ATCI: Reinforcement Learning

Free model Algorithms

Mario Martin

CS-UPC

February 20, 2025

Recap

- Definition of RL
- Framework
- Concepts learned:
  - Model
  - Policy: deterministic and non-deterministic
  - Reward functions, immediate reward
  - Discounted and undiscounted Long-term reward
  - ... γ, π, markovian condition
- Value functions
- Bellman equation
- Policy evaluation: Value iteration and algebraic method
- Optimal policy, greedy policy and relation with Value functions
- Dynamic programming methods: Value iteration, Policy iteration, asynchronous methods

Goal of this lecture

- Problems with Dynamic Programming methods
  - Sweep of full steps or random steps
  - **Need to know the model**
- We'll see now methods that do not require a model but only *experiences* to build evaluations of policies and also to find optimal policies
- Methods we'll see:
  - Monte-Carlo
  - Q-learning
  - Temporal differences; n-steps and TD(λ)
  - Sarsa, Expected Sarsa
- Off-line vs. on-line learning
- Importance Sampling

Monte-Carlo methods

[About expectations]

- Refresher about expectations:

$$\mathbb{E}\left[f(x)\right] = \sum_{x \in Val(X)} f(x)p(x)$$

[About expectations]

- Refresher about expectations:

$$\mathbb{E}\left[f(x)\right] = \sum_{x \in Val(X)} f(x)p(x)$$

$$\mathbb{E}_{x \sim p}[f(x)] = \sum_{x \in Val(X)} f(x)p(x)$$

- For continuous variables

$$E[f(X)] = \int_{-\infty}^{\infty} f(x)p(x)dx$$

[About expectations]

- Refresher about expectations:

$$\mathbb{E}\left[f(x)\right] = \sum_{x \in Val(X)} f(x)p(x)$$

$$\mathbb{E}_{x\sim p}[f(x)] = \sum_{x \in Val(X)} f(x)p(x)$$

- For continuous variables

$$E[f(X)] = \int_{-\infty}^{\infty} f(x)p(x)dx$$

- Expectation computation by sampling

$$\mathbb{E}_{x\sim p}[f(x)] \approx \frac{1}{T}\sum_{t=1}^{T} f(x^t)$$

[About expectations]

- Expectation computation by sampling (Monte Carlo)

$$\mathbb{E}_{x \sim p}[f(x)] \approx \frac{1}{T} \sum_{t=1}^{T} f(x^t)$$

Monte-Carlo Policy Evaluation

- Goal: learn V^π from episodes of experience under policy π

$$S_1, A_1, r_2, S_2, A_2, r_3, \ldots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$R_t = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-1} r_T$$

- Recall that the value function is the expected return:

$$V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] = \sum_\tau R_\tau p^\pi(\tau) \approx \frac{1}{N}\sum_{i=1}^{N} R_i$$

where R_i is obtained from state s under π distribution (following π)

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Monte-Carlo reinforcement learning

- MC uses the simplest possible idea: value = mean return. Instead of computing expectations, sample the long term return under the policy
- MC methods learn directly from episodes of experience
- MC is *model-free*: no explicit knowledge of environment mechanisms
- MC learns from complete episodes
  - Caveat: can only be applied to complete *episodic* environments (all episodes must terminate).

Monte-Carlo *policy evaluation*

Monte Carlo *policy evaluation*

Given π, the policy to be evaluated, initialize V randomly
$Returns(s) \longleftarrow$ empty list, $\forall \ \in S$
repeat
  Generate trial using π
  for each s in trial **do**
    $R \leftarrow$ long-term-return following s
    Append R to $Returns(s)$
    $V(s) \leftarrow average(Returns(s))$
  end for
until false

Monte-Carlo Policy Evaluation

- How to average results for $V(s)$? Every time-step t that state s is visited in an episode:
  - Increment counter $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + R_t$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow V^{\pi}(s)$ as $N(s) \rightarrow \infty$ for all states
- However, for each state you should store S and N.

