Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Winter Term 21/22

Adversarial Self-Supervised Learning with Digital Twins

Lecture-2: Introduction to Reinforcement Learning Model-Free Methods

Prof. Dr. Holger Giese (holger.giese@hpi.uni-potsdam.de)

Christian Medeiros Adriano (christian.adriano@hpi.de) - **"Chris"**

He Xu (he.xu@hpi.de)

"When solving a problem of interest, do not solve a more general problem as an intermediate step"

Vladimir Vapnik [1]

Multi-Armed Bandits

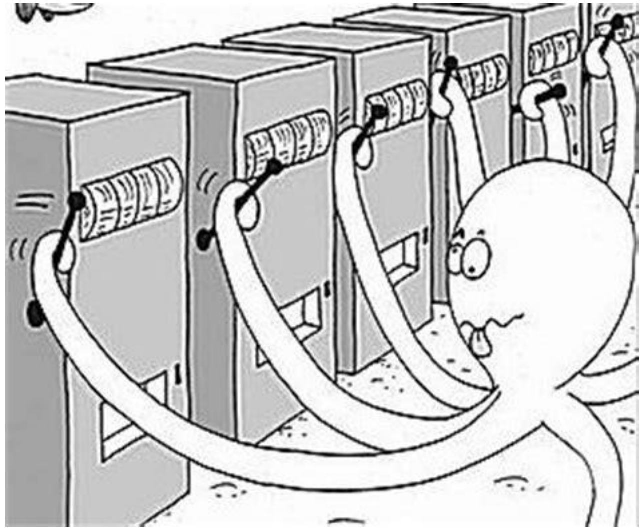
Why Multi-Armed Bandits (MAB)?

MAB allows to learn while making sequential decisions

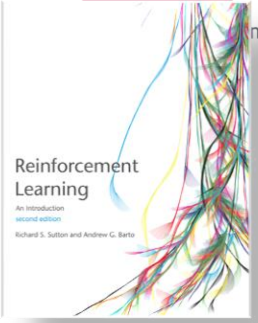
Algorithms are simple, elegant and with rigorous theoretical guarantees of performance

Many applications in practice!

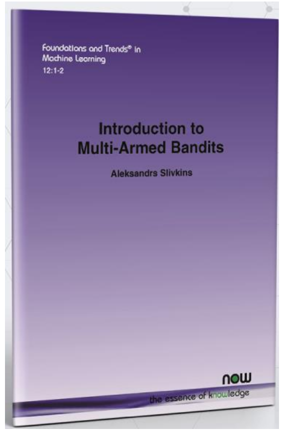
- Clinical trials [1,2,3,4]
- Pricing [5], Innovation [6], Auctions [7]
- Assortment/Recommendations [8]
- Adversting planning [9,10]
- A/B testing [11], Load balancing [12], Routing [13]



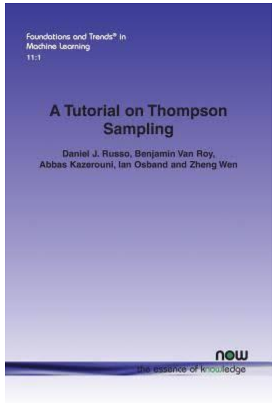
Textbooks (free access)



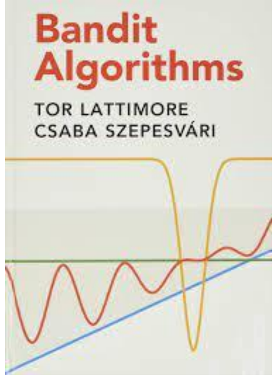
Sutton, R. S., & Barto, A. G. (2015). **Reinforcement learning: An introduction**. MIT press. http://richsutton.com



Aleksandrs Slivkins (2019). **Introduction to Multi-Armed Bandits**, Foundations and Trends® in Machine Learning: Vol. 12: No. 1-2, pp 1-286. https://arxiv.org/abs/1904.07272



Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband and Zheng Wen (2017**). A Tutorial on Thompson Sampling**
https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf



Tor Lattimore and *Csaba* Szepesvári, (2020). **Bandit Algorithms,**
https://tor-lattimore.com/downloads/book/book.pdf

Definition

The multi-armed bandit (MAB) problem consists of an agent (gambler) choice at each round of play one of K **arms**. Each arm has an unknown reward distribution.

Rewards are only obtained (observed) when the arm is chosen. The goal of the agent is to maximize the cumulative expected rewards over a time horizon (T).

The agent strategies consists of balancing the cost of acquiring knowledge about the arms (**exploration**) and maximizing the immediate rewards (**exploitation**).

The trade-off between exploration and exploitation implies a cost is called **regret** (L), which the agent wants to minimize over the time horizon T.

Topics

Exploration vs Exploitation

Regret

Strategies

- ε-Greedy
- Upper-confidence bounds (UCB)
- Bayesian Thompson sampling
- Contextual Bandits

Model

State = only one

Actions = number of arms

Reward = often normalized to be in [0,1]

Implications?

No transitions function to be learned

Only thing left to learn is the stochastic reward function

Problem Formalization

Tuple of (A,R)

- A is a known set of actions
- $R^a(r)$ is the reward

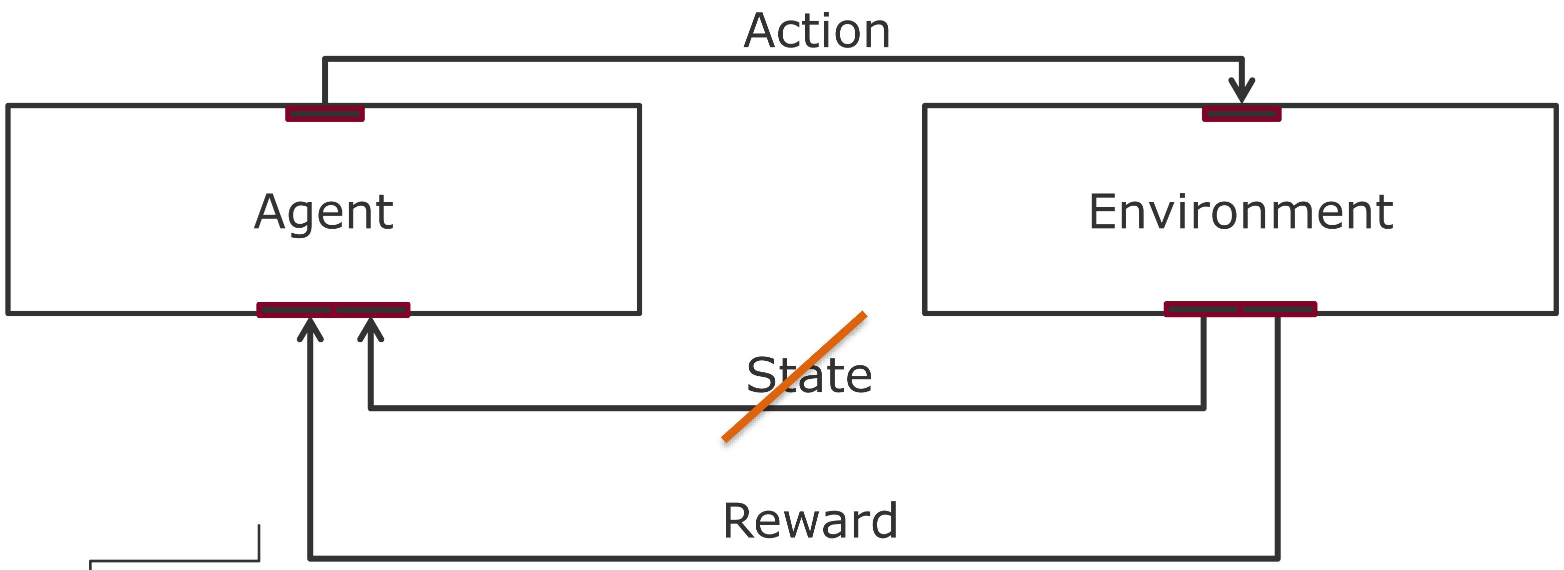
P(R=r|A=a) is an unknow probability distribution

At each step t, the agent selects one action $A_t \in A$

Environment generates a reward $R_t \sim R^{A_t}$

The goal is to maximize the cumulative reward $\sum_{T=1}^{n} R_T$

Agent and Environment Interaction



Agent's Goal: Maximize its reward

Our Goal: Discover a strategy that allows the agent to achieve its goal under various circumstances

Search, Estimate, Learn

Policy = $\pi(S, A)$

knowlege about the environment

Solution Formalization - Regret

Regret = how much I lost in a step by not taking the optimal action

The value of an action $q(a) = E[R|A = a]$

The optimal value $v^* = \mathrm{q}(a^*) = \max_{a \in A} q(a)$

Regret = $l_t = E[v^* - q(A_t)]$

Total Regret $L_t = E[\sum_{r=1}^{t} v^* - q(A_t)]$ ⟵ Goal: Minimize this

Approaches to Trade-off Explore vs Exploit

Random

- Greedy (no exploration)
- ϵ-Greedy (epsilon Greedy)

Systematically

- Upper Confidence Bounds
- Thompson Sampling
- Contextual Bandits

State Space

- Bayes Adaptive Bandits
- Gittin indices Bandits

ϵ-Greedy

Total reward of an action is Monte Carlo estimate of rewards obtain for that action

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t} r_\tau 1[a_\tau = a]$$

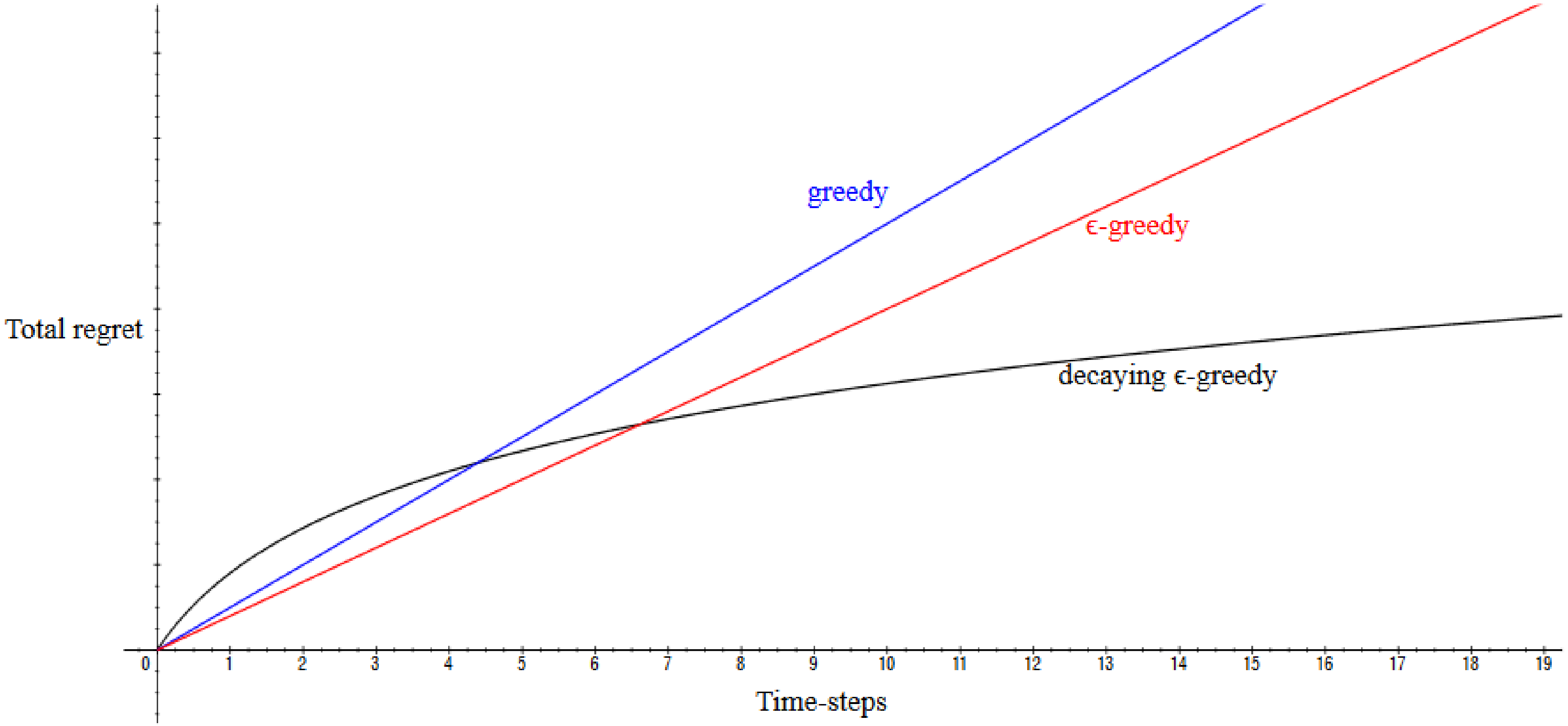
where 1 is a binary indicator function and N_t(a) counts the number of times that action a was selected.

