Reinforcement Learning

Jesús Fernández-Villaverde[1] and Galo Nuño[2]

August 7, 2023

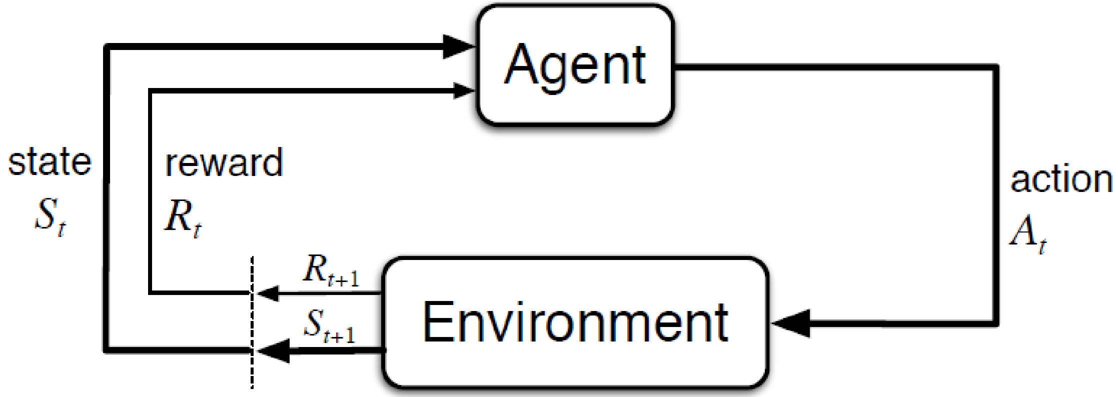
[1]University of Pennsylvania

[2]Banco de España

A short introduction

Reinforcement learning

- Main idea: Algorithms that use training information that evaluates the actions taken instead of deciding whether the action was correct.
- Purely *evaluative feedback* to assess how good the action taken was, but not whether it was the best feasible action.
- Related with approximate dynamic programming.


Agent
state
S_t
reward
R_t
R_{t+1}
S_{t+1}
Environment
action
A_t

Why reinforcement learning?

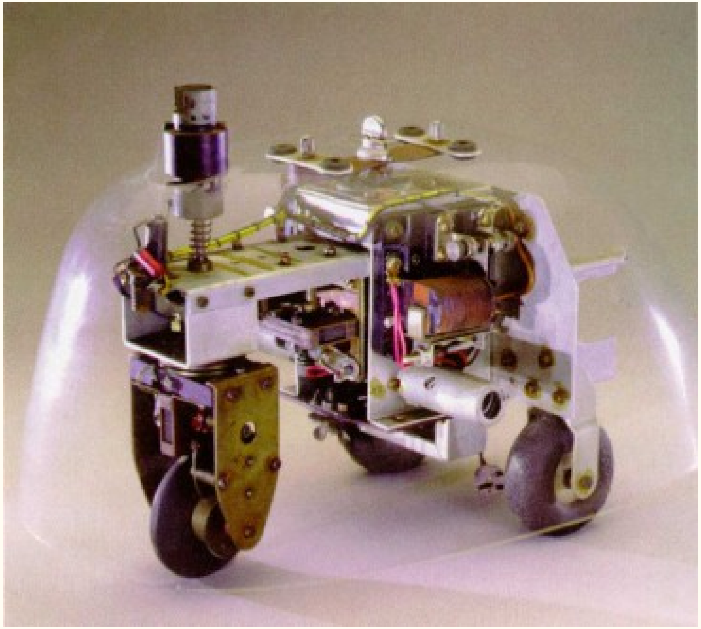
- Useful when:
  1. The dynamics of the state is unkown but simulation is easy: model-free vs. model-based reinforcement learning.
  2. Or the dimensionality is so high that we cannot store the information about the DP in a table.
- Work surprisingly well in a wide range of situations, although no methods that are guaranteed to work.
- Key for success in economic applications: ability to simulate fast (link with massive parallelization). Also, it complements very well with neural networks.

Comparison with alternative methods

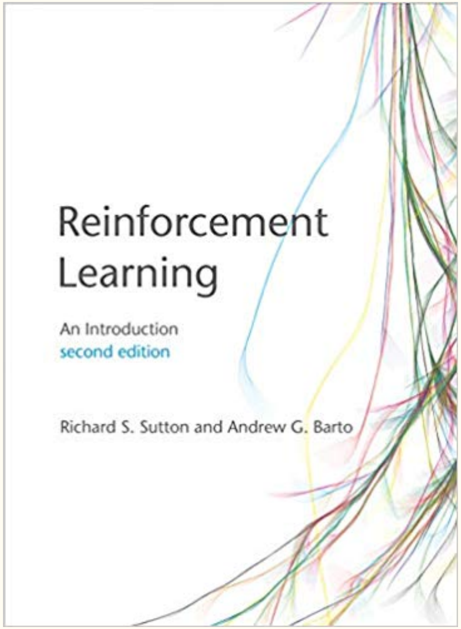
- Similar (same?) ideas are called approximate dynamic programming or neuro-dynamic programming.
- Traditional dynamic programming: we optimize over best feasible actions.
- Supervised learning: purely *instructive feedback* that indicates best feasible action regardless of action actually taken.
- Unsupervised learning: hard to use for optimal control problems.
- In practice, we mix different methods.
- Current research challenges:
    1. How do we handle associate behavior effectively?
    2. Zero- and few-shot learning.

Some history

- Ideas go back to at least Edward Thorndike (1874-1949).
- Grey Walter (1910 -1977)'s "mechanical tortoise" (1951).
- Marvin Minsky (1927-2016)'s 1954 Ph.D. thesis.
- Widrow, Gupta, and Maitra (1973): modified Least-Mean-Square (LMS) algorithm.
- Chris Watkins's development of Q-learning (1989).



