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

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

Dynamic Approximate Programming

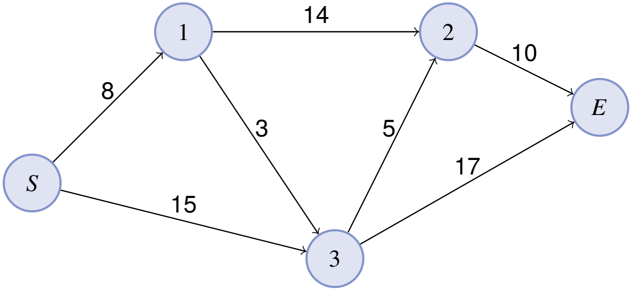
- From now on, we study the field of dynamic approximate programming (ADP) following Powell (2011)[1].
- As we already learned, there are many dialects in this field and we treat them here. This includes **reinforcment learning**, and a classic reference is Sutton & Barto[2]. For further references consider Powell (2011).
- The terminology "reinforcementßtems 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] Warren B Powell (2011). **Approximate Dynamic Programming: Solving the curses of dimensionality**. Bd. 703. John Wiley & Sons.

[2] R. S. Sutton und A. G. Barto (1998). **Reinforcement Learning : An Introduction**. MIT Press.

An Example

Let us start with a simple example.



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

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

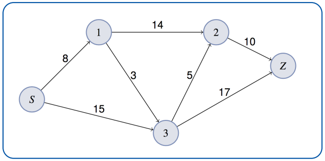
$$v_i := \text{cost from } i \text{ to } E$$

and we could iterate

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

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**. Let us introduce some notation for this. At time t, we start from a state S_t and can choose **action** a_t which leads to the transition to state S_{t+1} given by the **transition function** S, s.t.

$$S_{t+1} = S(S_t, a_t)$$

- Aditionally there is a **reward**, denoted by $C_t(S_t, a_t)$ and we define the value of being in state S_t by

$$V_t(S_t) = \max_{a_t} \{C_t(S_t, a_t) + V_{t+1}(S_{t+1})\}, \qquad S_t \in \mathscr{S}_t,$$

  cS_t denoting the possible states at time t.
- Let us visit some further examples.

Gambling

- Consider a gambler who plays T rounds, on an i.i.d. $(W_t)_{t=1,\dots,T}$ game with probability $p = \mathbb{P}(W_t = 1) > 1 - p$ of winning. We want to maximize $\mathbb{E}[\log(S_T)]$. It can be shown that it is optimal to proceed backwards in time using conditional expectations (this is dynamic programming)!
- Here, a_t is the amount he bets at t and we require $a_t \le S_{t-1}$. Then,

$$S_t = S_{t-1} + a_t W_t - a_t(1 - W_t).$$

- The value at time t, given his stock is in state S_t is

$$V_t(S_t) = \max_{0 \le a_{t+1} \le S_t} \mathbb{E}\big[V_{t+1}(S_{t+1}) | S_t\big].$$

- Now we proceed backwards. Clearly,

$$\begin{aligned} V_T(s) &= \log s \\ V_{T-1}(s) &= \max_{0\le a\le s} \mathbb{E}\big[V_T(s+aW_T-a(1-W_T))|S_{T-1}=s\big] \\ &= \max_{0\le a\le s} \Big(p\log(s+a)+(1-p)\log(s-a)\Big). \end{aligned}$$

- The maximum is attained for $a^*=(2p-1)s$ and $V_{T-1}(s)=\log(s)+K$, with costant $K=p\log(2p)+(1-p)\log(2(1-p))$. Backward in time we obtain

$$V_t(s)=\log S_t+K_t,$$

with an explicit constant K_t. Our **optimal policy** is

$$a_t=(2p-1)S_{t-1}.$$

The bandit problem

- When the distribution of the game is not known, one has to acquire information, and the classical example is the bandit problem. Consider a gambler who can choose betwee K machines.
- The probability of winning might be different and are **unknown** to us.
- A trade-off arises between playing only the optimal machine or trying other machines with (estimated) lower probability for minimizing the variance which is one-to-one to learning better their true probability.
- For a nite treatment, consider for example Richard Weber (1992). „On the Gittins Index for Multiarmed Bandits“. In: **Ann. Appl. Probab.** 2.4, S. 1024–1033.

