

Chapter 3 Probabilistic Reasoning

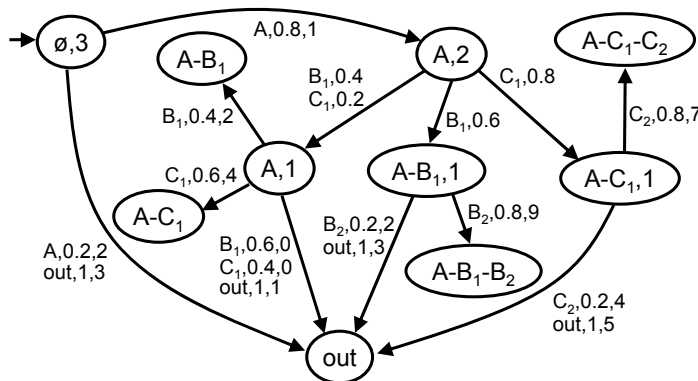
Paragraph 2 Markov Models

Example



- You enter a master program at a prestigious university.
- There are two tracks for the program which lasts 3 semesters, one course per semester: you can either take
 - the entry course A (semester 1) followed by the course sequence B_1 (semester 2) and B_2 (semester 3),
 - or entry course A (semester 1) followed by the course sequence C_1 (semester 2) and C_2 (semester 3).
- If you fail the entry course A, they kick you out of the program.
- You can leave early, right away or after one year. Getting credit for different courses qualifies you differently well for your dream job. Of course, fulfilling all requirements in one of the two tracks carries you furthest, but credit in individual courses is still worth something.
- Sequence B_1/B_2 is the best qualification, but also has the highest failure rate.

Example



Markov Decision Process

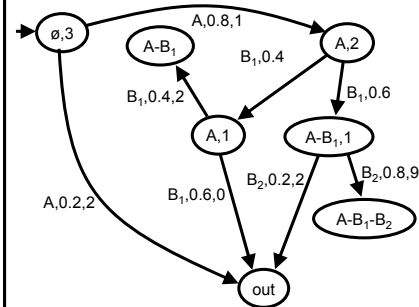
- Denote the start state by s_0 .
- Denote by $T(s,a,s')$ the probability that we move from state s to s' when taking action a .
- Denote by $R(s,a,s')$ the reward that we get when taking action a at s and ending up in state s' .
- A Markov Model is given by the triple (s_0, T, R) .
- The key feature of an MDP is the following:
 - Both the reward and the probability of moving to s' depend only on the current state s and the action taken! Both do not depend on the history how we got to state s !

Example



- Assume that we want to go for the B_1/B_2 sequence and stick to coming closest to this goal even if we fail a course.
- What is the expected qualification score that we will end up with?

Expected Utility



- Denote the expected utility of our policy to strive for B_1/B_2 at state s by $U^{B_1/B_2}(s)$. Denote by $B_1/B_2(s)$ the action suggested at state s .
- $U^{B_1/B_2}(s)$ = average reward over all random walks through the network following policy B_1/B_2 beginning at state s .

$$U^{B_1/B_2}(s) = \sum_{s'} (R(s, B_1/B_2(s), s') + U^{B_1/B_2}(s')) T(s, B_1/B_2(s), s')$$

Expected Utility

- In our example, the MDP is described by an acyclic graph. Then, all policies must reach a terminal state (we call such policies proper).
- Consequently, expected utilities can be computed in reverse-topological order.
- What happens when MDPs are cyclic?
 - What when policies are improper and rewards are unbounded?
 - How to compute expected utilities when a reverse topological ordering does not exist?

Expected Utility

- To address infinite rewards, we discount rewards that we will achieve in the distant future.
- $U^\pi(s)$ = discounted average reward over all random walks through the network following policy π beginning at state s .
- $U^\pi(s) = \sum_{s'} (R(s, \pi(s), s') + \gamma U^\pi(s')) T(s, \pi(s), s')$ for $0 < \gamma < 1$.
- These expected utilities are finite and can be computed by solving a linear equation system!
- Therefore, we have a way of evaluating a policy π . But how do we find a good policy?!

Policy Iteration

- Set $k:=0$, pick a random policy π^0 .
- While (true)
 - Determine $U^{\pi^k}(s)$.
 - For all states s
 - $\pi^{k+1}(s) := \operatorname{argmax}_a \sum_{s'} (R(s,a,s') + \gamma U^{\pi^k}(s')) T(s,a,s')$.
 - If $\pi^{k+1} = \pi^k$ the return π^k .
 - $k:=k+1$.

- How to evaluate a policy when the state space is huge?
- Why does this function terminate and return an optimal policy?

Iterative Policy Evaluation

- We can evaluate policy π by the following iterative Bellman update:
 - $U_{i+1}^{\pi}(s) = \sum_{s'} (R(s,\pi(s),s') + \gamma U_i^{\pi}(s')) T(s,\pi(s),s')$
- Why does the sequence of vectors U_i^{π} converge?
- Let U^{π} denote a solutions vector with $U^{\pi}(s) = \sum_{s'} (R(s,\pi(s),s') + \gamma U^{\pi}(s')) T(s,\pi(s),s')$.

