Artificial Intelligence

Alex Day

September 24, 2020

Contents

1 AI Basics **2**
1.1 Agents . . . 2
1.2 Searching . . . 3
1.3 State Space Graphs vs Search Trees . . . 3

2 Uninformed Search **4**
2.1 Breadth First Search . . . 4
2.2 Depth First Search . . . 4
2.3 Iterative Deepening . . . 4
2.4 Uniform Cost Search . . . 4
2.5 Search Algorithm Evaluation . . . 5
2.5.1 BFS Properties . . . 5
2.5.2 DFS Properties . . . 6
2.5.3 UCS properties . . . 6

3 Informed Search **6**
3.1 Search Heuristic . . . 6
3.2 Greedy Search . . . 7
3.3 A* . . . 7

4 Probabilities **8**

5 Markov Decision Process **8**
5.1 Stochasticity . . . 8
5.2 Gridworld . . . 8
5.3 Stochastic motion model . . . 9
5.4 Simple Game . . . 10
5.5 Markov Decision Process . . . 10

5.6 Policy . . . 11
5.7 Discounting . . . 12
5.8 Avoiding Infinite Rewards . . . 12
5.9 Revisitng MDPs . . . 12
5.10 Solving MDPs . . . 13
5.10.1 Value Iteration . . . 13
5.10.2 Policy Iteration . . . 14

6 Reinforcement Learning **15**
6.1 Definition . . . 15
6.2 Framework . . . 15
6.3 Model-Based Learning . . . 16
6.3.1 Model-Based Monte Carlo . . . 16
6.4 Model-Free Learning . . . 16
6.4.1 Example . . . 17
6.5 Q-Learning . . . 17
6.5.1 Active RL . . . 17
6.5.2 Tabular Q-Learning . . . 17
6.5.3 Properties . . . 17
6.5.4 Exploration vs Exploitation . . . 18

7 Adversarial Games **18**
7.1 Types of Games . . . 18
7.2 Deterministic Games . . . 19
7.3 Why are games hard? . . . 19
7.4 Zero Sum Games . . . 19
7.5 General Games . . . 19

1 AI Basics

1.1 Agents

- An **agent** is an entity that perceives and acts
- A **rational agent** selects actions that achieve the best (expected) outcome
- **Reflex agents** consider how the world is but do **not** consider future consequences of their actions
  - Can sometimes be rational, although not always

- **Planning agents** consider how the world would be based upon their actions and have some goal
  - Decisions are based on hypothesized consequences of actions
  - Not always the **best** action so they're note always rational

1.2 Searching

- In a Discrete Search Problem we are given:
  - A finite state space
  - A finite action space
  - A cost function
    - $Cost = C(Action, State, FutureState)$
    - The cost of an action is defined as the cost of moving from a state to some future state through that action
  - A transition model
    - $FutureState = Transition(Action, CurrentState)$
  - Start state and a goal state or goal test
  - We seek to find a minimum cost solution solution: a sequence of actions that lead from the start to the goal
  - We assume the cost of the solution is equal to the sum of the cost of each step

1.3 State Space Graphs vs Search Trees

- State Space Graphs
  - The state space forms a directed graph where the nodes are states and the edges are actions
  - Each state occurs only once
  - Goal test is a set of nodes
  - Rarely can build it in memory
- Search Trees
  - Root has the start state
  - Branches are actions

- The nodes show states but correspond to local PLANS
- Search trees can be expanded until the solution is found
  * Leaf nodes are called the frontier or the open list
  * Leaf nodes are nodes that have unexplored options

2 Uninformed Search

2.1 Breadth First Search

- Expand shallowest node first
- Frontier is a FIFO queue

2.2 Depth First Search

- Always expand the deepest node first
- Frontier is a LIFO queue (stack)

