LG Advanced Data Scientists Program
Deep Learning

[9: Reinforcement Learning (Part 2)]

Prof. Sungroh Yoon

Electrical & Computer Engineering | Seoul National University

(last compiled at 15:32:00 on 2020/02/28)

Outline

References

- books/papers:
  - Reinforcement Learning (2nd edition)[1] Link
  - Artificial Intelligence: A Modern Approach[2]
  - A brief survey of deep reinforcement learning[3]
- online resources:
  - Silver UCL class Link & ICML tutorial Link
  - Schulman MLSS tutorial Link
  - Abbeel & Schulman NIPS tutorial Link
  - UC Berkeley CS188 (AI) Link & CS294 (DRL) Link
  - Stanford CS234 (RL) Link

[1] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press

[2] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach.* Pearson Education Limited

[3] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*

Outline

Value-based methods

- estimate optimal value function $\Rightarrow$ derive optimal policy π^* therefrom
- learning $=$ changing values of states we visit
  - for more accurate value estimation (*e.g.* winning probabilities)
- to do this: we “______” the value of
  - s' : state after each move to
  - s : state before the move

$$
\begin{array}{ccc}
s & \xrightarrow{\text{state: move}} & s' \\
V(s) & \xleftarrow[\text{value: backup}]{} & V(s')
\end{array}
$$

i.e. current value of earlier state s:

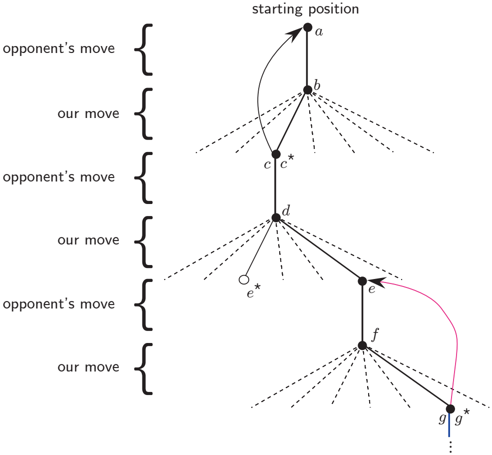
  - adjusted to be closer to value of later state s'

- learning involves a lot of backup operations

Example: a sequence of moves in a two-player game

- solid lines:
  - moves taken during a game
- dashed lines:
  - moves considered but not taken
  - discarded by "exploitation"
- exploratory moves

  e.g. our second move
  - taken even if another sibling move (leading to e^*) was better
  - "exploration"
- curved arrows
  - backups $\Rightarrow$ ______



(source: [Sutton and Barto, 2018][4])

[4] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press

Backup operations

- transfer value information *back*
  - to a state from its successor states

    or
  - to a state-action pair from its successor state-action pairs
- that is, "backup" refers to "______" of values
- $\boxed{\text{backups are at the heart of RL methods}}$

Three ways to do backup

1. **full** backup by *dynamic programming* (DP)

$$V(s) \leftarrow \mathbb{E}\left[r + \gamma V(s')\right]$$

2. **sample** backup by *Monte Carlo* (MC) learning

$$V(s) \leftarrow V(s) + \alpha\left[R - V(s)\right]$$

3. **sample** backup by *temporal-difference* (TD) learning

$$V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$$

- R : sample return (actual return from a trajectory)
- α : step-size parameter
  - a small positive fraction that influences ____________

more on way #3:

- use a simple rule to update $V(s)$

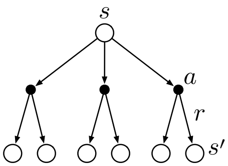
$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right] \\ \iff V(s) &\leftarrow \underbrace{(1-\alpha)}_{\substack{\text{weight on} \\ \text{old value}}} V(s) + \underbrace{\alpha}_{\substack{\text{weight on} \\ \text{new value}}} \left[r + \gamma V(s')\right] \end{aligned} \tag{1}$$

- update rule (1): an example of *temporal-difference* (TD) learning
  - changes are based on $\underbrace{r + \gamma V(s') - V(s)}_{\substack{\uparrow \\ \text{difference between estimates at two different times}}}$

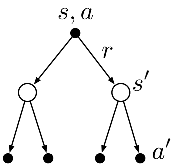
Backup diagram

- depict relationships that form the basis of ______ operations

  e.g. for dynamic programming to compute $V(s)$ and $Q(s, a)$:



$V(s) \leftarrow \mathbb{E}[r + \gamma V(s')]$



$Q(s, a) \leftarrow \mathbb{E}[r + \gamma Q(s', a')]$

- notations
  - open circle: a state
  - solid circle: a state-action pair

Taxonomy of value-based methods

two kinds of defining characteristics:

