Reinforcement Learning

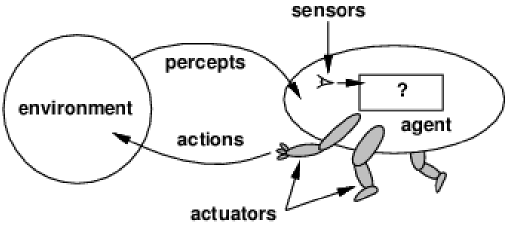
AIMA Chapters: 21.1, 21.2, 21.3.

Sutton and Barto, Reinforcement Learning: an Introduction, 2nd Edition: Chapters 6 (6.1 – 6.5)

Outline

- ♢ Reinforcement Learning: the basic problem
- ♢ Model based RL
- ♢ Model free RL (Q-Learning, SARSA)
- ♢ Exploration vs. Exploitation
- ♢ Slides partially based on the Book "Reinforcement Learning: an introduction" by Sutton and Barto and partially on course by Prof. Pieter Abbeel (UC Berkeley).
- ♢ Thanks to Prof. George Chalkiadakis for providing some of the slides.

Reinforcement Learning: basic ideas



◇ Reinforcement Learning: learn how to map situations to actions, so as to maximize a sequence of rewards.

◇ Key features for RL

- trial and error while interacting with the environment
- delayed reward (actions have effect in the future)

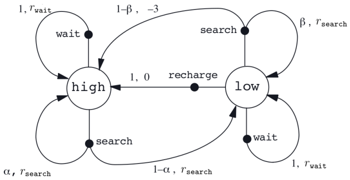
◇ Essentially we need to estimate the long term value of $V(s)$ and find $\pi(s)$

Reinforcement Learning: relationships with MDPs

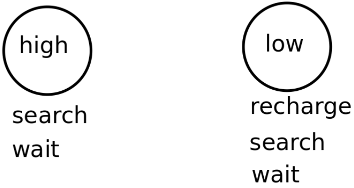
Guide an MDP without knowing the dynamics

- do not know which states are good/bad (no $R(s, a, s')$)
- do not know where actions will lead us (no $T(s, a, s')$)
- hence we must try out actions/states and collect the reward

Recycling robot example: RL



Planning



Learning

To use a model or not to use a model ?

- Model-Based methods methods try to **learn a model**
  - + avoid repeating bad states/actions
  - + fewer execution steps
  - + efficient use of data
- Model-Free methods methods try to **learn Q-function and policy** directly
  - + simplicity, no need to build and use a model
  - + no bias in model design

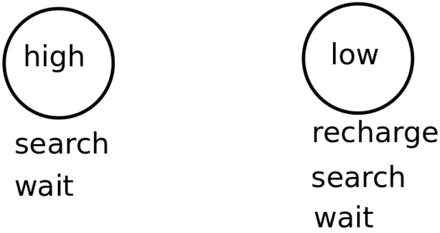
Example: Expected Age

- ♢ Model Based vs. Model Free approaches
- ♢ GOAL: compute expected age for this class.
- ♢ Given probability distribution of ages: $\mathbb{E}[A] = \sum_a P(a) \cdot a$
  - Model Based: estimate $\hat{P}(a)$
  - $\hat{P}(a) = \frac{num(a)}{N}$
  - $\mathbb{E}[A] \approx \sum_a \hat{P}(a) \cdot a$
  - where $num(a)$ is the number of students that have age a
  - works because we learn the right model
  - Model Free: no estimate
  - $\mathbb{E}[A] \approx \frac{1}{N} \sum_i a_i$
  - where a_i is the age value of person i
  - works because samples appear with right frequency

Learning a model: general idea

- Estimate $P(x)$ from samples
  - Acquire samples: $x_i \sim P(x)$
  - Estimate: $\hat{P}(x) = count(x)/k$
- Estimate $\hat{T}(s, a, s')$ from samples
  - Acquire samples: $s_0, a_0, s_1, a_1, s_2, \ldots$
  - Estimate $\hat{T}(s, a, s') = \frac{count(s_{t+1}=s', a_t=a, s_t=s)}{count(s_t=s, a_t=a)}$
- it works because samples appear with the right frequencies

Example: learning a model for the recycling robot



- Given Learning episodes:

E1 : $(L, R, H, 0), (H, S, H, 10), (H, S, L, 10)$

E2 : $(L, R, H, 0), (H, S, L, 10), (L, R, H, 0)$

E3 : $(H, S, L, 10), (L, R, H, 0), (H, S, L, 10)$

- Estimate $T(s, a, s')$ and $R(s, a, s')$

Model-Based methods

Algorithm 1 Model Based approach to RL

Require: A, S, S_0
Ensure: $\hat{T}, \hat{R}, \hat{\pi}$
Initialize $\hat{T}, \hat{R}, \hat{\pi}$
repeat
Execute $\hat{\pi}$ for a learning episode
Acquire a sequence of tuples $\langle (s, a, s', r) \rangle$
Update $\hat{T}$ and $\hat{R}$ according to tuples $\langle (s, a, s', r) \rangle$
Given current dynamics compute a policy (e.g., VI or PI)
until termination condition is met

