Learning via Monte-Carlo Methods and Temporal Difference

Joseph Le Roux

10/07/26

Outline

- MDP (Review)
- Monte-Carlo Methods
- Temporal Difference Methods
- Exploration vs. Exploitation

1
MDP (Review)

1 Markov Decision Process (MDP)

MDP (S, A, T, R, γ, H)**:**

- S is a set of states (of the environment)
- A is a set of actions (performed by the robot)
- $H \in \mathbb{N} \cup \{\infty\}$ maximum time (horizon)
- p_T transition distribution:

$$p_T(S_{t+1} = s' | S_t = s, A_t = a)$$

  (*dynamics*) represents uncertainty of action results

- $R : S \times A \times S \longrightarrow \mathbb{R}$ reward function $R(s, a, s')$
- $\gamma \in [0; 1]$ *discount* weighs down decisions in the future
- We add an initial state s_0 and a set of final states F (possibly empty)

1 MDP (trajectory and policy)

Trajectory

From s_0, sequence of (action, reward, state): $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{F-1}, a_{F-1}, r_F, s_F$, with $F \leq H$
gain at time t: sum of exponentially discounted rewards from t to the end of trajectory: $G_t = \sum_{k=t+1} \gamma^{k-t-1} r_k$

Policy π: Which action will the robot choose?

1. Deterministic: $\pi : S \longrightarrow A$
2. Stochastic : $\pi : S \times A \longrightarrow [0, 1]$ conditional distribution $\pi(a|s) = p(A_t = a|S_t = s)$

Remarks:

- the gain corresponds to 1 particular trajectory τ, but is omitted in notation!
- given deterministic π_d, one can define stochastic π: $\pi(a|s) = 1$ if $\pi_d(s) = a$, 0 otherwise
- given stochastic π, one can define deterministic (greedy) $\pi_g(s) = \operatorname{argmax}_a(\pi(a|s))$

1 MDP: State Values (1)

Policy And State Values

average gain (score) of a state: stochasticity of policy and/or transitions $\longrightarrow$ expectation

Expected Gain = *state value*

$v^\pi(s)$: average of possible gains from s, weighted by π and p_T:

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=t+1} \gamma^{k-t-1} R_k | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=t+2} \gamma^{k-t-1} R_k | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} | S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} | S_t = s] \end{aligned}$$

Remark

- R_k : random variable over rewards at time k, and r_k actual value chosen at time k in the trajectory.
- We only index expectations by π because this is the only distribution that can change in the MDP (p_T is fixed)

1 MDP: State Values (2)

Compute v^π in practice

start from $v^\pi(s) = \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1}|S_t = s]$

- $\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p_T(s'|s,a) R(s,a,s')$
- $\mathbb{E}_\pi[G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p_T(s'|s,a) \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']$
- $\mathbb{E}_\pi[G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p_T(s'|s,a) v^\pi(s')$

In summary:

- If s_t is terminal (either $s_t = s_H$, or s_t labelled as final) : $v^\pi(s_t) = 0$ (no more possible gain)
- otherwise:

$$v^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p_T(s'|s,a)(R(s,a,s') + \gamma v^\pi(s'))$$

Bellman (expectation) equation

state-action values

Expected gain given state s and action a $q(s,a) = \mathbb{E}_\pi[G_t|S_t = s; A_t = a]$
Analog recurrence:

- if s terminal $\forall a, q(s,a) = 0$
- otherwise:

$$q^\pi(s,a) = \sum_{s'} p_T(s'|s,a)(R(s,a,s') + \gamma v^\pi(s'))$$

2
Monte-Carlo Methods

2 *Monte-Carlo* Methods to compute state values

In the remainder, we assume that we do not know the dynamics p_T of MDPs

State Values v^π

cannot be computed directly via Bellman formula (we don't know p_T)

Idea: replace expectation by finite average!

This is called *Monte-Carlo* simulation

1. We have an agent in the environment, let's use it!
2. We let it act in the environment to collect trajectories
3. When collection is sufficient, approximate expectation over gains by averaging over collected trajectories: we have estimated v^π

Drawback

How many trajectories to have a accurate estimate of v^π? (short answer: an awful lot!)

2 MC Policy Evaluation: Approximating v^π with MC

MC Policy Evaluation Algorithm: compute v^π

Naive implementation! Horribly slow because we need to store `G` and compute average (will fix this later)

- Input : a policy π
- Initialisation
  - table V such that $\forall s \in S, V(s) \in \mathbb{R}$ (arbitrary initialisation, 0 for terminals)
  - table G (stores gains) such that $\forall s \in S, G(s)$ is the empty list

Loop:

- Generate a trajectory from s_0 and π: $s_0, a_0, r_1 \dots, s_{T-1}, a_{T-1}, r_T, s_T$
- $g \leftarrow 0$ *(gain of the final state of π)*
- for steps in trajectory $t = T-1, T-2 \dots, 0$ in reverse order
  - $g \leftarrow \gamma g + r_{t+1}$ *(gain of s_t given s_{t+1})*
  - $G(s_t) \leftarrow G(s_t) :: g$ *append new gain to the list of gains observed with s_t*
  - $V(s_t) \leftarrow \text{average}(G(s_t))$ *(update estimate)*

2 *Monte-Carlo* from state values to state-action values (1)

Remarks on previous algorithm

- Does not use Bellman, does not use p_T
- can also be used to compute q^π (replace $V(s)$ with $Q(s,a)$)
  - 👉 this is how we will use it in lab session

Remember why we want to compute v?