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally:

$$V_n(S_t) = \frac{1}{n}\sum_{i=1}^{n} R_i$$

$$V_n(S_t) = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right)$$

$$V_n(S_t) = \frac{1}{n}\left(R_n + (n-1)V_{n-1}(S_t)\right)$$

$$V_n(S_t) = \frac{1}{n}R_n + \frac{1}{n}\left((n-1)V_{n-1}(S_t)\right)$$

$$V_n(S_t) = \frac{1}{n}R_n + V_{n-1}(S_t) - \frac{1}{n}V_{n-1}(S_t)$$

$$V_n(S_t) = V_{n-1}(S_t) + \frac{1}{n}(R_n - V_{n-1}(S_t))$$

Incremental Monte-Carlo Updates

- Compute return R_t
- For each state S_t with return R_t

$$N(S_t) \leftarrow N(S_t) + 1$$
$$V(S_t) \leftarrow V(S_t) + \alpha(S_t)(R_t - V(S_t))$$

- where

$$\alpha(S_t) = \frac{1}{N(S_t)}$$

- Still we have to store the number of visits to each state: $N(S_t)$. Usually a **constant parameter** α in (0..1) is used:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_t - V(S_t))$$

[Incremental Monte-Carlo Updates]

- ... with side effect of forgetting old episodes: as higher the value, higher the influence of recent experiences in the estimations
- Notice that:

$$V_n(S) = V_{n-1}(S) + \alpha(R_n - V_{n-1}(S)) = \alpha R_n + (1-\alpha)V_{n-1}(S)$$

- So,

$$V_n(S) = \alpha R_n + (1-\alpha)V_{n-1}(S)$$
$$V_n(S) = \alpha R_n + (1-\alpha)\left(\alpha R_{n-1} + (1-\alpha)V_{n-2}(S)\right)$$
$$V_n(S) = \alpha R_n + (1-\alpha)\left(\alpha R_{n-1} + (1-\alpha)\left(\alpha R_{n-2} + (1-\alpha)V_{n-2}(S)\right)\right)$$
$$V_n(S) = \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \dots$$
$$V_n(S) = \alpha \sum_{i=0}^{n-1} \left[(1-\alpha)^i R_{n-i}\right] + (1-\alpha)^n R_0$$

[Incremental Monte-Carlo Updates]

- Trick used not only in Monte Carlo but on all methods
- Choose α carefully. Remember

$$V_n(S) = \alpha \sum_{i=0}^{n-1} \left[(1-\alpha)^i R_{n-i} \right] + (1-\alpha)^n R_0$$

- Usually α is low (0.01..0.2), but depends on the problem. Sometimes is good to forget old Long-term-Returns, for instance, *when you change the policy*!
- It is also usual to use a common α for all states that decrease with experiences (we'll see that on notebook).

Monte-Carlo *policy learning*

- Can we use the MC policy evaluation to learn a policy (like with PI)?

[from previous lecture: Policy iteration]

Policy Iteration (PI)

Initialize $\pi, \forall s \in S$ to a random action $a \in \mathcal{A}(s)$, arbitrarily
repeat
  $\pi' \leftarrow \pi$
  Compute V^{π} for all states using a *policy evaluation* method
  for each state s **do**

$$\pi(s) \leftarrow \arg\max_{a \in A} \sum_{s'} P^{a}_{ss'} \left[R(s') + \gamma V^{\pi}(s') \right]$$

  end for
until $\pi(s) = \pi'(s) \ \forall s$

Monte-Carlo *policy learning*

- Can we use the MC policy evaluation to learn a policy (like with PI)?
- If we want to take the *greedy action*, like in PI, to improve the policy then **we need the model!**

$$\pi(s) = \arg\max_{a \in A} \sum_{s'} P^a_{ss'} \left[R(s') + \gamma V^\pi(s') \right]$$

Monte-Carlo *policy learning*

- Can we use the MC policy evaluation to learn a policy (like with PI)?
- If we want to take the *greedy action*, like in PI, to improve the policy then **we need the model!**

$$\pi(s) = \arg\max_{a \in A} \sum_{s'} P^a_{ss'} \left[R(s') + \gamma V^\pi(s') \right]$$

- **Solution**: estimate Q^π function instead of V^π
- Now we can *greedify* the policy without the model:

$$\pi(s) = \arg\max_{a \in A} Q^\pi(s, a)$$

Monte-Carlo *policy learning*

- Another change wrt Policy iteration: We don't sweep the whole set of states to update the Value estimates, neither the policy
- We use a asynchronous version of PI where only evaluations of some states are updated
- States updated are from the experience collected by the agent in one learning episode
- ...But then we cannot go out of the learning loop never

Monte-Carlo *policy learning*

- Apply the *improvement-of-the-policy* idea to learn the optimal policy.

