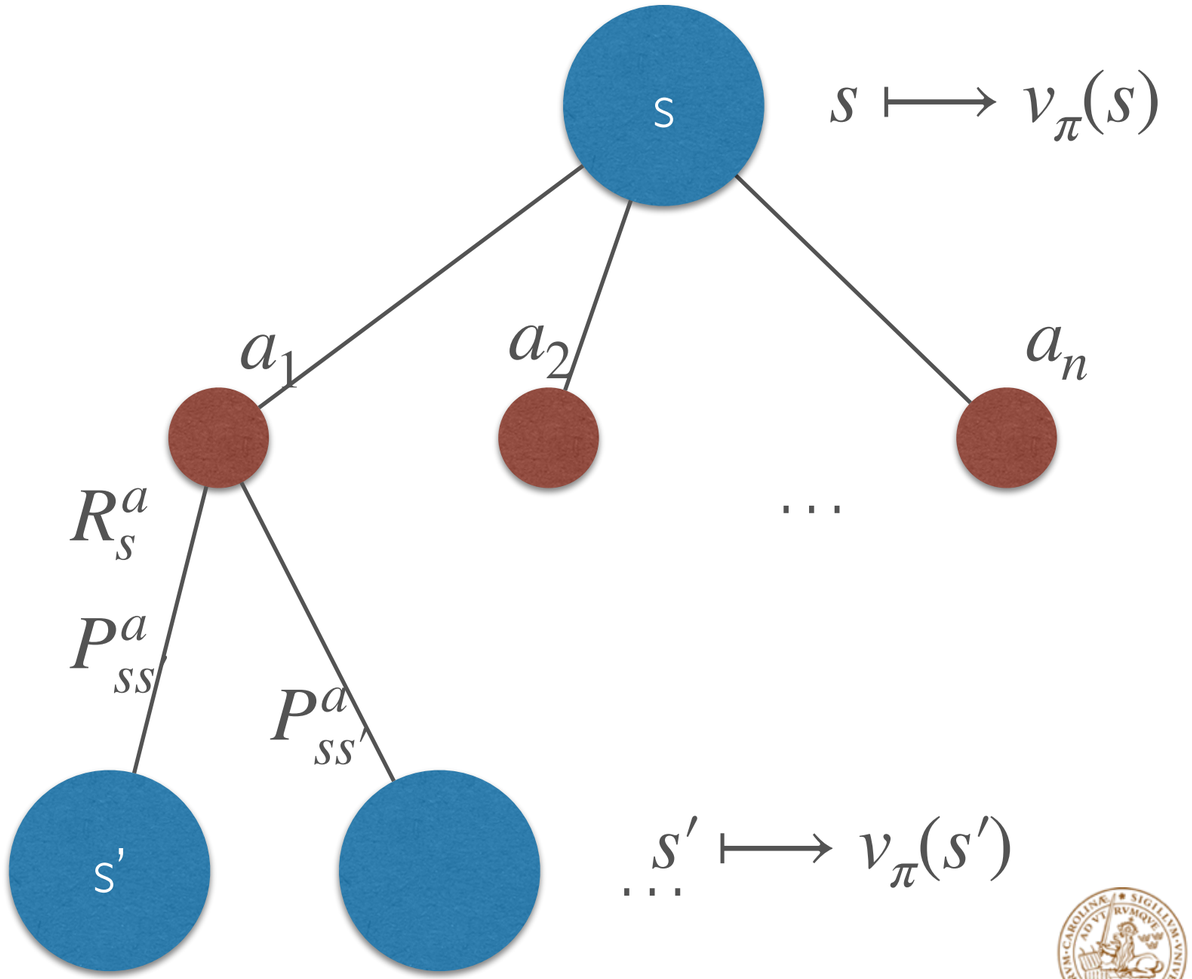
$$q_\pi(s, a) = E_\pi\left\{G_t \,|\, S_t = s, A_t = a\right\} = E_\pi\left\{R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a\right\}$$

LUND UNIVERSITY

Bellman Expectation Equation

- From state s, we can take different actions $a_i \in \mathbf{A}$ chosen according to $\pi(a \mid s)$
- Based on s and a we get a reward R_s^a

