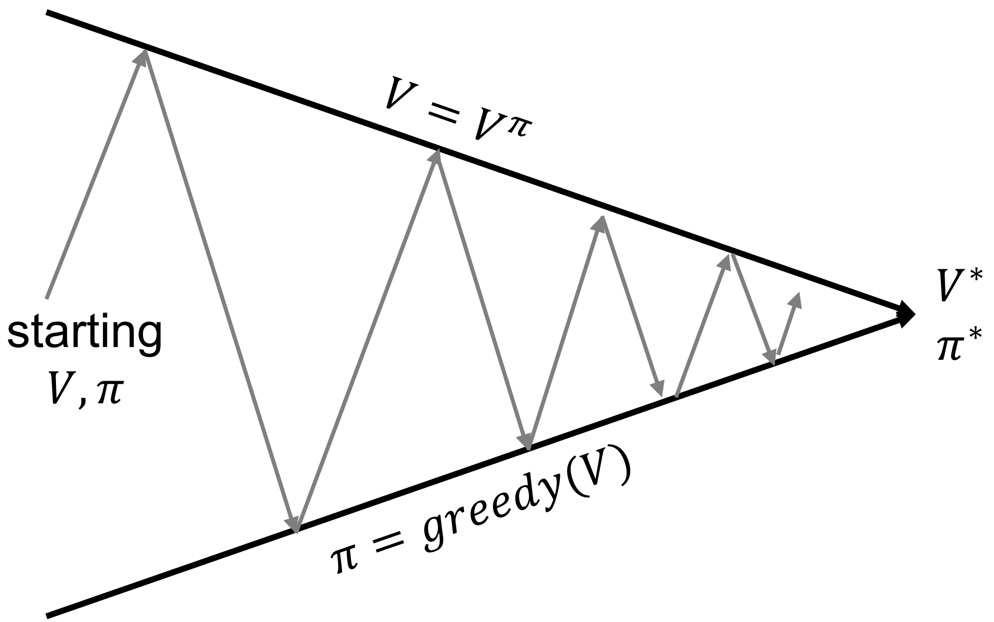
Summary on Model-free Methods

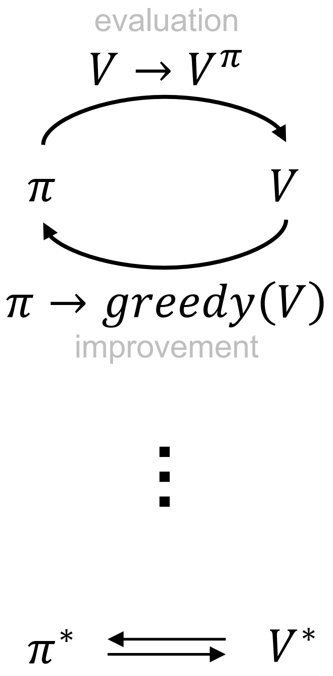
Christopher Mutschler

Recap: Dynamic Programming

- Dynamic Programming (DP) methods to find optimal controllers
  - DP methods are guaranteed to find optimal solutions for Q and V in polynomial time (in number of states and actions) and are exponentially faster than direct search
  - Policy Iteration computes the value function under a given policy to improve the policy while value iteration directly works on the states
    - Perform sweeps through the state set
    - Implement the Bellman equation update
    - Use bootstrapping
  - Require complete and accurate model of the environment
  - Have limited applicability in practice…
    - …as they need to know the dynamics of the environment!

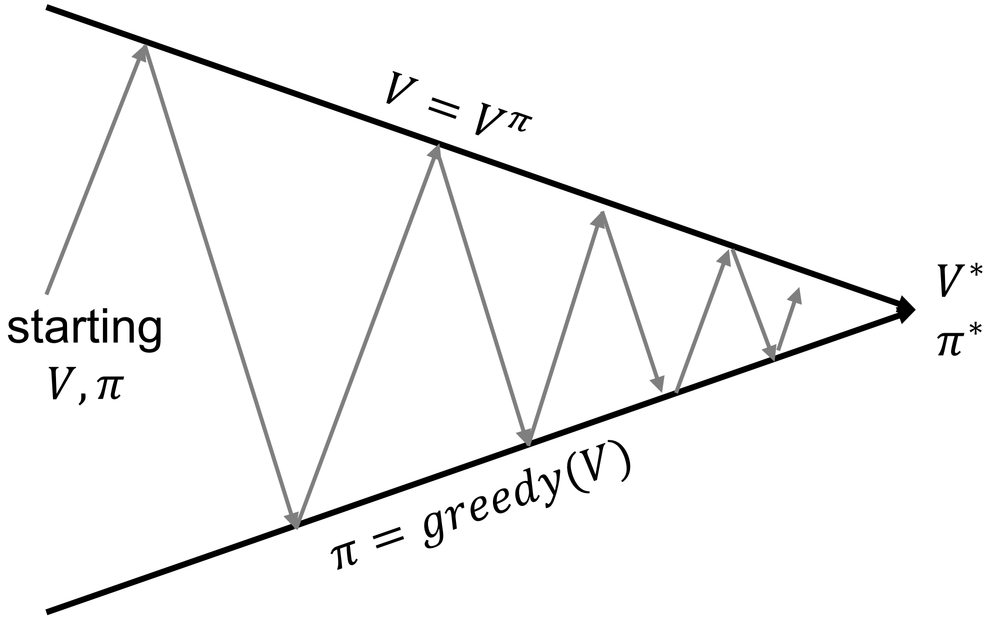
Recap: GPI (Generalized Policy Iteration)





