ECE586 MDPs and Reinforcement Learning **University of Illinois at Urbana-Champaign**
Spring 2019 **Acknowledgment: R. Srikant's Notes and Discussions**

Lecture 10: Q-Learning, Function Approximation, Temporal Difference Learning

Instructor: Dimitrios Katselis *Scribe: Z. Zhou, L. Buccafusca, W. Wei, C. Shih, K. Li, Z. Guo, D. Phan*

Previously, we have considered solving MDPs when the underlying model, i.e., the transition probabilities and the cost structure are completely known. If the MDP model is unknown or if it is known but it is computationally infeasible to be used directly except through sampling due to the domain size, then *simulation-based stochastic approximation algorithms* for estimating the optimal policy of MDPs can be employed. Here, "simulation-based" means that the agent/controller can observe the system trajectory under any choice of control actions and therefore, trajectory sampling from the MDP is performed. We will study two algorithms in this case:

(1) **Q-learning**, studied in this lecture: It is based on the Robbins–Monro algorithm (*stochastic approximation* (SA)) to estimate the value function for an unconstrained MDP. A primal-dual Q-learning algorithm can be employed for MDPs with monotone optimal policies. The Q-learning algorithm also applies as a suboptimal method for POMDPs.

(2) **Policy gradient algorithms**, which we will see in later lectures: Such algorithms rely on parametric policy classes, e.g., on the class of *Gibbs policies*. They employ gradient estimation of the cost function together with a stochastic gradient algorithm on the performance surface induced by the selected smoothly parameterized policy class $\mathcal{M} = \{\mu_\theta : \theta \in \mathbb{R}^d\}$ of stochastic stationary policies to estimate the optimal policy. Policy gradient algorithms apply to MDPs and constrained MDPs, while they yield suboptimal policy search methods for POMDPs.

Note: Determining the optimal policy of an MDP (or a POMDP) when the model parameters are unknown correspond to *stochastic adaptive control problems*. Stochastic adaptive control algorithms are of two types: **direct methods**, where the unknown MDP model is estimated simultaneously with updating the control policy, and **implicit methods** such as *simulation-based methods*, where the underlying MDP model is not directly estimated in order to compute the control policy[1]. Q-learning, Temporal Difference (TD) learning and policy gradient algorithms correspond to such simulation-based methods. Such methods are also called **reinforcement learning algorithms**.

Reinforcement Learning: Also called *neuro-dynamic programming* or *approximate dynamic programming*[2]. The first term is due to the use of neural networks with RL algorithms. Reinforcement learning is a branch of machine learning. It corresponds to learning how to map situations or states to actions or equivalently to learning how to control a system in order to minimize or to maximize a numerical performance measure that expresses a long-term objective. The agent is not told which actions to take, but instead must discover which actions yield the most reward or the least cost by trying them. Actions may affect the immediate reward or cost and the next situation or state. Thus, actions influence all subsequent rewards or costs. These two characteristics, i.e., the *trial-and-error search* and the *delayed rewards or costs*, are the two most distinguishing features of reinforcement learning. The main differences of reinforcement learning from other machine learning paradigms are summarized below:

[1] Often in the literature, the terms "direct" and "implicit or indirect" learning are used with reverse associations to methods. With the term "direct learning" several authors refer to simulation-based methods and the "directness" corresponds to "directly, without estimating an environmental model". Similarly, "indirect learning" is used for methods estimating first a model for the environment and then computing an optimal policy via "certainty equivalence". As an additional comment of independent interest, it is well-known in adaptive control theory that the certainty equivalence principle may lead to suboptimal performance due to the lack of exploration.

[2] Consider the very rich field known as *approximate dynamic programming*. Neuro-Dynamic Programming is mainly a theoretical treatment of the field using the language of control theory. Reinforcement Learning describes the field from the perspective of artificial intelligence and computer science. Finally, Approximate Dynamic Programming uses the parlance of operations research, with more emphasis on high dimensional problems that typically arise in this community.

(a) There is no supervisor, only a reward or a cost signal which reinforces certain actions over others.

(b) The feedback is typically delayed.

(c) The data are sequential and therefore, time is critical.

(d) The actions of the agent affect the (subsequent) data generation mechanism.

Two major classes of RL methods are[3]:

(i) **Policy-Iteration based algorithms or "actor-critic" learning**: As an initial comment, actor-critic learning is the (generalized) *learning analogue* for the Policy Iteration method of Dynamic Programming (DP), i.e., the corresponding approach that is followed in the context of reinforcement learning due to the lack of knowledge of the underlying MDP model and possibly due to the use of function approximation if the state-action space is large. More specifically, actor-critic methods implement *Generalized Policy Iteration*. Recall that policy iteration alternates between a complete policy evaluation and a complete policy improvement step. If sample-based methods or function approximation for the underlying value function or the Q-function[4] are employed, exact evaluation of the policies may require infinite many samples or might be impossible due to the function approximating class. Hence, RL algorithms simulating policy iteration must change the policy by relying on partial or incomplete knowledge of the associated value function. Such schemes are said to implement generalized policy iteration.

To make the discussion more concrete, recall the Policy Iteration method. For an initial policy $\mu_0 = \mu$ the following iteration is set:

- *Policy Evaluation*: Compute $J_{\mu_k} = (I - \alpha P_{\mu_k})^{-1}\bar{c}_{\mu_k}$ and let $Q_{\mu_k}(i,u) = \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) J_{\mu_k}(j)$.
- *Policy Improvement*: Update μ_k to the *greedy policy* associated with Q_{μ_k}, i.e., to $\mu_{k+1}(i) = \arg\min_u \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) J_{\mu_k}(j) = \arg\min_u Q_{\mu_k}(i,u)$ for all $i \in \mathcal{X}$.

We now want to perform the above two steps without access to the true dynamics and reward or cost structure. Assume that an initial policy $\mu_0 = \mu$ is chosen. The corresponding iteration implemented by an actor-critic method is subdivided as follows:

- *Policy Evaluation ("Critic")*: Estimate the value of the current policy of the actor or Q_{μ_k} by performing a *model-free policy evaluation*. This is a value prediction problem.
- *Policy Improvement ("Actor")*: Update μ_k based on the estimated Q_{μ_k} from the previous step.

A comment: As it is clear from the previous description, the *actor* performs policy improvement. This improvement can be implemented in a similar spirit as in policy iteration by moving the policy towards the greedy policy underlying the Q-function estimate obtained from the critic. Alternatively, policy gradient can be performed directly on the performance surface underlying a chosen parametric policy class. Moreover, the actor performs some form of *exploration* to enrich the current policy, i.e., to guarantee that all actions are tried. Roughly speaking, the exploration process guarantees that all state-action pairs are sampled sufficiently often. Exploration of all actions available at a particular state is important, even if they might be suboptimal with respect to the current Q-estimate. Moreover, the policy evaluation step produces an estimate of Q_{μ_k}. Clearly, a point of vital importance is that the policy improvement step monotonically improves the policy as in the model-based case.

Methods for policy evaluation ("critic") include:

- *Monte Carlo policy evaluation*.
- *Temporal Difference methods*: TD(λ), SARSA, etc.

[3] These classes fall within the more general class of *value-function based schemes*.

[4] or both sample-based methods with function approximation

(ii) **Value-Iteration based algorithms:** Such approaches are based on some online version of value iteration $\hat{J}_{k+1}(i) = \min_u \bar{c}(i,u) + a\sum_j P_{ij}(u)\hat{J}_k(j), \forall i \in \mathcal{X}$. The basic learning algorithm in this class is *Q-learning*. The aim of Q-learning is to approximate the optimal action-value function Q by generating a sequence $\{\hat{Q}_k\}_{k\geq 0}$ of such functions. The underlying idea is that if $\hat{Q}_k$ is "close" to Q for some k, then the corresponding greedy policy with respect to $\hat{Q}_k$ will be close to the optimal policy μ^* which is greedy with respect to Q.

10.1 Q-function and Q-learning

The Q-learning algorithm is a widely used *model-free* reinforcement learning algorithm. It corresponds to the Robbins–Monro stochastic approximation algorithm applied to estimate the value function of Bellman's dynamic programming equation. It was introduced in 1989 by Christopher J. C. H. Watkins in his PhD Thesis. A convergence proof was presented by Christopher J. C. H. Watkins and Peter Dayan in 1992. A more detailed mathematical proof was given by John Tsitsiklis in 1994, and by Dimitri Bertsekas and John Tsitsiklis in their book on Neuro-Dynamic Programming in 1996. As a final comment, although Q-learning is a cornerstone of the RL field, it does not really scale to large state-control spaces. Large state-control spaces associated with complex problems can be handled by using state aggregation or approximation techniques for the Q-values.

Recall that Bellman's dynamic programming equation for a discounted cost MDP is:

$$\begin{aligned} J^*(i) &= \min_{u\in\mathcal{U}} \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) J^*(j) \\ &= \min_{u\in\mathcal{U}} \bar{c}(i,u) + \alpha E[J^*(x_{k+1})|x_k = i, u_k = u]. \end{aligned} \tag{10.1}$$

Definition 10.1. *(Q-function or **state-action value function**) The (optimal) Q-function is defined by*

$$Q(i,u) = \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) J^*(j), \quad i \in \mathcal{X}, u \in \mathcal{U}. \tag{10.2}$$

If the Q-function is known, then $J^*(i) = \min_u Q(i,u)$ and the optimal policy is greedy with respect to Q i.e., $\mu^*(i) = \arg\min_u Q(i,u)$. Combining the previous results, we obtain:

$$\begin{aligned} Q(i,u) &= \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) \min_v Q(j,v) \\ &= \bar{c}(i,u) + \alpha E\left[\min_v Q(x_{k+1},v)|x_k = i, u_k = u\right]. \end{aligned} \tag{10.3}$$

By abusing notation, we will denote by T the operator in the right-hand side of the previous equation. Then, (10.3) can be alternatively written as:

$$Q = T(Q)\,, \tag{10.4}$$

where

$$\begin{aligned} T(Q)(i,u) &= \bar{c}(i,u) + \alpha \sum_j P_{ij}(u) \min_v Q(j,v) \\ &= E\left[c(x_k,u_k) + \alpha \min_v Q(x_{k+1},v)|x_k = i, u_k = u\right]. \end{aligned} \tag{10.5}$$

Theorem 10.2. *T is a contraction mapping (with respect to $\|\cdot\|_\infty$) with parameter $\alpha < 1$, i.e.*

$$\|T(Q_1) - T(Q_2)\|_\infty \leq \alpha \|Q_1 - Q_2\|_\infty\,.$$

Proof.

$$(T(Q_1) - T(Q_2))(i,u) = \alpha \sum_j P_{ij}(u) \left(\min_v Q_1(j,v) - \min_v Q_2(j,v) \right) .$$