◇ learning episode: a terminal state is reached or a given amount of time steps

◇ Always execute best action given current model: no exploration

Model Free Reinforcement Learning

◇ Want to compute an expectation weighted by $P(x)$:

$$\mathbb{E}[f(x)] = \sum_x P(x) f(x)$$

◇ Model-based estimate P(x) from samples then compute:

$$x_i \sim P(x),\ \hat{P}(x) = num(x)/N,\ \mathbb{E}[f(x)] \approx \sum_x \hat{P}(x) f(x)$$

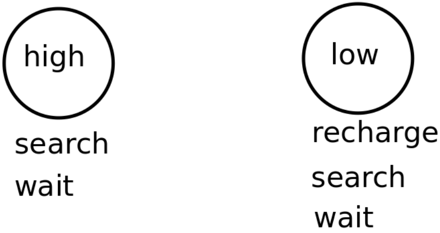
◇ Model-free estimate expectation directly from samples:

$$x_i \sim P(x),\ \mathbb{E}[f(x)] \approx \frac{1}{N} \sum_i f(x_i)$$

Evaluate Value Function from Experience

- ◇ Goal: compute value function given a policy π
- ◇ Average all observed samples
  - execute π for some learning episodes
  - compute sum of (discounted) reward every time a state is visited
  - compute average over collected samples

Example: direct value function evaluation for the recycling robot



- Given Learning episodes:

E1 : $(L, R, H, 0), (H, S, H, 10), (H, S, L, 10)$

E2 : $(L, R, H, 0), (H, S, L, 10), (L, R, H, 0)$

E3 : $(H, S, L, 10), (L, R, H, 0), (H, S, L, 10)$

- Estimate $V(s)$

Sample-Based Policy Evaluation

◇ Goal: improve estimate of V by considering the Bellman update (given a policy π)

$$V_\pi^{k+1}(s) = \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V_\pi^k(s'))$$

◇ Take samples for outcomes of s' and average

- $sample_1 = R(s, \pi(s), s_1^{'}) + \gamma V_\pi^k(s_1^{'})$
- $sample_2 = R(s, \pi(s), s_2^{'}) + \gamma V_\pi^k(s_2^{'})$
- $\ldots$
- $sample_N = R(s, \pi(s), s_N^{'}) + \gamma V_\pi^k(s_N^{'})$

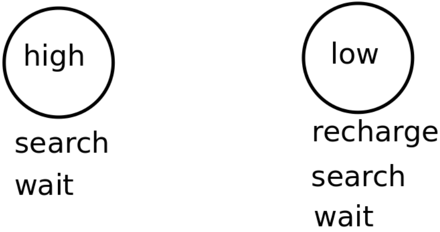
◇ $V_\pi^{k+1}(s) = \frac{1}{N} \sum_i sample_i$

Temporal Difference Learning

- Learn from every experience (not after an episode)
  - Update $V(s)$ after every action given the obtained (s, a, s', r)
  - if we see s' more often this will contribute more (i.e., we are exploiting the underlying T model)
- Temporal difference learning of values
  - compute a running average
  - Sample of $V_\pi(s)$: $sample = R(s, \pi(s), s') + \gamma V_\pi(s')$
  - Update $V_\pi(s)$: $V_\pi(s) \leftarrow (1-\alpha)V_\pi(s) + \alpha(sample)$
  - Temporal Difference: $V_\pi(s) \leftarrow V_\pi(s) + \alpha(sample - V_\pi(s))$
  - α must decrease over time for average to converge, simple option: $\alpha_n = \frac{1}{n}$

$$V_\pi(s) \leftarrow (1-\alpha)V_\pi(s) + \alpha(R(s, \pi(s), s') + \gamma V_\pi(s'))$$

Example: sample-based value function evaluation for the recycling robot



- Given Learning episodes:

  E1 : $(L, R, H, 0), (H, S, H, 10), (H, S, L, 10)$

  E2 : $(L, R, H, 0), (H, S, L, 10), (L, R, H, 0)$

  E3 : $(H, S, L, 10), (L, R, H, 0), (H, S, L, 10)$

- Estimate $V(s)$ considering the structure of bellman update

TD learning for control

- TD gives sample based policy evaluation given a policy
- We want to compute a policy based on $V(s)$
- Can not directly use V to compute π
  - $\pi(s) = \arg\max_a Q(s,a)$
  - $Q(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V(s'))$
- Key idea: we can learn Q-values directly!