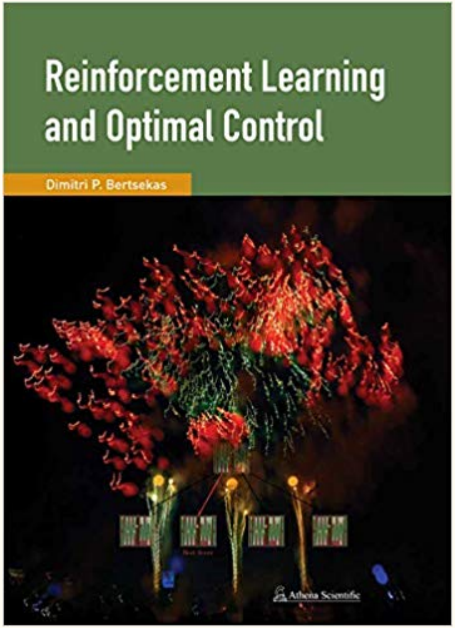
References



Reinforcement Learning

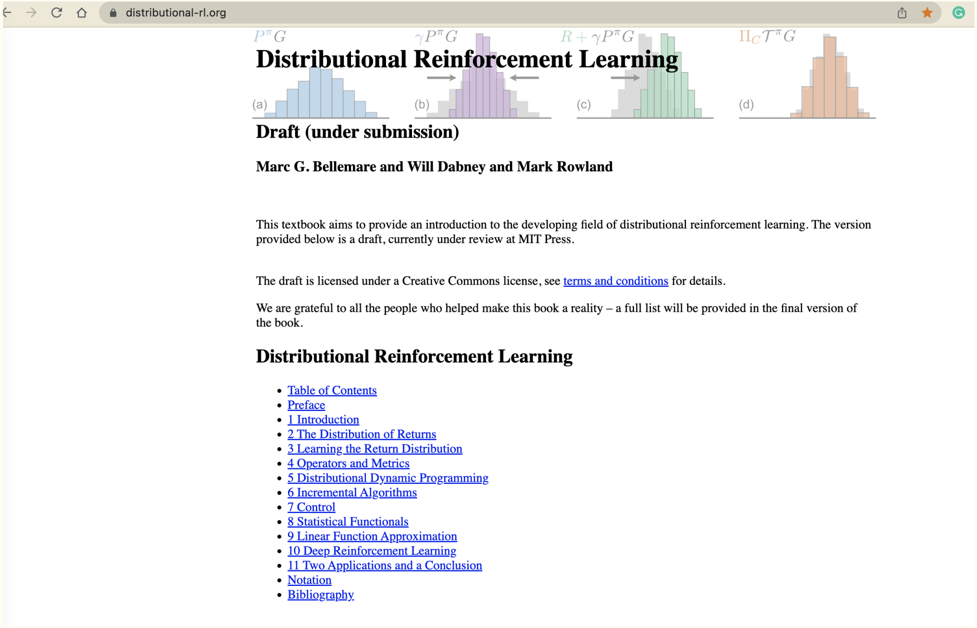
An Introduction
second edition

Richard S. Sutton and Andrew G. Barto


Reinforcement Learning and Optimal Control

Dimitri P. Bertsekas

Athena Scientific



Distributional Reinforcement Learning

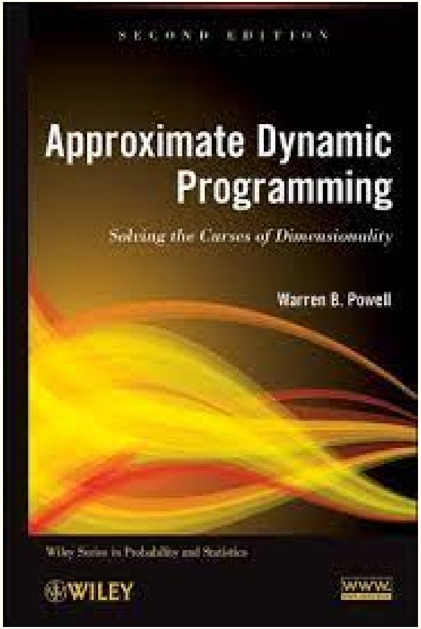
Draft (under submission)

Marc G. Bellemare and Will Dabney and Mark Rowland

This textbook aims to provide an introduction to the developing field of distributional reinforcement learning. The version provided below is a draft, currently under review at MIT Press.

We are grateful to all the people who helped make this book a reality – a full list will be provided in the final version of the book.

Distributional Reinforcement Learning


SECOND EDITION

Approximate Dynamic Programming

Solving the Curses of Dimensionality

Warren B. Powell

Wiley Series in Probability and Statistics

WILEY

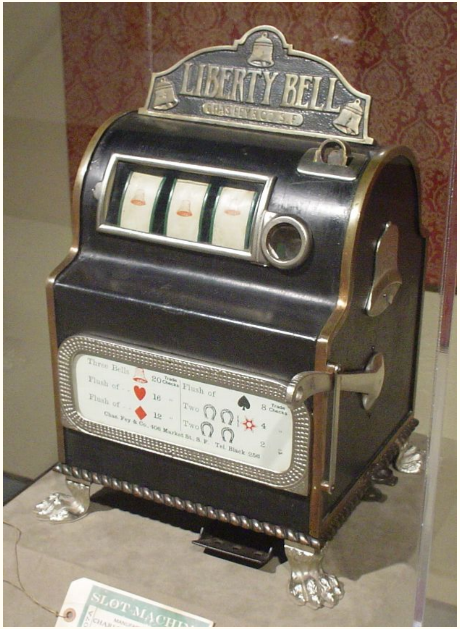
Two applications

- More than a general theory, reinforcement learning is a set of related ideas.
- Thus, I will present two applications taken from Sutton and Barto:
  1. The multi-armed bandit problem.
  2. Dynamic programming.
- Also, for more examples, see:
  1. `http://incompleteideas.net/book/code/code2nd.html` (and the links therein).
  2. `https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver`.
  3. `https://github.com/TikhonJelvis/RL-book/`.

