Reinforcement Learning 02

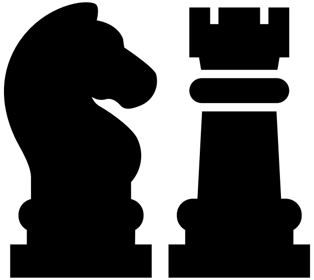
DS Development Group Presentations

Peter Nicholas S. Onglao | MNL

Background

GOAL:

Make an RL algorithm for chess (>2 presentations away) that can beat me



Contents

Sources

- UCL Course on Reinforcement Learning, David Silver
- Reinforcement Learning – An Introduction, Sutton and Barto
- Reinforcement Learning (GitHub repo), Denny Britz

1

Dynamic Programming

When the MDP is known

Dynamic Programming

- Dynamic – sequential/temporal component
- Programming – optimizing a “program” (policy)
- Break down complex problems into subproblems
    - Solve the subproblems
    - Combine solutions to subproblems

Dynamic Programming

- DP is a general solution method for problems with two properties:
  - Optimal substructure
    - *Principle of optimality* applies
    - Optimal solution can be decomposed into subproblems
    - Bellman equation – recursive decomposition
  - Overlapping subproblems
    - Subproblems recur many times
    - Solutions can be cached and reused
    - Value function

Dynamic Programming

- Assumes **full knowledge** of the MDP
- Used for *planning*
- Prediction
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and **policy $\boldsymbol{\pi}$**
  - Output: **value function $\boldsymbol{v_\pi}$**
- Control
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: **optimal value function $\boldsymbol{v_*}$** and **optimal policy $\boldsymbol{\pi_*}$**

Generalized Policy Iteration (GPI)

- Given a policy π:
  - Evaluate the policy π

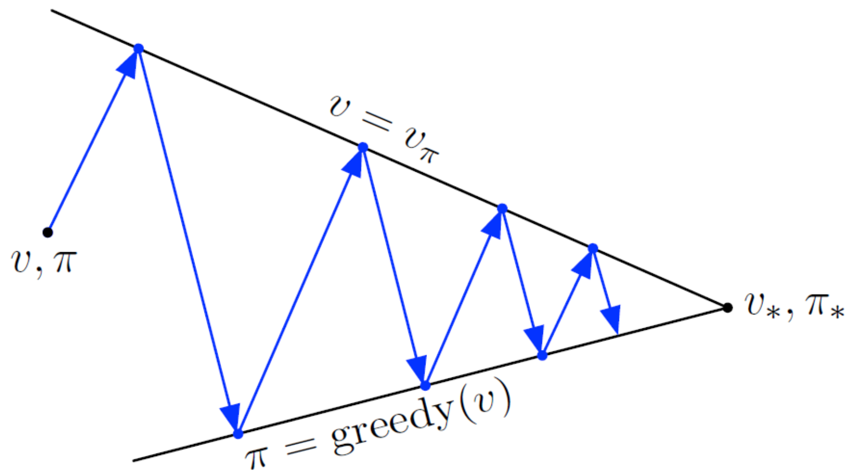
$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

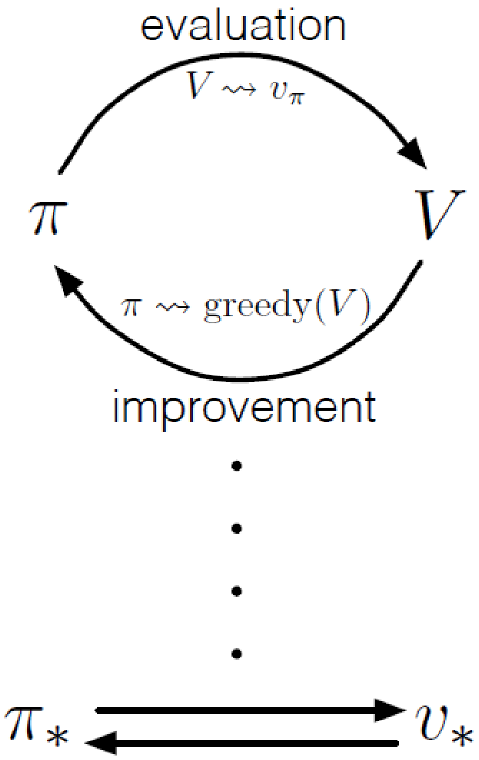
  - Improve the policy by acting greedily w/r v_π

$$\pi' = \text{greedy}(v_\pi)$$

- GPI: let policy-evaluation and policy-improvement interact
- Most RL methods are well described as GPI
- This process of policy iteration always converges to π^*

Generalized Policy Iteration (GPI)





Policy evaluation: Estimate v_π

Iterative policy evaluation

Policy improvement: Generate $\pi' \geq \pi$

Greedy policy improvement

Principle of Optimality

- An optimal policy can be subdivided into two components:
  - An optimal first action A_*
  - Followed by an optimal policy from successor state S'

- Theorem (Principle of Optimality)

  A policy $\pi(a|s)$
  achieves the optimal value from state s, $v_\pi(s) = v_*(s)$,
  iff for any state s' *reachable from* s,
  π *achieves the optimal value from state* s', $v_\pi(s') = v_*(s')$

Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by **one-step lookahead**

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right)$$

- Value iteration: apply these updates iteratively
- Start with final rewards and work backwards

Synchronous DP Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_{pi}(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

Asynchronous Dynamic Programming

- The methods described so far used *synchronous* backups
  - All states are backed up at the same time
- *Asynchronous DP* backs up states individually, in any order
- Reduces computation time
- Guaranteed to converge if all states continue to be selected

2

Monte-Carlo Learning

Learning from **complete** episodes with an **unknown MDP**

Monte-Carlo Learning

- **Model-free** – no need to know MDP transitions/rewards
- Learn directly from **complete** episodes of experience
- Simplest idea: use *empirical mean return* instead of *expected* return
  - $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
  - $V(s) = S(s)/N(s)$
    - $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$
- Drawback: can only be applied to *episodic* MDPs - all episodes must terminate

Monte-Carlo Policy Evaluation

- First-Visit MC Policy Evaluation
  - Update estimate the first time a state s is visited in an episode.
- Every-Visit MC Policy Evaluation
  - Update estimate every time a state s is visited in an episode.

3

Temporal Difference Learning

Learning from **incomplete** episodes
with an **unknown MDP**

Temporal-Difference Learning

- **Model-free** – no need to know MDP transitions/rewards
- Learn directly from **incomplete** episodes of experience by **bootstrapping**
- Update a guess towards a guess

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit MC
  - Update value toward **actual** return G_t

