Jaron Sanders, 2MMS50 – Stochastic Decision Theory

Topics: Multi-Armed Bandits, Reinforcement Learning

Jaron Sanders

Department of Mathematics and Computer Science

Learning objectives

The upcoming two weeks are about RL! You will learn:

1. how to do policy evaluation by Monte Carlo simulation;
2. about Temporal Difference (TD) learning, and we will analyze its convergence;
3. how *Stochastic Approximation* and *Robbins–Monro's algorithm* are as techniques involved in the study of convergence properties of learning algorithms;
4. about Q-learning, and how to apply it;
5. about the State–action–reward–state–action (SARSA) algorithm.

Course material

For this part of the course, we will use:

- Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*
- Dimitri P. Bertsekas and John Tsitsiklis, *Neuro-dynamic programming*, 1996

Course material

For this part of the course, we will use:

- Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*
- Dimitri P. Bertsekas and John Tsitsiklis, *Neuro-dynamic programming*, 1996

We also briefly introduce the research in:

- Ipek, Mutlu, Martínez, and Caruana designed in *Self-Optimizing Memory Controllers: A Reinforcement Learning Approach*, 2008
- Li, Chu, Langford, and Schapire applied in *A Contextual-Bandit Approach to Personalized News Article Recommendation*, 2010

A nonextensive overview of disciplines

Reinforcement Learning (RL) is studied in many disciplines, including

1. Game theory.
2. Control theory.
3. Operations research.
4. Information theory.
5. Simulation-based optimization.
6. Etc.

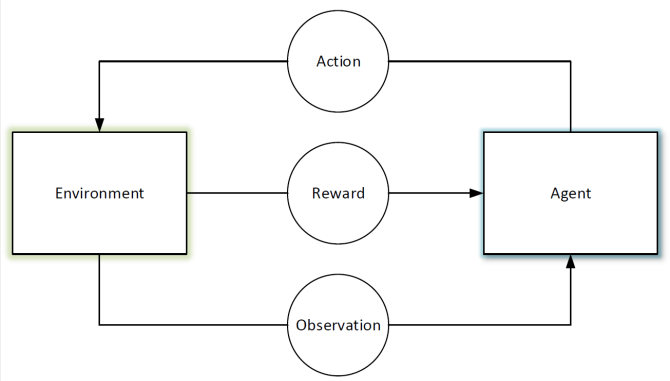
It is also called *approximate dynamic programming* or *neuro-dynamic programming*

Q. Did anyone hear of RL? If you did, where did you heard of it?

What RL algorithms do

At each time t, the agent receives an observation o_t, which typically includes the reward r_t. It then chooses an action a_t from the set of available actions, which is subsequently sent to the environment.

Q. Does RL suffer from the *exploration vs exploitation dilemma*?



Example: Optimizing memory control (1)

See §16.4 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

Computers use Dynamic Random Access Memory (DRAM).

A *DRAM controller* aims to efficiently interface between a processor and the DRAM. It needs to deal with dynamically changing patterns of read/write requests, while adhering to timing and resource constraints required by the hardware. This is a formidable scheduling problem.

Example: Optimizing memory control (1)

See §16.4 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

Computers use DRAM.

A *DRAM controller* aims to efficiently interface between a processor and the DRAM. It needs to deal with dynamically changing patterns of read/write requests, while adhering to timing and resource constraints required by the hardware. This is a formidable scheduling problem.

Ipek, Mutlu, Martínez, and Caruana designed in *Self-Optimizing Memory Controllers: A Reinforcement Learning Approach*, 2008, a RL memory controller.

State–of–the–art controllers did not yet take advantage of past scheduling experience, nor account for long-term consequences of decisions.

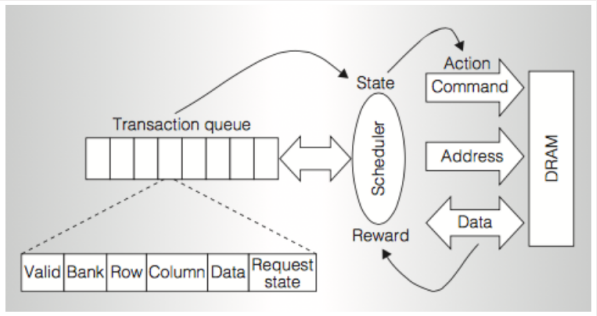
Example: Optimizing memory control (2)

See §16.4 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

They modeled the DRAM access process as an MDP:

- The states were the contents of the transaction queue.
- The actions actions were the commands to the DRAM system: *precharge, activate, read, write* and *NoOp*.

Penultimately, SARSA was used for the scheduling agent.



Example: Optimizing memory control (3)

See §16.4 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

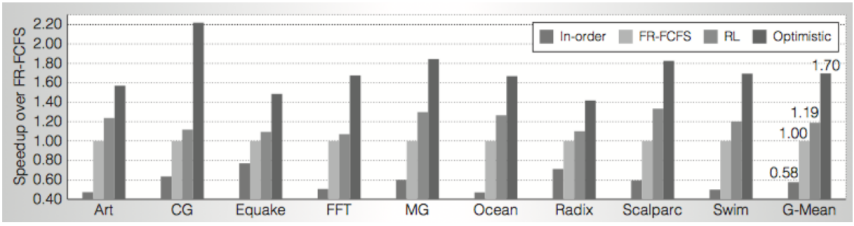


Figure 16.5: Performances of four controllers over a suite of 9 simulated benchmark applications. The controllers are: the simplest 'in-order' controller, FR-FCFS, the learning controller RL, and the unrealizable Optimistic controller which ignores all timing and resource constraints to provide a performance upper bound. Performance, normalized to that of FR-FCFS, is the inverse of execution time. At far right is the geometric mean of performances over the 9 benchmark applications for each controller. Controller RL comes closest to the ideal performance. ©2009 IEEE. Reprinted, with permission, from J. F. Martínez and E. İpek, Dynamic multicore resource management: A machine learning approach, Micro, IEEE, 29(5), p. 13.

Example: Personalized web services (1)

See §16.7 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

Personalizing web services is one approach to increasing users' satisfaction and increase the yield of a marketing campaign.

Q. Where, on the web, do you think you may have encountered Bandit or RL algorithms, hidden in the background?

Example: Personalized web services (1)

See §16.7 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

Personalizing web services is one approach to increasing users' satisfaction and increase the yield of a marketing campaign.

Q. Where, on the web, do you think you may have encountered Bandit or RL algorithms, hidden in the background?

A policy can recommend content based on a profile of the user's interests and preferences inferred from their history of online activity. To acquire feedback in realtime, it is common to monitor user clicks as an indicator of interest in a link.

Example: Personalized web services (2)

See §16.7 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

Marketing long used *A/B testing* on the web, but A/B testing does not personalize content delivery. To that end, one needs to add features describing individual users. This can be formalized as a Contextual MAB.

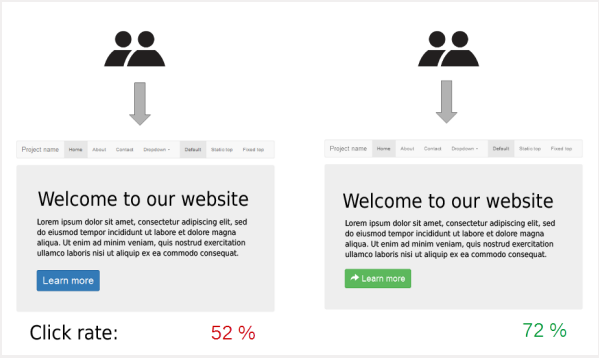


Image by Maxime Lorant, https://commons.wikimedia.org/wiki/File:A-B_testing_simple_example.png.

