Künstliche Intelligenz

Reinforcement Learning

Dr.-Ing. Stefan Lüdtke

Universität Leipzig

Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI)

Motivation

We know how to solve a MDP in case we know

- The state space $\mathcal{S}$
- The action set $\mathcal{A}$
- The transition model $\mathcal{P}^a_{ss'}$
- The expected rewards $\mathcal{R}^a_{ss'}$

But what do we do in case we're thrown into an unknown world and we have to *learn* these models from observation?

- Batch learning

And can we even learn $Q^\star(s, a)$ *without* having to build an explicit model?

- Reinforcement learning

Batch Learning

Assume that there is a sequence of transition samples $(s, a, r, s')_{1:N}$. A transition sample is also called *experience*. Clearly, we can use a sequence of experiences to create an empirical model:

$$\hat{\mathcal{S}} := \{s_i\} \cup \{s'_i\} \tag{1}$$

$$\hat{\mathcal{A}} := \{a_i\} \tag{2}$$

$$\hat{\mathcal{P}}^a_{ss'} := \frac{N^a_{ss'}}{N^a_s} \tag{3}$$

$$\hat{\mathcal{R}}^a_{ss'} := \frac{1}{N^a_{ss'}} \sum_{i=1}^{N} r_i \left[s_i = s \wedge a_i = a \wedge s'_i = s'\right] \tag{4}$$

$$\text{where } N^a_{ss'} := \sum_{i=1}^{N} [s_i = s \wedge a_i = a \wedge s'_i = s'] \tag{5}$$

$$N^a_s := \sum_{s' \in \hat{\mathcal{S}}} N^a_{ss'} \tag{6}$$

We can then use this empirical model for value or policy iteration.

Batch Learning

- The samples $(s, a, r, s')_{1:N}$ can be generated by an arbitrary policy, as long as this policy guarantees that, as $N \to \infty$, every action will be tried infinitely often in any state.
- Note that for trivial problems where transition model and reward structure are deterministic, only one sample is required for each combination of a and s (each sample representing an edge in the transition graph), and value iteration becomes the Bellman-Ford algorithm.

Example: Light Control

Consider a conference room with two major lights – stage and audience. The job is now to develop a light control system that learns to set the light according to the preferences of the room's users and, at the same time, learns to save energy, turing off lights if not needed.

The room state is described by six binary state variables:

- Room occupied / empty
- Presentation going on (yes/no)
- Room blinds up / down
- Daylight available / night
- Stage light on / off
- Audience light on / off

In addition, there are four control actions available:

- Both lights off
- Stage on, audience off
- Stage off, audience on
- Both lights on

Example: Light Control

Operation:

- Once per minute, the state is reported to the controller, and the controller is then required to select one of the four actions in response.
- If the light situation selected by the controller does not satisfy the room users, they will manually switch the light according to their preferences.

The reward is defined as follows:

- Each light turned on by the controller will cost 1 unit for energy consumption
- Each manual interaction by the users will cost 50 units for lack of satisfaction

So, the maximum negative reward is -102 (both lights turned on by the system, costing 2 units for energy, and both lights manually turned of by the users since they did not want any light).

Example: Light Control

Atomic States and Actions

A conversation between environment and controller will look like this:

Environment: "State is 45. Action?"
Controller: "7"
Environment: "State is 17, reward -5. Action?"
Controller: "0"
Environment: "State is 42, reward 12. Action?"
...

- States and actions are atomic.
- There is no "structure" of states or "similarity" that a learning system can used to transfer knowledge concerning one state to another state.
- This means: lots of experience is required.

What does "lots" mean?

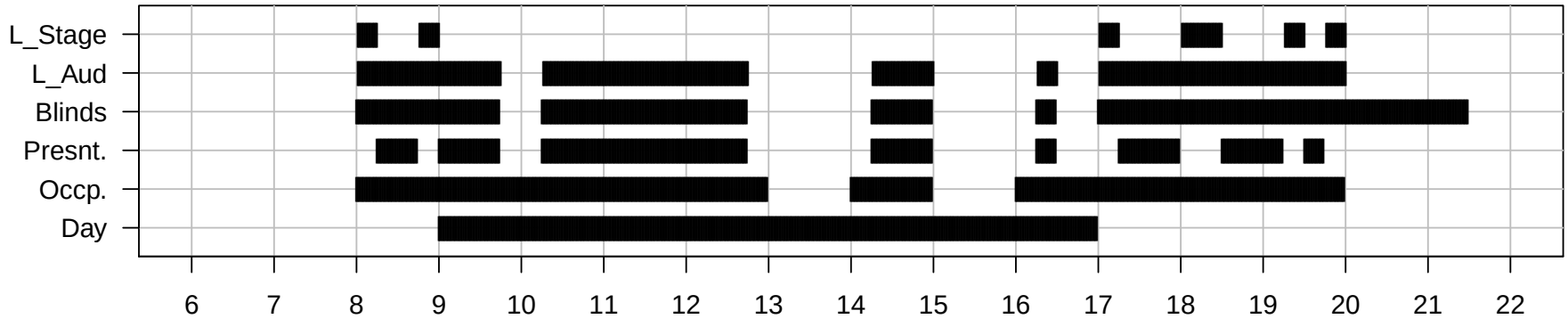
Example: Light Control: Simulation Study

