Reinforcement Learning

Workshop 1: An introduction and tabular methods

Netbrain.ml

Logistics

Based on

- "*Reinforcement Learning: An Introduction*" by Sutton and Barto (Introduction +Tabular Solution Methods)
- Introduction to Reinforcement Learning Lecture series by David Silver (1-5)

Workshop 1: An introduction and tabular methods

In the next two weeks …

- Workshop 2: Approximation methods
  - (Introduction to Approximation Methods)
- Workshop 3: Approximation methods
  - (A Deeper Look into Approximation Methods)

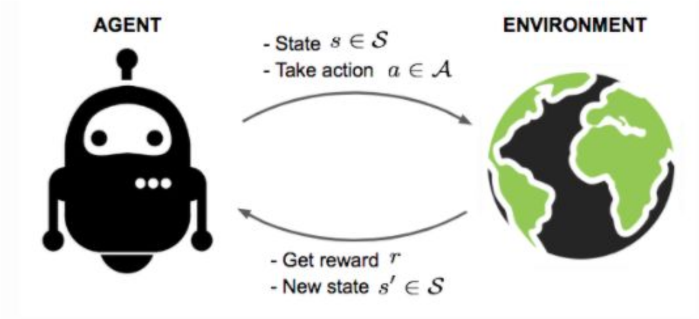
Slides and notebooks: https://netbrainml.github.io/workshop/

Introduction to Reinforcement Learning

What is Reinforcement Learning?

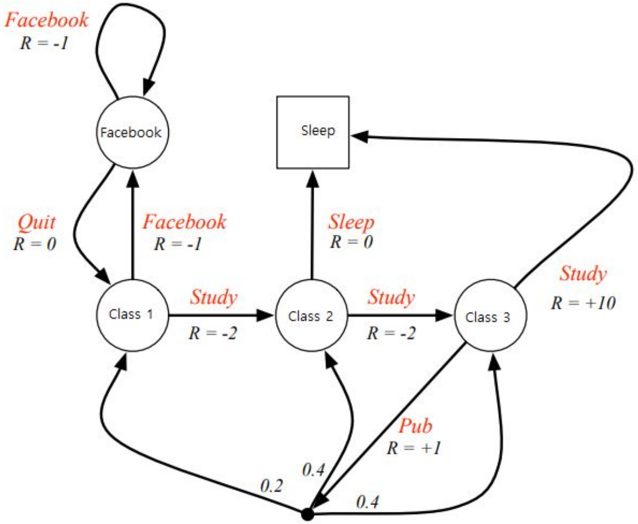
Learning how to optimally interact with a given environment.

1. An agent interacts with the environment by performing actions.
2. The environment returns a reward signal, and the next observation upon agent's action.
3. The agent must learn how to behave in an environment to maximize cumulative reward.


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Markov Decision Process



Formalize RL with MDP

All states have Markovian property

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1},s_{t-2}...s_0)$$

We define MDPs as…

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

S: set of states
A: set of actions
P: transition probability function

R: reward function
γ: discount factor

MDPs in Practice

Notice that MDPs work with states (which have Markovian property)

In practice, environments do not provide **states** to the agent, they give **observations** (which cannot be assumed to be Markovian, nor complete)

Thus, we can use the formalisms of partially observable MDP (POMDP).

The agent must create the state representation! In other words, the agent must have some mapping f, such that **f(observation) = state**.

This mapping can be complete history, RNNs, beliefs…

Elements of RL

RL contains two components:

1. Environment
    a. Model (Known or unknown)
2. Agent (model + one or more subcomponents)
    a. Model (Model-based)
    b. Policy (Policy-based or Actor-Critic)
    c. Value Function (Value-based or Actor Critic)

Model

The model is the behavior of the environment upon interactions.
Composed of two components:

Dynamics is the probability of going to state s’ and receiving reward r, given the agent is in state s and takes the action a.

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

Reward function is a mapping from the state, action to a scalar value.

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r|s, a)$$

If the agent has a model of the environment, then the agent is considered **model-based**. Otherwise, it is **model-free** approach.

Policy

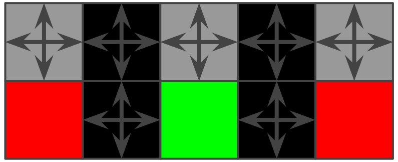
The policy is a function used to map state to action.

$$\pi(s) = a$$

It does not need to be deterministic, instead we can have a stochastic policy.

$$\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$$

Consider an environment as such:



▨ Local Action (0)

🟩 Win (+1)

🟥 Lose (-1)

⬛ Global Action (0)

Additionally, we can consider another trade-off:

Exploration vs exploitation:

Choose to explore and learn more about the environment

Exploit what is already known to be optimal

Value Function

The value function tells us how good a state and/or action is.

