Temporal-Difference Learning

Machine Learning

Daniele Loiacono

POLITECNICO MILANO 1863

Outline and References

- Outline
  - TD(0)
  - SARSA
  - Q-Learning

- References
  - Reinforcement Learning: An Introduction [RL Chapter 6 and 7]
  - Sample-based Learning Methods (Coursera)

Why temporal-difference?

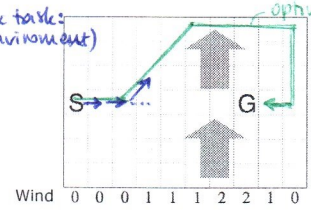
- DP requires the knowlege of the MDP dynamics
- MC learns from experience but requires complete episodes to perform the updates
  - It can be applied only to episodic task
  - Even in episodic task it might have problems:

Dynamic Programming was not enough because it required us to know the one step dynamics (and we tipically don't have this known).

Monte Carlo can learn from experience but there are some issues/limitations: it can be applied only to episodic tasks ⇒ it needs episodes to learn. The point is that MC needs the whole episode to START learning.

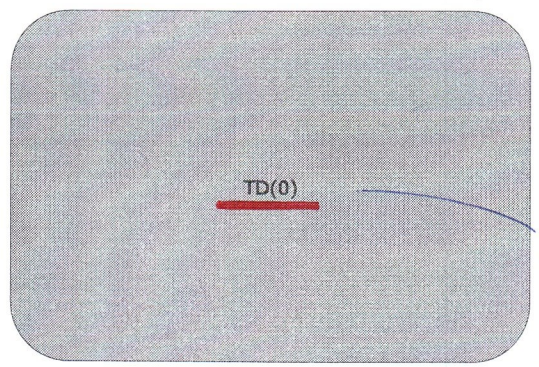
Episodic task: (windy environment)

optimal



We start from the start S and we have to reach the goal G. There is some wind that changes the transition to the next state.

The green policy is extremely difficult to find with an explorative policy, for example. If we start with the uniform random actions (↑ 25%, ↓ 25%, → 25%, ← 25%) we cannot really reach the goal by chance easily. Given that, in order to START Montecarlo, we need to reach the goal (so that the episode ends, we compute the returns and we start updating), this becomes a big issue.



It combines together Monte Carlo AND Dynamic Programming (it combines: SAMPLING (from MC): we don't need the knowledge of the dynamics, we need experience. and BOOTSTRAPPING: we exploit the recursive structures of the problem to learn/update the value of the value function in a state S based on the values of the value function in the states that are next to S)

Temporal-Difference Policy Evaluation – TD(0)

- Temporal-Difference combines MC (**model-free**) with DP (**bootstrapping**):

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]$$

we start from this: Monte Carlo policy evaluation (version which uses rolling means instead of incremental mean)

to update this we need the return G_t (which means that the episode must be completed)

Temporal-Difference Policy Evaluation – TD(0)

- Temporal-Difference combines MC (**model-free**) with DP (**bootstrapping**):

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

$$G_t \approx R_{t+1} + \gamma V(S_{t+1})$$

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

Temporal-Difference Error (δ_t)

we don't want the episode to be concluded to start doing something → we exploit the recursive structure of the return (we should use G_{t+1}, instead we use $V(S_{t+1})$ and we put "$\approx$")

The fact that G disappeared makes it possible to apply this update rule even when the episode is not terminated → we can start learning after each step

TD(0) Policy Evaluation

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

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

S,A R,S,A,S,A,R,S,A,S,A,R,...

TD(0) Policy Evaluation

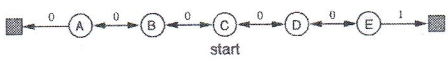
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

update
S,A,R,S,A,S,A,R,S,A,S,A,R,...

the update is still delayed in time, but now is not delayed at the end of the episode but just of 1 step (HUGE advantage)

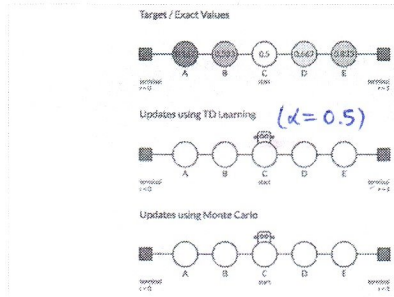
Random Walk Example

Comparison between MC and TD(0)