Example: Personalized web services (3)

See §16.7 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

E.g. Li, Chu, Langford, and Schapire applied in *A Contextual-Bandit Approach to Personalized News Article Recommendation*, 2010, the ε-Greedy, UCB, and Exp3 algorithms to personalize Yahoo!'s Front Page Today.

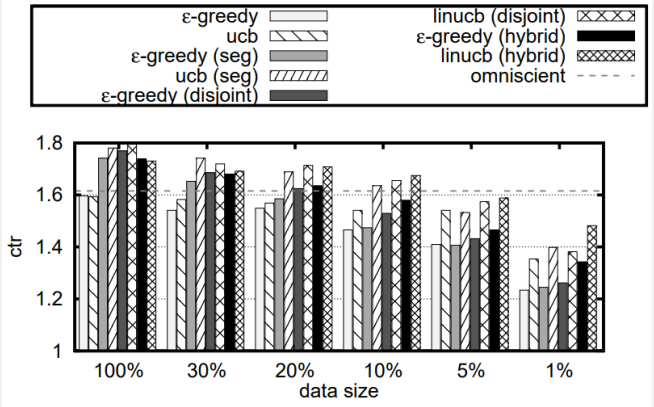
Their aim was to maximize the Click–Through Rate (CTR), i.e.,

$$\text{CTR} = \frac{\text{total \# of clicks}}{\text{total \# of visits}}. \qquad (33)$$

Figure 1: A snapshot of the "Featured" tab in the Today Module on Yahoo! Front Page. By default, the article at F1 position is highlighted at the story position.

Example: Personalized web services (4)

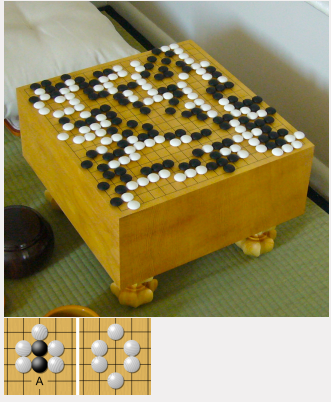
See Li, Chu, Langford, and Schapire applied in *A Contextual-Bandit Approach to Personalized News Article Recommendation*, 2010.



Example: Mastering the game of Go (1)

See §16.6 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

The ancient game of Go has challenged artificial intelligence researchers for decades. Until recently, no Go program had been able to play anywhere near the level of a human Go master.



Example: Mastering the game of Go (1)

See §16.6 in R. Sutton, A. Barto's *"Reinforcement Learning: An introduction"*.

The ancient game of Go has challenged artificial intelligence researchers for decades. Until recently, no Go program had been able to play anywhere near the level of a human Go master.

The game's goal is to capture an area of the board. While the rules are simple, they produce a very complex game.

Q.
Why are the complexities of Checkers $\ll$ Chess $\ll$ Go?

See `https://en.wikipedia.org/wiki/Go_(game)` for background.

Example: Mastering the game of Go (2)

A team at DeepMind (Silver et al., 2016) developed the program *AlphaGo* that broke this barrier by combining *deep artificial neural networks*, *supervised learning*, *Monte Carlo tree search*, and *reinforcement learning*.

Shortly thereafter, a similar version of *AlphaGo* won stunning victories over the 18-time world champion Lee Sedol. Its “move 37” has become legendary.

If your interest is peaked, watch the excellent 1.5 hour documentary *AlphaGo – The Movie* here: `https://youtu.be/WXuK6gekU1Y`

AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge

▲ The world's top Go player, Lee Sedol, lost the final game of the Google DeepMind challenge match. Photograph: Yonhap/Reuters

Preliminaries

Markov Decision Process (MDP)

RL is commonly modeled using MDPs. These have two principle components:

(a) *A discrete-time dynamic system whose state transition depends on a control*

Enumerate the states $1, \ldots, n$, and include possibly a termination state denoted by 0. When at state i, the control must be chosen from a finite set $U(i)$. At state i, the choice of a control u specifies the transition probability $p_{ij}(u)$ to the next state j.

Markov Decision Process (MDP)

RL is commonly modeled using MDPs. These have two principle components:

(a) *A discrete-time dynamic system whose state transition depends on a control*

Enumerate the states $1, \ldots, n$, and include possibly a termination state denoted by 0. When at state i, the control must be chosen from a finite set $U(i)$. At state i, the choice of a control u specifies the transition probability $p_{ij}(u)$ to the next state j.

(b) *A cost that accumulates additively over time, that depends on the states visited and controls chosen*

At the kth transition, we incur a cost $\alpha^k g(i,u,j)$ where $0 < \alpha \leq 1$.

Policies

We are interested in *policies*, preferably optimal in some sense.

Definition
A *policy* π is a sequence of functions $\mu_0, \mu_1, \ldots$ mapping states into controls with $\mu_k(i) \in U(i)$ for all states i.

Once a policy is fixed, the sequence of states $\{i_k\}_{k \geq 0}$ becomes a *Markov chain* with transition probabilities

$$\mathbb{P}[i_{k+1} = j | i_k = i] = p_{ij}(\mu_k(i)). \tag{34}$$

Finite horizon problems

In a *finite horizon problem*, cost accumulates over say $N < \infty$ stages.
Starting from an initial state i, the expected cost of a policy π is

$$J_N^\pi(i) = \mathbb{E}\left[\alpha^N G(i_N) + \sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \middle| i_0 = i\right], \tag{35}$$

where $\alpha^N G(i_N)$ is a *terminal cost* for ending up with final state i_N.

Finite horizon problems

In a *finite horizon problem*, cost accumulates over say $N < \infty$ stages.

Starting from an initial state i, the expected cost of a policy π is

$$J_N^\pi(i) = \mathbb{E}\left[\alpha^N G(i_N) + \sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \middle| i_0 = i\right], \tag{35}$$

where $\alpha^N G(i_N)$ is a *terminal cost* for ending up with final state i_N.

Denote the optimal N-stage *cost-to-go* starting from i by $J_N^*(i) = \min_\pi J_N^\pi(i)$. These can be considered components of an N-*stage optimal cost–to–go vector* J_N^*.

Infinite horizon problems

In an *infinite horizon problem*, the total expected cost starting from an initial state i and using a policy $\pi = \{\mu_0, \mu_1, \ldots\}$ is

$$J^{\pi}(i) = \lim_{N \to \infty} \mathbb{E}\Big[\sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \Big| i_0 = i\Big]. \tag{36}$$

Infinite horizon problems

In an *infinite horizon problem*, the total expected cost starting from an initial state i and using a policy $\pi = \{\mu_0, \mu_1, \ldots\}$ is

$$J^{\pi}(i) = \lim_{N\to\infty} \mathbb{E}\Big[\sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \Big| i_0 = i\Big]. \tag{36}$$

The *optimal cost–to–go vector* is given component-wise by $J^*(i) = \min_\pi J^\pi(i)$.

The cost-to-go of a stationary policy $\mu_1 = \mu_2 = \cdots = \mu$ will be denoted by $J^\mu(i)$. We say that μ is optimal if $J^\mu(i) = J^*(i)$ for all $i \in [n]$. The corresponding vector J^μ is referred to as the *cost–to–go vector of the stationary policy* μ.

Q. What is it that usually makes (36) well-defined, and not run of to ∞?

Dynamic Programming (DP) algorithm

This algorithm states that the optimal control choice must satisfy

$$J_k^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{k-1}^*)(j)\big) \quad \text{for} \quad i = 1, \ldots, n, k \in \mathbb{N}_+. \tag{37}$$

Hence J_N^* can be calculated recursively, starting with $J_0^* = G(i)$ for $i \in [n]$.

