Artificial Intelligence

Dynamic Programming

Marc Toussaint
University of Stuttgart
Winter 2019/20

Motivation:

So far we focussed on tree search-like solvers for decision problems. There is a second important family of methods based on dynamic programming approaches, including Value Iteration. The Bellman optimality equation is at the heart of these methods.

Such dynamic programming methods are important also because standard Reinforcement Learning methods (learning to make decisions when the environment model is initially unknown) are directly derived from them.

Markov Decision Process

Tree Search vs. Dynamic Programming

- *Tree search* and sampling (MCTS) is one way to collect information about possible futures
- *Dynamic Programming* is the core alternative, which computes a function over states that accumulates all information (value function, cost-to-go function)

Tree Search vs. Dynamic Programming

- *Tree search* and sampling (MCTS) is one way to collect information about possible futures
- *Dynamic Programming* is the core alternative, which computes a function over states that accumulates all information (value function, cost-to-go function)

- To enable Dynamic Programming, we need a (probabilistic) model of possible transitions. Previously this was the $\texttt{succ}$ function $s \mapsto \{s', ..\}$. Now we formulate transitions probabilistically $P(s'|s, a)$ as *Markov Decision Processes* (MDPs).

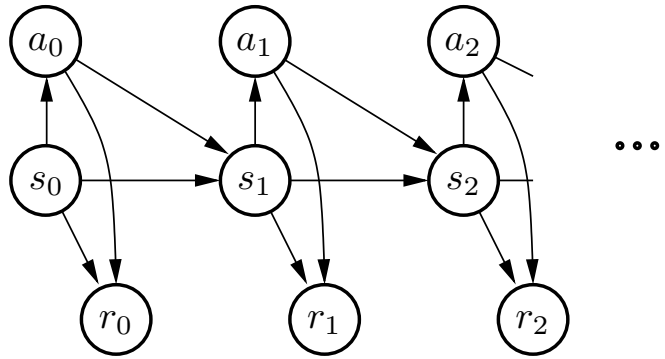
Tree Search vs. Dynamic Programming

- *Tree search* and sampling (MCTS) is one way to collect information about possible futures
- *Dynamic Programming* is the core alternative, which computes a function over states that accumulates all information (value function, cost-to-go function)

- To enable Dynamic Programming, we need a (probabilistic) model of possible transitions. Previously this was the `succ` function $s \mapsto \{s', ..\}$. Now we formulate transitions probabilistically $P(s'|s, a)$ as *Markov Decision Processes* (MDPs).

- MDPs are the basis of Reinforcement Learning, where $P(s'|s, a)$ is not know by the agent. A core reason for formalizing $P(s'|s, a)$ probabilistically is that the agent only has partial knowledge about transitions, due to limited experience.

Markov Decision Process



$$P(s_{0:T+1}, a_{0:T}, r_{0:T}; \pi) = P(s_0) \prod_{t=0}^{T} P(a_t|s_t;\pi)\ P(r_t|s_t,a_t)\ P(s_{t+1}|s_t,a_t)$$

- world's initial state distribution $P(s_0)$
- world's transition probabilities $P(s_{t+1} \mid s_t, a_t)$
- world's reward probabilities $P(r_t \mid s_t, a_t)$
- agent's *policy* $\pi(a_t \mid s_t) = P(a_0|s_0;\pi)$ (or deterministic $a_t = \pi(s_t)$)

- **Stationary MDP:**
  - We assume $P(s' \mid s, a)$ and $P(r|s,a)$ independent of time
  - We also define $R(s,a) := \mathsf{E}\{r|s,a\} = \int r\ P(r|s,a)\ dr$

MDP

- In basic discrete MDPs, the transition probability

$$P(s'|s,a)$$

  is just a table of probabilities

- The *Markov* property refers to how we defined state:
  History and future are conditionally independent given s_t

$$I(s_{t^+}, s_{t^-}|s_t) \ , \quad \forall t^+ > t, t^- < t$$

Dynamic Programming

State value function

- We consider a stationary MDP described by

$$P(s_0)\ , \quad P(s' \,|\, s, a)\ , \quad P(r \,|\, s, a)\ , \quad \pi(a_t \,|\, s_t)$$

- The **value** (*expected* discounted return) of policy π when started in state s is:

$$V^\pi(s) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \,|\, s_0 = s\}\ ,$$

with *discounting factor* $\gamma \in [0, 1]$

State value function

- We consider a stationary MDP described by

$$P(s_0)\ , \quad P(s' \,|\, s, a)\ , \quad P(r \,|\, s, a)\ , \quad \pi(a_t \,|\, s_t)$$