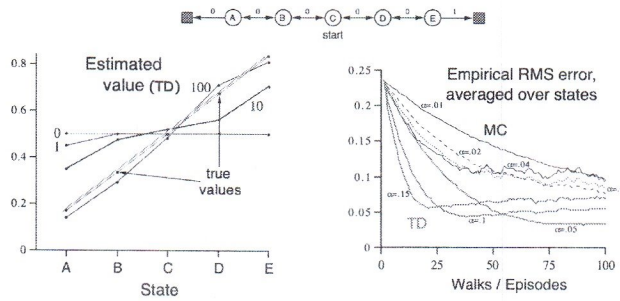
- Let C be the starting state
- Let π be left or right with equal probability in all states (random policy)
- Reward +1 on **right** termination, 0 otherwise
- Assuming $\gamma = 1$, $V_\pi(s)$ represents the probability of ending on the right side

Random Walk Example - Simulation



Shown in animation is that in both cases we converge to the target, however the two methods behave very differently when updating. Tipically, TD is much faster than MC.

Random Walk Example – Value Function



MC vs TD

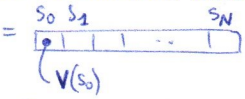
- TD can learn **before** knowing the final outcome
  - TD can learn **after every step**
  - MC must wait **until end of episode** before return is known
- TD can learn **without** the final outcome
  - TD can learn from **incomplete sequences**
  - MC can only learn form **complete sequences**
- TD works in continuing (non–terminating) environments MC only works for episodic (terminating) environments

MC vs TD: Bias-Variance

- MC target has lower bias
  - MC return $R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_{t+T}$ is an **unbiased** estimate of $V_\pi(S_t)$
  - TD target $R_{t+1} + \gamma V(S_{t+1})$ is a **biased** estimate of $V_\pi(S_t)$, as $V(S_{t+1}) \neq V_\pi(S_{t+1})$
- TD target has lower variance
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward
- Overall
  - MC works well with function approximation and it is not very sensitive to initial values
  - TD has problem with function approximation and it is more sensitive to initial values

In MC we have multiple sources of stochasticity (since we consider $R_{t+1}, R_{t+2}, \cdots, R_{t+T}$), instead, in TD we only have 2 sources of stochasticity (R_{t+1} and S_{t+1}).

We're used to represent the value function as a table:

$V =$ 

This is not feasible in a huge problem → in real problem we replace the tabular representation with a parametrized function (that can be anything: SVM, regression, ...)

Sampling and Bootstrapping: overview

- **Bootstrapping**: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- **Sampling**: update does not involve an expected value
  - MC samples
  - DP does not sample
  - TD samples

SARSA

TD(0) solves the evaluation policy problem. Can TD(0) be used also to learn a policy?

Let's go back to MC Policy Iteration

- Evaluation

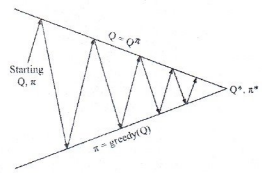
$$Q_\pi(s,a) \approx average[G_t | S_t = s, A_t = a]$$

What we do in SARSA is to replace the return with an approx.:

$$G_t \approx R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

- Improvement

$$\pi'(s) = \arg\max_a Q_\pi(s,a)$$

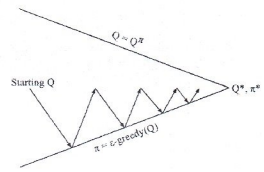


On-Policy Control with SARSA

because we're learning an action value function by following a policy: an the action value function we're learning is the one of the policy we're following

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

- Evaluation
  - SARSA
- Improvement
  - ε-Greedy Policy Improvement



On-Policy Control with SARSA

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

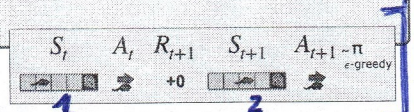
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

On-Policy Control with SARSA

* From here we get the name:
S_t = current state
A_t = current action
R_t = current reward (immediate reward)
S_{t+1} = next state
A_{t+1} = next action

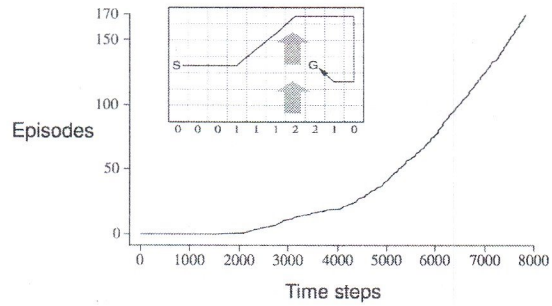
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from 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



the mouse want to reach the cheese, he goes right, he doesn't get the reward, he goes again right (accordingly with the policy that he follows, that is an ε-greedy policy based on the current knowledge of the problem) and then we use the new information to update the previous pair.
The information of 2 update the information of 1. *

Example: SARSA on Windy Gridworld ✓



Q-Learning

It modifies a little bit SARSA by modifying the on-policy into a OFF-POLICY

Q-Learning: Off-Policy TD Control

- SARSA, as Policy Iteration in DP, is based on Bellman Expectation Equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

$$Q_\pi(s,a) = \sum_{s',r} p(s', r|s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')\right)$$

We can see a parallel between the SARSA structure of the update rule and the update rule used in policy iteration in dynamic programming

- Q-Learning, as Value Iteration in DP, is based on Bellman Optimality Equation

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

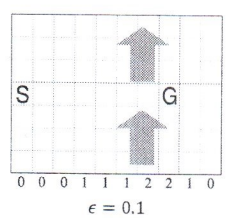
$$Q^*(s,a) = \sum_{s',r} p(s', r|s, a)\left(r + \gamma \max_{a'} Q^*(s', a')\right)$$

Similarly as above, the Q-learning takes ispiration from the value iteration in Dynamic Programming (which exploit the Bellman Optimality Equation instead of the Bellman Expectation Equation)

Q-Learning: Off-Policy TD Control

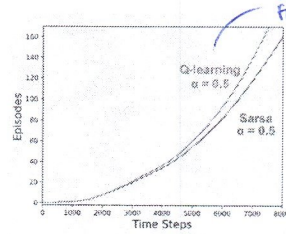
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  until S is terminal

Q-Learning vs SARSA: Windy Gridworld

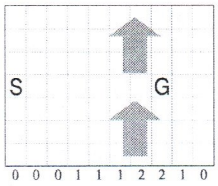


$\epsilon = 0.1$

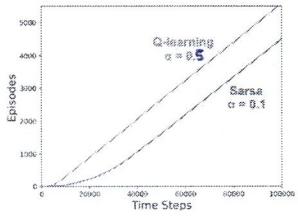
Q-learning learns slightly faster than SARSA



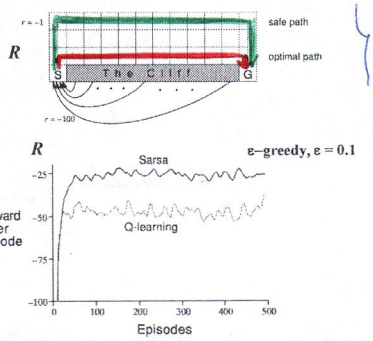
Q-Learning vs SARSA: Windy Gridworld



$\epsilon = 0.1$



Q-Learning vs SARSA: Cliff Walking Environment



We have a grid: we can move everywhere, we get -1 as reward everywhere, we have to reach a goal starting from state S, we have special states ("the cliff") and when we move into these states we have a deterministic transition back to the start and a reward that is -100.
The optimal path is the red (and Q-lear. is able to learn it). SARSA is not able to learn that path, it's going to learn the safest one (the green).
Why? Because SARSA goes with an ε-greedy policy → it won't choose a path which has a probability (small, but) > 0 to end up in the cliff. Q-learning doesn't have this problem because in the maximisation step it doesn't take into account the worst possible scenario (only the optimal one).
→ Depending on the problem settings we are into, we may want to prefer one or the other. (depending on how much we care about the safety w.r.t. the optimality)