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

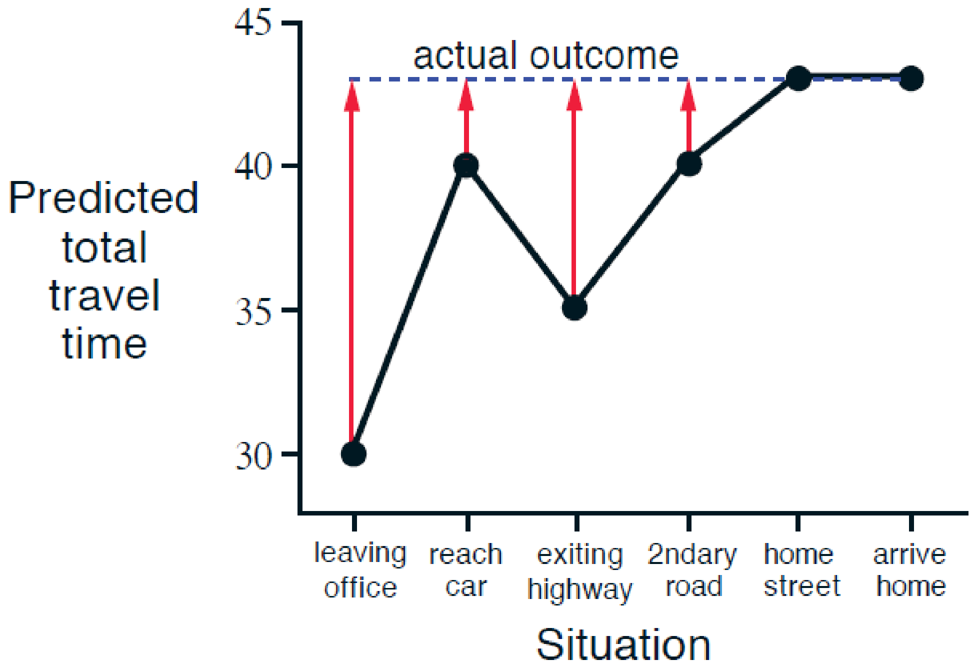
- Simplest TD: TD(0)
  - Update value toward **estimated** return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\textit{TD target}} - V(S_t)}_{\textit{TD error}})$$

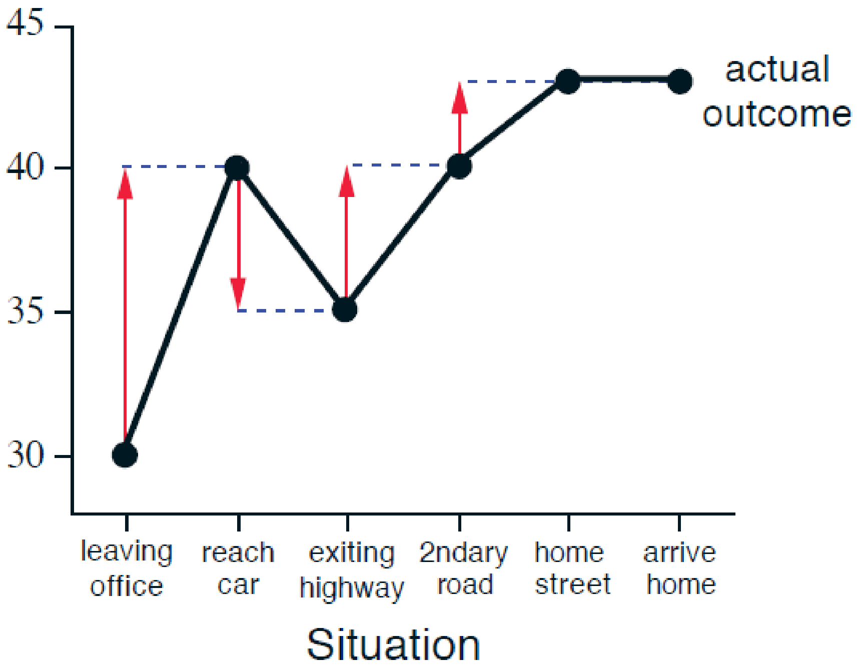
Driving Home Example

Situation/State	Elapsed Time (min)	Predicted Time Left (min)	Predicted Time: Total (min)
Leaving office	0	30	30
Reach car, raining	5	35	40
Exit highway	20	15	35
Behind truck	30	10	40
Home street	40	3	43
Arrive home	43	0	43

Driving Home Example



MC Learning



TD Learning

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
- TD can learn **without** final outcome
  - TD: can learn from incomplete sequences
  - MC: can only learn from complete sequences
  - TD: works in non-terminating environments
  - MC: only works for terminating environments

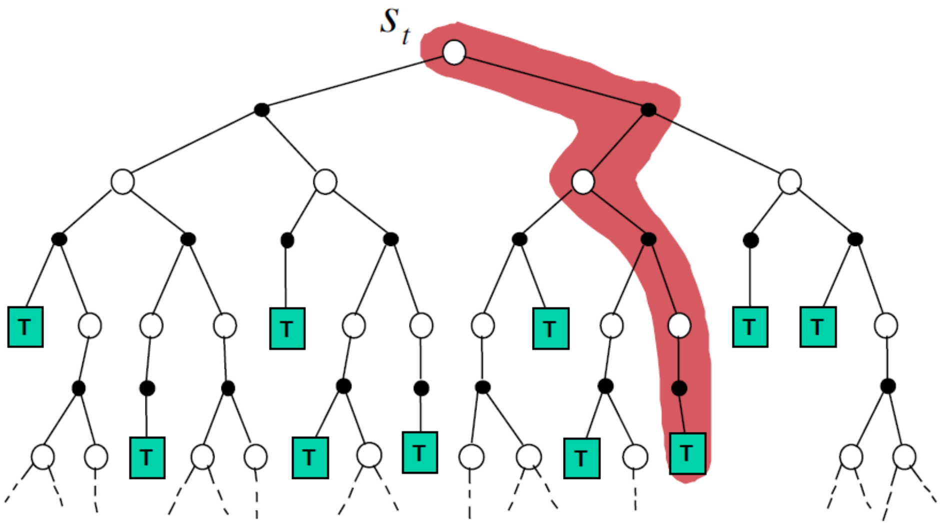
MC vs. TD

- MC has **high variance, zero bias**
  - Good convergence properties
  - Not very sensitive to initial value
  - Very simple to use & understand
  - Does not exploit Markov property
- TD has **low variance, some bias**
  - More efficient than MC
  - TD(0) converges to $v_\pi(s)$
  - More sensitive to initial value
  - Exploits Markov property