Dynamic Programming (DP) algorithm

This algorithm states that the optimal control choice must satisfy

$$J_k^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{k-1}^*)(j)\big) \quad \text{for} \quad i = 1, \dots, n, k \in \mathbb{N}_+. \tag{37}$$

Hence J_N^* can be calculated recursively, starting with $J_0^* = G(i)$ for $i \in [n]$.

Proof. Any policy π_k is of the form $\{u, \pi_{k-1}\}$. Therefore

$$J_k^*(i) = \min_{u \in U(i), \pi_{k-1}} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{k-1}^{\pi_{k-1}}(j)\big)$$

$$= \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \min_{\pi_{k-1}} J_{k-1}^{\pi_{k-1}}(j)\big) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_{k-1}^*(j)\big).$$

Stochastic shortest path problems (1)

Assume for this problem that $\alpha = 1$ and that there exists a *cost-free termination state*, 0, say.

Once the system reaches 0, it remains there at no further cost:

$$\text{that is, for all} \quad u \in U(0), \quad \text{it hold that} \quad p_{00}(u) = 1, g(0, u, 0) = 0. \tag{38}$$

We are interested in problems where reaching the termination state is inevitable, and to reach the termination state with minimum expected cost.

Stochastic shortest path problems (1)

Assume for this problem that $\alpha = 1$ and that there exists a *cost-free termination state*, 0, say.

Once the system reaches 0, it remains there at no further cost:

$$\text{that is, for all} \quad u \in U(0), \quad \text{it hold that} \quad p_{00}(u) = 1, g(0, u, 0) = 0. \tag{38}$$

We are interested in problems where reaching the termination state is inevitable, and to reach the termination state with minimum expected cost.

Q. Where do we encounter a stochastic shortest path problem in practice?

Stochastic shortest path problems (2)

Two special cases are worth mentioning at the outset:

(a) *The (deterministic) shortest path problem*
This is usually defined in terms of a direct graph consisting of n nodes plus a destination node 0, and the set of arcs (i, j) each have a cost c_{ij}.

Stochastic shortest path problems (2)

Two special cases are worth mentioning at the outset:

(a) *The (deterministic) shortest path problem*
This is usually defined in terms of a direct graph consisting of n nodes plus a destination node 0, and the set of arcs (i, j) each have a cost c_{ij}.

(b) *The finite horizon problem*
This can be converted into a stochastic shortest path problem by viewing time as an extra component of the state: i.e., $(i, k) \to (j, k+1)$ w.p. $p_{ij}(u)$, and state 0 corresponds to the end of the horizon.

Stationary and Proper Policies

Definition
A policy is called *stationary* if $\mu_0 = \mu_1 = \ldots = \mu$. A stationary policy μ is said to be *proper* if $\rho_\mu = \max_{i=1,\ldots,n} \mathbb{P}[i_n \neq 0 | i_0 = i, \mu] < 1$, otherwise it is said to be *improper*.

*** Ex. 30 —** Prove for $i \in [n]$ that if π is proper, then $\mathbb{P}[i_k \neq 0 | i_0 = i, \mu] \leq \rho_\mu^{\lfloor k/n \rfloor}$.

Stationary and Proper Policies

Definition
A policy is called *stationary* if $\mu_0 = \mu_1 = \ldots = \mu$. A stationary policy μ is said to be *proper* if $\rho_\mu = \max_{i=1,\ldots,n} \mathbb{P}[i_n \neq 0 | i_0 = i, \mu] < 1$, otherwise it is said to be *improper*.

* **Ex. 30 —** Prove for $i \in [n]$ that if π is proper, then $\mathbb{P}[i_k \neq 0 | i_0 = i, \mu] \leq \rho_\mu^{\lfloor k/n \rfloor}$.

We will work from now on under the following assumptions:
Assumption 1. There exists at least one proper policy.
Assumption 2. For every improper policy μ, the corresponding cost–to–go $J^\mu(i)$ is infinite for at least one state i.

Stationary and Proper Policies

Definition
A policy is called *stationary* if $\mu_0 = \mu_1 = \ldots = \mu$. A stationary policy μ is said to be *proper* if $\rho_\mu = \max_{i=1,\ldots,n} \mathbb{P}[i_n \neq 0 | i_0 = i, \mu] < 1$, otherwise it is said to be *improper*.

* **Ex. 30 —** Prove for $i \in [n]$ that if π is proper, then $\mathbb{P}[i_k \neq 0 | i_0 = i, \mu] \leq \rho_\mu^{\lfloor k/n \rfloor}$.

We will work from now on under the following assumptions:
Assumption 1. There exists at least one proper policy.
Assumption 2. For every improper policy μ, the corresponding cost–to–go $J^\mu(i)$ is infinite for at least one state i.

Ex. 31 — Are Assumptions 1, 2 met when *all* policies are proper?

Ex. 32 — Argue that $\sum_{j=0}^{n} p_{ij}(u) g(i,u,j) > 0 \, \forall i \neq 0, u \in U(i)$ is sufficient for Assumption 2.

Policy evaluation by Monte Carlo simulation

Policy evaluation by Monte Carlo simulation (1)

Consider the stochastic shortest path problem, and suppose μ is proper. We suppress p_{ij} and $g(i,j)$'s dependence on $\mu(i)$ for convenience.

Our goal will be to calculate by simulation the cost-to-go vector J^{μ}.

Q. How would you go about this?

Policy evaluation by Monte Carlo simulation (1)

Consider the stochastic shortest path problem, and suppose μ is proper. We suppress p_{ij} and $g(i,j)$'s dependence on $\mu(i)$ for convenience.

Our goal will be to calculate by simulation the cost-to-go vector J^μ.

Q. How would you go about this?

A. One possibility is to generate for each i, many sample trajectories. An alternative is to use each trajectory to obtain cost samples for all states visited by the trajectory, by considering the cost of the trajectory portion that starts at each intermediate state.

Ex. 33 — Explain the difference between *policy evaluation* and *policy iteration*.

Policy evaluation by Monte Carlo simulation (2)

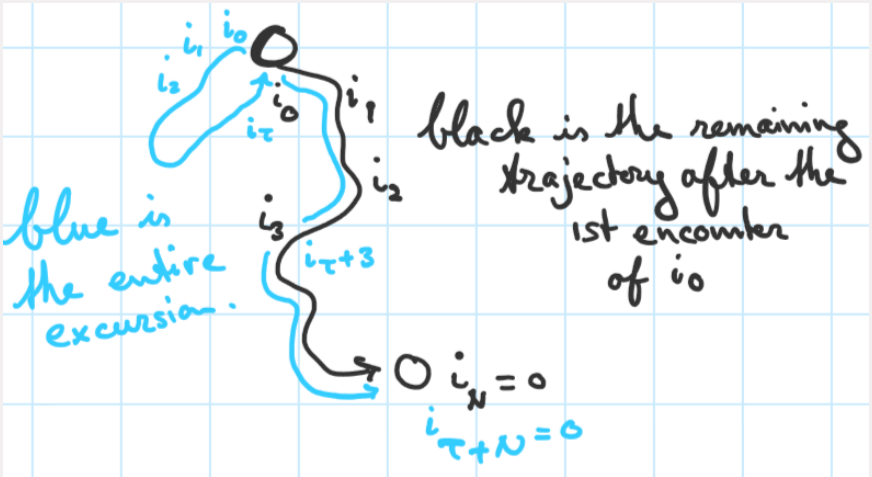
Suppose that we perform a number of simulation runs, each ending at the termination state 0. Consider the mth time a given state i_0 is encountered, and let $(i_0, i_1, \ldots, i_N)$ be the remainder of the corresponding trajectory. Let

$$c(i_0, m) = g(i_0, i_1) + \cdots + g(i_{N-1}, i_N). \tag{39}$$

Examine also the image on the next page :)

*** Ex. 34 —** Assume that different simulated trajectories are statistically independent, and that each trajectory is generated according to a Markov process determined by policy μ. Prove that for all i, m, then $J^\mu(i) = \mathbb{E}[c(i, m)]$.

