Reinforcement Learning

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter 2019

Presented by S. Tanimoto, University of Washington, based on material by Dan Klein and Pieter Abbeel - - University of California.

Outline

- Planning vs Learning
- Sample-Based Policy Evaluation
- Temporal Difference Learning
- Active Reinforcement Learning
- Q-Learning
- Exploration vs Exploitation
- Regret

Planning vs. Learning

- Planning:
  - The agent knows the whole MDP (including S, T and R).
  - If there is no noise, compute a plan (sequence of actions) from start to goal that maximizes total reward. (Unif. Cost Search can often do it).
  - If there is noise, compute an optimal policy, combining Value Iteration with one step of argmaxing from Q-values.
- Learning:
  - The agent does NOT know T and R at all. The agent doesn't even know S, but can recognize states it has visited before.
  - Model-based learning tries to build representations of S, T, and R, but can be very inefficient.
  - Sample-based methods (Temporal-Difference Learning, and Q-Learning) focus on learning good approximations to V and Q values directly.

Agent does not even know state space! Slowly gathering information about the system.

Approach 1: try to approximate T and R by making tons of transitions. Takes a lot of time, and some irrelevant work; some areas of state space we rarely ever visit! Bad!

we dont know T, so instead think of the ' RHS as the EXPECTATION; average of a bunch of samples

Sample-Based Policy Evaluation?

- We want to improve our estimate of V by computing these averages:

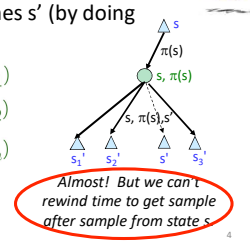
$$V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}_{k}(s')]$$

FIXED policy that we have given agent

- Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s,\pi(s),s'_1) + \gamma V^{\pi}_{k}(s'_1)$$
$$sample_2 = R(s,\pi(s),s'_2) + \gamma V^{\pi}_{k}(s'_2)$$
$$\cdots$$
$$sample_n = R(s,\pi(s),s'_n) + \gamma V^{\pi}_{k}(s'_n)$$
$$V^{\pi}_{k+1}(s) \leftarrow \frac{1}{n}\sum_{i} sample_i$$

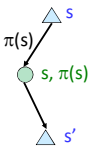
NO! We will average a dfferent way



passive learning: agent does not choose actions; we dictate a set of actions and it learns about state space from it
active learning: agent chooses actions

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often
    rather than take mean of samples after many trials, continuously update state value and learn!
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average



Sample of V(s): $sample = R(s,\pi(s),s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

temporal difference

alpha is our learning rate!

Exponential Moving Average

- Exponential moving average
  - The running interpolation update: $\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$
  - Makes recent samples more important:

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \ldots}{1 + (1-\alpha) + (1-\alpha)^2 + \ldots}$$

  - Forgets about the past (distant past values were wrong anyway)
- Decreasing learning rate (alpha) can give converging averages

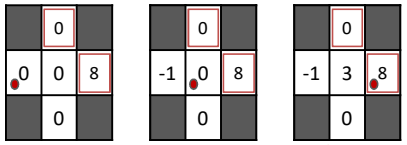
Example: Temporal Difference Learning

States

Assume: γ = 1, α = 1/2

Observed Transitions

B, east, C, -2 | C, east, D, -2



V(Bnew) = (1 - 1/2)V(B_old) + 1.2(-2 + 1*V(C))
= -1

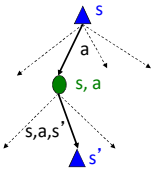
$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages
- However, if we want to turn values into a (new) policy, we don't have the needed Q values:

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

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

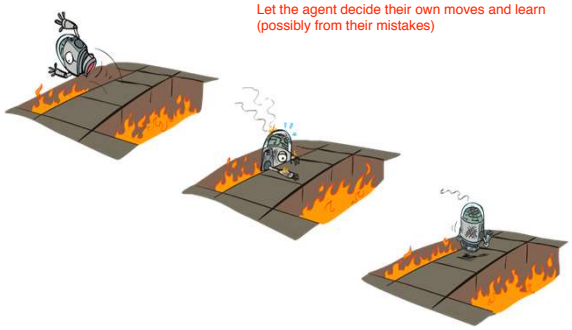


- Idea: learn Q-values, not values
- Makes action selection model-free too!

As the Q values are what allow us to determine the optimal policy.

Active Reinforcement Learning

Let the agent decide their own moves and learn (possibly from their mistakes)



Active Reinforcement Learning

- Full reinforcement learning: optimal policies (like value iteration)
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You choose the actions now
  - Goal: learn the optimal policy / values

- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens…

learn about states we have little information about

Move between good states we are already very familiar with

Prioritize exploration initially

Detour: Q-Value Iteration

- Value iteration: find successive (depth-limited) values
  - Start with $V_0(s) = 0$, which we know is right
  - Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

- But Q-values are more useful, so compute them instead
  - Start with $Q_0(s,a) = 0$, which we know is right
  - Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

k = "how many turns we can make" to decide value of our state. more correctly, the number of update iterations we run to approximate all our Q states

If we know T and R

Q-Learning

- Q-Learning: sample-based Q-value iteration $Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$

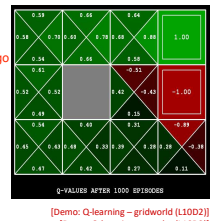
If we know T and R

- Learn Q(s,a) values as you go

dont know T or R; learn as we go

  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate: $sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$
  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

Video of Demo Q-Learning -- Gridworld

Video of Demo Q-Learning -- Crawler

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!

continuously explore and dont starve states (visit them all)

- This is called off-policy learning

- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)

How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy
  - Problems with random actions?
    - You do eventually explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Video of Demo Q-learning – Manual Exploration – Bridge Grid

Video of Demo Q-learning – Epsilon-Greedy – Crawler

Exploration Functions

- When to explore?
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring
- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

Note: this propagates the "bonus" back to states that lead to unknown states as well!

Regular Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

Video of Demo Q-learning – Exploration Function – Crawler

Regret

- Even if you learn the optimal polic you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards
- Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret