# Reinforcement Learning

## Free model Algorithms

Mario Martin

Mario Martin - CS-UPC

February 11, 2026

# Goal of this lecture

- Problems with Policy Iteration method:
  - Sweep of full steps or random steps
  - **Need to know the model for policy evaluation**
- 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

# Monte-Carlo

# Monte-Carlo Policy Evaluation

- Goal: learn $V^\pi$ from episodes of experience under policy $\pi$

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

- 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 $\pi$ distribution (following $\pi$)

- 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

- 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:

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

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

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

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

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

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

# 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** $\alpha$ in range $(0 \ldots 1)$ is used:

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

- The same equation is applied to $Q(s, a)$ value functions

# 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 $Q^\pi$ for all states using a *policy evaluation* method

  **for** each state $s$ **do**

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

  **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)?
- Adapt Assync Policy Iteration: We don't sweep the whole set of states to update the Value estimates, neither the policy
- How we select the states to update? States updated are from the experience collected by the agent in one learning episode
- We update $Q$ using Bellman equation
- Apply the *improvement-of-the-policy* idea to learn the optimal policy.

# Monte-Carlo *policy learning*

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

Initialize $\pi$ and $Q$ randomly:
**repeat**
  Generate trial using $\pi$
  **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.

# $\epsilon$-greedy exploration

- Simplest idea for ensuring continual exploration
- All $m$ actions are tried with non-zero probability
- With probability 1 - $\epsilon$ choose the greedy action
- With probability $\epsilon$ 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 $\pi$ and $Q$ randomly:
**repeat**
  Generate trial using $\epsilon-$**greedy strategy on** $\pi$
  **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 $\pi$ and $Q$ randomly:
**repeat**
  Generate trial using exploration method based on $\pi$
  **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 $\pi$ 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

- 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:

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

- This is known as *bootstrapping*

# Temporal Differences *policy evaluation*

**Temporal Differences *policy evaluation***

Given $\pi$ 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

# 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
- Can we use it for policy learning?... Yes. Q-learning algorithm

# 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
- Can we use it for policy learning?... Yes. Q-learning algorithm

# Update equation of Q-values when *learning* the policy

- Use of Bellman equation for policy evaluation is:

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

- But when learning, **we know the optimal policy is greedy**

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

- So, for the policy in the next state we assume the greedy policy wrt Q values

$$\begin{aligned} Q(s,a) \leftarrow & Q(s,a) + \alpha \left(r + \gamma Q(s', \arg\max_{a'} Q(s',a') - Q(s,a)\right) \\ & Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \end{aligned}$$

# Temporal Differences *policy learning*

**Temporal Differences *policy learning* (first version)**

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

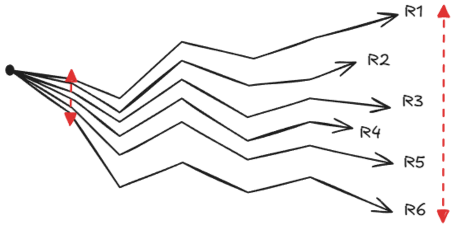
# Temporal Differences *policy learning*

**Q-learning: Temporal Differences *policy learning***

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

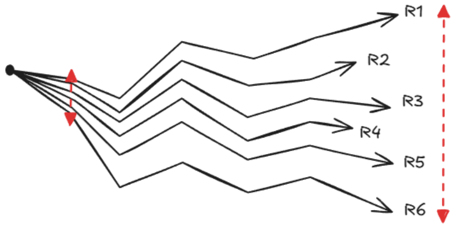
# Bias/variance trade-off MC an Q-learning

- 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 shows much lower variance than the MC return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one action, transition, reward
- Lower variance allow faster learning



# Bias/variance trade-off MC an Q-learning

- 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 shows much lower variance than the MC return:
  - Return depends on many random actions, transitions, rewards
  - TD target depends on one action, transition, reward
- Lower variance allow faster learning



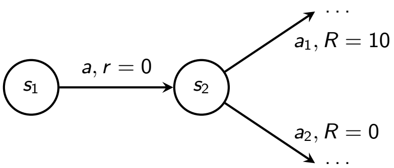
# On-policy vs. Off-policy learning: A subtle but important distinction

# 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 ($Q^*$) when in fact samples are obtained from different policy ($\epsilon$-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 $Q^\pi$ of the current policy $\pi$
  **Off-policy learning:** When Learning the value function $Q^\pi$ using another policy $\pi'$
- What about Q-learning and Monte-Carlo?