Application I: The multi-armed bandit problem

The multi-armed bandit problem

- You need to choose action a among k available options.
- Each option is associated with a probability distribution of payoffs.
- You want to maximize the expected (discounted) payoffs.
- But you do not know which action is best, you only have estimates of your value function (dual control problem of identification and optimization).
- You can observe actions and period payoffs.
- Go back to the study of "sequential design of experiments" by Thompson (1933, 1934) and Bellman (1956).


LIBERTY BELL
CHAS. FEY & CO. S.F.
Three Bells 20 Trade Checks
Flush of 16
Flush of 12
Flush of 8 Trade Checks
Two 4
Two 2
Chas. Fey & Co. 406 Market St., S. F. Tel. Black 256
SLOT-MACHINE

Theory vs. practice

- You can follow two pure strategies:
  1. Follow *greedy* actions: actions with highest expected value. This is known as *exploiting*.
  2. Follow *non-greedy* actions: actions with dominated expected value. This is known as *exploring*.
- This should remind you of a basic dynamic programming problem: what is the optimal mix of pure strategies?
- If we impose enough structure on the problem (i.e., distributions of payoffs belong to some family, stationarity, etc.), we can solve (either theoretically or applying standard solution techniques) the optimal strategy (at least, up to some upper bound on computational capabilities).
- But these structures are too restrictive for practical purposes outside the pages of *Econometrica*.

A policy-based method I

- Proposed by Thathachar and Sastry (1985).
- A very simple method that uses the averages $Q_n(a)$ of rewards $R_i(a), i = \{1, ..., n\}$, actually received:

$$Q_n(a) = \frac{1}{n} \sum_{i=1}^{n-1} R_i(a)$$

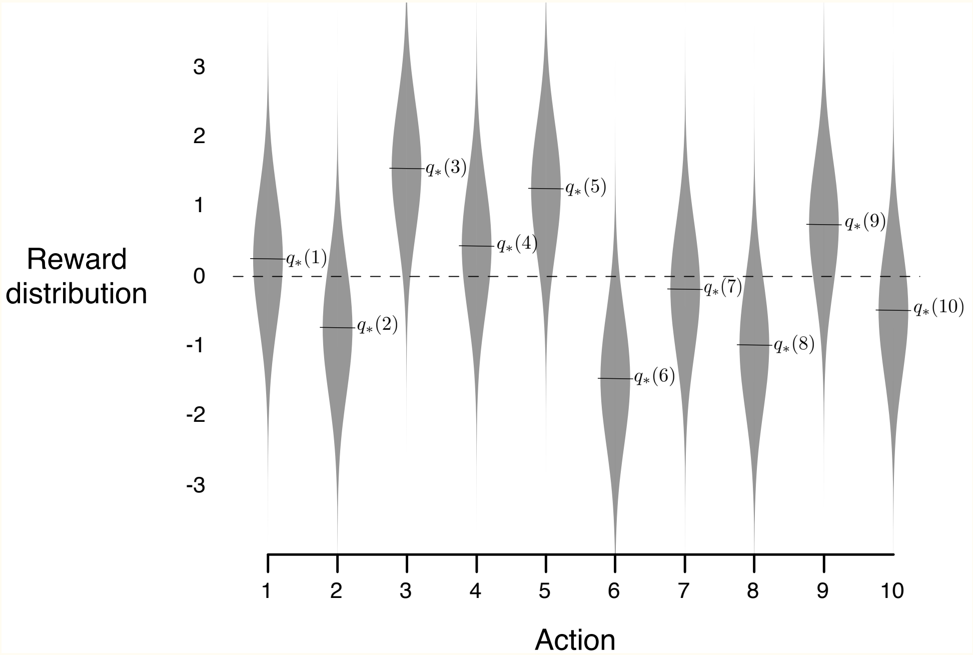
- We start with $Q_0(a) = 0$ for all k. Here (and later), we randomize among ties.
- We update $Q_n(a)$ thanks to the nice recursive update based on linearity of means:

$$Q_{n+1}(a) = Q_n(a) + \frac{1}{n} \left[R_n(a) - Q_n(a) \right]$$

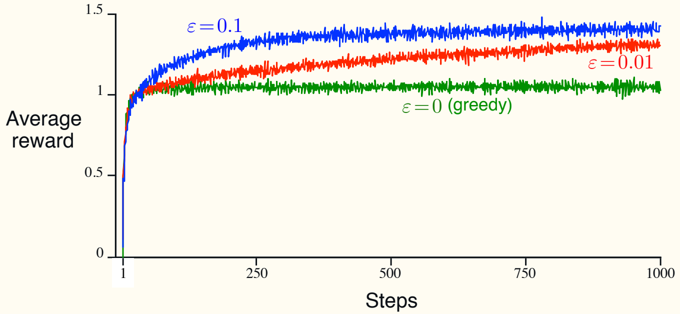
Averages of actions not picked are not updated.

A policy-based method II

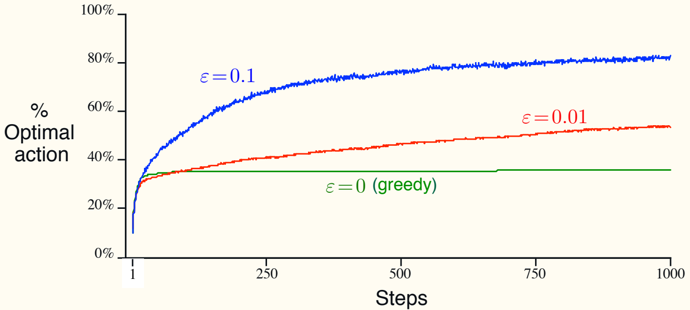
- How do we pick actions?
  1. Pure greedy method: $\arg\max_a Q_t(a)$.
  2. ϵ-greedy method. Mixed best action with a random trembling.
- Easy to generalize to more sophisticated strategies.
- In particular, we can connect with genetic algorithms (AlphaGo).