- The **value** (*expected* discounted return) of policy π when started in state s is:

$$V^\pi(s) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \,|\, s_0 = s\}\ ,$$

with *discounting factor* $\gamma \in [0, 1]$

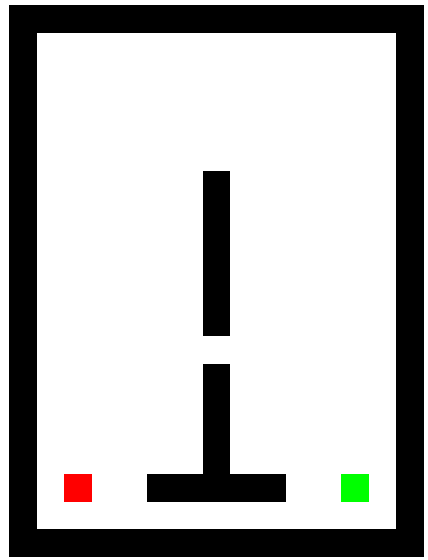
- A policy π^* is **optimal** iff

$$\forall s:\ V^{\pi^*}(s) = V^*(s)\ , \quad \text{where}\ \ V^*(s) = \max_\pi V^\pi(s)\ ,$$

i.e., simultaneously maximises the value in all states

(In MDPs there always exists (at least one) optimal deterministic policy.)

An example for a value function...



demo: `test/mdp runVI`

Values provide a gradient towards desirable states

Value function

- The value function V is a central concept in all of RL!

  Many algorithms can directly be derived from properties of the value function.

- In other domains (stochastic optimal control) it is also called *cost-to-go* function (cost $= -$reward)

Recursive property of the value function

$$
\begin{aligned}
V^{\pi}(s) &= \mathsf{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathsf{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s)) \; \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s)) \; V^{\pi}(s')
\end{aligned}
$$

Recursive property of the value function

$$
\begin{aligned}
V^\pi(s) &= \mathsf{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s; \pi\} \\
&= \mathsf{E}\{r_0 \mid s_0 = s; \pi\} + \gamma \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_0 = s; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s))\ \mathsf{E}\{r_1 + \gamma r_2 + \cdots \mid s_1 = s'; \pi\} \\
&= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s' \mid s, \pi(s))\ V^\pi(s')
\end{aligned}
$$

- We can write this in vector notation $\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{V}^\pi$ with vectors $\boldsymbol{V}^\pi_s = V^\pi(s)$, $\boldsymbol{R}^\pi_s = R(s, \pi(s))$ and matrix $\boldsymbol{P}^\pi_{ss'} = P(s' \mid s, \pi(s))$

- For stochastic $\pi(a|s)$:
$$V^\pi(s) = \textstyle\sum_a \pi(a|s) R(s, a) + \gamma \sum_{s', a} \pi(a|s) P(s' \mid s, a)\ V^\pi(s')$$

Bellman optimality equation

- Recall the recursive property of the value function

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\ V^{\pi}(s')$$

- Bellman optimality equation

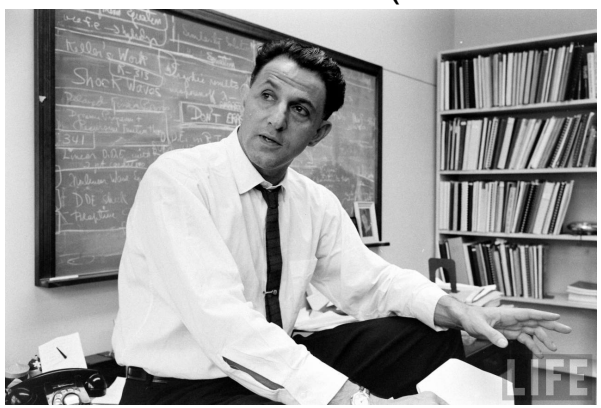
$$V^{*}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s, a)\ V^{*}(s') \right]$$

with $\pi^{*}(s) = \operatorname{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s, a)\ V^{*}(s') \right]$

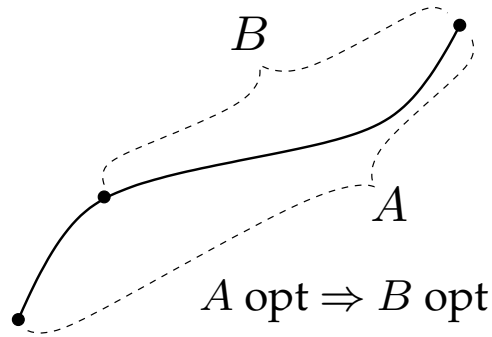
(Sketch of proof: If π would select another action than $\operatorname{argmax}_a[\cdot]$, then π' which $= \pi$ everywhere except $\pi'(s) = \operatorname{argmax}_a[\cdot]$ would be better.)