Exploration vs Exploitation

- Probability ϵ that we take a random action
- Probability (1-ϵ) that we take the best action: $q(a^*) = \max_{a \in A} q(a)$

Decaying ϵ-Greedy

Follows a decay series $\epsilon_1, \epsilon_2, \epsilon_3 \ldots \epsilon_n$



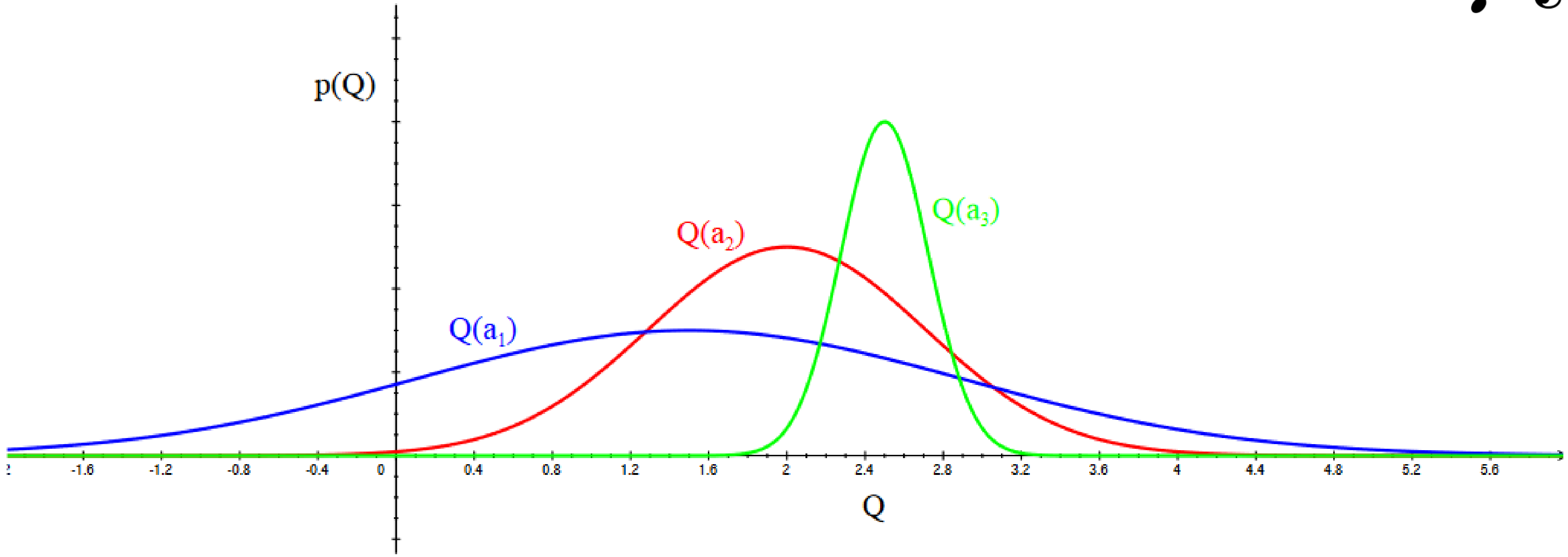
Source: [Silver 2015]

Systematic Exploration

Optimism in face of uncertainty

No catastrophic consequence of our decisions.
We either learn something new or obtain a maximum reward

(what doesn't kill you makes you stronger...)



Source: Silver, https://www.davidsilver.uk/wp-content/uploads/2020/03/XX.pdf

Upper Confidence Bounds

Select action that maximizes the action value with a given upper confidence bound

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \hat{Q}_t(a) + \hat{U}_t(a)$$

$$\mathbb{P}[\mathbb{E}[X] > \overline{X}_t + u] \leq e^{-2tu^2}$$

Hoeffdings Inequality

$$\mathbb{P}[Q(a) > \hat{Q}_t(a) + U_t(a)] \leq e^{-2tU_t(a)^2}$$

$$U_t(a) = \sqrt{\frac{2 \log t}{N_t(a)}}$$

Bayesian Bandits

Bayesian bandits exploit prior knowledge of rewards P(R).

Uses Bayesian Rule to update the probabilities of reward of each action.

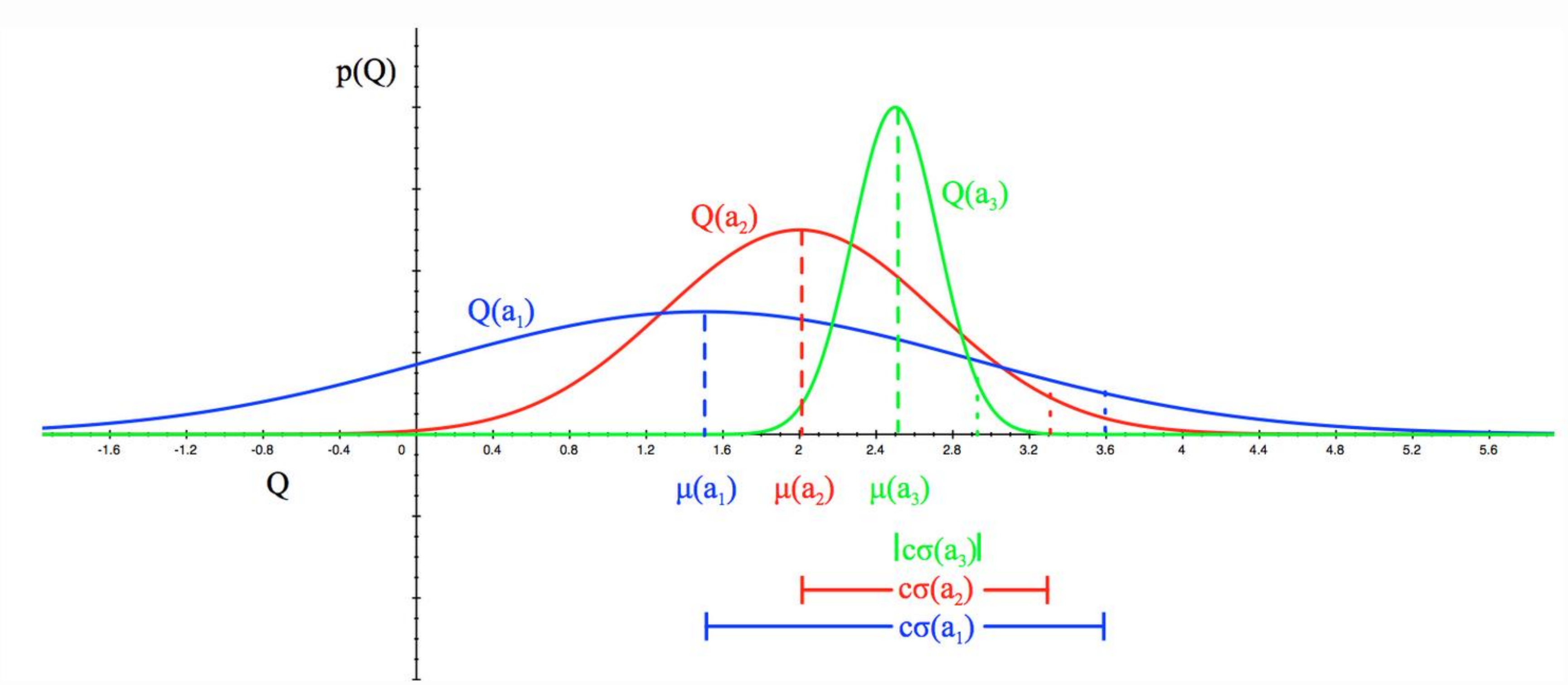
$$P(Reward|Action) = \frac{P(Action|Reward)\,P(Reward)}{P(Action)}$$

Bayesian Bandits depend on good knowledge about the prior distribution of rewards. When this is true, the search process for the best arm is more efficient, i.e., less useless exploration.

Bayesian Bandits Algorithm

The posterior probability can be used by the following algorithms:

- Upper confidence bounds (Bayesian UCB)
- Probability matching (Thompson sampling)



Source: Silver, https://www.davidsilver.uk/wp-content/uploads/2020/03/XX.pdf

Bayesian Bandits – Thompson Sampling

Each arm-bandit is represented by a beta distribution

<u>Algorithm</u>

$posterior = Beta(\alpha, \beta) \,.\, prior$

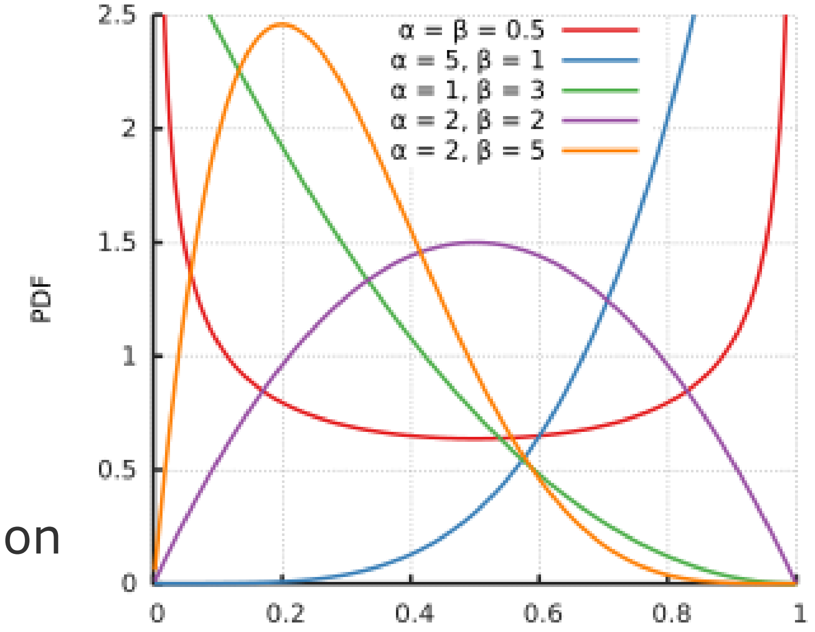
$initialize\ prior = Beta(1,1)$

$Beta(\alpha + 1, \beta)\ if\ we\ observe\ reward = 1$ (Bernoulli arms)

$Beta(\alpha, \beta + 1)\ if\ we\ observe\ reward = 0$

For each arm *i*, sample it proportionate to its posterior distribution

Observe the reward and update the Beta posterior for arm i_t



source: Wikipedia

<u>Advantages</u>:

- Non-parametric (do not need to set an upper-confidence bound)
- Does not assume that the mean is a good approximation of the arm reward
- Guarantees exploration and exploitation at the same time

Contextual Bandits

Problem: Potentially infinite or extremely large number of arms.