Unified View

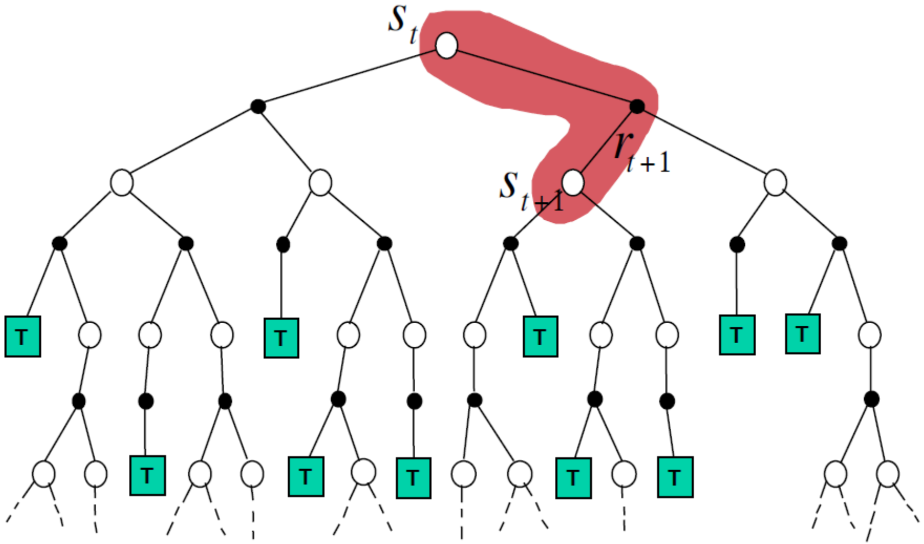
Monte Carlo

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



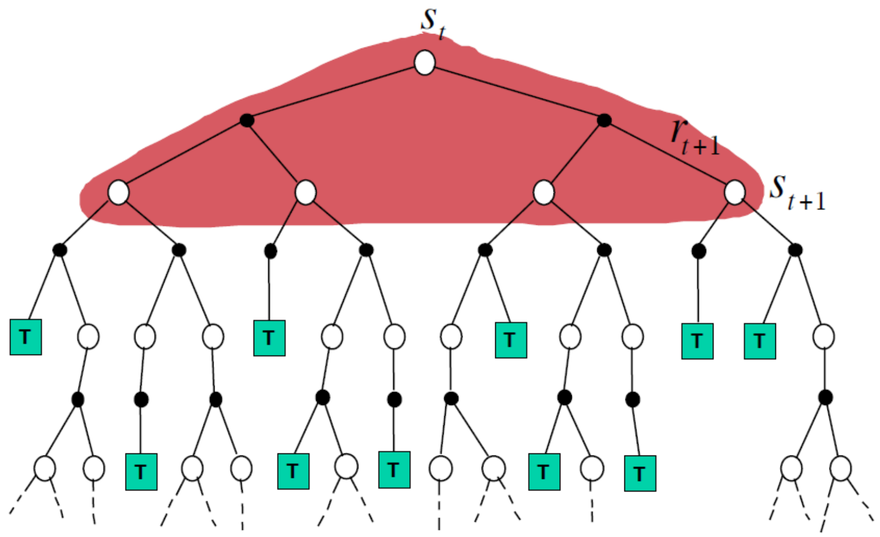
Temporal Difference

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

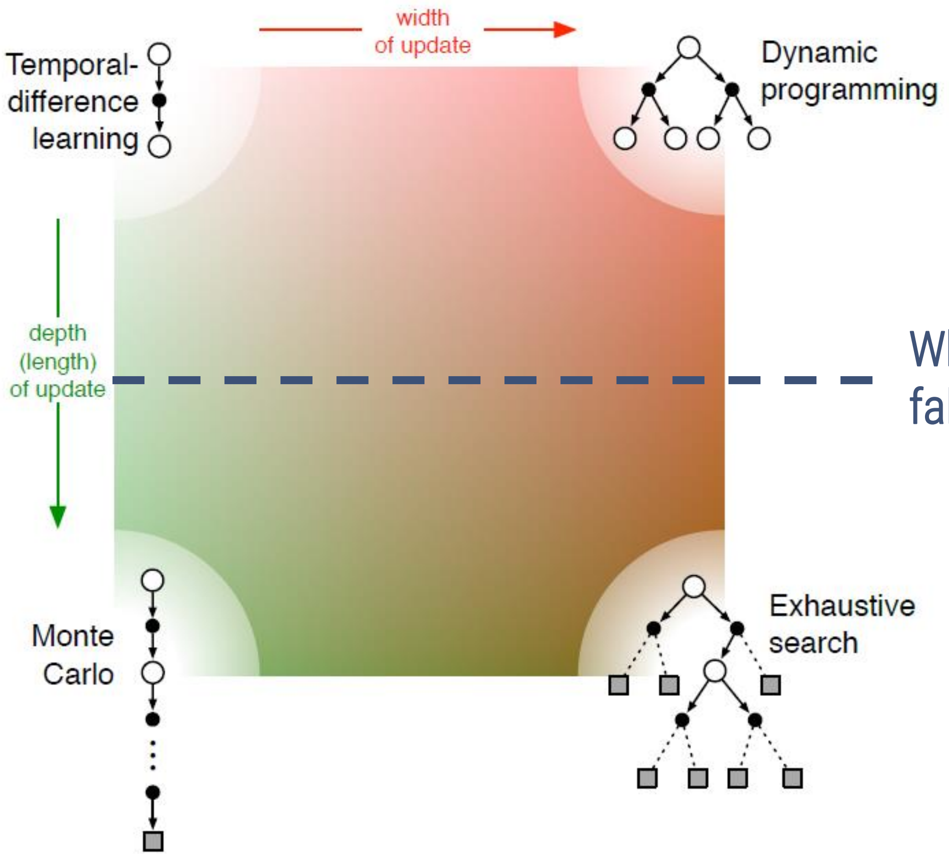


Dynamic Programming

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



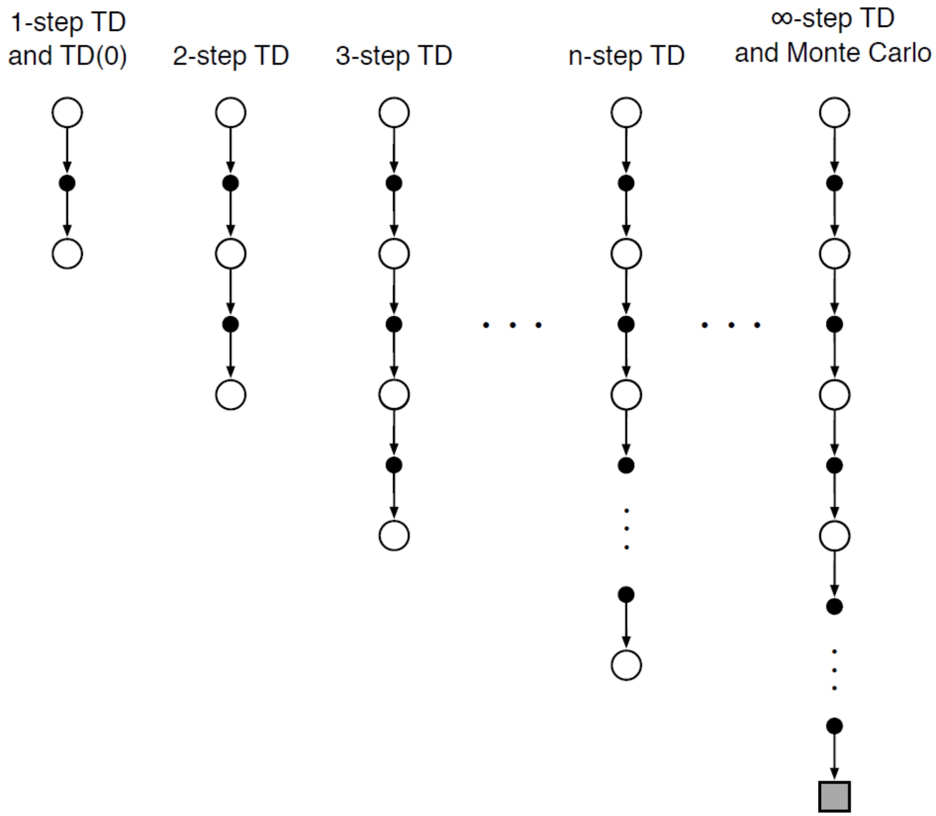
Unified View



What method falls here?