$$v_\pi(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_\pi(s') \right)$$

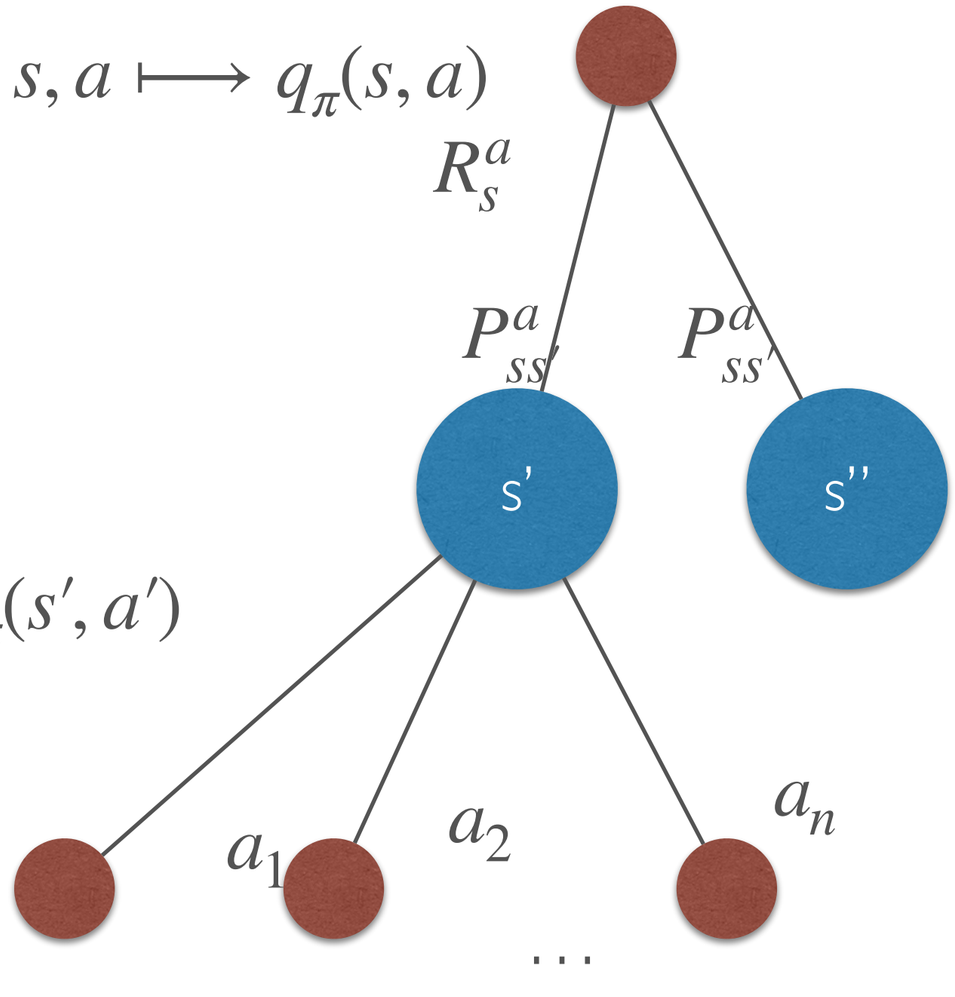


Bellman Expectation Equation

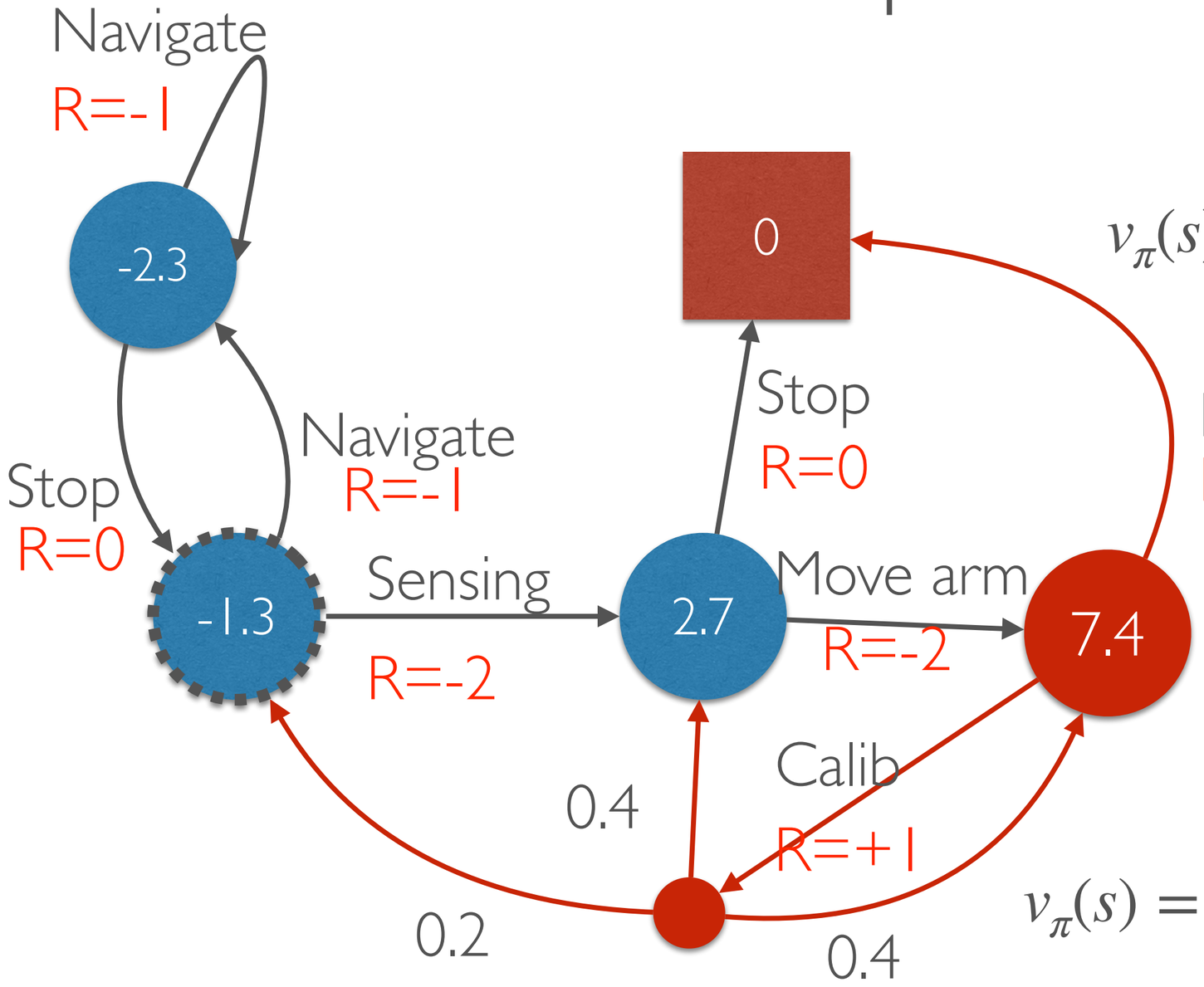
$$s, a \longmapsto q_\pi(s, a)$$

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a \sum_{a' \in \mathbf{A}} \pi(a' \mid s') q_\pi(s', a')$$

$$s', a_i \longmapsto q_\pi(s', a_i)$$



Bellman expectation Equation



$v_\pi(s)$ for $\pi(s,a) = 0.5,\ \gamma = 1$

$$v_\pi(s) = 0.5 \cdot \big(1 + 0.2 \cdot (-1.3) + 0.4 \cdot 2.7 + 0.4 \cdot v_\pi(s)\big) + 0.5 \cdot 10$$

$$v_\pi(s) = 7.4$$

$$v_\pi(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_\pi(s') \right)$$

LUND UNIVERSITY

Plan for Today

- How can the state value and the action value functions be computed?
  - ***What do we need to know for doing this?***
- How can we find an optimal policy?

- Start with a first simple solution: **Iterative Policy Evaluation** and **Policy Iteration**
- Then: first steps in **Model-free Prediction and Control:**
  - Monte Carlo Learning
- Outlook: **Temporal Difference Learning** (TD-Learning) and SARSA
  - SARSA is similar to the famous Q-Learning appoach.

Prediction, Control

- We have given a fully defined MDP $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$
- We also assume to be given some policy π
- We want to know the state values $v_\pi(s)$ of each state s for the policy.
  - This is called ***Prediction*** of how good the plan / policy is.
- Then, we want to know π^*, i.e., the optimal policy for the MDP
  - This is called ***Control***.

Policy Evaluation To Find v_π

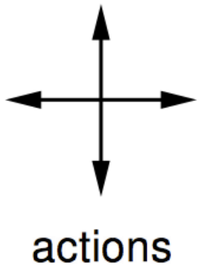
Iterative Policy Evaluation

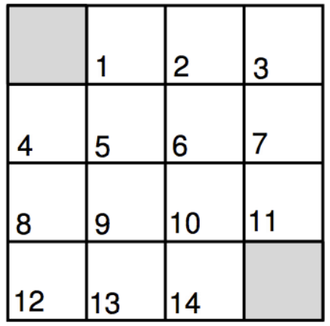
- Goal: evaluate a given policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$

- Initialise all state value to 0: $v_1(s) = 0$ for all $s \in S$
- At each iteration $k + 1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s

$$v_{k+1}(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s') \right)$$

Example: Random Policy in the Small Gridworld





$r = -1$ for all transition

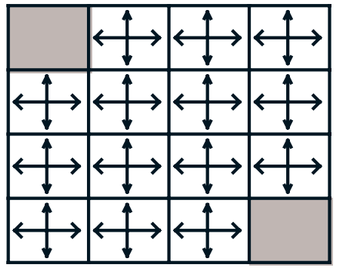
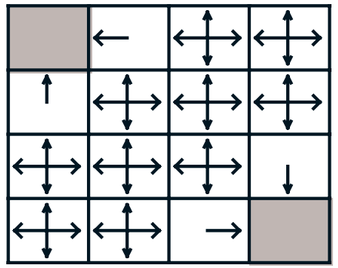
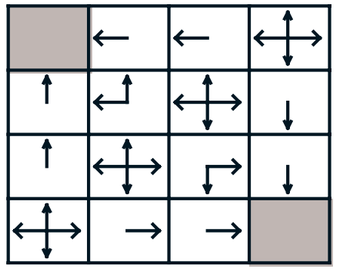
$\gamma = 1$

- Non-terminal states $1 \ldots 14$
- terminal state in grey
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n, \cdot) = \pi(e, \cdot) = \pi(s, \cdot) = \pi(w, \cdot)$$

LUND UNIVERSITY

Example: Random Policy in the Small Gridworld

	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 ← 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	

$$R = -1, \gamma = 1$$

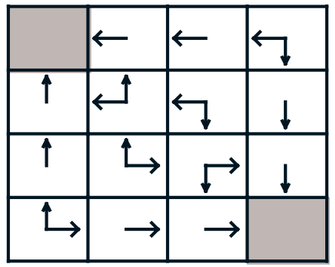
$$\pi(n, \cdot) = \pi(e, \cdot) = \pi(s, \cdot) = \pi(w, \cdot)$$

$$v_{k+1}(s) = \sum_{a \in \mathbf{A}} \pi(a \mid s) \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s') \right)$$

Example: Random Policy in the Small Gridworld

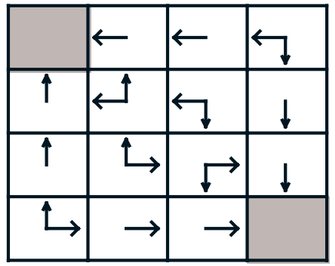
$k = 3$

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



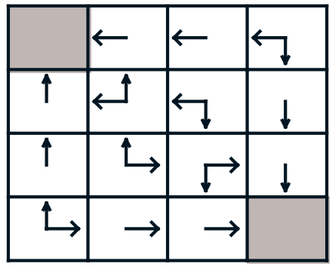
$k = 10$

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



$k = \infty$

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



optimal policy

Policy Iteration To Improve the Policy

How to improve the Policy

- Given a policy.
  - Evaluate the policy

$$v_\pi = E\left\{R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right\} = \sum_{a \in \mathbf{A}} \pi(a \mid s)\left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s')\right)$$

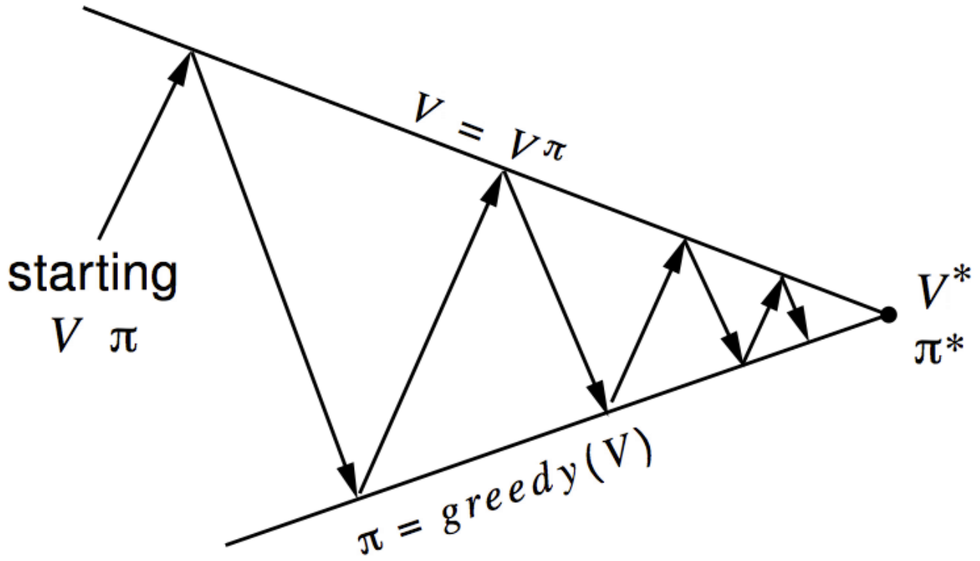
  - Improve the policy by acting greedily with respect to v_π