Return is the expected (discounted by $0 \leq \gamma \leq 1$) cumulative rewards at a given a trajectory. $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2} \ldots S_{t+k}, A_{t+k}, R_{t+k+1}, S_{t+k+1} \ldots)$

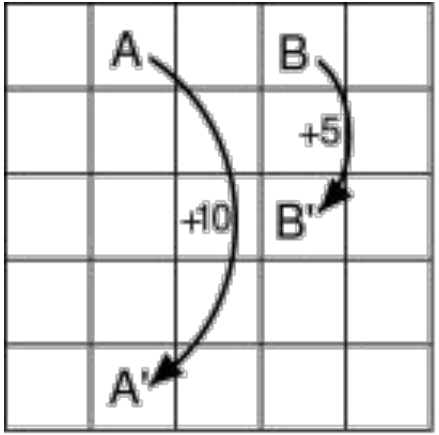
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

We can define two value functions (V, Q) where the input can be either state or state-action pair.

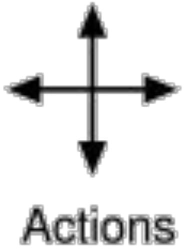
$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

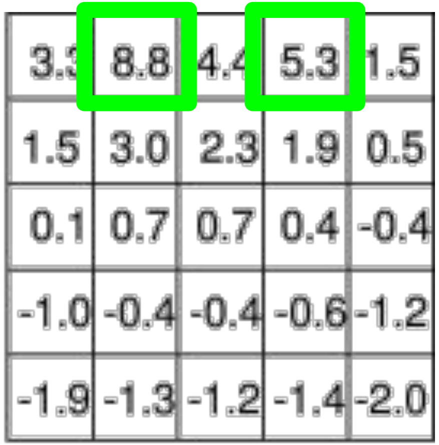
$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Gridworld and state-value function V(s)



(a)





3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

Bellman Equations

Notice that we can recursively define return

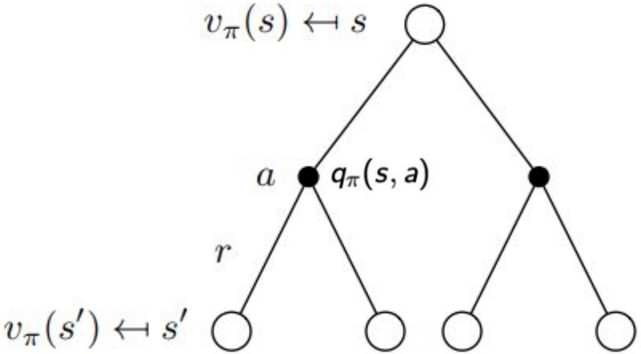
$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Thus, we can recursively define our value functions (Bellman Equations)

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ldots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \end{aligned}$$

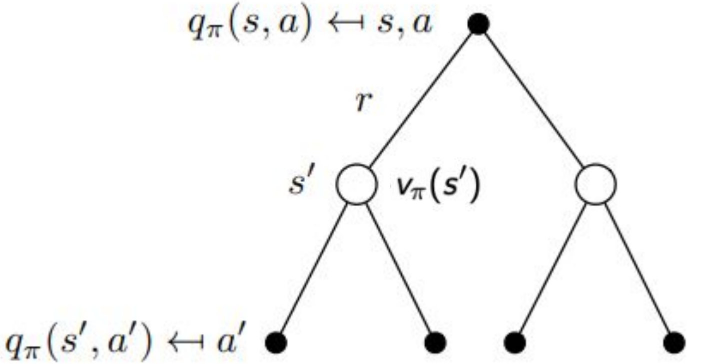
$$\begin{aligned} Q(s, a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a] \end{aligned}$$

Backup Diagrams



$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$



$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

We obtain the Bellman Expectation Equations

Optimality

Optimal equations defines the value functions and policy to maximize the expected cumulative return. Thus, $\pi' \geq \pi$ iff $V_{\pi'}(s) \geq V_{\pi}(s)$ for all s in S.

Optimal policy functions:

$$\pi_* = \arg\max_{\pi} V_\pi(s) = \arg\max_{\pi} Q_\pi(s, a)$$

Bellman Optimality Equations for Value:

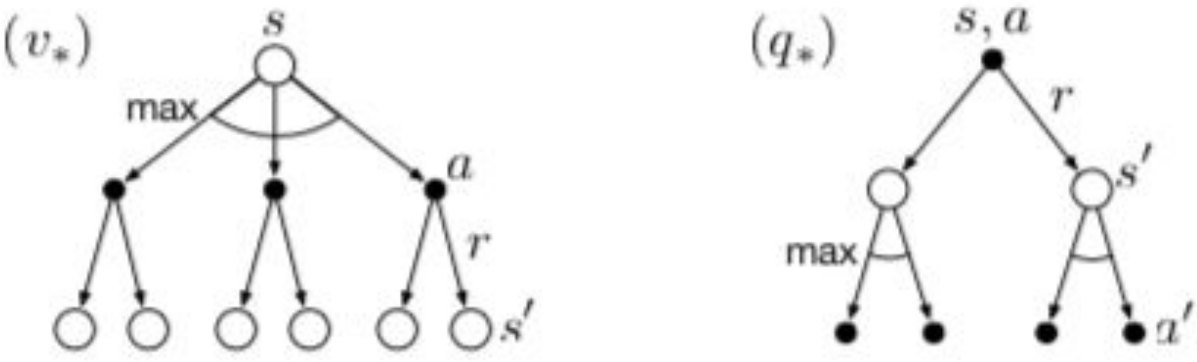
$$V_*(s) = \max_{\pi} V_\pi(s)$$

$$Q_*(s, a) = \max_{\pi} Q_\pi(s, a)$$

$$V_*(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r | s, a) V_*(s')\right)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} P(s', r | s, a) \; V_*(s')$$

Backup Diagrams for Optimal Value Functions



Objectives in RL

Learning: Agent learns a policy by interacting with an unknown model
Planning: Agent learns a policy with an known model

Prediction: Evaluate how good a policy is
    If we follow a fixed policy, how much reward will we get?
    ie) We can obtain a value function from prediction.

Control: Optimizes for policy for maximize return
    What is the best policy given a value function?

Notebook (Introduction)

Tabular Methods Introduction

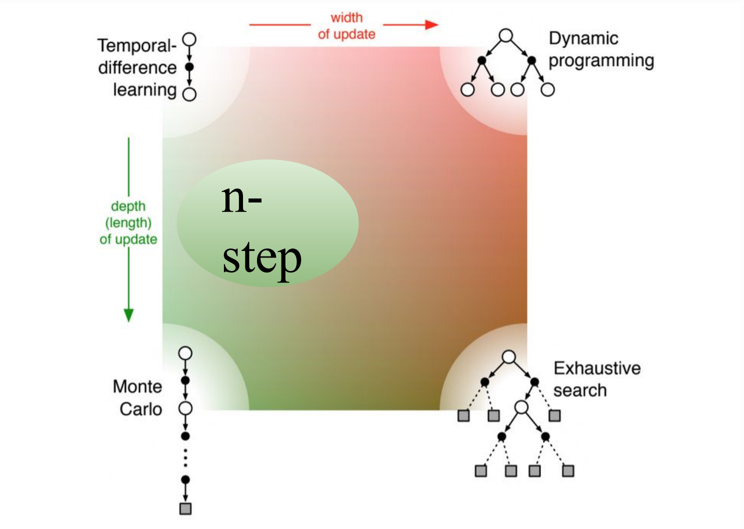
Basic Framework for Tabular Methods

Build up optimal policy/value function that maximizes cumulative expected return by storing past behaviors into arrays or tables.

Each entry would contain the current approximation for its respective state/value.

We will cover some algorithms utilizing:

Dynamic Programming
Monte Carlo Methods
Temporal-Difference Learning
n-step Methods



Tabular Methods with Dynamic Programming

Dynamic Programming

DP is used in RL as a **model-based** approach for **planning**.

In general, DP refers to devising overlapping subproblems, and using the optimal solutions to the subproblems build up to the overall solution.

Recall the recursive nature of the Bellman equation and stored value function

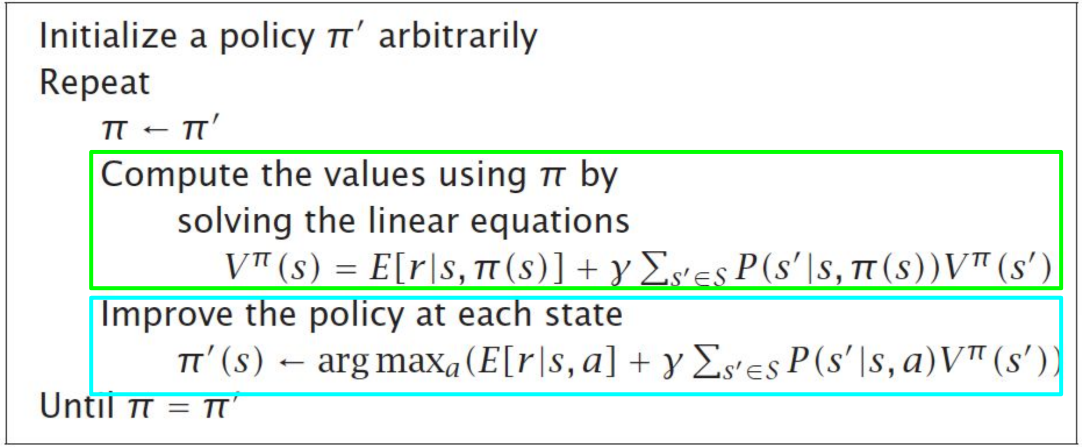
Subproblem 1: Prediction (Evaluate the current policy and obtain a value function)
Subproblem 2: Control (Improve upon the policy using the value function)

We will discuss two methods: Policy Iteration and Value Iteration.

Policy Iteration

Iterate policy evaluation and policy improvement until improvement stops.

- Policy evaluation computes the value function for the given policy
  - Uses Bellman Expectation Equation for Value
- Policy improvement uses the optimal policy equation to obtain the policy
  - Greedily acts on value function to form a better policy



```
Initialize a policy $\pi'$ arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using $\pi$ by
        solving the linear equations
            $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V^{\pi}(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s'))$
Until $\pi = \pi'$
```

Policy evaluation

Policy Improvement

Policy Evaluation

$$V_{t+1}(s) = \mathbb{E}_\pi[r + \gamma V_t(s')|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)(r + \gamma V_t(s'))$$

We can take an iterative approach using the expected update.

