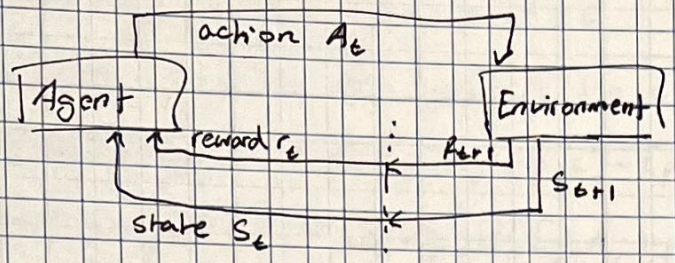
Decision Making & Reinforcement Learning
- Columbia University

Coursera

Week 1 - Decision Making & Utility Theory

Decision Making - The process in which an agent performs an action or set of actions in an environment



An agent may have sensors and actuators
- ↳ agent's actions depend on its percepts
  - ↳ may store percept sequence
- ↳ an agent function maps percept sequences to action

Agent programs implement agent functions
- ↳ depends on practicalities

Decision Making is performed so as to maximize utilities or achieve goals

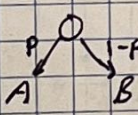
Utility Function $U: S \to \mathbb{R}$ ← state
- ↳ describe preferences or goals, rather than behaviors
- ↳ capture long-term consequences, as opposed to rewards

Principle of maximum expected Utility - a rational agent chooses actions so as to maximize expected utility, given its knowledge
- ↳ if an action a from state s yeilds states s',

expected Utility $EU(a) = \sum_{s'} Pr(s')\, U(s')$ ← utility, ← probability

↳ to maximize EU, choose $a^* = \arg\max_a EU(a)$

Utilities ultimately express an agent's preferences among different states
- ↳ a lottery is a set of possible outcomes w/ associated probabilities
  - ↳ an agent has preferences over both definite outcomes & lotteries



$L = [p, A; 1-p, B]$

Preference Notation: $A \succ B \Rightarrow$ preference

$A \sim B \Rightarrow$ indifference

$A \succsim B \Rightarrow$ either-or

Rational Preferences must satisfy certain axioms:

(a) Orderability: $A \succ B$, $B \succ A$, or $A \sim B$

(b) Transitivity: $A \succ B$ & $B \succ C \Rightarrow A \succ C$

(c) Continuity: $A \succ B \succ C \Rightarrow \exists p\, [p, A; 1-p, C] \sim B$

(d) Substitutability: $A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$

(e) Monotonicity: $A \succ B \Rightarrow (p > q$ iff $[p, A; 1-p, B] \succ [q, A; 1-q, B])$

(f) Decomposability: $[p, A; 1-p, [q, B; 1-q, C]] \sim [p, A; (1-p)q, B; (1-p)(1-q), C]$

Existence of Utilities:

Given a set of outcomes $S_1, \dots, S_n$ satisfying the preceding axioms, $\exists$ a utility function U s.t.

$$U(S_i) \geq U(S_j) \iff S_i \succeq S_j$$
$$U[p_1, S_1; \dots; p_n, S_n] = \sum_i p_i U(S_i)$$

→ U is not unique

↳ agent behaviors do not change if U is replaced w/ a positive affine transformation

$$U'(S) = aU(S) + b, \quad a > 0$$

Coming up with a Utility Function

Suppose $\exists$ a standard lottery w/ normalized utilities

$$[p, u_\top; 1-p, u_\bot]$$

$u_\top = 1$ ← best possible prize, $u_\bot = 0$ ← worst possible prize

the utility of a prize S is the value p s.t. $S \sim [p, u_\top; 1-p, u_\bot]$

Uncertain Utilities

an outcome S_1 strictly dominates an outcome S_2 if $U(S_1) > U(S_2)$

S_1 stochastically dominates S_2 if $Pr(S_1 \geq x) \geq Pr(S_2 \geq x)$ $\forall x$

↳ the CDF of S_1 is smaller than or equal to that of S_2 $\forall x$

an outcome is preferable to another if it is stochastically dominant across all attributes

↳ otherwise, may need to define multi-attribute utility function $U(x_1, \dots, x_n)$

↳ size grows exponentially

Special Case: if attributes are additive independent, we can write

$$U(x_1, \dots, x_n) = \sum_{i=1}^{n} k_i U_i(x_i)$$

uncertain attributes w/ weaker forms of independence may lead to multiplicative utility functions