Policy evaluation by Monte Carlo simulation (3)



Every-visit method (1)

Consequently, we can estimate

$$J^{\mu}(i) \approx J^{[K]}(i) \triangleq \frac{1}{K} \sum_{m=1}^{K} c(i,m) \tag{40}$$

following the Kth encounter with state i.

Every-visit method (1)

Consequently, we can estimate

$$J^{\mu}(i) \approx J^{[K]}(i) \triangleq \frac{1}{K} \sum_{m=1}^{K} c(i,m) \tag{40}$$

following the Kth encounter with state i.

This approximation can be calculated iteratively:

$$J^{[m+1]}(i) = J^{[m]}(i) + \gamma^{\{m\}}\big(c(i,m) - J^{[m]}(i)\big), \quad \gamma^{\{m\}} = \frac{1}{m}, \quad \text{starting} \quad J^{[0]}(i) = 0 \tag{41}$$

after each mth encounter of state i.

Ex. 35 — Prove that the recursion in (41) is equivalent to (40).

Every-visit method (2)

Alternatively, consider a trajectory $(i_0, \dots, i_N)$ and let $k \in [N]$. Note that this contains the subtrajectory $(i_k, i_{k+1}, \dots, i_N)$ which can be used to approximate J^μ.

Every-visit method (2)

Alternatively, consider a trajectory $(i_0, \dots, i_N)$ and let $k \in [N]$. Note that this contains the subtrajectory $(i_k, i_{k+1}, \dots, i_N)$ which can be used to approximate J^μ.

Concretely: at the end of a simulation run that generates $(i_0, \dots, i_N)$, calculate

$$J^{[k+1]}(i) \tag{42}$$
$$= J^{[k]}(i) + \begin{cases} \gamma(i_k)\big(g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + \cdots + g(i_{N-1}, i_N) - J^{[k]}(i_k)\big), & i = i_k, \\ 0 & \text{otherwise}, \end{cases}$$

for $k = 0, 1, \dots, N-1$. The stepsize $\gamma(i_k)$ is allowed to change each iteration.

Every-visit method (2)

Alternatively, consider a trajectory $(i_0, \dots, i_N)$ and let $k \in [N]$. Note that this contains the subtrajectory $(i_k, i_{k+1}, \dots, i_N)$ which can be used to approximate J^μ.

Concretely: at the end of a simulation run that generates $(i_0, \dots, i_N)$, calculate

$$J^{[k+1]}(i) \tag{42}$$
$$= J^{[k]}(i) + \begin{cases} \gamma(i_k)\big(g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + \cdots + g(i_{N-1}, i_N) - J^{[k]}(i_k)\big), & i = i_k, \\ 0 & \text{otherwise}, \end{cases}$$

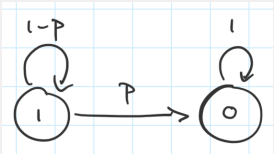
for $k = 0, 1, \dots, N-1$. The stepsize $\gamma(i_k)$ is allowed to change each iteration.

There are subtleties with the *every-visit method*. E.g., if a state i is encountered multiple times, $J^{[K]}(i)$ is updated using subtrajectories that are dependent.

Q. Why are these subtrajectories dependent? What might this imply for $J^{[K]}$?

Example: Bias in the every-visit method

Consider this Markov chain (with no / the trivial policy). Assume that state 0 is cost-free and absorbing, and that every transition out of state 1 carries a cost of 1.



Let N denote the number of transitions until the state becomes 0, starting from state 1.

Ex. 36 — Prove that $J^\mu(1) = 1/p = \mathbb{E}[N]$.

Ex. 37 — Consider a trajectory $(i_0 = 1, i_1, \ldots, i_{N-1}, i_N)$. Prove that for $m = 0, 1, \ldots, N-1$, the cost sample satisfies $c(1, m) = N - m$.

Ex. 38 — Prove now that $\mathbb{E}[J^{[N]}(1)] = (1+p)/(2p) \neq J^\mu(1)$, and conclude that there is indeed bias in the estimator.

Temporal Difference (TD)-learning

TD learning

TD learning refers to a class of model-free RL methods which learn by bootstrapping from the current estimate of the value function.

Like Monte Carlo methods, these sample from the environment.

Like Dynamic Programming (DP), these perform updates based on the estimates.

TD learning

TD learning refers to a class of model-free RL methods which learn by bootstrapping from the current estimate of the value function.

Like Monte Carlo methods, these sample from the environment.

Like DP, these perform updates based on the estimates.

TD methods are guaranteed to converge (for the single policy case, and with linearly parametrized function approximators).

See e.g. J. N. Tsitsiklis and B. Van Roy's *"An Analysis of Temporal-Difference Learning with Function Approximation",* IEEE Transactions on Automatic Control, Vol. 42, No. 5, May 1997, pp. 674-690.

Offline TD learning algorithm (1)

Let us first recall the update formula for Monte Carlo simulation:

"At the end of a simulation run that generates $(i_0, \ldots, i_N)$*, calculate*

$$J^{[k+1]}(i) = J^{[k]}(i) + \begin{cases} \gamma(i_k)\big(g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + \cdots + g(i_{N-1}, i_N) - J^{[k]}(i_k)\big), & i = i_k, \\ 0 & \text{otherwise}, \end{cases} \tag{43}$$

for $k = 0, 1, \ldots, N-1$*. The stepsize* $\gamma(i_k)$ *is allowed to change each iteration."*

TU/e

Offline TD-learning algorithm (2)

This update formula for Monte Carlo simulation can be written as

$$J(i_k) \leftarrow J(i_k) + \gamma\Big(\big(g(i_k, i_{k+1}) + J(i_{k+1}) - J(i_k)\big) + \big(g(i_{k+1}, i_{k+2}) + J(i_{k+2} - J(i_{k+1})\big) + \cdots + \big(g(i_{N-1}, i_N) + J(i_N - J(i_{N-1}))\big)\Big) \tag{44}$$

or equivalently

$$J(i_k) \leftarrow J(i_k) + \gamma\big(d_k + d_{k+1} + \cdots + d_{N-1}\big). \tag{45}$$

Here, the $d_k \triangleq g(i_k, i_{k+1}) + J(i_{k+1}) - J(i_k)$ are called *temporal differences*.

Q. What do temporal differences indicate about current estimates of $J^\mu(i)$?

Offline TD-learning algorithm (3)

The lth temporal difference becomes known as soon as the transition from i_l to i_{l+1} is simulated. This gives the possibility to carry out the update incrementally!

Specifically, observe that d_l appears in the update formula for $J(i_k)$ for every $k \leq l$. Therefore, set

$$J(i_k) \leftarrow J(i_k) + \gamma d_l = J(i_k) + \gamma\big(g(i_l, i_{l+1}) + J(i_{l+1}) - J(i_l)\big) \quad \text{for} \quad k = 0, \ldots, l, \tag{46}$$

as soon as the transition to i_{l+1} has been simulated. The stepsize γ can be different for every state i_k, and at every iteration.

