REINFORCEMENT LEARNING

FUNDAMENTALS + APPLICATIONS

WEEK 4

MONTE CARLO FOR PREDICTION AND CONTROL

MONTE CARLO IN RL

Learning Targets for the Week

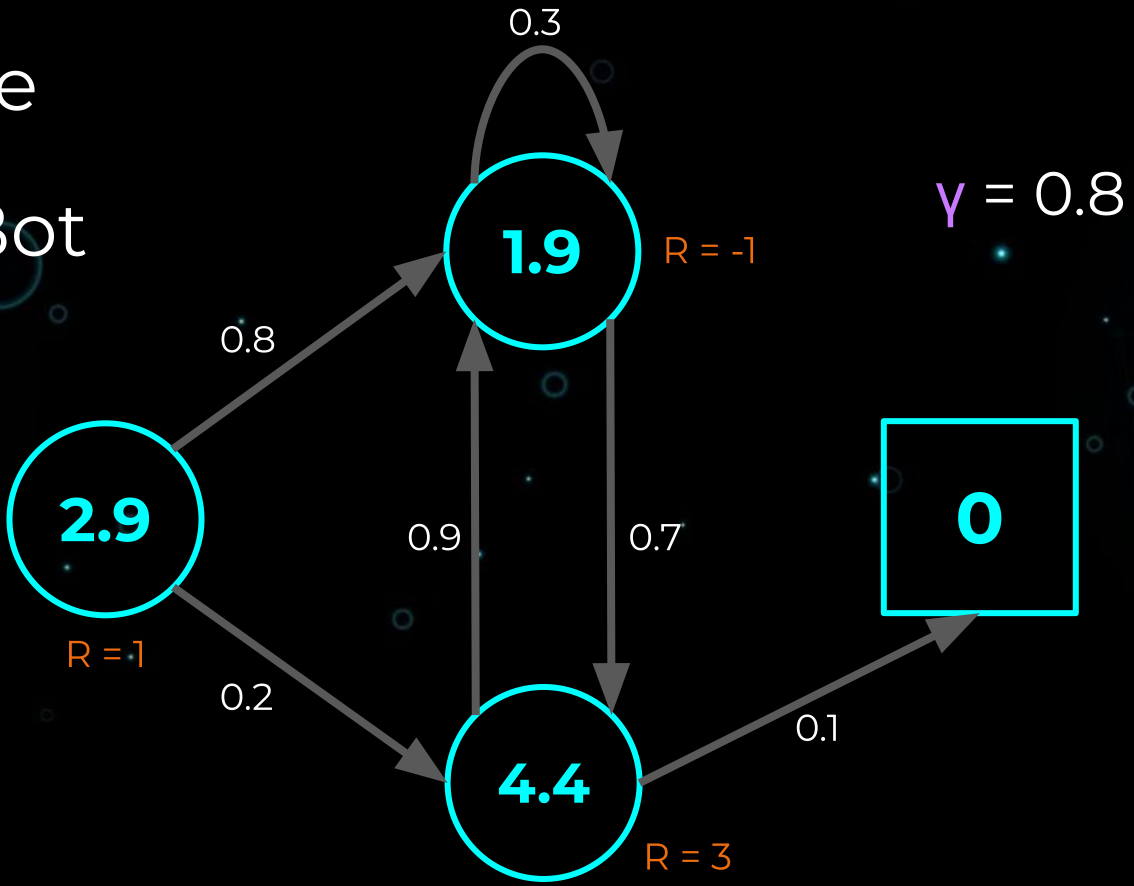
- What is Monte Carlo sampling? Why is it useful in RL? What are its drawbacks? How is it different from dynamic programming?
- How does first-visit MC work? Every visit?
- What is an “ε-soft” policy?
- What is off-policy learning?
- What is the difference between on-policy and off-policy?
- How do off-policy MC algorithms work? What are some problems that emerge in off-policy learning?
- What are importance weights, and why do we need them?