Value of Information

Suppose current best action for a problem is α: $EU(\alpha) = \max_a EU(a)$

Suppose we can learn new info that may change α
- ↳ we may learn something about a random variable E w/ possible outcomes e_i
- ↳ we may change action & thus utility depending on e_i

For each value $E = e_i$, the (possibly new) best action is α_i: $EU(\alpha_i | e_i) = \max_a EU(a | e_i)$

⇒ Value of Perfect Information is the <u>expected</u> improvement in expected utility:

$$VPI(E) = \left(\sum_e Pr(E=e) EU(\alpha_e | e) \right) - EU(\alpha)$$

Thm: VPI is non-negative; it is never disadvantageous to acquire more information (assuming no acquisition cost)

Maximization of VPI can be used to design an information gathering agent

Week 2 - Bandit Problems

Tradeoff between exploration & exploitation

Given analogy of n slot machines w/ unknown reward distribution,
- consider choosing a slot machine to be action a
  - A_t is action chosen & R_t is ~~reward~~ resultant reward @ time t

Action value is the expected reward of an action

$$Q^*(a) = E[R_t \mid A_t = a]$$

- build estimates of Q^* by trying different actions and recording results
  - One Method: first initialize all $Q_0(a)$, e.g. by trying each action once
    - Sample-averaging: $Q_t(a) = \dfrac{\text{sum of rewards from taking } a \text{ prior to } t}{\#\text{ of times taking } a \text{ prior to } t}$
    - More space efficient approach: $O(N) \to O(1)$

      Suppose R_t is the nth reward from a,

$$Q_{t+1}(a) = \frac{1}{n}\left((n-1)Q_t(a) + R_t\right) = Q_t(a) + \frac{1}{n}\left(R_t - Q_t(a)\right)$$

As more rewards are seen, individual reward contributions shrink
- smaller step size $\left(\frac{1}{n}\right)$

For nonstationary ~~ro~~ problems, weight more recent data more
- some constant step size α

$$Q_{t+1}(a) = Q_t(a) + \alpha(R_t - Q_t(a))$$
$$= (1-\alpha)^{t-1} R_1 + \alpha \sum_{i=2}^{t} (1-\alpha)^{t-i} R_i$$

- causes contribution of older data to decay exponentially

Deliberate initialization of $Q_0(a)$ can provide bias to the estimates
- optimistic - set all $Q_0 \gg 0$
  - agent is encouraged to explore

How to perform action selection while learning Q?

Given Q^*, the best strategy would be greedy action selection

$$A_t = \arg\max_a Q^*(a)$$

- ε-greedy action selection
  - select greedy action $1-\varepsilon$ of the time, but pick random action to explore instead ε of the time

As $t \to \infty$ following an ε-greedy strategy ensures that Q_t values converge near Q^* values

Choosing ε
- Small ε: Q_t values end up closer to Q^*, but convergence may be slow
- Large ε: Q_t values are less accurate, but more values are updated initially
- Variations may vary the value of ε
  - i.e. ε-first
    - set $\varepsilon = 1$ for a fixed # of trials, then set $\varepsilon = 0$ afterward
  - ε-decreasing
    - set ε to be a high initial value, then decrease over time

☆ While exploration is necessary to learn action values, it is not optimal for maximizing rewards recieved
- targeted exploration: focus on more "promising" actions

Upper Confidence Bound
- combines both exploration & exploitation

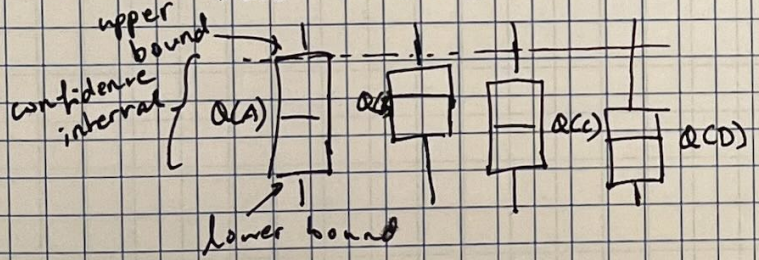
$$UCB(a) = Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}$$

