Dynamic Programming and Reinforcement Learning

Monte Carlo Techniques and Q-Learning (Week 4)

Rainer Schlosser, Alexander Kastius

Hasso Plattner Institute (EPIC)

May 12, 2022

Outline

- Questions?
- Today: Finally, dynamics do not have to be known

  Learning & optimizing from simulation

  Monte Carlo Simulations

  Q-Learning

Recap: Last Week

- Approximate Dynamic Programming
- Forward Dynamic Programming
- Simulation-based Approaches
- Exercises & Implementation
- Value iteration & Policy iteration

Solving MDP Problems via DP and RL

- Discrete Time MDP Problems **with full knowledge** (last weeks)
  - Optimal Solutions (curse of dimensionality)
  - ADP & relaxation concepts to attack larger problem sizes
  - Forward Dynamic Programming (simulation-based)

- Discrete Time MDP Problems **with less knowledge** (today)
  - Time-independent (stationary) infinite horizon framework
  - No knowledge about reward distributions or state transitions
  - Simulation-based evaluation of policies
  - Simulation-based optimization of policies

MDP Problems with Different Characteristics

Example Objective **State** **Action** **Events** Rewards

Airline Tickets

Hotel/Rental/Rail

Apparel/Seasonal/Events

Perishable Products

Distinguish:

Finite vs Infinite vs "Sink"

Inventory Mgmt.

Durable Products

E-Commerce

Resource Allocations

Tetris/Chess/Go

Self-driving

Distinguish:

System dynamics known vs unknown

Recap

- Considered setup: Infinite horizon (stationary)
- Stationary policy: $\pi(s)$ for all states $s \in S$

- Realized trajectory: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_t, a_t, r_t, \ldots$
- (Observed) disc. future reward in s_t: $G_t = G_t(s_t) = \sum_{k \geq 0} \gamma^k \cdot r_{t+k}$
- Recursion for G_t: $G_t(s = s_t) = r_t + \gamma \cdot G_{t+1}(s' = s_{t+1})$

- Sink: A final state will be reached at a **random** time T
- No sink: There is **no absorbing state** (cf. inventory prob.)

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(1) Generate/simulate *one* trajectory (the sink is reached at time T):

$s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$

For each trajectory compute all $G_t(s = s_t)$ (via recursion from r_T)

(2)

(3)

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(1) Generate/simulate *one* trajectory (the sink is reached at time T):

$s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$

For each trajectory compute all $G_t(s = s_t)$ (via recursion from r_T)

(2) For all $t = 0, 1, \ldots, T$ estimate the policy's value function $V^{(\pi)}(s)$ via:

$V^{(\pi)}(s_t) \leftarrow G_t(s_t)$

Can we do better?

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(1) Generate/simulate *one* trajectory (the sink is reached at time T):

$s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T$

For each trajectory compute all $G_t(s = s_t)$ (via recursion from r_T)

(2) For all $t = 0, 1, \ldots, T$ estimate the policy's value function $V^{(\pi)}(s)$ via:

$V^{(\pi)}(s_t) \leftarrow G_t(s_t)$

(3) Use more simulated trajectories!

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(3) Generate/simulate ***k*=1,...,*K* trajectories** (the sink is reached at time $T^{(k)}$):

$s_0^{(k)}, a_0^{(k)}, r_0^{(k)}, s_1^{(k)}, a_1^{(k)}, r_1^{(k)}, \ldots, s_{T^{(k)}}^{(k)}, a_{T^{(k)}}^{(k)}, r_{T^{(k)}}^{(k)}$

For each trajectory compute all $G_t^{(k)}(s = s_t^{(k)})$ (via recursion from $r_{T^{(k)}}^{(k)}$)

(4) How to update the estimation for $V^{(\pi)}(s)$?