MONTE CARLO IN RL

Model-Free Prediction and Control

- Estimates expected return via sampling and calculating the empirical mean of experienced returns.
- Depends on episodes. Environment must be episodic.
- For use when the environment (state space) is too complex to compute the value function via dynamic programming, or when the environment is unknown.
- Run episode -> calculate returns to each state -> append to list -> average.

Remember The GoBot



FIRST-VISIT MONTE CARLO

On-Policy Prediction

Initialize: π (to be evaluated), $\mathbf{V}(s) = 0\ \forall\ s$, $returns(s) = [\,]\ \forall\ s$

```
Repeat for N iterations:
    trajectory ← run_episode(π)
    visits ← get_first_visits(trajectory)
    For state in visits:
        G ← Return following state visit.
        returns[state].append(G)
        V(state) <- average(returns[state])
```

FIRST-VISIT MONTE CARLO

On-Policy Prediction : Example

Episode 1

S(A), **Go**(A), +3,
S(A), **Go**(B), +2,
S(B), **Go**(A), -4
S(A), **Go**(B), +4,
S(B), **Go**(T), -3
Terminate

Episode 2

S(B), **Go**(A), -2,
S(A), **Go**(B), +3,
S(B), **Go**(T), -3,
Terminate

V(A) = ½ * ((3 + 2 - 4 + 4 - 3) + (3 - 3)) = ½ * 2 = 1

V(B) = ½ * ((-4 + 4 - 3) + (-2 + 3 - 3)) = ½ * (-5) = -2.5

EVERY-VISIT MONTE CARLO

On-Policy Prediction

Initialize: π (to be evaluated), $\mathbf{V}(s) = 0\ \forall\ s$, $returns(s) = [\,]\ \forall\ s$

Repeat for N iterations:

    trajectory ← run_episode(π)
    For state in trajectory:
        G ← Return following state visit.
        returns[state].append(G)
        V(state) <- average(returns[state])

MONTE CARLO METHODS

On-Policy Control : ε-soft

- To achieve convergence to the optimal policy, Monte Carlo methods must ensure they visit every state-action pair (in the limit, i.e., “eventually”).

- Greedy on-policy control does not work because we might get “locked out” of some patterns of behaviour by greed.
  - Think back to bandits and the reasoning behind optimistic initial values: we want to avoid early samples completely forcing us away from trying other options.

- As a result, we need to include some element of guaranteed constant exploration into our control method. Epsilon-greedy is a way to do this, which is a type of ε -soft policy.

- An ε -soft policy gives every available action *at least* a probability of ε / |A(s)| of being chosen, i.e., epsilon divided by the number of actions available from state s.
  - Note that uniform random is an example of ε -soft .

MONTE CARLO METHODS

On-Policy Control : ε-soft

a

ε - Greedy

$$A \leftarrow \begin{cases} \text{argmax}_a Q(a) \text{ with probability } 1 - \varepsilon \\ \text{Random action with probability } \varepsilon \end{cases}$$

ε - Soft

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |A(s)| & \text{if } a = A^* \\ \varepsilon / |A(s)| & \text{if } a \neq A^* \end{cases}$$

FIRST-VISIT MONTE CARLO

On-Policy Control

Initialize: $\pi(a|s) \leftarrow$ arbitrary ε-soft policy,
$Q(s,a) = 0 \ \forall \ s,a$
$returns(s) = [\,] \ \forall \ s,a$