$N_t(a)$ - # of times a taken prior to t

c - exploration parameter ≥ 0

- for proper initialization, we can take each possible action once s.t. $N_t(a) = 1 \ \forall a$
- subsequent action selection then follows $A_t = \arg\max_a UCB(a)$
- can be seen as a "boosted" action value
  - exploration is more frequent when N_t is small and tapers out as N_t increases
  - $\ln(t)$ term slowly increases, so that exploration is never ruled out

UCB Visualization



UCB Implementation:

given set of actions A, learning rate α, exploration parameter c

For each $a \in A$, take a and assign associated reward to $Q(a)$

Initialize $N(a) = 1 \ \forall a \in A$

Loop:

Take action $a = \arg\max UCB(a')$ and observe reward r_a

Update $Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$ and $N(a) \leftarrow N(a) + 1$

Update α and/or c as desired

ε-greedy Implementation

given: set of actions A, exploration probability ε

Initialize $Q(a) = 0$ (or any other desired value) $\forall\, a \in A$

Loop:

- determine a probabailistically using ε
- Take action a & observe reward r
- Update $Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$
- Update α and/or ε as desired

Regret - expected decrease in cummulative rewards due to optimal (clairvoyant) strategy vs actual strategy taken

↳ one measure of regret ~~is~~ comes from the "error" in the action values $|Q^*(a) - Q(a)|$

For UCB, it can be shown that $Pr\left[|Q^*(a) - Q(a)| > c\sqrt{\frac{\ln t}{N_t(a)}}\right] \leq t^{-2c^2}$

↳ $c\sqrt{\frac{\ln t}{N_t(a)}}$ acts as a "baseline" for the action value error

↳ baseline decreases as N increases and increases (very slowly) as t increases

↳ likelihood that error > baseline decreases as t increases

Week 3 - Markov Decision Processes

MDP is a mathematical model for a sequential decision making problem w/ uncertainty
↳ can be used to help quantify optimal decision making

State space S & action spaces $A(s)$ for each state

Transition Function $T: S \times A \times S \rightarrow [0, 1]$, where $T(s, a, s') = Pr(s' | s, a)$

Reward Function $R: S \times A \times S \rightarrow \mathbb{R}$; $R(s, a, s')$
↳ captures uncertainty and stochasticity

$$\sum_{s'} T(s, a, s') = 1$$

Markov Property - transitions depend on a finitely many prev. states
↳ extends to decision making

(s, a, s') is collectively refferred to as a ~~triplet~~ transition
↳ (starting state, action taken, successor state)

Terminal state - no action from state

Living reward - small, negative value to encourage agent to reach goal

Finite Horizon MDP - process ends after some finite time T (terminal state s_T)

One definition of a state-action sequence:

$$\gamma = 1 \rightarrow V([s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T]) = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1})$$

To account for timing, apply a discount factor $0 < \gamma < 1$ to diminish future rewards

$$V([s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T]) = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1}) \gamma^t$$

For infinitely many transitions, we must use a discount factor $\gamma < 1$ so that rewards do not become unbounded

Choice of γ determines how myopic or forward-looking the agent is

$$V([s_0, a_0, s_1, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \frac{R_{max}}{1-\gamma}$$

A policy function $\pi: S \rightarrow A$ tells the agent what to do in any given state

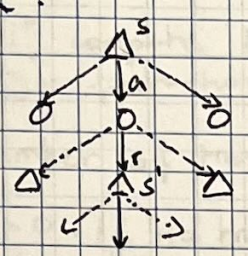
We can quantify policies using value functions

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right], \quad s_0 = s$$

We are generally interested in the <u>optimal</u> policy & value functions

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V^{\pi} \qquad V^* = \max_{\pi} V^{\pi}$$

Backup Diagram:



△ - state
→ - action
o - decision

Fixed action @ each state due to π
↳ expected value entails summing over values of successors

$$\Rightarrow V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right]$$

↳ set of $|S|$ linear equations in the $|S|$ unknowns $V^{\pi}(s)$
↳ Linear solvers can solve in $O(|S|^3)$ time

Bellman Optimality Equations

$$\begin{cases} V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \\ \pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right] \end{cases}$$

B-O Eqns are nonlinear
↳ cannot solve system of eqns. to find optimal policy

Assuming we can solve for V^*, it is feasible to find π^* using a brute force method