- Policy evaluation: Estimate v_{π}
  - **Any** policy evaluation algorithm
- Policy improvement: Generate $\pi' \geq \pi$
  - **Any** policy improvement algorithm

Recap: GPI w/ Dynamic Programming



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

Recap: Monte Carlo and TD Methods

- So far: We know our MDP model $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.
  - Planning by using dynamic programming
  - Solve a known MDP
- What if we don't know the model, i.e., $\mathcal{P}$ or $\mathcal{R}$ or both?
  - Assume that nobody tells us how the environment works
  - The agent has to find out by itself how to behave optimally
- We distinguish between 2 problems for unknown MDPs:
  - Model-free Prediction: Evaluate the future, given the policy π.
    (estimate the value function)
  - Model-free Control: Optimize the future by finding the best policy π.
    (optimize the value function)

Recap: Monte Carlo and TD Methods

- TD(0) vs. MC Policy Evaluation
  - Goal: learn value function v_π online from experience when we follow policy π

- Update $V(s)$ incrementally after each episode.
- For each state s with **actual return $\boldsymbol{G}$**:

$$N(s) \leftarrow N(s) + 1$$ (just increment visit counter)

$$V(s) \leftarrow V(s) + \frac{1}{N(s)} \left(\boldsymbol{G} - V(s)\right)$$ (update a bit → reduce error)

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

$$V(s) \leftarrow V(s) + \alpha\left(\boldsymbol{G} - V(s)\right).$$

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate]$$

Recap: Monte Carlo and TD Methods

- We do not need to way until the end of the episode to learn something!
- Motivation from real-life applications:
  1. You play chess and perform actions that end up in a bad situation (“bad” is your estimate!); fire-alarm goes on and you will never find out if you would have received a positive reward
  2. You are driving a car and from other policies you know that certain episodes led to death; if you are encountering a state that is close to that one you may have an estimate that currently you are doing not so well
  3. You look into the oven and see a burned cake. You do not really need to taste it…
  4. Riding home: you encounter different places and you can learn from single steps

- Humans usually do TD-Learning in practice!
  - Sometimes we call it intuition.
  - …and clearly there are also some ideas of transfer learning in it ☺

Recap: Monte Carlo and TD Methods

- TD(0) vs. MC Policy Evaluation
  - Goal: learn value function v_π online from experience when we follow policy π

- Simplest TD learning algorithm: TD(0)
- Update value **towards estimation $\hat{G}$**:

$$V(s) \leftarrow V(s) + \alpha(\hat{\boldsymbol{G}} - V(s))$$
$$\hat{\boldsymbol{G}} = \boldsymbol{r} + \boldsymbol{\gamma V(s')}$$ (estimated return)

- $\hat{G}$ is called the TD target
- $\hat{G} - V(s)$ is called the TD error.

- Update $V(s)$ incrementally after each episode.
- For each state s with **actual return G**:

$$N(s) \leftarrow N(s) + 1$$ (just increment visit counter)
$$V(s) \leftarrow V(s) + \frac{1}{N(s)}\left(\boldsymbol{G} - V(s)\right)$$ (update a bit → reduce error)