2.3 Iterative Deepening

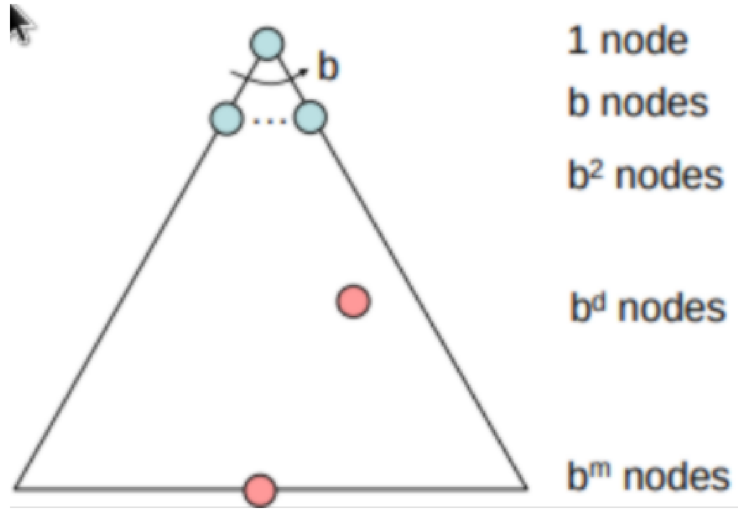
- Run DFS with depth limit 1
- Run DFS with depth limit 2
- Run DFS with depth limit ...
- DFS space complexity with BFS time complexity

2.4 Uniform Cost Search

- Expand least-cost node first
- Frontier is a priority queue
- Issues
  - Explores in all directions
  - No goal-oriented expansion

2.5 Search Algorithm Evaluation

- Completeness - Does this always find a solution if one exists
- Optimal - Find the least cost solution
- Time complexity - Time taken
- Space complexity - Space needed
- Useful quantities
  - b - branching factor of the tree (average number of successors for any node)
  - m - Maximum depth of the state space (meaning there are at max b^m nodes)
  - d - Depth of the shallowest goal node



2.5.1 BFS Properties

- Time complexity $\approx O(b^d)$
- Space complexity $\approx O(b^d)$
- It is complete (if d is finite)
- It is optimal only if step costs are equal or increasing as we move down the tree
- BFS requires a crazy amount of memory and time

2.5.2 DFS Properties

- It is complete if m is finite and the graph is acyclic
- Not optimal
- Time complexity $O(b^m)$ if $m \neq \infty$ and terrible if $m >> d$
- Space complexity $O(bm)$

2.5.3 UCS properties

- It is optimal
- It is complete if the cost of every action is at least $\epsilon > 0$
- Time
  - If C^* is the optimal cost the effective depth is $\frac{C^*}{\epsilon}$
  - It takes $O(b^{\frac{C^*}{\epsilon}})$ time and space

3 Informed Search

- **Informed Search Methods** use problem specific knowledge to solve a problem better
- **Idea**: Use an *evaluation* function $f(n)$ for each node n
  - Estimate "desirability" of each node
- Open is a priority queue sorted by increasing f-cost

3.1 Search Heuristic

- A heuristic function $h(n)$
  - Estimates how close the state at node n is to the goal state
  - Designed for a particular search problem
  - Common heuristics: Manhattan distance, Euclidean distance, etc.

3.2 Greedy Search

- Expand the node that appears to be closest to the goal at each step
- $f(n) = h(n)$
- Complete
- Not Optimal
- Time - $O(b^m)$
- Space - $O(b^m)$

3.3 A*

- Guide the search while avoid expanding expensive paths
- Evaluation function $f(n) = g(n) + h(n)$
- Admissible heuristics
  - Never overestimate true cost of the goal
- Consistent heuristics
  - $h(n) <= c(n, a, n') + h(n')$
  - Where c is a step cost function
  - All consistent heuristics are admissible
- Most of the work in A* lies on finding admissible heuristics
  - We can often find these by solving a *relaxed* version of the problem
    * The **key** idea is the optimal solution cost of the relaxed problem is no greater than the optimal solution cost of the real problem
- Given two heuristics h_1 and h_2 if $h_2(n) >= h_1(n) \forall n$ then h_2 **dominates** h_1 and is better for search
- Given m admissible heuristics $h_1, h_2, \ldots, h_3$ then $h(n) = max(h_1(n), h_2(n), \ldots, h_n(n))$ is also admissable and dominates any h_i
- A* has extensions that allow incremental, anytime, and pruning approahces

