Reinforcement Learning
Lecture 8

Gillian Hayes

1st February 2007

Algorithms for Solving RL: Monte Carlo Methods

- What are they?
- Monte Carlo Policy Evaluation
- First-visit policy evaluation
- Estimating Q-values
- On-policy methods
- Off-policy methods

Monte Carlo Methods

- **Learn** value functions
- **Discover** optimal policies
- Don't require environmental knowledge: $P^a_{ss'}$, $R^a_{ss'}$,
  cf. Dynamic Programming
- Experience : sample sequences of states, actions, rewards s, a, r
  : real experience, simulated experience
- Attains optimal behaviour

How Does Monte Carlo Do This?

- Divide experience into episodes
  – all episodes must terminate
  e.g. noughts-and-crosses, card games
- Keep estimates of value functions, policies
- Change estimates/policies at end of each episode

⇒ Keep track of $s_1, a_1, r_1, s_2, a_2, r_2, \ldots s_{T-1}, a_{T-1}, r_{T-1}, s_T$
  s_T = terminating state

- Incremental episode-by-episode
  NOT step-by-step cf. DP
- Average **complete** returns – NOT partial returns

Returns

- Return at time t: $R_t = r_{t+1} + r_{t+2} + \ldots r_{T-1} + r_T$ for each episode
  r_T is a terminating state
- Average the returns over many episodes starting from some state s.

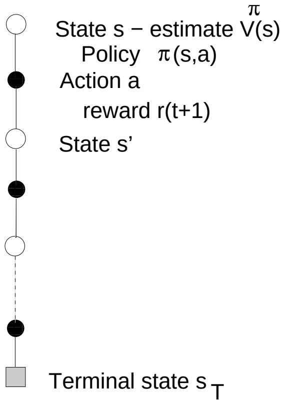
This gives the value function $V^\pi(s)$ for that state for policy π since the state value $V^\pi(s)$ is the expected cumulative future discounted reward starting in s and following policy π.

Monte Carlo Learning of V^π

MC methods estimate from experience: generate many "plays" from s, observe total reward on each play, average over many plays

1. Initialise
   - π = arbitrary policy to be evaluated
   - V = arbitrary value function
   - $Returns(s)$ an empty list, one for each state s
2. Repeat till values converge
   - Generate an episode using π
   - For each state appearing in the episode
     - R = return following first occurrence of s
     - Append R to $Returns(s)$
     - $V(s)$ = average $Returns(s)$

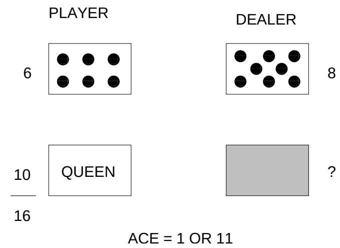
Backup Diagram for MC



One Episode – full episode needed before back-up.
cf DP which backs up after one move
Monte Carlo does **not** bootstrap but
Monte Carlo does sample

Blackjack

Sum on cards to be as close to 21 as possible



Player:

- HIT = take another card
- STICK → Dealer's turn
  or GOES BUST > 21 → loses

- Dealer's fixed strategy
  STICK if ≥ 17
  HIT if < 17

Outcome: if $> 21 \Rightarrow$ LOSE
  CLOSEST TO 21 $\Rightarrow$ WIN
  EQUALLY CLOSE $\Rightarrow$ DRAW

Blackjack: MC Episodic Task

* Reward +1, -1, 0 for win, lose, draw

* Reward within game = 0

* No discount $\Rightarrow$ Return = +1, -1, 0

* Actions: HIT, STICK

* States (sum on own cards, dealer's face-up card, usable ace): 200
  if sum on own cards < 11 no decision,
  always HIT

* Example policy π: If own sum < 20 HIT
  Else STICK

- Play many games
- Average returns (first-visit MC) following each state
- $\Rightarrow$ True state-value functions
  * Easier than DP $\Rightarrow$ That needs $P^a_{ss'}, R^a_{ss'}$
  * Easier to generate episodes than calculate probabilities

Policy Iteration (Reminder)

– Policy evaluation: Estimate V^π or Q^π for fixed policy π
– Policy improvement: Get a policy better than π

Iterate until optimal policy/value function is reached

So we can do Monte Carlo as the Policy Evaluation step of Policy Iteration because it computes the value function for a given policy. (There are other algorithms we can use.)

First-visit MC vs. Every-visit MC

In each episode observe return following **first** visit to state s

Number of first visits to s must $\rightarrow \infty$

Converges to $V^\pi(s)$

cf. Every-visit MC

Calculate V as the average over return following **every** visit to state s in a set of episodes

Good Properties of MC

Estimates of V for each state are independent
– no bootstrapping

Compute time to calculate changes (i.e. V of each state) is independent of number of states

If values of only a few states needed, generate episodes from these states $\Rightarrow$ can ignore other states

Can learn from actual/simulated experience

Don't need $P^a_{ss'}$, $R^a_{ss'}$,

Estimating Q-Values

$Q^\pi(s,a)$ – similarly to V

Update by averaging returns following first visit to that state-action pair

Problem

If π deterministic, some/many (s,a) never visited

MUST EXPLORE!