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

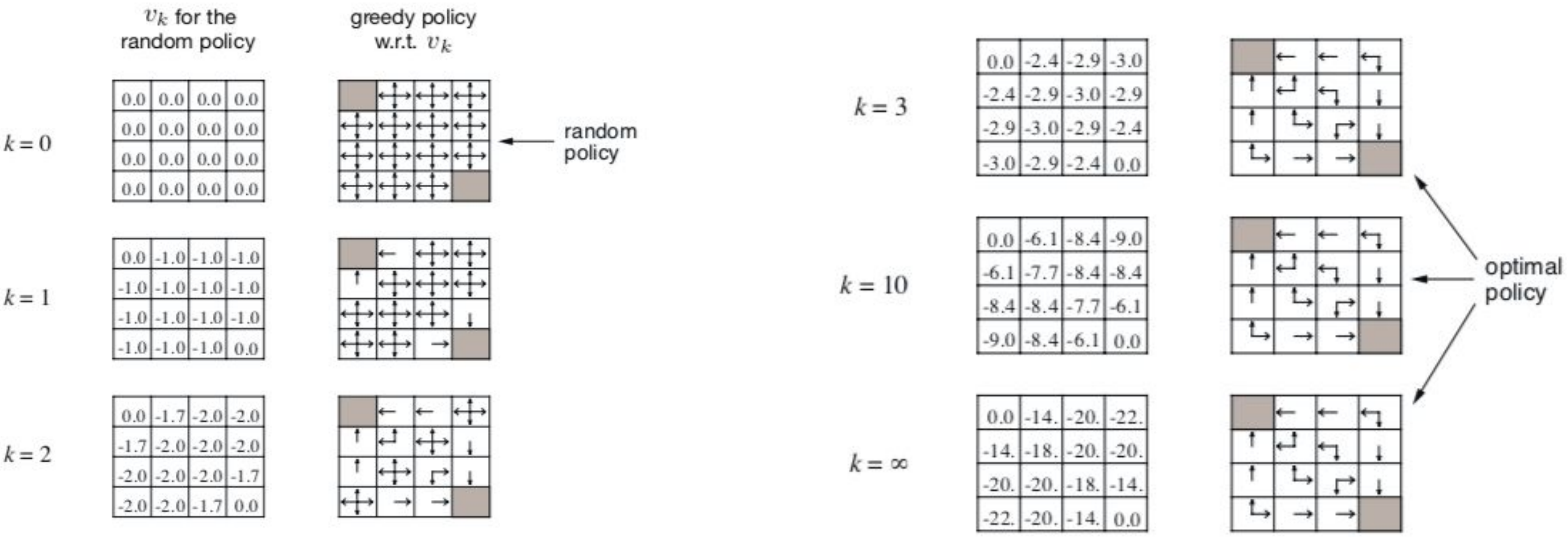
We show that $v_k \rightarrow v^*$ as $k \rightarrow \infty$, where each k is an iteration/sweep across all states.

How do we know when we obtained the value function for the given policy?

Enforce ε-convergence for stopping condition is $[\Sigma_s|V'(s) - V(s)| < \varepsilon]$

Iterate k times over all states to obtain V(s)

Visualization of Iterative Policy Evaluation



$k = 0$

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

$k = 3$

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

Policy Improvement

$$\begin{aligned}\pi'(s) &\doteq \operatorname*{argmax}_a q_\pi(s,a)\\ &= \operatorname*{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]\\ &= \operatorname*{argmax}_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big],\end{aligned}$$

We can act greedily upon the value function to obtain our updated policy.

How is the updated policy more optimal than the past policy?

We have to show the following equation are true for all states.

$$v_{\pi'}(s) \geq v_\pi(s).$$

So we know that,

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

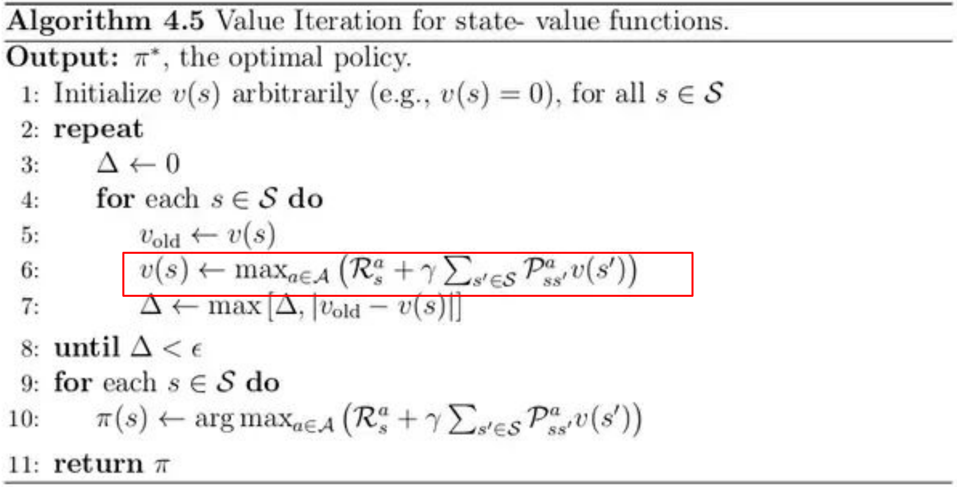
$$\begin{aligned}v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]\\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right]\\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right]\\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)\end{aligned}$$

Value Iteration

Find optimal value function first using an update rule that combines policy improvement and truncated policy evaluation (updates V(s) after one sweep).

