McGill University

ECSE 506

Course Project

Reinforcement Learning using TD (λ) and Q-Learning

Author
Sadia Khaf

Course Instructor
Dr. Aditya Mahajan

April 28, 2020

Contents

Chapter 1

Introduction

In this Chapter, we introduce reinforcement learning, and define Q-factors and policy iteration that form the basis of reinforcement learning.

1.1 Reinforcement Learning

Contrary to supervised and un-supervised machine learning algorithms, reinforcement learning works on the principle of some immediate and/or delayed *reward*. An *agent* interacts with its *environment* by taking certain *actions* that maximize its cumulative rewards. Reinforcement learning represents a class of learning algorithms where no explicit *labels* or training data is given to the algorithm. Instead, some actions are reinforced by a high reward and others are discouraged by low or negative rewards. The trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning [1]. The examples of agents and environments for reinforcement learning are

- Moves made by players in a game of chess, a move may seem detrimental at first but may have a high long-term benefit. This is an example of high delayed reward.
- A robot vacuum cleaner making decisions about whether to collect more trash or go back to charging station.
- A dog learning to play fetch through positive reinforcement.

In all of these examples, the agent's actions affect the future state of the environment but they cannot fully predict it so the agent has to frequently observe its environment and decide the future actions accordingly.

1.2 Q-Factors and Policy Iteration

The *Q-factor* of a state control pair (i, u) using a stationary policy μ is defined as,

$$Q^{\mu}(i,u) = \sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + J^{\mu}(j)), \tag{1.1}$$

which is the cost for starting in state i, choosing policy u in that state, and choosing policy μ in all subsequent states. The transition probability from state i to state j using policy u is represented by $p_{ij}(u)$, and $g(i,u,j)$ represents corresponding per step cost. One way to evaluate J^{μ} for calculating Q-factors is the Monte Carlo (MC) simulation method where the cost-to-go for each state is calculated from multiple trajectories involving state i, averaged over the number of times state i is visited [2]. In MC simulations, all trajectories need to have a terminal state, and an update of Q-factors is performed at the end of the episode i.e., when the terminal state has been reached. Rewards or returns in MC are averaged over several episodes.

After evaluating the Q-factors, the policy improvement step can be performed as,

$$\bar{u}(i) = \arg\min_{u \in \mathcal{U}} Q^{\mu}(i,u), \quad i = 1, \ldots, n, \tag{1.2}$$

which is essentially an approximate policy iteration since the Q-factors and the cost-to-go functions are the Monte Carlo simulation based estimates rather than exact values. Great care must be exercised in choosing the initial states of the simulated trajectories since cost-to-go estimates will be based on cost samples chosen from the states visited by these trajectories. We may have large number of samples from frequently visited states but fewer samples from others, and in the estimates may be inaccurate for the less frequently visited states.

In this Chapter, we introduced the basic notations for Q-factors that will be used throughout this report. In the next Chapter, we introduce the temporal difference methods and their variants.

Chapter 2

Temporal Difference Methods

In this Chapter, we discuss the cost-to-go update methods using Monte Carlo policy evaluation algorithm that incrementally updates $J(i)$. For any trajectory $i_0, i_1, \ldots, i_N$, with $i_N = 0$, we define $i_k = 0$ for all $k > N$ and similarly $g(i_k, i_{k+1}) = 0$ for $k \geq N$. Once a trajectory $(i_0, i_1, \ldots, i_N)$ is generated, the cost estimates $J(i_k)$, $k = 0, \ldots, N-1$, are updated according to

$$J(i_k) := J(i_k) + \gamma(g(i_k, i_{k+1}) + g(i_{k+1}, i_{k+2}) + \cdots + g(i_{N-1}, i_N) - J(i_k)), \tag{2.1}$$

where γ is the step size that can vary from iteration to iteration (usually set to $1/N$).

2.1 TD (λ)

The Robbins-Monro stochastic approximation method based on Bellman's update method is,

$$J(i_k) := J(i_k) + \gamma \sum_{m=k}^{\infty} \lambda^{m-k} d_m, \tag{2.2}$$

where γ is the step size and can change from one iteration to another, and the *temporal difference* d_m is defined as,

$$d_m = g(i_m, i_{m+1}) + J^{\mu}(i_{m+1}) - J^{\mu}(i_m). \tag{2.3}$$

The family of algorithms defined in Eq. 2.2 is known as TD(λ), for each value of λ. These algorithms are a trade-off between Monte Carlo, where all stages until the terminal state are considered, and TD(0) or one step TD where only the immediate step is considered.