4 Probabilities

- A random variable X, represents an event whose outcome is unknown
- P is a probability distribution that assigns weight to outcomes
  - X = weather tomorrow
  - $X \in \{sunshine, rain, thunder\}$
  - $P(X = sunshine) = 0.5$, $P(X = rain) = 0.25$, $P(X = thunder) = 0.25$
- Probabilities are non-neg and sum to one
- As we get more evidence the probabilities may change
- The **expected value**, E, of a function of a random variable is the average, weighted by the probability distribution over outcomes

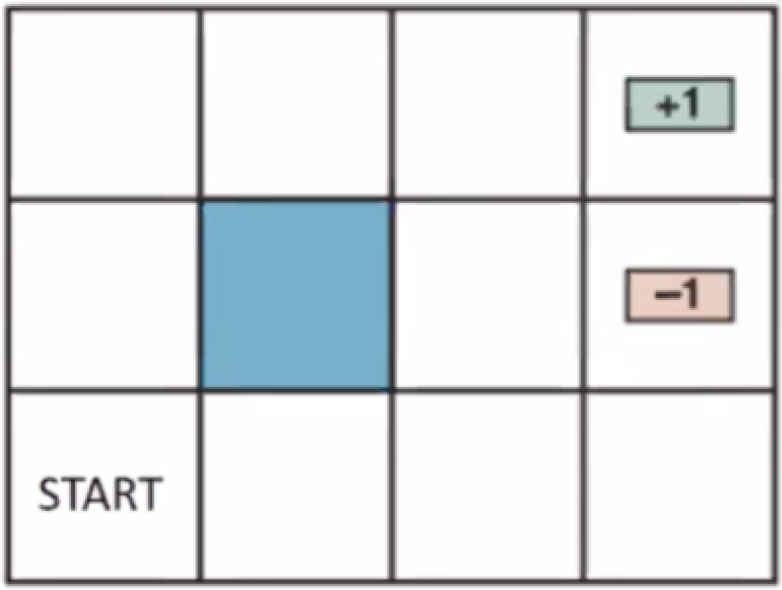
5 Markov Decision Process

5.1 Stochasticity

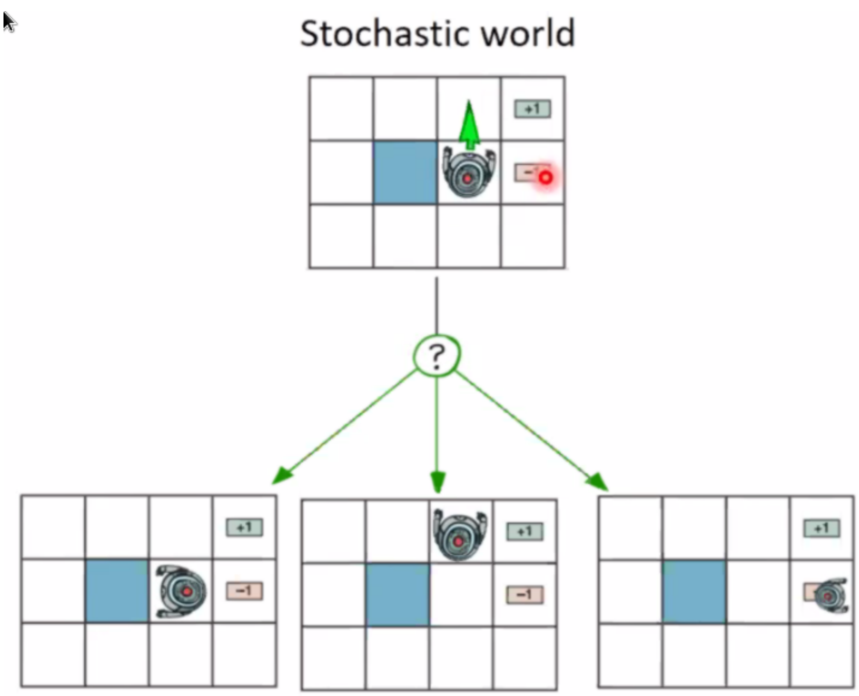
- Sometimes you cannot rely that a given action from a specific state may always take you to a certain state
- How can we act optimally in the face of randomness

