Stochastic Machine Learning
Chapter 11 - Reinforcement learning

Thorsten Schmidt

Abteilung für Mathematische Stochastik

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

SS 2024

Dynamic Approximate Programming

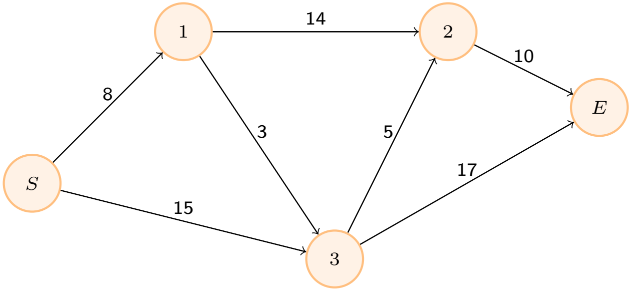
- From now on, we study the field of dynamic approximate programming (ADP) following Powell (2011)[1].
- As we already learned, there are many dialects in this field and we treat them here. This includes **reinforcment learning**, and a classic reference is Sutton & Barto[2]. For further references consider Powell (2011).
- The terminology "reinforcement" stems from behavioural sciences. A positive reinforcer is something that increases the probability of a preceding response (in contrast to a negative reinforcer, like an electronic shock), see also Watkins (1989).
- Examples are: moving a robot, investing in stocks, playing chess or go.
- The system contains four main elements: a **policy**, a **reward funciton**, a **value function** and (optional) a **model** of the environment.

[1] Powell (2011).

[2] Sutton and Barto (1998).

An Example

Let us start with a simple example.



It is our goal to find the shortest path from Start to End.

- By $\mathcal{I}$ we denote the set of intersections (S,1,...,E),
- if we are at intersection i we can go to $j \in \mathcal{I}_i^+$ at cost c_{ij},
- we start at S and end in E. Denote

  $$v_i := \text{cost from } i \text{ to } E$$

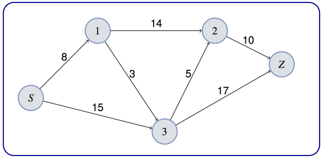
  and we could iterate

  $$v_i \leftarrow \min\left\{v_i, \min_{j \in \mathcal{I}_i}(c_{ij} + v_j)\right\}, \quad v_i \in \mathcal{I}$$

  and stop if the iteration does not change.

Iteration	S	1	2	3	E
1	∞	∞	∞	∞	0
2	∞	∞	10	15	0
3	30	18	10	15	0
4	26	18	10	15	0

- What is an efficient algorithm for solving this problem ?



- This is a **shortest-path problem** and therefore inherently difficult to solve.
- We can formulate this in the setting of MDPs (short recap)[3].

- We have a set $D_n(x)$ of possible actions at time n when the system is in state x. A decision rule at n is a measurable mapping f_n such that $f_n(x) \in D_n(x)$ for all $x \in E$.
- A **policy** is a collection of actions $\pi = (f_0, \dots, f_{N-1})$. We assume that the set of policies is non-empty.
- The dynamics of the model is specified via the transition kernel

$$Q_n(\cdot|x, s).$$

- Hence, the dynamics and with it the probability for evaluation depends on π. We denote

$$P^{\pi}_{n,x}(\cdot) := P^{\pi}(\cdot|X_n = x)$$

  and by $E^{\pi}_{n,x}$ the associated expectation.

[3] See Bäuerle and Rieder (2011) for details and further information.

- Our aim is to **maximize** the contribution given by the reward $r_n(x, a)$.
- Our goal is to aim at

$$\sup_\pi E^\pi\Big[\sum_{n=0}^N r_n(X_n, f_n(X_n))\Big],$$

where $r_N(x, a) = g_N(x)$ does not depend on a any more.

Remark

In general the supremum need not be measurable which causes a number of delicate problems, see Bertsekas and Shreve (2004) for a detailed treatment. The reason can be traced back to the fact that a projection of a Borel set need not be Borel (which leads to the fruitful notion of analytic sets, however).