Iterative Policy Evaluation

- Lemma (Contraction): $\|U_{i+1}^{\pi} - U^{\pi}\|_{\infty} \leq \gamma \|U_i^{\pi} - U^{\pi}\|_{\infty}$.
- Proof: Set $\Delta_i := \|U_i^{\pi} - U^{\pi}\|_{\infty}$.
 $U_i^{\pi}(s) \leq U^{\pi}(s) + \Delta_i$ implies
 $U_{i+1}^{\pi}(s) = \sum_{s'} (R(s,\pi(s),s') + \gamma U_i^{\pi}(s')) T(s,\pi(s),s')$
 $\leq \sum_{s'} (R(s,\pi(s),s') + \gamma U^{\pi}(s')) T(s,\pi(s),s') + \gamma \Delta_i \sum_{s'} T(s,\pi(s),s')$
 $= U^{\pi}(s) + \gamma \Delta_i$
Analogously, $U_i^{\pi}(s) \geq U^{\pi}(s) - \Delta_i$ implies
 $U_{i+1}^{\pi}(s) \geq U^{\pi}(s) - \gamma \Delta_i$.
And thus:
 $\|U_{i+1}^{\pi} - U^{\pi}\|_{\infty} = \max_s |U_{i+1}^{\pi}(s) - U^{\pi}(s)| \leq \gamma \Delta_i = \gamma \|U_i^{\pi} - U^{\pi}\|_{\infty}$.

Value Iteration

- Consequently, the maximum error between U_i^{π} and U^{π} decreases exponentially fast.
- Moreover, the previous lemma implies that there is one unique solution U^{π} to our system of equations.
- Interestingly, we could use the same argument as before to show convergence of the update procedure $U_{i+1}(s) = \max_a \sum_{s'} (R(s,a,s') + \gamma U_i(s')) T(s,a,s')$.
- This procedure is known as value iteration and can be used as an alternative to policy iteration.

Policy Iteration

- Set $k:=0$, pick a random policy π^0 .
- While (true)
 - Determine $U^{\pi^k}(s)$.
 - For all states s
 - $\pi^{k+1}(s) := \operatorname{argmax}_a \sum_{s'} (R(s,a,s') + \gamma U^{\pi^k}(s')) T(s,a,s')$.
 - If $\pi^{k+1} = \pi^k$ then return π^k .
 - $k:=k+1$.

- How to evaluate a policy when the state space is huge?
- Why does this function terminate and return an optimal policy?

Termination and Optimality

- By choosing the action that achieves the maximum expected profit at each state, we know that $U^{\pi^{k+1}}(s) \geq U^{\pi^k}(s)$ for all s .
- Consequently, every consecutive policy π^{k+1} is better than its predecessor π^k . In particular, this implies that no policy is repeated!
- As there are only finitely many policies, policy iteration must terminate and return the best overall policy.

Remarks

- Instead of waiting for the computation of U^{π^k} to converge, we can just run a couple of update steps and then compute a new policy right away. This is known as modified policy iteration.
- Instead of updating both U^{π^k} and π^k for all states s at the same time, we could just do it for a subset of states in each iteration. If we do this carefully, we can still show convergence. This is known as asynchronous policy iteration.
- In practice, it is surprisingly fast and usually preferred over value iteration.

A Different Setting

- Our initial example assumed that transition probabilities and rewards are known a priori.
- In practice, this is usually not the case. Moreover, even when we know the data, the previous setting requires us to specify an input model that has size quadratic in the number of states - which can be prohibitively large in real-size scenarios.

Reinforcement Learning

- In reinforcement learning, we try to compute a good policy while learning transition model and reward function as we explore the environment.
- The obvious trade-off that we are facing is that of exploration (to learn more) and exploitation (to make use of what we know already).

Q-Learning

- Recall our equilibrium equation for the utility of state s under policy π :
$$U^\pi(s) = \sum_{s'} (R(s, \pi(s), s') + \gamma U^\pi(s')) T(s, \pi(s), s')$$
- We define the expected state-action utility under policy π as:
$$Q^\pi(s, a) := \sum_{s'} (R(s, a, s') + \gamma U^\pi(s')) T(s, a, s')$$

Q-Learning

- For the optimal utilities, we get:

$$U(s) = \max_a Q(s, a)$$

whereby we define the optimal state-action utility as

$$Q(s, a) := \sum_{s'} (R(s, a, s') + \gamma U(s')) T(s, a, s').$$

- With probability ϵ we choose a random action a_t , else we choose $\operatorname{argmax}_b Q(s_{t+1}, b)$ as our next action a_t .
- When moving from state s_t to s_{t+1} by taking action a_t and while observing reward r_t , we update the state-action pairs as follows:
$$Q(s_t, a_t) := (1 - \alpha) Q(s_t, a_t) + \alpha [r_t + \gamma \max_b Q(s_{t+1}, b)]$$
- Here, α is an exploitation/exploration parameter that should decrease as we learn more about the environment.

SARSA

- Like Q-learning, also the following RL-algorithm maintains state-action pairs during optimization. On top of this, it also maintains the current policy π .
- Assume, while moving from state s_t to s_{t+1} by taking action a_t , we obtain reward r_t . Already now, we determine the next action a_{t+1} : With probability ϵ , we choose a_{t+1} randomly. Otherwise, we set
$$a_{t+1} := \pi(s_{t+1})$$
- Then, we update the state-action pairs and the policy:
$$Q(s_t, a_t) := (1 - \alpha) Q(s_t, a_t) + \alpha [r_t + \gamma Q(s_{t+1}, a_{t+1})]$$

$$\pi(s_t) := \operatorname{argmax}_a Q(s_t, a)$$
- Repeat.

SARSA

- Note that SARSA updates the state-action pairs according to the actual action that we choose to take rather than the one that would lead to optimal expected profit given our current state of knowledge.
- Q-learning does not make such mistakes when choosing to explore a random action rather than the expected optimal.
- SARSA is called an on-policy strategy, Q-learning and off-policy strategy. As a consequence of more accurate updates, Q-learning often converges faster.