Start of Intermezzo #3

Stochastic Iterative Algorithms

Stochastic approximation (1)

Say we want to solve a system of equations $H(r) = r$ where $H : \mathbb{R}^n \to \mathbb{R}^n$.

Ex. 39 — Say there exists a cost function s.t. $H(r) = r - \nabla f(r)$. Argue that we are then trying to find a critical point of f.

The fixed-point algorithm

$$r^{\{t+1\}} = H(r^{\{t\}}), \text{ or its small stepsize version } r^{\{t+1\}} = (1-\gamma)r^{\{t\}} + \gamma H(r^{\{t\}}) \quad (47)$$

are possible algorithms to use.

Ex. 40 — Say there exists a cost function s.t. $H(r) = r - \nabla f(r)$. Show that the small-step size version in (47) reduces to the gradient method.

Stochastic approximation (2)

Suppose now that the functional form of H is not known precisely, or that exact evaluation of $H(r)$ is difficult. However, suppose we *do* have access to a random variable $H(r) + W$, where W is a random noise term.

We may then consider the algorithm

$$r^{[t+1]} = (1 - \gamma)r^{[t]} + \gamma(H(r^{[t]}) + W^{[t]}), \tag{48}$$

which is called a *stochastic iterative* or *stochastic approximation* algorithm.

If the noise is sufficiently small, this should still work!

Stochastic approximation (2)

Suppose now that the functional form of H is not known precisely, or that exact evaluation of $H(r)$ is difficult. However, suppose we *do* have access to a random variable $H(r) + W$, where W is a random noise term.

We may then consider the algorithm

$$r^{[t+1]} = (1 - \gamma) r^{[t]} + \gamma (H(r^{[t]}) + W^{[t]}), \tag{48}$$

which is called a *stochastic iterative* or *stochastic approximation* algorithm.

If the noise is sufficiently small, this should still work!

Note that this algorithm is sensitive to the choice of γ:

- Positively, smaller γ reduces the effect of noise.
- Negatively, smaller γ slows down progress.

Robbins–Monro algorithm (1)

Let V be a random variable with known probability distribution $p(v|r)$ that depends on r. Suppose we are interested in solving

$$\mathbb{E}_{p(\cdot|r)}[g(r, V)] = r, \tag{49}$$

where g is a known function.

Robbins–Monro algorithm (1)

Let V be a random variable with known probability distribution $p(v|r)$ that depends on r. Suppose we are interested in solving

$$\mathbb{E}_{p(\cdot|r)}[g(r, V)] = r, \tag{49}$$

where g is a known function.

We may use the deterministic algorithm

$$r^{\{t+1\}} = (1 - \gamma) r^{\{t\}} + \gamma \mathbb{E}_{p(\cdot|r^{\{t\}})}[g(r^{\{t\}}, V)], \tag{50}$$

but evaluating the expectation may be undesirable / impossible.

If we use a single *sample* instead, we obtain the *Robbins–Monro algorithm*

$$r^{[t+1]} = (1 - \gamma) r^{[t]} + \gamma g(r^{[t]}, V^{[t]}). \tag{51}$$

*** Ex. 41 —** Prove that (51) is a special case of (48).

Convergence of stochastic approximation (1)

Theorem (Bertsekas, Tsitsiklis, "Neuro-dynamic programming", p. 157–158)

Consider an algorithm of the form

$$r^{[t+1]} = (1 - \gamma^{\{t\}}(i))r^{[t]}(i) + \gamma^{\{t\}}(i)\big((H^{[t]}r^{[t]})(i) + w^{[t]}(i) + u^{[t]}(i)\big), \quad t \in \mathbb{N}_0. \tag{52}$$

Assume that:

(a) *The stepsizes* $\gamma^{\{t\}}(i) \geq 0$ *and satisfy* $\sum_t \gamma^{\{t\}}(i) > \infty$, $\sum_t (\gamma^{\{t\}})^2(i) < \infty$.

(b) *The noise terms* $w^{[t]}(i)$ *satisfy:* (i) *For every* i *and* t, *we have* $\mathbb{E}[w^{[t]}(i)|\mathcal{F}_t] = 0$. (ii) *Given any norm* $\|\cdot\|$ *on* $\mathbb{R}^n$, *there exist constants* A *and* B *s.t.* $\mathbb{E}[(w^{[t]})^2(i)|\mathcal{F}_t] \leq A + B\|r^{[t]}\|^2 \; \forall i, t$.

(c) *There exist a vector* $r^\star$, *a positive vector* ξ, *and a scalar* $\beta \in [0, 1)$ *s.t.* $\|H^{[t]}r^{[t]} - r^\star\|_\xi \leq \beta\|r^{[t]} - r^\star\|_\xi \; \forall t$.

(d) *There exists a nonnegative random sequence* θ_t *that converges to zero w.p. one, and is s.t.* $|u^{[t]}(i)| \leq \theta^{[t]}(\|r^{[t]}\|_\xi + 1) \; \forall i, t$.

Then, $r^{[t]}$ *converges to* $r^\star$ *with probability one.*

TU/e

Convergence of stochastic approximation (2)

Consider an algorithm

$$X_{n+1} = X_n + a_n\big(h(x_n) + \xi_{n+1}\big) \tag{53}$$

where for all n, $\mathbb{E}[\xi_{n+1}|\mathcal{F}_n] = 0$ almost surely. Here, $\mathcal{F}_n = \sigma(X_i; i \leq n)$.

Theorem

If h is L-Lipschitz, $\sum a_n = \infty$, $\sum a_n^2 < \infty$, $\sup_n \|X_n\| < \infty$ a.s., and for all n, $\mathbb{E}[\|\xi_{n+1}\|^2|\mathcal{F}_n] \leq K(1 + \|X_n\|^2)$ a.s., then the algorithm trajectories look like those of $\dot{x} = h(x)$. Moreover if $\dot{x} = h(x)$ has a unique globally asymptotically stable point $x^\star$, then $\lim_{n\to\infty} X_n = x^*$ almost surely.

TU/e

End of Intermezzo #3

Let's go on with TD(λ)-learning

TD(λ): Stochastic approximation on a Bellman equation (1)

TD(λ) is a *Robbins–Monro stochastic approximation method* for solving

$$\text{the equations} \quad J^{\mu}(i_k) = \mathbb{E}\Big[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1})\Big] \quad \text{for the unknowns} \quad J^{\mu}(i_k). \tag{54}$$

The challenge is ofcourse that the expectation is assumed *inevaluable*.

TD(λ): Stochastic approximation on a Bellman equation (1)

TD(λ) is a *Robbins–Monro stochastic approximation method* for solving

$$\text{the equations} \quad J^\mu(i_k) = \mathbb{E}\Big[\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1})\Big] \quad \text{for the unknowns} \quad J^\mu(i_k). \tag{54}$$

The challenge is ofcourse that the expectation is assumed *inevaluable*.

Starting from Bellman's equation $J^\mu(i_k) = \mathbb{E}[g(i_k, i_{k+1}) + J^\mu(i_{k+1})]$, the resulting stochastic approximation method takes the form

$$J(i_k) \leftarrow J(i_k) + \gamma\big(g(i_k, i_{k+1}) + J(i_{k+1}) - J(i_k)\big) \tag{55}$$

with such update being executed each time i_k is visited.