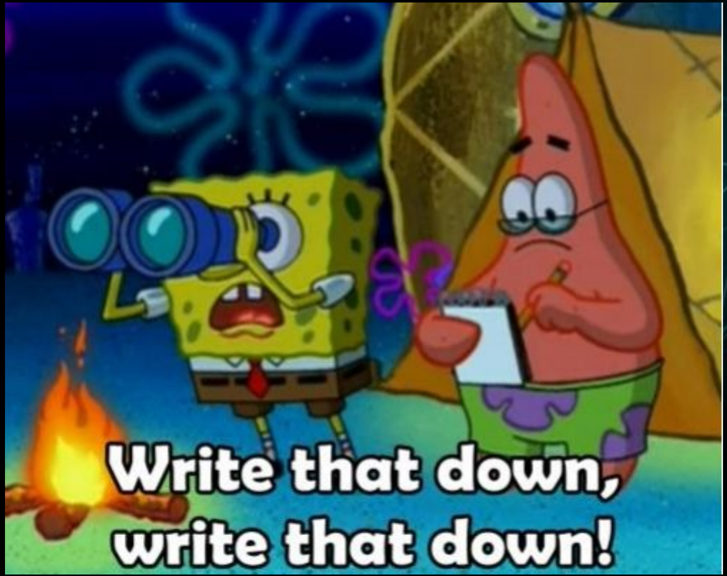
a

While true:
    trajectory ← run_episode(π)
    first_visits ← get_first_visits(trajectory)
    For state, action in first_visits:
        G ← Return following state, action pair.
        returns[state, action].append(G)
        Q(s,a) <- average(returns[state, action])
    For state in trajectory:
        $A^* = \text{argmax}\,Q(s,a)$
        For all actions available from state ($a \in A(s)$):

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |A(s)| & \text{if } a = A^* \\ \varepsilon / |A(s)| & \text{if } a \neq A^* \end{cases}$$

$$\frac{\varepsilon}{|A(s)|}$$

ON-POLICY VS. OFF-POLICY



ON-POLICY VS. OFF-POLICY

- What we've been doing so far.
- "Learn on the job"
- One policy used for both learning (prediction) and improvement (control).
- Exploration must be part of the single policy.
- More simple.

- The new concept.
- "Look over someone's shoulder"
- Two policies, one for learning and one for improvement.
- Exploration can be isolated just to the learning policy.
- More complex but more flexible (e.g., enables learning from historical data, other agents/policies, human examples, ...)

OFF-POLICY LEARNING

Behavioural Policy

b

The Explorer

Generates Experiences

Prediction

→

Value Estimates or Estimator (V, Q, Q-Network, …)

→

Target Policy

π

The Exploiter

Learns from Experiences

Control

OFF-POLICY LEARNING

- Behavioural policy *b* must ensure *coverage:* It must be able to visit all states the optimal policy would visit.
  - We can learn π_* **even if *b* is random**, so that is an easy way to get coverage.

- We must include some correction for the difference between the behavioural policy and the target policy. This is usually done through *importance weights*.

OFF-POLICY LEARNING

Ordinary Importance Sampling

$$\frac{\text{The probability of choosing action } a \text{ in state } s \text{ under target policy } \pi}{\text{The probability of choosing action } a \text{ in state } s \text{ under behavioural policy } b} \qquad \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Ordinary Importance Weight

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

To be clear, this is the weighted relative probability of the trajectory up to timestep T-1.

Which is just a lot of multiplication of probability ratios (arithmetic).

The ordinary importance weight of the return from time t through T-1 is equal to the ratio between the state-action probability under the target and behavioural policies, multiplied by the ratio at time t+1, … multiplied by the ratio at time T-1.

OFF-POLICY LEARNING

Ordinary Importance Sampling

$$\mathbb{E}_{X\sim P}[f(X)] = \Sigma P(X)f(X)$$

$$= \Sigma Q(X)\,\frac{P(X)}{Q(X)}\,f(X)$$

$$= \mathbb{E}_{X\sim Q}\frac{P(X)}{Q(X)}\,f(X)$$

OFF-POLICY LEARNING

Ordinary Importance Sampling

- If an action is very likely in target policy π, but unlikely in behavioural policy b, the numerator is large compared to the denominator and the transition is more heavily weighted.
    - If this difference is huge, e.g. $\pi(A_t|S_t) = .9$ vs. $b(A_t|S_t) = .001$, the weight is also huge and the variance explodes. To fix this we use *weighted importance sampling*.

