Autonomous and Adaptive Systems

Temporal Difference Learning

Mirco Musolesi

mircomusolesi@acm.org

Temporal-Difference Learning

- Temporal-difference (TD) methods like Monte Carlo methods can learn directly from experience.
- Unlike Monte Carlo methods, TD method update estimates based in part on other learned estimates, without waiting for the final outcome (we say that they *bootstrap*).
- We will first consider the problem of prediction (TD prediction) first (i.e., we fix a policy π and we try to estimate the value v_π for that given policy).
- Then we will consider the problem of finding an optimal policy (TD control).

Review/Preliminaries

- We consider three key RL problems:
  - The *prediction problem*: the estimation of v_π and q_π for a fixed policy π.
  - The *policy improvement problem*: the estimation of v_π and q_π while trying at the same time to improve the policy π.
  - The *control problem*: the estimation of an optimal policy π_*.

TD Prediction

- Both TD and Monte Carlo methods for the prediction problem are based on experience.
- Roughly speaking, Monte Carlo methods wait until the return following the visit is known, then use that return as a target for $V(S(t))$.
- An every-visit Monte Carlo method suitable for non-stationary environment is: $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$.

where G_t is the actual return following time t and α is a constant size parameter. This is not based on the average values but on a weighted average (you can get the average if you consider instead $\frac{1}{n}$ as step-size parameter).

TD Prediction

- Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S(t))$, because only at that point it is possible to calculate $G(t)$.
- TD methods instead need to wait only until the next step.
- At time $t + 1$ they immediately make a useful update from a target using the observed reward R_{t+1} and the estimate $V(S_{t+1})$.

TD(0)

- The TD(0) method is based on the following update:

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

on transition to S_{t+1} and receiving R_{t+1}.

This method is also called 1-step TD.

- Essentially, the target for the Monte Carlo update is G_t, whereas the target for the TD update is $R_{t+1} + \gamma V(S_{t+1})$.

TD(0)

Input: the policy π to be evaluated

Algorithm parameter: step size $\alpha \in (0,1]$

Initialise $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:

  Initialise S

  Loop for each step of episode:

    $A \leftarrow$ action given by π for S

    Take action A, observe R, S'

    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

    $S \leftarrow S'$

  until S is terminal

TD(0)

- Note that the quantity in brackets in the TD(0) update is a sort of error, measuring the difference between the estimated value of S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$.

- The *TD error* is defined as:

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

- The TD error at each time is the error in the estimate *made at that time*.

- It is interesting to note that, since the TD error depends on the next state and next reward, it is not actually available until one time step later.

  - In other words, δ_t is the error in $V(S_t)$ available at a time $t + 1$.

Advantages of TD Prediction Methods

- Compared to Dynamic Programming methods, TD methods do not require a model of the environment, of its reward and next-state probability distributions.
- Compared to Monte Carlo methods, TD methods are implemented in an online, fully incremental fashion.
  - With Monte Carlo methods, one must wait until the end of an episode (i.e., when the return is known), instead with TD methods, we need to wait only one time step.
  - Why does this matter?
    - Some applications have very long episodes.
    - Some applications are actually continuing tasks.

Theoretical Basis of TD(0)

- Even if the learning process happens step-by-step, we have convergence guarantees supporting the methods presented in the lecture (see Sections 6.2 and 9.4 of Barto and Sutton 2018).
- More precisely, for any fixed policy π, TD(0) has been proved to converge to v_π, in the mean for a constant step-size parameter if it is sufficiently small, and with probability 1 if the step-size decreases given stochastic approximation conditions (see Section 2.7 of Barto and Sutton 2018).
- But what is the faster in terms of convergence between dynamic programming, Monte Carlo and TD?
  - It's still an open question, in case you are looking for a research topic!

On-Policy and Off-Policy Control

- We assume the following definitions:
  - ***On-policy control***: the agent learns and updates its policy **using** the rewards obtained by the actions selected by means of the policy itself.
  - ***Off-policy control*:** the agent learns and updates its policy **not using** the rewards obtained by the actions selected by means of the policy itself.

Sarsa: On-Policy TD Control

- We now consider the use of TD prediction methods for the control problem.
- For an on-policy method, we must estimate $q_\pi(s, a)$ for the current behaviour policy π and for all the states s and actions a.
- Above, we considered the transitions from state to state and we learned the values of states.
- Now we consider the transitions from state-action pair to state-action pair and learn the values of state-action pair.

Sarsa: On-Policy TD Control

- Formally, these cases are identical: they are both Markov chain with a reward process. The theorems assuring the converge of state values under TD(0) also apply to the corresponding algorithm for action values:
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
- This update is done after every transition from a non-terminal state S_t.
- If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1}) \leftarrow 0$.

Sarsa: Online TD(0) Control

- The update of Sarsa uses all the elements of the quintuple: $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
  - This indicates a transition from a state to the next.
  - This quintuple gives rise to the name *Sarsa*.
- We can design an on-policy control algorithm based on the Sarsa prediction method.
  - As in all on-policy methods, we continually estimate q_π for the behaviour policy π.
  - At the same time, we assume a greedy policy using Q values.
    - By doing so we will have a convergence of the Q values to q_*.