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

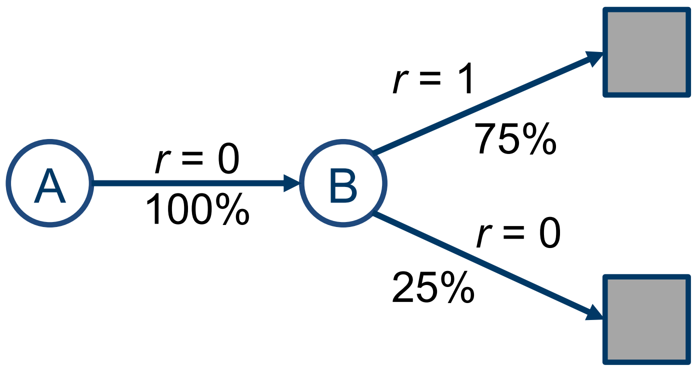
$$V(s) \leftarrow V(s) + \alpha\left(\boldsymbol{G} - V(s)\right).$$

$$\underbrace{\qquad\qquad}_{NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate]}$$

Recap: Monte Carlo and TD Methods

- Which one should I use? Does it make any difference?
- Example: You are the predictor!
  - Two states A, B; no discounting; 8 episodes of experience; keep iterating

```
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0
```



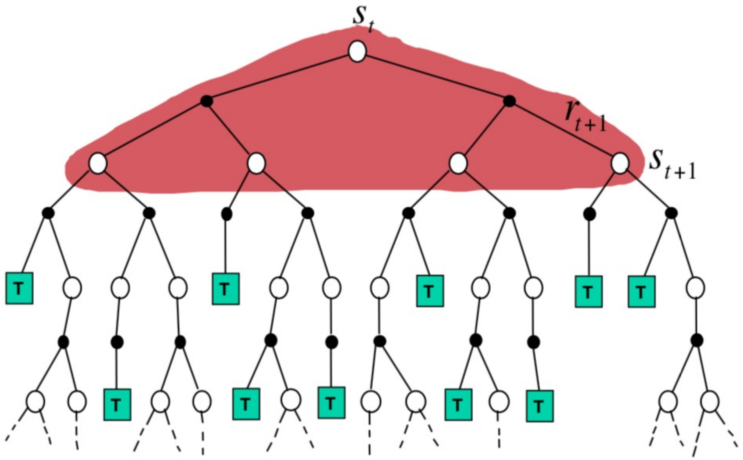
Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

- What is $V(S = A)$ and $V(S = B)$?
  - MC: $V(A) = 0$ $V(B) = 0.75$
  - TD: $V(A) = 0.75$ $V(B) = 0.75$