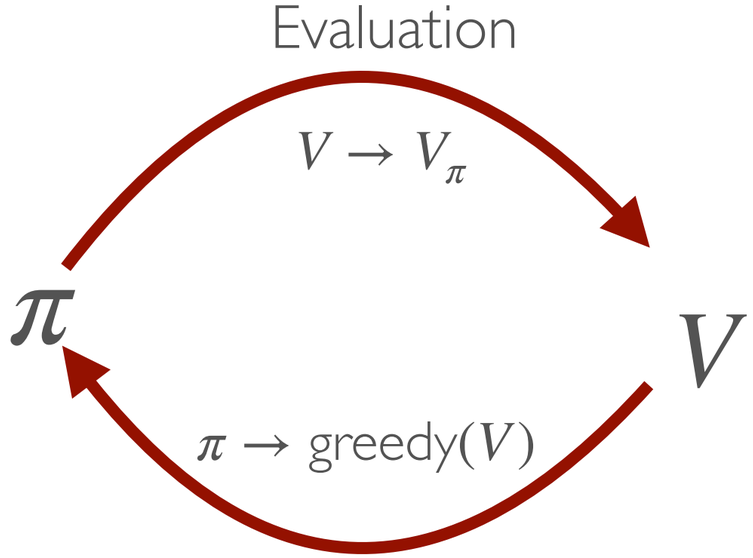
$$\pi' = \text{greedy}(v_\pi) \qquad \pi'(s) = \arg\max_{a \in A}\left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s')\right)$$

- In the Gridworld the improved policy was optimal $\pi^{k \geq 3} = \pi^*$
- In general, more iterations are needed
- But this process of **policy iteration** always converges to π^*

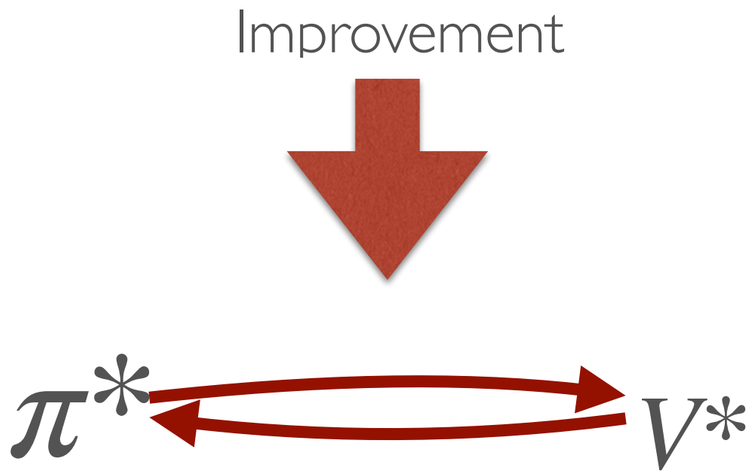
LUND UNIVERSITY

Policy Iteration



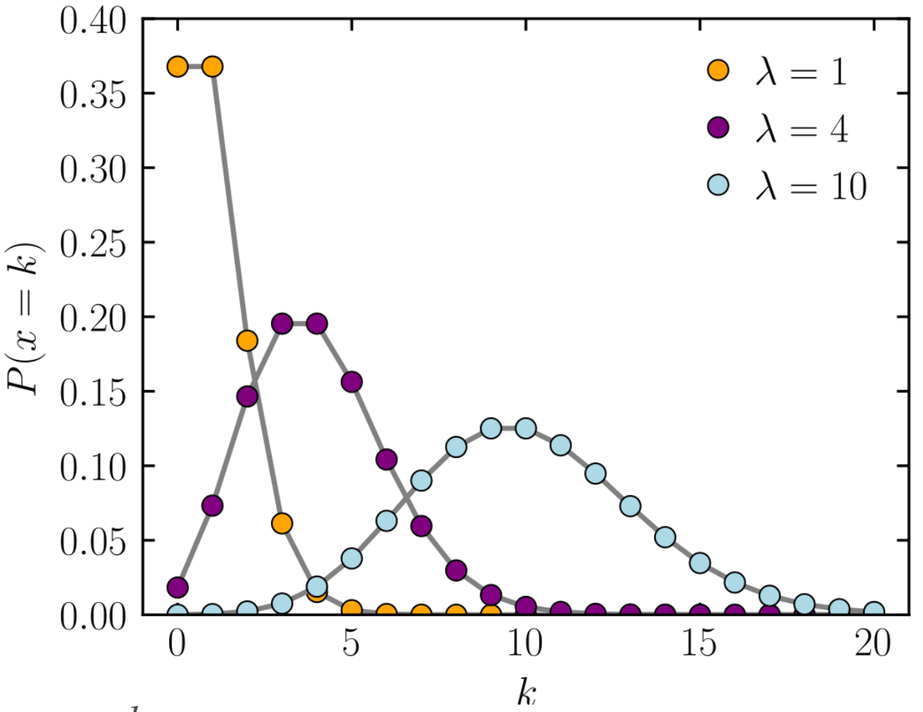


- **Policy Evaluation**: Estimate v_π
  - Iterative policy evaluation
- **Policy improvement**: Generate $\pi' \geq \pi$
  - Greedy policy improvement



LUND UNIVERSITY

Christmas Task: Play with Jack's Car Rental



- **States**: Two locations, maximum 20 cars at each location
- **Actions**: Move up to 5 cars between locations overnight
- **Reward**: 10€ for each car rented if available, -2€ for moving a car
- **Transitions**: Cars returned and requested randomly
  - Poisson distribution, *k* returns/requests with probability $P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$
  - 1st location: average requests=3, average returns=3
  - 2nd location: average requests=4, average returns =2

Policy: how many cars to move?

LUND UNIVERSITY

Policy Improvement

Consider a deterministic policy $a = \pi(s)$

We can *improve* the policy by acting greedily $\pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$

This improves the value from any state s over one step

Blackboard

- If improvement stops

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

- Bellman optimality equation has been satisfied

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

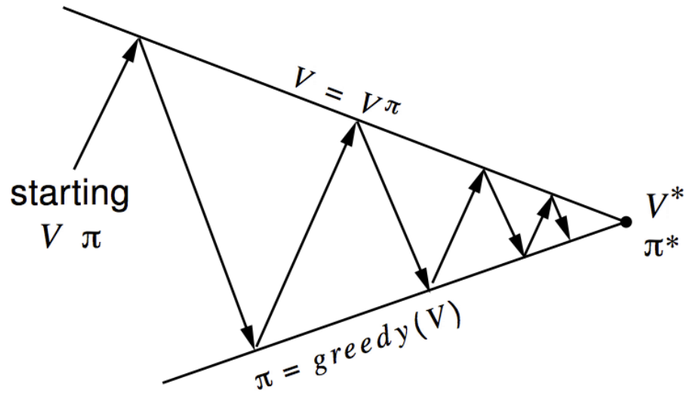
- Thus, $v_{\pi}(s) = v^*(s)$ for all $s \in S$
- and π is the optimal policy.