SARSA

Algorithm parameters: step size $\alpha \in (0,1], \epsilon > 0$

Initialise Q(s,a), for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$ arbitrarily (except end states)

Loop for each episode:

Initialise S

Choose A from $\mathcal{A}(s)$ using policy derived from Q (e.g., ϵ-greedy)

Loop for each step of episode:

Take action A, observe R, S'

Choose A' from $\mathcal{A}(s')$ using policy derived from Q (e.g., ϵ-greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

$$S \leftarrow S'$$

$$A \leftarrow A'$$

until S is terminal

Q-learning: Off-policy TD Control

- Q-learning is one of the classic RL algorithms.
- Q-learning is an off-policy TD control algorithm, defined by:

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

- In this case, the learned action-value function Q directly approximates q_*, the optimal action-value function, independent of the policy being followed.
  - Policy still matters since it determines which state-action pairs are visited/updated. However, only requirement for convergence is that all pairs continue to be updated.
  - Early convergence proofs.

Learning from Delayed Rewards

Christopher John Cornish Hellaby Watkins

King's College

Thesis Submitted for Ph.D.

May, 1989

Machine Learning, 8, 279–292 (1992)
© 1992 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Technical Note
Q-Learning

CHRISTOPHER J.C.H. WATKINS
25b Framfield Road, Highbury, London N5 1UU, England

PETER DAYAN
Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9EH, Scotland

Abstract. Q-learning (Watkins, 1989) is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

This paper presents and proves in detail a convergence theorem for Q-learning based on that outlined in Watkins (1989). We show that Q-learning converges to the optimum action-values with probability 1 so long as all actions are repeatedly sampled in all states and the action-values are represented discretely. We also sketch extensions to the cases of non-discounted, but absorbing, Markov environments, and where many Q values can be changed each iteration, rather than just one.

Q-Learning

Algorithm parameters: step size $\alpha \in (0,1), \epsilon > 0$

Initialise $Q(s, a)$ for all $s \in \mathcal{S}^+$ arbitrarily, $a \in \mathcal{A}(s)$ except for terminal states that is set to 0.

Loop for each episode:

Initialise S

Loop for each step of episode:

Choose A from S using policy derived from Q (e.g., ϵ-greedy)

Take action A, observe R, S'