Recap: DP vs. MC vs. TD

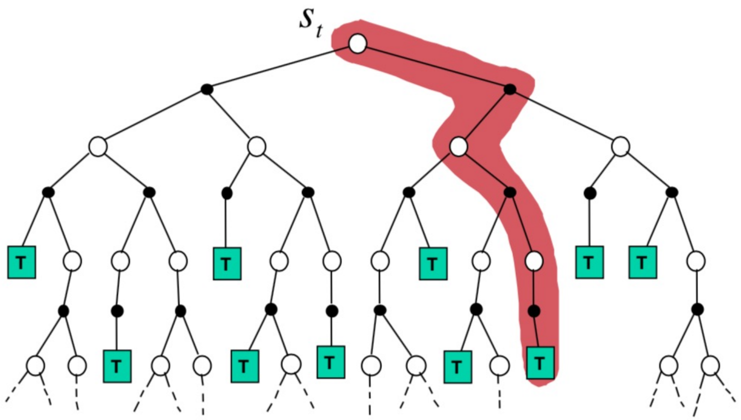
DP Backup

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



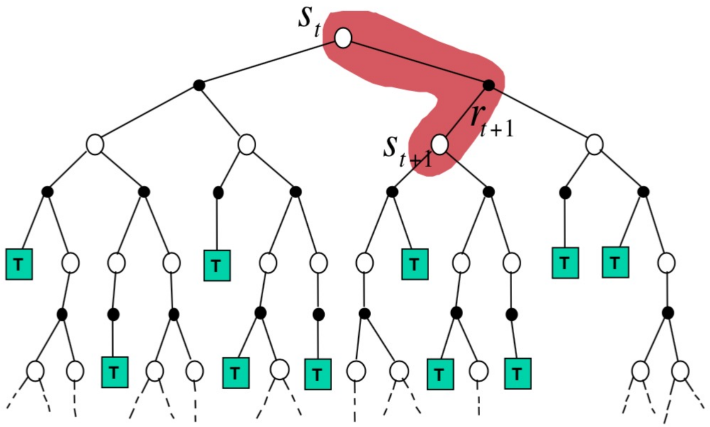
MC Backup

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



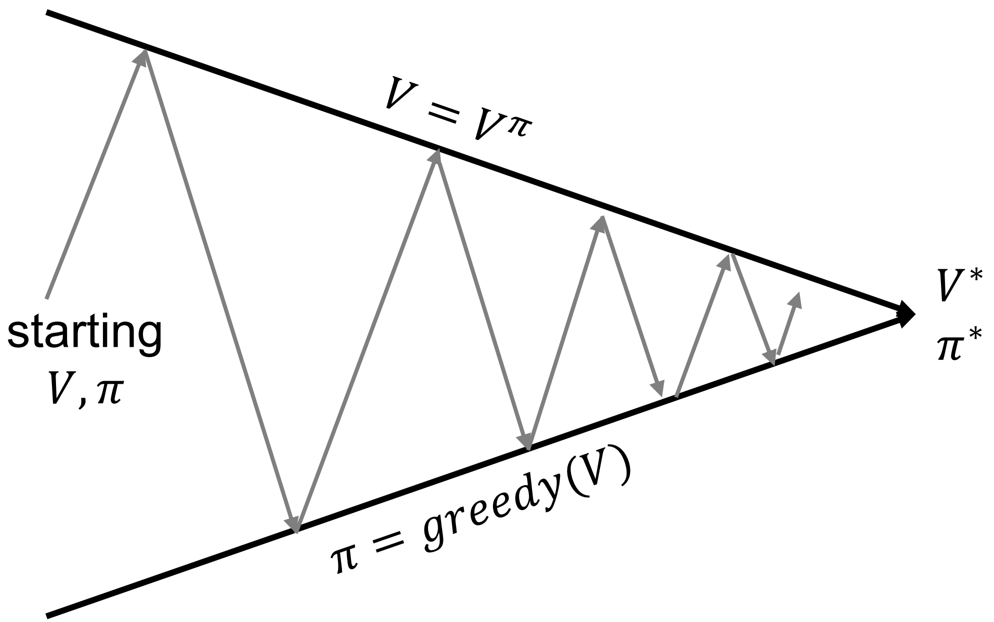
TD Backup

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



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Recap: GPI w/ Monte-Carlo Evaluation



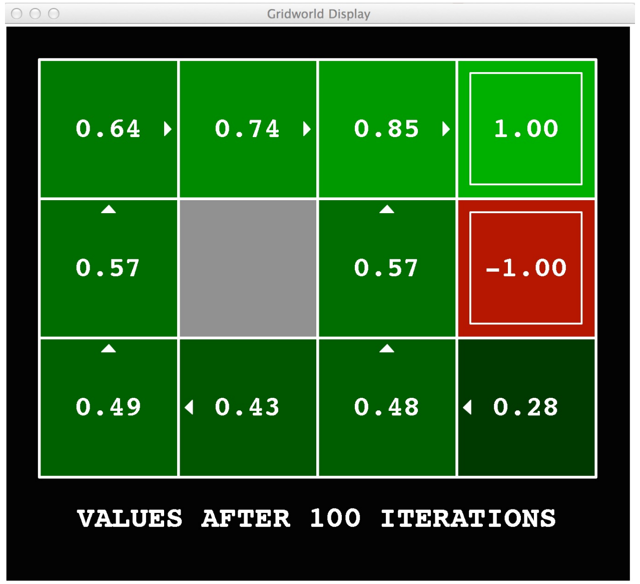
- Policy evaluation: Estimate v_π
  - Monte-Carlo policy evaluation, $V = V^{\pi}$?
- Policy improvement: Generate $\pi' \geq \pi$
  - Greedy Policy Improvement? → **Problem #1: Requires a model!**

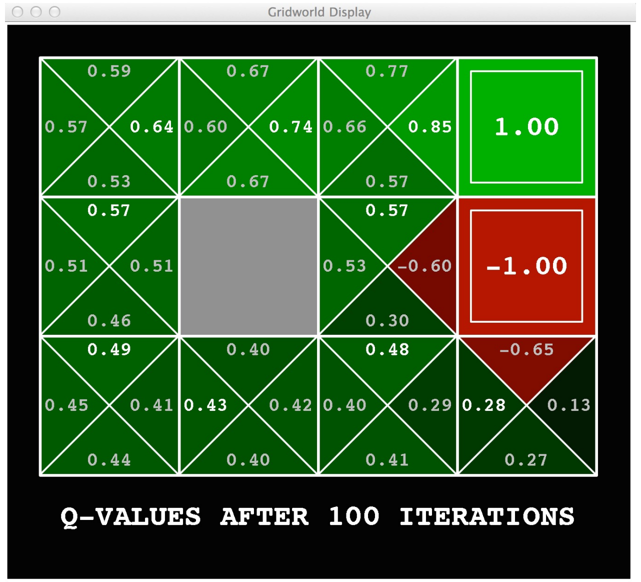
State-Value- vs. Action-Value-Function

- noise $= \ 0.2$, $\gamma = 0.9$, $r = 0$ per time-step; $r = +1$ @ [4,3], $r = -1$ @ [4,2]