Week 4 - Dynamic Programming

Problem - V^* is a function of infinitely many successor state values

- ↳ Idea #1 - Approximate a discounted infinite sequence with a infficiently long discounted finite sequence
- ↳ Idea #2 - The utility of a sequence w/ $i+1$ rewards can be computed if we have the utility of its subsequence w/ i rewards

To solve for V^*, compute the time-limited values of successively larger finite-horizon MDPs

Let i be the number of transitions taken starting from a state s

Base Case: $i=0$. If no transitions allowed, $V_0(s) = 0 \ \forall s$
↳ No actions taken, no rewards recieved

$i=1$: $V_1(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_0(s')]$

$\vdots$

Bellman Update Equation:

$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

as $i \to \infty$, $V_i \to V^*$

Value Iteration - a dynamic programming approach to solving for V^*

- ↳ compute time-limited values V_i for all states, then use them to
- ↳ continue for successively increasing i until convergence - compute V_{i+1}

E.g.

Initialize $V_0(s) \leftarrow 0 \ \forall s \in S$

Loop from $i=0$

$\forall s \in S$

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')]$$

Until $\max_s |V_{i+1}(s) - V_i(s)| < \varepsilon$ (small threshold)

Convergence of Value Iteration

Value Iteration is guaranteed to converge & return V^*, or an approximation within ε

The Bellman Update is a contraction mapping

Fact (1) - Bellman Update does not change optimal values V^* (fixed point)

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Define max norm (error) between V_i and true values V^*

$$\|V_i - V^*\| = \max_s |V_i(s) - V^*(s)|$$

Fact (2)

max norm satisfies $\|V_{i+1} - V^*\| \leq \gamma \|V_i - V^*\|$

$\Rightarrow$ after k iterations, max error is bounded by

$$\gamma^k \frac{|r_{max}|}{1-\gamma}$$

Given V^*, we can solve for π^* in $O(|S|^2|A|)$

↳ only relative (not absolute) V^* required

rare for $|S|^3$ to be dominating term

Policy Iteration $O(|S|^3 + |S|^2|A|)$

↳ a policy can be computed @ any point during value iteration

↳ we can improve on policy directly → better values → better policy → ...

Initialize $\pi_0(s)$ arbitrarily $\forall s \in S$

Loop from $i=0$:

Compute V^{π_i} for policy π_i (policy evaluation)

Given V^{π_i}, find new policy π_{i+1} (policy improvement)

Until $\pi_{i+1} = \pi_i$

Iterative Policy Evaluation: (iterative approach to solve for V^π)

Initialize V_0^π (eg. all equal to 0, or values from a similar policy)

Loop from $i=0$:

$\forall s \in S$

$$V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^\pi(s')]$$

Until $\max_s |V_{i+1}^\pi(s) - V_i^\pi(s)| < \varepsilon$ (small threshold)

Policy Improvement

Take the "greediest" action at each state

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

If π_i is already optimal, $V^{\pi_i} = V^*$ and $\pi_{i+1} = \pi_i$
Otherwise, V^{π_i} can be moved closer to V^* by changing some actions

Policy Improvement Thm for proof of convergence

Week 5 - Partially Observable MDPs

MDP formalism assumes full observability of the environment

Many real-world problems only have partial observability
- ↳ agent does not know its true state
- ↳ a degree of belief of being in different states

Partially Observable MDP (POMDP)

States S, actions A
Transitions $T: S \times A \times S \to [0,1]$; rewards $R: S \times A \times S \to \mathbb{R}$
Set of possible observations or evidence Z
Sensor model $O: S \times Z \to [0,1]$; $O(s, z) = Pr(z|s)$

Policies that map from states to actions are no longer sufficient
- ↳ define a belief state $b: S \to [0,1]$ as a probability distribution
  - ↳ $b(s)$ is probability that agent is in state s
  - ↳ continuous state space - infinitely many distributions

Belief State Update

How does b change, given action a & observation z?