Solution: Use features of one arm learn about other arms.

Expected reward for arm *i* at time *t*

$$R(t) = x_{i,t} \,.\, \theta$$

where $x_{i,t}$ is a vector of features and θ are set of parameters

Algorithm arm *i* and observes the $R_t = x_{i,t} \,.\, \theta + \eta_t$

The optimal arm depends on the context $x_t^* = \max_t x_{i,t} \,.\, \theta$

Minimize the regret

$$Regret_t = \sum_t (x_t^* . \theta - x_{i,t} \,.\, \theta)$$

Other Bandit Models

Adversarial Bandits: do not assume that rewards have a distribution (see [1-2])

Combinatorial bandits: special case of Linear Bandits where the actions are multi-dimensional (see [3-4]). See ideas for ranking with bandits in [5].

Markovian bandits: reward processes are neither i.i.d. nor adversarial. Arms are Markov processes with their own state space (see [6-7])

Infinitely many-armed bandits: continuum-armed bandits (arms are in a Euclidean space), which means that the set of possible actions are very large (see [8-9])

References

[1] Chapter 27 in Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. *Cambridge Press, 2020*
[2] https://banditalgs.com/2016/10/01/adversarial-bandits/
[3] Chapter 30 in Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. *Cambridge Press, 2020*
[4] Kleinberg, R. D. (2005). Nearly tight bounds for the continuum-armed bandit problem. In *Advances in Neural Information Processing Systems* (pp. 697-704).
[5] Chapter 32 in Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. *Cambridge Press, 2020*

[6] John C Gittins. Bandit processes and dynamic allocation indices. Journal of the Royal

Statistical Society. Series B (Methodological), pages 148{177, 1979.

[7] John Gittins, Kevin Glazebrook, and Richard Weber. Multi-armed bandit allocation indices.

John Wiley & Sons, 2011.

[8] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. On bayesian upper condence bounds for bandit problems. In Articial Intelligence and Statistics, pages 592-600, 2012.

[9] Chapter-7 in Lattimore, T., & Szepesvári, C. (2020). Bandit algorithms. *Cambridge Press, 2020*

Interesting video lectures

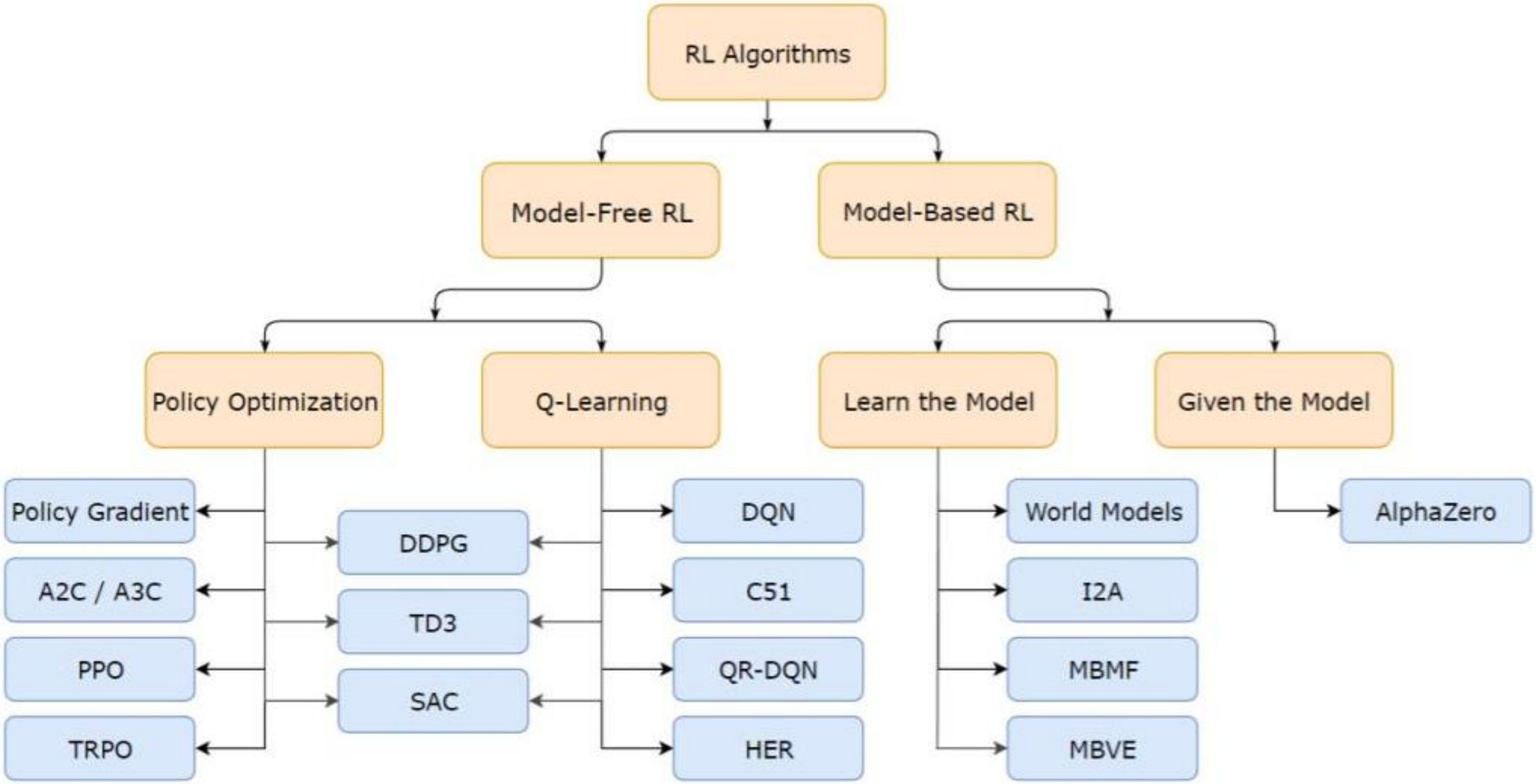
Shipra Agrawal, 2019, Advances in Multiarmed Bandits for Sequential Decision Making https://www.youtube.com/watch?v=7F0jPUyb7m4&pbjreload=10	Good intuition about Bandit algorithms and explanation of the
David Silver, 2015, Lecture 9: Exploration and Exploitation https://www.youtube.com/watch?v=sGuiWX07sKw&pbjreload=10	Great visual explanations, clear formulations of algorithms, and the connection between bandits and MDPs
Tor Lattimore, 2018, Two lectures at Winter Schools for Quant. Sys. Biology Part-1 (https://www.youtube.com/watch?v=xN11-epRuSU&pbjreload=10 Part-2 (https://www.youtube.com/watch?v=NyyLr6F4bkI&pbjreload=10	Good intuitions about the derivations and the proofs
Pascal Poupart, 2018, Lectures at Uni Waterloo, Canada Lecture-1 (https://www.youtube.com/watch?v=Qy-vum3GH-s&pbjreload=10) Lecture-2 (https://www.youtube.com/watch?v=jlcbEZTgisQ&pbjreload=10)	Higher level exposition and insightful Q&A with the students
Charles Isbell & Michael Littman, 2015, Set of Coursera videos https://www.youtube.com/watch?v=rETmf4NnlPM&list=PL__ycckD1ec_yNMjDl-Lq4-1ZqHcXqgm7&index=128&pbjreload=10	Step-by-step explanations with fun discussions about various bandit topics (algorithms, optimality, metrics, connection with MDPs)
Nando Freitas, 2013, Lecture at Uni British Columbia, Canada https://www.youtube.com/watch?v=vz3D36VXefI&list=PLE6Wd9FR--EdyJ5lbFl8UuGjecvVw66F6&index=10	Great graphics and explanations about the relation between Bandits and Bayesian Optimization

Reinforcement Learning

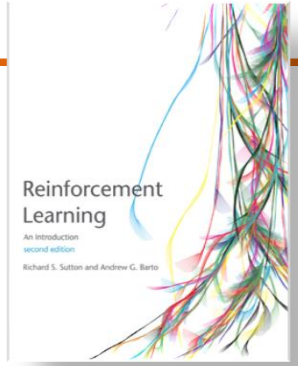
Topics

- Review of MDP Learning Algorithms
- Concepts
- Model-based algorithms (Value or Policy iteration)
- Model-free algorithms (Sarsa and Q-Learning)
- Model-based architectures
  - Dyna-Q
  - POMDP / Hidden-Markov Models

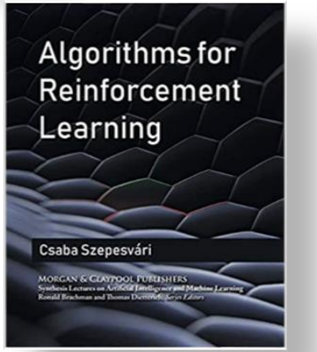
Brief Taxonomy of RL Methods



Resources used



Sutton, R. S., & Barto, A. G. (2015). ***Reinforcement learning: An introduction***. MIT press. http://richsutton.com



Csaba Szepesvári (2010). ***Algorithms for reinforcement learning*** Morgan and Claypool. https://sites.ualberta.ca/~szepesva/rlbook.html

Lectures at UCL from DeepMind Research Scientists:

- David Silver, 2015, https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf
- Hado Van Hansselt, 2018 https://www.youtube.com/watch?v=nnxHlg-2WgA&list=PLqYmG7hTraZBKeNJ-JE_eyJHZ7XgBoAyb&index=4

Basic concepts - Structure

Agent: external to the environment, take actions and has access to external states

Environment: constrains the agent, has hidden and visible states

State: visible information about the environment after an action was taken

Reward: value obtained after taking an action or being at a state

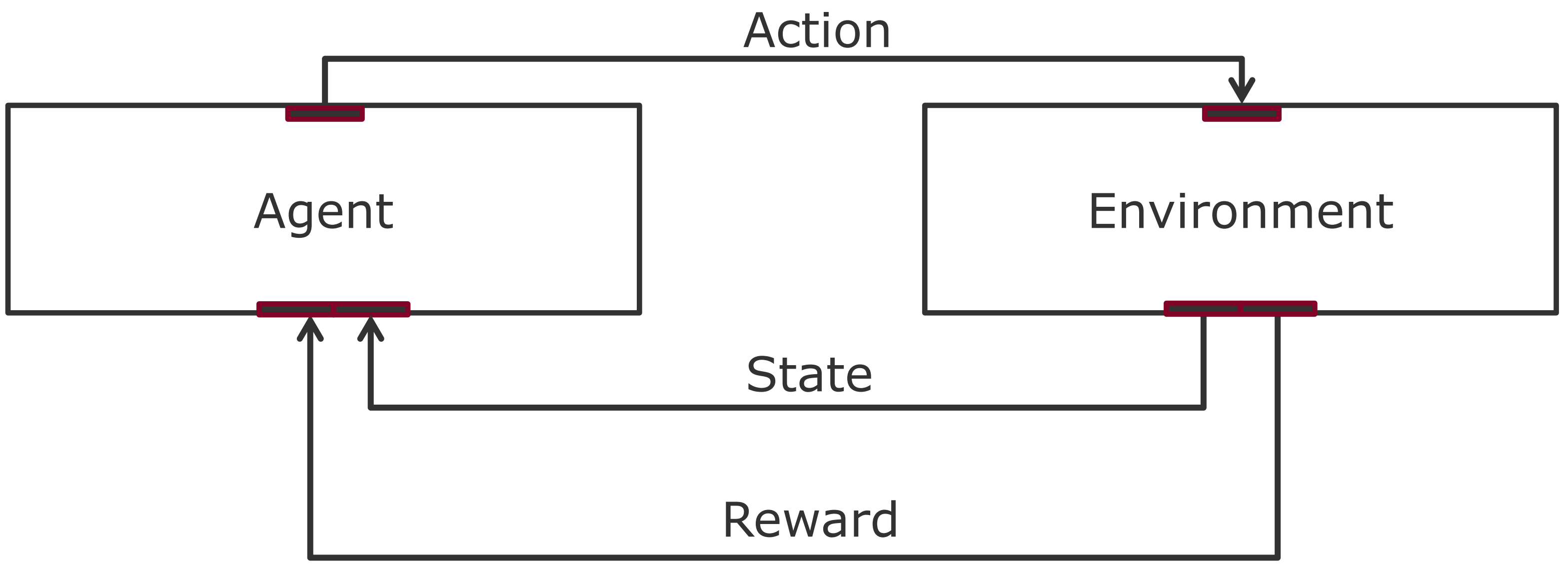
Action: taken by an agent, might cause a change in state

Goal: Maximize rewards by using an optimal policy

Policy: set of state action pairs

Markov property: we only need to know the current state to predict the next state

Agent and Environment Interaction



Agent's Goal: Maximize its reward

Our Goal: Discover a strategy that allows the agent to achieve its goal under various circumstances

Search, Estimate, Learn

Policy = $\pi(S, A)$

knowlege about the environment

State

History of observations $h_t = (O_1, O_2, \ldots, O_t)$

The state at given time is function of this history $S_t = F(h_t)$

hence, the state summarizes all information needed to make decisions

Intuition (why is this a good idea?)