It follows that

$$\begin{aligned}
|(T(Q_1) - T(Q_2))(i,u)| &\leq \alpha \sum_j P_{ij}(u) \left| \min_v Q_1(j,v) - \min_v Q_2(j,v) \right| \\
&\leq \alpha \sum_j P_{ij}(u) \max_v |Q_1(j,v) - Q_2(j,v)| \\
&\leq \alpha \sum_j P_{ij}(u) \max_{r,v} |Q_1(r,v) - Q_2(r,v)| \\
&= \alpha \max_{r,v} |Q_1(r,v) - Q_2(r,v)| \sum_j P_{ij}(u) \\
&= \alpha \left\| Q_1 - Q_2 \right\|_\infty ,
\end{aligned}$$

which leads to $\left\| T(Q_1) - T(Q_2) \right\|_\infty \leq \alpha \left\| Q_1 - Q_2 \right\|_\infty$ since the obtained bound is valid for any $(i,u) \in \mathcal{X} \times \mathcal{U}$. We further note that the second inequality is due to the fact that for two vectors x and y of equal dimension:

$$\left| \min_i x_i - \min_i y_i \right| \leq \max_i |x_i - y_i| .$$

□

Lemma 10.3. *For two vectors x and y of equal dimension,*

$$\left| \min_i x_i - \min_i y_i \right| \leq \max_i |x_i - y_i| .$$

Proof. Assume without loss of generality that $\min_i x_i \geq \min_i y_i$ and that $i_1 = \arg\min_i y_i$. Then:

$$\left| \min_i x_i - \min_i y_i \right| = \min_i x_i - \min_i y_i = \min_i x_i - y_{i_1} \leq x_{i_1} - y_{i_1} \underbrace{=}_{(*)} |x_{i_1} - y_{i_1}| \leq \max_i |x_i - y_i|.$$

$(*)$ is due to the fact that $y_{i_1} = \min_i y_i \leq \min_i x_i \leq x_{i_1}$. □

Note: For two vectors x and y of equal dimension, $|\max_i x_i - \max_i y_i| \leq \max_i |x_i - y_i|$. This inequality is useful in the case of a problem with rewards. It can be used to show the contraction property of the corresponding T operator defined as $T(Q)(i,u) = \bar{r}(i,u) + \alpha \sum_j P_{ij}(u) \max_v Q(j,v)$ in this case.

Since T is a contraction mapping, we can use value iteration to compute $Q(i,u)$ if the MDP model is known. In practice, the MDP model is unknown. This problem is addressed via Q-learning.

10.1.1 Q-learning

By now, it should be clear that Q-learning means *learning the Q-function*. In the following, we introduce two versions of Q-learning: **synchronous** and **asynchronous**. Before this, we examine the rationale behind Q-learning and its connection with the fundamental Robbins-Monro algorithm.

Observe that (10.1) has an expectation **inside** the minimization, while (10.3) has an expectation **outside** the minimization. This crucial observation forms the basis for using stochastic approximation algorithms to estimate the Q-function. Note that (10.3) can be written as:

$$E\left[h(Q)(x_k, u_k, x_{k+1}) | x_k = i, u_k = u \right] = 0, \quad \forall (i,u) \in \mathcal{X} \times \mathcal{U} \tag{10.6}$$

where

$$h(Q)(x_k, u_k, x_{k+1}) = c(x_k, u_k) + \alpha \min_v Q(x_{k+1}, v) - Q(x_k, u_k). \tag{10.7}$$

The Robbins-Monro algorithm can be used to estimate the solution of (10.6) via the recursion:

$$\begin{aligned}\hat{Q}_{k+1}(x_k, u_k) &= \hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k) h(\hat{Q}_k)(x_k, u_k, x_{k+1}) \\ &= \hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k) \left(c(x_k, u_k) + \alpha \min_v \hat{Q}_k(x_{k+1}, v) - \hat{Q}_k(x_k, u_k) \right)\end{aligned} \tag{10.8}$$

or

$$\hat{Q}_{k+1}(x_k, u_k) = (1 - \varepsilon_k(x_k, u_k))\hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k) \left(c(x_k, u_k) + \alpha \min_v \hat{Q}_k(x_{k+1}, v) \right). \tag{10.9}$$

Note: As we will see later on, the term $h(Q)(x_k, u_k, x_{k+1})$ is very similar to the *temporal difference* in TD schemes for policy evaluation (specifically, to TD(0)), except for the minimization operation applied to $\hat{Q}(x_{k+1}, v)$. Defining the temporal difference to incorporate the minimization (or maximization for rewards) operator, Q-learning corresponds to an instance of temporal difference learning.

Remark: It turns out that the temporal differences underlying Q-learning do not telescope. This is due to the fact that Q-learning is an inherently *off-policy* algorithm. We clarify more the notion of off-policy algorithms after the deterministic Q-learning example that we provide in the following.

The decreasing stepsize sequences (or sequences of *learning rates*) $\{\varepsilon_k(i,u)\}_{k\geq 0}$ for any $(i,u) \in \mathcal{X} \times \mathcal{U}$ must satisfy $\sum_k \varepsilon_k(i,u) = \infty$ and $\sum_k \varepsilon_k^2(i,u) < \infty$ in the context of stochastic approximation. These constraints are also called *Robbins-Monro conditions*. A possible choice is

$$\varepsilon_k(i,u) = \frac{\varepsilon}{N_k(i,u)}, \tag{10.10}$$

where $\varepsilon > 0$ is a constant and $N_k(i,u)$ is the number of times the state-action pair (i,u) has been visited until time k by the algorithm. The algorithm is summarized as a two-timescale stochastic approximation algorithm in the following table:

Q-learning Algorithm (Two time-scale implementation)
slow time scale
for $n = 1, 2, \ldots$ **do**
Update the policy as $\mu_n(i) = \arg\min_u \hat{Q}_{n\bar{T}}(i,u), \forall i \in \mathcal{X}$
Fast time scale
for $k = n\bar{T}, n\bar{T}+1, \ldots, (n+1)\bar{T}-1$ **do**
Given the state x_k, choose $u_k = \mu_n(x_k)$.
Simulate (or sample) the next state as $x_{k+1} \sim P_{x_k, x_{k+1}}(u_k)$.
Update the Q-values as:
$\hat{Q}_{k+1}(x_k, u_k) = (1 - \varepsilon_k(x_k, u_k))\hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k) \left(c(x_k, u_k) + \alpha \min_v \hat{Q}_k(x_{k+1}, v) \right)$,
where $\varepsilon_k(x_k, u_k)$ are given by (10.10).
endfor
endfor

This two-time scale implementation of Q-learning with policy updates during an infinitely long trajectory appears in the literature, but it is *not* the standard form of this algorithm.

Remarks:

1. *Fast Time Scale*: The Q-function is updated by applying the same policy for a fixed period $\bar{T}$ of time slots referred to as the *update interval*.
2. *Slow Time Scale*: Policy update.
3. *Simulate (or sample) the next state as* $x_{k+1} \sim P_{x_k,x_{k+1}}(u_k)$: The Q-learning algorithm does not require explicit knowledge of the transition probabilities, but simply access to the controlled system to measure its next state x_{k+1} when an action u_k is applied. Via the same access to the controlled system, the cost signal $c(x_k, u_k)$ is observed.
4. Under some conditions, the Q-learning algorithm for a finite state MDP converges almost surely to the optimal solution of Bellman's equation.

Note: The above implementation explicitly describes a way such that the observed trajectory is generated. Clearly, updated policies participate in this trajectory formation.

In the literature, instead of the previous two-time scale form which incorporates policy update steps by considering the greedy policies with respect to the underlying Q-function estimates at the end of update intervals, Q-learning is usually presented in the following two variations, where the generation of the underlying system trajectory is not explicitly specified:

Synchronous Q-learning: At time $k+1$, we update the Q-function as

$$\begin{aligned}\hat{Q}_{k+1}(i,u) &= \hat{Q}_k(i,u) + \varepsilon_k \left(c(i,u) + \alpha \min_v \hat{Q}_k(j,v) - \hat{Q}_k(i,u)\right) \\ &= (1-\varepsilon_k)\hat{Q}_k(i,u) + \varepsilon_k \left(c(i,u) + \alpha \min_v \hat{Q}_k(j,v)\right) \quad \forall (i,u) \in \mathcal{X} \times \mathcal{U}. \end{aligned} \tag{10.11}$$

This version is known as *synchronous Q-learning* because the update is taken for all state-action pairs (i,u) per iteration. Here, $x_{k+1} = j$ when $u_k = u$ is applied to state $x_k = i$. In other words, at stage k, a random variable $x_{k+1} = x_{k+1}(i,u)$ is simulated with probability $P_{i\cdot}(u)$ to implement the update of the Q-value for each state action-pair (i,u).

Asynchronous Q-learning: Suppose that $x_k = i$. With probability p_k we choose $u_k = \arg\min_u \hat{Q}_k(x_k,u)$ and with probability $(1-p_k)$ we choose any action uniformly at random. This ensures that all state-action pairs (i,u) are *explored* instead of just *exploiting* the current knowledge of $\hat{Q}_k$. At time $k+1$, we update the Q-function as

$$\hat{Q}_{k+1}(i,u) = (1-\varepsilon_k(i,u))\hat{Q}_k(i,u) + \varepsilon_k(i,u)\left(c(i,u) + \alpha \min_v \hat{Q}_k(j,v)\right)$$

and for $(x,v) \neq (i,u)$:

$$\hat{Q}_{k+1}(x,v) = \hat{Q}_k(x,v)\,.$$

In practice, asynchronous Q-learning is used. Moreover, the term "Q-learning" is reserved almost exclusively for asynchronous Q-learning.

Stochastic Approximation Rationale in relevance to (10.4): We want to solve $Q = T(Q)$. Borrowing the idea from stochastic approximation, we can use the iteration

$$\hat{r}_{k+1} = (1-\varepsilon_k)\hat{r}_k + \varepsilon_k(T(\hat{r}_k) + \text{noise}),$$

to solve an equation of the form $r = T(r)$ or $T(r) - r = 0$.

Policy Selection: For the described synchronous and asynchornous Q-learning schemes, either we theoretically wait until convergence and then we choose the greedy policy with respect to the limit, or we stop the iteration earlier, at some $\hat{Q}_k$ and we extract the corresponding greedy policy as our control choice.