Take action a:

$$Pr(s'|b,a) = \sum_s Pr(s'|s,a)\, b(s)$$

Make observation z: (apply Bayes' rule using sensor model)

$$Pr(s'|b,a,z) \propto Pr(z|s')\, Pr(s'|b,a)$$
$$= Pr(z|s') \sum_s Pr(s'|s,a)\, b(s)$$

Normalize ($\sum_s b(s) = 1$)

For every POMDP $\exists$ an equivalent observable MDP
- ↳ state space is simply the (continuous) set of belief states
- ↳ Transition probability $Pr(b'|b,a)$ = probability of observing z

$$Pr(z|b,a) = \sum_{s'} Pr(z|s')\, Pr(s'|b,a) = \sum_{s'} Pr(z|s') \sum_s Pr(s'|s,a)\, b(s)$$

↳ sum of "unnormalized" likelihoods of b'

$$R(b,a,b') = \sum_s b(s) \sum_{s'} Pr(s'|s,a)\, R(s,a,s')$$

A policy $\pi: S \to A$ for a belief MDP maps from belief states to actions

$$V^\pi(b) = \sum_{b'} T(b, \pi(b), b')\left[R(b, \pi(b), b') + \gamma V^\pi(b')\right]$$

Bellman Equations for optimal value function & policy

$$\begin{cases} V^*(b) = \max_a \sum_{b'} T(b,a,b')\left[R(b,a,b') + \gamma V^*(b')\right] \\ \pi^*(b) = \arg\max_a \sum_{b'} T(b,a,b')\left[R(b,a,b') + \gamma V^*(b')\right] \end{cases}$$

Theoretically value iteration still works for finding optimal values V^*

$$V_{i+1}(b) \leftarrow \max_a \sum_{b'} T(b,a,b')[R(b,a,b') + \gamma V_i(b')]$$

↳ potentially infinitely many belief states

Finding optimal policies for POMDPs is computationally intractable

Planning online using look-ahead search

- ↳ make decisions while interacting w/ environment
- ↳ look ahead a finite # of steps from current belief state
  - ↳ solving a finite horizon MDP
- ↳ use estimates of state utilities instead of exact values

~~Suppose~~ d-step lookahead: $O(|A|^d |Z|^d)$ time to compute all successors

- ↳ small d works for small γ
- ↳ sampling methods may be used to cut down on branching factor

Tricks for improving performance of look-ahead search

Reuse Info:

- use memoization (identical subtrees may appear)
- value approximations may be computed offline (e.g. exact MDP)

Search Wisely:

- Tree branches can be pruned based on lower/upper bound info
- Orderings of expanded nodes can be informed by heuristics
- Estimate evidence probabilities by sampling

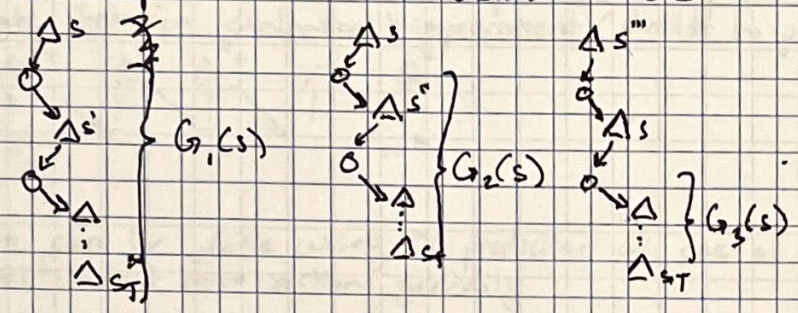
Week 6 - Monte-Carlo Methods

Monte Carlo Methods - generate sampled experience and average them for different states and actions

$$V^{\pi}(s) = E\left[\sum_{t=0} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

↳ approximate the expectation by taking averages of sample reward sequences over multiple episodes

Each sample can be thought of as a path from a given node to a leaf node in a search tree



$$\ldots V(s) = \frac{1}{N}\sum_{i=1}^{N} G_i(s)$$

Monte-Carlo: Estimate state values by averaging utilities over multiple episodes

First-visit MC - a value is estimated after first visit to state within episode

Initialize $V^{\pi}(s) \leftarrow 0$, $Count(s) \leftarrow 0 \quad \forall s \in S$

Loop:

Generate episode E following π: $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$

For first occurrence of each state $s_t \in E$:

$G \leftarrow \sum_{j=0}^{T-t-1} \gamma^j r_{j+t+1}$

$V^{\pi}(s_t) \leftarrow (Count(s_t) \times V^{\pi}(s_t) + G) / (Count(s_t) + 1)$

$Count(s_t) \leftarrow Count(s_t) + 1$

Estimates of different state values are independent

↳ accuracy of $V^{\pi}(s)$ does not depend on accuracy of $V^{\pi}(s')$

↳ contrast to DP approach

⇒ Computational complexity of estimating specific state values is independent of $|S|$