Markov decision problems

- We give a short introduction into the field[3]. Assume that the state space $\mathscr{S}$ if **finite**.
- We have a set $\mathscr{A}_t(s)$ of possible actions at time t when the system is in state s. An action at t is a measurable mapping a_t such that $a_t(s) \in \mathscr{A}_t(s)$ for all $s \in \mathscr{S}$.
- A **policy** is a collection of actions $\pi = (a_0, \ldots, a_{T-1})$. We assume that the set of policies is non-empty.
- The dynamics of the model is specified via the (conditional) transition matrix
$$(p_t(s_{t+1}|s_t, a_t))_{s_{t+1}, s_t \in \mathscr{S}}$$
specifying $\mathbb{P}(S_{t+1} = s_{t+1}|S_t = s_t, a_t) = p_t(s_{t+1}|s_t, a_t)$.
- Hence, the dynamics and with it the probability for evaluation depends on π. We denote
$$\mathbb{P}^{\pi}_{t,s}(\cdot) := \mathbb{P}^{\pi}(\cdot|S_t = s)$$
and by $\mathbb{E}^{\pi}_{t,s}$ the associated expectation.

[3]See N. Bäuerle und U. Rieder (2011). **Markov decision processes with applications to finance**. for details and further information.

- Our aim is to **maximize** the contribution given by the functions $C_t(s,a)$ where $C_T(s,a) = C_T(s)$ does not depend on a. We additionally assume that the contribution is sufficiently integrable.
- Our goal is to aim at

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

For example, we could consider $C_t(s,a) = \gamma^t C(s,a)$ with possible discounting factor $\gamma > 0$.

The Bellman Equation

- The key to dynamic programming is that in our set-up, allowing the policy to depend on the full history does not improve the maximal expected reward, see Theorem 2.2.3. in Bäuerle&Rieder (2011).
- We define the **value function** by

$$V_t(s) = \sup_\pi \mathbb{E}^\pi_{t,s}\left[\sum_{s=t}^{T} C_t(S_t, a_t)\right].$$

Remark

In general V_t need not be measurable which causes a number of delicate problems, see D. P. Bertsekas und S. Shreve (2004). **Stochastic optimal control: the discrete-time case***. 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
$$C_t^*(s) := \sup_{a_t \in \mathscr{A}_t} \left(C_t(s,a_t) + \mathbb{E}\Big[V_{t+1}(S_{t+1}) | S_t = s, a_t \Big] \right) \tag{1}$$
Recall, that S_{t+1} also depends on $a_t = a_t(s)$ (which we suppress in the notation).
- The optimal policy can be computed backward by **reward iteration**. Let a_t^* be a maximizing policy, that is a_t^* achieves C_t^* in Equation (1).
- One can now show that the **Bellman equation** holds, i.e.
$$V_t(s) = C_t^*(s) \quad t = 0, \dots, T.$$
- Under an additional (mild) structural assumption, one may verify that there always exist optimal policies π^* which can be obtained by maximizing the value function in each period (Theorem 2.3.8. in Bäuerle Rieder).

Algorithm

Step 0 Initialize by the terminal condition $V_T(S_T)$ and set $t = T - 1$

Step 1 Compute

$$V_t(s) = \sup_{a_t \in \mathscr{A}_t} \left(C_t(s, a_t) + \mathbb{E}\left[V_{t+1}(S_{t+1}) | S_t = s, a_t \right] \right)$$

for all $s \in \mathscr{S}$

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

Infinite-time-horizon

- For this case several algorithms exist, to name **value iteration** and **policy iteration** which will not be discussed here, see Powell Section 3.3. ff.
- For more mathematical details (and there are many!) we refer to Powell, Bäuerle&Rieder and the excellent source Bertsekas&Shreve.

Approximate dynamic programming (ADP)

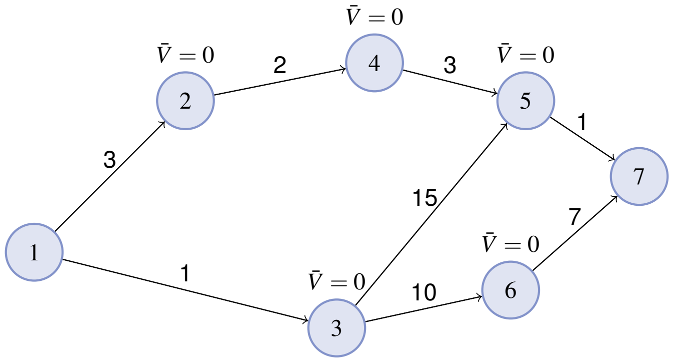
- While we introduce a nice theory beforehand, the core equation

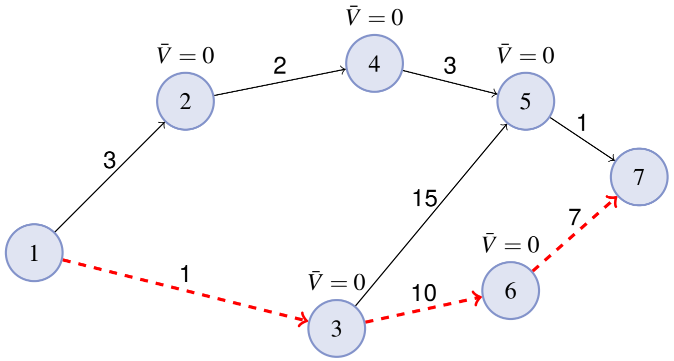
$$\sup_{\pi} \mathbb{E}^{\pi}\left[\sum_{0=1}^{T} C_t(S_t, a_t)\right]$$

  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.