Some amazing facts

- **Policy evaluation** always converges to v_π
- **Policy Iteration** always converges to π^*
- *Synchronous* approach used above
- *Asynchronous* approach converges as well if all states are selected
- How do we know we have reached v_π ?
  - How many repetitions k ?
- How do we know we have reached π^* ?
- Simply stop after $k = 1$ and update after each iteration
  - This is equivalent to ***value iteration***

Generating a new improved π'

Improving the same π



LUND UNIVERSITY

Value Iteration

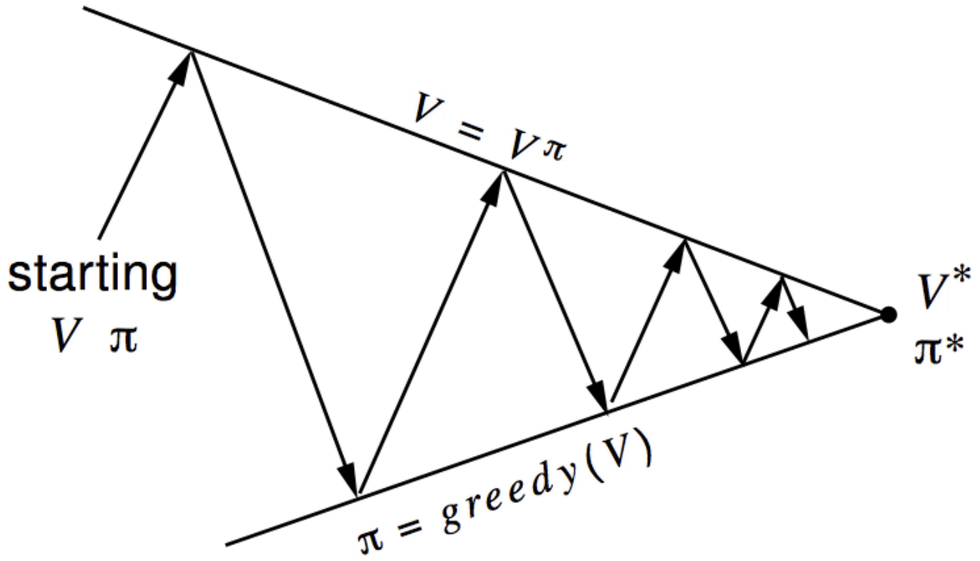
- Goal: find the optimal policy π
- Solution: Iterative application of the Bellman expectation equation $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- Initialise all state value to 0: $v_1(s) = 0$ for all $s \in S$
- At each iteration $k + 1$
  - For all states $s \in S$
    - Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s

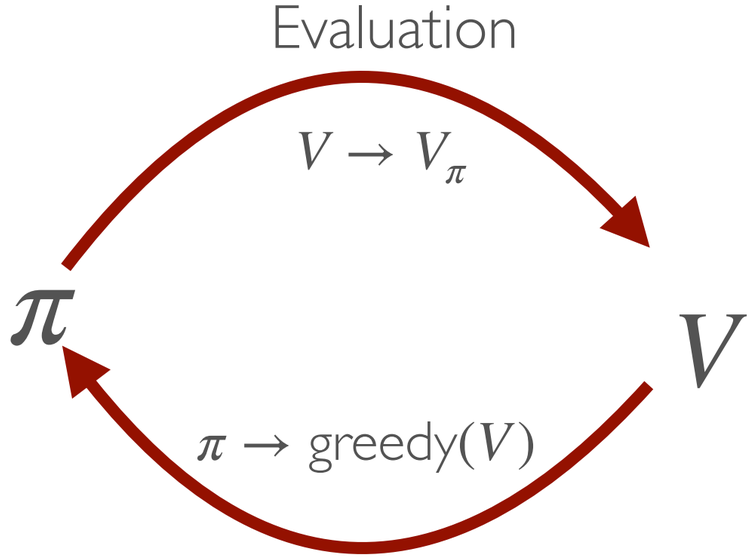
$$v_{k+1}(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_k(s') \right)$$

- Note: there is no(!!) explicit policy here
  - we simply go to the state with the biggest reward
- Intermediate value functions might not correspond to any policy.

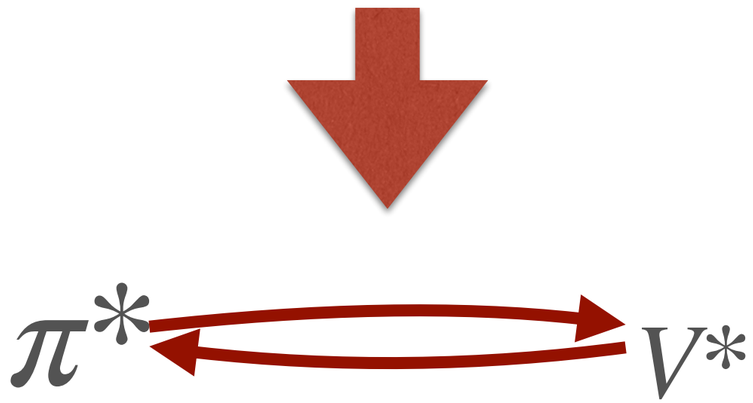
Done when there are no more updates

Generalized Policy Iteration





- **Policy Evaluation**: Estimate v_π
  - ***Any*** policy evaluation approach
- **Policy improvement**: Generate $\pi' \geq \pi$
  - ***Any*** policy improvement approach



LUND UNIVERSITY

Summary so far

- Identified the optimal policy, state value function and action value function
- **Policy Iteration** and **Value Iteration** for finding the optimal policy.
  - Value Iteration: very simple approach, but still finds π^*
- The MDP needs to be known completely
  - ⛔ The transition probabilities $P^a_{s,s'}$ are usually not known
  - ⛔ It's not obvious how one can extend this to continuous actions and states
  - ➕ Everything is mathematically exact, expressible and analysable and best for small problems.

Summary so far

- Transition function $P^a_{s,s'}$? Rewards R^a_s ?
  - Two common cases
    1. The MDP is *unknown* but the experience is available: *Let's try things out and see what happens*…
    2. The MDP is known but too complicated to use.
- **Goal**:
  - ***Estimate*** the value function for an *unknown* MDP
  - ***Optimise*** the value function and the policy of an *unknown* MDP

Monte Carlo Learning for Model-Free Prediction and Control

Concrete Example: Blackjack

- **States** (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- **Action stick**: Stop receiving cards (and terminate)
- **Action twist**: Take another card (no replacement)
- **Reward** for **stick**:
  - +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards
- **Reward** for **twist**:
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- **Transitions**: automatically twist if sum of cards < 12

LUND UNIVERSITY

Learning from Episodes

- The goal: estimate the value function of an *unknown* MDP
- The trick: Use experiences!!

We learn v_π from **episodes of experience** under the policy π.

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

Monte Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience / observations
- MC uses simplest possible idea: value = sample mean of the return

Reminder: Expected Return

- Remember:
  - the ***return*** $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
  - The ***value function*** is the expected return: $v_\pi(s) = E_\pi \left\{ G_t \,|\, S_t = s \right\}$

- The goal is to estimate the expected return from the observed episodes

$$E^1 : S_1^1, A_1^1, R_2^1, \ldots, S_k^1$$
$$E^2 : S_1^2, A_1^2, R_2^2, \ldots, S_k^2$$
$$\ldots$$
$$E^n : S_1^n, A_1^n, R_2^n, \ldots, S_k^n$$

LUND UNIVERSITY

Monte Carlo Policy Evaluation

Counting and Averaging

- Episodes $E^1, E^2, \ldots, E^n$ are given as training data
- For every episode $E^i : S_1^i, A_1^i, R_2^i, \ldots, S_k^i$
  - **First** times step t where state s is visited in episode E_i
    - increment counter $N(s) \leftarrow N(s) + 1$
    - Increment total return $S(s) \leftarrow S(s) + G_t$
- The estimated mean return: $V(s) \leftarrow S(s)/N(s)$

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

- Law of large numbers: $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

Monte Carlo Policy Evaluation

Counting and Averaging

- Episodes $E^1, E^2, \ldots, E^n$ are given as training data
- For every episode $E^i : S_1^i, A_1^i, R_2^i, \ldots, S_k^i$
  - **Every** times step t that state s is visited in episode E^i
    - increment counter $N(s) \leftarrow N(s) + 1$
    - Increment total return $S(s) \leftarrow S(s) + G_t$
- The estimated mean return: $V(s) \leftarrow S(s)/N(s)$

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

- Law of large numbers: $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

We have to have all episodes in memory!!