MC is particularly effective if we want to focus on certain states & ignore others

w/o T or R, cannot compute $\pi(s) = \arg\max_a \underbrace{\sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]}_{Q(s,a)}$

Instead, learn state-action values at the chance nodes

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

$Q(s,a)$ is the expected utility of s after ~~following~~ taking action a and following π thereafter

$$Q(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$
$$= \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$

In RL, first learn Q values, then use these to compute V^* or π^*

Monte Carlo Control

Goal: Learn optimal (or close to optimal) policy via MC RL

Alternate between evaluation & improvement (similar to policy iteration)

$$\begin{tikzcd}
Q^{\pi_0} \arrow[dr] & Q^{\pi_1} \arrow[dr] & Q^{\pi_2} \arrow[dr] & \cdots \arrow[dr] & Q^{*} \\
\pi_0 \arrow[u] & \pi_1 \arrow[u] & \pi_2 \arrow[u] & \cdots \arrow[u] & \pi^* \arrow[u]
\end{tikzcd}$$

Evaluation can be done using MC prediction w/ one or more episodes

Improvement: select new actions greedily

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

Could (should) incorporate explorations into MC samples

ε-greedy policy

Stochastic Policy: $\pi(a|s) = \begin{cases} 1-\varepsilon & \text{for } a = \arg\max_{a'} Q(s,a') \\ \frac{\varepsilon}{|A(s)|+1} & \text{for all other actions } a \end{cases}$

ε-greedy MC control

Initialize $Q^{\pi}(s,a) \leftarrow 0$, $\text{Count}(s,a) \leftarrow 0$, $\forall s \in S$, $\forall a \in A$

Initialize $\varepsilon \leftarrow$ some small number, $\pi \leftarrow$ an ε-greedy policy

Loop:

Generate new episode E following π: $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T$

For first occurrence of each pair $(s_t, a_t) \in E$:

$G \leftarrow \sum_{j=0}^{T-t-1} \gamma^j r_{j+t+1}$

$Q^{\pi}(s_t, a_t) \leftarrow (\text{Count}(s_t, a_t) \times Q^{\pi}(s_t, a_t) + G) / (\text{Count}(s_t, a_t) + 1)$

$\text{Count}(s_t, a_t) \leftarrow \text{Count}(s_t, a_t) + 1$

Problem: ε-greedy MC control doesn't actually give Q & π^*

↳ Since we have $\varepsilon > 0$ & occassionally converge while learning, the state-action values & policy converge <u>under that condition</u>

On-Policy Learning - learn the values for the policy that we are following, or the behavior policy

If we wanted the optimal policy, we would have to decrease ε over time

Off-Policy Learning - use the behavior policy b to learn about a different target policy π
- ↳ more general than on-policy, but slower to converge
- ↳ off-policy prediction - estimate values for π using episodes from b

We require that all state-actions allowed by π are also allowed by b

$$\pi(a_j|s_j) > 0 \Rightarrow b(a_j|s_j) > 0$$

Suppose we observe an episode from b: $s_t, a_t, s_{t+1}, a_{t+1}, \dots, s_{T-1}, a_{T-1}$

Importance Sampling Ratio: relative likelihood of subset of states and actions under different policy π

$$w_t = \prod_{j=t}^{T-1} \frac{\pi(a_j|s_j)}{b(a_j|s_j)}$$

↳ use this weight the state-action pair under b into one as if generated by π

$$Q(s_t, a_t) = E[w_{t+1} G_t]$$

Off-Policy MC Control with Weighted Importance Sampling

Initialize $Q^{\pi}(s,a) \leftarrow 0$, $W(s,a) \leftarrow 0$, $\forall s \in S$, $\forall a \in A$
Initialize $b \leftarrow \varepsilon$-greedy behavior policy, $\pi \leftarrow$ greedy policy
Loop:
Generate new episode E following b: $s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T$
For first occurrence of each pair $(s_t, a_t) \in E$:
$G \leftarrow \sum_{j=0}^{T-t-1} \gamma^j r_{j+t+1}$, $w \leftarrow \prod_{j=t+1}^{T-1} \pi(a_j|s_j) / b(a_j|s_j)$
$Q^{\pi}(s_t, a_t) \leftarrow (W(s_t, a_t) \times Q^{\pi}(s_t, a_t) + w \times G) / (W(s_t, a_t) + w)$
$W(s_t, a_t) \leftarrow W(s_t, a_t) + w$
$\pi(s_t) \leftarrow \arg\max_a Q^{\pi}(s, a)$