10.1.2 An Illustrative Case: Deterministic Q-learning

Consider a deterministic MDP model:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) \\ c_k &= c(x_k, u_k). \end{aligned}$$

Let the performance metric for some policy μ be the discounted cost objective:

$$J_\mu(i) = \sum_{k=0}^{\infty} \alpha^k c_k, \quad x_0 = i, u_k = \mu(x_k).$$

Then, $J^*(i) = \min_\mu J_\mu(i), \forall i \in \mathcal{X}$. Moreover, the Q-function is defined is this case as:

$$Q(i,u) = c(i,u) + \alpha J^*(f(i,u)), \quad \forall (i,u) \in \mathcal{X} \times \mathcal{U}.$$

Clearly, $J^*(i) = \min_u Q(i,u)$. Moreover, the above definition can be given only in terms of the Q-function as follows:

$$Q(i,u) = c(i,u) + \alpha \min_v Q(f(i,u), v), \quad \forall (i,u) \in \mathcal{X} \times \mathcal{U}.$$

Q-learning in this setup is implemented as follows:

Deterministic Q-learning Algorithm
Initialization: $\hat{Q}_0(i,u) = Q_0(i,u), \forall (i,u) \in \mathcal{X} \times \mathcal{U}$
for $k = 1, 2, \ldots$ **do**
Update: $\hat{Q}_{k+1}(x_k, u_k) = c_k + \alpha \min_v \hat{Q}_k(x_{k+1}, v)$.
endfor

Note: This algorithm relies on a particular system trajectory. However, no reference on how to choose the actions is made.

Theorem 10.4. *Consider the previous deterministic MDP model and the described Q-learning algorithm for this scenario. If each state-action pair is visited infinitely often, then $\hat{Q}_k(i,u) \xrightarrow[k\to\infty]{} Q(i,u)$ for every state-action pair $(i,u) \in \mathcal{X} \times \mathcal{U}$.*

Proof. Let $\Delta_k = \|\hat{Q}_k - Q\|_\infty = \max_{i,u} \left|\hat{Q}_k(i,u) - Q(i,u)\right|$. Then, at every time instant $k+1$ we have:

$$\begin{aligned} \left|\hat{Q}_{k+1}(i,u) - Q(i,u)\right| &= \left|c_k + \alpha \min_v \hat{Q}_k(x_{k+1}, v) - \left(c_k + \alpha \min_v Q(x_{k+1}, v)\right)\right| \\ &= \alpha \left|\min_v \hat{Q}_k(x_{k+1}, v) - \min_v Q(x_{k+1}, v)\right| \\ &\leq \alpha \max_v \left|\hat{Q}_k(x_{k+1}, v) - Q(x_{k+1}, v)\right| \\ &\leq \alpha \max_{x,v} \left|\hat{Q}_k(x, v) - Q(x, v)\right| = \alpha \Delta_k. \end{aligned}$$

Consider now any time interval $\{k_1, k_1 + 1, \ldots, k_2\}$ in which each state-action pair is visited at least once. Then, by the previous derivation:

$$\Delta_{k_2} \leq \alpha \Delta_{k_1},$$

with an elementwise interpretation of the inequality. Since as $k \to \infty$ there are infinite many such intervals and $\alpha < 1$, we conclude that $\Delta_k \to 0$ as $k \to \infty$. □

Remark: Guaranteeing that each state-action pair is visited infinitely often ensures that as $k \to \infty$ there are infinite many time intervals $\{k_1, k_1 + 1, \ldots, k_2\}$ such that each state-action pair is visited at least once. To ensure this requirement, some form of *exploration* is often used. The speed of convergence may depend critically on the efficiency of the exploration.

Key Aspect: As mentioned before, there is no reference to the policy used to generate the system trajectory, i.e., an arbitrary policy may be used for this purpose. For this reason, Q-learning is an *off-policy* algorithm. An alternative way to explain the term "off-policy learning" is *learning a policy by following another policy*. A distinguishing characteristic of off-policy algorithms like Q-learning is the fact that the update rule *need not have any relation to the underlying learning policy* generating the system trajectory. The updates of $\hat{Q}_k(x_k, u_k)$ in the Deterministic Q-learning algorithm and in (10.8)-(10.9) depend on $\min_v \hat{Q}_k(x_{k+1}, v)$, i.e., the estimated Q-functions are updated on the basis of hypothetical actions or more precisely *actions other than those actually executed*. On the other hand, *on-policy* algorithms learn the underlying policy as well. Moreover, on-policy algorithms update value functions strictly on the basis of the experience gained from executing some (possibly nonstationary) policy.

10.1.3 Used policy during learning

Q-learning is a fairly simple algorithm to implement. Additionally, it permits the use of arbitrary policies to generate the training data provided that in the limit, all state-action pairs are visited and therefore are updated infinitely often. Action sampling strategies to achieve this requirement in closed-loop learning are *ϵ-greedy action selection schemes* or *Boltzmann exploration* that we provide in the sequel[5]. More generally, any *persistent* exploration method will ensure the aforementioned target requirement. With appropriate tuning, asymptotic consistency can be achieved.

10.2 Convergence of Q-learning

The Q-learning algorithm can be viewed as a stochastic process to which techniques of stochastic approximation can be applied. The following proof of convergence relies on an extension of Dvoretzky's (1956) formulation of the classical Robbins-Monro (1951) stochastic approximation theory to obtain a class of converging processes involving the maximum norm.

Theorem 10.5. *A random iterative process $\Delta_{n+1}(x) = (1 - a_n(x))\Delta_n(x) + b_n(x)F_n(x)$ converges to zero with probability 1 under the following assumptions:*

1. *The state space is finite.*
2. *$\sum_n a_n(x) = \infty, \sum_n a_n^2(x) < \infty, \sum_n b_n(x) = \infty, \sum_n b_n^2(x) < \infty$ and $E[b_n(x)|\mathcal{F}_n] \leq E[a_n(x)|F_n]$ uniformly with probability 1.*
3. *$\|E[F_n(x)|\mathcal{F}_n]\|_w \leq \gamma \|\Delta_n\|_w$ for $\gamma \in (0, 1)$.*
4. *$\mathrm{Var}(F_n(x)|\mathcal{F}_n) \leq C(1 + \|\Delta_n\|_w)^2$, where C is some constant.*

Here, $\mathcal{F}_n = \{\Delta_n, \Delta_{n-1}, \ldots, F_{n-1}, \ldots, a_{n-1}, \ldots, b_{n-1}, \ldots\}$ stands for the history at step n. $F_n(x), a_n(x)$ and $b_n(x)$ are allowed to depend on the past insofar as the above conditions remain valid. Finally, $\|\cdot\|_w$ denotes some weighted maximum norm.

[5] We have seen both ϵ-greedy schemes and Boltzmann exploration in the context of multi-armed bandit problems in a previous lecture file.

Weighted Maximum Norm: Let $w = [w_1, \ldots, w_n]^T \in \mathbb{R}^n$ be a positive vector. Then, for a vector $x \in \mathbb{R}^n$ the following weighted max-norm based on w can be defined:

$$\|x\|_w = \max_{1 \le i \le n} \left| \frac{x_i}{w_i} \right|. \tag{10.12}$$

This norm induces a matrix norm. For a matrix $A \in \mathbb{R}^{n \times n}$ the following weighted max-norm based on w can be defined:

$$\|A\|_w = \max_{\|x\|_w = 1} \{\|Ax\|_w | x \in \mathbb{R}^n\}. \tag{10.13}$$

For $w = [1, \ldots, 1]^T$ the usual maximum norm is recovered.

The process Δ_n will generally represent the difference between a stochastic process of interest and some optimal value (e.g., the optimal value function).

Theorem 10.6. ***(Q-learning convergence)*** *Given an MDP with finite state and action spaces, the Q-learning algorithm, given by the update rule*

$$\hat{Q}_{k+1}(x_k, u_k) = (1 - \varepsilon_k(x_k, u_k))\hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k)\left(c(x_k, u_k) + \alpha \min_v \hat{Q}_k(x_{k+1}, v)\right) \tag{10.14}$$

converges with probability 1 *to the optimal Q-function, i.e., the one with values given by (10.2), as long as*

$$\sum_k \varepsilon_k(x, u) = \infty \quad \textit{and} \quad \sum_k \varepsilon_k^2(x, u) < \infty, \quad \forall (x, u) \in \mathcal{X} \times \mathcal{U} \tag{10.15}$$

uniformly with probability 1 *and* $\mathrm{Var}(c(x, u))$ *is bounded.*

Note: Under the usual consideration that $\varepsilon_k(x_k, u_k) \in [0, 1)$, (10.15) requires that all state-action pairs are visited *infinitely often*.

Proof. Let

$$\Delta_k(x, u) = \hat{Q}_k(x, u) - Q(x, u),$$

where $Q(x, u)$ is given by (10.2). Subtracting from both sides of (10.14) $Q(x_k, u_k)$, we obtain:

$$\Delta_{k+1}(x_k, u_k) = (1 - \varepsilon_k(x_k, u_k))\Delta_k(x_k, u_k) + \varepsilon_k(x_k, u_k)\left(\underbrace{c(x_k, u_k) + \alpha \min_v \hat{Q}_k(x_{k+1}, v) - Q(x_k, u_k)}_{F_k(x_k, u_k)}\right) \tag{10.16}$$

by defining

$$F_k(x, u) = c(x, u) + \alpha \min_v \hat{Q}_k(X(x, u), v) - Q(x, u),$$

where $X(x, u)$ is a random sample state obtained by the Markov chain with state space $\mathcal{X}$ and transition matrix $P(u) = [P_{ij}(u)]$. Therefore, the Q-learning algorithm has the form of the process in the previous theorem with $a_n(x) = b_n(x) \leftarrow \varepsilon_n(x, u)$. It is now easy to see that

$$E[F_k(x, u)|\mathcal{F}_k] = T(\hat{Q}_k)(x, u) - Q(x, u),$$

where T is given by (10.5). Using the fact that T is a contraction mapping and $Q = T(Q)$, we further have:

$$E[F_k(x, u)|\mathcal{F}_k] = T(\hat{Q}_k)(x, u) - T(Q)(x, u).$$

Therefore,

$$\|E[F_k(x,u)|\mathcal{F}_k]\|_\infty = \|T(\hat{Q}_k) - T(Q)\|_\infty \leq \alpha\|\hat{Q}_k - Q\|_\infty = \alpha\|\Delta_k\|_\infty.$$

Finally,

$$\begin{aligned}\mathrm{Var}(F_k(x,u)|\mathcal{F}_k) &= E\left[(F_k(x,u) - E[F_k(x,u)|\mathcal{F}_k])^2\,|\mathcal{F}_k\right] \\ &= E\left[\left(c(x,u) + \alpha\min_v \hat{Q}_k(X(x,u),v) - T(\hat{Q}_k)(x,u)\right)^2|\mathcal{F}_k\right] \\ &= \mathrm{Var}\left(c(x,u) + \alpha\min_v \hat{Q}_k(X(x,u),v)|\mathcal{F}_k\right) \leq C(1+\|\Delta_k\|_\infty)^2,\end{aligned}$$

where the last step is due to the (at most) linear dependence of the argument on $\hat{Q}_k(X(x,u),v)$ and the underlying assumption that the variance of $c(x,u)$ is bounded. □

Asymptotic Convergence Rate of Q-learning: For discounted MDPs with discount factor $\frac{1}{2} < \alpha < 1$, the asymptotic rate of convergence of Q-learning is $O\left(\frac{1}{k^{\delta(1-\alpha)}}\right)$ if $\delta(1-\alpha) < \frac{1}{2}$ and $O\left(\sqrt{\frac{\log\log k}{k}}\right)$ if $\delta(1-\alpha) \geq \frac{1}{2}$, provided that the state-action pairs are sampled from a fixed probability distribution. Here, $\delta = \frac{p_{\min}}{p_{\max}}$ is the ratio of the minimum and maximum state-action occupation frequencies.