Caution! Monte Carlo *policy learning* with a subtle error

Initialize π and Q randomly:
repeat
  Generate trial using π
  for each s, a in trial **do**
    $R \leftarrow$ long-term-return following s, a
    $Q(s,a) \leftarrow Q(s,a) + \alpha(R - Q(s,a))$
  end for
  for each s in trial **do**
    $\pi(s) = \arg\max_{a \in A} Q(s,a)$
  end for
until false

Monte-Carlo *policy learning*

- What's wrong?
  - Algorithm tries to implement asynchronous version of policy iteration... but remember... there states are selected for updating **randomly**.
  - Now states to be updated depend on the current policy, so we cannot guarantee convergence.
- New important concept: **Exploration vs. Exploitation**
  - All pairs (s,a) should have probability non-zero to be updated.
  - At same time, we want to evaluate the current policy
- Several ways to balance two concepts.

ϵ-greedy exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability 1 - ϵ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s, a') \\ \epsilon/m, & \text{otherwise} \end{cases}$$

where $m = |\mathcal{A}(s)|$

Monte-Carlo *policy learning*

- Apply the *improvement-of-the-policy* idea to learn the optimal policy.

Monte Carlo *policy learning*

Initialize π and Q randomly:
repeat
  Generate trial using $\epsilon-$**greedy strategy on** π
  for each s, a in trial **do**
    $R \leftarrow$ long-term-return following s, a
    $Q(s, a) \leftarrow Q(s, a) + \alpha(R - Q(s, a))$
  end for
  for each s in trial **do**
    $\pi(s) = \arg\max_{a \in A} Q(s, a)$ **// ties randomly broken**
  end for
until false

Monte-Carlo *policy learning*

Monte Carlo *policy learning*

Initialize π and Q randomly:
repeat
  Generate trial using exploration method based on π
  for each s, a in trial **do**
    $R \leftarrow$ long-term-return following s, a
    $Q(s,a) \leftarrow Q(s,a) + \alpha(R - Q(s,a))$
  end for
  for each s in trial **do**
    $\pi(s) = \arg\max_{a \in A} Q(s,a)$
  end for
until false

Monte-Carlo *policy learning*

Monte Carlo *policy learning*

Initialize π and Q randomly:
repeat
  Generate trial using exploration method on greedy policy derived from Q values
  for each s, a in trial **do**
    $R \leftarrow$ return following s, a
    $Q(s, a) \leftarrow Q(s, a) + \alpha(R - Q(s, a))$
  end for
until false

Notes about Exploration

About exploration

- Ideally, exploration should not be constant during training.
- It should be larger at the beginning and lower after a lot of experience is accumulated (*why?*)...

About exploration

- Ideally, exploration should not be constant during training.
- It should be larger at the beginning and lower after a lot of experience is accumulated (*why?*)... but never disappear (*why?*)

About exploration

- Ideally, exploration should not be constant during training.
- It should be larger at the beginning and lower after a lot of experience is accumulated (*why?*)... but never disappear (*why?*)
- This requirement is asked in convergence proofs of most RL algorithms, f.i. in MC.
- In ϵ-greedy, this is implemented with variable ϵ starting from 1 and decreasing with number of experiences until a minimum ϵ value from which does not decrease further, f.i:

$$\epsilon = \max(1/(\alpha T), 0.1)$$

where T is the number of Trials done and α is a constant that controls decrease of exploration

About exploration

- Another popular way to explore is using **Softmax exploration** or **Gibb's exploration** or **Boltzman exploration**.
- Idea is that probability depends on the value of actions, with bias of exploration towards more promising actions
- Softmax action selection methods grade action probabilities by estimated values

$$P(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}$$

  where parameter τ is called *temperature* and decreases with experience
- When τ is very large, all actions with roughly same probability of being selected. When τ is low, almost certainty of selecting the action with higher Q-value.

About exploration

- A hot topic of research
- We want to explore *efficiently* the state space
- A lot of other more complex mechanisms based on criteria
  - Less explored state, action pairs
  - Higher changes in value of state action pair
  - Bases on recency of last exploration
  - Uncertainty on estimation of values
  - Error in an agent's ability to predict the consequence of action (curiosity)
  - ...