n-step Lookahead

Look n steps into the future



n-step Return

- n-step return

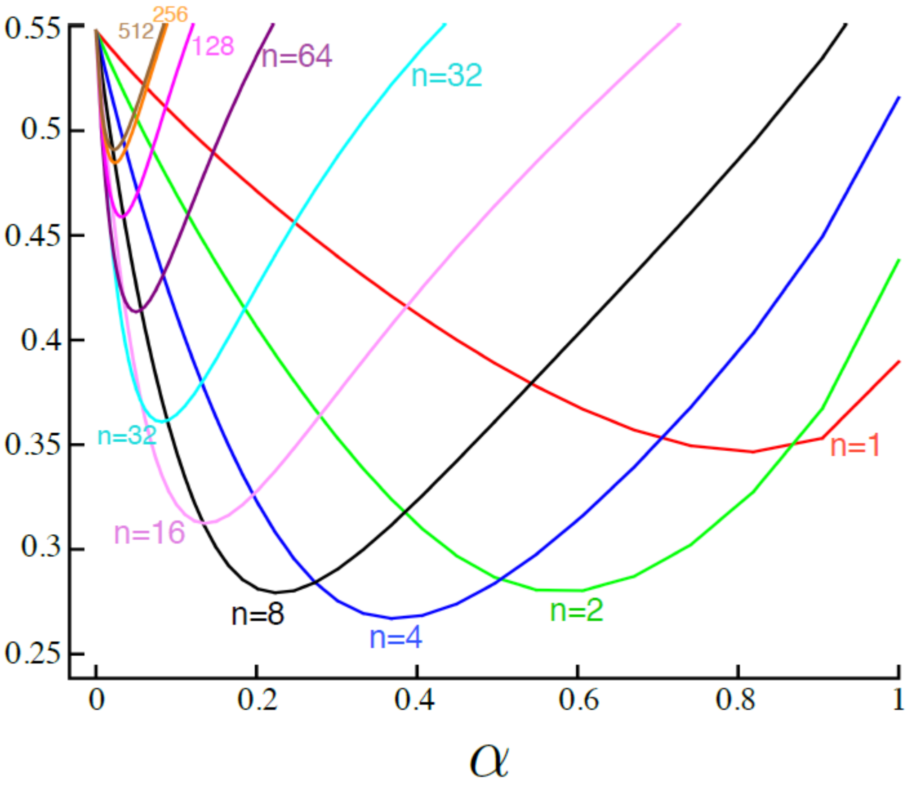
$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step TD learning

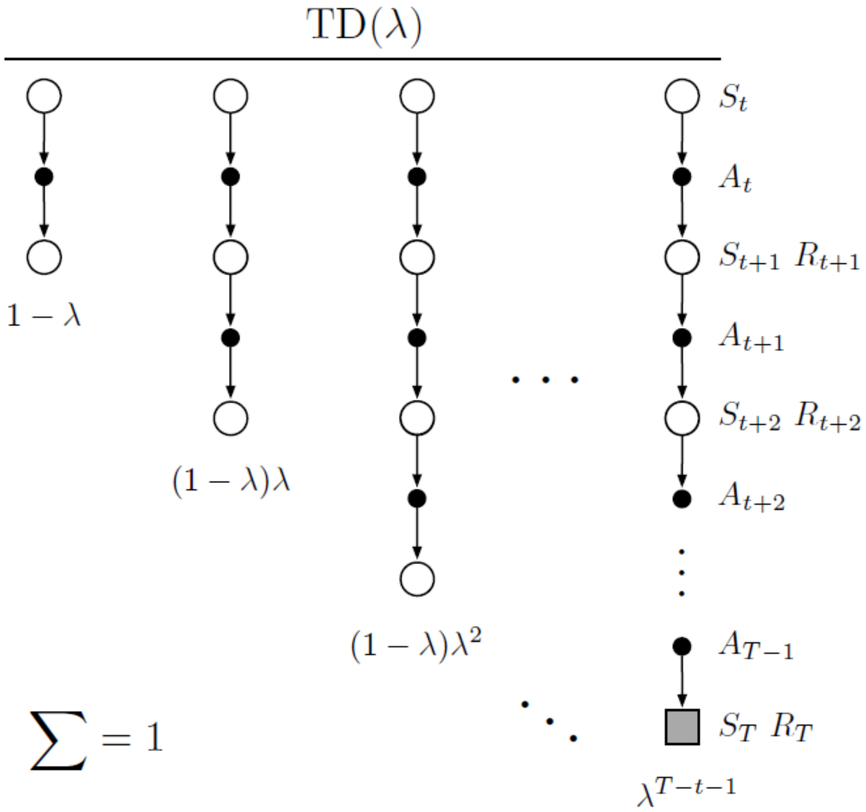
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$$

n-step TD

Average RMS error over 19 states and first 10 episodes



TD(λ)



- Concept: get the best of all worlds
- Average n-step returns over all n, using 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)

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

4

Model Free Control

Optimizing with an unknown MDP

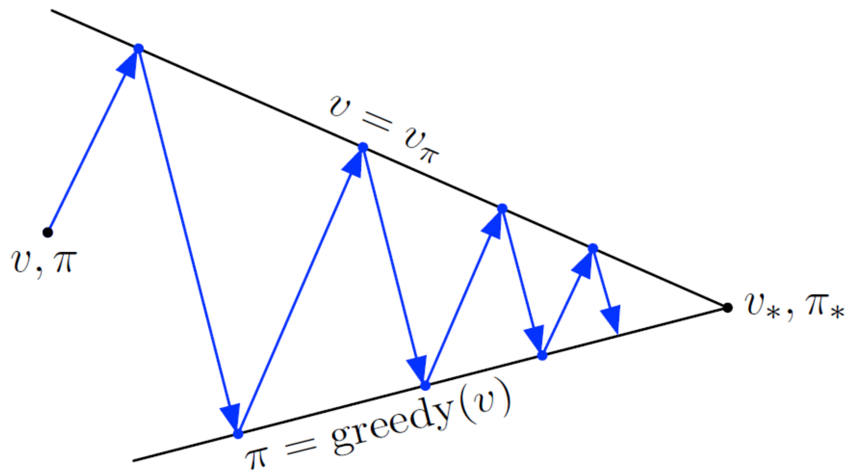
Model-free Reinforcement Learning

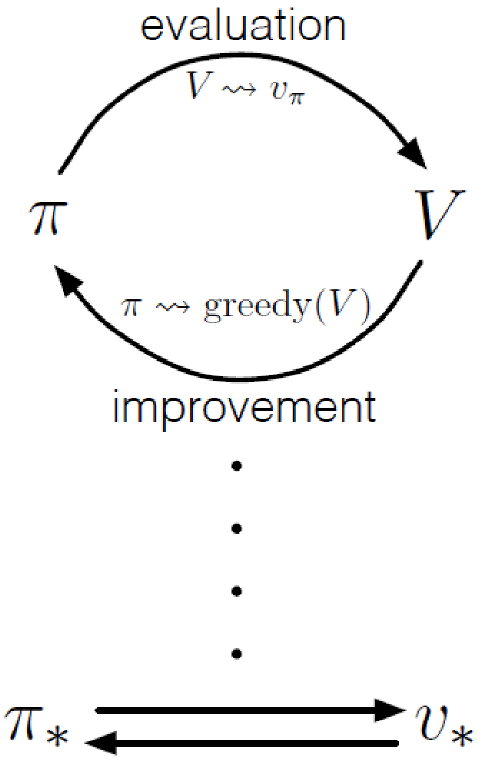
- Model-free prediction
  - **Estimate** the value function of an unknown MDP
- Model-free control
  - **Optimise** the value function of an unknown MDP

On and Off-Policy Learning

- On-policy learning
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - Learn about policy π from experience sampled from μ

Generalized Policy Iteration (GPI)