Remark: In the context of function approximation discussed in subsequent sections, we refer to the so-called *ODE method* to show convergence of stochastic recursions. In the end of this file, we provide a brief appendix with a comment on explicitly looking into Q-learning convergence via the ODE method. For simplicity, we focus there on the convergence of the *synchronous* Q-learning algorithm. A similar comment can be made for the asynchronous Q-learning algorithm as well.

10.3 Boltzmann and ϵ-Greedy Explorations

Boltzmann exploration chooses the next action according to the following probability distribution:

$$p(u_k = u|x_k = x) = \frac{\exp\left(-\frac{\hat{Q}_k(x,u)}{T}\right)}{\sum_v \exp\left(-\frac{\hat{Q}_k(x,v)}{T}\right)}. \tag{10.17}$$

T is called *temperature*. In the literature, Boltzmann exploration is also known as *softmax approximation* (when rewards are employed instead of costs and the sign in the exponents is plus instead of minus). In statistical mechanics, the softmax function is known as the *Boltzmann* or *Gibbs distribution*. In general, the softmax function has the following form,

$$\sigma(y;\beta)_i = \frac{\exp(\beta y_i)}{\sum_{j=1}^n \exp(\beta y_j)}, \quad \forall i = 1,...,n, y = (y_1,...,y_n) \in \mathbb{R}^n, \beta > 0. \tag{10.18}$$

It is easy to check that the softmax function defines a probability mass function (pmf) over the index set $\{1,...,n\}$. For $\beta = 0$ (or $T = \infty$), all indices are assigned an equal mass, i.e., the underlying pmf is uniform. As β increases, indices corresponding to larger elements in y are assigned higher probabilities. In this sense, "softmax" is a smooth approximation of the "max" function. When $\beta \to +\infty$, we have that

$$\lim_{\beta\to+\infty} \sigma(y;\beta)_i = \lim_{\beta\to+\infty} \frac{\exp(\beta y_i)}{\sum_{j=1}^n \exp(\beta y_j)} = \mathbf{1}_{\{i=\arg\max_j y_j\}}.$$

In other words, the softmax function assigns all the mass to the maximum element of y as $\beta \to +\infty$. We note here that this holds if the maximum element of y is unique, otherwise all the mass is assigned (uniformly) to the set of maxima of y.

In the form of (10.17), u_k becomes the solution of

$$\min_v \hat{Q}_k(x_k, v),$$

i.e., the policy becomes greedy as $T \to 0$ (In the form of (10.18), $\lim_{\beta \to -\infty} \sigma(y;\beta)_i = 1_{\{i=\arg\min_j y_j\}}$, if the minimum element of y is unique, otherwise all the mass is assigned (uniformly) to the set of minima of y).

In practice, depending on the learning goal, we may choose a temperature schedule such that $T_k \to 0$ as $k \to \infty$, which guarantees that $\arg\min_v \hat{Q}_k(x_k, v)$ is used as control in the limit $k \to \infty$. This corresponds to a *decaying exploration scheme*. We can also choose T to have a constant value. *Constant temperature Boltzmann exploration* corresponds to a *persistent exploration scheme*. We discuss decaying and persistent exploration shortly.

In **ϵ-greedy exploration**, $u_k = \arg\min_v \hat{Q}_k(x_k, v)$ is used with probability $1 - \epsilon$ and with probability ϵ any action chosen at random is taken. This corresponds to a *persistent exploration scheme*. The value of ϵ can be reduced over time, gradually moving the emphasis from exploration to exploitation. The last scheme is a paradigm of *decaying exploration*.

Remarks:

1. Although the above exploration methods are often used in practice, they are *local schemes*, i.e., they rely on the current state and therefore convergence is often slow.
2. More broadly, there are schemes with a more global view of the exploration process. Such schemes are designed to explore "interesting" parts of the state space. We will not pursue such schemes here.

Decaying and Persistent Exploration: We now clarify the difference between *decaying exploration* and *persistent exploration* schemes that exist in the literature. Decaying exploration schemes become over time greedy for choosing actions, while persistent exploration schemes do not. The advantage of decaying exploration is that the actions taken by the system may converge to the optimal ones eventually, but with the price that their ability to adapt slows down. On the contrary, persistent exploration methods can retain their adaptivity forever, but with the price that the actions of the system will not converge to optimality in the standard sense.

A class of decaying exploration schemes that is of interest in the sequel is defined as follows:

Definition 10.7. ***(Greedy in the Limit with Infinite Exploration) (GLIE)****: Consider exploration schemes satisfying the following properties:*

- *Each action is visited infinitely often in every state that is visited infinitely often.*
- *In the limit, the chosen actions are greedy with respect to the learned Q-function with probability* 1.

The first condition requires that exploration is performed indefinitely. The second condition requires that the exploration is decaying over time and the emphasis is gradually passed to exploitation.

ϵ-greedy GLIE exploration: Let $x_k = i$. Pick the corresponding greedy action with probability $1 - \epsilon_k(i)$ and an action at random with probability $\epsilon_k(i)$. Here, $\epsilon_k(i) = \frac{\nu}{N_k(i)}, 0 < \nu < 1$ and $N_k(i)$ is the number of visits to the current state $x_k = i$ by time k.

Boltzmann GLIE exploration:

$$p(u_k = u | x_k = x) = \frac{\exp\left(-\beta_k(x)\hat{Q}_k(x,u)\right)}{\sum_v \exp\left(-\beta_k(x)\hat{Q}_k(x,v)\right)}, \tag{10.19}$$

where

$$\beta_k(x) = \frac{\log N_k(x)}{C_k(x)} \quad \text{and} \quad C_k(x) \geq \max_{v,v'} |\hat{Q}_k(x,v) - \hat{Q}_k(x,v')|.$$

10.4 SARSA

As we already mentioned, in Q-learning, the policy used to estimate Q is irrelevant as long as the state-action space is adequately explored. In particular, to obtain $\hat{Q}_{k+1}(x_k, u_k)$ we use $\min_v \hat{Q}_k(x_{k+1}, v)$ in the Q-update instead of the actual control used in the next step. SARSA (State–Action–Reward–State–Action) is another scheme for updating the $Q-$function. The corresponding update in this case is:

$$\hat{Q}_{k+1}(x_k, u_k) = (1 - \varepsilon_k(x_k, u_k))\hat{Q}_k(x_k, u_k) + \varepsilon_k(x_k, u_k)\left(c(x_k, u_k) + \alpha\hat{Q}_k(x_{k+1}, u_{k+1})\right) \tag{10.20}$$

and $\hat{Q}_{k+1}(x,u) = \hat{Q}_k(x,u)$ for all $(x,u) \neq (x_k, u_k)$. Note that $\hat{Q}_k(x_{k+1}, u_{k+1})$ replaces $\min_v \hat{Q}_k(x_{k+1}, v)$ in (10.9).

This scheme also converges if the greedy policy $u_{k+1} = \arg\min_v \hat{Q}_{k+1}(x_{k+1}, v)$ is chosen as $k \to \infty$. In this case, due to this convergence,

$$\lim_{k\to\infty} \min_v \hat{Q}_{k+1}(x_{k+1}, v) = \lim_{k\to\infty} \min_v \hat{Q}_k(x_{k+1}, v)$$

for any fixed state $x_{k+1} = j$ and therefore, SARSA and Q-learning iterations will coincide in the limit.

Note: SARSA and Q-learning iterations coincide if a greedy learning policy is applied (i.e., always the greedy action with respect to the current Q-estimate is chosen).

Remarks:

1. SARSA is an *on-policy* scheme because in the update (10.20) the action u_{k+1} that was taken according to the underlying policy is used. Compare this update with the *off-policy* Q-learning update in (10.9), where the Q-values are updated using the greedy action corresponding to $\min_v \hat{Q}_k(x_{k+1}, v)$.

2. As we will see later on, when the underlying policy μ is fixed, SARSA is equivalent to TD(0) applied to state-action pairs.

3. The rate of convergence of SARSA coincides with the rate of convergence of TD(0) and is $O\left(\frac{1}{\sqrt{k}}\right)$. This is due to the fact that these algorithms are standard linear stochastic approximation methods. Nevertheless, the corresponding constant in this rate of convergence will be heavily influenced by the choice of the stepsize sequence, the underlying MDP model and the discount factor α.

4. SARSA can be extended to a multi-step version known as SARSA(λ). The notion of a *multi-step extension* will be clarified later on, in our discussion on TD learning and specifically on the TD(λ) algorithm. We will not pursue this concept any further here.

Convergence of SARSA: As mentioned earlier, for off-policy methods like Q-learning, the only requirement for convergence is that each state-action pair is visited infinitely often. For on-policy schemes like SARSA which learn the Q-function of the underlying learning policy, exploration has to eventually become small to ensure convergence to the optimal policy[6].

Theorem 10.8. *In finite state-action MDPs, the SARSA estimate $\hat{Q}_k$ converges to the optimal Q-function and the underlying learning policy μ_k converges to an optimal policy μ^* with probability 1 if the exploration scheme (or equivalently the learning policy) is GLIE and the following additional conditions hold:*

1. *The learning rates satisfy $0 \leq \varepsilon_k(i,u) \leq 1$, $\sum_k \varepsilon_k(i,u) = \infty$, $\sum_k \varepsilon_k^2(i,u) < \infty$ and $\varepsilon_k(i,u) = 0$ unless $(x_k, u_k) = (i,u)$.*
2. *$\mathrm{Var}(c(i,u)) < \infty$ (or $\mathrm{Var}(r(i,u)) < \infty$ for a problem with rewards).*
3. *The controlled Markov chain is* communicating*: every state can be reached from any other with positive probability (under some policy).*

Note: **Classification of MDPs**:

- **Recurrent or Ergodic**: The Markov chain corresponding to every deterministic stationary policy consists of a single recurrent class.
- **Unichain**: The Markov chain corresponding to every deterministic stationary policy consists of a single recurrent class plus a possibly empty set of transient states.
- **Communicating**: For every pair of states $(i,j) \in \mathcal{X} \times \mathcal{X}$, there exists a deterministic stationary policy under which j is accessible from i.

10.5 Q-approximation

Consider finite state-action spaces. If the size of the table representation of the Q-function is very large, then the Q-function can be appropriately approximated to cope with this problem. Function approximation can be also applied in the case of infinite state-action spaces.