- if we bootstrap
  - we update estimates based on other ________ (not true target)
- if we sample
  - we do not compute but just sample an expectation

	sample backup	**full backup**
bootstrap (shallow backup)	temporal-difference (TD) learning	dynamic programming (DP)
no bootstrap (deep backup)	Monte Carlo (MC) learning	exhaustive search

example: sample-backup methods

- **Monte-Carlo** (MC) learning
  - go all the way to ___ of a trajectory and
  - estimate the value just by looking at sample return

  $\Rightarrow$ no bootstrapping

- **temporal-difference** (TD) learning[5]
  - just look one step ahead and
  - estimate the value after one step using one-step lookahead value estimate

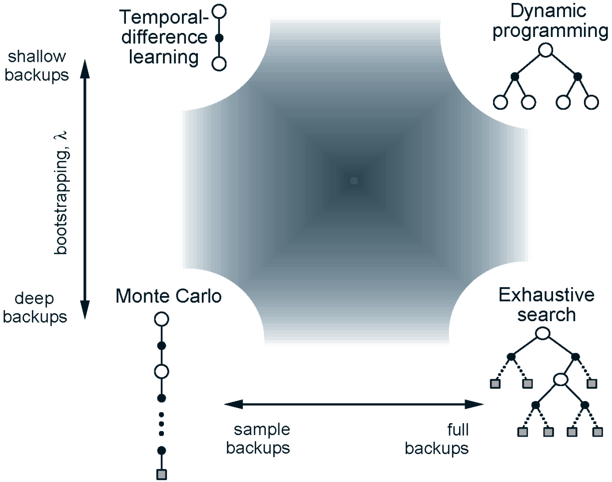
  $\Rightarrow$ bootstrapping

- TD(λ): generalize/unify[6]
  - use arbitrary # of lookaheads

[5] more precisely, one-step TD or TD(0)

[6] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press

Unified view of RL



(source: [Sutton and Barto, 2018][7])

[7] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press

Outline

Dynamic programming (DP)

- refers to a collection of algorithms
  - that can compute optimal policies given a perfect model as an MDP

- classical DP algorithms: of limited utility in RL because
  - assume a perfect model
  - computationally very ________

- why do we learn DP then?
  - essential foundation for many $\underbrace{\text{RL methods}}$

    ↑
    achieve much the same effect as DP with less computation and no model

- key idea of DP (and of RL in general):
  - use of value functions to organize/structure the search for good policies
  - if find optimal value fcns (V^* or Q^*) $\Rightarrow$ easily obtain optimal policies

- etymology
  - dynamic: sequential/temporal component to the problem
  - programming: optimizing a "program" (*i.e.* a policy)
- properties required to apply DP
  - optimal substructure
  - overlapping subproblems
- MDPs satisfy both
  - Bellman equations gives recursive decomposition
  - value function stores and _____ solutions

Richard E. Bellman (1920-1984)

(source: Wikipedia)

Planning in MDP

- dynamic programming: used for planning in MDP
  - assumes ___ knowledge of MDP
- prediction problem
  - input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - output: value function V^π
- control problem
  - input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - output: optimal value function V^*
    optimal policy π^*

How to compute the value of a state?

- two scenarios:

1. no explicit policy is given
   - value iteration: find optimal V^*
   - byproduct: optimal policy π^* (policy extraction)
2. policy π is given
   - policy evaluation
   - components of policy iteration (policy ________ $\rightarrow$ policy improvement)

- we assume deterministic policy $\pi(s) = a$ for clarity
  - same framework applies to stochastic policy $\pi(a \mid s)$

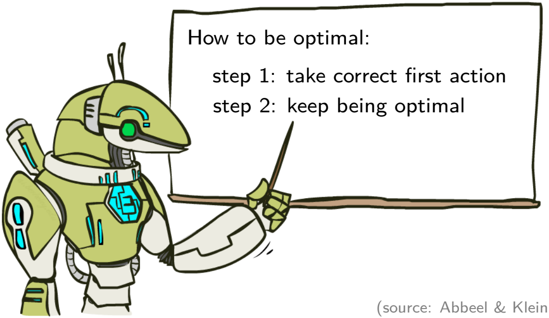
Scenario #1: no policy is given

- big question: how to act in each state?
  - no explicit policy is given
  - ⇒ agent should decide which action to take
- principle of maximum expected utility (MEU)[8]
  - a rational agent should choose
    - the action that maximizes agent's expected utility
- that is, we can assume the agent act optimally
  - then try to find the optimal ____ of each state
  - this will also lead to finding optimal policy