Policy evaluation: Estimate v_π

Iterative policy evaluation

Policy improvement: Generate $\pi' \geq \pi$

Greedy policy improvement

Model-Free Policy Iteration

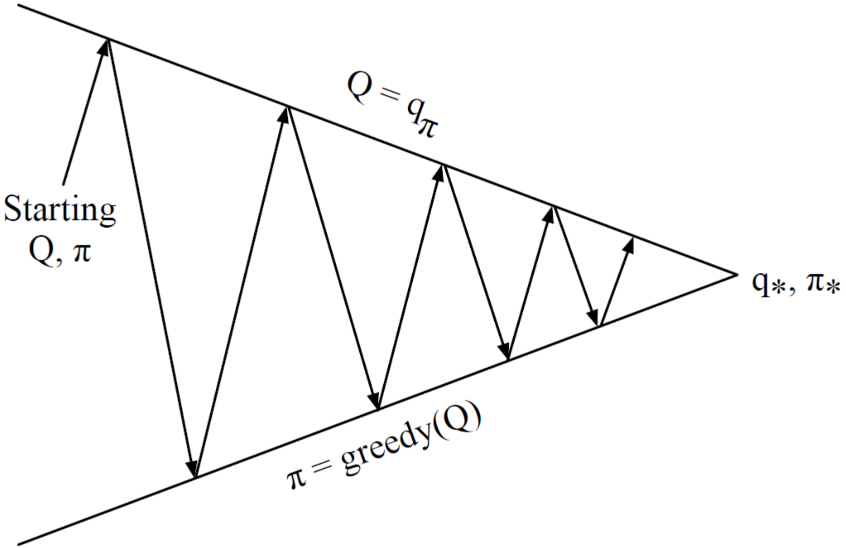
- Greedy policy improvement over $\boldsymbol{V(s)}$ requires model of MDP

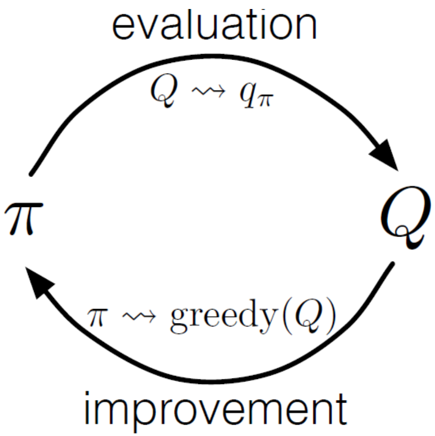
$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left(\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s') \right)$$

- Greedy policy improvement over $\boldsymbol{Q(s,a)}$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a)$$

Generalised Policy Iteration with Q





Policy evaluation: MC policy evaluation, $Q = q_\pi$
Policy improvement: Greedy policy improvement?

Example of Greedy Action Selection

- There are two buttons:
- You press the left and get reward 0

$V(left) = 0$

- You press the right and get reward $+1$

$V(right) = +1$

- You press the right and get reward $+3$

$V(right) = +2$

- You press the right and get reward $+2$

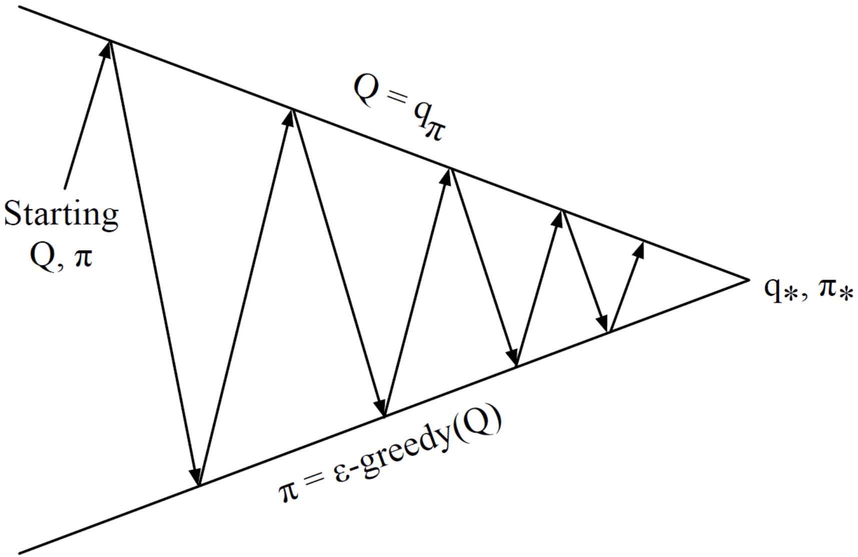
$V(right) = +2$

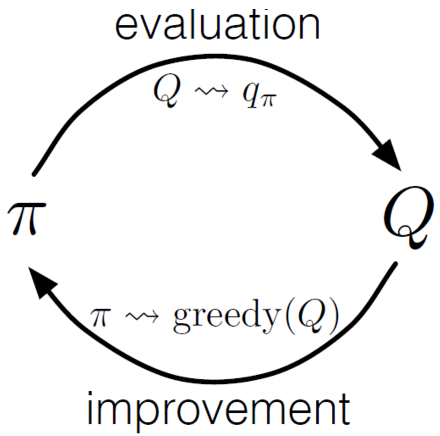
ϵ-Greedy Exploration

- Simplest idea for ensuring **continual exploration**
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$, choose the greedy action
- With probability ϵ, choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/\mathrm{m} + 1 - \epsilon, & \text{if } \mathrm{a}^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

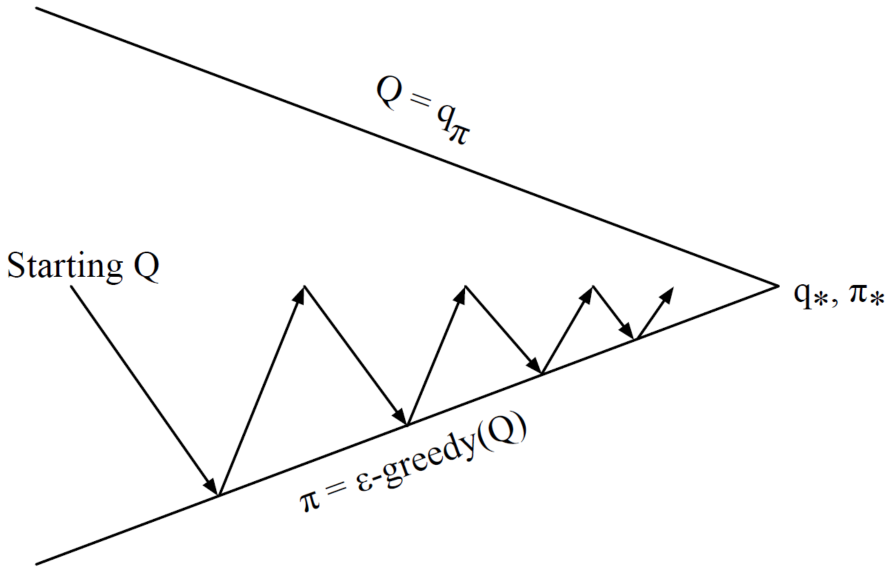
Monte-Carlo Policy Iteration





Policy evaluation: MC policy evaluation, $Q = q_\pi$
Policy improvement: ϵ-greedy policy improvement
Note: we do this over multiple episodes