5.2 Gridworld

- Noisy motion model
  - 80% the action N takes the agent North (if there is no wall)
  - 10% N goes West, 10% N goes east
  - If there is a wall the agent stays put
- The agent receives rewards at each step and a big reward if it exits at +1 and a bad reward if it exits at -1



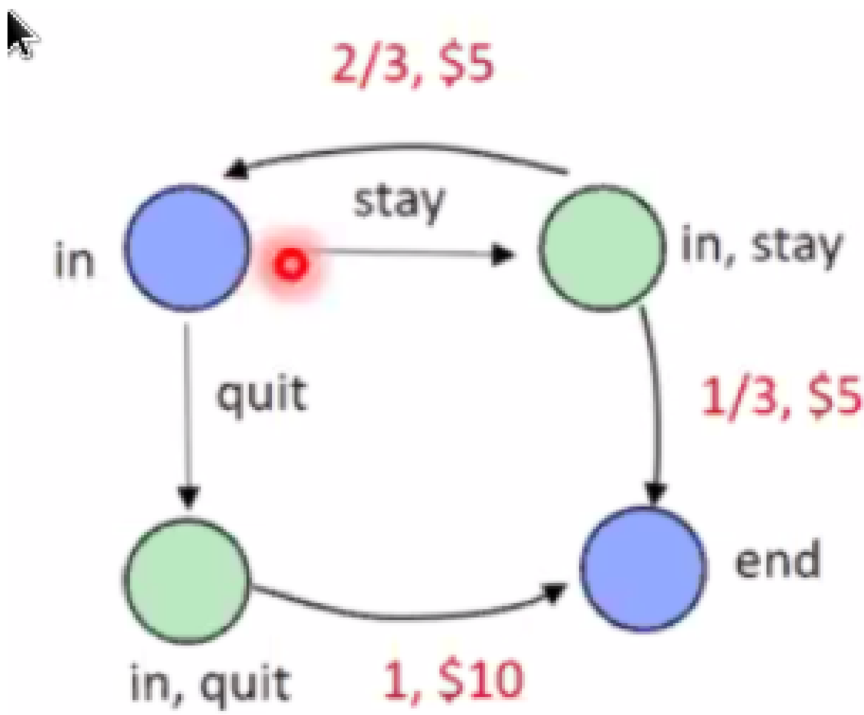
5.3 Stochastic motion model



5.4 Simple Game

- At each round:

    1. Stay or quit
    2. If quit: you get \$10
    3. If stay: you get \$5 and then roll a die
        (a) If the result is 1 or 2 the game ended
        (b) Otherwise the game continues to the next round