Update value function using Bellman Optimality Equation

Then obtain the policy by acting greedily on the optimal value function.



```
Algorithm 4.5 Value Iteration for state- value functions.
Output: $\pi^*$, the optimal policy.
Initialize $v(s)$ arbitrarily (e.g., $v(s) = 0$), for all $s \in \mathcal{S}$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
        $v_{\text{old}} \leftarrow v(s)$
        $\boxed{v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s')\right)}$
        $\Delta \leftarrow \max\left[\Delta, |v_{\text{old}} - v(s)|\right]$
until $\Delta < \epsilon$
for each $s \in \mathcal{S}$ do
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v(s')\right)$
return $\pi$
```

Notebook
(Dynamic Programming)

Tabular Methods with Monte Carlo Methods

Monte Carlo Methods

MC methods are **model-free** approaches that learn off of **complete episodes**, calculating and using the expected return to obtain the value and policy.

Note: Works only for **episodic** MDPs = Has a terminal state

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{k=T} \gamma^k R_{t+k+1},$$

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

Prediction/Policy Evaluation: Obtain the value function through trial and error

Control: Obtain the policy from the value function

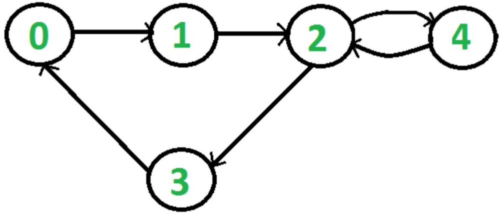
Policy Evaluation with MCM

Run an episode with some policy, and then calculate the return G at each time step

  At each timestep t, say we are at state s

  Append G_t to Returns[s]

V(s) = average(Returns[s])



Two options to estimating value:

  First-Visit MC: Include return of only the **first** visit of the state in each episode

  Every-Visit MC: Include returns of **all** visits of the state in each episode

How do we make sure we update every state? (Exploration)

  Exploring starts: Have nonzero prob. of starting at any state

  ε-greedy/ε-soft policies: Have nonzero prob. of selecting a random action.

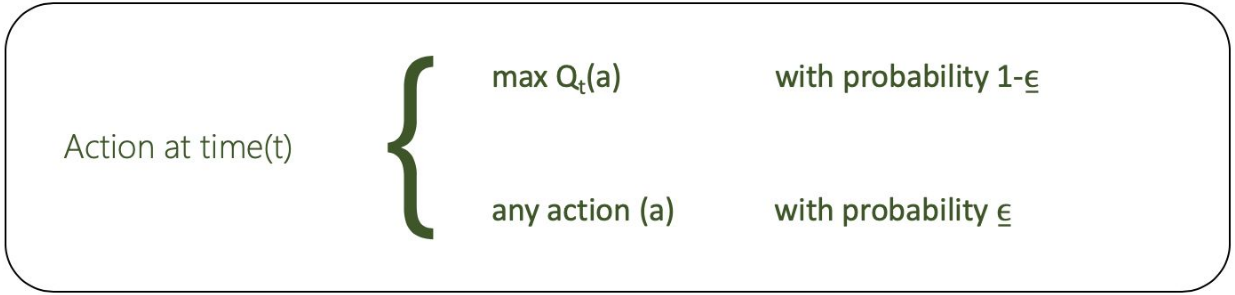
  Have two ε-soft policies (One for exploration, one for exploitation)

ε-soft vs ε-greedy

ε-soft: Allow each action in every state have some minimal chance (>0) of being selected

ε-greedy is a form of ε-soft:

Have some probability (=ε/|A| where A is the set of all actions) of choosing any action, and have a probability (=1-ε+ε/|A|) of choosing the greedy action.



$$\text{Action at time(t)} \begin{cases} \max Q_t(a) & \text{with probability } 1-\epsilon \\ \text{any action (a)} & \text{with probability } \epsilon \end{cases}$$

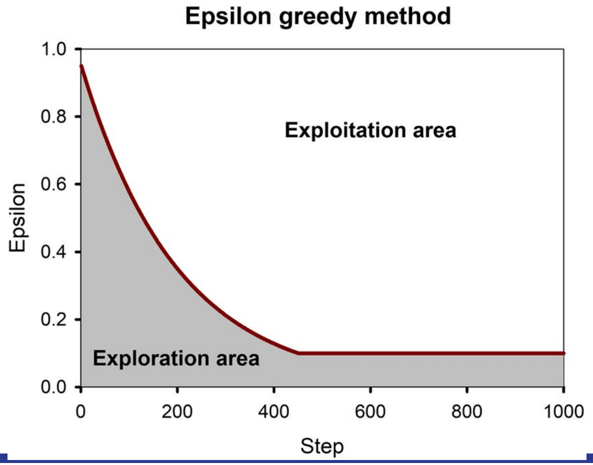
GLIE : Greedy in the Limit with Infinite Exploration

Given an ε-greedy policy,

We assume all state-action pairs are explored infinite times, and make the policy converge to a greedy policy

To implement this, ε can decay over time.