- Conditioned on the histories the actions are independent.
- This allows us to factor one decision into small decisions, or actions.

$$\pi(a_{0:T}) = \prod_{t=0}^{T} \pi(a_t | h_t)$$

Markov Property

The Future is independent of the Past if we know the Present

$$(S_{t+1} \perp S_{t-1}) \mid S_t$$

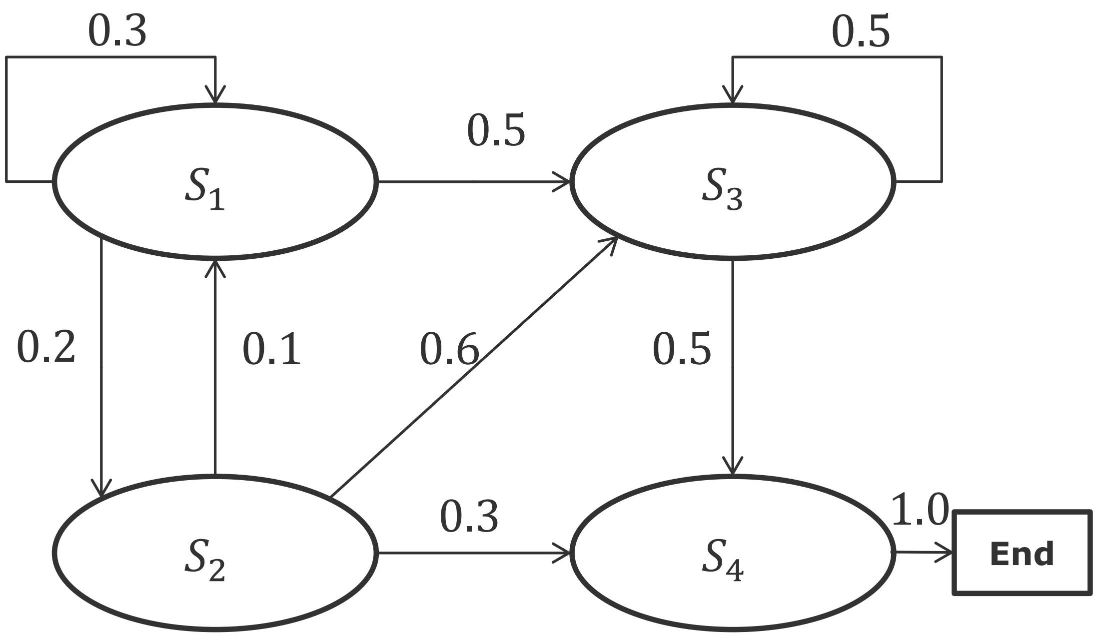
Or equivalently

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_t, S_{t-1}, \dots, S_{t-n})$$

<u>What are the implications?</u>

- The present state needs to hold all the information necessary to predict the future
- Future inherently stochastic
- State space explosion

Markov Chain



Transition matrix

Transition model $T = \{S_t, P, S_{t+1}\}$

	S_1	S_2	S_3	S_4
S_1	0.3	0.2	0.5	0.0
S_2	0.1	0.0	0.6	0.3
S_3	0.0	0.0	0.5	0.5
S_4	0.0	0.0	0.0	0.0

Problems of learning and planning
Problems of prediction and control

Return (G) and Discount Factor (γ)

Return is the sum of discounted rewards

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

where $\boldsymbol{G_t}$ is the total reward at step $\boldsymbol{t}$ after collecting future rewards in **T** steps discounted by $\boldsymbol{\gamma} \in [\mathbf{0}, \mathbf{1}]$

Why Discount Factor?

- depreciates future rewards based on their distant from the present
- avoids infinite returns because of cycles in the Markov Chain

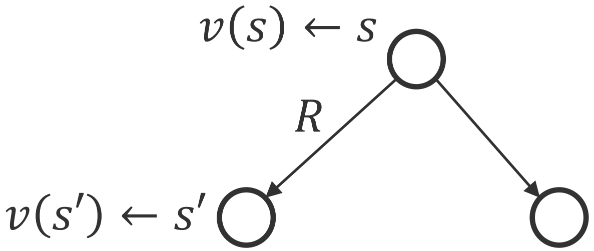
Value Function

State Value Function

- $V_\pi(s) = how\ good\ is\ to\ be\ in\ State\ \boldsymbol{s}\ if\ I\ follow\ policy\ \boldsymbol{\pi}\ ?$
- It is the value of being at a certain state
- It is not a random variable; it is the <u>expectation</u> over the future rewards

Bellman Equation

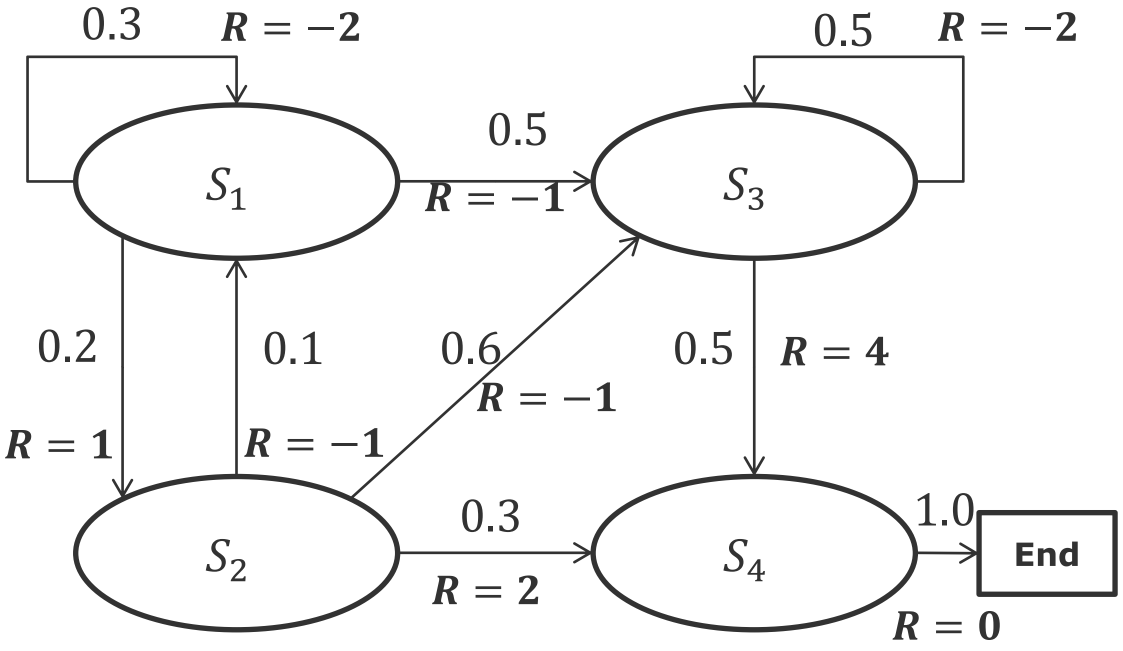
$$v(s) = E[R_{t+1} + \gamma . v(S_{t+1}) \mid S_t = s]$$

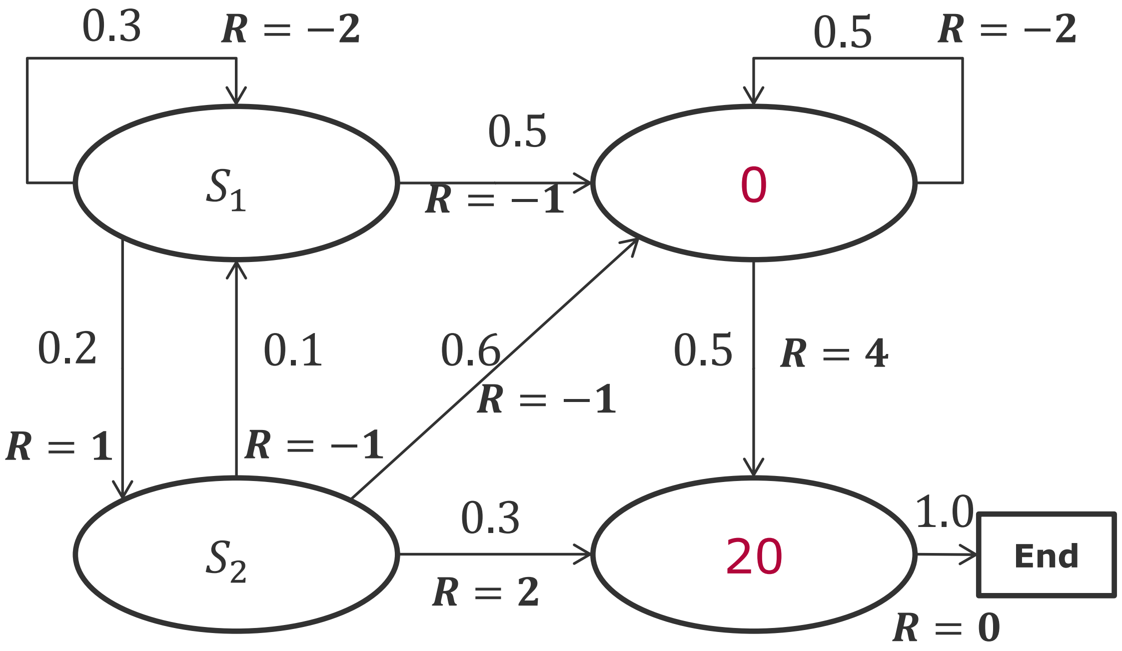


One-Step Lookahead idea

$$v(s) = R + \gamma \sum_{s' \in S} P_{ss'} \, v(s')$$

Markov Reward Process (MRP)





Assuming γ=1

Initialize value states with zero

Value of S_3 = 4 + 20*0.5 -2 +0.5*0.0 =14

Bellman equation in matrix form

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

$$v = R + \gamma P v$$

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \cdots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

$$v = (1 - \ \gamma P v)^{-1} \ R$$

Feasible for small number of states

Linear version of the Bellman Equation

Markov Decision Process (MDP)

MDP = Markov Chain + Actions

Actions allow us to have control over the decision-making process

Policy = distribution over actions given states $\pi(s, a) = P(A_t = a | S_t = s)$

<u>Implications to the model</u>:

- Transition matrix for each action
- Results of actions can be stochastic (non-deterministic)

Action Value Function

$$Q_\pi(s, a) = how\ good\ is\ to\ be\ in\ State\ \mathbf{s}\ and\ to\ take\ action\ \mathbf{a}\ if\ I\ follow\ policy\ \boldsymbol{\pi}\ ?$$

- i.e., $Q_\pi(s, a)$ is the expected return of being at a state and taking a certain action and following a certain policy

Bellman Optimatility Equation

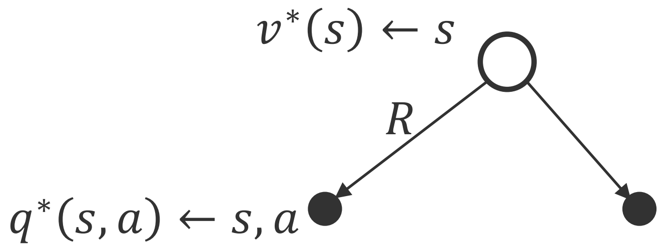
How to find the best path in the MDP?