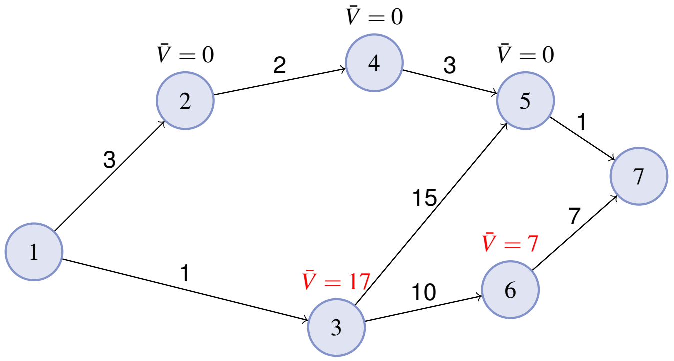


The approximation of the value function is **optimistic**: $\bar V = 0$ at all states. We start (forward !) in node 1 and choose the node where

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

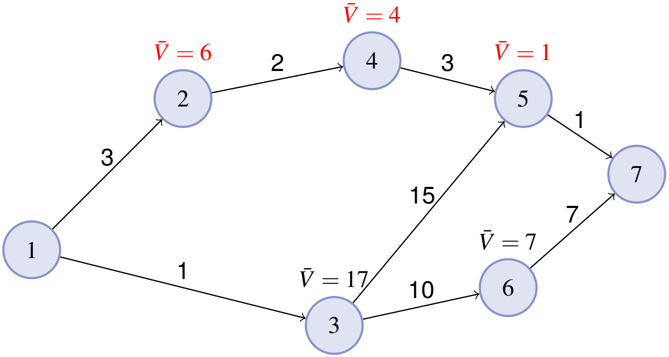
is minimal. This means we choose $1-3-6-7$ and update (!) accordingly:

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



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

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

- 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

$$\hat{v}_t^n = \max_{a \in \mathscr{A}_t(s_t)} \left\{ C_t(s_t, a) + \mathbb{E}[\bar{V}_{t+1}^{n-1}(S_{t+1}) | S_t = s_t] \right\}$$

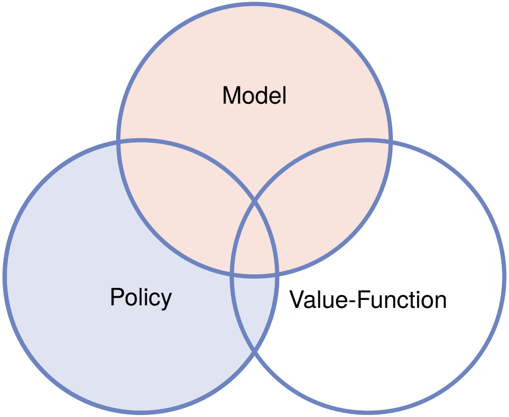
   and continue iteratively until $t = T$.
4. Finally, we update $\bar{V}$ by letting

$$\bar{V}_t^n(s) = \begin{cases} \hat{v}_t^n, & \text{if } s = s_t \\ \bar{V}_t^{n-1}(s) & \text{otherwise.} \end{cases}$$

- 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.

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 $Q(s,a)$ 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 $V_0(\cdot)$.

[4] 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!)[5]

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

SARSA

- SARSA is an algorithm to **estimate** the value of a fixed policy, which will be important, for example, for policy iteration.
- The name stems from the following acronym: suppose we are in state s, take action a, observe afterwards the reward r, go to state s' and take action a'.
- More precisely, we estimate the value of the policy π by iterating (infinitely often for convergence) as follows: the policy comes with the rule $A^\pi(s)$ of choosing an action. So starting from state S^n, we choose $a^n = A^\pi(S^n)$.
- Given our transition law, we simulate S^{n+1}. Naturally, then we choose $a^{n+1} = A^\pi(S^{n+1})$ and approximate the value of being in state S^n and taking action a^n by the one-step prediction

$$\hat{q}^n(S^n, a^n) = C(S^n, a^n) + \gamma \bar{Q}^{n-1}(S^{n+1}, a^{n+1}).$$

- Then, we use q^n to update $\bar{Q}^{n-1}$, just as in Q-learning.