$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}} \left\{ \underbrace{\mathcal{R}(s,a)}_{\text{first step}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)\, V^{\pi^*}(s')}_{\text{subsequent steps}} \right\}$$

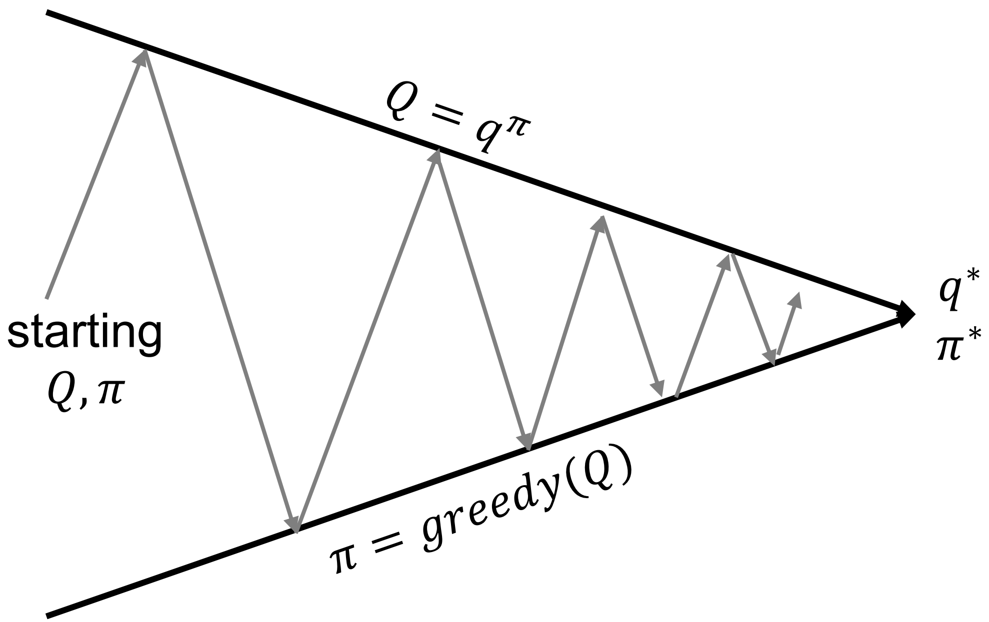
$$Q^{\pi^*}(s,a) = \underbrace{\mathcal{R}(s,a)}_{\text{first step}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} Q^{\pi^*}(s',a')}_{\text{subsequent steps}}$$





http://ai.berkeley.edu/reinforcement.html

Recap: GPI w/ Action-Value Functions (MC)



- Policy evaluation: Estimate v_π
  - Monte-Carlo policy evaluation, $Q = q_\pi$
- Policy improvement: Generate $\pi' \geq \pi$
  - Greedy Policy Improvement? → **Problem #2: We only estimate v‘s following π!**

Recap: GPI w/ Action-Value Functions (MC)

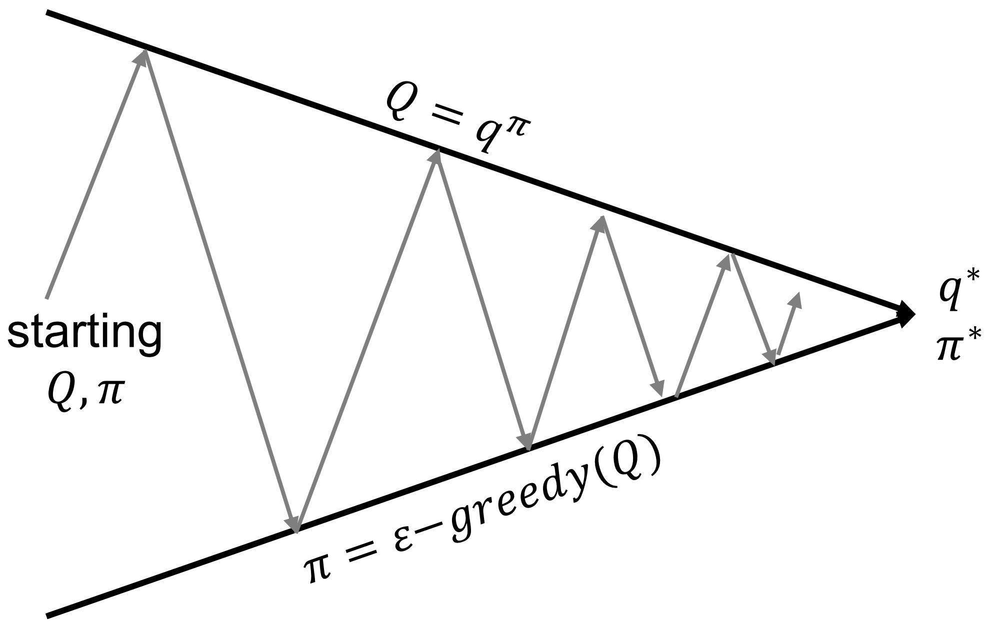
- Exploration vs. Exploitation
- Example: There are two doors in front of you
  - You open the left door and get reward 0
    V(left) = 0, V(right) = NaN
  - You open the right door and get reward +1
    V(left) = 0, V(right) = +1
  - You open the right door and get reward +3
    V(left) = 0, V(right) = +2
  - You open the right door and get reward +2
    V(left) = 0, V(right) = +2