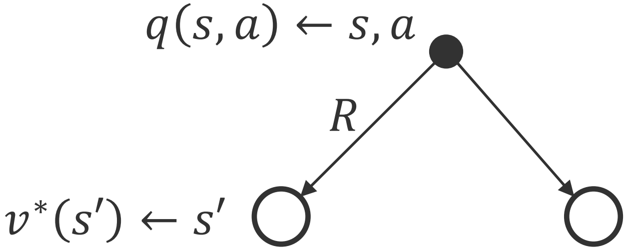
Bellman equation estimates V(s) by looking ahead of possible future trajectories in the state-space

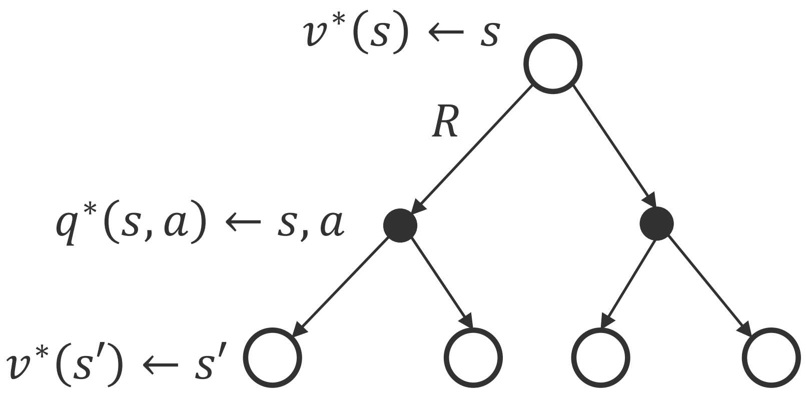
Intuition

If we behave optimally for the first step and then also behave optimally for the next step, we will find the optimal solution for the MDP. However, note that to behave optimally I need to average over all possible paths that are opened by my first step.

Bellman Optimality Equations for MDPs







$$v^*(s) = \max_a q^*(s,a)$$

The value function of a state is the maximum that I can get after taking an action

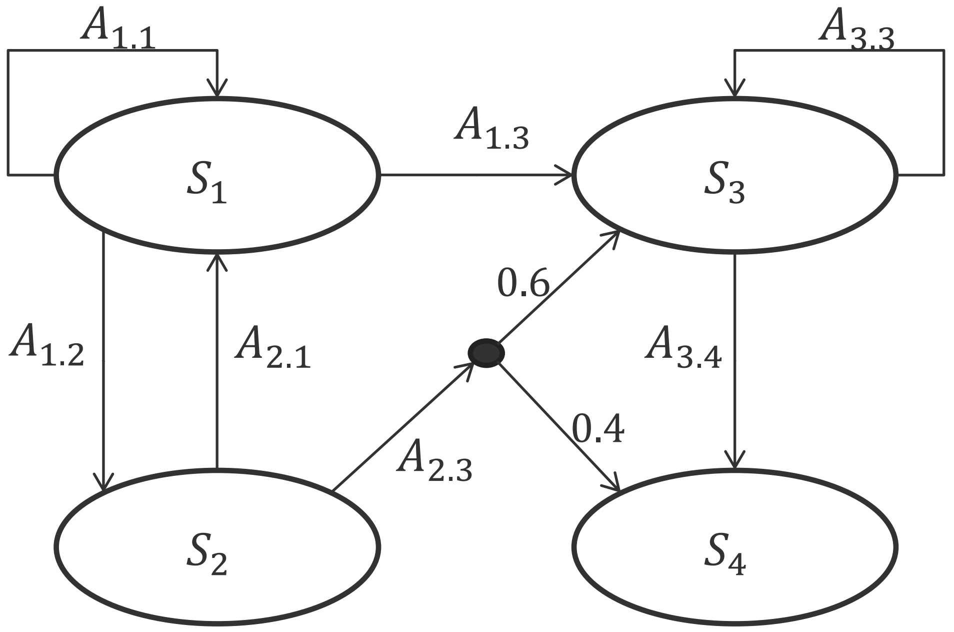
$$q^*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s')$$

The optimal value is the expected return of the multiple states we can end up after taking action **a**

$$v^*(s) = \max_a \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v^*(s') \right)$$

Episodes

Episode	Transitions	Σ Reward
1	$\{S_1, A_{1.3}, S_3\}, \{S_3, A_{3.3}, S_3\}\{S_3, A_{3.4}, S_4\}$	90
2	$\{\boldsymbol{S_1}, \boldsymbol{A_{1.2}}, \boldsymbol{S_2}\}, \{S_2, A_{2.3}, S_3\}\{S_3, A_{3.4}, S_4\}$	110
3	$\{\boldsymbol{S_1}, \boldsymbol{A_{1.2}}, \boldsymbol{S_2}\}, \{S_2, A_{2.3}, S_4\}$	60
4	$\{\boldsymbol{S_2}, \boldsymbol{A_{2.1}}, \boldsymbol{S_1}\}, \{S_1, A_{1.3}, S_3\}\{S_3, A_{3.4}, S_4\}$	120
5	$\{S_3, A_{3.3}, S_3\}\{S_3, A_{3.4}, S_4\}$	40
6	$\{S_2, A_{2.3}, S_3\}, \{S_3, A_{3.3}, S_3\}\{S_3, A_{3.4}, S_4\}$	80
7	$\{S_2, A_{2.4}, S_4\}$	40



What can we infer from these episodes?

- We can infer the $R(\{S_1, A_{1.2}, S_2\}) = 60$
- We do not need to know all transitions
- But we are not 100% sure because transitions are not deterministic

Model-Based Algorithms

Prediction versus Control Problems

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Prediction

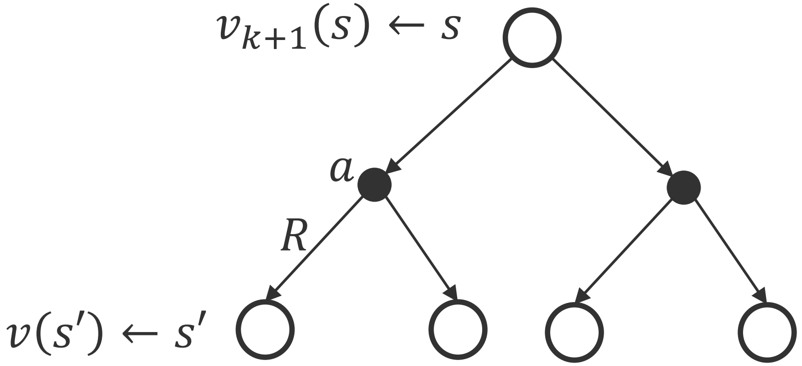
- Goal: How much reward I am going to obtain from this MDP?
- Input: An MDP (S, A, P, R, γ) and Policy π
- Output: Value Function $V_\pi(s)$

Control

- Goal: Among all possible policies, what is the best policy for this MDP?
- Input: An MDP (S, A, P, R, γ)
- Output: an optimal Value Function $V_\pi^*(s)$, which corresponds to an optimal policy π*

Control
For-loop:
{ Prediction }

Policy Evaluation and Policy Iteration



One step lookahead

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(\underbrace{R_s^a}_{\text{current Reward}} + \underbrace{\gamma \sum_{s' \in S} P_{ss'}^a v(s')}_{\text{discounted future Reward}} \right)$$

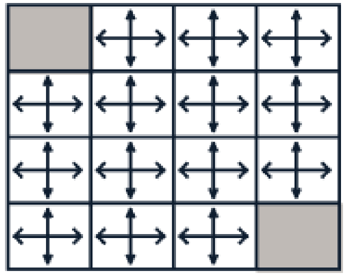
Evaluate: $v^{k+1}(s) = R^{\pi} + \gamma P^{\pi} v^k$

Improve: $\pi'(s) = \max_{a \in A} q_\pi(s, a)$ *Greedy!*

v_k for the Random Policy | Greedy Policy w.r.t. v_k

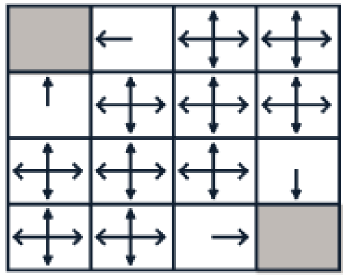
$k = 0$

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



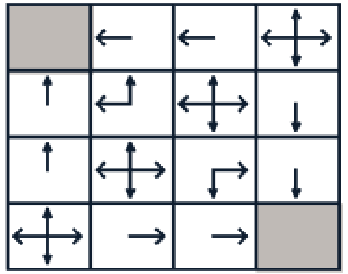
$k = 1$

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



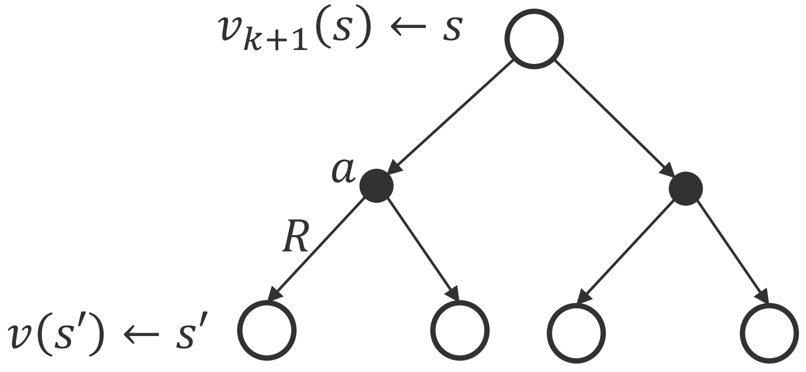
$k = 2$

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



Source: Silver, 2015, https://www.davidsilver.uk/wp-content/uploads/2020/03/DP.pdf

Value Iteration



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

$$v^{k+1}(s) = \max_{a \in A}\left(R^a + \gamma P^a v^k\right)$$

<u>Algorithm</u>:

For 1+k iteration:

  For all states in S:

    Update $v^{k+1}(s)$ from v^k(s)

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

<u>Source</u>: Silver, 2015, https://www.davidsilver.uk/wp-content/uploads/2020/03/DP.pdf

Model-Free Algorithms

Monte Carlo Policy Evaluation

Sample the rewards for each action across multiple Episodes

Similar to what we have seen in the Multi-Armed Bandits

Discounted Rewards for an episode ending at time T>t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Value function is the Expected Return

$$V_\pi = E[G_t \mid S_t = s, \pi]$$

Caveat: Multiple episodes necessary to update the state value function under a policy $V_\pi(S)$

Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

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

Source: Silver, 2015, https://www.davidsilver.uk/wp-content/uploads/2020/03/MC-TD.pdf

Incremental Monte Carlo Updates

Updates $V_\pi(S)$ after each episode

- Increments visits $\mathrm{N}(S_t)++$
- Update $V_\pi(S) = V_\pi(S) + \frac{1}{\mathrm{N}(S_t)}(G_t - V_\pi(S))$

We are essentially tracking a running average, in other words, we are forgetting older episodes. Rate of forgetting can be further parameterized with a learning factor alpha

$$V_\pi(S) = V_\pi(S) + \alpha(G_t - V_\pi(S))$$

Advantages:

- Non-stationary estimator
- Memoryless algorithms

Caveats:

- Requires complete episodes must terminate (reach terminal state)
- High variance, low bias (because G_t is an unbiased estimate of future rewards)

Temporal Difference Learning

Recalling Policy Iteration

$$v^{t+1}(s) = R^{\pi} + \gamma P^{\pi} v^{t}$$

$$v_{t+1}(S_t) = E[\underbrace{R_{t+1} + \gamma v_t(S_{t+1})}_{\text{Bootstrapping}} | \, S_t = s, A_t \sim \pi(s_t)]$$

Monte Carlo

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

TD Learning

$$V(S_t) \leftarrow V(S_t) + \alpha\,(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD-Error}} - V(S_t))$$

Advantages:

- Updates with partial episodes
- Size of partial episode is parameterizable (lambda)

Caveat: High Bias, but less variance (because update depends on only one action)

Batch Learning = Limited Budget

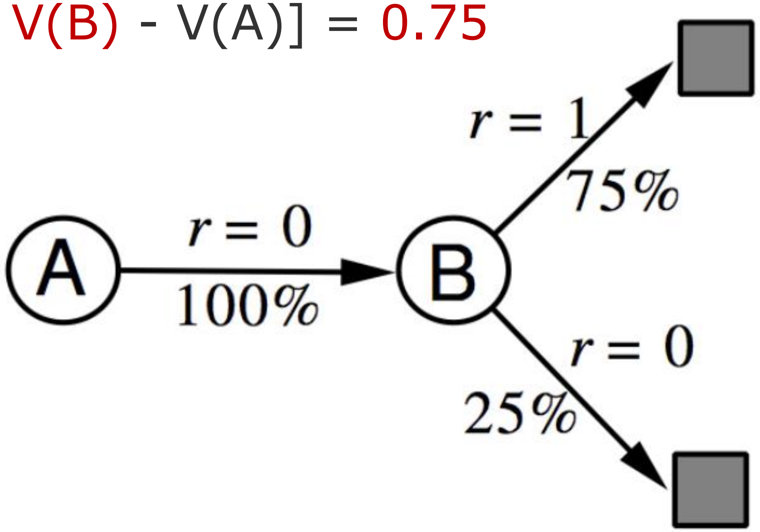
HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

The batch dataset has 8 episodes with the following (state, return) values

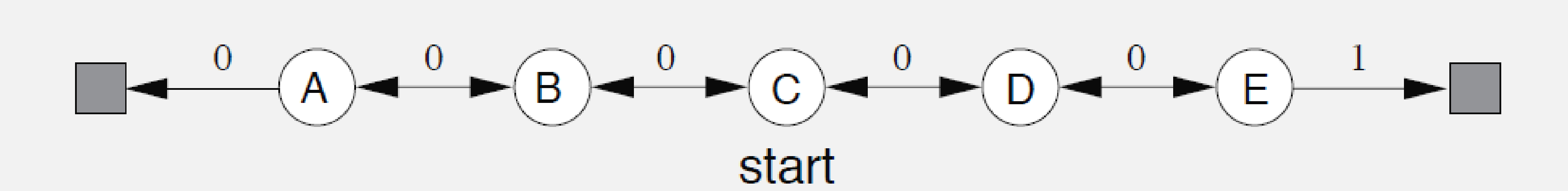
Episode	Data
1	(A,0), (B,0)
2	(B,1)
3	(B,1)
4	(B,1)
5	(B,1)
6	(B,1)
7	(B,1)
8	(B,0)

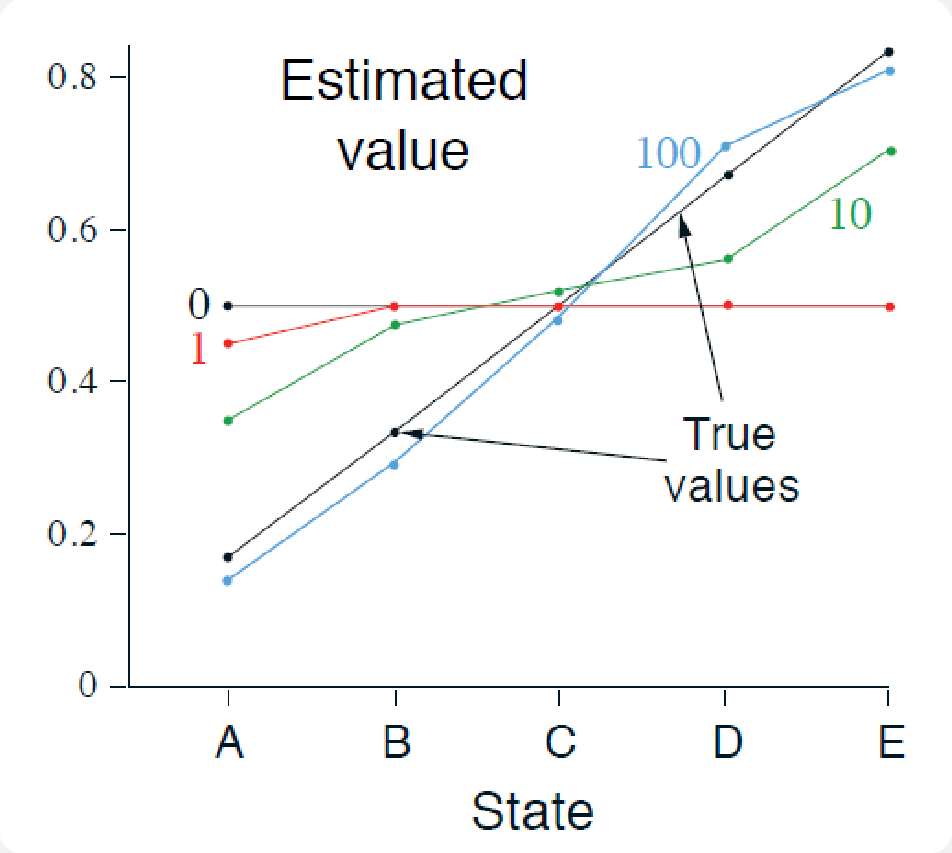
What are the value function estimates for A and B?

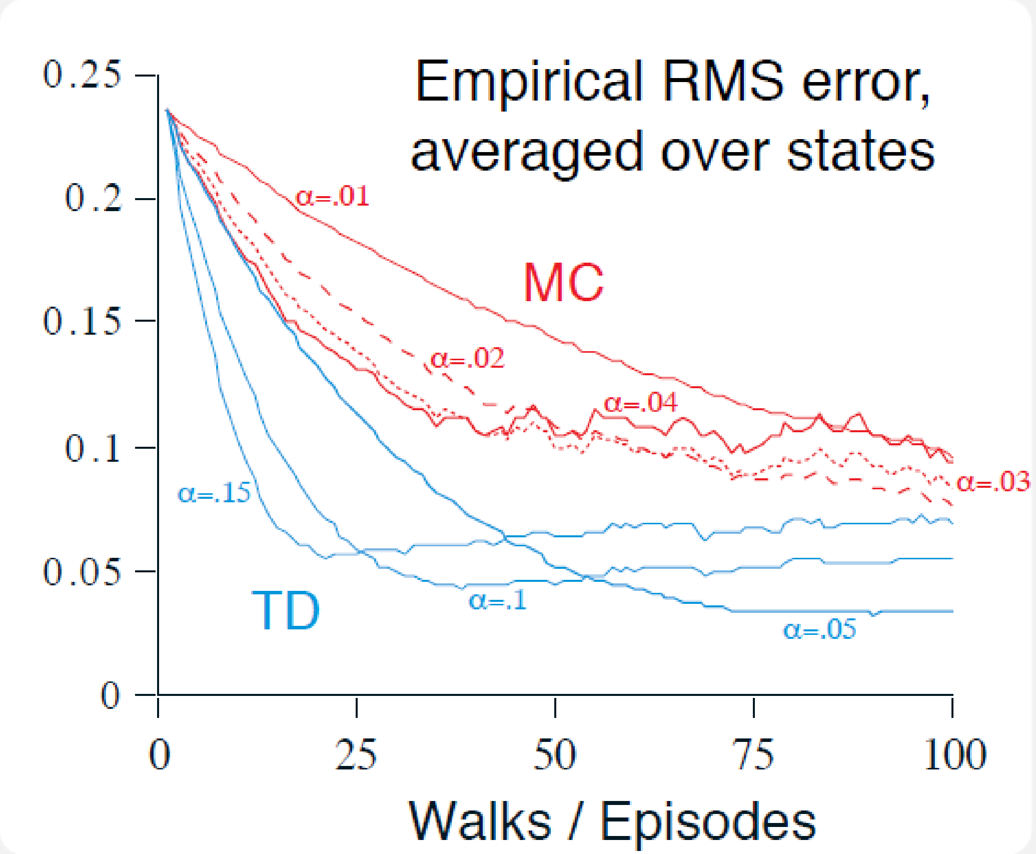
- Monte Carlo:
  - (A) = 0/8 = 0,
  - V(B) = 6/8 = 0.75
- Temporal Difference:
  - V(A) = R(A) + [R(B) + V(B) - V(A)] = 0.75
  - V(B) = 0.75



Temporal Difference (TD) vs Monte Carlo (MC)







source: page 125 [Sutton & Barto 2018]

Summary of pros and cons

Monte Carlo

- High Variance, Low Bias
- Good Convergence properties (even with function approximation)
- Not very sensitive to initial values
- Simple to understand and use
- More efficient in non-Markov environments, because it does not explore the Markov Property

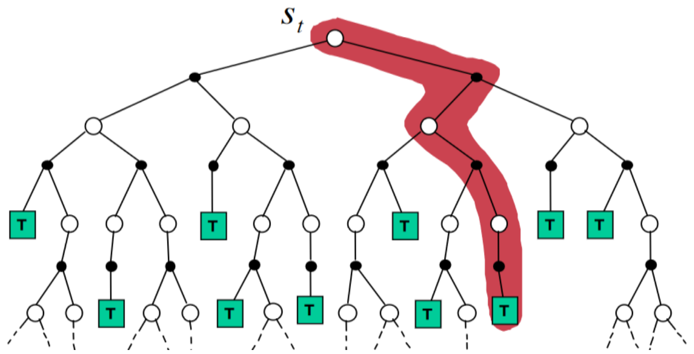
TD

- Low Variance, High Bias
- TD(zero) converges to $v_\pi(s)$ (not always with function approximation)
- More sensitive to initial values
- More efficient in Markov environments, because it does explore the Markov Property