3
2
1
0
-1
-2
-3
Reward distribution
$q_*(1)$
$q_*(2)$
$q_*(3)$
$q_*(4)$
$q_*(5)$
$q_*(6)$
$q_*(7)$
$q_*(8)$
$q_*(9)$
$q_*(10)$
1
2
3
4
5
6
7
8
9
10
Action



$\varepsilon = 0.1$
$\varepsilon = 0.01$
$\varepsilon = 0$ (greedy)
Average reward
1.5
1
0.5
0
1
250
500
750
1000
Steps



$\varepsilon = 0.1$
$\varepsilon = 0.01$
$\varepsilon = 0$ (greedy)
% Optimal action
100%
80%
60%
40%
20%
0%
1
250
500
750
1000
Steps

A more general update rule

- Let us think about a modified update rule with $\alpha \in (0,1]$:

$$Q_{n+1}(a) = Q_n(a) + \alpha \left[R_n(a) - Q_n(a)\right]$$

- This is equivalent, by recursive substitution, to:

$$Q_{n+1}(a) = (1-\alpha)^n Q_1(a) + \alpha \sum_{i=1}^{n-1} \alpha(1-\alpha)^{n-i} R_i(a)$$

- Better rule to think about non-stationary problems.
- We can also have a time-varying $\alpha_n(a)$, but, to ensure convergence with probability 1 as long as:

$$\sum_{i=1}^{\infty} \alpha_n(a) = \infty$$

$$\sum_{i=1}^{\infty} \alpha_n^2(a) = \infty$$

Improving the algorithm, I

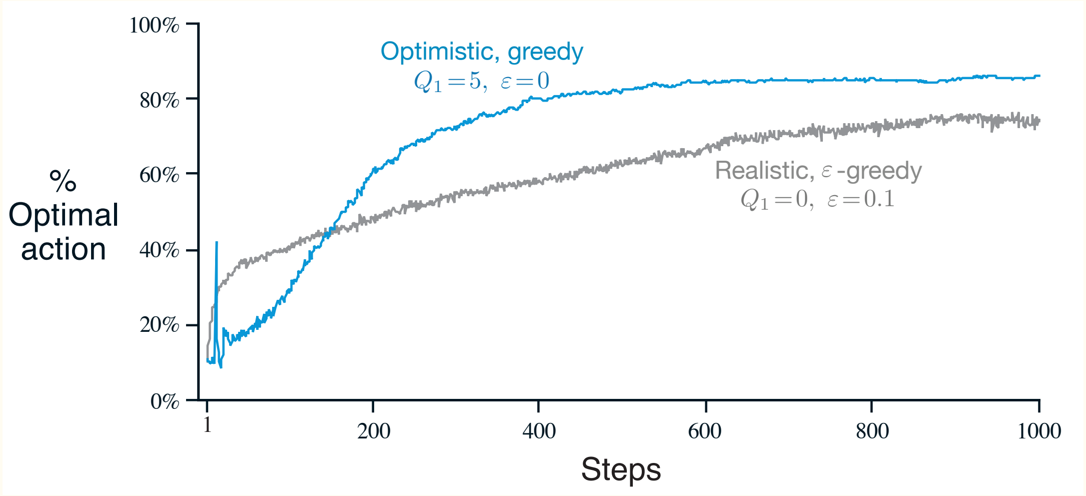
- We can start with "optimistic" Q_0 (i.e., a biased estimates of the returns) to induce exploration.
    - Intuition: most initial choices will be "disappointing" and decision-maker will switch to alternatives.
- We can implement an upper-confidence-bound action selection:

$$\arg\max_a \left[Q_n(a) + c\sqrt{\frac{\log n}{N_n(a)}} \right]$$

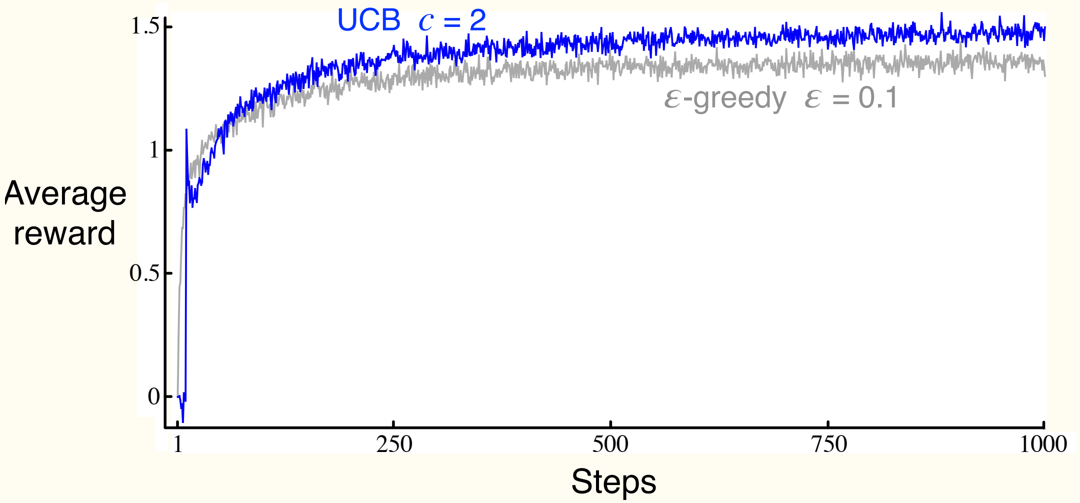
where $N_n(a)$ is how many times action a has been picked before.

- Intuition: we pick non-greedy actions depending on the uncertainty of their estimate. All actions are selected in some moment, but actions with lower value estimates, or that have already been selected frequently, will be selected with lower frequency as n grows.