- to compute optimal π
- problem: previously seen methods from v to π need p_T 🤯

But if we know $q(s,a)$, we can compute π without p_T

Depends on the form of π we want:

- deterministic (greedy) $\pi(s) = \operatorname{argmax}_a q(s,a)$ $\longrightarrow$ this is a consequence of BOE
- stochastic: several solutions (see next slide)

2 *Monte-Carlo* from state values to state-action values (2)

But if we know $q(s,a)$, we can define compatible π *without p_T:

Exponential distribution (aka *softmax*):

$\pi(a|s) = \frac{\exp(q(s,a))}{\sum_{a'} \exp(q(s,a'))}$

- define the probability by *ranking* all q values
- often too flat (will often return suboptimal a)

ε-greedy distribution

$\pi(a|s) \geq \frac{\varepsilon}{|A|}, \forall a$ but with tunable preference for optimal a

- with probability ε pick an action following uniform distribution over A (i.e. $p = \frac{1}{|A|}$)
- otherwise (with probability $1 - \varepsilon$) deterministic greedy (i.e. pick the best action)

🤔 This amounts to (proof on blackboard):

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|A|} & \text{if } a \neq \operatorname{argmax}_a q(s,a) \\ \end{cases}$$

2 MC Policy Iteration: Approximate π^* via q

After each trajectory, we re-evaluate q^π with MC and improve the policy.

MC Algorithm for policy iteration via q and (ϵ-greedy) π : compute optimal policy

- Input : parameter $0 < \epsilon \leq 1$
- Init:
  - table $Q(s,a) \longleftarrow 0$ (or random)
  - $\pi \longleftarrow \epsilon$-greedy initialised from Q (see previous slide)
  - table G such that $\forall s \in S, a \in A, G(s,a)$ = empty list
- Loop:
  - generate trajectory $s_0, a_0, r_1 \dots, s_{T-1}, a_{T-1}, r_T, s_T$
  - $g \longleftarrow 0$
  - for all steps $t = T-1, T-2 \dots, 0$
    - $g \longleftarrow \gamma g + r_{t+1}$
    - $G(s_t, a_t) \longleftarrow G(s_t, a_t) :: g$ and $Q(s_t, a_t) \longleftarrow$ average$(G(s_t, a_t))$
  - set π from Q (see previous slide) *(quiz: is this update π to a $\geq$ policy 🤔?)*

2 MC Policy Iteration: Approximate π^* via q

After each trajectory, we re-evaluate q^π with MC and improve the policy.

MC Algorithm for policy iteration via q and (ϵ-greedy) π : compute optimal policy

- Input : parameter $0 < \epsilon \leq 1$
- Init:
  - table $Q(s,a) \longleftarrow 0$ (or random)
  - $\pi \longleftarrow \epsilon$-greedy initialised from Q (see previous slide)
  - table G such that $\forall s \in S, a \in A, G(s,a)$ = empty list
- Loop:
  - generate trajectory $s_0, a_0, r_1 \dots, s_{T-1}, a_{T-1}, r_T, s_T$
  - $g \longleftarrow 0$
  - for all steps $t = T-1, T-2 \dots, 0$
    - $g \longleftarrow \gamma g + r_{t+1}$
    - $G(s_t,a_t) \longleftarrow G(s_t,a_t) :: g$ and $Q(s_t,a_t) \longleftarrow \text{average}(G(s_t,a_t))$
  - set π from Q (see previous slide) *(quiz: is this update π to a $\geq$ policy 🤔?)*

Remark: First-Visit vs. Every-Visit

Above, we present the every-visit version: $G(s_t,a_t) \longleftarrow G(s_t,a_t) :: G$ is always performed.
Alternatively the G update can be performed first-visit: only the first time (s_t,a_t) appears in the trajectory.

(hint: see function evaluate in 'TP')

```
terminated, truncated = False, False
state = initial_state
while not (terminated or truncated):
    action = give_me_an_action(state, pi)
    newstate,reward,terminated,truncated,_ = env.step(action)
    ...
```

- need to store states, actions and rewards
- need to compute gains

3 Temporal Difference Methods

3 Monte-Carlo and Bellman

We have seen that:

- definition of V: $V(s) = \mathbb{E}[G_t|S_t = s]$
- update MC : $V(s) \leftarrow$ average$\{g_t$ such that $s_t = s\}$
- update Bellman : $V(s) \leftarrow \mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[V(s')|S_t = s]$