So...

* Exploring starts: start every episode at a different (s,a) pair

* Or always use ϵ-greedy or ϵ-soft policies
– stochastic, where $\pi(s,a) > 0$

Optimal Policies – Control Problem

Policy Iteration on Q

$\pi_0 \rightarrow_{PE} Q^{\pi^0} \rightarrow_{PI} \pi_1 \rightarrow_{PE} Q^{\pi^1} \rightarrow_{PI} \pi_2 \ldots \rightarrow_{PI} \pi^* \rightarrow_{PE} Q^*$

- Policy Improvement: Make π greedy w.r.t. current Q
- Policy Evaluation: As before, with ∞ episodes

Or episode-by-episode iteration. After an episode:

- policy evaluation (back-up)
- improve policy at states in episode
- eventually converges to optimal values and policy

Can use exploring starts: MCES – Monte Carlo Exploring Starts to ensure coverage of state/action space

Algorithm: see e.g. S+B Fig. 5.4

Monte Carlo: Estimating $Q^\pi(s, a)$

- If π deterministic, some (s, a) not visited $\Rightarrow$ can't improve their Q estimates

  MUST MAINTAIN EXPLORATION!

- Use exploring starts $\rightarrow$ optimal policy

- Use an ϵ-soft policy

  ON-POLICY CONTROL $\rightarrow$ ϵ-greedy policy

  OFF-POLICY CONTROL $\rightarrow$ optimal policy

On-Policy Control

Evaluate and improve the policy used to generate behaviour

Use a soft policy:

$\pi(s, a) > 0 \;\; \forall s, \forall a$ GENERAL SOFT POLICY DEFINITION

$\pi(s, a) = \frac{\epsilon}{|A|}$ if a not greedy ϵ-GREEDY

$= 1 - \epsilon + \frac{\epsilon}{|A|}$ if a greedy

$\pi(s, a) \geq \frac{\epsilon}{|A|} \;\; \forall s, \forall a$ ϵ-SOFT

POLICY ITERATION

$Evaluation$: as before $Improvement$: move towards ϵ-greedy policy (not greedy)

Avoids need for exploring starts

ϵ-greedy is "closer" to greedy than other ϵ-soft policies

Off-Policy Control

- Behaviour policy π' generates moves

- But in off-policy control we learn an Estimation policy π. How?

We need to:

- compute the weighted average of returns from behaviour policy

- the weighting factors are the probability of them being in estimation policy,

- i.e. weight each return by relative probability of being generated by π and π'

In detail...

Can You Learn π While Following π'?

We need: Estimation policy $\pi(s,a) > 0 \Rightarrow$ Behaviour policy $\pi'(s,a) > 0$

So if we want to estimate it, it MUST appear in behaviour policy

On the ith first visit to state s, let:

$p_i(s) =$ probability of getting subsequent sequence of states and actions from π (ESTIMATION)

$p'_i(s) =$ probability of getting subsequent sequence of states and actions from π' (BEHAVIOUR)

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$$p'_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$R'_i(s) =$ return observed

Then after n_s returns experienced from state s (so episodes in which s occurs):

$$V^\pi(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R'_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}}$$

$$p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$$p'_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}$$

$$\frac{p_i(s_t)}{p'_i(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

Doesn't depend on environment

Off-Policy MC Algorithm

How to use this formula to get Q-values?

- Use *Behaviour Policy* π' to generate moves
  – must be soft so that all (s,a) continue to be explored

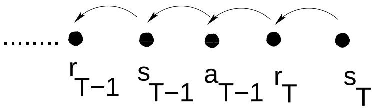
- Evaluate and improve *Estimation Policy* π
  – converges to optimal policy

So...

1. BP π' generates episode

2. EP π is deterministic and gives the greedy actions w.r.t. the current estimate of Q^π

3. Start at end of episode, work backwards



till BP and EP give divergent actions, e.g. back to time t

4. For this chain of states and actions compute

$$\frac{p_i(s_t)}{p'_i(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

i.e. we are able to find out about state s_t

π is deterministic so $\pi(s_k, a_k)$ etc. $= 1$

So

$$\frac{p_i(s_t)}{p'_i(s_t)} = \prod_{k=t}^{T_i(s)-1} \frac{1}{\pi'(s_k, a_k)}$$

5.

$$Q(s,a) = \frac{\sum \frac{p_i}{p'_i} R'}{\sum \frac{p_i}{p'_i}}$$

averaged over no. times this (s,a) has been visited, say N

$R' =$ return for the chain of states/actions (see 3) following (s,a) (it's different for each of the N visits)

6. Do for each (s,a) in chain (see 3)

7. Improve π (estimation policy) to be greedy w.r.t. Q:

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

   (Still deterministic)

Takes a long time because we can only use the information from the end of the episode in each iteration.

Summing Up

- MC methods learn V and Q from experience – sample episodes.
- Don't need to know dynamics of environment.
- Can learn from simulated experience.
- Can focus them on those parts of the state space we're interested in.
- May be less harmed by violations of Markov property, because they don't bootstrap.
- Need to maintain sufficient exploration – exploring starts or on-policy or off-policy methods.