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(3) Generate/simulate ***k*=1,...,*K* trajectories** (the sink is reached at time $T^{(k)}$):

$$s_0^{(k)}, a_0^{(k)}, r_0^{(k)}, s_1^{(k)}, a_1^{(k)}, r_1^{(k)}, \ldots, s_{T^{(k)}}^{(k)}, a_{T^{(k)}}^{(k)}, r_{T^{(k)}}^{(k)}$$

For each trajectory compute all $G_t^{(k)}(s = s_t^{(k)})$ (via recursion from $r_{T^{(k)}}^{(k)}$)

(4) For all $t = 0, 1, \ldots, T$ of a run *k* **update** the estimation for $V^{(\pi)}(s)$ using a learning rate parameter $\eta \in (0,1)$ as follows:

$$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot G_t(s_t^{(k)}) + (1-\eta) \cdot V^{(\pi)}(s_t^{(k)})$$

Monte-Carlo Estimation (with Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics**?

(3) Generate/simulate ***k*=1,...,*K* trajectories** (the sink is reached at time $T^{(k)}$):

$$s_0^{(k)}, a_0^{(k)}, r_0^{(k)}, s_1^{(k)}, a_1^{(k)}, r_1^{(k)}, \ldots, s_{T^{(k)}}^{(k)}, a_{T^{(k)}}^{(k)}, r_{T^{(k)}}^{(k)}$$

For each trajectory compute all $G_t^{(k)}(s = s_t^{(k)})$ (via recursion from $r_{T^{(k)}}^{(k)}$)

(4) For all $t = 0, 1, \ldots, T$ of a run *k* **update** the estimation for $V^{(\pi)}(s)$ using a learning rate parameter $\eta \in (0,1)$ as follows:

$$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot G_t(s_t^{(k)}) + (1-\eta) \cdot V^{(\pi)}(s_t^{(k)})$$

$$= \eta \cdot \left(G_t(s_t^{(k)}) - V^{(\pi)}(s_t^{(k)})\right) + V^{(\pi)}(s_t^{(k)})$$

Temporal Difference Learning (without Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics** when there is no sink (to start to compute $G_t(s_t) = r_t + \gamma \cdot G_{t+1}(s_{t+1})$)

Temporal Difference Learning (without Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics** when there is no sink (to start to compute $G_t(s_t) = r_t + \gamma \cdot G_{t+1}(s_{t+1})$)

- Idea: To estimate $G_{t+1}(s_{t+1})$ use $V^{(\pi)}(s_{t+1}) = E\left(G_{t+1}(s_{t+1})\right)$:-)

Temporal Difference Learning (without Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics** when there is no sink (to start to compute $G_t(s_t) = r_t + \gamma \cdot G_{t+1}(s_{t+1})$)

- Idea: To estimate $G_{t+1}(s_{t+1})$ use $V^{(\pi)}(s_{t+1}) = E\left(G_{t+1}(s_{t+1})\right)$:-)

- Replace $$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot \underbrace{G_t(s_t^{(k)})}_{r_t^{(k)}+\gamma\cdot G_{t+1}(s_{t+1}^{(k)})} + (1-\eta)\cdot V^{(\pi)}(s_t^{(k)}),$$ cf. MCE (4),

  by $$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot \left(r_t^{(k)} + \gamma \cdot \underbrace{V^{(\pi)}(s_{t+1}^{(k)})}_{=E\left(G_{t+1}(s_{t+1}^{(k)})\right)} \right) + (1-\eta)\cdot V^{(\pi)}(s_t^{(k)})$$

Temporal Difference Learning (without Sink)

- Performance $V^{(\pi)}(s)$ of a given policy $\pi(s)$ **under unknown dynamics** when there is no sink (to start to compute $G_t(s_t) = r_t + \gamma \cdot G_{t+1}(s_{t+1})$)

- Idea: To estimate $G_{t+1}(s_{t+1})$ use $V^{(\pi)}(s_{t+1}) = E\left(G_{t+1}(s_{t+1})\right)$:-)