Optimistic choice



Upper-confidence-bound choice



Improving the algorithm, II

- We can have a gradient bandit algorithms based on a softmax choice:

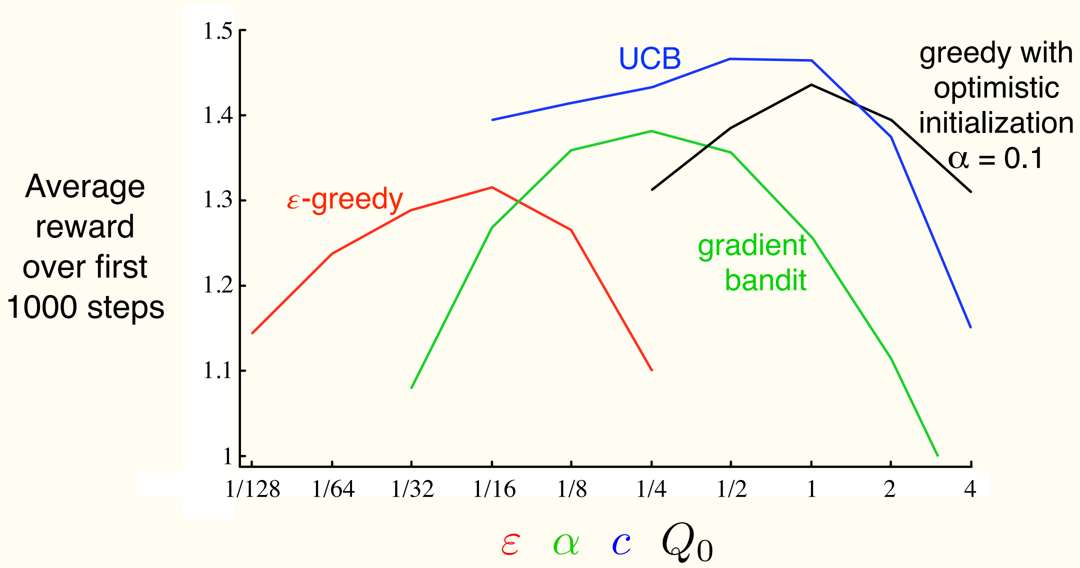
$$\pi_n(a) = \mathrm{P}(A_n = a) = \frac{e^{H_n(a)}}{\sum_{b=1}^{k} e^{H_n(b)}}$$

where

$$\begin{aligned} H_{n+1}(A_n) &= H_n(A_n) + \alpha(1 - \pi_n(A_n))\left(R_n(a) - \overline{R}_n\right) \\ H_{n+1}(a) &= H_n(a) - \alpha\pi_n(a)\left(R_n(a) - \overline{R}_n\right) \text{ for all } a \neq A_n \end{aligned}$$

- This is a slightly hidden version of an SGD algorithm.

A comparison



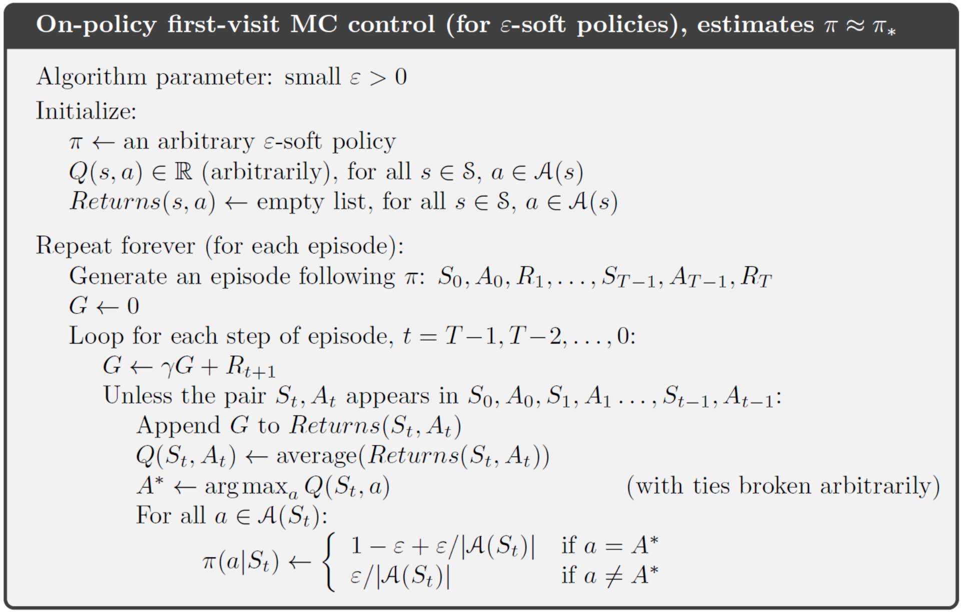
Application II: Dynamic programming

Dynamic programming

- How does all of this apply to more complex problems such as dynamic programming?
- A quick review of existing algorithms:
    1. Policy function iteration.
    2. Value function iteration.
    3. Asynchronous versions of both.
    4. Generalized policy iteration.
    5. Projection and perturbation (different perspective).

Reinforcement learning dynamic programming algorithms, I

- Monte Carlo prediction.
- We just simulate an arbitrary policy and compute rewards.
- Then, from time to time, we update the policy given our estimate of the value function.
- As with the bandit problem, we can introduce some randomness in the policy function (ϵ-greedy policies).
- Useful when:
    1. Describing the environment is hard (or impossible!). Think about card games with all their alternatives.
    2. We want to concentrate on the exploration of a subset of state values.