Remarks

1. MC does not use Bellman and must estimate $G_t \longrightarrow$ must generate complete trajectories
2. Bellman constrains consistency between state values of successors: we want $\{\mathbb{E}[R_{t+1}|S_t = s] + \gamma\mathbb{E}[V(s')|S_t = s]\} - V(s) = 0$
3. Bellman can be applied for s as soon as V is available for successors s'
4. Bellman on one trajectory only (no expectation): $V(s_t) \leftarrow r_{t+1} + \gamma V(s_{t+1})$
   - we call $\tau_t = r_{t+1} + \gamma V(s_{t+1})$ the TD target at t, and $\delta_t = \tau_t - V(s_t)$ the TD error at t

Temporal Difference Methods: *the best of both worlds!*

- sample a trajectory like MC
- use Bellman (on this trajectory) to speed up convergence

3 Efficient Monte-Carlo Update

Review MC computation of v. The MC update for the k^{th} update $V(s)$ is

$$
\begin{aligned}
V^k(s) &= \frac{1}{k}\sum_{i=1}^{k} G(s)[i] \quad \text{in other words } V(s_t) \leftarrow \text{average}(G(s_t)) \\
&= \frac{1}{k}\Big(\sum_{i=1}^{k-1} G(s)[i]\Big) + \frac{1}{k} G(s)[k] \\
&= \frac{1}{k}\{(k-1)V^{k-1}(s) + G(s)[k]\} \\
&= \frac{k-1}{k} V^{k-1}(s) + \frac{1}{k} g \\
&= (1-\alpha_k)V^{k-1}(s) + \alpha_k g \qquad \text{with } \alpha_k = \frac{1}{k} \\
&= V^{k-1}(s) + \alpha_k\{g - V^{k-1}(s)\}
\end{aligned}
$$

MC algorithm update can be written as:

$V(s_t) \leftarrow V(s_t) + \alpha\Big(G_t - V(s_t)\Big)$ Use this version in 'TP', not the naive one!! (and adapt to Q)

3 Temporal Difference Methods

From MC update to TD update via *Bellman*

$$V(s_t) \quad \leftarrow \quad V(s_t) + \alpha\Big(G_t - V(s_t)\Big)$$

1. When no error, we should have $V(s_t) = G_t$
2. But also have by definition of G that $G_t = r_{t+1} + \gamma G_{t+1}$, so

$$V(s_t) \quad \leftarrow \quad V(s_t) + \alpha\Big(r_{t+1} + \gamma G_{t+1} - V(s_t)\Big)$$

1. When no error, we should also have $V(s_{t+1}) = G_{t+1}$

$\Rightarrow$ We can replace G_t by $r_{t+1} + \gamma V(s_{t+1})$ in MC update. We obtain:

$$V(s_t) \quad \leftarrow \quad V(s_t) + \alpha\Big(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big)$$

For q:

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

We only use Q in the remainder, since it can be used to compute optimal policies directly

3 SARS'A' Algorithm

Let us modify MC algorithm by changing updates $Q(s,a)$ to TD updates

- each step can be seen as a tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ (or a slice) of the current trajectory
- hence the name of the algorithm!

On Policy Temporal Difference Policy Iteration

- Input: step size $\alpha > 0$ and threshold $\epsilon > 0$
- Init:
  - Table $Q(s,a), s \in S, a \in A$ s.t. if s is terminal, $Q(s,a) = 0$

Loop:

- $s \leftarrow S_0$ (initialise state)
- Choose a from Q and s (choose from ε-greedy policy)
- While s is not terminal
  - from s, a get reward and next state r, s'
  - Choose a' from Q and s'
  - $Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$ (TD Update)
  - $s \leftarrow s'$ and $a \leftarrow a'$

3 SARS'A' : on-policy vs. off-policy

Behavior policy vs. Target policy

so far, only one policy π, but it can be convenient to distinguish:

- π the target (optimal) policy, the policy improved after each update
- μ the behaviour policy, used to collect data, here (s, a, r, s', a') tuples

SARSA is on-policy

we say an algorithm is

on-policy if $\mu = \pi$: we collect data with (our estimate of) π

off-policy otherwise, $\mu \neq \pi$ collecting is performed using another policy

Off-Policy Updates

Using the sampled a' for evaluating $V(s_{t+1})$ (and thus G_{t+1}) may lead to unstability. We can either average over actions or take the greedy action.

Replace only (TD Update) (and simplify the rest if needed)

Expected-SARSA

Use an average of next actions instead of only a' for estimation:

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

Q-Learning

Start approximating as an update from q_* directly, always use the best action for estimation of gain:

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

This is the *one-trajectory* version of the Bellman Optimality Equation (q/q version)

4
Exploration vs. Exploitation

4 Exploration vs. Exploitation tradeoff

Exploration when $\epsilon \approx 1$

1. guarantees all states get visited eventually
2. prevents from concentrating only on known good states

This is the behaviour we want at the beginning of the SARSA loop

Exploitation when $\epsilon \approx 0$

1. use the knowledge of past experiences to guide the agent more
2. avoid trajectories and states which are bad, and useless for SARSA update

This is the behaviour we want at the end of the SARSA loop

In practice: from exploration to exploitation

We start with ϵ close to 1, and decrease it *little by little* at each iteration of the loop until 0.