TD(λ): Stochastic approximation on a Bellman equation (2)

More generally, instead of taking into account only immediate cost, we could take into account $l+1$ transitions. Consider the $(l+1)$-step Bellman equation:

$$J^{\mu}(i_k) = \mathbb{E}\Big[\sum_{m=0}^{l} g(i_{k+m}, i_{k+m+1}) + J^{\mu}(i_{k+l+1})\Big]. \tag{56}$$

TD(λ): Stochastic approximation on a Bellman equation (2)

More generally, instead of taking into account only immediate cost, we could take into account $l+1$ transitions. Consider the $(l+1)$-step Bellman equation:

$$J^{\mu}(i_k) = \mathbb{E}\Big[\sum_{m=0}^{l} g(i_{k+m}, i_{k+m+1}) + J^{\mu}(i_{k+l+1})\Big]. \tag{56}$$

In the absence of a preference for one l over another, we could then e.g. consider this *weighted multistep Bellman equation* with $\lambda \in [0,1]$:

$$J^{\mu}(i_k) = (1-\lambda)\mathbb{E}\Big[\sum_{l=0}^{\infty} \lambda^l \Big(\sum_{m=0}^{l} g(i_{k+m}, i_{k+m+1}) + J^{\mu}(i_{k+l+1})\Big)\Big]. \tag{57}$$

TD(λ): Stochastic approximation on a Bellman equation (3)

Interchanging the order summation and use $(1-\lambda)\sum_{l=m}^{\infty}\lambda^l = \lambda^m$ to find

$$J^{\mu}(i_k) = \mathbb{E}\Big[(1-\lambda)\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1})\sum_{l=m}^{\infty}\lambda^l + \sum_{l=0}^{\infty} J^{\mu}(i_{k+l+1})(\lambda^l - \lambda^{l+1})\Big]$$

$$= \mathbb{E}\Big[\sum_{m=0}^{\infty}\lambda^m\Big(g(i_{k+m}, i_{k+m+1}) + J^{\mu}(i_{k+m+1}) - J^{\mu}(i_{k+m})\Big)\Big] + J^{\mu}(i_k). \qquad (58)$$

TD(λ): Stochastic approximation on a Bellman equation (3)

Interchanging the order summation and use $(1-\lambda)\sum_{l=m}^{\infty}\lambda^l = \lambda^m$ to find

$$J^\mu(i_k) = \mathbb{E}\Big[(1-\lambda)\sum_{m=0}^{\infty} g(i_{k+m}, i_{k+m+1})\sum_{l=m}^{\infty}\lambda^l + \sum_{l=0}^{\infty} J^\mu(i_{k+l+1})(\lambda^l - \lambda^{l+1})\Big]$$

$$= \mathbb{E}\Big[\sum_{m=0}^{\infty}\lambda^m\Big(g(i_{k+m}, i_{k+m+1}) + J^\mu(i_{k+m+1}) - J^\mu(i_{k+m})\Big)\Big] + J^\mu(i_k). \tag{58}$$

In terms of temporal differences, $J^\mu(i_k) = \mathbb{E}[\sum_{m=k}^{\infty}\lambda^{m-k}d_m] + J^\mu(i_k)$, and the Robbins–Monro stochastic approximation based on (58) is

$$J(i_k) \leftarrow J(i_k) + \gamma\sum_{m=k}^{\infty}\lambda^{m-k}d_m, \tag{59}$$

where γ is a stepsize parameter that may change.

TU/e

TD(λ): Limiting cases

We now have a family of algorithms known as TD(λ), one for each value of λ.

- When $\lambda = 1$, we recover the *Monte Carlo policy evaluation*.
- When $0 < \lambda < 1$, as $\lambda \downarrow 0$, we tend to discount the effect of the temporal differences of state transitions in the far future on the cost estimate of the current state more.
- When $\lambda = 0$, the algorithm is based on the one-step Bellman equation.

TD(λ): Limiting cases

We now have a family of algorithms known as TD(λ), one for each value of λ.

- When $\lambda = 1$, we recover the *Monte Carlo policy evaluation*.
- When $0 < \lambda < 1$, as $\lambda \downarrow 0$, we tend to discount the effect of the temporal differences of state transitions in the far future on the cost estimate of the current state more.
- When $\lambda = 0$, the algorithm is based on the one-step Bellman equation.

You may now wonder if there is a best choice of λ.

The general conclusion is that intermediate values of λ seem to work best. Unfortunately, there is an intricate interplay with the stepsize, and an all-encompassing answer seems illusive.

General Offline TD methods

We now introduce a generalization of TD(λ) that gives more flexibility, and for which a single convergence proof can be given.

General Offline TD methods

We now introduce a generalization of TD(λ) that gives more flexibility, and for which a single convergence proof can be given.

Suppose that a sample trajectory $i_0, i_1, \ldots$ has been generated and let d_m be the corresponding temporal differences. Postulate an update formula of the form

$$J(i) \leftarrow J(i) + \gamma \sum_{m=1}^{\infty} z_m(i) d_m. \tag{60}$$

Here, the $z_m^{[t]}$ are called the *eligibility coefficients*.

Example

If $z_m(i) = \sum_{\{j | m_j \leq m\}} \lambda^{m-m_j}$, then we recover the every-visit TD(λ) method.
If $z_m(i) = \lambda^{m-m_1}$ for $m \geq m_1$, then we find the so-called *first-visit* TD(λ) method.

Ex. 42 — Prove the latter fact.

Convergence of Offline TD Methods (1)

Let $\mathcal{F}_t$ represent the history of the algorithm up to the point at which the simulation of the t-th trajectory is to commence. Let $J^{[t]}$ be the estimate of the cost-to-go vector available at time t.

Based on $\mathcal{F}_t$, we choose an initial state $i_0^{[t]}$ for the t-th trajectory and stepsizes $\gamma^{\{t\}}(i), i = 1, \ldots, n$. We generate a trajectory $i_0^{[t]}, i_1^{[t]}, \ldots, i_{N_t}^{[t]}$ where $N^{[t]}$ is the first time that the trajectory reaches state 0. We then update

$$J^{[t+1]}(i) = J^{[t]}(i) + \gamma^{\{t\}}(i) \sum_{m=0}^{N^{[t]}-1} z_m^{[t]}(i) d_m^{[t]} \tag{61}$$

where $d_m^{[t]} = g(i_m^{[t]}, i_{m+1}^{[t]}) + J^{[t]}(i_{m+1}^{[t]}) - J^{[t]}(i_m^{[t]})$.

Convergence of Offline TD Methods (2)

Assume that the eligibility coefficients satisfy:

- $z_m^{[t]}(i) \geq 0$
- $z_{-1}^{[t]}(i) = 0$
- $z_m^{[t]}(i) \leq z_{m-1}^{[t]}(i)$ if $i_m^{[t]} \neq i$
- $z_m^{[t]}(i) \leq z_{m-1}^{[t]}(i) + 1$ if $i_m^{[t]} = i$
- $z_m^{[t]}(i)$ is completely determined by $\mathcal{F}_t$ and $i_0^{[t]}, \ldots, i_m^{[t]}$

Convergence of Offline TD Methods (3)

We allow the possibility that no update of $J(i)$ is carried out *even if* a trajectory visits state i. However, for $J(i)$ to converge to the correct value, there should be enough trajectories that lead to a nontrivial update of $J(i)$. For this reason, we need another assumption.

Convergence of Offline TD Methods (3)

We allow the possibility that no update of $J(i)$ is carried out *even if* a trajectory visits state i. However, for $J(i)$ to converge to the correct value, there should be enough trajectories that lead to a nontrivial update of $J(i)$. For this reason, we need another assumption.