For example, we set ε = 1/k, where k is the number of episode already updated the value



[0] Sajedian, Iman & Lee, Heon & Rho, Junsuk. (2019). Double-deep Q-learning to increase the efficiency of metasurface holograms. Scientific Reports. 9. 10899. 10.1038/s41598-019-47154-z.

Off-policy Approach

“Learning about one way of behaving, called the target policy, from data generated by another way of selecting actions, called the behavior policy” (Maei)

Have a policy for exploration and a policy for exploitation

Use a policy to create samples (called the behavioral policy = B)

Obtain another policy (called the target policy = π) using the samples

Given that the samples are sampled from B,

We want the obtain the expectation of these samples (value) wrt π.

Notice we cannot simply take the average of the samples to obtain the value.

Instead, we must use importance sampling.

[0] Maei, H. R., Szepesv´ari, C., Bhatnagar, S., & Sutton, R. S. (2010) Toward off-policy learning control with function approximation. ICML, pages 719–726

Importance Sampling

We want to find the expectation (value) of the samples X wrt target π

$$\begin{aligned}\mathbb{E}_\pi[X] &\doteq \sum_{x \in X} x\pi(x) \\ &= \sum_{x \in X} x\pi(x)\frac{b(x)}{b(x)} \\ &= \sum_{x \in X} x\frac{\pi(x)}{b(x)}b(x) \quad = \sum_{x \in X} x\rho(x)b(x) \quad = \mathbb{E}_b\left[X\rho(X)\right]\end{aligned}$$

From this derivation, we simply need to find the weighted average over all the samples by the importance sampling ratio ρ

Incremental Updates

Iteratively calculating the mean/value function.

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

For nonstationary problems, we should replace 1/k with a fixed constant **α**

Notebook
(Monte Carlo Methods)

Tabular Methods with Temporal-Difference Learning

Temporal-Difference (TD) Learning

TD methods are **model-free** approaches that learns by **bootstrapping**.

Can update estimation of value function within the next time step

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big] \longleftrightarrow V(S_t) \leftarrow V(S_t) + \alpha\boxed{\Big[\boxed{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)\Big]}$$

TD methods can work off of incomplete trajectories, which is especially useful for continuing or long episodic tasks.

Does TD converge?

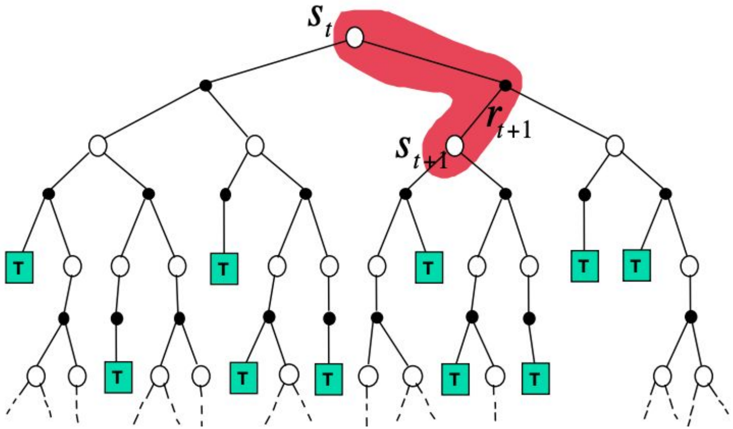
TD(0): Use the equation above to update the value function

TD target in red

TD error in green

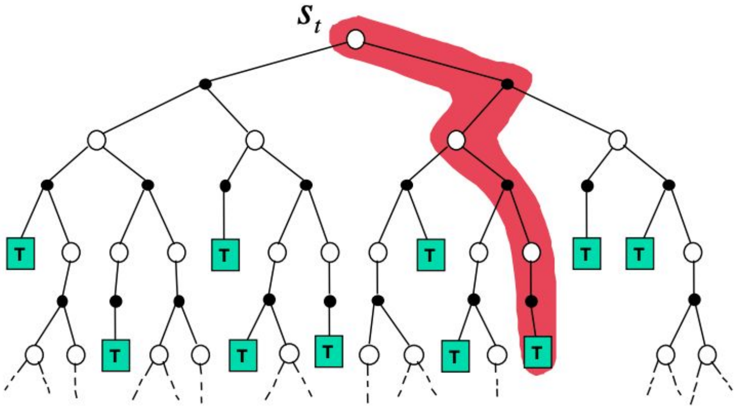
Visualization of TD vs MC

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



TD

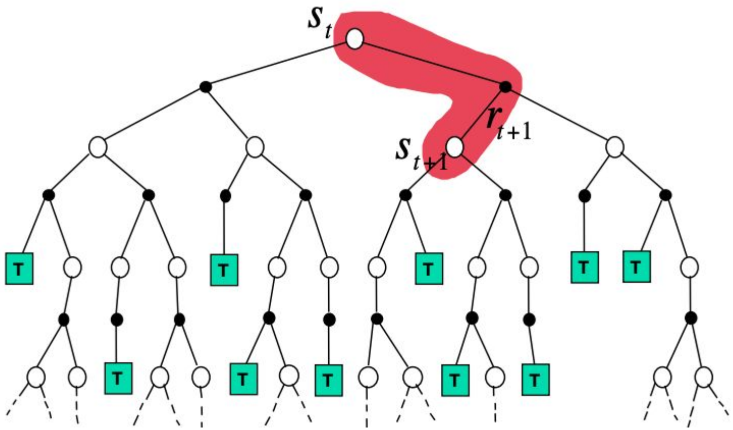
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t)\right)$$