# 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:
  - $(s_1, a) \rightarrow (s_2, a_2) \rightarrow \ldots$
  - $(s_1, a) \rightarrow (s_2, a_1) \rightarrow \ldots$
- Which is the value that Monte Carlo will obtain for $Q(s_1, a)$?
- Which is the value that Q-learning will obtain for $Q(s_1, a)$?

# Two observed episodes (from an exploratory behavior policy)

The behavior policy is exploratory, so from $s_2$ it sometimes picks $a_1$ and sometimes $a_2$.

Assume $\gamma = 1$. Two sample episodes starting from $(s_1, a)$:

1. $(s_1, a) \rightarrow (s_2, a_2) \rightarrow \cdots$
   Return from $(s_1, a)$: $R = 0 + \gamma 0 = 0$
2. $(s_1, a) \rightarrow (s_2, a_1) \rightarrow \cdots$
   Return from $(s_1, a)$: $R = 0 + \gamma 10 = 10$

# Monte Carlo update (On-policy)

Monte Carlo estimates the action-value by averaging **observed returns** under the **same policy** that generated the data (the behavior policy).

$$Q_{\mathrm{MC}}(s_1, a) \approx \mathbb{E}_{\pi_b}[R \mid s_1, a]$$

With two equally-likely observed returns:

$$Q_{\mathrm{MC}}(s_1, a) = \frac{0 + 10}{2} = 5$$

# Q-learning update (Off-policy)

Q-learning uses a **greedy target** regardless of what happens in the whole episode (so valid for both episodes!):

$$Q_{\mathrm{QL}}(s_1, a) \leftarrow Q_{\mathrm{QL}}(s_1, a) + \alpha \Big[ r + \gamma \max_{a'} Q(s_2, a') - Q_{\mathrm{QL}}(s_1, a) \Big]$$

Here the one-step target is:

$$r + \gamma \max_{a'} Q(s_2, a') = 0 + \gamma \max(10, 0)$$

because $\gamma = 1$:

$$\text{Target} = 10 \quad \Rightarrow \quad Q_{\mathrm{QL}}(s_1, a) \text{ is pushed toward } 10$$

# 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

# 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 extended TD($\lambda$)

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

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

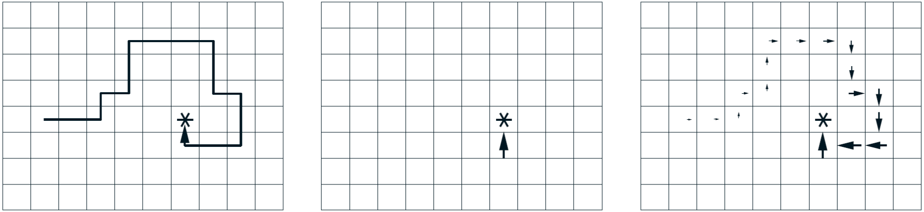
- Can be rewritten for episodes as:

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

- Unifies different algorithms:
  - When $\lambda = 0$ we have TD(0), the standard Q-learning method
  - When $\lambda = 1$ we have the standard MC method

# Temporal Differences intuition

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



Monte-Carlo    Q-learning    TD($\lambda$)

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

# Temporal Differences extended: conclusions

- Very good to **estimate values for a given policy**
- Difficult to implement
- It is a mix between on-policy and off-policy
- You need more parameters to guess ($n$ or $\lambda$)