10.5.1 Function approximation

Generic (Value) Function Approximation: To demonstrate some ideas, we first consider function approximation for a generic function $J : \mathcal{X} \to \mathbb{R}$. Let $\mathcal{J} = \left\{ J_\theta : \theta \in \mathbb{R}^K \right\}$ be a family of real-valued functions on the state space $\mathcal{X}$. Suppose that any function in $\mathcal{J}$ is a linear combination of a set of K fixed linearly independent (basis) functions $\phi_r : \mathcal{X} \to \mathbb{R}$, $r = 1, 2, \ldots, K$. More explicitly, for $\theta \in \mathbb{R}^K$,

$$J_\theta(x) = \sum_{r=1}^{K} \theta_r \phi_r(x) = \phi^T(x)\theta.$$

A usual assumption is that $\|\phi(x)\|_2 \leq 1$ uniformly on $\mathcal{X}$, which can be achieved by normalizing the basis functions. Compactly,

$$J_\theta = \Phi\theta,$$

[6]For example, SARSA is often implemented in practice with ϵ-greedy exploration for a diminishing ϵ (ϵ-greedy GLIE exploration).

where

$$J_\theta = \begin{bmatrix} J_\theta(1) \\ \vdots \\ J_\theta(X) \end{bmatrix} \quad \text{and} \quad \Phi = \begin{bmatrix} \phi^T(1) \\ \vdots \\ \phi^T(X) \end{bmatrix}.$$

The components $\phi_r(x)$ of vector $\phi(x)$ are called *features of state* x and Φ is called a *feature extraction*. Φ has to be judiciously chosen based on our knowledge about the problem.

Feature Selection: In general, features can be chosen in many ways. Suppose that $\mathcal{X} \subset \mathbb{R}$. We can then use a polynomial, Fourier or wavelet basis up to some order. For example, a polynomial basis corresponds to choosing $\phi(x) = [1, x, x^2, \ldots, x^{K-1}]^T$. Clearly, the set of monomials $\{1, x, x^2, \ldots, x^{K-1}\}$ is a linearly independent set and forms a basis for the vector space of all polynomials with degree $\leq K-1$. A different option is to choose an orthogonal system of polynomials, e.g., Hermite, Laguerre or Jacobi polynomials, including the important special cases of Chebyshev (or Tchebyshev) polynomials and Legendre polynomials.

If $\mathcal{X}$ is multi-dimensional, then the *tensor product construction* is a commonly used way to construct features. To elaborate, let $\mathcal{X} \subset \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$ and $\phi_i : \mathcal{X}_i \to \mathbb{R}^{k_i}, i = 1, 2, \ldots, n$. Then, the tensor product $\phi = \phi_1 \otimes \phi_2 \otimes \cdots \otimes \phi_n$ will have $\prod_{i=1}^n k_i$ components for a given state vector x, which can be indexed using multi-indices of the form $(i_1, i_2, \ldots, i_n), 1 \leq i_r \leq k_r, r = 1, 2, \ldots, n$. With this notation, $\phi_{(i_1,i_2,\ldots,i_n)}(x) = \phi_{1,i_1}(x_1)\phi_{2,i_2}(x_2) \cdot \phi_{n,i_n}(x_n)$. If $\mathcal{X} \subset \mathbb{R}^n$, then a popular choice for $\{\phi_1, \phi_2, \ldots, \phi_n\}$ are the *Radial Basis Function* (RBF) networks

$$\phi_i(x_i) = \left[G(|x_i - x_i^{(1)}|), \ldots, G(|x_i - x_i^{(k_i)}|)\right]^T, \quad x_i^{(r)} \in \mathbb{R}, \quad r = 1, 2, \ldots, k_i,$$

where G is some user-determined function, often a Gaussian function of the form $G(x) = \exp(-\gamma x^2)$ for some parameter $\gamma > 0$. Moreover, *kernel smoothing* is also an option for some appropriate choice of points $x^{(i)}, i = 1, 2, \ldots, K$:

$$J_\theta(x) = \sum_{i=1}^K \theta_i \underbrace{\frac{G(\|x - x^{(i)}\|)}{\sum_{j=1}^K G(\|x - x^{(j)}\|)}}_{\tilde{G}^{(i)}(x)}.$$

Here, $\tilde{G}^{(i)}(x) \geq 0, \forall x \in \mathcal{X}$ and $\forall i \in \{1, 2, \ldots, K\}$. Due to $\sum_{i=1}^K \tilde{G}^{(i)}(x) = 1$ for any $x \in \mathcal{X}$, J_θ is an example of an *averager*. Averager function approximators are *non-expansive* mappings, i.e., $\theta \to J_\theta$ is non-expansive in the maximum norm. Therefore, such approximators are suitable for use in reinforcement learning, because they can be nicely combined with the contraction mappings in this framework.

The previous brief reference to feature selection methods clearly does not exhaust the list of such methods. In general, many methods for this purpose are available. The interested reader is referred to the relevant literature.

Q-function approximation: In a similar spirit as before, consider basis functions of the form $\phi_r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, $r = 1, 2, \ldots, K$. Now $\phi_r(x, u), r = 1, 2, \ldots, K$ correspond to the features of a given state-action pair (x, u). For $\theta \in \mathbb{R}^K$,

$$Q_\theta(x, u) = \sum_{r=1}^K \theta_r \phi_r(x, u) = \phi^T(x, u)\theta.$$

Compactly,

$$Q_\theta = \Phi\theta,$$

where

$$Q_\theta = \begin{bmatrix} Q_\theta(1,1) \\ \vdots \\ Q_\theta(X,U) \end{bmatrix} \quad \text{and} \quad \Phi = \begin{bmatrix} \phi^T(1,1) \\ \vdots \\ \phi^T(X,U) \end{bmatrix}.$$

10.5.2 Actor-Critic Learning with Function Approximation

Under the described function approximation approach, the problem of learning an optimal policy can be formalized in terms of learning θ. If $K \ll |\mathcal{X} \times \mathcal{U}| = XU$, then we have to learn a much smaller vector θ than the Q-vector.

Consider first the following approximate Q-Value Iteration algorithm for a known MDP model:

- Set $t = 0$ and use an initial guess $\hat{\theta}_0$.
- Compute $\tilde{Q}_{t+1} = T(\hat{Q}_t)$, where $\hat{Q}_t = \Phi\hat{\theta}_t$, $T(\hat{Q})(x,u) = E[c(x_k,u_k) + a\min_v \hat{Q}(x_{k+1},v)|x_k = x, u_k = u]$.
- Find the best approximation to $\tilde{Q}_{t+1}$ by solving the following projection problem,

$$\underbrace{\min_\theta \frac{1}{2}\sum_{(x,u)} w(x,u)(\phi^T(x,u)\theta - \tilde{Q}_{t+1}(x,u))^2}_{\text{weighted least squares problem}} = \min_\theta \frac{1}{2}\left\|\Phi\theta - \tilde{Q}_{t+1}\right\|_W^2$$

  where $w(x,u)$ are some positive weights and $W = \text{diag}(w(1,1),\ldots,w(X,U))$. We note here that $\|x\|_W^2 = x^T W x$ and W is a symmetric positive definite matrix. Moreover, $\|x\|_W = \|W^{1/2}x\|_2$, where $W^{1/2}$ is the square root of W and $\|\cdot\|_2$ is the ℓ_2-norm[7].

- Set $t \leftarrow t+1$, update $\hat{\theta}_t$ as

$$\hat{\theta}_{t+1} = \arg\min_\theta \frac{1}{2}\left\|\Phi\theta - \tilde{Q}_{t+1}\right\|_W^2,$$

  and repeat.

Compact Version: More compactly, the above algorithm can be written as:

$$\hat{\theta}_{t+1} = \arg\min_\theta \frac{1}{2}\left\|\Phi\theta - T(\Phi\hat{\theta}_t)\right\|_W^2, t = 0,1,2,\ldots \tag{10.21}$$

for some initialization $\hat{\theta}_0$.

Fix now a stationary policy μ. Assume that the transition probabilities $P_{ij}(\mu(i))$ are known for this policy, as well as the cost structure. The associated Q-function at the state-action pairs $(i,\mu(i))$ (i.e., the corresponding value function J_μ) can be approximated and computed by straightforwardly adapting the above algorithm. To formalize the corresponding iteration, the operator T is defined in this case as:

$$T(\hat{Q})(x) = T(\hat{Q})(x,\mu(x)) = E[c(x_k,u_k) + a\hat{Q}(x_{k+1},\mu(x_{k+1}))|x_k = x, u_k = \mu(x)]. \tag{10.22}$$

The following example aims to show that the choice of the weighted norm $\|\cdot\|_W$ is critical for guaranteeing convergence of the above scheme.

10.5.2.1 Example

Consider a two-state MDP with no cost and let μ be a stationary policy. The corresponding Markov chain is[8]:

[7] The notation $\|\cdot\|_{W,2}$ is used sometimes for this norm to differentiate it, e.g., from definitions of weighted max-norms as in (10.12). Nevertheless, we simplify notation by discarding the subscript "2". We work exclusively with this weighted ℓ_2-norm for the rest of this file.

[8] A stationary policy and an MDP induce a *Markov reward process*. A more proper term in our case is a *Markov cost process*.

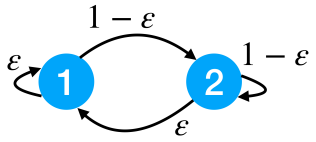

Figure 10.1: Two-state Markov chain.

We suppress from the subsequent notation of all the involved functions the action u, since the policy μ is assumed to be fixed and therefore $u = \mu(i)$ when at state i.