MC

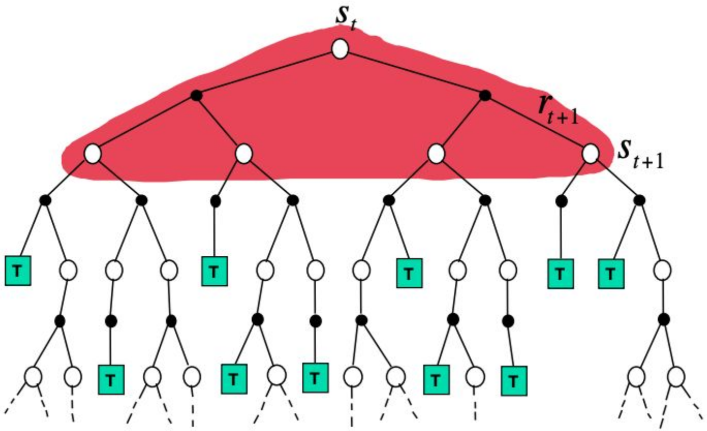
Visualization of TD vs DP

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



TD

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

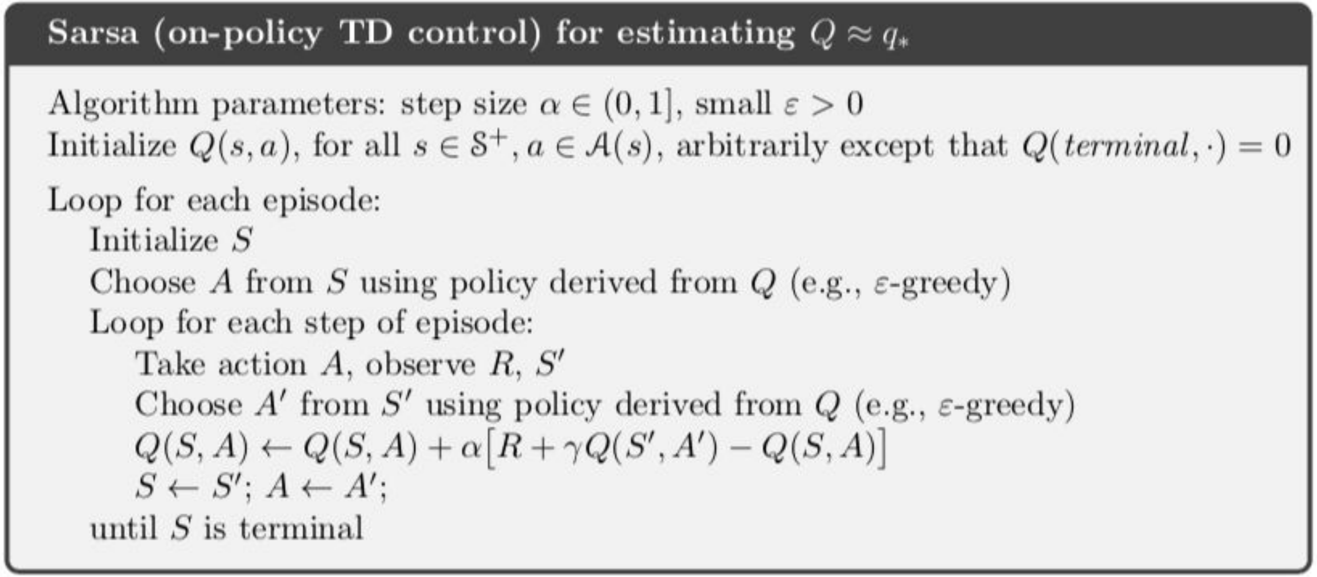


DP

SARSA

On-policy TD control algorithm that updates the state-action value using TD(0)

Converges proven if GLIE conditions are met



```
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., $\varepsilon$-greedy)
   Loop for each step of episode:
      Take action $A$, observe $R$, $S'$
      Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-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
```

Expected SARSA

Take the expectation over all actions for the TD target.