Trade-off in Choosing λ

The value $\lambda = 1$ corresponds to the case of Monte Carlo policy evaluation method, and the case $\lambda = 0$ refers to the one step Bellman policy evaluation. That value of λ between 0 and 1 determines the discounting effect on the cost estimates of the current state by the *temporal differences* of state transitions far in the future. This is different from the discount factor β in the dynamic programming, which refers to the urgency of the problem, or the importance of immediate costs/rewards compared to the ones far in the future. Since TD(0) uses only one step to perform the update, and the reward/cost that we observe may have been caused by an action several time steps back in the past, TD(0) will generally be slow to converge, and is biased. On the other hand, TD(1) is unbiased but will have a high variance since it uses the outcome of the whole trajectory to perform an update.

Offline and Online Variants

To understand the offline and online variants of the algorithm, we define the *first visit* and *every visit* update methods. The every visit method is given by,

$$J(i) := J(i) + \gamma \sum_{j=1}^{M} \sum_{m=m_j}^{\infty} \lambda^{m-m_j} d_m, \tag{2.4}$$

a state is counted each time it is visited, unlike the first visit method where it is counted only the first time it is visited. The first visit variant is given by,

$$J(i) := J(i) + \gamma \sum_{m=m_1}^{\infty} \lambda^{m-m_1} d_m, \tag{2.5}$$

where m_1 is the time of the first visit to state i. The offline variant is the simplest case of TD(λ) where all of the updates are carried out simultaneously according to Eq. 2.4 whereas the online variant evaluates the running sums one term at a time, following each transition. The difference between the two variants is of the second order in γ and becomes inconsequential as γ diminishes to zero but their useful varies depending on the scenario and can be found in detail in [2].

2.2 Convergence of Offline TD Methods

The convergence of offline TD methods is shown in [2]. The complete proof is fairly long but we will discuss the important assumptions and propo-

sitions. The notation z_m^t is used to represent the *eligibility traces* and the algorithm can be written as:

$$J_{t+1}(i) = J_t(i) + \gamma_t(i) \sum_{m=0}^{N_t-1} z_m^t(i) d_{m,t}, \tag{2.6}$$

where $d_{m,t} = g(i_m^t, i_{m+1}^t) + J_t(i_{m+1}^t) - J_t(i_m^t)$. The convergence follows from Prop. 4.5 in [2] which proves the convergence of stochastic approximation methods based on weighted maximum norm pseudo-contractions using the assumptions discussed below.

2.2.1 Assumptions on Eligibility Coefficients and their Impact

The assumptions on eligibility coefficients simply state $z_m(i) \geq 0$, the coefficients remain zero until the first time the state i is visited, and are incremented at most by 1 with each visit to the state. If the increment assumption does not hold, the algorithm and the convergence results will still hold for single visit variant but will not capture the multiple visits variant of the algorithm. It is also assumed at these coefficients are completely determined by $i_0^t, \ldots, i_m^t$.

2.2.2 Assumptions on Step-size and their Impact

Let us define $T^i = \{t | q_t(i) > 0\}$. The proof requires the following assumptions, among others, on the step-sizes γ_t.

Assumptions on Step-sizes γ_t

1. $\gamma_t(i) \geq 0$ for all $t \in T^i$, and $\gamma_t(i) = 0$ for $t \notin T^i$.
2. $\sum_{t \in T^i} \gamma_t(i) = \infty$ for all i.
3. $\sum_{t \in T^i} \gamma_t^2(i) < \infty$ for all i.

The most common choices for the step sizes are $1/t$, $1/k$, and $1/k+1$ with k being the number of times an update of $J(i)$ was carried out. Mathematically, assumption 2 is not enforced automatically for $1/t$ unless an additional assumption is imposed that requires $t \in T^i$ at least once in K time steps where K is a constant. In most reinforcement learning techniques, this is natural and loosely translates to the more familiar assumption that *every state is visited frequently often*. For step size $1/k$ we would have to

check at each time step if t belongs to T^i which may not be easy to enforce. Naturally, the third choice is the easy to verify and easy to enforce.

2.3 Convergence of Online TD Methods

The proof of convergence for the online version of the algorithm puts an additional assumption on the step-stizes. The rest of the proof follows a similar structure and uses Prop. 4.5 in [2] to show convergence.

Additional assumption on eligibility coefficients

Assume that coefficients $z_m^t(i)$ are bounded above by a deterministic constant C.