→ Are you sure that this is the best door?

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

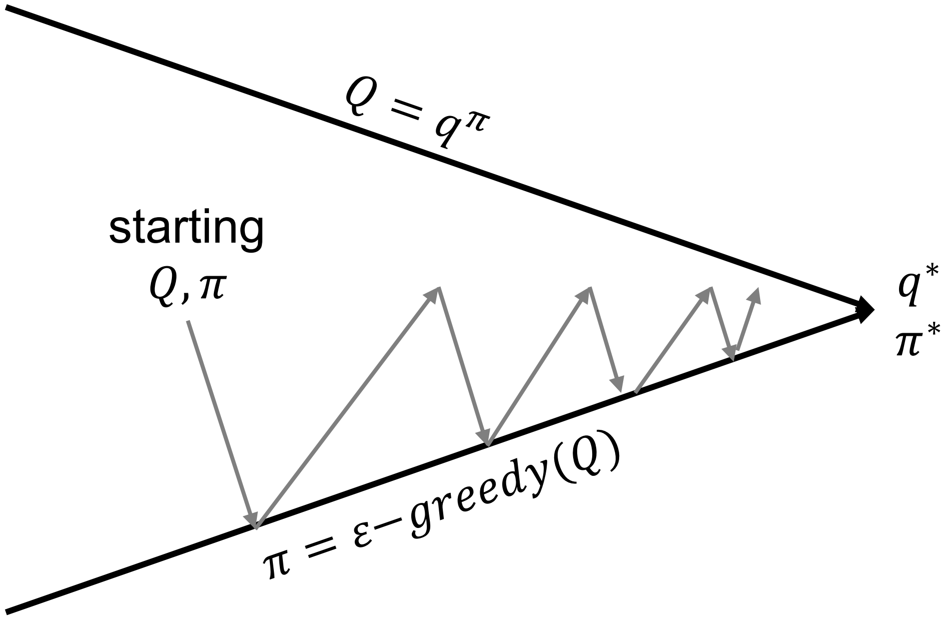
David Silver: Lectures on Reinforcement Learning. UCL Course on RL. 2015.

Recap: GPI w/ Action-Value Functions (MC)



- Policy evaluation: Estimate v_π
  - Monte-Carlo policy evaluation, $Q = q_\pi$
- Policy improvement: Generate $\pi' \geq \pi$
  - **ε-greedy** policy improvement

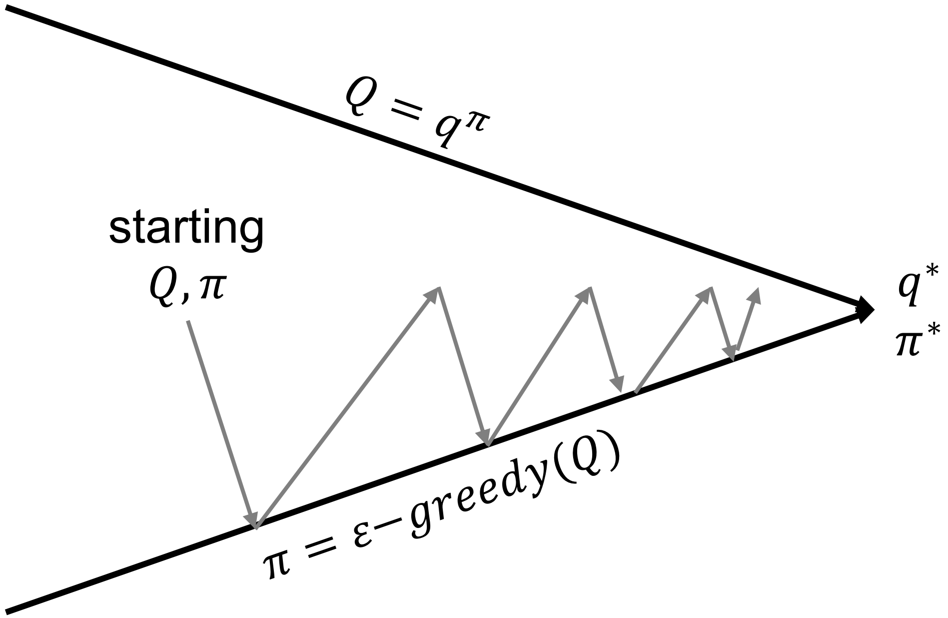
Recap: GPI w/ Action-Value Functions (MC)