On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

Off-policy MC control, for estimating $\pi \approx \pi_*$

```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$    (with ties broken consistently)

Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$    (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

Reinforcement learning dynamic programming algorithms, II

- Temporal-difference (TD) learning:

$$V^{n+1}(s_t) = V^n(s_t) + \alpha\left(r_{t+1} + \beta V^n(s_{t+1}) - V^n(s_t)\right)$$

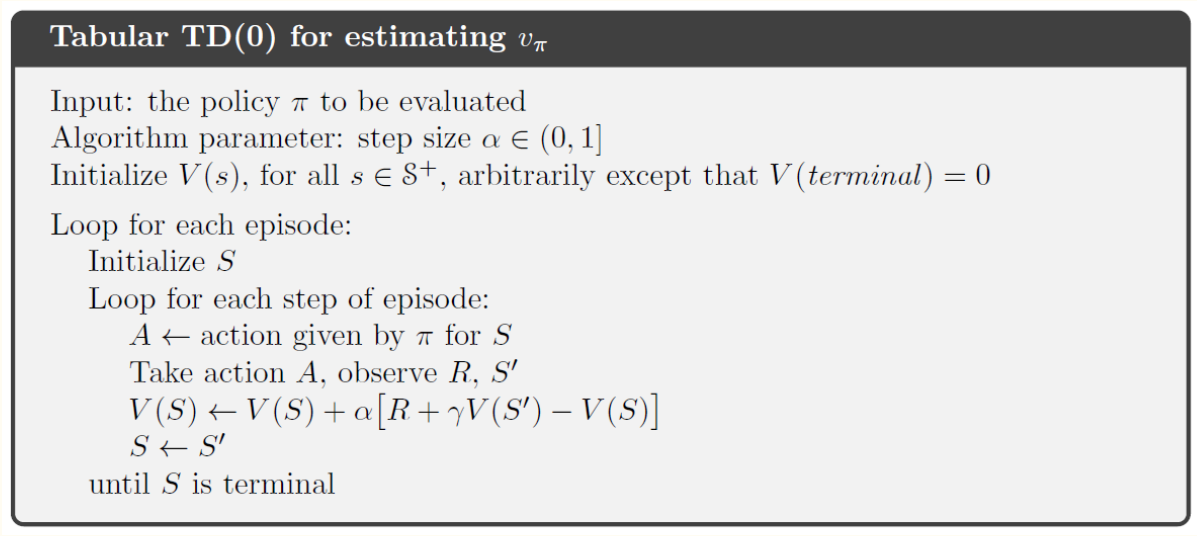
- SARSA $\Rightarrow$ On-policy TD control:

$$Q^{n+1}(a_t, s_t) = Q^n(a_t, s_t) + \alpha\left(r_{t+1} + \beta Q^n(a_{t+1}, s_{t+1}) - Q^n(a_t, s_t)\right)$$

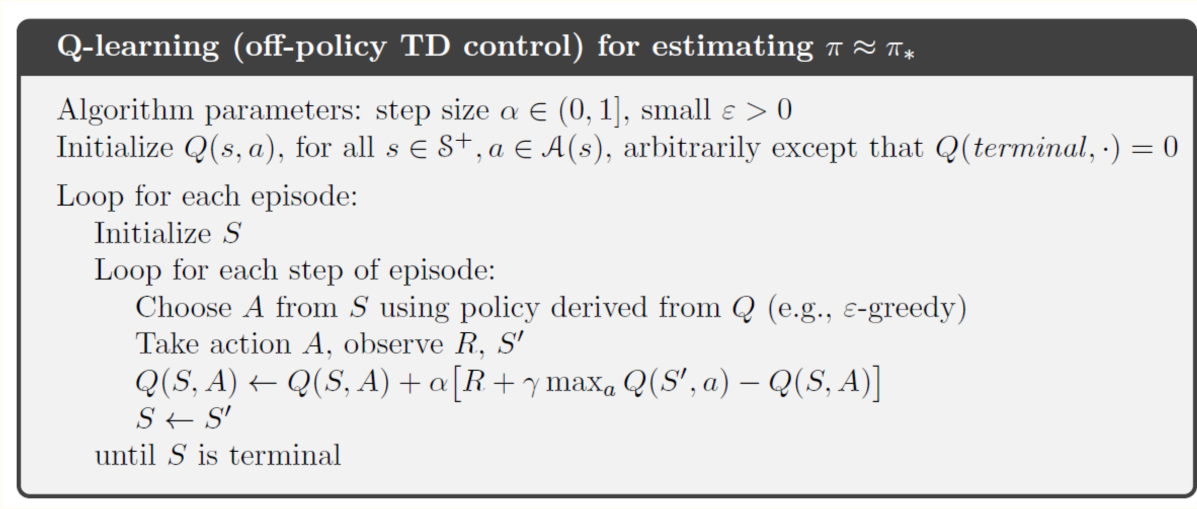
  Definition of $Q^n(a_t, s_t)$.

- Q-learning $\Rightarrow$ Off-policy TD control:

$$Q^{n+1}(a_t, s_t) = Q^n(a_t, s_t) + \alpha\left(r_{t+1} + \beta \max_{a_{t+1}} Q^n(a_{t+1}, s_{t+1}) - Q^n(a_t, s_t)\right)$$

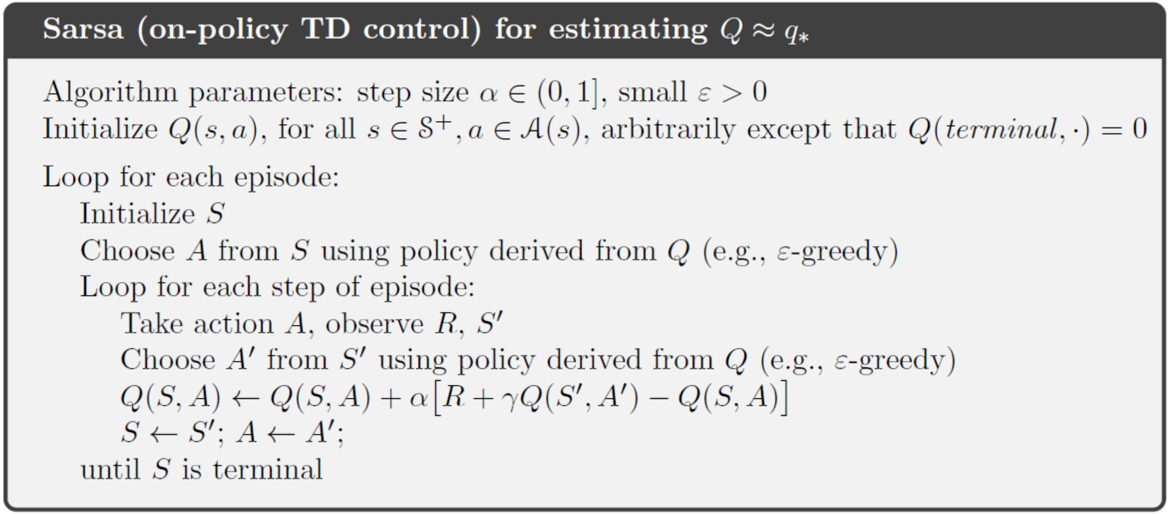