[8] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Pearson Education Limited

Bellman optimality equation

- Bellman found a way to estimate $V^*(s)$
- $V^*(s)$: the optimal value of any state s
  - sum of all discounted future rewards the agent can expect on average after it reaches s, assuming it acts ________



(source: Abbeel & Klein)

- Bellman showed: if the agent acts optimally
  - **Bellman ________ equation** applies: $\forall s$

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} \mathbb{E}_{s'} \left[r(s, a, s') + \gamma V^*(s') \right] \\
&= \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s') \left[r(s, a, s') + \gamma V^*(s') \right]
\end{aligned}
\tag{2}
$$

  - $T(s, a, s')$: transition probability from s to s', given that agent chose a
  - $r(s, a, s')$: reward from s to s', given that agent chose a
  - γ : discount rate

- that is, optimal value $V^*(s)$

  $=$ average reward after taking one optimal action

  $+$ expected optimal value of all possible next states

Value iteration

- eq (2): ________
  - linear algebra techniques are not applicable
  - instead we solve it using an *iterative* approach called value iteration

- $\boxed{\text{value iteration}}$
  - main idea: turn Bellman equation into update rule

    Bellman update:

    $$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V_k(s')]$$

  - an instance of dynamic programming

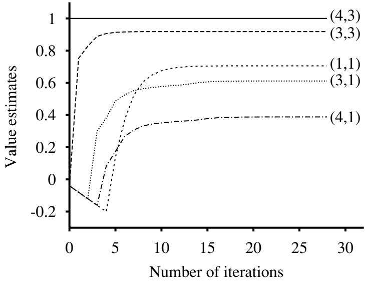
- VI algorithm sketch
  1. start with some initial guess $V_0(s), \forall s$
  2. for every iteration k, update value function estimates, $\forall s$:

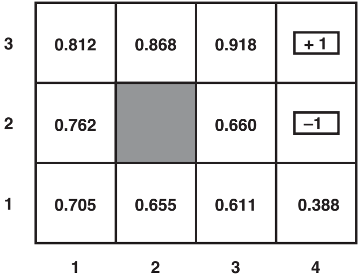
$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V_k(s')]$$

  3. stop when appropriate (*e.g.* when no significant change occurs)

- amazing facts:
  - convergence is guaranteed: as $k \to \infty$, $V_k(s) \to V^*(s)$
  - unique solution
  - corresponding ______ is optimal

- **example:** grid world





(source: [Russell and Norvig, 2016][9])

  - left: evolution of values of selected states using VI
  - right: state values[10]

---

[9] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Pearson Education Limited

[10] $\gamma = 1, r(s) = -0.004$

Q-value iteration

- Q-values are more useful (*e.g.* see policy extraction on page 27)

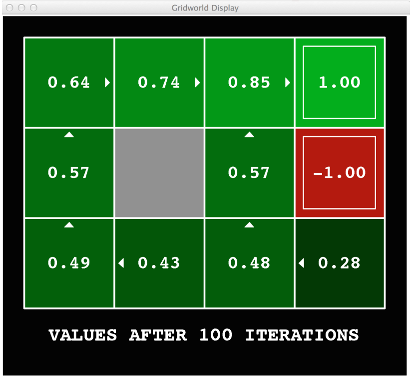
Bellman update:

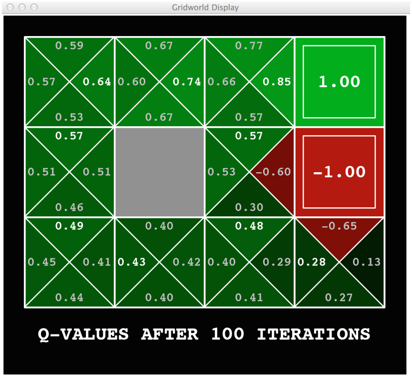
$$Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') \left[r(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

c.f. (state) value iteration

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V_k(s')]$$

- **example:** grid world





(source: Abbeel & Klein)

- left: state values (noise $= 0.2$, $\gamma = 0.9$, living reward $= 0$)
- right: Q-values (same condition)

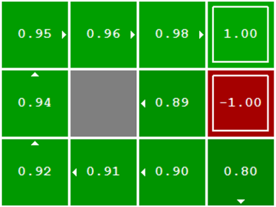
Policy extraction

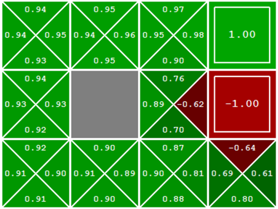
- getting policy implied by values
  - from $V^*(s)$: not obvious

