Artificial Intelligence

Albert-Ludwigs-Universität Freiburg

Thorsten Schmidt
Abteilung für Mathematische Stochastik

www.stochastik.uni-freiburg.de
thorsten.schmidt@stochastik.uni-freiburg.de
SS 2017

Our goal today

Approximate dynamic programming
- The basic idea
- Q-Learning

Literature (incomplete, but growing):

- I. Goodfellow, Y. Bengio und A. Courville (2016). **Deep Learning**. http://www.deeplearningbook.org. MIT Press
- D. Barber (2012). **Bayesian Reasoning and Machine Learning**. Cambridge University Press
- R. S. Sutton und A. G. Barto (1998). **Reinforcement Learning : An Introduction**. MIT Press
- G. James u. a. (2014). **An Introduction to Statistical Learning: With Applications in R**. Springer Publishing Company, Incorporated. ISBN: 1461471370, 9781461471370
- T. Hastie, R. Tibshirani und J. Friedman (2009). **The Elements of Statistical Learning**. Springer Series in Statistics. Springer New York Inc. URL: https://statweb.stanford.edu/~tibs/ElemStatLearn/
- K. P. Murphy (2012). **Machine Learning: A Probabilistic Perspective**. MIT Press
- CRAN Task View: Machine Learning, available at https://cran.r-project.org/web/views/MachineLearning.html
- UCI ML Repository: http://archive.ics.uci.edu/ml/ (371 datasets)
- Warren B Powell (2011). **Approximate Dynamic Programming: Solving the curses of dimensionality**. Bd. 703. John Wiley & Sons
- A nice resourse is https://github.com/aikorea/awesome-rl

Approximate dynamic programming (ADP)

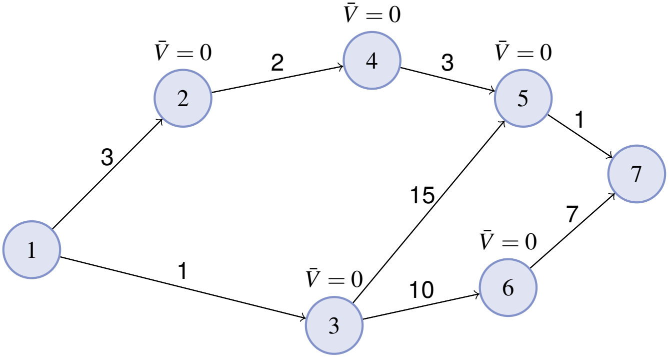
- While we introduce a nice theory beforehand, the core equation

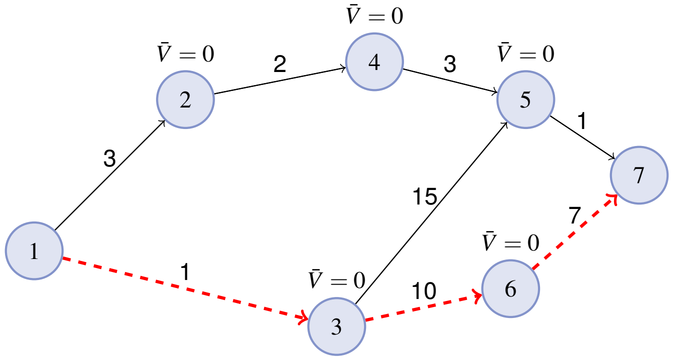
$$\sup_{\pi} \mathbb{E}^{\pi}\left[\sum_{0=1}^{T} C_t(S_t, a_t)\right]$$

  my be intractable even for very small problems.
- ADP now offers a powerful set of strategies to solve these problems approximately.
- The idea stems from the 1950's while a lot of the core work was done in the 80's and 90's.

- We have the problem of curse of dimensionality in **state space**, **outcome space** and **action space**.

We illustrate this by an example: we approximate the value function by the function $\bar{V}$ which we update iteratively.



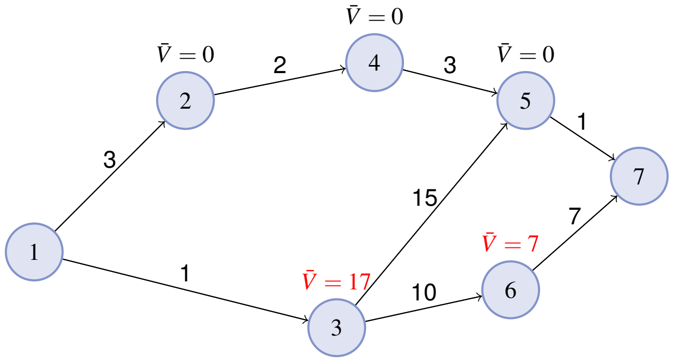


The approximation of the value function is **optimistic**: $\bar{V}=0$ at all states. We start (forward !) in node 1 and choose the node where

$$c_{ij}+\bar{V}(j)$$

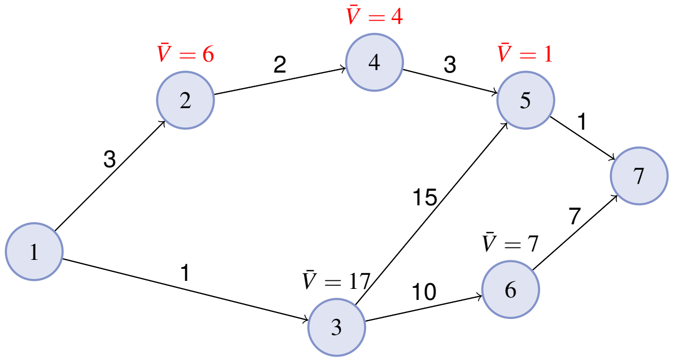
is minimal. This means we choose $1-3-6-7$ and update (!) accordingly:

$$\bar{V}(3)=17, \quad \bar{V}(6)=7.$$



Now (because we initiated optimistically) we take another round and go $1-2-4-5-7$, again updating the weights.