- If the action is unlikely under π but likely under b, the transition is more lightly weighted.

OFF-POLICY LEARNING

Ordinary vs. Weighted Importance Sampling

Normal Monte Carlo Sample Averaging

$$V(s) = \frac{\Sigma G_t}{T(s)}$$

The sum of the returns experienced following visits to state s, divided by the number of times we have visited state s.

OFF-POLICY LEARNING

Ordinary vs. Weighted Importance Sampling

Unfortunate Ordinary Importance Sampling

$$V(s) = \frac{\Sigma p_{t:T-1} G_t}{T(s)}$$

Each return is scaled, but the average is still calculated using the number of visits T(s)

Clearly Superior Weighted Importance Sampling

$$V(s) = \frac{\Sigma p_{t:T-1} G_t}{\Sigma p_{t:T-1}}$$

This makes the average of the returns a true *weighted* average, where the weights are the ratio between the likelihood of the action under the target policy vs. the behavioural policy.

OFF-POLICY LEARNING

Weighted Importance Sampling: The Easy Way

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n]$$

$W_i = p_{t:T(t)-1}$

$C_n = \Sigma W_i$

Just like in bandits, we can incrementally update our weighted average return. Whew!!!

All we do is divide the latest weight W_n by the sum of the weights so far C_n, and then multiply that by the difference between the current return G_n and our current weighted return V_n.

OFF-POLICY LEARNING

All three equations in incremental form

Sample Averaging

$$V(S_t) = V(S_t) + (1/n)[G_t - V(S_t)]$$

Ordinary Importance Sampling

$$V(S_t) = V(S_t) + (1/n)[p_{t:T(t)-1}G_t - V(S_t)]$$

Weighted Importance Sampling

$$V_{n+1} = V_n + \frac{W_n}{C_n}[G_n - V_n]$$

$$W_i = p_{t:T(t)-1}$$

$$C_n = \Sigma W_i$$

MONTE CARLO

Off-Policy Prediction with Weighted Importance Sampling

Input: π (arbitrary target policy),
Initialize $\forall$ s, a: $Q(s,a) \leftarrow 0$, $C(s,a) \leftarrow 0$

Repeat for N iterations:
- $b \leftarrow$ any policy with coverage of π
- trajectory = generate_episode(b)
- $G \leftarrow 0$
- $W \leftarrow 1$
- For T = T-1, ..., 0 (i.e., run through trajectory in reverse):
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t,A_t) \leftarrow C(S_t,A_t) + W$
  - $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + (W / C(S_t,A_t)) \cdot [G - Q(S_t,A_t)]$
  - $W \leftarrow W \cdot [\pi(A_t|S_t) / b(A_t|S_t)]$

  If $W = 0$ (i.e., if an action is selected that would never be selected
  In the target policy; $\pi(A_t|S_t) = 0$), exit loop.

MONTE CARLO

Off-Policy Control

Initialize $\forall\ s, a$: $Q(s,a) \leftarrow 0$, $C(s,a) \leftarrow 0$,
$\pi \leftarrow$ **argmax** $Q(S_t,A_t)$ **ties broken consistently, not randomly*

Repeat for N iterations:
$b \leftarrow$ any soft policy
trajectory = generate_episode(b)
$G \leftarrow 0$
$W \leftarrow 1$
For $T = T-1, \ldots, 0$ (i.e., run through trajectory in reverse):
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t,A_t) \leftarrow C(S_t,A_t) + W$
$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + (W / C(S_t,A_t)) \cdot [G - Q(S_t,A_t)]$
$\pi(S_t) \leftarrow$ **argmax** $Q(S_t,A_t)$ **ties broken consistently, not randomly*
If $A_t \neq \pi(S_t)$ then exit loop
$W \leftarrow W \cdot [1 / b(A_t|S_t)]$