Comparing the three models

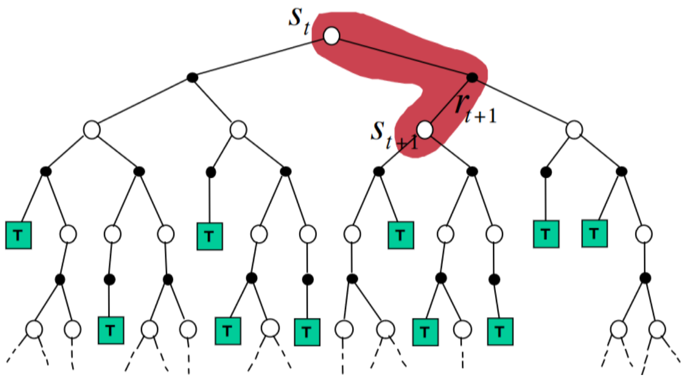
Monte Carlo

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



Temporal Difference

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



Bootstrapping: updating = to estimate

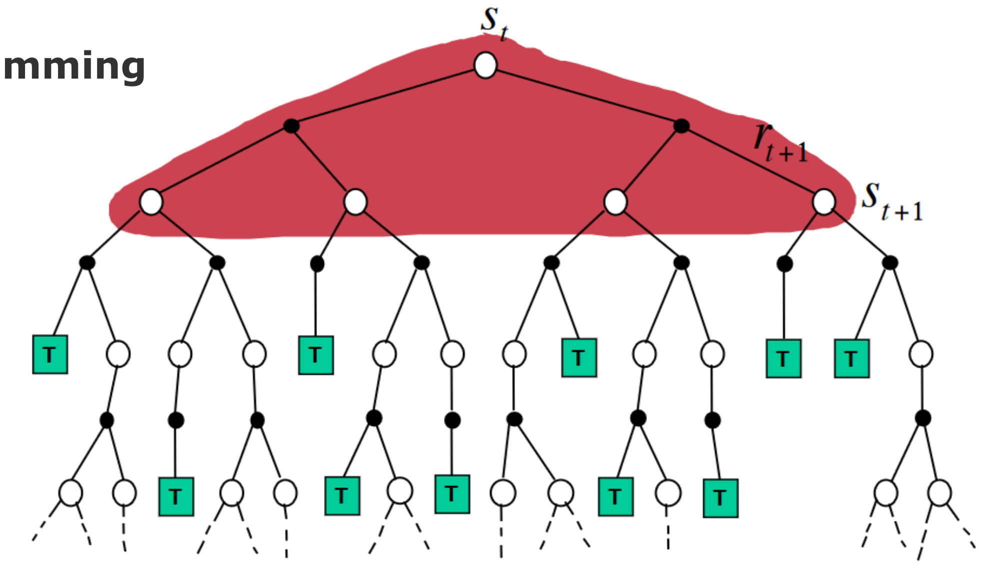
- MC does not bootstrap
- DP and TD bootstrap

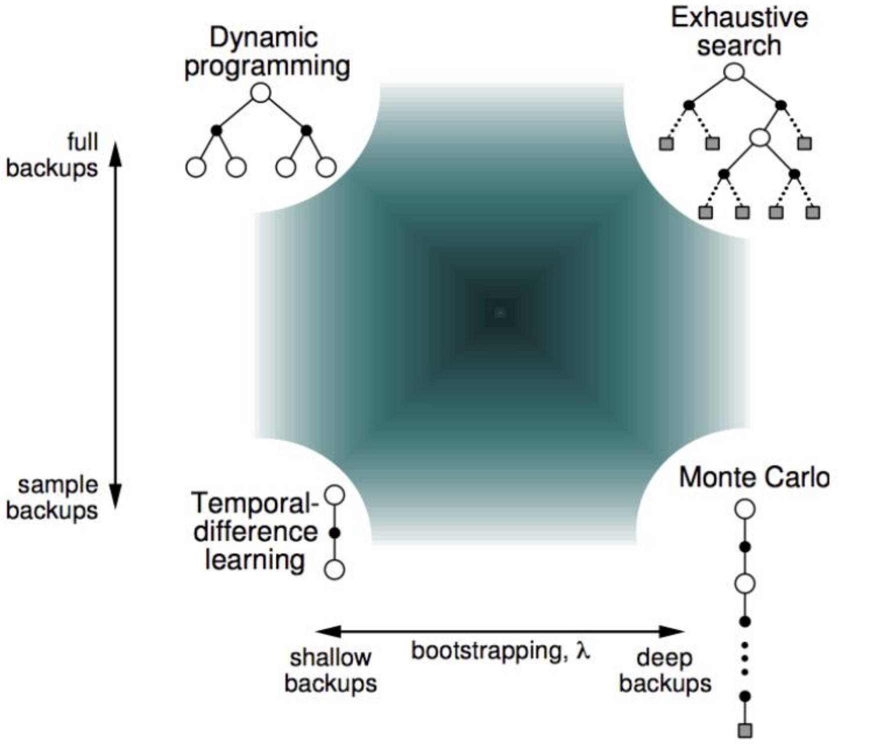
Sampling: updating = to sample an expectation

- MC and TD sample
- DP does not sample

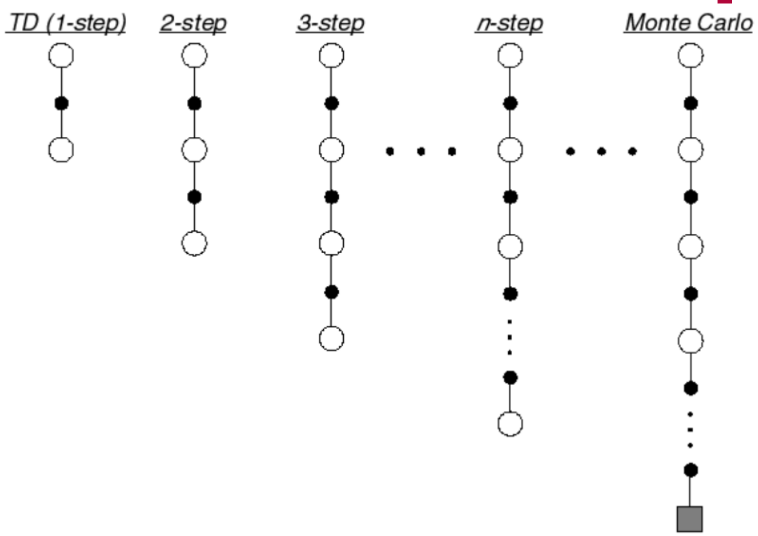
$$V(S_t) \leftarrow \mathbb{E}_\pi\,[R_{t+1} + \gamma V(S_{t+1})]$$

Dynamic programming





Trade-off between TD(0) and MC = TD(lambda)



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

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$

Define the n-step return

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

n-step temporal-difference learning

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

Which n-value is the best?

- Alpha = Learning rate
- Gamma = Discount rate

TD(λ) Which n value is the best?

Average RMS error over 19 states and first 10 episodes

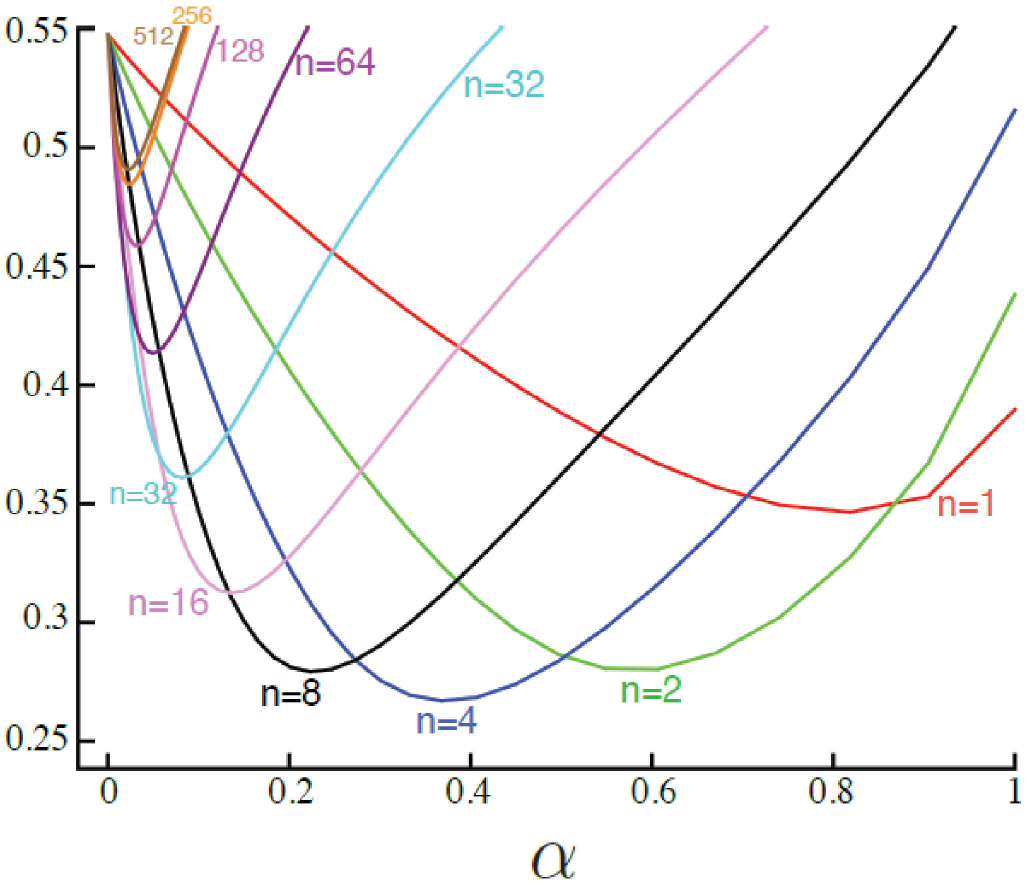
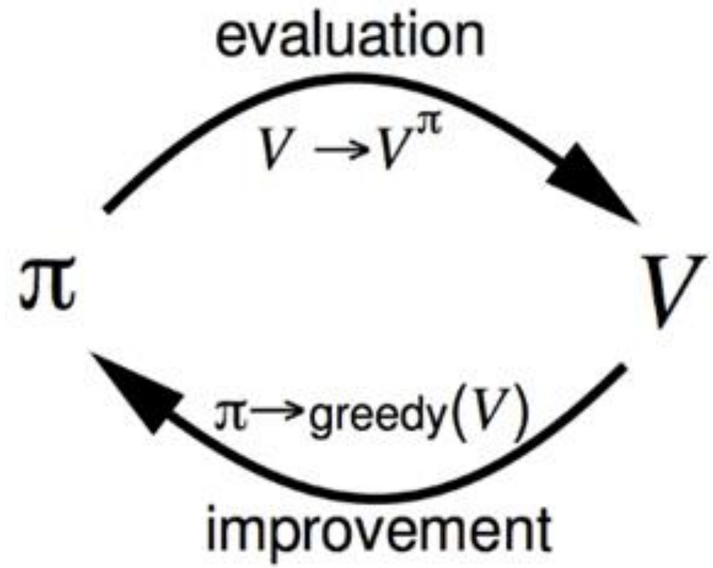


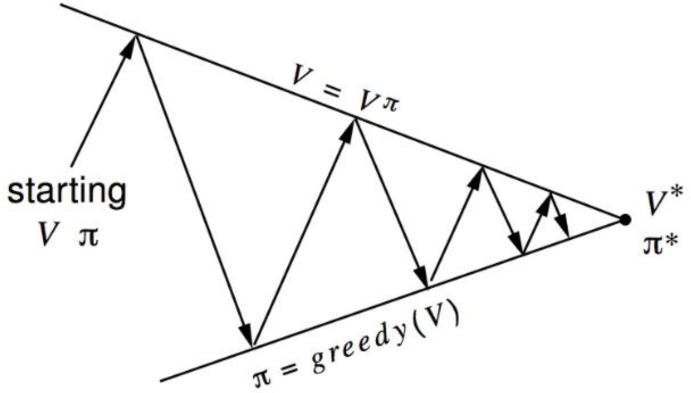
Figure 7.2: Performance of n-step TD methods as a function of α, for various values of n, on a 19-state random walk task (Example 7.1). ■