- Policy evaluation: Estimate v_{π}
  - Monte-Carlo policy evaluation, $Q \approx q_{\pi}$
- Policy improvement: Generate $\pi' \geq \pi$
  - ε-greedy policy improvement

Every episode

Recap: GPI w/ Action-Value Functions (TD)

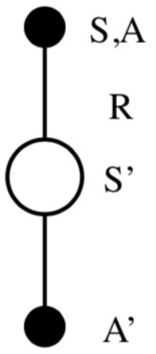


- Policy evaluation: Estimate v_π
  - SARSA, $Q \approx q_\pi$
- Policy improvement: Generate $\pi' \geq \pi$
  - ε-greedy policy improvement

Every time step

Q-Learning and SARSA Algorithms

- SARSA algorithm (on-policy control)
  - Apply TD to $Q(s, a)$
  - Use ε-greedy policy improvement
  - Update at every time-step



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

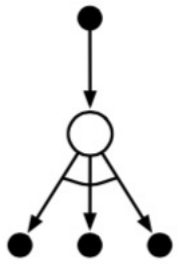
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop 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

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Q-Learning and SARSA Algorithms

- Q-Learning algorithm (off-policy control)
  - Evaluate one policy while following another
  - Can re-use experience gathered from old policies



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

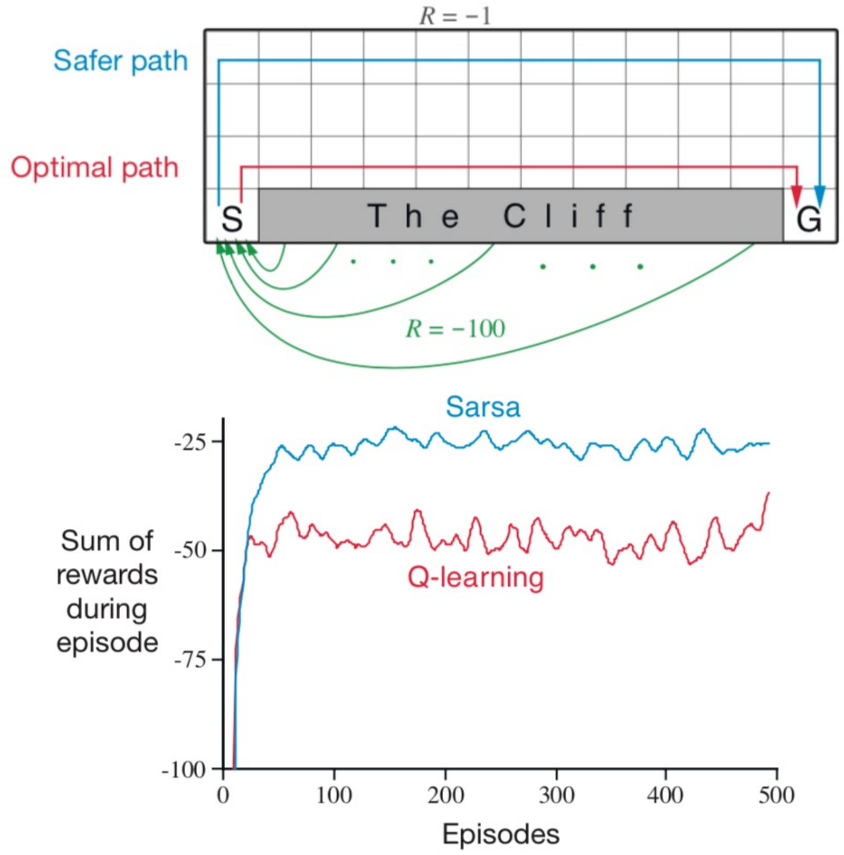
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Recap: Q-Learning and SARSA Algorithms