Since the underlying MDP model is known for this problem, we can implement the previously described algorithm for the operator T given by (10.22). First, for each node of the Markov chain we have:

$$\begin{aligned} T(\hat{Q})(1) &= c(1) + \alpha \sum_j P_{1j} \hat{Q}(j) \\ &= 0 + \alpha P_{11} \hat{Q}(1) + \alpha P_{12} \hat{Q}(2) \\ &= \alpha\varepsilon \hat{Q}(1) + \alpha(1-\varepsilon)\hat{Q}(2) \\ T(\hat{Q})(2) &= c(2) + \alpha \sum_j P_{2j} \hat{Q}(j) \\ &= 0 + \alpha P_{21} \hat{Q}(1) + \alpha P_{22} \hat{Q}(2) \\ &= \alpha\varepsilon \hat{Q}(1) + \alpha(1-\varepsilon)\hat{Q}(2) \end{aligned} \tag{10.23}$$

Compactly:

$$T(\hat{Q}) = \begin{bmatrix} T(\hat{Q})(1) \\ T(\hat{Q})(2) \end{bmatrix} = \begin{bmatrix} \alpha\varepsilon & \alpha(1-\varepsilon) \\ \alpha\varepsilon & \alpha(1-\varepsilon) \end{bmatrix} \begin{bmatrix} \hat{Q}(1) \\ \hat{Q}(2) \end{bmatrix}. \tag{10.24}$$

Since there is no cost associated with this problem, we can easily conclude that $Q(1) = Q(2) = 0$, where Q corresponds to the (optimal) Q-function and $Q(i) = Q(i, \mu^*(i)), i = 1, 2$ with μ^* being an optimal policy. Moreover, at any state any action is optimal and therefore μ is an optimal policy. Consider the function class:

$$\mathcal{Q} = \{\Phi\theta : \theta \in \mathbb{R}\}$$

with $\Phi = [1, 2]^T$. Clearly, $Q \in \mathcal{Q}$, since $Q = \Phi \cdot 0$. We now note that:

$$T(\Phi\theta) = \begin{bmatrix} \alpha\varepsilon & \alpha(1-\varepsilon) \\ \alpha\varepsilon & \alpha(1-\varepsilon) \end{bmatrix} \begin{bmatrix} \theta \\ 2\theta \end{bmatrix}. \tag{10.25}$$

If the standard Euclidean norm is used for projection, the projection objective becomes:

$$\min_\theta \frac{1}{2} \left\| \Phi\theta - \tilde{Q}_{t+1} \right\|_2^2 = \min_\theta \frac{1}{2}[\theta - \hat{\theta}_t(\alpha\varepsilon + 2\alpha(1-\varepsilon))]^2 + \frac{1}{2}[2\theta - \hat{\theta}_t(\alpha\varepsilon + 2\alpha(1-\varepsilon))]^2. \tag{10.26}$$

Differentiating with respect to θ and setting the derivative to 0, we obtain

$$\hat{\theta}_{t+1} = \underbrace{\frac{3}{5}\alpha(2-\varepsilon)}_{\tilde{\alpha}} \hat{\theta}_t = \tilde{\alpha}\hat{\theta}_t, \tag{10.27}$$

which corresponds to the exact form of (10.21) for this particular problem. If $\tilde{\alpha} > 1$, then the algorithm diverges (unless $\hat{\theta}_0 = 0$). Clearly, this can happen for many α and ε. Fortunately, using a different weighted norm, we can guarantee convergence for any α, ε.

Let

$$W = \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix} \tag{10.28}$$

for some positive constants w_1, w_2. Using $\|\cdot\|_W$ based on such a diagonal matrix in the projection objective, we have:

$$\min_{\theta} \frac{1}{2} \left\| \Phi\theta - \tilde{Q}_{t+1} \right\|_W^2 = \frac{1}{2} w_1 [\theta - \hat{\theta}_t(\alpha\varepsilon + 2\alpha(1-\varepsilon))]^2 + \frac{1}{2} w_2 [2\theta - \hat{\theta}_t(\alpha\varepsilon + 2\alpha(1-\varepsilon))]^2. \tag{10.29}$$

Differentiating with respect to θ and setting the derivative to 0, we obtain the recursion:

$$\hat{\theta}_{t+1} = \frac{\alpha(2-\varepsilon)(w_1 + 2w_2)}{(w_1 + 4w_2)} \hat{\theta}_t. \tag{10.30}$$

Idea: Choose the weights w_1, w_2 as the entries of the stationary distribution for the Markov chain in Fig. 10.1:

The stationary distribution of the Markov chain can be obtained by solving the equation $\pi = \pi P$ using the constraint $\sum_i \pi_i = 1$. $\pi = \pi P$ yields in this case $(1-\varepsilon)\pi_1 = \varepsilon\pi_2$. Furthermore, $\pi_1 + \pi_2 = 1$. Therefore, $\pi_1 = \varepsilon$ and $\pi_2 = 1 - \varepsilon$. We now let $w_1 = \pi_1$ and $w_2 = \pi_2$. The update rule can be rewritten as:

$$\begin{aligned} \hat{\theta}_{t+1} &= \frac{\alpha(2-\varepsilon)(\varepsilon + 2(1-\varepsilon))}{(\varepsilon + 4(1-\varepsilon))} \hat{\theta}_t \\ &= \underbrace{\alpha \left(1 - \frac{\varepsilon(1-\varepsilon)}{4 - 3\varepsilon} \right)}_{\tilde{a}} \hat{\theta}_t = \tilde{\alpha} \hat{\theta}_t. \end{aligned} \tag{10.31}$$

Clearly, $\tilde{\alpha} < 1$ for any α, ε in this case and therefore, the algorithm converges. This suggests that one should perhaps use $W = \mathrm{diag}(\pi)$ to define the projection objective, where π is the stationary distribution of the Markov chain for the underlying stationary policy (assuming that such an invariant measure exists).

10.5.2.2 Algorithm

With $W = \mathrm{diag}(\pi)$, the algorithm becomes:

$$\hat{\theta}_{t+1} = \arg\min_{\theta} \frac{1}{2} \left\| \Phi\theta - T(\Phi\hat{\theta}_t) \right\|_{\pi}^2, t = 0, 1, 2, \ldots \tag{10.32}$$

for some initial $\hat{\theta}_0$. Here, we abuse notation to write[9] $\|\cdot\|_\pi$ for $\|\cdot\|_{\mathrm{diag}(\pi)}$. Also, $\hat{Q} = [\hat{Q}(i)] = [\phi^T(i)\hat{\theta}]$.

10.5.2.3 Convergence Proof for (10.32)

Denote by Π the projection operator defined by the optimization problem:

$$\min_{\theta : \hat{Q} = \Phi\theta} \frac{1}{2} \left\| \hat{Q} - z \right\|_{\pi}^2,$$

[9] and of course we recall that this is a simplified notation for $\|\cdot\|_{\pi,2}$, which differs from the weighted max-norm in (10.12).

i.e., the solution of the above problem is $\bar{Q} = \Pi(z)$. Then, the algorithm given by (10.32) becomes

$$\hat{Q}_{t+1} = \Pi(T(\hat{Q}_t)). \tag{10.33}$$

We will show that $\Pi(T(\cdot))$ is a contraction mapping in $\|\cdot\|_\pi$, assuming that we know the underlying MDP model.

Proof. First, note that

$$\|\Pi(T(\hat{Q}_1)) - \Pi(T(\hat{Q}_2))\|_\pi \leq \|T(\hat{Q}_1) - T(\hat{Q}_2)\|_\pi, \tag{10.34}$$

because the projection operator Π is *non-expansive* with respect to $\|\cdot\|_\pi$. This can be easily proved by the first-order optimality condition for convex optimization and by applying Cauchy–Schwarz inequality (prove it!).

Moreover:

$$\begin{aligned}
\|T(Q_1) - T(Q_2)\|_\pi^2 &= \|\alpha P_\mu (Q_1 - Q_2)\|_\pi^2 \\
&= \alpha \sum_i \pi_i \left[\sum_j P_{ij}(\mu(i))(Q_{1j} - Q_{2j}) \right]^2 \\
&\leq \alpha \sum_i \pi_i \sum_j P_{ij}(\mu(i))(Q_{1j} - Q_{2j})^2 \qquad \left(\text{Jensen's inequality} : (E[x])^2 \leq E[x^2]\right) \\
&= \alpha \sum_j (Q_{1j} - Q_{2j})^2 \sum_i \pi_i P_{ij}(\mu(i)) \\
&= \alpha \sum_j \pi_j (Q_{1j} - Q_{2j})^2 \\
&= \alpha \|Q_1 - Q_2\|_\pi^2.
\end{aligned} \tag{10.35}$$

Combining the above results we obtain:

$$\|\Pi(T(\hat{Q}_1)) - \Pi(T(\hat{Q}_2))\|_\pi \leq \alpha \|Q_1 - Q_2\|_\pi^2, \tag{10.36}$$

which completes our proof. □

Conclusion: The algorithm (10.32) or equivalently (10.33) converges.

10.5.2.4 Unknown MDP Model: Learning

Assume now that the MDP model is unknown. Consider for the moment (10.32) and set

$$C(\theta; \tilde{Q}_{t+1}) = \frac{1}{2} \left\| \Phi\theta - \underbrace{T(\Phi\hat{\theta}_t)}_{\tilde{Q}_{t+1}} \right\|_\pi^2.$$

Then, $\hat{\theta}_{t+1}$ corresponds to the solution of

$$\nabla_\theta C(\theta; \tilde{Q}_{t+1}) = 0.$$

Performing the calculations

$$\begin{aligned}
\sum_i \pi_i \left(\phi^T(i)\theta - \tilde{Q}_{t+1}(i) \right) \phi(i) = 0 \\
\text{or} \quad \sum_i \pi_i \left(\phi^T(i)\theta - E\left[c(x_k) + \alpha \phi^T(x_{k+1})\hat{\theta}_t | x_k = i \right] \right) \phi(i) = 0
\end{aligned} \tag{10.37}$$

Idea: Similar to standard Q-learning or SARSA: Try the recursion:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + \varepsilon_t \left(c(x_t) + \alpha \phi^T(x_{t+1}) \hat{\theta}_t - \phi^T(x_t) \hat{\theta}_t \right) \phi(x_t). \tag{10.38}$$

Note: In practice, to evaluate the estimated Q-function at all state action-pairs, $u_t = \mu(x_t)$ is combined with some exploration process. The recursion can be then modified to account for u_t using $u_{t+1} = \mu(x_{t+1})$ in an online operation.

10.5.2.5 ODE method

Stochastic approximation suggests that under some conditions if the ODE

$$\dot{\theta} = -f(\theta)$$

converges to θ^* as $t \to \infty$, then the corresponding stochastic difference equation also converges to $\theta^\star$ almost surely. In our case, focusing again on a stationary policy μ:

$$f(\theta) = E\left[\left(\phi^T(x_t)\theta - c(x_t) - \alpha \phi^T(x_{t+1})\theta \right) \phi(x_t) \right], \tag{10.39}$$

where the expectation is taken in steady state (cf. (10.37)). The associated stochastic recursion is (10.38). Here, we have suppressed u_t, u_{t+1} from the notation, since $u_t = \mu(x_t), u_{t+1} = \mu(x_{t+1})$. Writing $f(\theta)$ more explicitly, we have:

$$f(\theta) = \sum_i \pi_i \left(\phi^T(i)\theta - c(i) - \sum_j P_{ij} \phi^T(j)\theta \right) \phi(i) = \sum_i \pi_i \left[\phi^T(i)\theta - T(\Phi\theta)(i) \right] \phi(i).$$

Let $D = \mathrm{diag}(\pi)$. Then,

$$f(\theta) = \Phi^T D \Phi\theta - \Phi^T D T(\Phi\theta).$$

Now the problem becomes finding the equilibrium θ^* which satisfies $f(\theta^*) = 0$. Solving this equation, we obtain:

$$\Phi^T D \Phi\theta^* - \Phi^T D T(\Phi\theta^*) = 0$$

or

$$\theta^* = \left(\Phi^T D \Phi \right)^{-1} \Phi^T D T(\Phi\theta^*). \tag{10.40}$$

10.5.2.6 Uniqueness of the Fixed Point Solution

Does the fixed point equation in (10.40) have a unique solution? Indeed, we are going to show that there is a unique solution θ^* to this equation.