- Replace $$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot \underbrace{G_t(s_t^{(k)})}_{r_t^{(k)} + \gamma \cdot G_{t+1}(s_{t+1}^{(k)})} + (1-\eta) \cdot V^{(\pi)}(s_t^{(k)}),$$ cf. MCE (4),

  by $$V^{(\pi)}(s_t^{(k)}) \leftarrow \eta \cdot \left(r_t^{(k)} + \gamma \cdot \underbrace{V^{(\pi)}(s_{t+1}^{(k)})}_{=E\left(G_{t+1}(s_{t+1}^{(k)})\right)} \right) + (1-\eta) \cdot V^{(\pi)}(s_t^{(k)})$$

$$= \eta \cdot \left(r_t^{(k)} + \gamma \cdot V^{(\pi)}(s_{t+1}^{(k)}) - V^{(\pi)}(s_t^{(k)}) \right) + V^{(\pi)}(s_t^{(k)})$$

Towards Optimized Policies

- We can learn/simulate the performance $V^{(\pi)}(s)$ of a **given** policy $\pi(s)$ **under unknown dynamics**

- Can we optimize state-dependent actions based on $V^{(\pi)}(s)$?

Towards Optimized Policies

- We can learn/simulate the performance $V^{(\pi)}(s)$ of a **given** policy $\pi(s)$ **under unknown dynamics**

- Can we optimize state-dependent actions based on $V^{(\pi)}(s)$?

$$V^{(\pi)}(s_t^{(k)}) = \max_{a_t \in A} E\left(r_t^{(k)} + \gamma \cdot V^{(\pi)}(s_{t+1}^{(k)})\right) \quad ???$$

- Missing coupling element?

Towards Optimized Policies

- We can learn/simulate the performance $V^{(\pi)}(s)$ of a **given** policy $\pi(s)$ **under unknown dynamics**

- Can we optimize state-dependent actions based on $V^{(\pi)}(s)$?

$$V^{(\pi)}(s_t^{(k)}) = \max_{a_t \in A} E\left(r_t^{(k)} + \gamma \cdot V^{(\pi)}(s_{t+1}^{(k)})\right) \quad ???$$

- Missing coupling element: anticipation of state transitions!

- Solution options:

Towards Optimized Policies

- We can learn/simulate the performance $V^{(\pi)}(s)$ of a **given** policy $\pi(s)$ **under unknown dynamics**

- Can we optimize state-dependent actions based on $V^{(\pi)}(s)$?

$$V^{(\pi)}(s_t^{(k)}) = \max_{a_t \in A} E\left(r_t^{(k)} + \gamma \cdot V^{(\pi)}(s_{t+1}^{(k)})\right) \quad ???$$

- Missing coupling element: anticipation of state transitions!

- Solution options: – estimate state transition probabilities

  – learn *state-action-values* (more efficient)

Q-Values

- $V^{(\pi)}(s)$ expected disc. future rewards of a given policy $\pi(s)$
- $Q^{(\pi)}(s,a)$??

Q-Values

- $V^{(\pi)}(s)$ expected disc. future rewards of a given policy $\pi(s)$
- $Q^{(\pi)}(s,a)$ expected disc. future rewards of a given policy $\pi(s)$ when instead **now playing *a* in *s*** and then again continue to play $\pi(s)$

$$Q^{(\pi)}(s,a) := E\left(G_t \qquad\qquad \mid s_t = s, a_t = a, \forall k > t \; use \; a_k = \pi(s_k)\right)$$

$$= E\left(r_t + \gamma \cdot V^{(\pi)}(s_{t+1}) \mid s_t = s, a_t = a\right)$$ (effect of state transitions is "included"!)

Q-Values

- $V^{(\pi)}(s)$ expected disc. future rewards of a given policy $\pi(s)$
- $Q^{(\pi)}(s,a)$ expected disc. future rewards of a given policy $\pi(s)$ when instead **now playing *a* in *s*** and then again continue to play $\pi(s)$

$$Q^{(\pi)}(s,a) := E\left(G_t \qquad\qquad | \, s_t = s, a_t = a, \forall k > t \; use \; a_k = \pi(s_k)\right)$$

$$= E\left(r_t + \gamma \cdot V^{(\pi)}(s_{t+1}) \,|\, s_t = s, a_t = a\right)$$ (effect of state transitions is “included”!)