- Define the value funciton

$$V_n(x) := \sup_\pi E^\pi\Big[\sum_{k=n}^N r_n(X_n, f_n(X_n) \,|\, X_n = x\Big]. \tag{1}$$

- Under the structural assumption, the **Bellman equation** holds, i.e.

$$V_n = \mathcal{T}_n V_{n+1},$$

with initial condition $V_N = r_N$ and

$$\mathcal{T}_n v(x) = \sup_{a \in D_n(x)} r_n(x, a) + \int v(x')\, Q_n(dx'|x, a).$$

Algorithm

Step 0 Initialize by the terminal condition $V_N(X_T)$ and set $n = N - 1$

Step 1 Compute

$$V_n(x) = \mathcal{T}_n V_{n+1}(x)$$

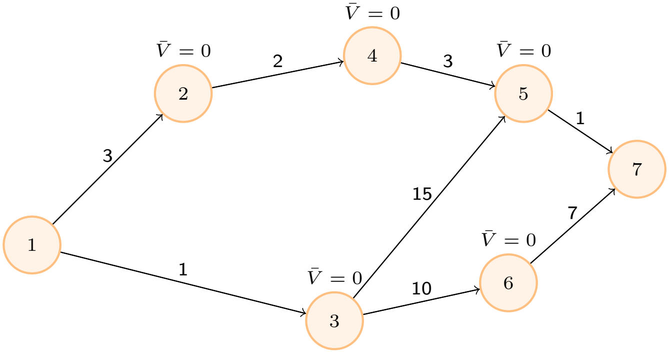
for all $x \in E$ (and possibly find the optimiser f^*)

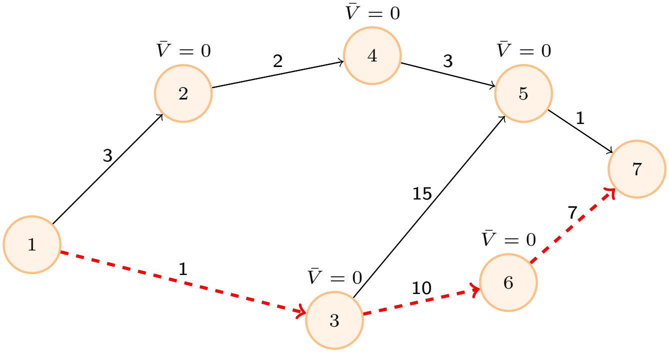
Step 2 Decrement n and repeat Step 1 until $t = 0$

Approximate dynamic programming (ADP)

- While we introduce a nice theory beforehand, the core equation
$$\sup_{\pi} E^{\pi}\left[\sum_{n01}^{N} r_n(X_n, f_n(X_n))\right]$$
my be intractable even for very small problems (higher than dimension 3!)
- 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 numbers at the arrows denote the associated **costs**. $\bar{V}$ hence is also the cost here. We go forward at the smallest cost.



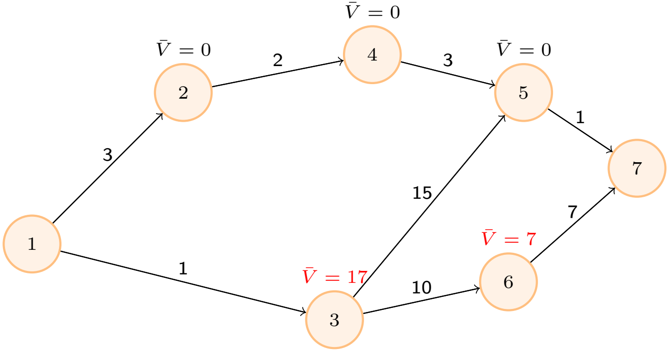


The approximation of the value function is **optimistic**: $\bar{V} = 0$ at all states. We start (forward !) in node $i = 1$ and choose the node j where

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

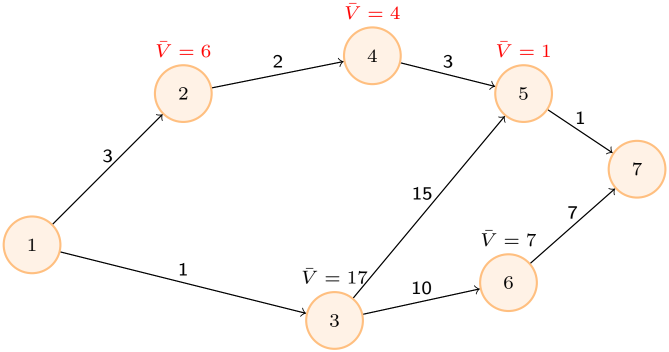
is minimal. This means we choose $1 \to 3 \to 6 \to 7$ and update V accordingly:

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



Now we take another round and go $1 \to 2 \to 4 \to 5 \to 7$, 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) or balancing a standing stick on a platform (or robot hand).

Approximate dynamic programming - 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

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

- Then we proceed iteratively.

Basic ADP algorithm

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

(i) simulate a path $X(\omega) =: (x_0, x_1, \dots, x_N)$.