Proof. We already know that $J = \Pi(T(J))$ has a unique solution because $\Pi(T(\cdot))$ is a contraction mapping. Recall that $\Pi(z)$ solves

$$\min_{\theta : y = \Phi\theta} \frac{1}{2} \|y - z\|_\pi^2 \tag{10.41}$$

or

$$\begin{aligned} &\min_\theta \frac{1}{2} \|\Phi\theta - z\|_\pi^2 \\ \text{or} \quad &\min_\theta \frac{1}{2} (\Phi\theta - z)^T D (\Phi\theta - z). \end{aligned} \tag{10.42}$$

Let $S(\theta) = \frac{1}{2}(\Phi\theta - z)^T D(\Phi\theta - z)$. By setting the gradient of $S(\theta)$ to zero, we have

$$\begin{aligned} & \nabla_\theta S(\theta) = 0 \\ \text{or} \quad & \Phi^T D(\Phi\theta - z) = 0 \\ \text{or} \quad & \theta = (\Phi^T D\Phi)^{-1}\Phi^T Dz. \end{aligned} \tag{10.43}$$

Multiplying both sides of the last equation by Φ and using the facts that $z = T(J)$ and $J = \Phi\theta$ we obtain:

$$J = \Phi\theta = \Phi(\Phi^T D\Phi)^{-1}DT(J) = \Phi(\Phi^T D\Phi)^{-1}DT(\Phi\theta). \tag{10.44}$$

Because $J = \Pi(T(J))$ has a unique solution, i.e., equation (10.44) has a unique solution $J^* = \Phi\theta^*$, the fixed point equation

$$\theta = (\Phi^T D\Phi)^{-1}\Phi^T DT(\Phi\theta) \tag{10.45}$$

has a unique solution θ^* (due to the implicit assumption that Φ is full column rank). □

Next, we will prove that the ODE $\dot{\theta} = -f(\theta)$ converges to θ^* as $t \to \infty$ using an appropriate Lyapunov function.

10.5.2.7 Convergence of the ODE to the Equilibrium Point

Recall that θ^* satisfies

$$\Phi^T D\Phi\theta^* - \Phi^T DT(\Phi\theta^*) = 0 \tag{10.46}$$

or equivalently

$$f(\theta^*) = 0. \tag{10.47}$$

We now have:

$$\begin{aligned} \dot{\theta} &= -f(\theta) \\ &= -f(\theta) + f(\theta^*) \\ &= -\Phi^T D\Phi(\theta - \theta^*) + \Phi^T D(T(\Phi\theta) - T(\Phi\theta^*)). \end{aligned} \tag{10.48}$$

Consider the following (usual) Lyapunov function:

$$L(\theta) = \frac{1}{2}\|\theta - \theta^*\|_2^2 = \frac{1}{2}(\theta - \theta^*)^T(\theta - \theta^*). \tag{10.49}$$

Taking the derivative of the Lyapunov function with respect to time, we obtain:

$$\begin{aligned} \dot{L}(\theta) &= (\nabla_\theta L(\theta))^T \frac{d\theta}{dt} \\ &= (\nabla_\theta L(\theta))^T \dot{\theta} \\ &= (\nabla_\theta L(\theta))^T [-f(\theta) + f(\theta^*)] \\ &= (\theta - \theta^*)^T [-\Phi^T D\Phi(\theta - \theta^*) + \Phi^T D(T(\Phi\theta) - T(\Phi\theta^*))] \\ &= -(\theta - \theta^*)^T \Phi^T D\Phi(\theta - \theta^*) + (\theta - \theta^*)^T \Phi^T D(T(\Phi\theta) - T(\Phi\theta^*)) \\ &= -\|\Phi(\theta - \theta^*)\|_\pi^2 + (\theta - \theta^*)^T \Phi^T D^{1/2} D^{1/2}(T(\Phi\theta) - T(\Phi\theta^*)). \end{aligned} \tag{10.50}$$

Recall the Cauchy–Schwarz inequality inequality:

$$|x^T y| \le \|x\|_2 \|y\|_2, \qquad \forall x, y \in \mathbb{R}^n. \tag{10.51}$$

We apply this inequality to the second term in (10.50) as follows:

$$\begin{aligned}(\theta-\theta^*)^T\Phi^T D^{1/2}D^{1/2}(T(\Phi\theta)-T(\Phi\theta^*)) &= (\theta-\theta^*)^T\Phi^T(D^{1/2})^T D^{1/2}(T(\Phi\theta)-T(\Phi\theta^*))\\ &= [D^{1/2}\Phi(\theta-\theta^*)]^T[D^{1/2}(T(\Phi\theta)-T(\Phi\theta^*))]\\ &\leq |[D^{1/2}\Phi(\theta-\theta^*)]^T[D^{1/2}(T(\Phi\theta)-T(\Phi\theta^*))]|\\ &\leq \|D^{1/2}\Phi(\theta-\theta^*)\|_2\|D^{1/2}(T(\Phi\theta)-T(\Phi\theta^*))\|_2\\ &= \|\Phi(\theta-\theta^*)\|_\pi\|T(\Phi\theta)-(\Phi\theta^*)\|_\pi.\end{aligned}$$

Since $T(\cdot)$ is a contraction mapping with respect to $\|\cdot\|_\pi$, we obtain:

$$\begin{aligned}\|\Phi(\theta-\theta^*)\|_\pi\|T(\Phi\theta)-(\Phi\theta^*)\|_\pi &\leq \|\Phi(\theta-\theta^*)\|_\pi\alpha\|\Phi\theta-\Phi\theta^*\|_\pi\\ &= \alpha\|\Phi(\theta-\theta^*)\|_\pi^2.\end{aligned}$$

As a consequence:

$$\dot{L}(\theta)\leq -(1-\alpha)\|\Phi(\theta-\theta^*)\|_\pi^2<0 \tag{10.52}$$

for any θ and $\dot{L}(\theta)=0$ only when $\theta=\theta^*$. Therefore, we conclude that $\dot{\theta}=-f(\theta)$ converges to θ^*. Thus, we also conclude that, under some conditions, the associated stochastic recursion (10.38) converges to θ^* almost surely as $t\to\infty$.

10.5.2.8 Actor-Critic Algorithm

We have just shown that our approximation scheme above can obtain the Q-function for a fixed policy μ at the state action pairs $(i,\mu(i))$. Therefore, for an initial policy μ_0, our algorithm can proceed as follows to find an *optimal* policy:

1. Using the previous scheme *with exploration*, estimate the Q-function for all state-action pairs when the underlying stationary policy used is μ_k. Let $\hat{Q}$ be the obtained estimate.
2. Next, update the policy to the greedy selection $\mu_{k+1}(x)=\arg\min_u \hat{Q}(x,u)$.
3. Repeat.

The above scheme corresponds to an actor-critic algorithm, which performs policy iteration with function approximation.

10.5.3 Q-learning with Function Approximation

Consider again the function class $\mathcal{Q}=\{Q_\theta=\Phi\theta:\theta\in\mathbb{R}^K\}$ for some feature extraction. Then, the *Q-learning algorithm with function approximation* is formalized as:

$$\hat{\theta}_{t+1}=\hat{\theta}_t+\varepsilon_t\left(c(x_t,u_t)+\alpha\min_v \phi^T(x_{t+1},v)\hat{\theta}_t-\phi^T(x_t,u_t)\hat{\theta}_t\right)\phi(x_t,u_t).$$

This iteration coincides with (10.38) when exploration is implemented, except for the definition of the temporal difference inside the parentheses which includes the minimization $\min_v \phi^T(x_{t+1},v)\hat{\theta}_t$ (off-policy characteristic) instead of $\phi^T(x_{t+1},u_{t+1})\hat{\theta}_t$.

Note: Q-learning is not guaranteed to converge when combined with function approximation.

10.6 Model-Free Policy Evaluation Methods

In this section we discuss **Monte Carlo policy evaluation** and **Temporal Difference learning**, which are model-free policy evaluation methods. Policy evaluation algorithms estimate the value function J_μ or Q_μ for some given policy μ. Many problems can be cast as value prediction problems. Estimating the probability of some future event, the expected time until some event occurs and J_μ or Q_μ underlying some policy μ in an MDP are all value prediction problems. In the context of actor-critic learning and policy evaluation, these are on-policy algorithms, and the considered policy μ is assumed to be stationary or *approximately* stationary. Monte-Carlo methods are based on the simple idea of using sample means to estimate the average of a random quantity. It turns out that the variance of these estimators can be high and therefore, the quality of the underlying value function estimates can be poor. Moreover, Monte Carlo methods in closed-loop estimation usually introduce bias. In contrast, TD learning can address these issues. As a final note, TD learning was introduced by Richard S. Sutton in 1988 and it is widely considered as one of the most influential ideas in reinforcement learning.

10.6.1 Monte Carlo Policy Evaluation

Monte Carlo methods learn from complete *episodes of experience* without using *bootstrapping*, i.e., sampling with replacement. The corresponding idea is described as follows: Suppose that μ is a stationary policy. Assume that we wish to evaluate the value function in the discounted scenario

$$J_\mu(i) = E^\mu \left[\sum_{k=0}^{\infty} \alpha^k c(x_k, u_k) \middle| x_0 = i \right]$$

or the total cost in an *episodial* problem

$$J_\mu(i) = E^\mu \left[\sum_{k=0}^{N} c(x_k, u_k) \middle| x_0 = i \right],$$

where the horizon N is a random variable. To evaluate the performance, multiple trajectories of the system can be simulated using policy μ starting from arbitrary initial conditions up to termination or stopped earlier (truncation of trajectories). Consider a particular trajectory. After visiting state i at time t_i, $J_\mu(i)$ is estimated as

$$\hat{J}_\mu(i) = \sum_{k=t_i}^{\tilde{N}} \alpha^{k-t_i} c(x_k, u_k) \quad \text{or} \quad \hat{J}_\mu(i) = \sum_{k=t_i}^{\tilde{N}} c(x_k, u_k)$$

for the discounted or the episodial problem. Here, $\tilde{N}$ is the terminal time instant of the corresponding trajectory. For the discounted cost problem, $\tilde{N}$ can be chosen large enough to guarantee a small error. Suppose further that we have M such estimates $\hat{J}_{\mu,1}(i), \hat{J}_{\mu,2}(i), \ldots, \hat{J}_{\mu,M}(i)$. Then, $\hat{J}_\mu(i)$ is finally estimated as

$$\hat{J}_\mu(i) = \frac{1}{M} \sum_{r=1}^{M} \hat{J}_{\mu,r}(i).$$

Clearly, obtaining such estimates for every $i \in \mathcal{X}$ corresponds to estimating J_μ.