```
Tabular TD(0) for estimating $v_\pi$

Input: the policy $\pi$ to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```



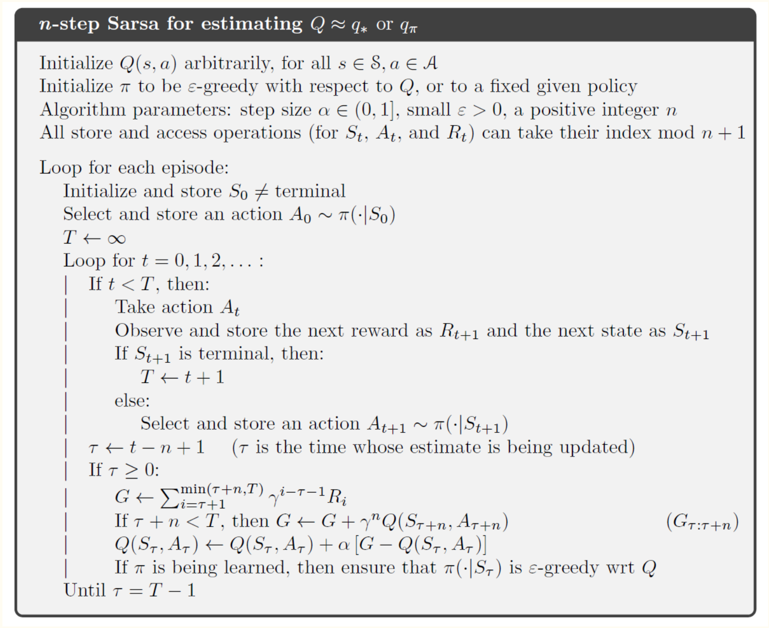
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
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., $\varepsilon$-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
```



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
```

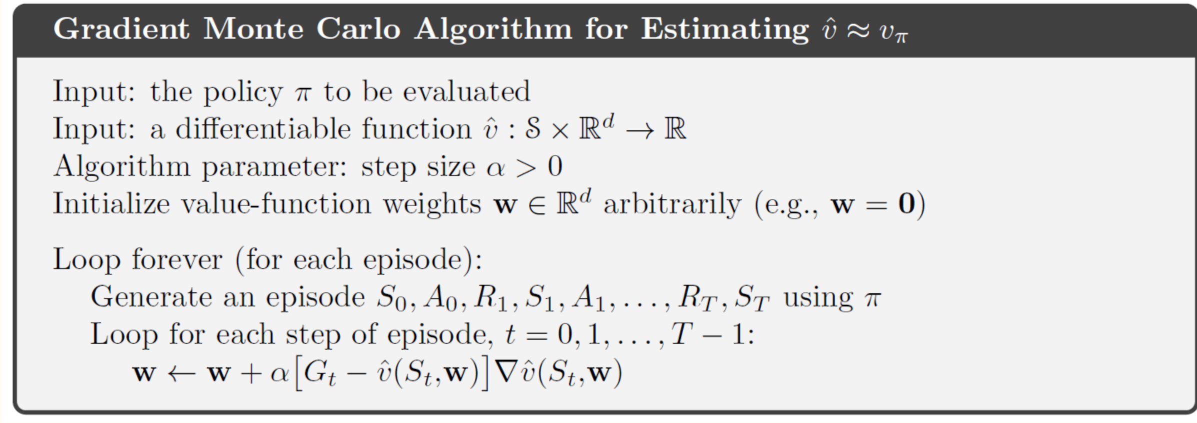


n-step Sarsa for estimating $Q \approx q_*$ or q_π

```
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $\pi$ to be $\varepsilon$-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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$
    |   $\tau \leftarrow t - n + 1$   ($\tau$ 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 $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$        $(G_{\tau:\tau+n})$
    |       $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$
    |       If $\pi$ is being learned, then ensure that $\pi(\cdot|S_\tau)$ is $\varepsilon$-greedy wrt $Q$
    Until $\tau = T - 1$
```

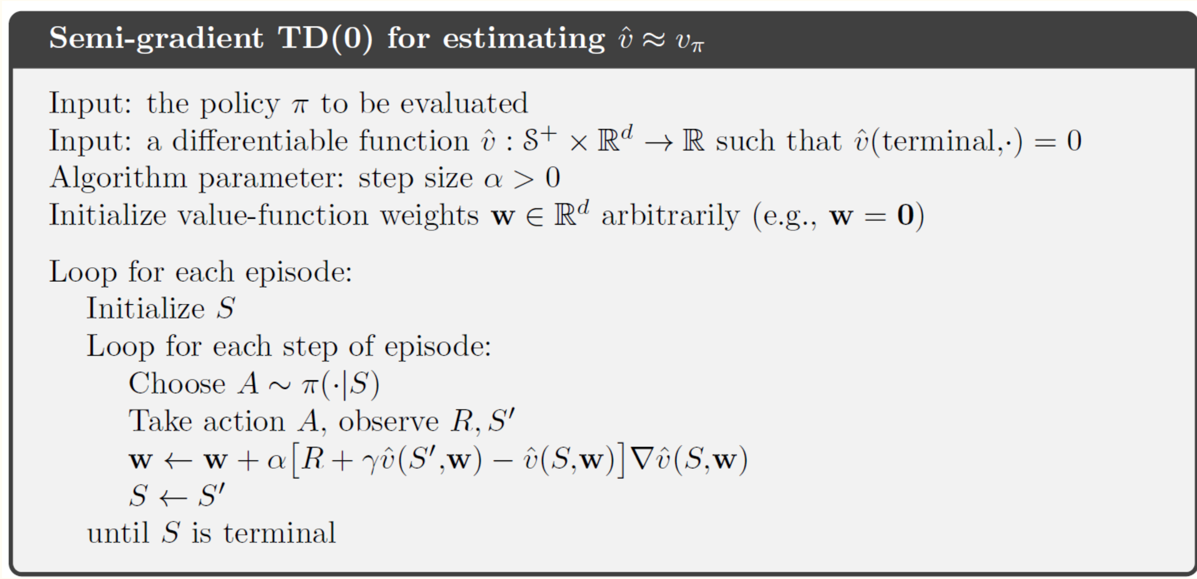
Reinforcement learning dynamic programming algorithms, III