Temporal Differences methods: Q-learning

Temporal Differences *policy evaluation*

- Monte-Carlo methods compute expectation of Long-term-Reward averaging the return of several trials.
- Average is done after termination of the trial.
- We saw in previous lecture that Bellman equation also allow to estimate expectation of Long-term-Reward

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma Q^\pi(S_{t+1}, \pi(S_{t+1})) | S_t = s, A_t = a] \end{aligned}$$

- Computing expectations with world model:

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} \left(r(s') + \gamma \, Q^\pi(s', \pi(s')) \right)$$

Temporal Differences *policy evaluation*

- How to get rid of the world-model?
- Q-value function and averaging, like in the case of MC

$$Q(S_t, a) \leftarrow Q(S_t, a) + \alpha(R_t(s_t) - Q(S_t, a))$$

- But now substitute R_t with Bellman equation:

$$Q(S_t, a) \leftarrow Q(S_t, a) + \alpha \left[r_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1})) - Q(S_t, a) \right]$$

or

$$Q(S_t, a) \leftarrow (1 - \alpha) Q(S_t, a) + \alpha \left[r_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1})) \right]$$

- This is called *bootstrapping*

Temporal Differences *policy evaluation*

Temporal Differences *policy evaluation*

Given π initialize Q randomly:

repeat

  $s \leftarrow$ initial state of episode

  repeat

    $a \leftarrow \pi(s)$

    Take action a and observe s' and r

    $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s', \pi(s')) - Q(s,a)\right)$

    $s \leftarrow s'$

  until s is terminal

until convergence

MC and TD comparison

- Goal: learn Q^π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $Q(s,a)$ toward actual return R_t

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(R_t - Q(s_t,a_t))$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $Q(s_t,a_t)$ toward estimated return $r_{t+1} + \gamma V(s_{t+1})$

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - Q(s_t,a_t))$$

  - Remember that $Q(s_t,\pi(s_t)) = V(s_t)$
  - $r_{t+1} + \gamma V(s_{t+1})$ is called the TD target
  - $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - Q(s_t,a_t)$ is called the TD error

Advantages and disadvantages of MC vs. TD

- TD can learn *before* knowing the final outcome
  - TD can learn online 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 from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/variance trade-off

- Return $R_t = r_{t+1} + r_{t+2} + \ldots + \gamma^{T-1} r_T$ is *unbiased* estimate of $V^\pi(S_t)$
- True TD target $r_{t+1} + V^\pi(s_{t+1})$ is unbiased estimate of $V^\pi(s_t)$ but, while learning, TD target $r_{t+1} + V(s_{t+1})$ is a *biased* estimate of $V^\pi(s_t)$
- TD target is much lower variance than the return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one random action, transition, reward

Temporal Differences *policy evaluation*

- On-line learning: evaluation is embedded in generation of the experience.
- It can be applied to non-episodic tasks
- You don't need to end episode to learn
- Like in MC you don't need the World model
- In practice faster: Takes profit on Markovian property

MC vs TD learning

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Can we use it for policy learning?
- Natural idea: use TD instead of MC in our learning loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Temporal Differences *policy learning*

Q-learning: Temporal Differences policy learning

Given π initialize Q randomly:
repeat
  $s \leftarrow$ initial state of episode
  repeat
    Set a using f.i. $\epsilon-$greedy strategy on π
    Take action a and observe s' and r
    $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s', \pi(s')) - Q(s,a))$
    $\pi(s) = \arg\max_{a \in A} Q(s,a)$ // ties randomly broken
    $s \leftarrow s'$
  until s is terminal
until false

Temporal Differences *policy learning*

Q-learning: Temporal Differences policy learning

Given π initialize Q randomly:
repeat
  $s \leftarrow$ initial state of episode
  repeat
    Set a using f.i. $\epsilon-$greedy strategy based on Q values
    Take action a and observe s' and r
    $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
    $s \leftarrow s'$
  until s is terminal
until false

Temporal Differences extended

Temporal Differences extended

- Bootstrapping in Bellman equation is done from next state:

$$\begin{aligned} V^{\pi}_{(1)}(s) &= \mathbb{E}_{\pi}[R_t | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s] \end{aligned}$$

- But we can obtain estimation from 2 steps in the future also:

$$\begin{aligned} V^{\pi}_{(2)}(s) &= \mathbb{E}_{\pi}[R_t | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 (r_{t+3} + \ldots)) | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 R_{t+2} | S_t = s] \\ &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 V^{\pi}(S_{t+2}) | S_t = s] \end{aligned}$$