Define $q^{[t]}(i) = \mathbb{P}[\exists m : z_m^{[t]}(i) > 0 | \mathcal{F}_t]$, and let $T^i = \{t | q^{[t]}(i) > 0\}$. Assume that:

- For any fixed i, t, $z_m^{[t]}(i)$ must be equal to 1 the first time that it becomes positive.
- There exists a deterministic constant $\delta > 0$ s.t. $q^{[t]}(i) \geq \delta$ for all $t \in T^i$ and i.
- $\gamma^{\{t\}}(i) \geq 0$ for all $t \in T^i$, and $\gamma^{\{t\}}(i) = 0$ for $t \notin T^i$.
- For all i, $\sum_{t \in T^i} \gamma^{\{t\}}(i) = \infty$ and $\sum_{t \in T^i} (\gamma^{\{t\}})^2(i) < \infty$ for all i.

Convergence of Offline TD Methods (3)

We allow the possibility that no update of $J(i)$ is carried out *even if* a trajectory visits state i. However, for $J(i)$ to converge to the correct value, there should be enough trajectories that lead to a nontrivial update of $J(i)$. For this reason, we need another assumption.

Define $q^{[t]}(i) = \mathbb{P}[\exists m : z_m^{[t]}(i) > 0 | \mathcal{F}_t]$, and let $T^i = \{t | q^{[t]}(i) > 0\}$. Assume that:

- For any fixed i, t, $z_m^{[t]}(i)$ must be equal to 1 the first time that it becomes positive.
- There exists a deterministic constant $\delta > 0$ s.t. $q^{[t]}(i) \geq \delta$ for all $t \in T^i$ and i.
- $\gamma^{\{t\}}(i) \geq 0$ for all $t \in T^i$, and $\gamma^{\{t\}}(i) = 0$ for $t \notin T^i$.
- For all i, $\sum_{t \in T^i} \gamma^{\{t\}}(i) = \infty$ and $\sum_{t \in T^i} (\gamma^{\{t\}})^2(i) < \infty$ for all i.

Q. In which *three topics* specifically have you now seen the last assumption?

Convergence of Offline TD Methods (4)

Theorem (Bertsekas, Tsitsiklis, “Neuro-dynamic programming”, p. 210–217)

Consider the offline TD algorithm as described above, and let the prior assumptions hold. Assume that the policy under consideration is proper. Then, for all i, $J^{[t]}(i)$ converges to $J^{\mu}(i)$ with probability one.

Convergence of Offline TD Methods (4)

Theorem (Bertsekas, Tsitsiklis, "Neuro-dynamic programming", p. 210–217)
Consider the offline TD algorithm as described above, and let the prior assumptions hold. Assume that the policy under consideration is proper. Then, for all i, $J^{[t]}(i)$ converges to $J^{\mu}(i)$ with probability one.

A very brief proof sketch...
First, one rewrites the algorithm into the form of (52). Second, one proceeds with a step-by-step verification of Proposition 18's sufficient conditions. The heart of the proof is showing the contraction property of $H^{[t]}$.

Q-learning

What is *Q*-learning?

Wikipedia's description rings true:

> *"Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations."*

`https://en.wikipedia.org/wiki/Q-learning`

What is *Q*-learning?

Wikipedia's description rings true:

> *"Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations."*

`https://en.wikipedia.org/wiki/Q-learning`

Q refers to the function that returns the reward used to provide reinforcement. We may call it the *quality* of an action taken in a given state if you want.

What is *Q*-learning?

Wikipedia's description rings true:

> *"Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations."*

`https://en.wikipedia.org/wiki/Q-learning`

Q refers to the function that returns the reward used to provide reinforcement. We may call it the *quality* of an action taken in a given state if you want.

Q. What is the difference between policy iteration, and *Q*-learning?

What is *Q*-learning?

Wikipedia's description rings true:

> *"Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations."*

`https://en.wikipedia.org/wiki/Q-learning`

Q refers to the function that returns the reward used to provide reinforcement. We may call it the *quality* of an action taken in a given state if you want.

Q. What is the difference between policy iteration, and *Q*-learning?

Q. Which do you think is more powerful: Bandit algorithms, or *Q*-learning? And, what are (dis)advantages of either?

Two references containing convergence analyses

There are convergence results and approximation error bounds for Q-learning type methods for certain special types of function approximation (e.g., state aggregation) for instance here:

J. N. Tsitsiklis and B. Van Roy, “Feature-Based Methods for Large Scale Dynamic Programming”, Machine Learning, Vol. 22, 1996, pp. 59-94.

Two references containing convergence analyses

There are convergence results and approximation error bounds for *Q*-learning type methods for certain special types of function approximation (e.g., state aggregation) for instance here:

J. N. Tsitsiklis and B. Van Roy, “Feature-Based Methods for Large Scale Dynamic Programming”, Machine Learning, Vol. 22, 1996, pp. 59-94.

Q-learning and the TD(0) methods (with a lookup table representation) are viewed as *Stochastic Approximation methods* for solving *Bellman’s equation*. Their convergence is established by first developing a stochastic approximation theory for the case where the iteration mapping is a contraction with respect to a weighted maximum norm. See e.g.:

J. N. Tsitsiklis, “Asynchronous Stochastic Approximation and Q-learning”, Machine Learning, 16, 1994, pp. 185-202. Correction.

Optimal Q-factor (1)

Assume we are dealing with a stochastic shortest path problem, and that:
Assumption 1. There exists at least one proper policy.
Assumption 2. For every improper policy μ, the corresponding cost–to–go $J^{\mu}(i)$ is infinite for at least one state i.

Optimal Q-factor (2)

Define the *optimal Q-factor corresponding to a pair* (i,u) *with* $u \in U(i)$, by

$$Q^\star(0,u) = 0, \quad \text{and for } \ i \in [n], \quad Q^\star(i,u) = \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + J^\star(j)\big). \tag{62}$$

TU/e

Optimal Q-factor (2)

Define the *optimal Q-factor corresponding to a pair* (i,u) *with* $u \in U(i)$, by

$$Q^\star(0,u) = 0, \quad \text{and for } \ i \in [n], \quad Q^\star(i,u) = \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + J^\star(j)\big). \tag{62}$$

Next, Bellman's equation can be written as

$$J^\star(i) = \min_{u \in U(i)} Q^\star(i,u). \tag{63}$$

Substituting (63) into (62), we obtain the relation

$$Q^\star(i,u) = \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + \min_{v \in U(j)} Q^\star(j,v)\big). \tag{64}$$

Optimal Q-factor (2)

Define the *optimal Q-factor corresponding to a pair* (i,u) *with* $u \in U(i)$, by

$$Q^\star(0,u) = 0, \text{ and for } i \in [n], \quad Q^\star(i,u) = \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + J^\star(j)\big). \tag{62}$$

Next, Bellman's equation can be written as

$$J^\star(i) = \min_{u \in U(i)} Q^\star(i,u). \tag{63}$$

Substituting (63) into (62), we obtain the relation

$$Q^\star(i,u) = \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + \min_{v \in U(j)} Q^\star(j,v)\big). \tag{64}$$

*** Ex. 43 —** Prove that the optimal Q-factors in (62) are the *unique solution* to the system of equations in (64) with the condition that $Q(0,u) = 0$.

Value iteration algorithm

For $\gamma \in (0, 1]$, we can write a generic version of the value iteration algorithm as

$$Q(i,u) \leftarrow (1-\gamma)Q(i,u) \sum_{j=0}^{n} p_{ij}(u)\big(g(i,u,j) + \min_{v \in U(j)} Q(j,v)\big). \tag{65}$$

Now, Q-learning is an approximate version of this iteration, in which the expectation is replaced by a single sample, i.e.,

$$Q(i,u) \leftarrow (1-\gamma)Q(i,u) + \gamma\Big(g(i,u,j) + \min_{v \in U(j)} Q(j,v)\Big) \tag{66}$$

It is a combination of value iteration and simulation.