Week 7 - Temporal-Difference Learning

State Value Update Expression

$$V^\pi(s) \leftarrow \frac{N(s)V^\pi(s) + G_i(s)}{N(s)+1} = V^\pi(s) + \frac{1}{N(s)+1}\left(G_i(s) - V^\pi(s)\right)$$

May choose to give more weight to recent returns

↳ Constant α MC exponentially decays weights on past returns by learning rate α

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha\left(G_i(s) - V^\pi(s)\right)$$

MC requires episodic structure - what about infinite horizon problems?

↳ One-step TD (TD(0)) - replace "target" term ($G_i(s)$) with the sum of immediate reward w/ discounted successor state value

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha\overbrace{\left(\underbrace{r_{t+1} + \gamma V^\pi(s')}_{\text{target}} - V^\pi(s)\right)}^{\text{TD error } \delta_t}$$

$$= V^\pi(s_t) = V^\pi(s_t) + \alpha\,\delta_t$$

TD(0) for Prediction

Given policy π, $0 < \alpha < 1$
Initialize $V^\pi(s) \leftarrow 0$
Loop:
  Initialize starting state s if needed
  Generate sequence $(s, \pi(s), r, s')$
  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s))$
  $s \leftarrow s'$

TD methods perform updates immediately with no episodic structure
↳ useful if problems are continuing tasks or have long episodes

For sufficiently small α, average values of V^π converge to true values
If α is constant, V^π is prone to fluctuation even near convergence
↳ in practice, decrease α to 0 over time

May also use constant-α MC & TD(0) as offline algorithms

Suppose we have a fixed set of samples in place of underlying model
We can run TD(0) on the samples until values converge

We can perform batch updating - making a single TD update based on the sum of all sample TD errors in sample set

Batch Updating

Given: training data $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$; learning rate α
Initialize $V^{\pi}(s) \leftarrow 0$
Loop until values converge:
  $\delta(s) \leftarrow 0 \ \forall s$
  For each state s_t in training data:
    $\delta(s_t) \leftarrow \delta(s_t) + (r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$
  For each state s:
    $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \delta(s)$

TD learning for control

↳ apply TD updates to Q values rather than state values

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

Which successor state Q value $Q(s',a')$ to use?
↳ consider on-policy (SARSA) vs off-policy learning (Q-learning)

SARSA

Given: step size α, exploration rate ε
Initialize $Q(s,a) \leftarrow 0$, behavior policy π (e.g. ε-greedy)
Loop:
  Initialize starting state s, action $a = \pi(s)$ if needed
  Generate sequence (s, a, r, s'), $a' \leftarrow \pi(s')$
  $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
  $s \leftarrow s'$, $a \leftarrow a'$

Q-Learning

Given: step size α, exploration rate ε
Initialize $Q(s,a) \leftarrow 0$, behavior policy π (e.g. ε-greedy)
Loop:
  Initialize starting state s if needed, action $a = \pi(s)$
  Generate sequence (s, a, r, s')
  $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$ ← action a' selected according to target policy
  $s \leftarrow s'$

SARSA vs. Q-Learning

SARSA	Q-Learning
- agents are more "cautious" as they worry about low rewards from exploration	- Agents are more "optimistic" as only the best (greedy) actions at each state matter

SARSA & Q-Learning are identical if behavior and target policy are the same, or if exploration is removed

Week 8 - Generalization Methods in RL

Monte-Carlo estimates a state value using rewards of an entire episode

Temporal Difference computes an update to a state value using the reward of a single transition

n-step TD - target value consists of n steps of rewards followed by successor state value

$$G_{t:t+n} = \sum_{j=0}^{n-1} \gamma^j r_{j+t+1} + \gamma^n V_{t+n-1}(s_{t+n})$$

$$V_{t+n}(s_t) \leftarrow V_{t+n-1}(s_t) + \alpha(G_{t:t+n} - V_{t+n-1}(s_t))$$

n-step TD still bootstraps, but uses n "true" rewards, while using a state value estimate $V(S_{t+n})$ to approximate the rest