2.3.1 Effect of Additional Assumption

The additional assumption is met easily for the first-visit TD(λ) for all values of λ because the previous assumption on increment by at most 1 bounds $z_m^t(i)$ by 1. For the cases with $\lambda < 1$, the eligibility coefficients are bounded, because of the Geometric series sum, by $1/(1-\lambda)$. For the every-visit case of TD(1), the authors still believe that the proof of convergence holds but a concrete proof is not provided. Some of the later works like [3] suggested working with *replacing traces* method that works by resetting the trace to 1 every time a state is visited, rather than adding to the previous value. The suggested method also bounds the coefficients to 1 and the proof of convergence holds.

In this Chapter, we discussed TD(λ), its variants, and proofs of convergence of the algorithms along with the effect of various assumptions. In the next Chapter, we introduce Q-learning.

Chapter 3

Q-Learning

In this Chapter, we present the model free computational method called Q-learning, the algorithm and the convergence of Q-learning, and the discounted Q-learning method. Q- learning is used whenever there is no explicit structure of the cost function and the model of the system is not available. Q-learning works by updating the optimal Q-factors. Let us define the optimal Q-factor $Q^*(i,u)$ corresponding state-action pair (i,u), with $u \in U(i)$, by letting $Q^*(0,u) = 0$ and,

$$Q^*(i,u) = \sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + J^*(j)), \quad i = 1,\ldots,n \tag{3.1}$$

where,

$$J^*(i) = \min_{u \in U(i)} Q^*(i,u). \tag{3.2}$$

Combining 3.1 and 3.2, we get,

$$Q^*(i,u) = \sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + \min_{v \in U(i)} Q^*(i,v)). \tag{3.3}$$

3.1 The Q-Learning Algorithm

The value iteration in terms of Q-factors can be written as,

$$Q(i,u) := (1-\gamma)Q(i,u) + \gamma \sum_{j=0}^{n} p_{ij}(u)(g(i,u,j) + \min_{v \in U(j)} Q(j,v)), \tag{3.4}$$

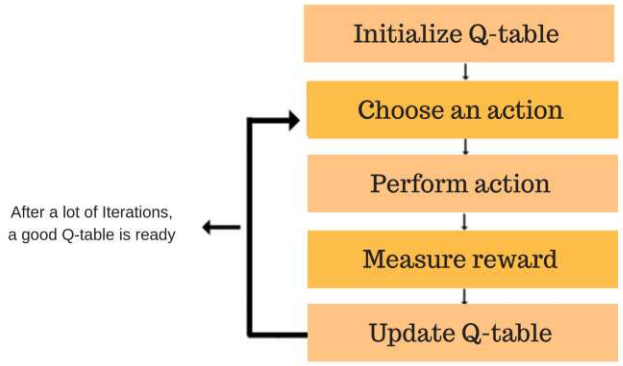

Figure 3.1: Q-learning Algorithm

where $\gamma \in (0, 1]$ is the step size that can vary from one iteration to another. The Q-learning algorithm is an approximate version of this iteration where the expectation w.r.t. j is replaced by a single sample as,

$$Q(i,u) := (1-\gamma)Q(i,u) + \gamma(g(i,u,j) + \min_{v \in U(j)} Q(j,v)), \tag{3.5}$$

where j and $g(i,u,j)$ are generated from (i,u) by simulation. The general form of Q-learning algorithm is given in [1]. More specifically, time t is assumed to be discrete and $T^{i,u}$ is the set of times update $Q(i,u)$ is performed. Let $\mathcal{F}_t$ denote the history of algorithm up to and including the time when step-size $\gamma_t(i,u)$ is chosen, but before the simulated transitions and costs are generated. The update algorithm is,

$$Q_{t+1}(i,u) = (1-\gamma_t(i,u))Q_t(i,u) + \gamma_t(i,u)(g(i,u,\bar{i}) + \min_{v \in U(\bar{i})} Q_t(\bar{i},v)), \tag{3.6}$$

where the state $\bar{i}$ is picked at random with $p_{i\bar{i}}(u)$. Moreover, $\gamma_t(i,u) = 0$ for $t \notin T^{i,u}$, and $Q_t(0,u) = 0$ for all t. The Q-table is a term used for a lookup table where we store the value of Q-factors for each state action pair and Fig. 3.1 shows the way this table is updated.

3.2 The Convergence of Q-Learning

In this section, we prove that Q-learning algorithm converges to the optimal Q-factors $Q^*(i,u)$ with probability 1.