Recursive computation of mean

- The mean μ_k can be computed recursively from the sequence $x_1, x_2, \ldots$

$$
\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) \\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k}\underbrace{(x_k - \mu_{k-1})}_{\text{error-term}}
\end{aligned}
$$

Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i:
  - For every state S_t with return G_t
    - increment counter $N(s) \leftarrow N(s) + 1$
    - ~~Increment total return $S(s) \leftarrow S(s) + G_t$~~
- The estimated mean return: ~~$V(s) \leftarrow S(s)/N(s)$~~

Incremental Monte-Carlo Updates

- Update each $V(s)$ after each episode E^i:
  - For every state S_t with return G_t
    - increment counter $N(s) \leftarrow N(s) + 1$
    - ~~Increment total return $S(s) \leftarrow S(s) + G_t$~~
    - The estimated mean return: ~~$V(s) \leftarrow S(s)/N(s)$~~

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

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

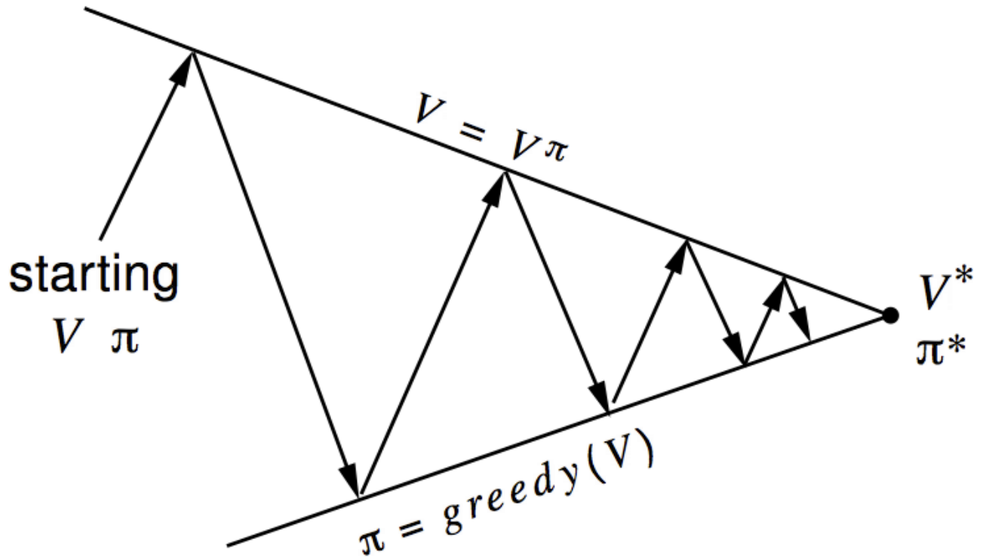
We can work incrementally!!
Sample one episode at a time

Optimizing the Policy

Monte Carlo Control

Policy Iteration

- **Policy Evaluation**: Estimate v_π
  - Iterative Policy Evaluation
  - Monte Carlo Policy Evaluation
- **Policy improvement**: Generate $\pi' \geq \pi$
  - Greedy policy improvement **???**



LUND UNIVERSITY

Computing the Policy

- Greedy policy improvement over $V(s)$ requires knowledge of the MDP:

$$\pi'(s) = \arg\max_{a \in A} \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v_\pi(s') \right)$$

- Greedy improvement over $Q(s,a)$ is model-free.

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

This is the Trick!!!

Computed incrementally exactly like $V(S_t)$ with incremental monte carlo updates!!

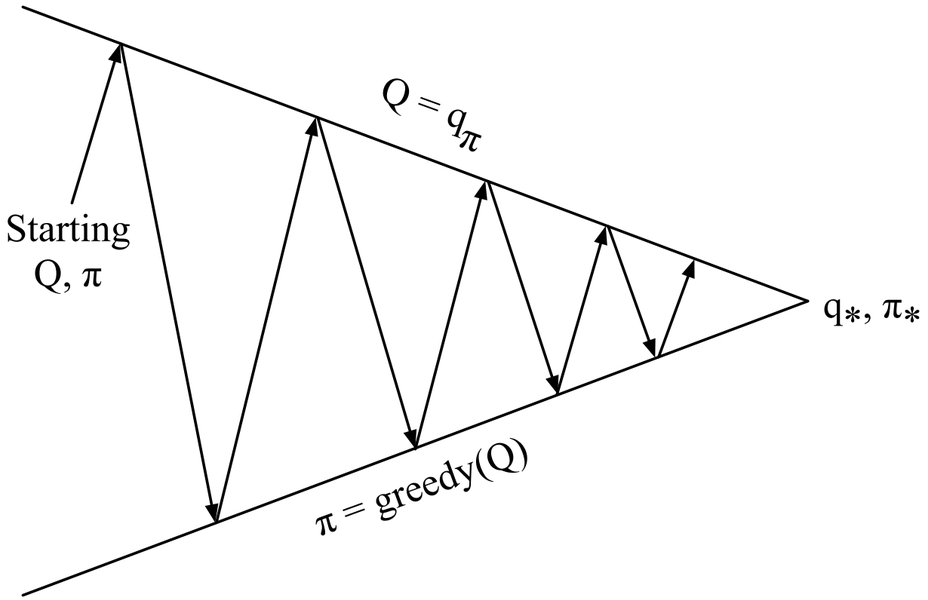
- What we need is a statistic of state transitions s given an action a

$$S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_k$$

Generalized Policy Iteration for Action Value Function

- Policy Evaluation:
  - Monte Carlos Policy Evaluation of $Q = q_\pi$
- Policy Improvement:
  - Greedy policy improvement **???**

$$\pi'(s) = \arg\max_{a \in A} Q(s, a) \; ??$$



LUND UNIVERSITY

Example of a greedy action

- Two restaurants you can choose from
  - The top one was closed and we got no food.
    - $V(top) = 0$
  - You go to the bottom one and get a reward +1
    - $V(bottom) = +1$
  - You go to the bottom one and get a reward +2
    - $V(bottom) = +2$
  - You go to the bottom one and get a reward +1
    - $V(bottom) = +1$
  - …
  - Are you sure you chose the right restaurant?

LUND UNIVERSITY

We learn on the job!!!
The policy we can find when using greedy improvement will depend on the policy we have now.
We very probably miss certain state-action pairs.

ϵ-greedy exploration

- *Try out something new for a change!!* **exploration vs exploitation**
- Simplest idea is to keep exploring: **ϵ-greedy**
  - *All* m actions are tried out with above zero probability:
  - With probability $\epsilon > 0$ we choose an action at random
  - With probability $1 - \epsilon$ we use the greedy action

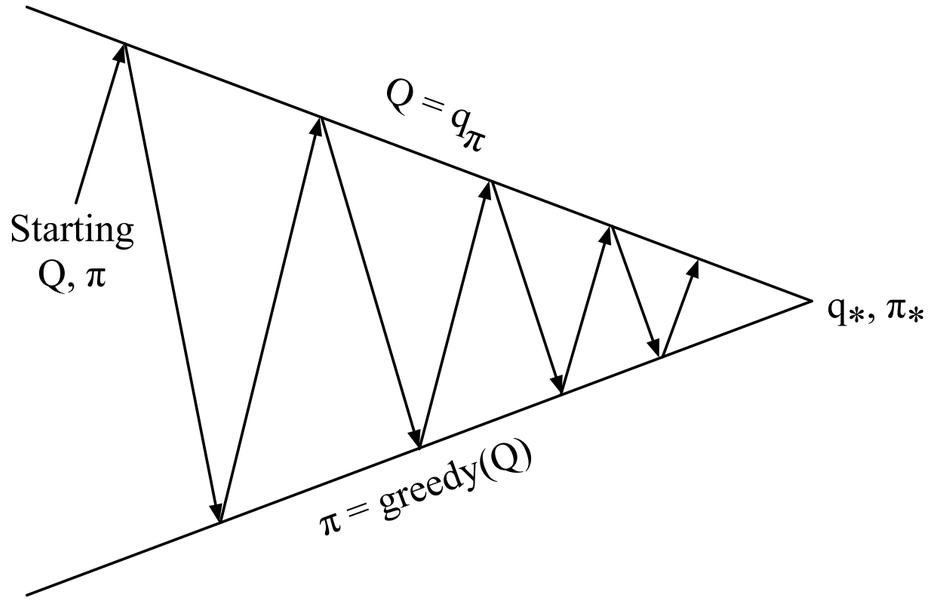
$$\pi'(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a' \in A} Q(s, a') \\ \epsilon/m & \text{otherwise} \end{cases}$$

ϵ-greedy Policy Improvement

Theorem: For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π improves the value function: $v_{\pi'}(s) \geq v_\pi(s)$

Monte Carlo Control

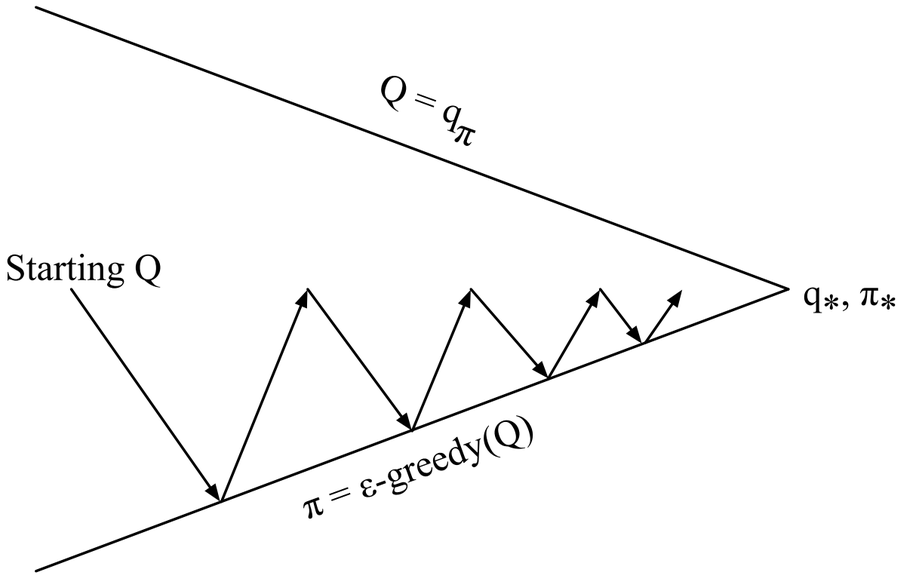
- Policy Evaluation:
  - Monte Carlos Policy Evaluation of $Q = q_\pi$
- Policy Improvement:
  - ϵ-greedy policy improvement



LUND UNIVERSITY

Monte Carlo Control more efficient

- For every episode:
  - Policy Evaluation:
    - Monte Carlos Policy Evaluation of $Q \approx q_\pi$
  - Policy Improvement:
    - Generate a new episode with ϵ-greedy policy



LUND UNIVERSITY

GLIE Monte Carlo Control

- **Greedy in the Limit with Infinite Exploration** (GLIE)
- Two principles:
  1. keep exploring
  2. make sure that the policy eventually becomes greedy!
     - needed for the Bellman optimality equation.