Monte-Carlo Control

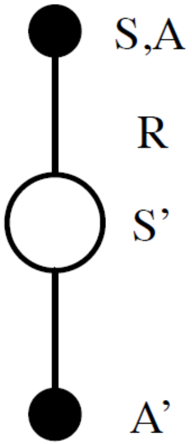


Every episode (update with ‘fresh’ data):
Policy evaluation: MC policy evaluation, $Q \approx q_\pi$
Policy improvement: ϵ-greedy policy improvement

MC vs. TD Control

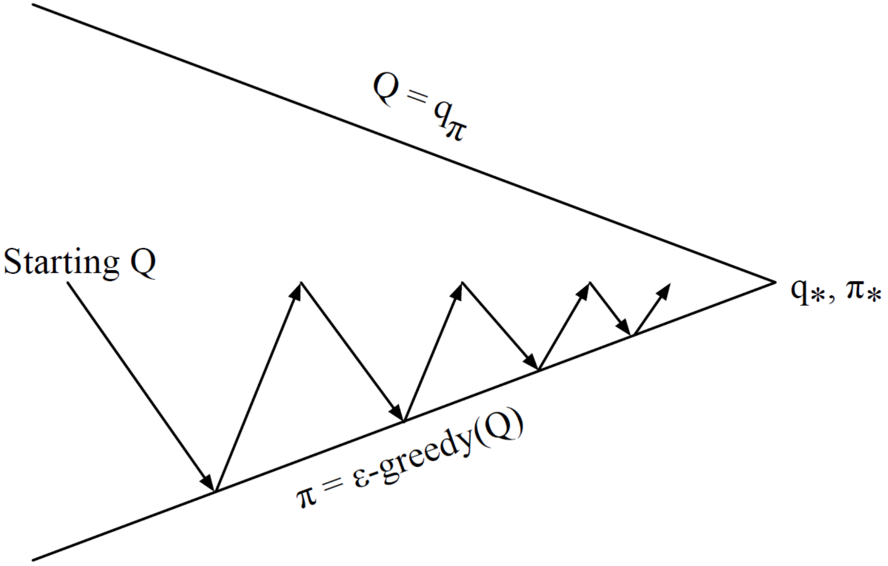
- TD has several advantages over MC:
  - Lower variance
  - Online
  - Incomplete sequences
- Use TD instead of MC in our control loop
  - Apply TD to $Q(\mathcal{S}, \mathcal{A})$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Updating Q with SARSA



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

On-Policy Control with Sarsa

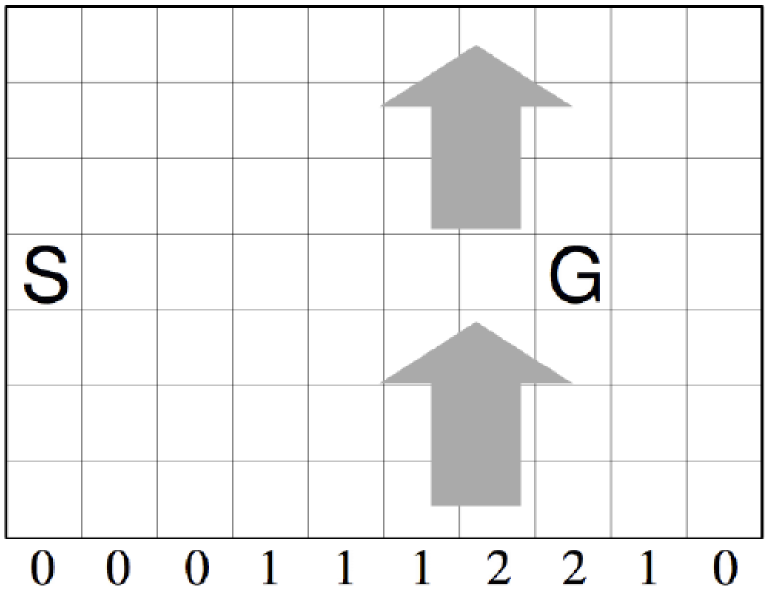


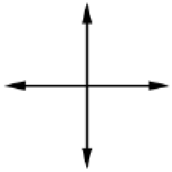
Every time-step:

Policy evaluation: Sarsa, $Q \approx q_\pi$

Policy improvement: ϵ-greedy policy improvement

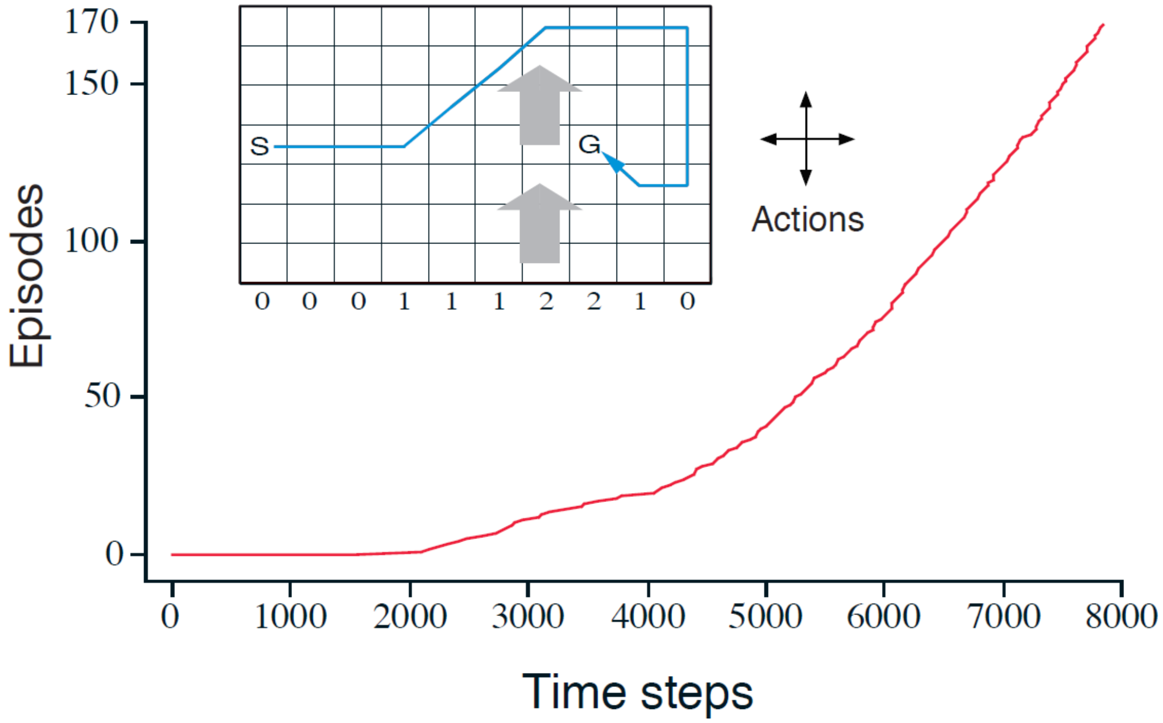
Windy Gridworld Example



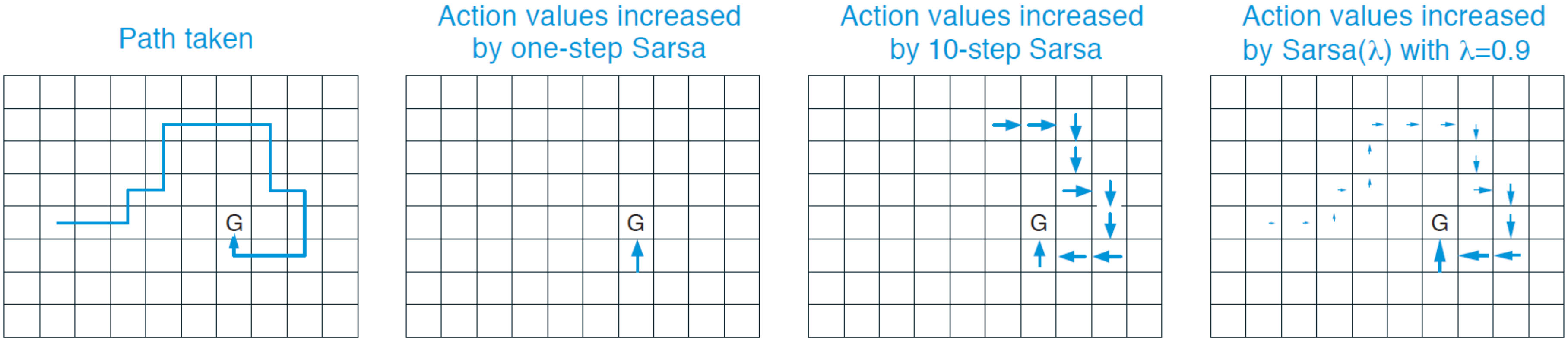


- Reward = -1 per time step, until goal is reached
- Undiscounted

Sarsa on the Windy Gridworld



SARSA(λ)



Path taken
Action values increased by one-step Sarsa
Action values increased by 10-step Sarsa
Action values increased by Sarsa(λ) with λ=0.9
G
G
G
G

Q-Learning

- We now consider off-policy learning of action-values $Q(s,a)$
- No importance sampling is required
- Next action is chosen using behavior policy $A_{t+1} \sim \mu(\cdot \mid S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot \mid S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

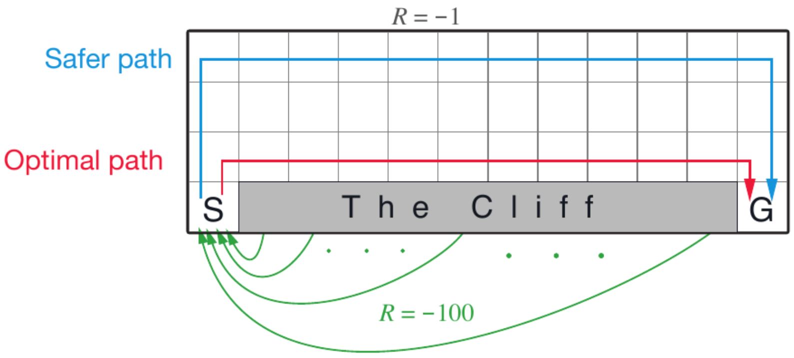
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\big)$$

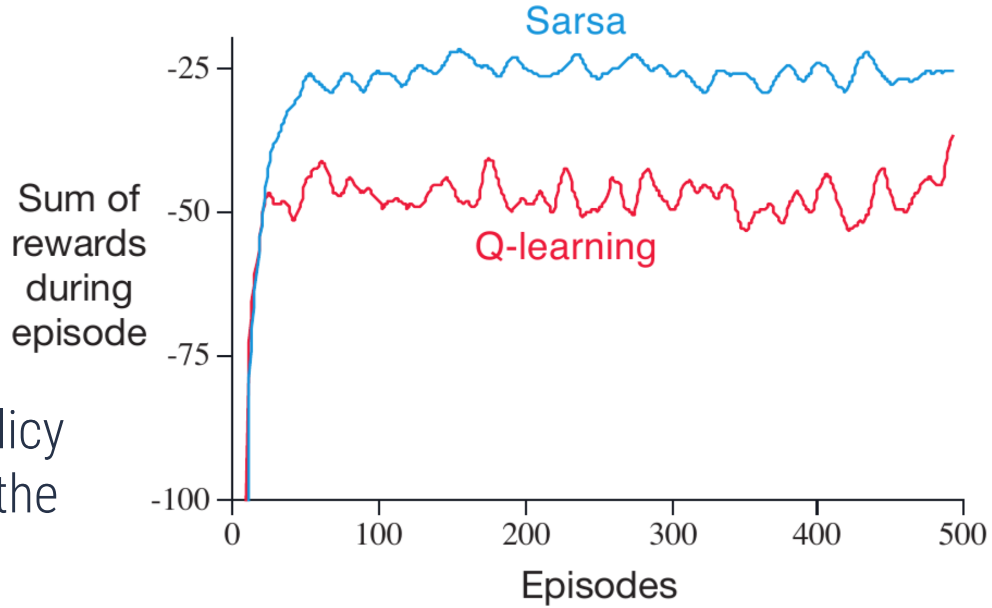
Off-Policy Control with Q-Learning

- We now allow both behavior and target policies to improve
- The target policy π is greedy w/r $Q(s,a)$
- The behavior policy μ is ϵ-greedy w/r $Q(s,a)$
- Q-learning target simplifies as:

$$
\begin{aligned}
& R_{t+1} + \gamma Q(S_{t+1}, A') \\
= & R_{t+1} + \gamma Q\left(S_{t+1}, \operatorname*{argmax}_{a'} Q(S_{t+1}, a')\right) \\
= & R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')
\end{aligned}
$$

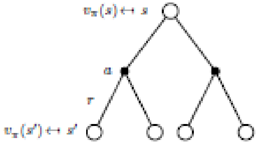
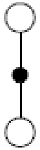
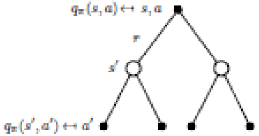
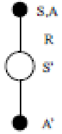
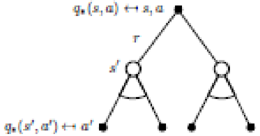
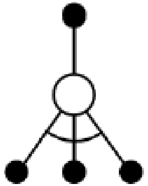
Q-Learning vs Sarsa





- Sarsa: on-policy, Q-Learning: off-policy
- We use Sarsa when we care about the agent's performance while learning (expensive robot)
- We use Q-learning when we don't mind the agent 'suffering'

Relationship Between DP and TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

5

Summary and Code

Examples of algorithms applied via Python

Summary

- Dynamic Programming (DP)
  - Assumes perfect knowledge of the MDP
- Monte Carlo (MC) and Temporal Difference (TD) Learning
  - When the MDP is unknown
  - MC and TD both learn directly from experience
  - MC uses full episodes
  - TD uses bootstrapping
    - SARSA: On-Policy TD Control
    - Q-Learning: Off-Policy TD Control