- Example: Cliff Walking
  - Every transition has reward of -1, falling off the cliff gives a reward of -100 and ends the episode
  - No discounting
  - Assume we use ε-greedy (0.1) for SARSA and Q-Learning, no decay.
- SARSA chooses the safe route, because SARSA incorporates the current policy (ε-greedy)
- Q-Learning chooses the optimal path (and falls of the cliff using the ε-greedy)



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

See also: https://medium.com/init27-labs/understanding-q-learning-the-cliff-walking-problem-80198921abbc

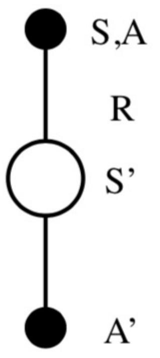
Recap: Q-Learning vs. SARSA

- Q-Learning estimates the return (total discounted future reward) for state-action pairs assuming a greedy policy (although it may follow an explorative policy)
- Instead, SARSA estimates the return for state-action pairs assuming the current policy (that it also follows)
- If the current policy is also a greedy policy, then the distinction disappears.
- SARSA will also get to the Q-Learning result if we decay ε (carefully!)

Recap: Q-Learning vs. SARSA

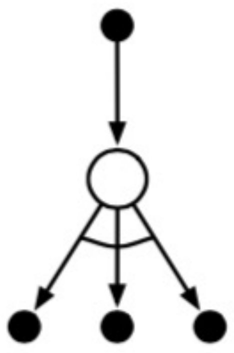
SARSA algorithm (on-policy control)

+ Processes each sample immediately
+ Minimal update cost per sample
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits instabilities under approximate representations
- Poses constraints on sample collection (on-policy)
- Requires careful handling on the policy greediness



Q-Learning algorithm (off-policy control)

+ Processes each sample immediately
+ Minimal update cost per sample
+ Poses no constraints on sample collection (off-policy)
- Requires a huge number of samples
- Requires careful schedule for the learning rate
- Makes minimal use of each sample
- The ordering of samples influences the outcome
- Exhibits (even more) instabilities under approximate representations



Teaser: Alphastar

Challenge:

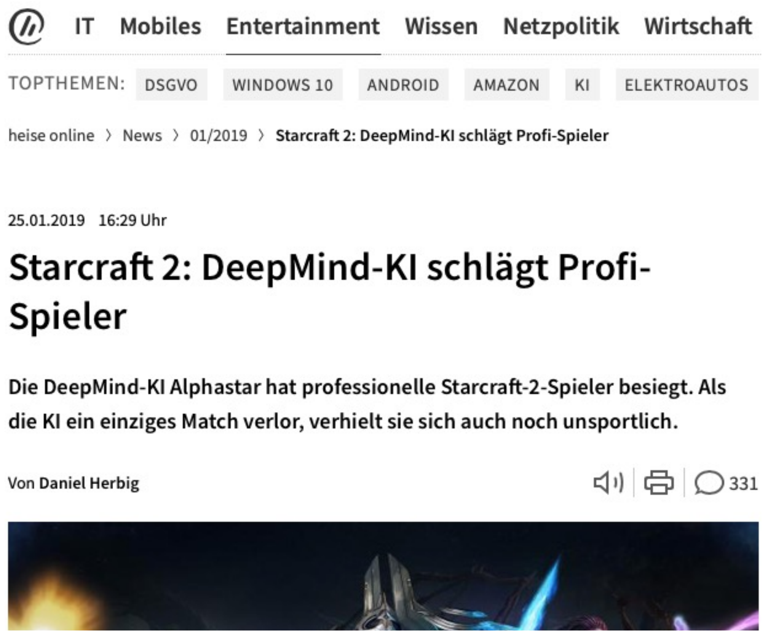
- Game theory: many „good“ strategies
- Imperfect information: crucial information is hidden
- Long-term planning: early actions pay off much later
- Real time: continual to game clock
- Large action space: hierarchical action space

Solution:

- Many nice tricks ☺
- LSTMs, autoregressive policy heads with pointer networks, multi-agent centralized value baselines, …
- It is really about population modelling!

More Info:

https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/


IT Mobiles Entertainment Wissen Netzpolitik Wirtschaft

TOPTHEMEN: DSGVO WINDOWS 10 ANDROID AMAZON KI ELEKTROAUTOS

heise online › News › 01/2019 › Starcraft 2: DeepMind-KI schlägt Profi-Spieler

25.01.2019 16:29 Uhr

Starcraft 2: DeepMind-KI schlägt Profi-Spieler

Die DeepMind-KI Alphastar hat professionelle Starcraft-2-Spieler besiegt. Als die KI ein einziges Match verlor, verhielt sie sich auch noch unsportlich.

Von Daniel Herbig

331