Reinforcement Learning

Lecture 5: Monte Carlo methods

Robert Lieck

Durham University

Lecture overview

Lecture covers chapter 5 in Sutton & Barto [1] and adaptations from David Silver [2]

1. **Introduction**
   - history of Monte Carlo methods
   - definition
2. **Monte Carlo prediction**
   - overview
   - definition
   - incremental means
   - prediction with incremental updates
3. **Monte Carlo control**
   - policy iteration using action-value function
   - don’t just be greedy!
   - ϵ-greedy exploration
   - greedy at the limit of infinite exploration
   - Monte Carlo tree search

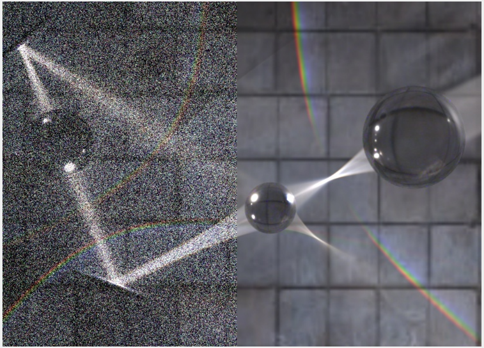
History: Monte Carlo methods

Invented by Stanislaw Ulman in the 1940s, when trying to calculate the probability of a successful Canfield solitaire. He randomly lay the cards out 100 times, and simply counted the number of successful plays.

Widely used today, for example:

- Path tracing in compute graphics
- Computational physics, chemistry, ...
- Grid-free PDE solvers [3]

Example: Monte Carlo path tracing



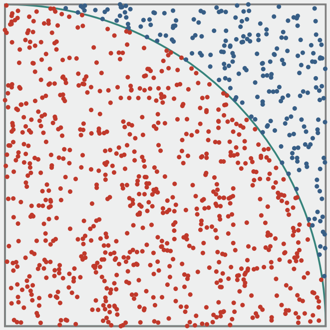
Definition: Monte Carlo method

Apply repeated random sampling to obtain numerical results for difficult or otherwise impossible problems

General approach:

1. Define a domain of possible inputs
2. Generate inputs randomly from a probability distribution over the domain
3. Perform a deterministic computation on the inputs
4. Aggregate (e.g. average) the results

Example: approximating π



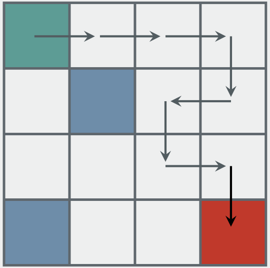
Monte Carlo reinforcement learning overview

Overview: MC reinforcement learning

Monte Carlo reinforcement learning learns from **episodes of experience**:

1. Empirical risk minimisation
2. It's **model-free** (requires no knowledge of MDP transitions/rewards)
3. Learns from complete episodes (you have to play a full game from start to finish)
4. One simple idea: the value function = the empirical mean return

MC RL samples complete episodes



Monte Carlo reinforcement learning definition

Definition: MC reinforcement learning

Putting this together, we sample episodes from experience under policy π

$$S_1, A_1, R_2, S_2, A_2, ..., S_k \sim \pi,$$

where we're going to look at the total discounted reward (the return) at each timestep onwards

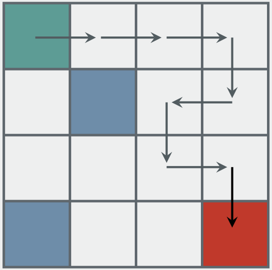
$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T,$$

and our value function as the expected return

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s].$$

With MC reinforcement learning, we use an empirical mean instead of the expected return.

Example: episode

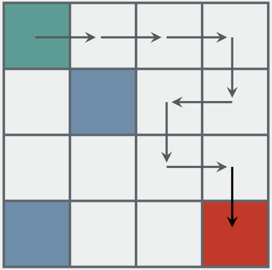


Definition: Incremental means

RL algorithms use incremental means, where $\mu_1, \mu_2, \ldots$ from a sequence is computed incrementally

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)
\end{aligned}$$

Example: episode



Definition: MC prediction, incremental updates

Putting this together, we sample episodes from experience under policy π

$$S_1, A_1, R_2, S_2, A_2, ..., S_T \sim \pi,$$

and every time we visit a state, we're going to increase a visit counter, then we will use our running mean:

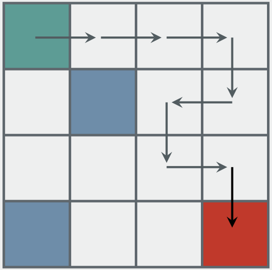
$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

It's common to also just track a running mean and forget about old episodes:

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

Example: episode



Problem: model-free learning. **Solution:** Q

Simply greedily improving the policy over $V(s)$ requires a model:

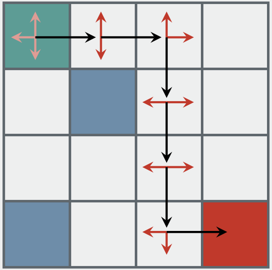
$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s'),$$

whereas greedy policy improvement over $Q(s,a)$ is model-free:

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

Follow along in Colab:

Example: caching Q-values



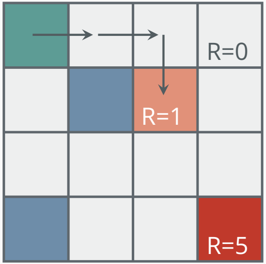
Monte Carlo methods don't just be greedy!