- This is the **principle of optimality** in the stochastic case

Richard E. Bellman (1920-1984)



Bellman's principle of optimality



$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^*(s') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^*(s') \right]$$

Value Iteration

- *How can we use this to compute V^*?*
- Recall the Bellman optimality equation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ V^*(s') \right]$$

- **Value Iteration:** (initialize $V_{k=0}(s) = 0$)

$$\forall s:\ V_{k+1}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a)\ V_k(s') \right]$$

stopping criterion: $\max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$

- Note that V^* is a **fixed point** of value iteration!
- Value Iteration converges to the optimal value function V^* (proof below)

`demo: test/mdp runVI`

State-action value function (Q-function)

- *We repeat the last couple of slides for the Q-function...*
- The *state-action value function* (or **Q-function**) is the expected discounted return when starting in state s and taking first action a:

$$\begin{aligned} Q^\pi(s,a) &= \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, a_0 = a\} \\ &= R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, Q^\pi(s', \pi(s')) \end{aligned}$$

(Note: $V^\pi(s) = Q^\pi(s, \pi(s))$.)

- Bellman optimality equation for the Q-function

$$Q^*(s,a) = R(s,a) + \gamma \textstyle\sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

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

Q-Iteration

- Recall the Bellman equation:

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

- **Q-Iteration:** (initialize $Q_{k=0}(s,a) = 0$)

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

stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \leq \epsilon$

- Note that Q^* is a **fixed point** of Q-Iteration!
- Q-Iteration converges to the optimal state-action value function Q^*

Proof of convergence

- Let $\Delta_k = ||Q^* - Q_k||_\infty = \max_{s,a} |Q^*(s,a) - Q_k(s,a)|$

$$
\begin{aligned}
Q_{k+1}(s,a) &= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_k(s',a') \\
&\leq R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} \Big[Q^*(s',a') + \Delta_k \Big] \\
&= \Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \Big] + \gamma \Delta_k \\
&= Q^*(s,a) + \gamma \Delta_k
\end{aligned}
$$

  similarly: $Q_k \geq Q^* - \Delta_k \;\Rightarrow\; Q_{k+1} \geq Q^* - \gamma \Delta_k$

- The proof translates directly also to value iteration

For completeness**

- **Policy Evaluation** computes V^π instead of V^*: Iterate:

$$\forall s: \; V^\pi_{k+1}(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) \; V^\pi_k(s')$$

  Or use matrix inversion $\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{P}^\pi)^{-1} \boldsymbol{R}^\pi$, which is $O(|S|^3)$.

- **Policy Iteration** uses V^π to incrementally improve the policy:
  1. Initialise π_0 somehow (e.g. randomly)
  2. Iterate:
     - **Policy Evaluation:** compute V^{π_k} or Q^{π_k}
     - **Policy Update:** $\pi_{k+1}(s) \leftarrow \text{argmax}_a \, Q^{\pi_k}(s, a)$

`demo: test/mdp runPI`

Summary: Bellman equations

- Discounted infinite horizon:

$$V^*(s) = \max_a Q^*(s,a) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\ \max_{a'} Q^*(s',a')$$

- With finite horizon T (non stationary MDP), initializing $V_{T+1}(s) = 0$

$$V_t^*(s) = \max_a Q_t^*(s,a) = \max_a \left[R_t(s,a) + \gamma \sum_{s'} P_t(s' \mid s,a)\ V_{t+1}^*(s') \right]$$

$$Q_t^*(s,a) = R_t(s,a) + \gamma \sum_{s'} P_t(s' \mid s,a)\ \max_{a'} Q_{t+1}^*(s',a')$$

- This recursive computation of the value functions is a form of **Dynamic Programming**

Comments & relations

- Tree search is a form of **forward** search, where heuristics (A^* or UCB) may optimistically estimate the value-to-go
- Dynamic Programming is a form of **backward** inference, which exactly computes the value-to-go backward from a horizon
- UCT also estimates $Q(s, a)$, but based on Monte-Carlo rollouts instead of exact Dynamic Programming

- In deterministic worlds, Value Iteration is the same as *Dijkstra backward*; it labels all nodes with the value-to-go ($\leftrightarrow$ cost-to-go).

- In *control theory*, the Bellman equation is formulated for continuous state x and continuous time t and ends-up:

$$-\frac{\partial}{\partial t}V(x,t) = \min_u \Big[c(x,u) + \frac{\partial V}{\partial x} f(x,u)\Big]$$

  which is called *Hamilton-Jacobi-Bellman* equation.
  For linear quadratic systems, this becomes the *Riccati equation*

Comments & relations

- The Dynamic Programming principle is applicable in any domain – but inefficient if the state space is large (e.g. relational or high-dimensional continuous)
- It requires iteratively computing a value function over the whole state space

Conclusions

- We covered two basic types of planning methods
  - Tree Search: forward, but with backward heuristics
  - Dynamic Programming: backward

- Dynamic Programming explicitly describes optimal policies. Exact DP is computationally heavy in large domains $\rightarrow$ approximate DP
  Tree Search became very popular in large domains, esp. MCTS using UCB as heuristic