- One possibility: ϵ-greedy policy with slow decay of ϵ: e.g., $\epsilon_k = \frac{1}{k}$

GLIE Monte Carlo Control

- Generate a new episode E^k using your policy $\pi : S_1, A_1, R_2, S_2, A_2, R_3, \ldots, R_T$
- For each state S_t and action A_t in the episode

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right)$$

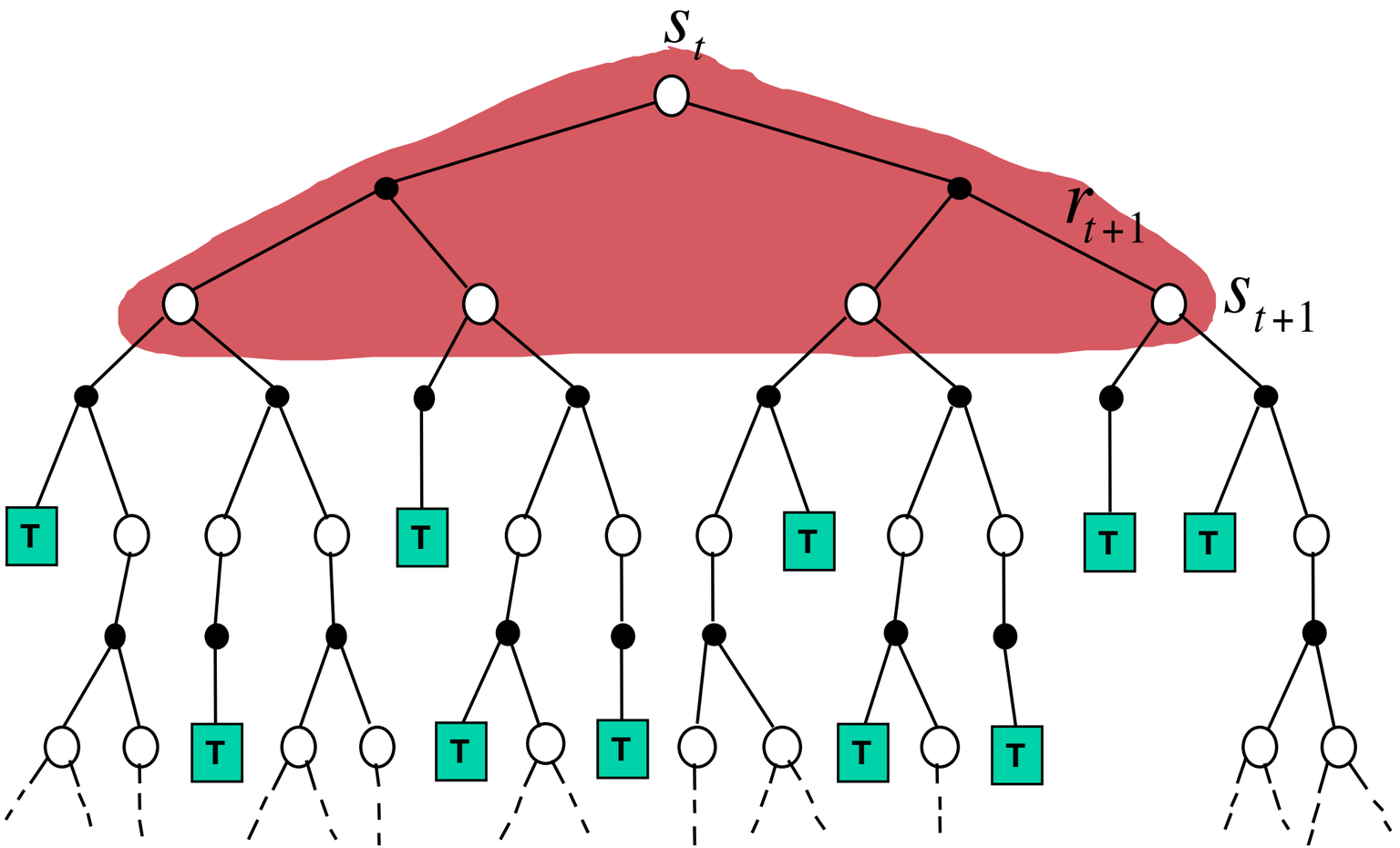
- Improve the policy based on the new $Q(S, A)$

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

GLIE Monte Carlos converges to the optimal $q^*(s, a)$

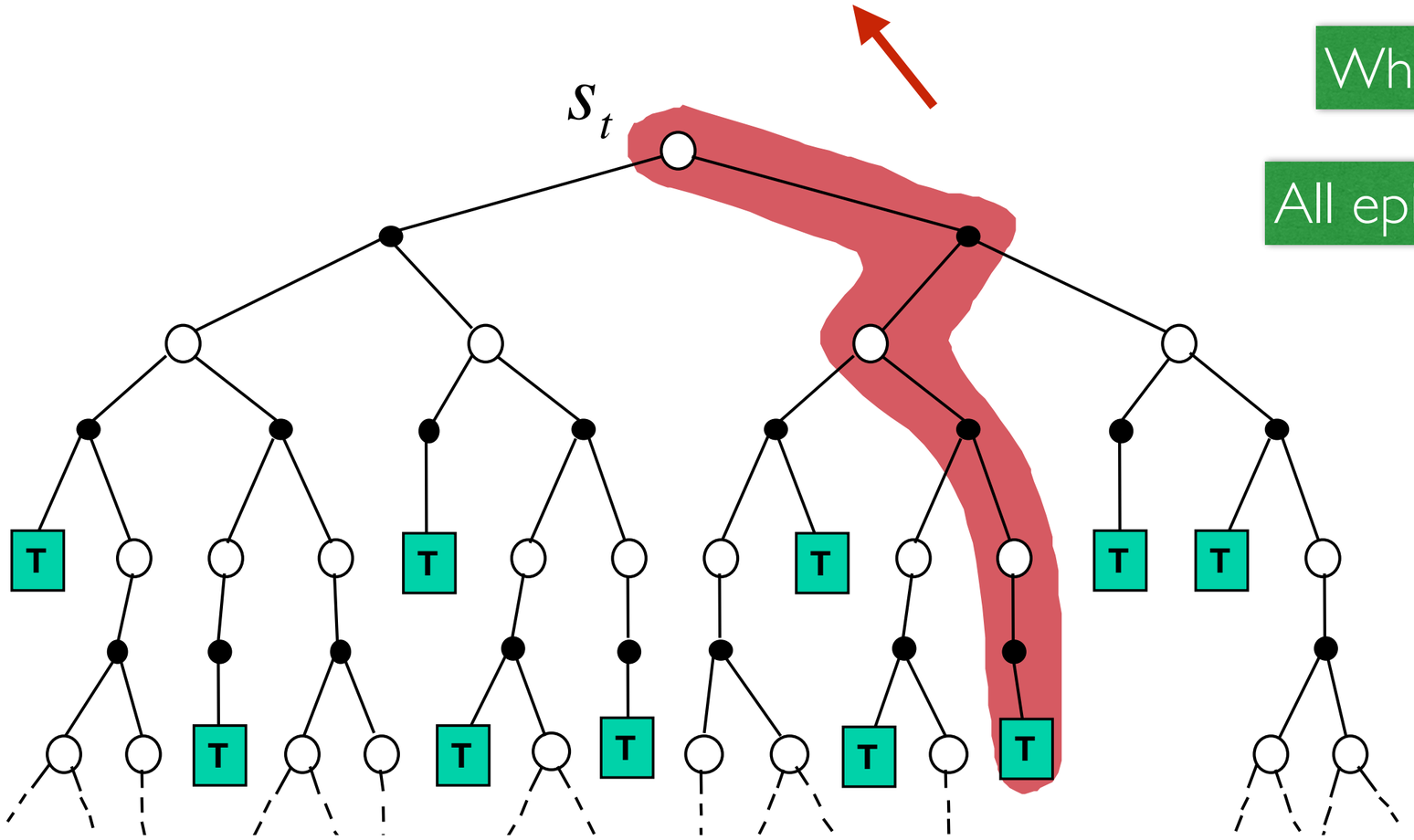
Dynamic programming backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



Monte Carlo Backup

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



What's the problem?

All episodes must end!!

What was the Bellman Eq again?

$$v_\pi(s) = E_\pi\left\{G_t \,|\, S_t = s\right\} = E_\pi\left\{R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s\right\}$$

Temporal Difference Learning: TD-Learning

- Goal: learn v_π from online experiences under the policy π

- Simplest temporal difference learning algorithm: TD(0)
  - Update $V(S_t)$ towards the ***estimated return*** $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called *TD error*

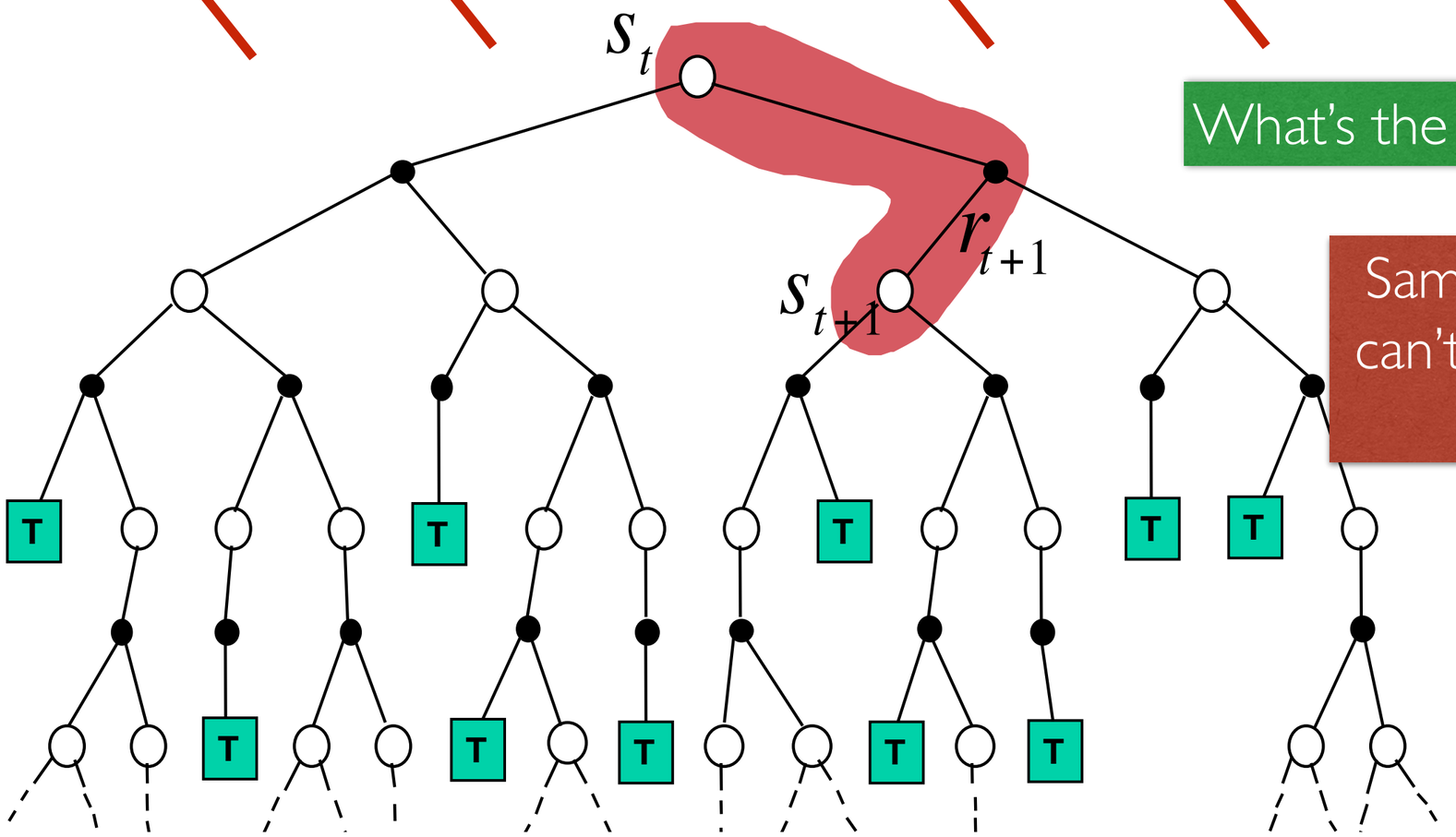
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

Comparison TD vs MC

- TD: updates ***online towards the best guess***, *before* the final outcome
- MC has to wait until the episode is complete and we know the final return G_t
- TD learns from incomplete episodes, e.g., without final outcome.
- MC requires complete episodes

Temporal Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



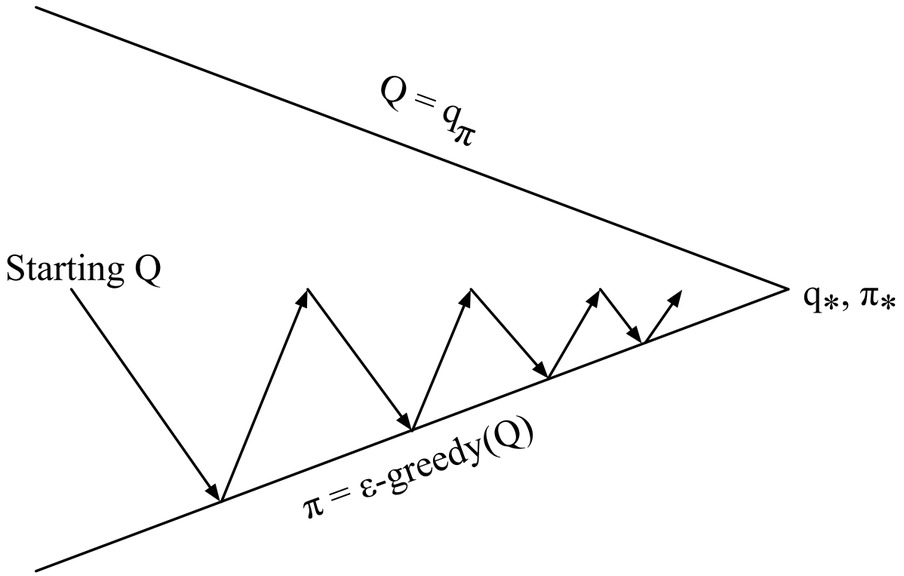
What's the problem this time?

Same as with MC: we can't optimise π based on $V(S)$

LUND UNIVERSITY

Policy Optimisation with TD

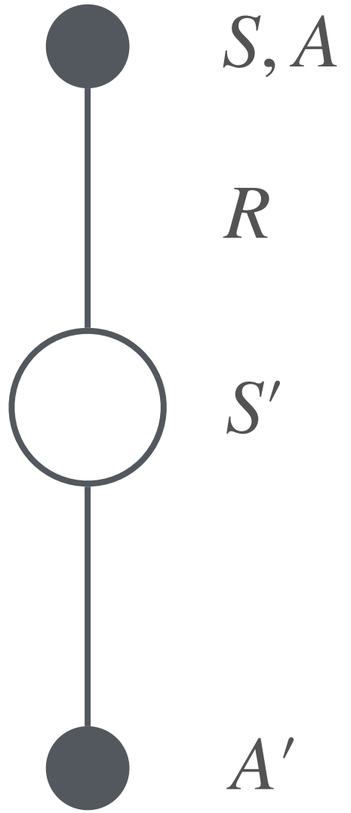
- For every episode:
  - Policy Evaluation:
    - TD-Learning idea to find $Q \approx q_\pi$
  - Policy Improvement:
    - ϵ-Greedy policy improvement