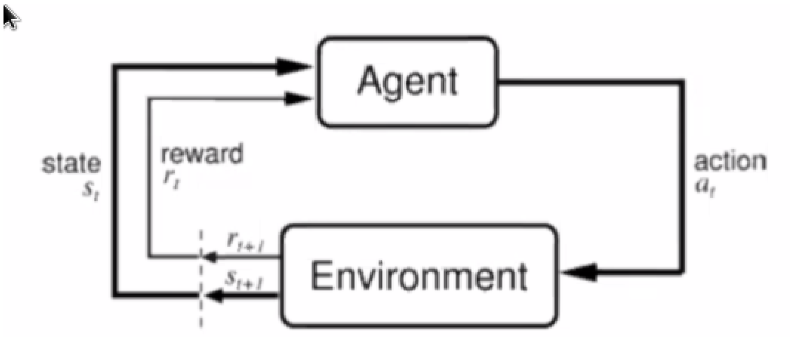


In a Markov Decision Process there are state nodes (in blue), chance nodes (in green), choice edges (in black text), and reward edges with the probability and reward (in red)

5.5 Markov Decision Process

- A set of states S
- A set of actions A

- A transition function $T(s, a, s')$
  - Also called the model or dynamics
  - Sometimes $P(s'|s, a)$
- A reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
  - A start state s_0 (and maybe a terminal state)
- MDPs are non-deterministic search problems



- In a MDP, "Markov" means that the action outcomes depend only on the current state

$$P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$
$$=$$
$$P(S_{t+1} = s'|S_t = s_t, A_t = a_t)$$

- This is just like search: in the 1st assignment the children of an expanded state depended only on the current node not how you got there

5.6 Policy

- In deterministic we seek an optimal sequence of actions from start to goal
- In stochastic we seek an optimal policy $\pi^* : S \to A$
  - Policy π gives an action for each state
- Following a policy yields a random path

- The utility, U, of a policy is the (discounted) sum of the rewards along the path
- The goal is to find an optimal policy, π^* that maximizes the expected utility

5.7 Discounting

- It's reasonable to maximize the sum of rewards
- It's also reasonable to prefer rewards now to rewards later
- Solution: values of rewards decay exponentially over time based on some discount factor $0 <= \gamma <= 1$
- We discount so that algorithms can converge and have theoretical guarantees

5.8 Avoiding Infinite Rewards

- **Problem**: If the game lasts forever, do we get infinite rewards?
- Solutions:
  - Introduce some artificial time horizon H
    - Gives nonstationary policies (π depends on the time left)
  - Discounting: use $0 < \gamma < 1$
  - Absorbing state: guarantee that for every policy a terminal state will eventually be reached

5.9 Revisitng MDPs

- Same definition as before but with:
  - Discount factor γ
  - Horizon H (can be ∞)
- MDP quantities so far
  - Policy π: Choice of action for each state
  - Utility $U_\pi = \Sigma_{t=0}^{H} \gamma^t R(s_t)$ Sum of discounte d rewards