- Value-based methods: applications of deep learning.
- Policy-gradient methods: again, deep learning.
- Actor-critic methods.



Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using $\pi$
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \big[ G_t - \hat{v}(S_t,\mathbf{w}) \big] \nabla \hat{v}(S_t,\mathbf{w})$
```



Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

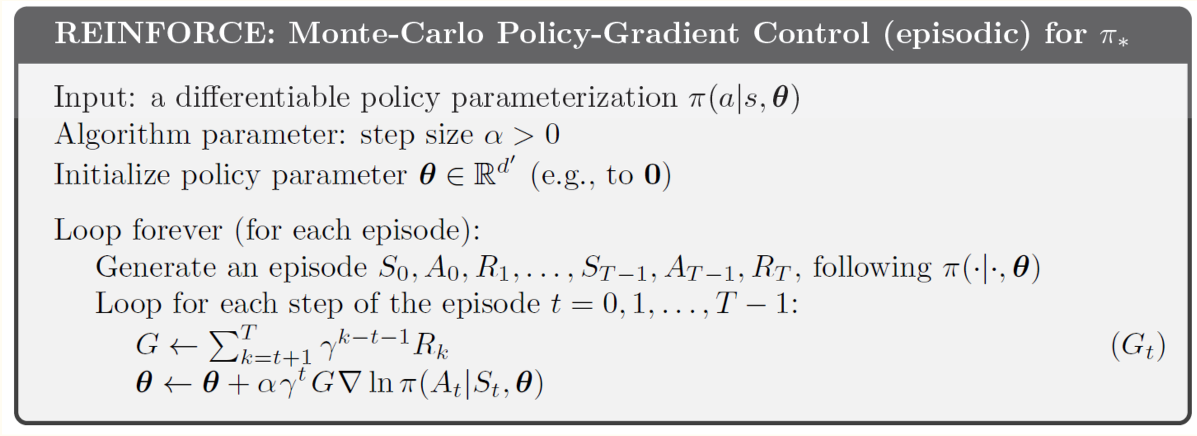
```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action $A$, observe $R, S'$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until $S$ is terminal
```

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

```
Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R, S'$
        If $S'$ is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., $\varepsilon$-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$
```

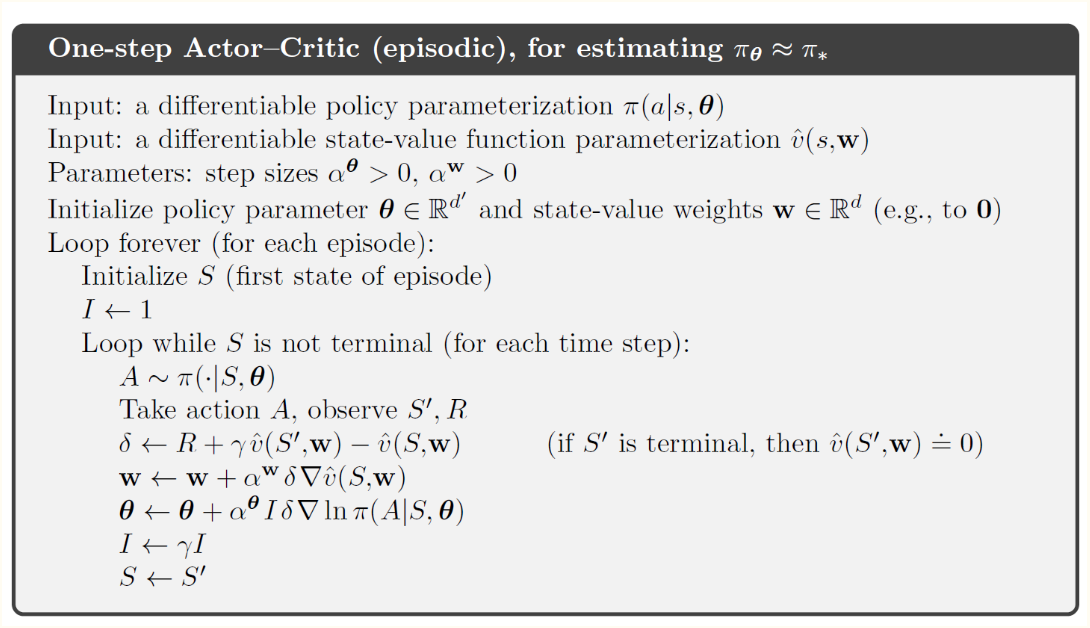
Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize $S$
    $\mathbf{z} \leftarrow \mathbf{0}$                                        (a $d$-dimensional vector)
    Loop for each step of episode:
    |   Choose $A \sim \pi(\cdot|S)$
    |   Take action $A$, observe $R, S'$
    |   $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S,\mathbf{w})$
    |   $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
    |   $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    |   $S \leftarrow S'$
    until $S'$ is terminal
```



REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$                    $(G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$          (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```