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

Expected SARSA:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \boxed{\gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}]} - Q(S_t, A_t) \Big] \\ &\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \Big], \end{aligned}$$

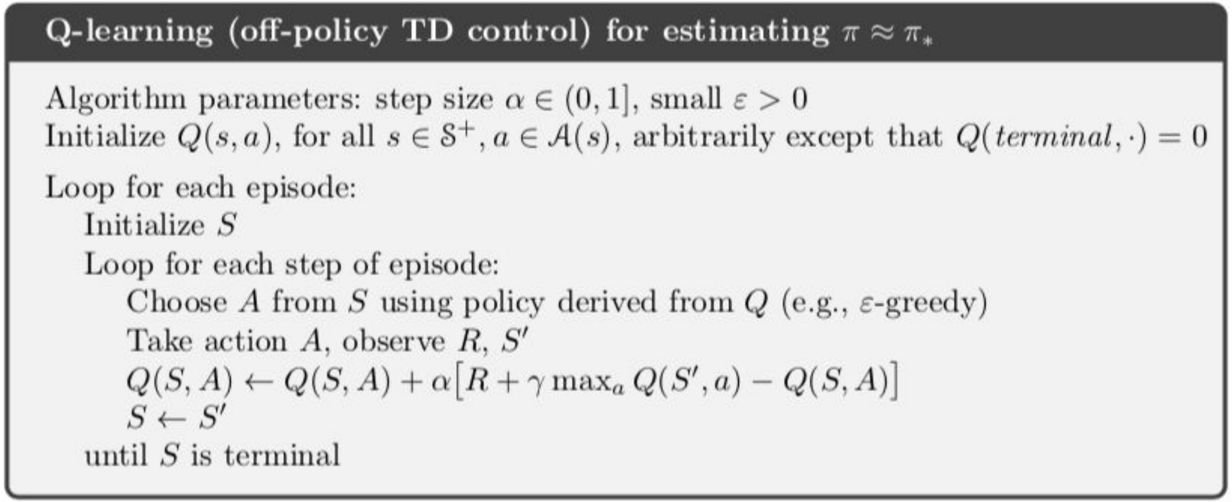
Why use Expected SARSA?

Is Expected SARSA on-policy or off-policy?

Q-Learning

Off-policy TD control that uses an approximate optimal state-action value as TD target:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} \boxed{+ \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t) \Big].$$



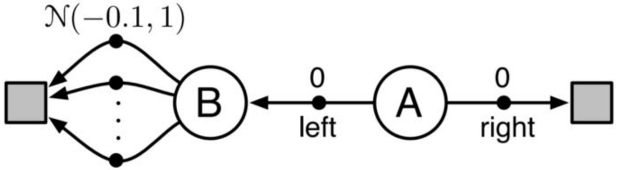
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
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., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Double Learning

For Q-learning, there is **maximization bias**, where the value function is overestimated given some noise/variance in environment or inability to generalize.



Double Q-Learning: Have two independent value function and update one of them on each sample with (prob = 0.5). This will make it less likely that both value functions are overestimating the same action.

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

Notebook
(Temporal Difference Learning)

Tabular Methods with N-step Bootstrapping

n-step Bootstrapping

Instead of updating after every timestep (TD-Learning) or an entire episode (MCM)

Obtain the n-step return:

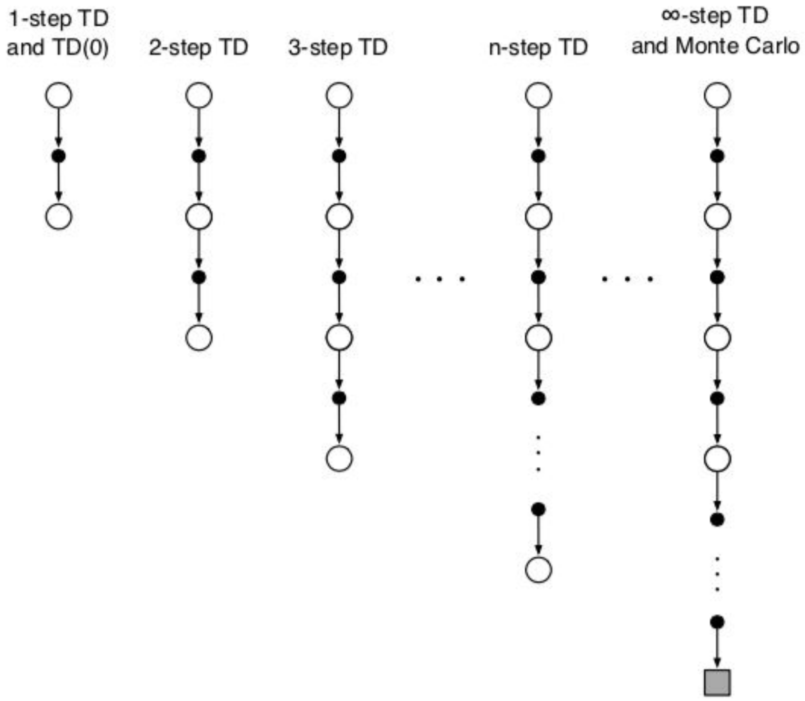
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Update the value function using the n-step return:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]$$

The above procedure is called TD(n)

Backup diagrams for n-step Bootstrapping



n-step SARSA

Recall SARSA updates:

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

So, we take n-steps and estimate the return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

We let the estimated return be the “TD target”

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

For Expected SARSA, n-step bootstrapping would follow similarly to this procedure

Notebook
(n-Step Bootstrapping)