(ii) at $n = 0$ we compute

$$\hat{v}_0^n = \mathcal{T}_0 \bar{V}_1^{k-1}$$

(iii) Thereafter, we solve (forward !)

$$\hat{v}_n^k = \mathcal{T}_n \bar{V}_{n+1}^{k-1}$$

and continue iteratively until $n = N - 1$.

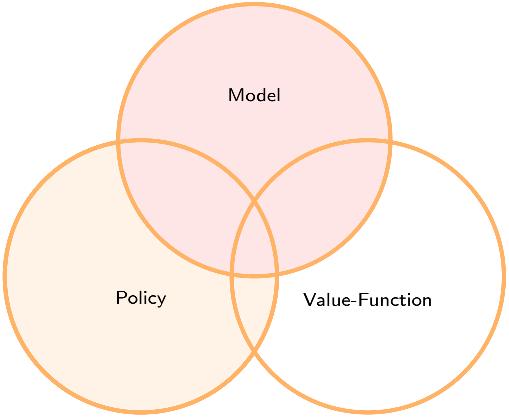
(iv) Finally, we update $\bar{V}$ by letting

$$\bar{V}_n^k(x) = \begin{cases} \hat{v}_n^k, & \text{if } x = x_t \\ \bar{V}_n^{k-1}(x) & \text{otherwise.} \end{cases}$$

- 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.

Overview

We have three ingredients: model, policy, value-function. Consequently, we have associated groups: model-free / model-based, value-based and policy-based.



- Due to the supremum, the Bellman equation is non-linear and a variety of methods for solving it exist:
- Value iteration
- Policy iteration
- Q-Learning
- SARSA

We start with Q-Learning, value and policy iteration typically apply to ∞-time horizon problems, but will be discussed shortly as well.

Q-Learning

A first model-free approach is the following:

- The Q-learning ADP was proposed in Watkins[4] (an interesting read).
- The idea is again to approximate the value function. This time we look at the function $Q(x,a)$ which gives the value of action a when being in state x, 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 choose action a we observe the reward $r(X_n, a)$ and the next state X_{n+1}.
- We call an algorithm **greedy**, if it bases its decision on the value function.
- Assume we are only interested in $V_0(\cdot)$.

[4]Watkins (1989).

Q-Learning

- Start with an initial $\bar{Q}^0$.
- Suppose we are in step k and at position x^k. We choose action a^k greedy, i.e.
$$a^k := \arg \max_{a \in D(x)} \bar{Q}^{k-1}(x^k, a).$$
- We observe $r(X^k, a^k)$ and X^{k+1}.
- Compute
$$\hat{q}^k = r(X^k, a^k) + \gamma \bar{Q}^{k-1}(x^{k+1}, a^k)$$
and update with **stepsize** or **learning rate** α_k:
$$\bar{Q}^k(X^k, a^k) = (1 - \alpha_k)\bar{Q}^{k-1}(X^k, a^k) + \alpha_{k-1}\hat{q}^k$$

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 Pr"ollochs (Freiburg!)[5]

[5] https://github.com/nproellochs/ReinforcementLearning

Value iteration

Very similar to backward dynamic programming we can iterate the value function to find an optimal solution.

- We start with $v = 0$,
- for each $x \in \mathcal{X}$ we set
$$V^k(x) = \max_{a \in \mathcal{A}} \left(r(x,a) + \gamma \, E_{x,a}[V^{k-1}(X)] \right)$$
- and stop if $\| V^k - V^{k-1} \|$ is sufficiently small.

See Powell, Section 3.10.3 for a proof of the convergence.

The magic of Reinforcement Learning

Its incredible performance in games[6]:

- RL play checkers **perfect**
- Backgammon, Scrabble, Poker **superhuman**.
- They play Chess and Go on the level of a Grandmaster

[6] See David Silvers lectures

They produce interesting behaviour and results.

- https://www.youtube.com/watch?v=CIF2SBVY-J0
- Implementation in R:
  http://www.rblog.uni-freiburg.de/2017/04/08/
  reinforcementlearning-a-package-for-replicating-human-behavior-in-r/

Bäuerle, N. and U. Rieder (2011). **Markov decision processes with applications to finance**.

Bertsekas, D. P. and S. Shreve (2004). **Stochastic optimal control: the discrete-time case**.

Powell, Warren B (2011). **Approximate Dynamic Programming: Solving the curses of dimensionality**. Vol. 703. John Wiley & Sons.

Sutton, R. S. and A. G. Barto (1998). **Reinforcement Learning : An Introduction**. MIT Press.

Watkins, Christopher John Cornish Hellaby (1989). „Learning from delayed rewards“. PhD thesis. King's College, Cambridge.