- Note, $Q^{(\pi)}(s,\pi(s)) = V^{(\pi)}(s)$, i.e., V is a special case of Q
- And it allows to optimize policies!??

Q-Values

- $V^{(\pi)}(s)$ expected disc. future rewards of a given policy $\pi(s)$
- $Q^{(\pi)}(s,a)$ expected disc. future rewards of a given policy $\pi(s)$ when instead **now playing *a* in *s*** and then again continue to play $\pi(s)$

$$Q^{(\pi)}(s,a) := E\left(G_t \qquad\qquad \mid s_t = s, a_t = a, \forall k > t \; use \; a_k = \pi(s_k)\right)$$

$$= E\left(r_t + \gamma \cdot V^{(\pi)}(s_{t+1}) \mid s_t = s, a_t = a\right)$$ (effect of state transitions is “included”!)

- Note, $Q^{(\pi)}(s,\pi(s)) = V^{(\pi)}(s)$, i.e., V is a special case of Q
- Allows to optimize: $a^*(s) = \arg\max_{a_t \in A} \{Q(s,a_t)\}$ cf. policy iteration (!)

Estimating Q-Values (of a Policy) using SARSA

(1) Play a **given** policy $\pi(s)$, i.e., observe s_t, a_t, r_t and also s_{t+1}, a_{t+1}

(2) Update the Q-value estimate

Estimating Q-Values (of a Policy) using SARSA

(1) Play a **given** policy $\pi(s)$, i.e., observe s_t, a_t, r_t and also s_{t+1}, a_{t+1}

(2) Update the Q-value estimate (start with random values or 0) via:

$$Q^{(\pi)}(s_t, a_t) \leftarrow \eta_t \cdot \left(r_t + \gamma \cdot Q^{(\pi)}(s_{t+1}, a_{t+1})\right) + (1-\eta_t) \cdot Q^{(\pi)}(s_t, a_t)$$

where the learning rate η_t may be reduced over time
to obtain estimates that remain constant, e.g., using $\eta_t := 1/t$

Estimating Q-Values (of a Policy) using SARSA

(1) Play a **given** policy $\pi(s)$, i.e., observe s_t, a_t, r_t and also s_{t+1}, a_{t+1}

(2) Update the Q-value estimate (start with random values or 0) via:

$$Q^{(\pi)}(s_t, a_t) \leftarrow \eta_t \cdot \left(r_t + \gamma \cdot Q^{(\pi)}(s_{t+1}, a_{t+1})\right) + (1-\eta_t) \cdot Q^{(\pi)}(s_t, a_t)$$

$$= \eta_t \cdot \left(r_t + \gamma \cdot Q^{(\pi)}(s_{t+1}, a_{t+1}) - Q^{(\pi)}(s_t, a_t)\right) + Q^{(\pi)}(s_t, a_t)$$

where the learning rate η_t may be reduced over time
to obtain estimates that remain constant, e.g., using $\eta_t := 1/t$

(3) If the policy is changed also the Q-values change

(4) Can we find an optimal policy?

Estimating Q-Values (of a Policy) using SARSA

(1) Play a **given** policy $\pi(s)$, i.e., observe s_t, a_t, r_t and also s_{t+1}, a_{t+1}

(2) Update the Q-value estimate (start with random values or 0) via:

$$Q^{(\pi)}(s_t, a_t) \leftarrow \eta_t \cdot \left(r_t + \gamma \cdot Q^{(\pi)}(s_{t+1}, a_{t+1})\right) + (1-\eta_t) \cdot Q^{(\pi)}(s_t, a_t)$$
$$= \eta_t \cdot \left(r_t + \gamma \cdot Q^{(\pi)}(s_{t+1}, a_{t+1}) - Q^{(\pi)}(s_t, a_t)\right) + Q^{(\pi)}(s_t, a_t)$$

where the learning rate η_t may be reduced over time
to obtain estimates that remain constant, e.g., using $\eta_t := 1/t$

(3) If the policy is changed also the Q-values change

(4) An ε-greedy version of $\pi_t(s) = \arg\max_{a \in A} Q_t(s, a)$ is guaranteed
to converge to the optimal policy (as all pairs *s* and *a* are reachable).