Temporal Differences extended

- In general we could extend that to the *n-steps estimator of long-term reward*.

$$
\begin{aligned}
V^{\pi}_{(n)}(s) &= \mathbb{E}_{\pi}[R_t | S_t = s] \\
&= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_n + + \gamma^n r_{n+1} \ldots | S_t = s] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^n V^{\pi}(S_{t+n}) | S_t = s\right]
\end{aligned}
$$

Temporal Differences extended: n-step estimators

- All estimators of expectation are valid, but different bias and variance.
- Which one to use?
- Any of them is Ok at the end, but different learning speed with different value of n.
- Implementation of the algorithm is easy. For each episode
  1. Execute n actions, keep rewards
  2. Apply update
     $V^{\pi}(S_t) = \alpha V^{\pi}(S_t) + (1-\alpha)\sum_{k=0}^{n} \gamma^k r_{t+k+1} + \gamma^n V^{\pi}(S_{t+n})$

Temporal Differences n-steps *policy evaluation*

All store and access operations (for S_t and R_t) can take their index mod n

Temporal Differences n-steps policy evaluation

Given π and n, initialize Q randomly:
for each episode **do**
  $s \leftarrow$ initial state of episode, and $T \leftarrow \infty$
  for $t = 0, 1, 2..$ **do**
    if $t < T$ **then**
      Take action $a \leftarrow \pi(s)$ and observe and store s' and r
      If s' is terminal $T \leftarrow t + 1$
    end if
    $\tau \leftarrow t - n + 1$
    if $\tau \geq 0$ **then**
      $R \leftarrow \sum_{i=\tau+1}^{min(\tau+n,T)} \gamma^{i-\tau-1} r_i$
      If $\tau + n < T$ then: $R \leftarrow R + \gamma^n V(s_{\tau+n})$
      $Q(s, a) \leftarrow Q(s, a) + \alpha (R - Q(s, a))$
    end if
    $s \leftarrow s'$
  end for
end for

Temporal Differences n-steps *policy evaluation*

- Advantages:
  - It generalizes Temporal-Differences and Monte Carlo methods:
    - when $n = 1$, it is equivalent to Q-learning
    - when $n = \infty$ (or H) we have MC algorithm
  - an intermediate n often learn Q-values faster
  - applicable to both continuing and episodic problems
  - per-step computation is small and uniform, like TD
- There are some disadvantages:
  - need to choose n value
  - need to remember the last n states
  - learning is delayed by n steps

Temporal Differences extended TD(λ)

- Another option. Instead of using one estimator, update using an **average** of them
- For practical purposes, use a geometric average ($0 \leq \lambda \leq 1$)

$$V_\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} V_{(n)}$$

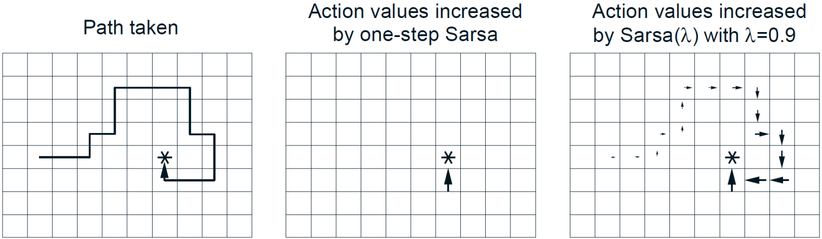
- Can be rewritten for episodes as:

$$V_\lambda(S_t) = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} V_{(n)}(S_t) + \lambda^{T-t-1} R_t$$

- Unifies different algorithms:
  - When $\lambda = 0$ we have TD(0), the standard Q-learning method
  - When $\lambda = 1$ we have the standard MC method
- In general for other values of λ we use a smart incremental implementation using **eligibility traces** (chapter 12, Sutton book)

Temporal Differences intuition

- Benefits of temporal differences using larger n-step than TD(0)



- In general, faster propagation of rewards and, so, faster learning.

Temporal Differences extended: conclusions

- Very good to **estimate values for a given policy**
- More difficult to apply to control, because trace of experience describe one policy but we are *estimating another one*

Temporal Differences extended: conclusions

- Very good to **estimate values for a given policy**
- More difficult to apply to control, because trace of experience describe one policy but we are *estimating another one*
- What?