Advantage: Expectation of using true rewards is guaranteed to be better for learning V^π than the corresponding value estimate

$$\max_{s_t} |E(G_{t:t+n}) - V^\pi(s_t)| \leq \gamma^n \max_{s_t} |V_{t+n-1}(s_t) - V^\pi(s_t)|$$

→ max error of expected n-step return is upper bounded by discounted max error of estimate V_{t+n-1}

n-step SARSA

$$G_{t:t+n} = \sum_{j=0}^{n-1} \gamma^j r_{j+t+1} + \gamma^n Q_{t+n-1}(s_{t+n}, a_{t+n})$$

$$Q_{t+n}(s,a) \leftarrow Q_{t+n-1}(s_t, a_t) + \alpha(G_{t:t+n} - Q_{t+n-1}(s_t, a_t))$$

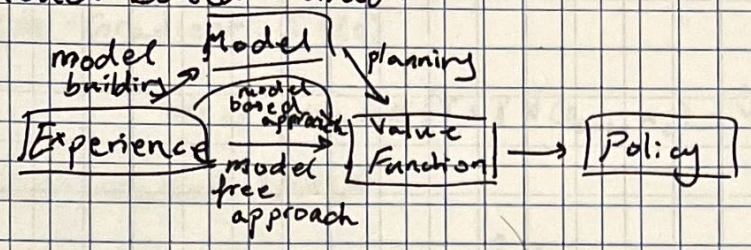
Off-Policy Learning

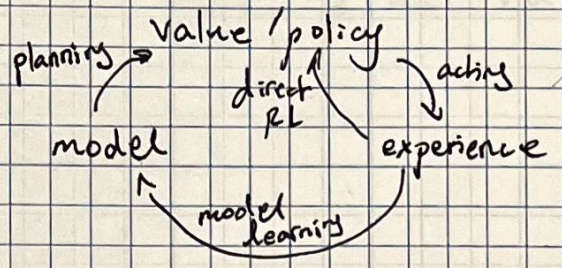
$$V_{t+n}(s_t) \leftarrow V_{t+n-1}(s_t) + \alpha \rho_{t:t+n-1}(G_{t:t+n} - V_{t+n-1}(s_t))$$

$\rho_{t:t+n-1}$ is the likelihood that subsequent n actions took place after s_t

$$Q_{t+n}(s_t, a_t) \leftarrow Q_{t+n-1}(s_t, a_t) + \alpha \rho_{t+1:t+n}(G_{t:t+n} - Q_{t+n-1}(s_t, a_t))$$

Model Based Methods





Dyna-Q algorithm

Initialize $Q(s,a)$ and $Model(s,a)$ $\forall s,a$
Loop:
- Initialize s if needed, action a using behavior policy
- Generate sequence (s, a, r, s')
- Perform Q-learning update on $Q(s,a)$
- Perform model update: $Model(s,a) \leftarrow r, s'$
- Loop as desired:
  - $\sigma, a \leftarrow$ previously seen state-action pair
  - Generate $(\sigma, a, \rho, \sigma')$ using $Model(\sigma, a)$
  - Perform Q-learning update on $Q(\sigma, a)$

Function Approximation

↳ V & Q may not be represented in some tabular format
↳ instead of learning $V^{\pi}(s)$, learn $\hat{V}(s, w)$ by learning a set of weights w
  ↳ objective is to minimize an error function

Stochastic Gradient Descent

$$w_{t+1} = w_t + \alpha(v_t - \hat{V}(s_t, w_t)) \nabla \hat{V}(s_t, w_t) \quad \text{for} \quad 0 < \alpha < 1$$

Gradient Monte Carlo

↳ episode return G_t is an unbiased estimate of $V^{\pi}(s_t)$

$$w_{t+1} = w_t + \alpha(G_t - \hat{V}(s_t, w_t)) \nabla \hat{V}(s_t, w_t)$$

Semi-Gradient TD(0)

$$w_{t+1} = w_t + \alpha(r + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)) \nabla \hat{v}(s_t, w_t)$$

Semi-Gradient SARSA

$$w_{t+1} = w_t + \alpha(r + \gamma \hat{Q}(s_{t+1}, a_{t+1}, w_t) - Q(s_t, a_t, w_t)) \nabla \hat{Q}(s_t, a_t, w_t)$$