Results in the SARSA approach: Closely related to famous Q-Learning

LUND UNIVERSITY

Action-Value Function with SARSA



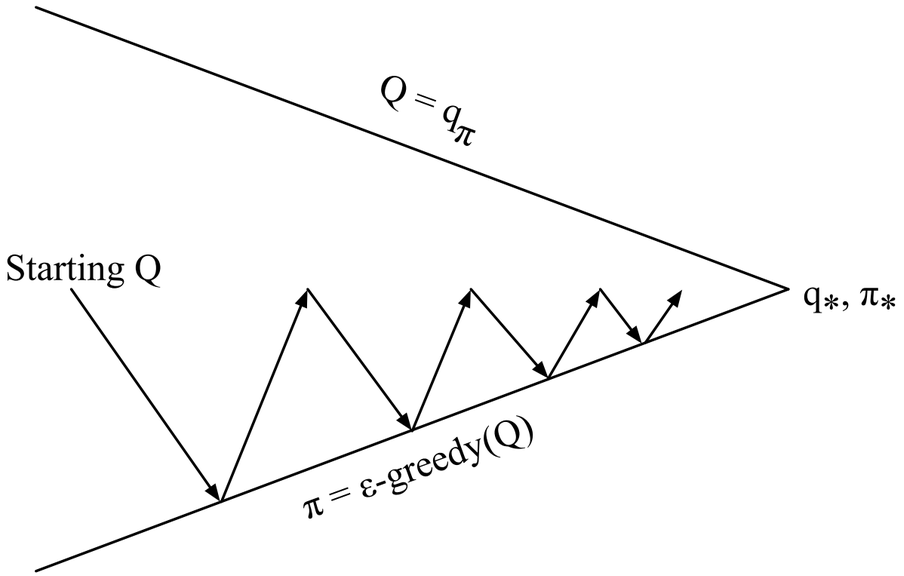
- At **every time step** we improve our policy
  - S, A: In state S choose A according to ϵ-greedy policy π
  - R, S' sampled from the environment
  - A' chosen ϵ-greedy: according to ϵ-greedy policy π

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} \left(G_t - Q(S_t, A_t)\right)$$

On Policy Control with SARSA

- For **every single time step**:
  - Policy Evaluation:
    - **SARSA** update of $Q \approx q_\pi$
  - Policy Improvement:
    - ϵ-greedy policy improvement
    - no explicit policy necessary



LUND UNIVERSITY

SARSA Algorithm for on-policy control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

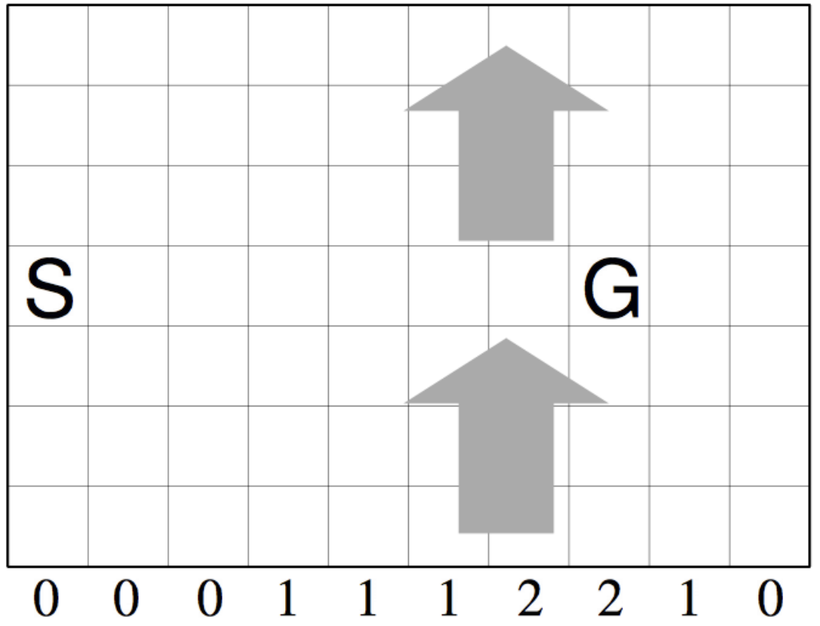
Convergence of SARSA

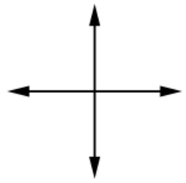
- **Theorem**: *SARSA converges to the optimal action-value function* $Q(S, A) \rightarrow q^*(s, a)$, *under the following conditions:*
  - *GLIE sequence of policies* $\pi_t(a, s)$
  - *Robbins-Monro sequence of step-sizes* α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

  - $\alpha_k = \frac{1}{k}$ *works*
  - SARSA almost always works, no matter GLIE and α_t 😀

Windy Gridworld example



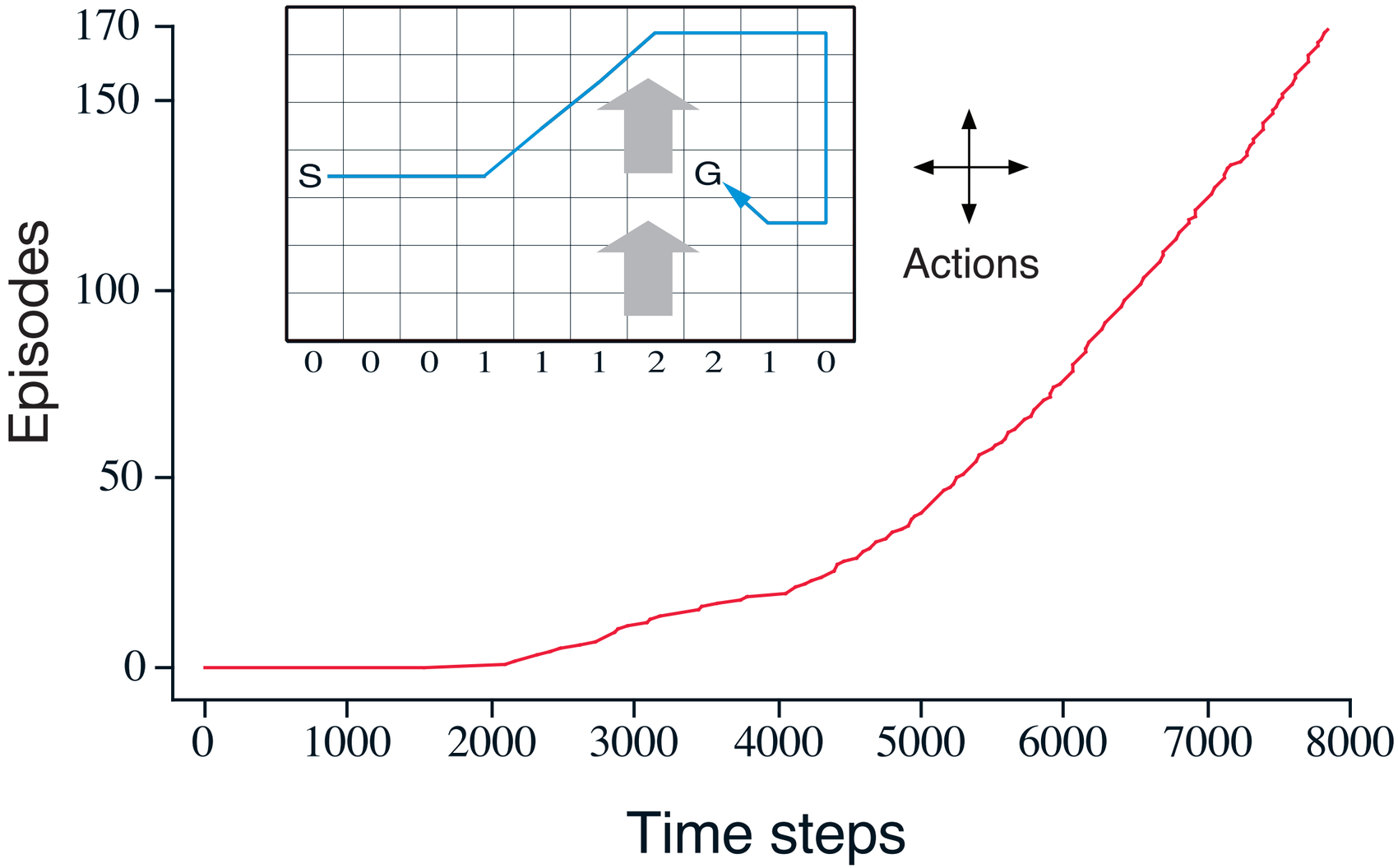


standard moves

wind pushing upward n gridpoints

- Reward $= -1$ per time step until goal is reached
- undiscounted: $\gamma = 1$

SARSA on the Windy Gridworld



LUND UNIVERSITY

What we did today

- First real RL approach for finding an optimised policy
- MC is *model-free*: no knowledge of the MDP transitions / rewards are needed
- Can apply MC only to *episodic* MDPs: all episodes must end.
- based on observations only
- policy optimisation done via the state action value Q
- ϵ-greedy approach for finding the best policy while exploring at the same time.
- GLIE Monte Carlo Control allows to slowly reduce ϵ to 0 while assuring convergence (for proof, see Sect 5 of the book)
- Exploit the Bellman Equation for TD-Learning and SARSA.