Convergence conditions

Theorem 3.2.1 *$Q_t(i,u)$ converges to $Q^*(i,u)$ with probability 1 if $Q_t(i,u)$ is guaranteed to be bounded with probability 1, there exists a proper policy, and, $\sum_{t=0}^{\infty}\gamma_t(i,u)=\infty, \quad \sum_{t=0}^{\infty}{\gamma_t}^2(i,u)<\infty, \quad \forall i,u\in U(i)$*

Proof: The complete proof can be found in [2] and [4]. We will summarize the highlights here. The proof starts by defining a mapping H,

$$(HQ)(i,u)=\sum_{j=0}^{n}p_{ij}(u)(g(i,u,j)+\min_{v\in U(j)}Q(j,v)), \quad i\neq 0, u\in U(i). \quad (3.7)$$

The proof proceeds by showing that H is a weighted maximum norm contraction. The convergence of the algorithm is shown by the following proposition.

Proposition

Proposition 3.2.1 *Let r_t be the sequence generated by the iteration $r_{t+1}(i)=(1-\gamma_t(i))r_t(t)+\gamma_t(i)((Hr_t)(i)+w_t(i))$. We assume the following.*

1. *The stepsizes $\gamma_t(i)$ are non-negative and satisfy $\sum_{t=0}^{\infty}\gamma_t(i)=\infty, \quad \sum_{t=0}^{\infty}{\gamma_t}^2(i)=\infty$*
2. *The noise terms satisfy*
   (a) *For every i and t we have $\mathbb{E}[w_t(i)|\mathcal{F}_t]=0$.*
   (b) *Given any norm $||.||$ on $\mathbb{R}^n$, there exist constants A and B such that $\mathbb{E}[w_t^2(i)|\mathcal{F}_t]\leq A+B||r_t||^2, \quad \forall i,t$.*
3. *The mapping H is a weighted maximum norm pseudo-contraction.*

Then, r_t converges to r^* with probability 1. Furthermore, it is proved in the end that when all one stage costs $g(i,u,j)$ are non-negative, and the step-sizes satisfy $\gamma_t(i,u)\leq 1$ and $Q_0(i,u)\geq 0, \quad \forall(i,u)$ then the sequence Q_t generated by Q-learning algorithm is bounded and therefore converges to Q^*. The simplest way to summarize it is that algorithm converges as long as all actions are repeatedly sampled in all states and the action-values are represented discretely.
Proof: See [2].

3.3 Q-Learning for discounted problems

A discounted Q-learning algorithm is defined as,

$$Q(i,u) := (1-\gamma)Q(i,u) + \gamma(g(i,u,j) + \alpha \min_{v \in U(j)} Q(j,v)), \qquad (3.8)$$

where α is the discount factor. A discounted problem can also be handled by converting it into a stochastic shortest path problem. The discounted Q-factors also converge to the optimal Q-factors $Q^*(i,u)$ with probability 1 under the same assumptions on the step-sizes.

3.4 Exploration vs. Exploitation

All possible states and all potentially beneficial actions should be explored for Q-learning algorithm to perform well. Under a greedy policy, only $(i, \mu(i))$ pairs are considered, where μ is the current greedy policy and certain profitable actions u may never be explored. The notion of exploration in Q-learning algorithms introduces a way to choose some actions randomly at some times. One possible way is to choose the greedy action with probability $1-\epsilon$ and choose a random action with probability ϵ. Some works also suggest gradually changing the value of ϵ as the algorithm progresses [1]. Another way is to choose some intervals in which actions are chosen randomly. An improvement over ϵ greedy methods is the contextual ϵ greedy methods where the agent is encouraged to explore in non-critical states but exploit with a higher probability in critical states.

In this Chapter, we discussed Q-learning algorithm and the assumptions under which Q-learning algorithm converges to the optimal Q values. In the next Chapter, we discuss a critical analysis of the differences between TD(λ) and Q-learning.

Chapter 4

Critical Analysis of Techniques

In this Chapter, we discuss our analysis of some key parameters and assumptions about Q-learning and TD(λ). We also discuss how certain assumptions can be imposed/relaxed in the context of cognitive radio networks.

4.1 TD (λ) vs Q-Learning

The key difference between Q-learning and TD(λ) is that in Q-learning, we use a probabilistic estimate of *all* the trajectories and minimize over them to chose the best action at each step whereas in Monte Carlo and TD, we sample one of the possible trajectories, calculate estimated returns from it, and then average over multiple episodes. Its a complete trajectory till the end of the episode in case of MC and can be a partial one in case of TD. Another difference is that in Q-learning the Q-factors of state action pairs (i, u) are updated after each iteration, i.e., they are replaced by the new estimates and the process continues. In TD methods however, we update the estimates of cost-to-go not only for the immediate preceding step but other steps further back in the past (or future in the future view of TD methods, both future view and past view are equivalent). The eligibility coefficients in TD(λ) enable us to incorporate the information about frequency and recency of visits to different states with respect to the final state which is not incorporated in Q-learning. Q-learning can be considered TD(0) method since it is based on just the immediate one step reward. On the other hand, in MC methods, update for each state is based on the entire sequence of received rewards from that state until the end of the episode. The N-step TD methods lie between MC and TD(0), since the update is based on an

intermediate number of steps, more than one but less than all of them until termination. Therefore, Q-learning algorithm is reinforcement learning problems is called one-step TD method, while MC-methods are called infinite-step TD methods.