A celebrated model-free RL method: Q-Learning

- ♢ Q-Learning: sample based Q-Value iteration
- ♢ Value iteration:

$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_k(s'))$

- ♢ Q-Value iteration: write Q recursively over k
  - $Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma max_{a'} Q_k(s',a'))$
  - can find optimal Q-Values iteratively
  - recall we can not use the model (no T no R)

Sample based Q-Value iteration

◇ Compute an expectation based on samples:
$\mathbb{E}(f(x)) = \frac{1}{N}\sum_i f(x_i)$

◇ Our sample: $R(s, a, s') + \gamma max_{a'} Q_k(s', a')$

◇ Learn $Q(s, a)$ values as you go:

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

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(R(s, a, s') + \gamma max_{a'} Q(s', a'))$$

Properties for Q-Learning

- Q-Learning converges to optimal policy
  - if you explore enough
  - if you make the learning rate small enough
  - ... but not decrease it too quickly
- Action selection does not impact on convergence
  - Off Policy Learning: learn optimal policy without following it
- BUT to guarantee convergence you have to visit every state/action pair infinitely often

Q-Learning: pseudo-code

```
Initialize Q(s,a), ∀s ∈ 𝒮, a ∈ 𝒜(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        Q(S,A) ← Q(S,A) + α[R + γ max_a Q(S',a) − Q(S,A)]
        S ← S';
    until S is terminal
```

♢ ϵ-greedy: choose best action most of the time, but every once in a while (with probability ϵ) choose randomly amongst all action (with equal probabiliy)

SARSA: on-policy alternative for model free RL

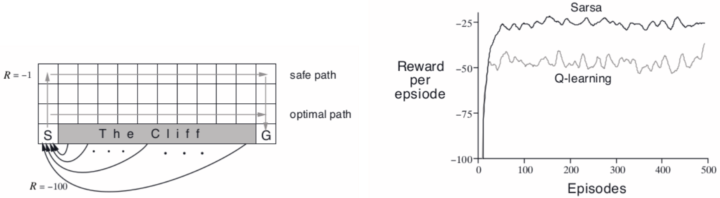
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

♢ SARSA: derives from tuple: (S, A, R, S', A')

♢ Characterized by the fact that we compute next action based on policy (on-policy)

♢ If the policy converges (in the limit) to the greedy policy (and every state/action pairs are visited infinitely often) SARSA converges to optimal $Q^*(s, a)$

SARSA vs Q-Learning



◇ Q-Learning learns the optimal policy but occasionally fails due to ϵ-greedy action selection.

◇ SARSA, being on-policy has a better on-line performance

The Exploration Vs. Exploitation Dilemma

◇ To explore or to exploit ?

- Stay/be happy with whay I already know or
- attempt to test other states-action pairs ?

◇ RL: the agent should explicitly explore the environment to acquire knowledge

◇ Act to improve the estimate of the value function (exploration) or to get high (expected) payoffs (exploitation) ?

◇ Reward maximization requires exploration, but too much exploration of irrelevant parts can waste time.

- choice depends on particular domain and learning technique.

Exploration vs. Exploitation: standard approaches

♢ Key point: to guarantee convergence to optimal we need to explore every state-action pairs sufficiently often in the long run.

♢ Main methods used in practice:

- ϵ-greedy:
  - choose greedily most of the time (probability 1-ϵ)and choose randomly with probability ϵ
- soft-max (or Boltzmann)
  - choose action a with probability $p(a) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$
  - T is a parameter (often called temperature)
  - high T $\rightarrow$ all actions are equiprobable (we explore more)
  - low T $\rightarrow$ greater difference in selection probability towards actions with highest Q (we exploit more)

Exploration functions

◇ Key point: include bonus to explore new parts of the state space inside the Q-Update

◇ Main idea: explore areas if we are not sure they are bad (optimism in face of uncertainty)

◇ Exploration function

- Consider an estimate u and visit count n and compute $f(u, n) = u + k/n$
  - regular update: $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a, s') + \gamma max_{a'} Q(s, a'))$
  - modified update: $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a, s') + \gamma max_{a'} f(Q(s, a'), N(s', a'))$
- $N(s', a')$ is our n (number of times we visited a state-action pair)
- k is a fixed parameter

Summary

- ♢ RL: agent tries to learn what to do while acting
- ♢ Assume an underlying unknown MDP
- ♢ Model based methods: try to learn dynamics and then compute policy
- ♢ Model free methods: try to directly estimate Q-values for state-action pairs
  - Q-learning one of the most interesting off-policy method
- ♢ Exploration vs. Exploitation trad-off
  - depends on specific domain techniques
  - practical approaches are ϵ-greedy or soft max