Convergence of Q-learning

From Bertsekas, Tsitsiklis, “Neuro-dynamic programming”, p. 248.

Theorem (Convergence of Q-learning)

Suppose that for all $i, u \in U(i)$, $\sum_{t=0}^{\infty} \gamma_t(i,u) = \infty$, $\sum_{t=0}^{\infty} \gamma_t^2(i,u) < \infty$. Then $Q_t(i,u)$ converges with probability one to $Q^\star(i,u)$ for every i and u, in each of the following cases:

(a) *If all policies are proper.*

(b) *If Assumptions 2.1 and 2.2 hold (there exists a proper policy and every improper policy has infinite cost for some initial state), and if $Q_t(i,u)$ is guaranteed to be bounded, with probability one.*

SARSA-learning

SARSA's original reference

State–action–reward–state–action (SARSA) is an algorithm for learning a Markov decision process policy, used in the RL area of machine learning. It was proposed in the technical note:

On-line Q-learning using connectionist systems, G.A. Rummery, M. Niranjan (1994)

SARSA was originally referred to as Modified Connectionist Q-Learning (MCQ-L).

SARSA's original reference

SARSA is an algorithm for learning a Markov decision process policy, used in the RL area of machine learning. It was proposed in the technical note:

On-line Q-learning using connectionist systems, G.A. Rummery, M. Niranjan (1994)

SARSA was originally referred to as MCQ-L.

For more background, see §6.4–6, §7.2, §10.2–5, §12.7 of Sutton's book:

Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*

SARSA's name explained

The name reflects that the function for updating the Q-value depends on:

- the current state of the agent s_t,
- the action the agent chooses a_t,
- the reward r_t the agent gets for choosing this action,
- the state s_{t+1} that the agent enters after taking that action,
- and finally the next action a_{t+1} the agent chooses in its new state.

SARSA's name explained

The name reflects that the function for updating the Q-value depends on:

- the current state of the agent s_t,
- the action the agent chooses a_t,
- the reward r_t the agent gets for choosing this action,
- the state s_{t+1} that the agent enters after taking that action,
- and finally the next action a_{t+1} the agent chooses in its new state.

The acronym for the quintuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ is SARSA. Some authors use a slightly different convention and write the quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, depending to which time step the reward is formally assigned.

SARSA algorithm (1)

Let's discuss the algorithm by following §6.6 of:

Dimitri P. Bertsekas and John Tsitsiklis, *Neuro-dynamic programming*, 1996

SARSA requires us to generalize to the case we have approximate Q-factors of the form $\hat{Q}(i,u,r)$ available, where r is a parameter vector.

Q. Imagine we are in control of a network of queues. Amongst others, what control parameters can the r represent?

SARSA algorithm (1)

Let's discuss the algorithm by following §6.6 of:

Dimitri P. Bertsekas and John Tsitsiklis, *Neuro-dynamic programming*, 1996

SARSA requires us to generalize to the case we have approximate Q-factors of the form $\hat{Q}(i,u,r)$ available, where r is a parameter vector.

Q. Imagine we are in control of a network of queues. Amongst others, what control parameters can the r represent?

Since we can only control r, we can compute $\nabla_r \hat{Q}(i,u,r)$, which indicates in which direction r should be updated. This heuristic gives the update

$$r \leftarrow r + \gamma \nabla_r \hat{Q}(i,u,r)\big(g(i,u,j) + \alpha \min_{v \in U(j)} \hat{Q}(j,v,r) - \hat{Q}(i,u,r)\big). \tag{67}$$

SARSA algorithm (2)

Suppose now that we generate the state trajectory $i_0, i_1, \ldots$ according to a fixed policy μ, that is by employing the actions $u_k = \mu(i_k)$. The heuristic becomes

$$r \leftarrow r + \gamma \nabla_r \hat{Q}(i_k, u_k, r)\big(g(i_k, u_k, j) + \alpha \hat{Q}(i_{k+1}, u_{k+1}, r) - \hat{Q}(i_k, u_k, r)\big). \tag{68}$$

SARSA algorithm (2)

Suppose now that we generate the state trajectory $i_0, i_1, \ldots$ according to a fixed policy μ, that is by employing the actions $u_k = \mu(i_k)$. The heuristic becomes

$$r \leftarrow r + \gamma \nabla_r \hat{Q}(i_k, u_k, r)\big(g(i_k, u_k, j) + \alpha \hat{Q}(i_{k+1}, u_{k+1}, r) - \hat{Q}(i_k, u_k, r)\big). \tag{68}$$

Next, (68) can be used for policy evaluation, followed by a policy update to

$$\overline{\mu}(i) = \arg \min_{u \in U(i)} \hat{Q}(i, u, r) \quad \text{and this process can then be repeated.} \tag{69}$$

SARSA algorithm (2)

Suppose now that we generate the state trajectory $i_0, i_1, \ldots$ according to a fixed policy μ, that is by employing the actions $u_k = \mu(i_k)$. The heuristic becomes

$$r \leftarrow r + \gamma \nabla_r \hat{Q}(i_k, u_k, r)\big(g(i_k, u_k, j) + \alpha \hat{Q}(i_{k+1}, u_{k+1}, r) - \hat{Q}(i_k, u_k, r)\big). \tag{68}$$

Next, (68) can be used for policy evaluation, followed by a policy update to

$$\overline{\mu}(i) = \arg \min_{u \in U(i)} \hat{Q}(i, u, r) \quad \text{and this process can then be repeated.} \tag{69}$$

** **Ex. 44 —** Convince yourself that this update procedure identical to offline TD(0)'s parametric variant update rule?

Hint. For $\lambda \in [0, 1]$, offline TD(λ)'s parametric-variant update rule is given by

$$r \leftarrow r + \gamma d_k \sum_{m=0}^{k} \lambda^{k-m} \nabla_r \tilde{J}(i_m, r) \tag{70}$$

SARSA algorithm (3)

At a typical step, we have a current parameter vector r_k, are at some state i_k, and have chosen an action u_k. We then simulate a transition to the next state i_{k+1}, and generate the decision u_{k+1} to be used at the next state as follows.

SARSA algorithm (3)

At a typical step, we have a current parameter vector r_k, are at some state i_k, and have chosen an action u_k. We then simulate a transition to the next state i_{k+1}, and generate the decision u_{k+1} to be used at the next state as follows.

Let $0 < \varepsilon \ll 1$. With probability ε, we choose an action u_{k+1} randomly and uniformly from the set $U(i_{k+1})$ and with probability $1 - \varepsilon$, we choose an action u_{k+1} that minimizes $\hat{Q}(i_{k+1}, u, r_k)$ over all $u \in U(i_{k+1})$.

SARSA algorithm (3)

At a typical step, we have a current parameter vector r_k, are at some state i_k, and have chosen an action u_k. We then simulate a transition to the next state i_{k+1}, and generate the decision u_{k+1} to be used at the next state as follows.

Let $0 < \varepsilon \ll 1$. With probability ε, we choose an action u_{k+1} randomly and uniformly from the set $U(i_{k+1})$ and with probability $1 - \varepsilon$, we choose an action u_{k+1} that minimizes $\hat{Q}(i_{k+1}, u, r_k)$ over all $u \in U(i_{k+1})$.

Finally, at that point, we update

$$r_{k+1} = r_k + \gamma \nabla \hat{Q}(i_k, u_k, r_k)\big(g(i_k, u_k, i_{k+1}) + \alpha \hat{Q}(i_{k+1}, u_{k+1}, r_k) - \hat{Q}(i_k, u_k, r_k)\big). \quad (71)$$

Now that algorithm is what is called SARSA!