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

$$S \leftarrow S'$$

until S is terminal

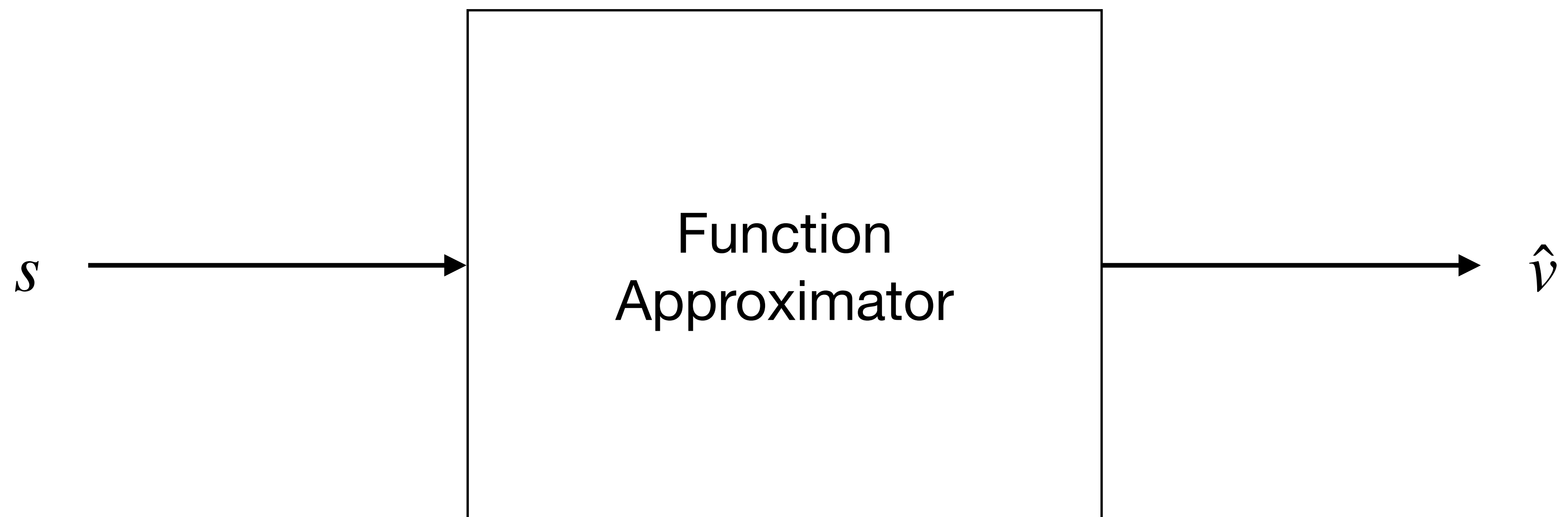
Summary

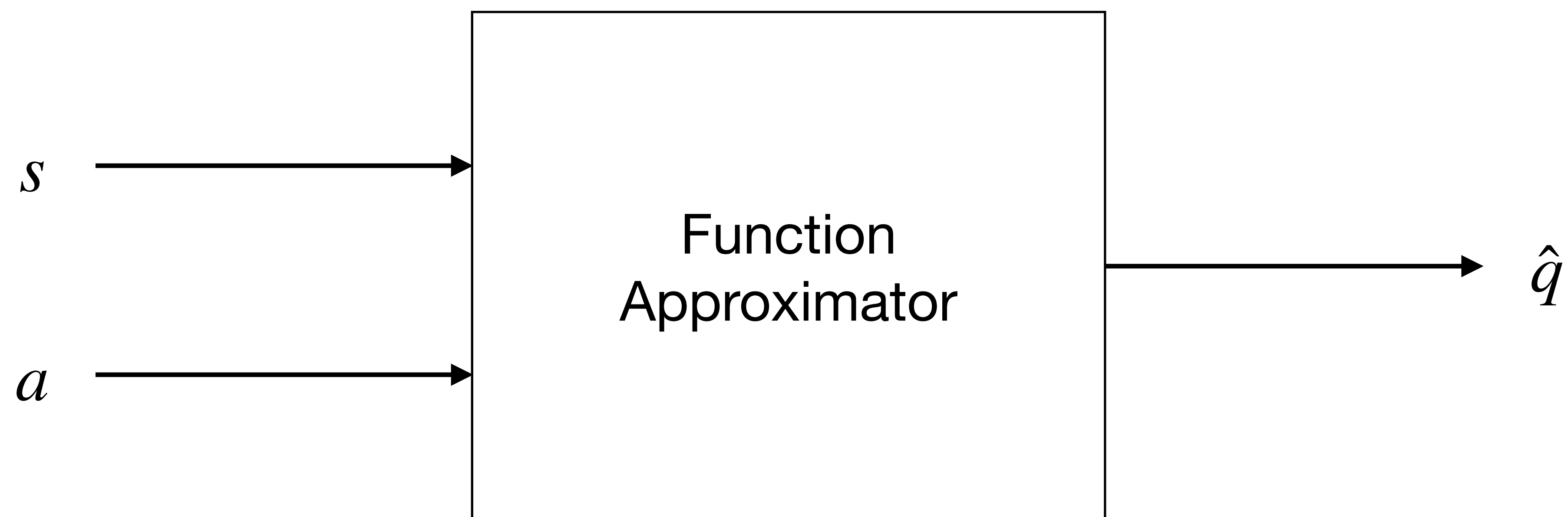
- The methods introduced in the lectures are among the most-used methods in RL.
- These methods are usually referred to as tabular methods, since the state-action space can fit in a table.
  - Table with 1 row per state-action entry.
  - What happens if you can’t fit all the state-action entry in a table?
    - We need *function approximation* rather than tables.

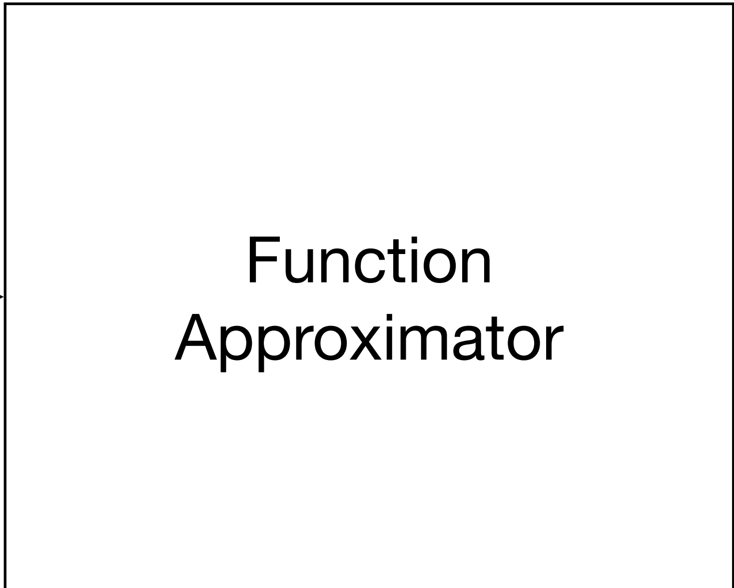
Summary

- Function Approximation will provide a mapping between a state or state-action to a value function.

- More precisely, a value-function approximation is a function with in input the state (or the state and action), which gives in output the value function for the state (or the state and action).


s
Function
Approximator
$\hat{v}$


s
a
Function
Approximator
$\hat{q}$



$\hat{q}$

References

- Chapter 6 of Barto and Sutton. Introduction to Reinforcement Learning. Second Edition. MIT Press. 2018.