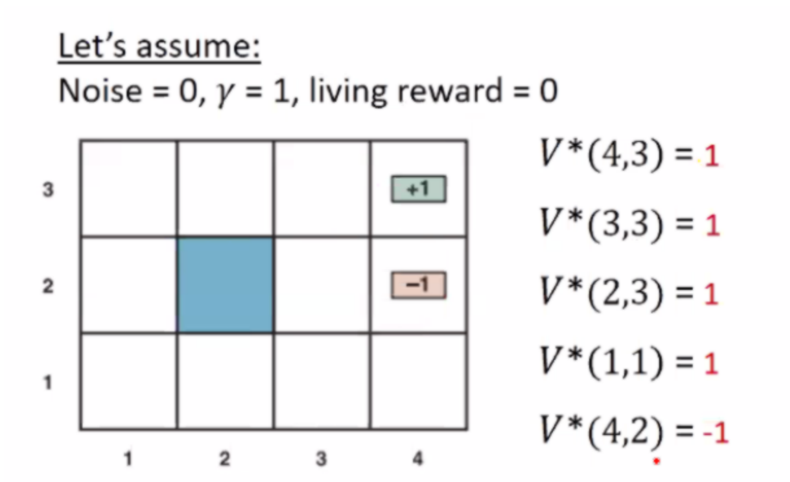
5.10 Solving MDPs

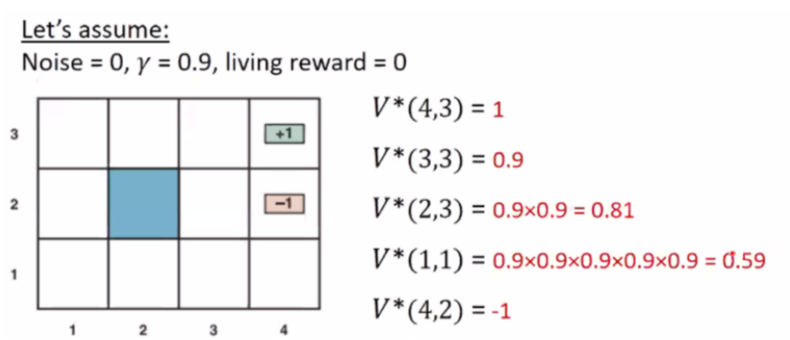
5.10.1 Value Iteration

- $\pi^* = \text{argmax}_\pi \mathbb{E}[\Sigma_{t=0}^{H} \gamma^t R(s_t)|\pi]$

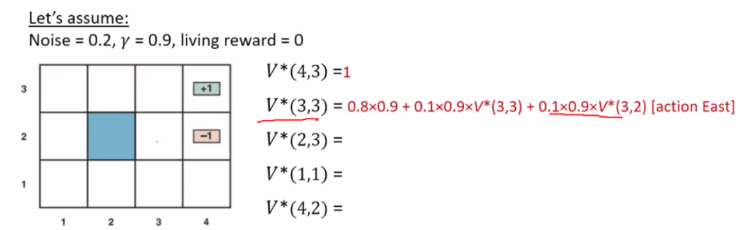
1. Optimal value Function V^*

    - $V^*(s)$ is the sum of discounted rewards starting in s and acting optimally





    - Closed form for V^* is CVV$^* = max_a(R(s, a, s') + \gamma V^*(s'))$



2. Bellman Equation $V^*(s) = max_a \mathbb{E}[R(s,a,s') + \gamma V^*(s')]$

3. Q-Values

   - $Q^*(s,a)$ = expected utility starting at state s, taking action a, and then acting optimally

     - **Bellman equation**

     $$Q^*(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \underbrace{\gamma \max_{a'} Q^*(s',a')}_{})$$

5.10.2 Policy Iteration

1. Extracting Policy from V^*
   - It's not obvous
   - We need to keep track of the optimal policy during value iteration
   - This is called policy extraction, since the policy is informed by the value
2. Extracting policy from Q^*
   - Just take the max from each
   - Optimal policy is implicit
3. Issues with Value iteration
   - Slow $O(S^2A)$ per iteration
   - The "max" at each state rarely changes
   - The policy often converges before the actions
4. Policy evaluation
   - In value iteration we max over all actions to compute optimal values

- If we fixed some policy $\pi(s)$, then only one action per state
  - $V^\pi(s)$ =expected total discounted rewards starting in s and following π
  - The value depends now on which policy we fixed
- Iterate and converge at optimal policy rather than optimal value