Modes of Operation: Every state can be visited multiple times at every trajectory that we simulate using μ. This observation yields several ways of obtaining estimates $\hat{J}_{\mu,r}(i)$ for every $i \in \mathcal{X}$.

- $\hat{J}_\mu(i)$ is a single estimate of $J_\mu(i)$ if the trajectory starts at i and $\hat{J}_\mu(i)$ assumes summation up to $\tilde{N}$. Clearly, such an approach requires multiple trajectories to be initialized at every state $i \in \mathcal{X}$ to obtain M estimates $\hat{J}_{\mu,1}(i), \hat{J}_{\mu,2}(i), \ldots, \hat{J}_{\mu,M}(i)$ for every i.

- We can obtain an estimate $\hat{J}_{\mu,r}(i)$ each time state i is visited on the same trajectory. Every time the corresponding summation will be taken up to $\tilde{N}$. This approach implies that we can obtain multiple estimates $\hat{J}_{\mu,r}(i)$ per trajectory and across trajectories. Clearly, value estimates of this form from the same trajectory are heavily correlated.
- A single estimate $\hat{J}_{\mu,r}(i)$ can be at most obtained from any trajectory by performing a summation up to termination starting at the first visit of i in the course of a particular trajectory.

Result: $\hat{J}_\mu(i) \xrightarrow[M\to\infty]{} J_\mu(i), \forall i \in \mathcal{X}$ for all the above schemes if each state i is visited sufficiently often. This depends on *the choice of* μ and the *initial points* of the trajectories.

10.6.2 Temporal Difference Learning for Policy Evaluation

Consider an infinite discounted cost problem and let μ be a stationary policy. Recall the definition of the T_μ operator defined in earlier lectures:

$$T_\mu(J)(i) = E\left[c(x_0,\mu(x_0)) + \alpha\sum_j P_{x_0 j}(\mu(x_0)))J(j)\middle| x_0 = i\right] = \bar{c}(i,\mu(i)) + \alpha\sum_j P_{ij}(\mu(i))J(j)$$

or more compactly $T_\mu(J) = \bar{c}_\mu + \alpha P_\mu J$. Recall also that T_μ is a contraction mapping with fixed point the value function J_μ. The *fixed policy Value Iteration method* computes J_μ via successive approximations by performing the recursion $J_{k+1} = T_\mu(J_k)$ for some initial guess J_0 of J_μ. By the contaction property of T_μ, $J_k \xrightarrow[k\to\infty]{} J_\mu$.

Assume now that the underlying MDP model is unknown. We therefore require some form of learning of J_μ in this case. Let $\hat{J}_0$ be an initial guess of J_μ, e.g., $\hat{J}_0 = J_0$. Suppose that we simulate the system using policy μ and at times k and $k+1$ we observe $x_k, c(x_k,\mu(x_k))$ and x_{k+1} (we also observe $c(x_{k+1},\mu(x_{k+1}))$). Let $\hat{J}_k$ be an estimate of J_μ at time k. Then, $c(x_k,\mu(x_k)) + \alpha\hat{J}_k(x_{k+1})$ is clearly a noisy estimate of $T_\mu(J_k)(x_k)$ in the fixed policy Value Iteration recursion, i.e., of $J_{k+1}(x_k)$. Because the aforementioned estimate is noisy, we can use it to update $\hat{J}_k(x_k)$ only by a sufficiently small amount which guarantees convergence.

Idea: Stochastic Approximation Recursion similar to Q-learning, SARSA, etc.:

$$\begin{aligned}\hat{J}_{k+1}(x_k) &= (1-\varepsilon_k(x_k))\hat{J}_k(x_k) + \varepsilon_k(x_k)\left(c(x_k,\mu(x_k)) + \alpha\hat{J}_k(x_{k+1})\right)\\ &= \hat{J}_k(x_k) + \varepsilon_k(x_k)\left(\underbrace{c(x_k,\mu(x_k)) + \alpha\hat{J}_k(x_{k+1}) - \hat{J}_k(x_k)}_{\delta_k}\right)\\ &= \hat{J}_k(x_k) + \varepsilon_k(x_k)\delta_k, \end{aligned} \tag{10.53}$$

where δ_k is the so called **temporal difference** or **temporal difference error**. We note here that the update is asynchronous in the sense that if $x_k = i$, then $\hat{J}_{k+1}(i) = \hat{J}_k(i) + \varepsilon_k\delta_k$ and $\hat{J}_{k+1}(j) = \hat{J}_k(j)$ for all $j \neq i$. This scheme is known in the literature as the **TD(0) algorithm**. If $\varepsilon_k(x_k) \leq 1$, then the convex combination in the first line of (10.53) implies that $\hat{J}_{k+1}(x_k)$ is moved by an amount controlled by $\varepsilon(x_k)$ towards $c(x_k,\mu(x_k)) + \alpha\hat{J}_k(x_{k+1})$ incorporating new information into the value estimate. Also, since $c(x_k,\mu(x_k)) + \alpha\hat{J}_k(x_{k+1})$ incorporates $\hat{J}_k(x_{k+1})$ which is itself an estimate, i.e., the new information also relies on the current value estimate $\hat{J}_k$, the method uses **bootstrapping**.

Convergence properties of TD(0) algorithm:

1. Using the previously mentioned stepsize sequence $\varepsilon_k(i) = \frac{\varepsilon}{N_k(i)}$ for some $\varepsilon > 0$, where $N_k(i)$ is the number of visits to state i by time k and assuming that all states are visited *infinitely often*, $\hat{J}_k \xrightarrow[k\to\infty]{} J_\mu$ with probability 1.

To elaborate a bit more on this point, we note that if TD(0) converges, then it must converge to some $\tilde{J}$ such that[10]

$$E[\delta_k | x_k = i, \hat{J}_k = \tilde{J}] = E[c(x_k, \mu(x_k)) + \alpha \hat{J}_k(x_{k+1}) - \hat{J}_k(x_k) | x_k = i, \hat{J}_k = \tilde{J}] = 0, \quad \forall i \in \mathcal{X}.$$

Equivalently, $T_\mu(\tilde{J}) - \tilde{J} = 0$ and since T_μ has a unique fixed point, we conclude that $\tilde{J} = J_\mu$. Moreover, with the assumptions $\sum_k \varepsilon_k(i) = \infty, \sum_k \varepsilon_k^2(i) < \infty$ and by the ODE method, TD(0) will follow closely trajectories of the linear ODE:

$$\dot{J} = \bar{c}_\mu + (\alpha P_\mu - I)J.$$

The eigenvalues of $\alpha P_\mu - I$ lie in the open left half complex plane[11] and hence, the ODE is **globally asymptotically stable**. Therefore, by the ODE method for stochastic approximation, $\hat{J}_k \xrightarrow[k\to\infty]{} J_\mu$ with probability 1.

2. If $\varepsilon_k = \varepsilon > 0$ and each state is visited infinitely often, then "tracking" is achieved, i.e., $\hat{J}_k$ will be close to J_μ in the long run. More precisely, $\forall \epsilon, \delta > 0$

$$\lim_{k\to\infty} P(|\hat{J}_k - J_\mu| > \delta) \leq \epsilon,$$

for sufficiently small $\varepsilon > 0$.

TD with ℓ-step look ahead: Instead of δ_k, one can use

$$\sum_{r=0}^{l-1} \alpha^r \delta_{k+r} = \sum_{r=0}^{l-1} \alpha^r c(x_{k+r}, \mu(x_{k+r})) + \alpha^l \hat{J}_k(x_{k+l}) - \hat{J}_k(x_k)$$

in (10.53):

$$\hat{J}_{k+1}(x_k) = \hat{J}_k(x_k) + \varepsilon_k(x_k) \sum_{r=0}^{l-1} \alpha^r \delta_{k+r}. \tag{10.54}$$

Here, $\hat{J}_k(\cdot)$ is the value function estimate used in δ_{k+r} for any r.

TD(λ) Algorithm with $0 \leq \lambda \leq 1$: Considering all future temporal differences with a *geometric* weighting of decreasing importance we obtain the following algorithm:

$$\hat{J}_{k+1}(x_k) = \hat{J}_k(x_k) + \varepsilon_k(x_k) \sum_{r=0}^{\infty} (\lambda\alpha)^r \delta_{k+r}. \tag{10.55}$$

Again, $\hat{J}_k(\cdot)$ is the value function estimate used in δ_{k+r} for any r.

λ is called **trace-decay parameter** and it controls the amount of *bootstrapping*: For $\lambda = 0$ we obtain the TD(0) algorithm and for $\lambda = 1$ a Monte Carlo method (or the TD(1) method) for policy evaluation.

Convergence of TD(λ): Similar to TD(0) but often faster than TD(0) and Monte-Carlo methods if λ is judiciously chosen. This has been experimentally verified, especially when function approximation is used for the value function. In practice, good values of λ are determined by trial and error. Also, the value of λ can be modified even during the algorithm without affecting convergence.

[10] More generally, this condition should hold at least for all states that are sampled infinitely often.

[11] $\lambda_i(\alpha P_\mu - I) = \alpha\lambda_i(P_\mu) - 1$ and since $\alpha|\lambda_i(P_\mu)| < 1$, it turns out that $\mathrm{Re}\{\alpha\lambda_i(P_\mu) - 1\} < 0$.

A An explicit view of synchronous Q-learning convergence via the ODE method

Consider the first equation in the synchronous Q-learning recursion (10.11). Compactly, this recursion can be written in matrix form as follows:

$$\hat{Q}_{k+1} = \hat{Q}_k + \varepsilon_k \left(h(\hat{Q}_k) + M_k \right). \tag{10.56}$$

Here, $\hat{Q}_k = [\hat{Q}_k(x,u)]$ is the $X \times U$ matrix of Q-value estimates at time k, $h(\hat{Q}_k)$ is the $X \times U$ matrix

$$h(\hat{Q}_k) = [h(\hat{Q}_k)(x,u)] = [T(\hat{Q}_k)(x,u) - \hat{Q}_k(x,u)]$$

and M_k is the $X \times U$ (martingale difference) matrix

$$M_k = [M_k(x,u)] = \alpha \left[\min_v \hat{Q}_k(x_{k+1}(x,u), v) - \sum_j P_{xj}(u) \min_v \hat{Q}_k(j,v) \right].$$

Suppose that the usual conditions hold:

$$\sum_k \varepsilon_k = \infty \quad \text{and} \quad \sum_k \varepsilon_k^2 < \infty.$$

Then, the associated ODE for the last stochastic matrix recursion is:

$$\dot{Q}(t) = T(Q)(t) - Q(t).$$

Recall that T is a contraction mapping with parameter α. It turns out that this ODE has a unique globally stable fixed point, the synchronous Q-learning algorithm is stable and the stochastic recursion (10.56) converges to the optimal Q, which is the unique fixed point of T.