Infinite-Horizon Problems

Starting from the Bellman Equation

$$V_t(s) = \sup_{a_t \in \mathscr{A}_t} \left(C_t(s,a_t) + \mathbb{E}\Big[V_{t+1}(S_{t+1}) | S_t = s, a_t \Big] \right)$$

we would intuitively arrive at a infinity-time horizon formulation

$$V(s) = \sup_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t^\pi(S_t)) \right].$$

Define

$$P^{\pi,t} = \prod_{s=0}^{t-1} P^\pi$$

where P^π is the one-step transition matrix given policy π. Then

$$\begin{aligned} V_0^\pi(s) &= \sum_{u=0}^{\infty} \gamma^{u-t} P^{\pi,u-t} C_t(s, A^\pi(s)) \\ &= C_0^\pi(s) + \gamma P^\pi \sum_{u=1}^{\infty} \gamma^{u-t} P^{\pi,u-t} C_t(s, A^\pi(s)) \\ &= C_0^\pi(s) + \gamma P^\pi V_0^\pi(s). \end{aligned}$$

In simple cases this equation can be solved and we arrive at

$$V^\pi = (1 - \gamma P^\pi)^{-1} C_0^\pi. \tag{2}$$

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 $s \in \mathscr{S}$ we set

$$V^n(s) = \max_{a \in \mathscr{A}} \left(C(s,a) + \gamma \mathbb{E}_{s,a}[V^{n-1}(S)] \right)$$

- and stop if $\| V^n - V^{n-1} \|$ is sufficiently small.

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

Policy iteration

While the value equation starts from the Bellman equation, policy iteration starts from Equation (2),

$$V^\pi = (1-\gamma P^\pi)^{-1} C_0^\pi.$$

- Starting with a policy π^0.
- Set $\pi' = \pi^{n-1}$. We compute $c^{n-1} = C_0^{\pi'}(s, A^{\pi'})$ and solve

  $$(1-\gamma P^{\pi'})V^{\pi'} = c^{n-1}$$

  fir $V = V'$. The policy π^n is the solution of

  $$\arg\max_{a \in \mathscr{A}} \big(C(a) + \gamma P^\pi V' \big).$$

- Iterate until convergence.

Exploration-Expectation

In the search for an optimal policy, we may not always meet all possible states. One way out of this is to use a **randomized** strategy, which is called ε-greedy.

- This strategy chooses with probability $1-\varepsilon$ the optimal strategy and
- with probability ε a random strategy which allows us to explore.

Proof of the Bellman equation

Approximating the value function for fixed π

- As already illustrated it is important to approximate the value function efficiently.
- We will always denote by $\hat{V}$ the approximation (estimation) of our value function.
- If we have a state S^n and the associated estimate $\hat{V}^n$ at hand, we can update **or** estimate the $\hat{V}^{n+1}$, depending on what method we choose (again see Powell for numerous such approaches).
- We illustrate this once more: for N times we simulate as follows
  1. Simulate S_0^n, for $t = 0, \ldots, T$ choose $a_t^\pi = A^\pi(S_t^n)$, and simulate S_{t+1}^n depending on a_t^n
  2. Given this, we compute $\hat{V}^n = \sum_{t=0}^T \gamma^t C(S_t^n, a_t^n)$
  3. Finally we use $(S^n, \hat{V}^n)_{n=1,\ldots,N}$ to fit $\bar{V}^\pi$

Temporal Differences

A very well-known approach in this regard are **temporal differences**. We choose $\gamma = 1$ first. Clearly

$$\begin{aligned}\hat{V}_t^n &= \sum_{u=t}^{T} C(S_u^n, a_u^n) \\ &= \sum_{u=t}^{T} \Big(C(S_u^n, a_u^n) - (\bar{V}_u^{n-1}(S_u) - \bar{V}_{u+1}^{n-1}(S_{u+1})) \Big) + \bar{V}_t^{n-1}(S_t) - \bar{V}_{T+1}^{n-1}(S_{T+1})\end{aligned}$$

We use the freedom to set $V_{T+1} = 0$ and obtain the nice representation

$$\hat{V}_t^n = \bar{V}_t^{n-1}(S_t) + \sum_{u=t}^{T} \delta_u^\pi \tag{3}$$

with temporal differences

$$\delta_t^\pi = C(S_t^n, a_t^n) + \bar{V}_{t+1}^{n-1}(S_{t+1}) - \bar{V}_t^{n-1}(S_t).$$

Using a stochastic gradient algorithm leads to $\bar{V}_t^n(S_t^n) = \bar{V}_t^{n-1}(S_t^n) - \alpha_n(\bar{V}_t^{n-1}(S_t^n) - \hat{V}_t^n)$ and we arrive (including discounting now and a time-discount λ) at

$$\bar{V}_t^n(S_t) = \bar{V}_t^{n-1}(S_t) + \alpha_{n-1} \sum_{u=t}^{T} (\gamma\lambda)^{u-t} \delta_u^\pi .$$

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/`