Optimal Q-Values using Tabular Q-Learning (QL)

(1) Play the current policy $\pi(s)$, i.e., observe s_t, a_t, r_t and s_{t+1}

(2) Update the Q-value estimate

Optimal Q-Values using Tabular Q-Learning (QL)

(1) Play the current policy $\pi(s)$, i.e., observe s_t, a_t, r_t and s_{t+1}

(2) Update the Q-value estimate (start with random values or 0 in t=0) via:

$$Q(s_t, a_t) \leftarrow \eta_t \cdot \left(r_t + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) \right) + (1 - \eta_t) \cdot Q(s_t, a_t)$$

where the learning rate η_t may be reduced over time
to obtain estimates that remain constant, e.g., using $\eta_t := 1/t$

(3) Can we find an optimal policy?

Optimal Q-Values using Tabular Q-Learning (QL)

(1) Play the current policy $\pi(s)$, i.e., observe s_t, a_t, r_t and s_{t+1}

(2) Update the Q-value estimate (start with random values or 0 in t=0) via:

$$Q(s_t, a_t) \leftarrow \eta_t \cdot \left(r_t + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) \right) + (1 - \eta_t) \cdot Q(s_t, a_t)$$

$$= \eta_t \cdot \left(r_t + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t) \right) + Q(s_t, a_t)$$

where the learning rate η_t may be reduced over time
to obtain estimates that remain constant, e.g., using $\eta_t := 1 / t$

(3) An ε-greedy version of $a_t := \pi(s_t) = \arg\max_{a \in A} Q(s_t, a)$ is guaranteed to converge to the optimal policy (as all pairs s and a are reachable).

On-Policy vs. Off-Policy Learning

SARSA

- Tuples are generated by the policy that we want to learn the values for
- Future estimations of a Q-value still depends on the policy
- The requirements for the policy forces us to generate the tuples in the specified order using the most current iteration of the policy
- If the policy used to generate the tuples is different, the values will change
- **This style of algorithm is called on-policy**

Q-Learning

- The tuples can be generated by **any** policy at any time, the generated Q-values will be the same
- The only requirement to ensure convergence: every combination of *s* and *a* that is visited repeatedly in endless time
- **This style of algorithm is called off-policy**
- Improves SARSA by shortening the learning process with off-policy learning

Summary (Solving Discrete Time MDPs via ADP)

SARSA & Q-Learning

(+) no system knowledge is required
(+) provides near-optimal solutions for infinite horizon MDPs
(+) guaranteed convergence
(+) numerically simple
(+) general applicable
(+) obtain good heuristics

(–) updates only for single "visited" states (cf. large state spaces)
(–) results are stochastic (due to simulated next states)
(–) hyper parameter tuning (e.g., learning + exploration rate)

Next: Deep QL, allows to attack larger problems

Overview

Week	Dates	Topic	
1	April 21	Introduction	
2	April 25/28	Finite + Infinite Time MDPs	
3	May 2/5	Approximate Dynamic Programming (ADP) + DP Exercise	
4	May 12	Q-Learning (QL)	(not Mon May 9)
5	May 16/19	**Deep Q-Networks (DQN)**	
6	May 23	DQN Extensions	(not Thu May 26 “Himmelfahrt”)
7	May 30/June 2	Policy Gradient Algorithms	
8	June 9	Project Assignments	(not Mon June 6 “Pfingstmontag”)
9	June 13/16	Work on Projects: Input/Support	
10	June 20/23	Work on Projects: Input/Support	
11	June 27/30	Work on Projects: Input/Support	
12	July 4/7	Work on Projects: Input/Support	
13	July 11/14	Work on Projects: Input/Support	
14	July 18/21	Final Presentations	
	Sep 15	Finish Documentation	