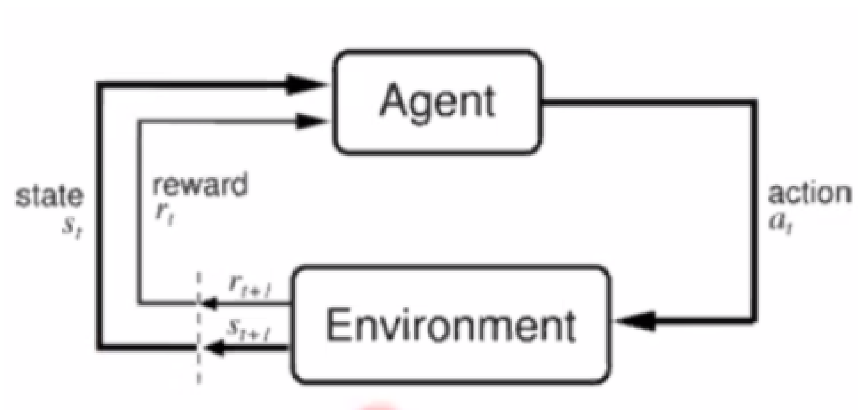
6 Reinforcement Learning

6.1 Definition

- MDP but we don't know P and R

6.2 Framework

At every timestep t the agent picks an action a_t and the environment changes to a new state s_t and the agent is given some reward r_t



6.3 Model-Based Learning

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

6.3.1 Model-Based Monte Carlo

1. Learn empirical MDP using Monte Carlo simulation
   (a) Episodic data: $s_0, a_0, r_1, s_1, a_1, ..., s_T$
   (b) Estimate transitions and rewards

   - $$\hat{P}(s' \mid s, a) = \frac{\#\text{ times } (s,a,s') \text{ occurs}}{\#\text{ times } (s,a) \text{ occurs}}$$

   (c) Discover each $\hat{R}$ when we experience (s, a, s')
2. Estimates converge to the truth
3. Solve using value/policy iteration

6.4 Model-Free Learning

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	MC, TD Learning

6.4.1 Example

-

6.5 Q-Learning

- Used to compute V^*, Q^*, and π^* in a model free, unknown MDP

6.5.1 Active RL

- Full reinforcement learning
- Learner makes choices
- Exploration vs Exploitation

6.5.2 Tabular Q-Learning

- Sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

- Learn Q-values as you go
  - Recieve a sample (s, a, r, s')
  - Consider previous estimate $\hat{Q}(s, a)$
  - Consider your new sample estimate
    * target $= R(s, a, s') + \gamma max_{a'} \hat{Q}(s', a')$
  - Incorperate the new estimate into a running average using some learning rate α
    * $\hat{Q}(s, a) \leftarrow (1 - a)\hat{Q}(s, a) + \alpha[\text{target}]$

6.5.3 Properties

- Q-learning converges to optimal policy, even if you're acting suboptimally if the policy visits every path
- This is called off-policy learning
  - On policy: Learn values of the policy used to generate samples

  - Off policy: Learn the value of another policy

- Caveats

  - Lot of exploration
  - Make the learning rate small enough evantually
  - Basically, in the limit, it doesn't matter how you select actions!

6.5.4 Exploration vs Exploitation

- All states should be visited infinitely often

- Multi-arm bandit

1. How to sample actions?

   - Choose random actions?
     - You need to visit a lot of cells to get enough data of optimal paths
   - Choose action greedily?
     - Never try the other paths that could be higher value?
   - Answer: start by random and transition to greedy to converge
   - Epsilon-greedy
     - With a (small) probability ϵ, act randomly
     - With a (large) probability $1 - \epsilon$, act on current policy

7 Adversarial Games

7.1 Types of Games

- Axes

  - Deterministic vs Stochastic
  - Perfect Information
  - One, two, or more players?
  - Zero sum?

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker scrabble

7.2 Deterministic Games

- Search Problem
  - States: S (starts at s_0)
  - Players: $P = 1 \dots N$
  - Actions: A
  - Transition Function $S \times A \to S$
  - Terminal test: $S \to \{\text{true}, \text{false}\}$
  - Terminal utilities: $S \times P \to \mathbb{R}$
- Solution for a *player* p is a policy $\pi_p : S \to A$
  - π_p is defined when it is p's turn to play

7.3 Why are games hard?

- The utility comes at the end of the game
- Each state is controlled by different players

7.4 Zero Sum Games

- Agents have opposite utilities
- Single value on maximizes that the other minimizes
- Adversarial, pure competition

7.5 General Games

- Agents have independent utilities
- Cooperation, indifference, competition, etc.