Temporal Differences extended: conclusions

- Very good to **estimate values for a given policy**
- More difficult to apply to control, because trace of experience describe one policy but we are *estimating another one*
- What?
- You will understand in next slides.
- Let's go now to discuss the concepts of on-policy and off-policy learning.

On-policy vs. Off-policy learning: Sarsa algorithm

Off-policy vs. On-policy learning

- When learning value functions of a policy, we sample *using the policy* to estimate them
- In Q-learning, the method tries to learn the value function of the optimal policy (V^*) when in fact samples are obtained from different policy (ϵ-greedy policy)
- A subtle point with implications about the convergence of the algorithms to the optimal solution
- We'll do the following distinction:

  On-policy learning: When learning the value function V^π of the current policy π

  Off-policy learning: When Learning the value function V^π using another policy π'

Off-policy vs. On-policy learning

- In this sense, Q-learning is an example of off-policy learning.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q(s', \pi(s')) - Q(s,a)\right)$$

- Policy for which we learn values:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q^*(s', a') - Q(s,a)\right)$$

- But we use another policy (ϵ-greedy policy):

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s,a') \\ \epsilon/m, & \text{otherwise} \end{cases}$$

where $m = |\mathcal{A}(s)|$

Off-policy vs. On-policy learning

- Trivia. What about MC learning? Is it on-policy or off-policy learning?

Updating action-value functions with SARSA

- Let's try to implement an on-policy learning version of Q-learning
- **Sarsa: on-policy TD(0) learning**
- In Q-learning, update equation was:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \overbrace{\max_{a'} Q(s',a')}^{\text{Could be not the action executed}} - Q(s,a))$$

Updating action-value functions with SARSA

- Let's try to implement an on-policy learning version of Q-learning
- **Sarsa: on-policy TD(0) learning**
- In Q-learning, update equation was:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \overbrace{\max_{a'} Q(s',a')}^{\text{Could be not the action executed}} - Q(s,a))$$

- Update equation is sarsa:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \underbrace{Q(s',a')}_{\text{Now } a \text{ is the action executed}} - Q(s,a))$$

SARSA algorithm for on-policy control

SARSA: on-policy learning

Initialize $Q(s,a), \forall s \in S, a \in \mathcal{A}(s)$, arbitrarily, and
$Q(terminal - state, \cdot) = 0$
for each episode **do**
  Choose initial state s
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  for each step of episode **do**
    Execute action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s'; a \leftarrow a'$
  end for
end for

SARSA algorithm for on-policy control

It can be proved that Sarsa converges to the optimal policy under the following conditions:

1. Greedy in the Limit of Infinite Exploration (GLIE):
   - All state–action pairs are explored infinitely many times:

$$\lim_{t \to \infty} N_k(s, a) = \infty$$

   - The policy converges on a greedy policy (f.i. ϵ decreases inversely proportional to the number of experiences)

$$\lim_{t \to \infty} \pi_t(a|s) = \arg\max_{a'} Q^{\pi_t}(s, a')$$

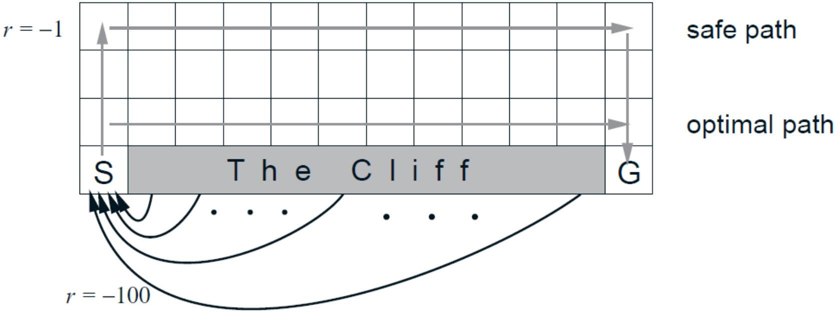
2. Robbins–Monro sequence of step–sizes α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Practical differences btw Sarsa and Q-learning

Cliff-Walk example ($\epsilon = 0.1$):



Practical differences btw Sarsa and Q-learning

Cliff-Walk example: Reward during learning

As ϵ decreases, sarsa tends to Q-learning

Practical differences btw Sarsa and Q-learning

- In the cliff–walking task:
  - Q–learning: learns optimal policy along edge
  - Sarsa: learns a safe non–optimal policy away from edge
- ϵ–greedy algorithm
  - For $\epsilon = 0$ Q-learning and Sarsa are identic.