4.2 Limitations of the Modeling Assumptions

In this section we briefly discuss some modeling assumptions and what happens if they are violated.

4.2.1 What happens if the state space is not finite?

Both Q-learning and TD(λ) assume finite and discrete state space but what happens if the state space is not finite? When the state space is infinite or too large, tabular reinforcement learning becomes infeasible. To solve this limitation, function approximation to model the value function or policy can be used. Practical examples include deep-reinforcement learning and deep Q Networks (DQNs). We can model the infinite or very large size of state using neural networks (those states are the inputs to the network), the output of neural network is still required to be finite. In other words, in case of DQN, the outputs of neural network correspond to the predicted Q-values of all possible actions. Therefore, normal DQN methods cannot be used in continuous action spaces. There are ways to extend reinforcement learning to continuous action spaces as well, including actor-critic methods and policy gradient methods. Truncation and discretization is another way to handle continuous and infinite state spaces by putting some upper confidence bounds on the discretization error. Doya extended Sutton's work on reinforcement learning to continuous time and space in [5].

4.3 Speed of convergence compared to Value Iteration

The temporal difference methods and Q-learning algorithm work with simulation based samples of the value functions and Q-factors. Contrary to that, dynamic programming, and consequently value iteration, require the knowledge of transition probabilities. In other words, value iteration is a model based method whereas Q-learning and TD learning are model free methods. For small MDPs with known dynamics, value iteration often converges very fast but for large sate space, it involves solving a system of large number of linear equations which becomes very difficult. Q-learning

and TD methods obviously converge slower than value iteration because they learn from trial and error and have to *explore* before exploiting but can be made more efficient by carefully changing ϵ that controls the trade-off between exploration vs. exploitation. Some works suggest using goal conditioned value functions in temporal difference methods to achieve faster convergence than model free methods, and better asymptotic performance than model based methods [6].

4.4 Modelling Assumptions in Cognitive Radio Networks

In this section, we discuss how we can move from model free methods to model based methods in cognitive radio networks when some of the dynamics of the environment are known or can be assumed under some conditions.

4.4.1 Exploiting the Partial Model Information

In cognitive radio networks, the licensed users of a channel are called primary users (PUs) and the unlicensed users are called secondary users (SUs). SUs can access the channel to transmit their data whenever the PUs are idle but have to handover the channel as soon as a PU starts its transmission. The transmission patterns of PUs can be learnt over time and we can exploit this information to formulate a partial model of the environment. Let $C_t \in \mathcal{C} := \{0, 1\}$ denote the channel occupancy status. Let status go from 1 to 0 with $1 - q_0$ and 1 to 1 with q_0. Similarly, let status change from 0 to 1 with $1 - q_i$ and 0 to 0 with q_i. Then, $P(S_{t+1}|S_t, a_t) = P(C_{t+1}|C_t)P(E_{t+1}|C_t, E_t, a_t)$, where E_t denotes the energy of the system. We can exploit this information to formulate the spectrum sensing problem as a Partially Observable Markov Decision Process (POMDP) [7].

In this Chapter, we discussed the modelling assumptions and what happens if they are violated. In the next and final Chapter of the report, we provide the conclusion and future work.

Chapter 5

Conclusion

In this report, we reviewed the sections relevant to temporal difference methods and Q-learning from Chapter 5 in [2] and analyzed the modeling assumptions along with their impact. Temporal difference methods provide an array of sampling and averaging algorithms between one step look ahead and full episode Monte Carlo. These methods assign weight to the actions further back in time and allow course correction mid journey.

The Q-learning algorithm is a dynamic programming based method that uses Bellman method to look at estimated returns from all trajectories in the next step and optimizes the actions at each step. All these tabular methods work well when we have finite state space but for very large continuous or infinite state space, very large computations and storage is required so function approximations are used for most practical reinforcement learning problems.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Athena Scientific, 1996.

[3] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine learning*, vol. 22, no. 1-3, pp. 123–158, 1996.

[4] F. S. Melo, "Convergence of Q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[5] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[6] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep RL for model-based control," *arXiv preprint arXiv:1802.09081*, 2018.

[7] S. Park and D. Hong, "Optimal spectrum access for energy harvesting cognitive radio networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 12, pp. 6166–6179, 2013.