Now (because we initiated optimistically) we take another round and go $1-2-4-5-7$, again updating the weights.



We have found the optimal path !

The above example (still a deterministic one) shows a number of interesting features:

- We proceed forward - which is suboptimal, but repeat until we have found an optimal (or close-to-optimal) solution.
- The value function is approximated.
- The choice of the initial $\bar{V}$ can make us explorative or less explorative - it will become important further on to have this in mind.
- Typical examples are the learning of a robot (for example to stop a ball).

The basic idea

- There are many variants of ADP - here we look at the basic idea: we proceed forward and approximate $\bar{V}$ iteratively.
- We start with an initial approximation

$$\bar{V}_t^0(s), \qquad \text{for all } t = 0, \ldots, T-1,\ s \in \mathscr{S}.$$

- Then we proceed iteratively.

Basic ADP algorithm

Starting from $\bar{V}^{n-1}$ we proceed as follows:

1. simulate a path $S(\omega) =: (s_0, s_1, \dots, s_T)$.
2. at $t = 0$ we compute
$$\hat{v}_0^n = \hat{v}_0^n(\omega) = \max_{a \in \mathscr{A}_0(s_0)} \left\{ C_0(s_0, a) + \mathbb{E}[\bar{V}_1^{n-1}(S_1) | S_0 = s_0] \right\}.$$
3. Thereafter, we solve
$$\hat{v}_t^n = \max_{a \in \mathscr{A}_t(s_t)} \left\{ C_t(s_t, a) + \mathbb{E}[\bar{V}_{t+1}^{n-1}(S_{t+1}) | S_t = s_t] \right\}$$
and continue iteratively until $t = T$.
4. Finally, we update $\bar{V}$ by letting
$$\bar{V}_t^n(s) = \begin{cases} \hat{v}_t^n, & \text{if } s = s_t \\ \bar{V}_t^{n-1}(s) & \text{otherwise.} \end{cases}$$

- Note that we still need to be able to compute the expectation (from the transition probabilities). This might be difficult (and for example for a robo running around in the world, infeasible and unwanted)
- We only update $\bar{V}$ for those states we visit. We therefore need to make sure that we are explorative enough to visit sufficiently many states
- We might get caught in a circle and a convergence proof is lacking.

Q-Learning

- The Q-learning ADP was proposed in Watkins[1] (an interesting read).
- The idea is again to approximate the value function. This time we look at the function $Q(s,a)$ which gives the value of action a when being in state s, i.e. we are looking for
$$Q : S \times a$$
- This gives an immediate hand on the optimal policy, $a^*(s) = \arg\max_a Q(s,a)$.
- Again, we proceed iteratively. The assumption we make is that once we chooce action a we observe the contribution $\hat{C}(S_t,a)$ and the next state S_{t+1}.
- We call an alogrithm **greedy**, if it bases its decision on the value function.
- Assume we are only interested in $V_0(\cdot)$.

[1] Christopher John Cornish Hellaby Watkins (1989). „Learning from delayed rewards“. Diss. King's College, Cambridge.

Q-Learning

- Start with an initial $\bar{Q}^0$.
- Suppose we are in step n and at position S^n. We choose action a^n greedy, i.e.
$$a^n := \arg \max_{a \in \mathscr{A}(S^n)} \bar{Q}^{n-1}(S^n, a).$$
- We observe $\hat{C}(S^n, a^n)$ and S^{n+1}.
- Compute
$$\hat{q}^n = \hat{C}(S^n, a^n) + \gamma \bar{Q}^{n-1}(S^{n+1}, a^n)$$
and update with **stepsize** or **learning rate** α_n:
$$\begin{aligned}\bar{Q}^n(S^n, a^n) &= (1-\alpha_n)\bar{Q}^{n-1}(S^n, a^n) + \alpha_{n-1}\hat{q}^n \\ &= \bar{Q}^{n-1}(S^n, a^n) + \alpha_n \Big(\hat{C}(S^n, a^n) + \gamma \bar{Q}^{n-1}(S^{n+1}, a^n) - \bar{Q}^{n-1}(S^n, a^n) \Big).\end{aligned}$$

Note that no expectation needs to be taken nor any model comes into play.

- A simple implementation just stores the values of Q in a table, which might be less efficient if the spaces get bigger.
- One possibility to solve this issue is to use an artificial network to learn this function (by the universal approximation theorem this is always possible), leading to "deep reinforcement learning" schemes, as proposed by DeepMind for playing Atari Games.
- Other variants concern speeding up the rates of convergence, as in its current form Q-Learning can be quite slow.

Implementations

- A variety of implementations are available:
- Car steering
  `http://blog.nycdatascience.com/student-works/capstone/reinforcement-learning-car/`
- a nice blog by Andrej Karpathy about the Atari game pong
  `http://karpathy.github.io/2016/05/31/rl/`
- The R package ReinforcementLearning from N Pröllochs (Freiburg!)