Practical differences btw Sarsa and Q-learning

- In the cliff–walking task:
  - Q–learning: learns optimal policy along edge
  - Sarsa: learns a safe non–optimal policy away from edge

- ϵ–greedy algorithm
  - For $\epsilon = 0$ Q-learning and Sarsa are identic. But then no exploration.
  - For $\epsilon \rightarrow 0$ gradually, both converge to optimal.

Expected Sarsa

- In sarsa, we use the current policy to estimate returns
- However, we can do better: **Expected Sarsa**
- In sarsa, each episode uses one sample of action taken by the policy.
- ... but we know the policy probabilities to select one action (f.i. in ϵ-greedy procedure), so we can use it.
- Sarsa update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

- Expected sarsa update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \sum_a \pi(a'|s') Q(s',a') - Q(s,a))$$

- Same convergence guarantees and less variance than original Sarsa

Expected SARSA

Expected SARSA

Initialize $Q(s, a), \forall s \in S, a \in \mathcal{A}(s)$, arbitrarily, and $Q(s, \cdot) = 0$, $\forall s \in$ set of terminal states.

for each episode **do**
  Choose initial state s
  for each step of episode **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Execute action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \sum_{a'} \pi(a'|s')Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  end for
end for

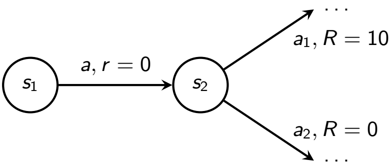
Off-policy vs. On-policy learning

- When learning value functions of a policy, we sample *using the policy* to estimate them
- In Q-learning, the method tries to learn the value function of the optimal policy (V^*) when in fact samples are obtained from different policy (ϵ-greedy policy)
- A subtle point with implications about the convergence of the algorithms to the optimal solution
- We'll do the following distinction:

On-policy learning: When learning the value function V^π of the current policy π

Off-policy learning: When Learning the value function V^π using another policy π'

Exercise: Off-policy vs. On-policy learning



- With this info, we know that $Q(s_2, a_1) = 10$ and $Q(s_2, a_2) = 0$
- Let's assume we obtain the following two experiences following the exploratory policy:
  - $(s, a) \rightarrow (s_2, a_2) \rightarrow ...$
  - $(s, a) \rightarrow (s_2, a_1) \rightarrow ...$
- Which is the value that Monte Carlo will obtain for $Q(s, a)$?
- Which is the value that Q-learning will obtain for $Q(s, a)$?

Exercise: Off-policy vs. On-policy learning

- MonteCarlo estimates $Q(s, a) = 5$, so it actually computes the behaviour policy, so it is *on-policy*
- Q-learning estimates $Q(s, a) = 10$. This is not the long term return from s of the behaviour policy. It is the return of the greedy policy.
- So Q-learning evaluates a different policy that the one used to collect the data. This is the definition of a *off-policy* algorithm.
- Notice that, using Q-learning, Q-values are not affected by bad results due to exploration. This is good because we can explore and still evaluate the greedy policy.
- In the limit, we can generate data using a random policy and still obtain the optimal policy!

Temporal Differences extended: conclusions

- Return to Temporal Differences extended
- Very good to **estimate values for a given policy**
- More difficult to apply to control, because trace of states visited follow one policy but we are estimating another one
- Remember this conclusions?

Temporal Differences extended: conclusions

- Return to Temporal Differences extended
- Very good to **estimate values for a given policy**
- More difficult to apply to control, because trace of states visited follow one policy but we are estimating another one
- Remember this conclusions?
- n-steps estimators and TD(λ) can be easily implemented to control for Sarsa and Extended Sarsa, because they are on-policy learning methods.

Sarsa(λ)

Initialize $Q(s,a)$ arbitrarily
loop
  $e(s,a) = 0$, for all s, a
  Initialize s, a
  repeat
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ–greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    for all s, a **do**
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    end for
    $s \leftarrow s'; a \leftarrow a';$
  until s is terminal
end loop

Temporal Differences extended: conclusions

- In off-policy learning, it is more difficult to implement *n-steps* methods
- However, n-steps still can be used in off-policy learning (f.i Q-learning). **The trick is to use a dynamic** n. When an exploratory action is taken then stop the trace of action from which to update Peng's Q(λ).
- In some implementations, authors ignore the problem (off-policy with n-steps) and, in some problems, it works well in practice.