Algorithm: greedy MC that will get stuck

```
Q = np.zeros([n_states, n_actions])
n_visits = np.zeros([n_states, n_actions])

for episode in range(num_episodes):
  s = env.reset(), done = False, result_list = []
  while not done:
→   a = np.argmax(Q[s, :])
    s', reward, done, _ = env.step(a)
    results_list.append((s, a))
    result_sum += reward
    s = s'

  for (s, a) in results_list:
    n_visits[s, a] += 1.0
    α = 1.0 / n_visits[s, a]
    Q[s, a] += α * (result_sum − Q[s, a])
```



Definition: ϵ-greedy exploration

The simplest idea to avoid local minima is:

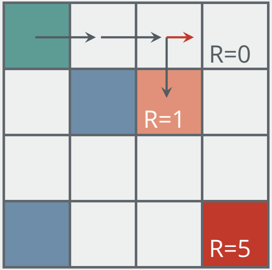
- choose a random action with probability ϵ
- choose the action greedily with probability $1 - \epsilon$
- where all m actions are tied with non-zero probability

This gives the updated policy:

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

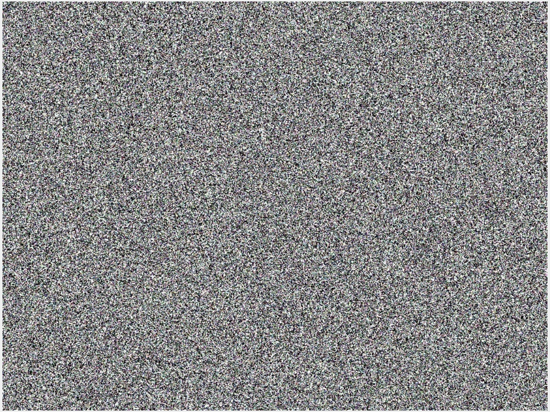
Proof of convergence in Equation 5.2 of [1]

Problem: local minima



Monte Carlo methods don't just explore!

Asymptotically we can't just explore...



Definition: greedy at the limit with infinite exploration (GLIE)

Defines a schedule for exploration, such that these two conditions are met:

1. You continue to explore everything

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

2. The policy converges on a greedy policy:

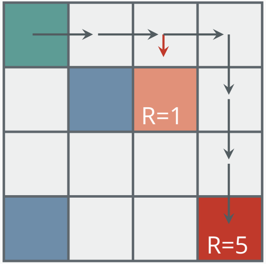
$$\lim_{k \to \infty} \pi_k(a|s) = \mathbf{1}(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a'))$$

Monte Carlo methods greedy at the limit of infinite exploration

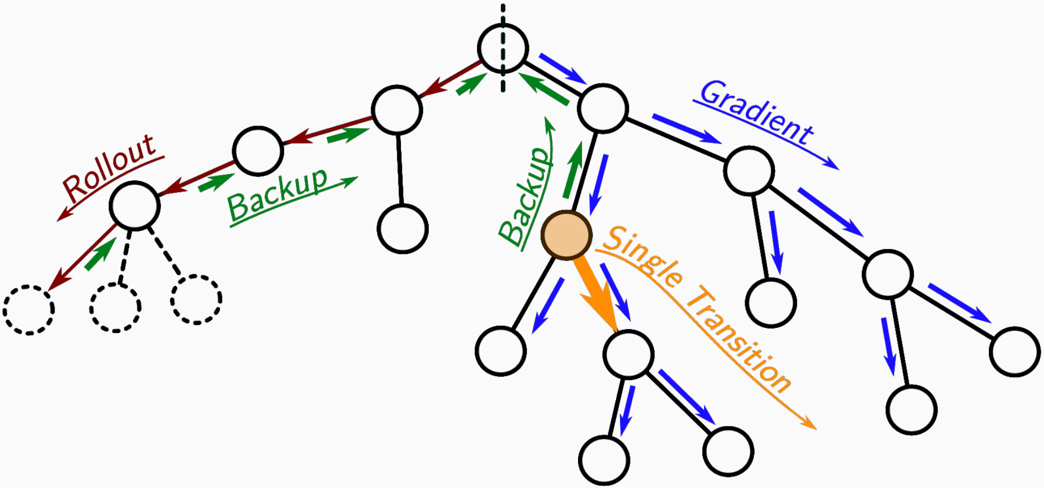
Algorithm: greedy at the limit of ∞ exploration

```
..
for episode in range(num_episodes):
  s = env.reset(), done = False, result_list = []
  while not done:
→   ϵ = min(1.0, 10000.0/(episode+1))
     if np.random.rand() > ϵ:
       a = np.argmax(Q[s, :])
     else:
       a = env.action_space.sample()
    s', reward, done, _ = env.step(a)
    results_list.append((s, a))
    result_sum += reward
    s = s'

  for (s, a) in results_list:
    n_visits[s, a] += 1.0
    α = 1.0 / n_visits[s, a]
    Q[s, a] += α * (result_sum − Q[s, a])
```



Monte Carlo methods Monte Carlo tree search



Take Away Points

Summary

In summary, Monte Carlo RL methods:

- are a solution to the reinforcement learning problem
- require training with complete episodes
- are model-free
- can balance exploration vs exploitation
- eventually converge on the optimal action-value function

References I

[1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction (second edition). Available online. MIT press, 2018.

[2] David Silver. Reinforcement Learning lectures. https://www.davidsilver.uk/teaching/. 2015.

[3] Rohan Sawhney and Keenan Crane. "Monte Carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains". In: ACM Transactions on Graphics (TOG) 39.4 (2020), pp. 123–1.