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V^*(s')]$$

  - from $Q^*(s, a)$: ______

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a)$$

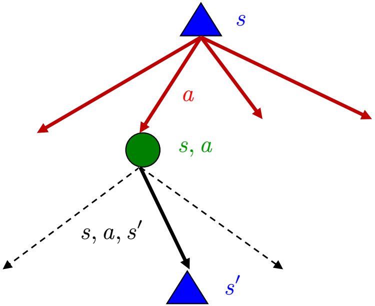




(source: Abbeel & Klein)

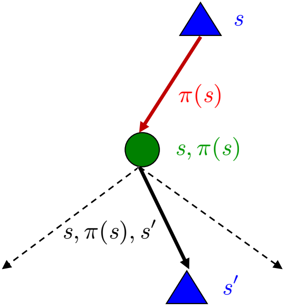
Scenario #2: fixed policy π is given

- no π given (act optimally):



  - max over all actions to compute the optimal values

- π is given (do what π says to do):



  - simpler: only ___ action per state

Utilities for a fixed policy

- another basic operation
  - compute $V(s)$ under fixed (generally non-optimal) policy π
- define $V^\pi(s)$, the utility of state s under a fixed policy π
  - $V^\pi(s)$: expected total discounted rewards starting in s and following π
- **Bellman __________ equation**

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[r(s, \pi(s), s') + \gamma V^\pi(s')]$$

c.f. Bellman optimality equation

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V^*(s')]$$

Policy evaluation

- how to calculate state values for a fixed policy π?
- method $\#1$: $\boxed{\text{iterative policy evaluation}}$
  - turn Bellman __________ equation into update (like value iteration)

$$V_0^\pi(s) = 0$$
$$V_{k+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') \left[r(s, \pi(s), s') + \gamma V_k^\pi(s') \right]$$

  - cost: $O(|\mathcal{S}|^2)$ per iteration
  - full backup[11] (c.f. MC or TD learning)
  - convergence to V^π can be proved: $V_0 \to V_1 \to \cdots \to V^\pi$
- method $\#2$: use a linear system solver
  - without $\max$, Bellman equations are just a linear system

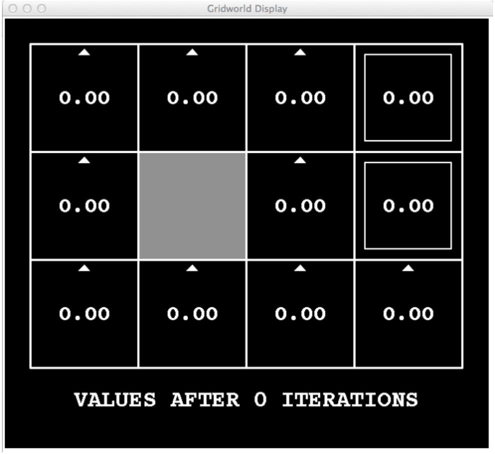
[11] based on all possible next states rather than on a sample next state

Problems with value iteration

- VI repeats Bellman updates

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \sum_{s'} T(s, a, s')[r(s, a, s') + \gamma V_k(s')]$$

- problems
  1. slow: $O(|\mathcal{S}|^2 |\mathcal{A}|)$ per iteration
  2. "max" operation at each state rarely changes
  3. policy often converges long before _____



(animation can be viewed only in Acrobat Reader)

Policy iteration

- an alternative approach to getting optimal policy
  - still optimal (always converges to π^*)
  - can converge much faster than VI in some cases

- repeat the following (over i):
  1. policy evaluation: for fixed current (not optimal) π_i
     - find state values with policy evaluation ($k \to \infty, \forall s$)

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s, \pi_i(s), s') \left[r(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

  2. policy improvement
     - one-step look-ahead, ______ update: $\forall s$

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

Outline

Summary

- value-based reinforcement learning methods
  - estimate optimal value function $V^*(s)$ or $Q^*(s, a)$
  - $\Rightarrow$ then find optimal policy π^* therefrom
  - key operation: backup (= update of $V(s)$ using $V(s')$)
  - defining characteristic #1: sample vs full backup
  - defining characteristic #2: shallow (=bootstrap) vs deep backup

- tabular methods: represent value function by lookup table
  - dynamic programming: full + shallow backup
    - value iteration and policy iteration
  - temporal-difference (TD) learning: sample + shallow backup
    - Q-learning (off-policy) and SARSA (on-policy)
  - Monte Carlo (MC) learning: sample + deep backup

- value function approximation by deep neural net
  - deep Q-network (DQN): experience replay with fixed Q-learning target