Control
Improving the policy to find optimal value-function

Generalized Policy Iteration



Convergence



π* ==V*

1. Policy Evaluation:
- What? estimates V^{π}
- How? Iterative policy evaluation →

2. Policy Improvement:
- What? generates a π′ ≥ π
- How? Greedy policy improvement →

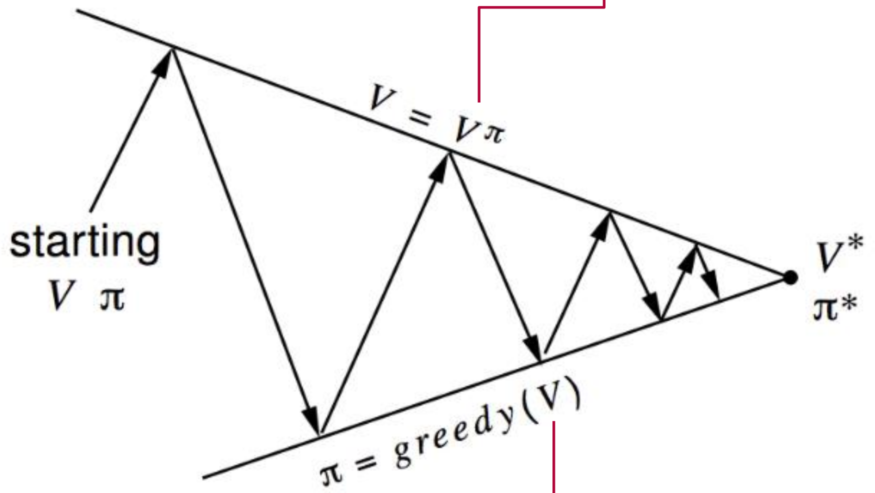
Variations to these components:

- Monte Carlo Evaluation

- epsilon-Greedy (fixed exploration rate)
- Monte Carlo GLIE (decaying exploratoin rate)
- TD(0) and TD(lambda)

Monte Carlo Control

$$v^{k+1}(s) = \max_{a \in A}\left(R^a + \gamma P^a v^k\right)$$



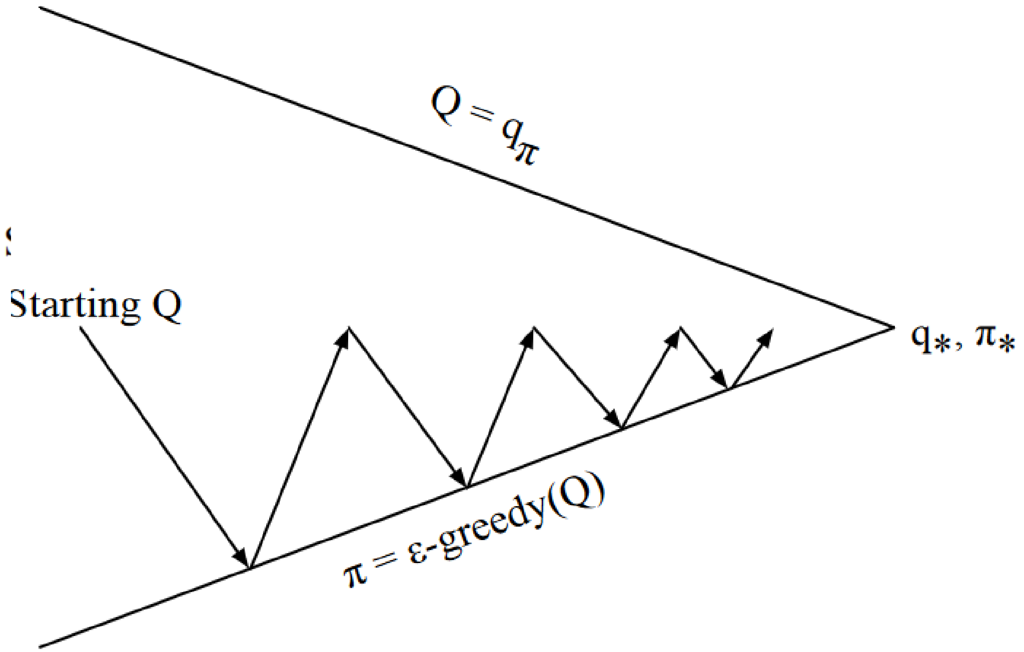
No guarantee that will find the optimum

Control

for-loop:

{ **Prediction** }

<u>Solution part-1</u>: use Q because it is based on state, action pairs



<u>Solution part-2</u>: add randomness in the choice of action (e-Greedy)

GLIE Monte Carlo Control

GLIE – **G**reedy in the **L**imit with **I**nfinite in the limit **E**xploration

- Decays epsilon, for instance ε

$$\varepsilon = \frac{1}{k}, \text{ where k is the episode count}$$

- All <state,action> pairs are explored indefinitely

$$\lim_{k\to\infty} N_k(s,a) = \infty$$

- The policy π converges to a greedy policy

$$\lim_{k\to\infty} \pi_k(s,a) = I(a = \max_{a'} q_k(s,a'))$$

- Theorem shows that it also converges to the optimal value-function

$$\lim_{k\to\infty} \pi_k(s,a) = I(a = \max_{a'} q_k(s,a'))$$

<u>Advantage:</u> Converges to the optimal greedy policy

<u>Caveat:</u> Still expensive, even with partial episodes

On-Policy versus Off-Policy Control

On-Policy

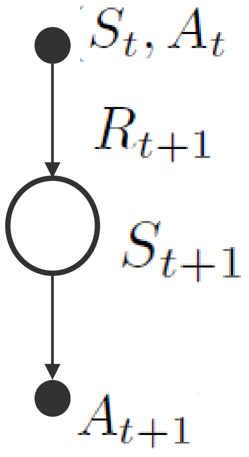
- Learn on the job
- How? Learn about policy π from experience sampled from π

Off-Policy

- Look over someone else's shoulder
- How? Learn about policy π from experience sampled from the environment

Sarsa: On-Policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]. \qquad (6.7)$$



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

<u>Advantage:</u> Converges to the optimal greedy policy

<u>Caveat:</u> Still expensive, even with partial episodes

Q-Learning: off-policy TD Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \underbrace{\max_a Q(S_{t+1}, a)}_{\text{Choose among all possible actions}} - Q(S_t, A_t)\Big].$$

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

Comparing Sarsa and Q-Learning

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

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 $Q(terminal, \cdot) = 0$

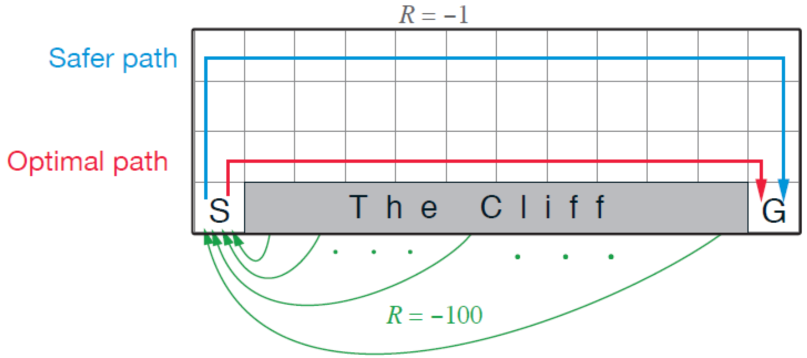
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

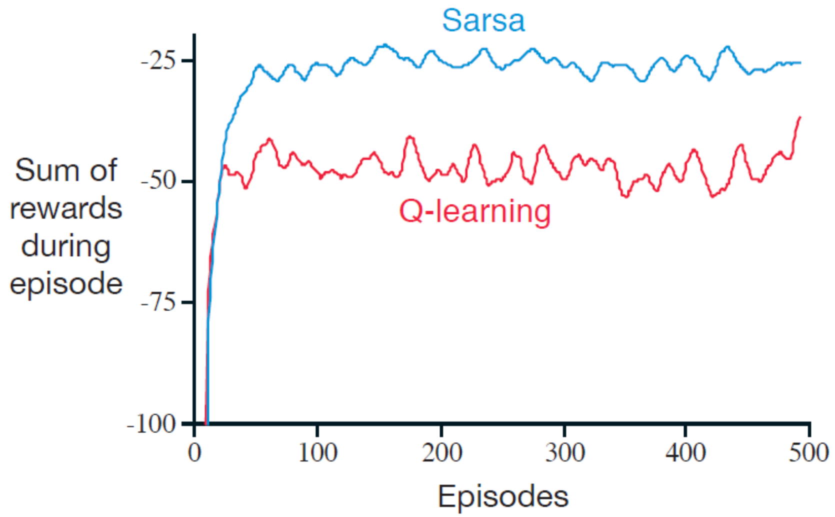
- Q-Learning does not follow any given policy to update the Q-value
- However, Q-Learning still follows a “behavioral” policy in order to determine which state-action pairs to visit

Trade-off between Sarsa and Q-Learning

Gridworld example from [Sutton & Barto 2018] page 132.

Goal: is to go from S to G avoiding the Cliff.





N-Step Sarsa TD(λ)

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T - n, \quad (7.4)$$

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right], \qquad 0 \leq t < T, \quad (7.5)$$

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \left[G - Q(S_\tau, A_\tau) \right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Advanced Model-Free Architectures

- DQN
- Policy Gradient
- A2C, A3C
- PPO
- TRPO
- DDPG
- SAC

To be continued in the next lecture

Interesting lectures

David Silver, 2015, University College London Lecture 1 https://www.youtube.com/watch?v=2pWv7GOvuf0 Lecture 2 https://www.youtube.com/watch?v=lfHX2hHRMVQ&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=2&pbjreload=10 Lecture 3 https://www.youtube.com/watch?v=Nd1-UUMVfz4&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=3&pbjreload=10 Course website: https://www.davidsilver.uk/teaching/	Bottom-up explanations based on introducing the simpler models before adding the complexity that justify more complex models. Explains the math behind concepts.
Pascal Poupert, 2018, University of Waterloo, Canada Lecture 1b: https://www.youtube.com/watch?v=yOWBb0mqENw&list=PLdAoL1zKcqTXFJniO3Tqqn6xMBBL07EDc&index=2&pbjreload=10 Lecture 2a: https://www.youtube.com/watch?v=yOWBb0mqENw&list=PLdAoL1zKcqTXFJniO3Tqqn6xMBBL07EDc&index=2&pbjreload=10 Lecture 2b: https://www.youtube.com/watch?v=mjyrRG7RD84&list=PLdAoL1zKcqTXFJniO3Tqqn6xMBBL07EDc&index=5&pbjreload=10 Course website: https://cs.uwaterloo.ca/~ppoupart/teaching/cs885-spring20/schedule.html	More conceptual level with examples of applications.
Charles Isbell & Michael Littman, 2015, Georgia Tech https://www.youtube.com/watch?v=rETmf4NnlPM&list=PL__yckD1ec_yNMjDl-Lq4-1ZqHcXqgm7&index=128&pbjreload=10	Step-by-step explanations with fun discussions and lot of small examples to illustrate each core MDP principle
Nando Freitas, 2015, Lectures Oxford University, UK Lecture 15 https://www.youtube.com/watch?v=kUiR0RLmGCo Lecture 16 https://www.youtube.com/watch?v=dV80NAlEins Course Website: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/	Lectures in the context of Deep Learning, but still provide the necessary concepts of RL.

End

Incremental implementations

Budget

State Space		Infinite	Finite
	Finite	Model-Based	Model-Free
	Infinite	Model-Free or Approximate	Approximate