Dynamic Programming Example

- Gambler’s Problem (from Sutton and Barto); Goal: reach exactly $100

Step 1: BET

Step 2: TOSS COIN

Heads → Gain the amount bet

Tails → Lose the amount bet

Step 3: WIN/LOSE

Dynamic Programming Example

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

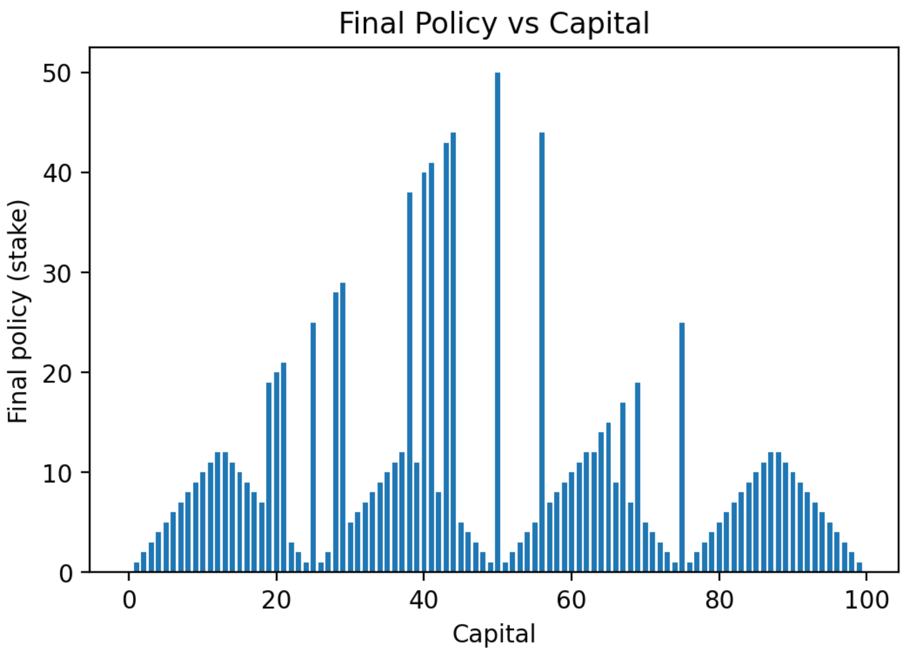
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

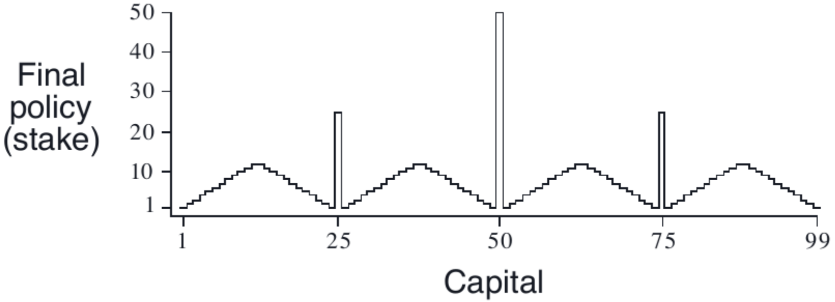
```python
while True:
    # Stopping condition
    delta = 0
    # Update each state...
    for s in range(1, 100):
        # Do a one-step lookahead to find the best action
        A = one_step_lookahead(s, V, rewards)
        best_action_value = np.max(A)
        # Calculate delta across all states seen so far
        delta = max(delta, np.abs(best_action_value - V[s]))
        # Update the value function. Ref: Sutton book eq. 4.10.
        V[s] = best_action_value
    # Check if we can stop
    if delta < theta:
        break
```

```python
# Create a deterministic policy using the optimal value function
policy = np.zeros(100)
for s in range(1, 100):
    # One step lookahead to find the best action for this state
    A = one_step_lookahead(s, V, rewards)
    best_action = np.argmax(A)
    # Always take the best action
    policy[s] = best_action
```

Dynamic Programming Example

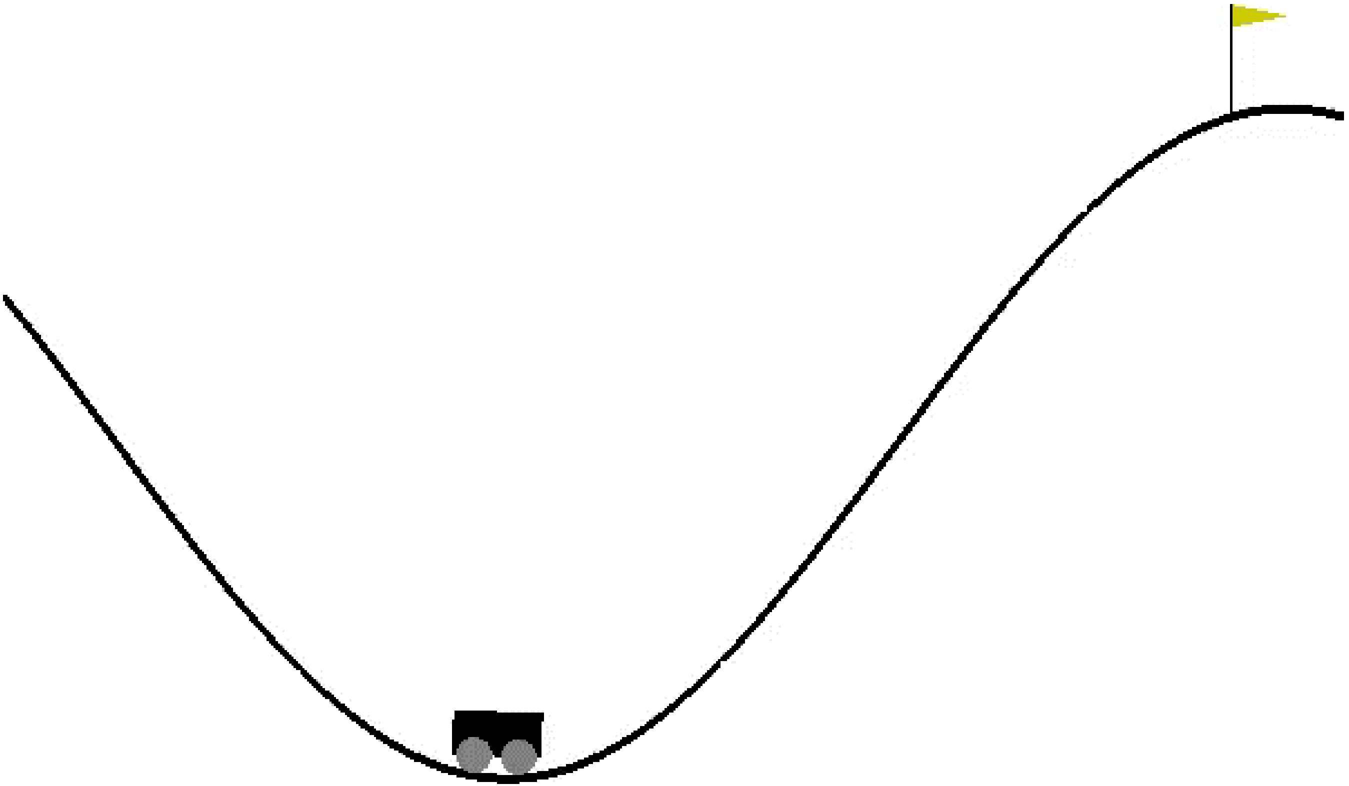


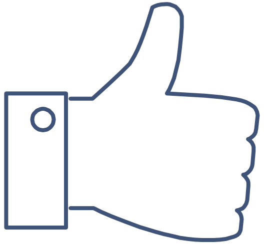
My Solution



Book Solution

Sneak-Peak: Actor-Critic for Mountain Car





Thank you!

Sources

- UCL Course on Reinforcement Learning, David Silver
- Reinforcement Learning – An Introduction, Sutton and Barto
- Reinforcement Learning (GitHub repo), Denny Britz

CREDITS

Special thanks to:

- Presentation template by SlidesCarnival
- Photographs by Startup Stock Photos