- A simulation consists of a number of simulation days.
- Each simulation day consist of 24 slots of 1 hour duration.
- From 9:00 to 17:00 daylight is available
- From 8:00 to 20:00 the room may be occupied. The probability for the room being occupied for a 1 hour slot is 75%.
- A one hour slot is divided into four quarter hour slices. If the room is occupied, each slice has a probability of 60% for being used for presentation.

Example: Light Control: Simulation Study

- At each start of a slice, the room blinds will be adjusted:
  - If the room is in use and it is daylight, blinds will be closed if a presentation going on, otherwise they will be opened.
  - If the room is in use and it is night, there is a 60% chance that the blinds will be lowered (for privacy?)
  - If the room is empty, blinds will always be opened at day. At night, closed blinds will have a 30% chance of being opened.
- Then lighting is set according to the following rules:
  - Audience light is turned on if the room is occupied and if either the blinds are down or there is no daylight. Otherwise, it will be turned off.
  - Stage light will be turned on if audience light is on and if there is no presentation going on.

Example: Light Control: Simulation Study



6:00 to 22:00 excerpt from a sample simulation day.
“Lunch break" at 13:00 and “coffee break" at 15:00 are disconcerting coincidences between the simple simulation model (which has no notion of such breaks) and everyday reality created by pure chance.

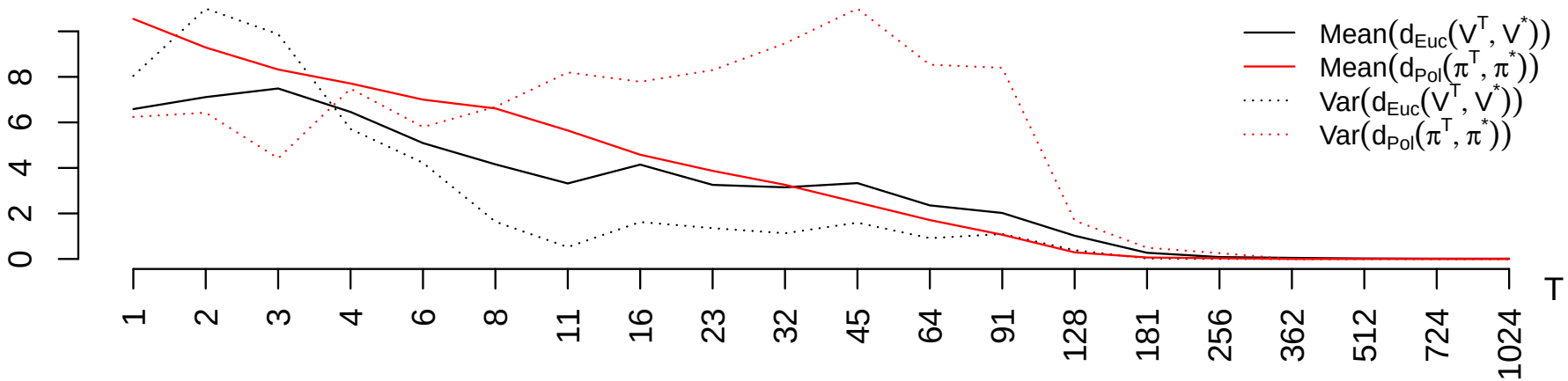
Example: Light Control: Simulation Study

Obtaining Samples

- Observe the current simulation state s_t
- select a random action a_t
- send this to the simulation and get back the reward r_t and the new state s_{t+1}.
- Collect the quadruple (s_t, a_t, r_t, s_{t+1}) as transition sample.
- Run the sampling process from time 1 to time T and …
- … use the resulting sequence of transition samples for building an empirical model …
- … which is then solved by value iteration.

Example: Light Control: Simulation Study

Results



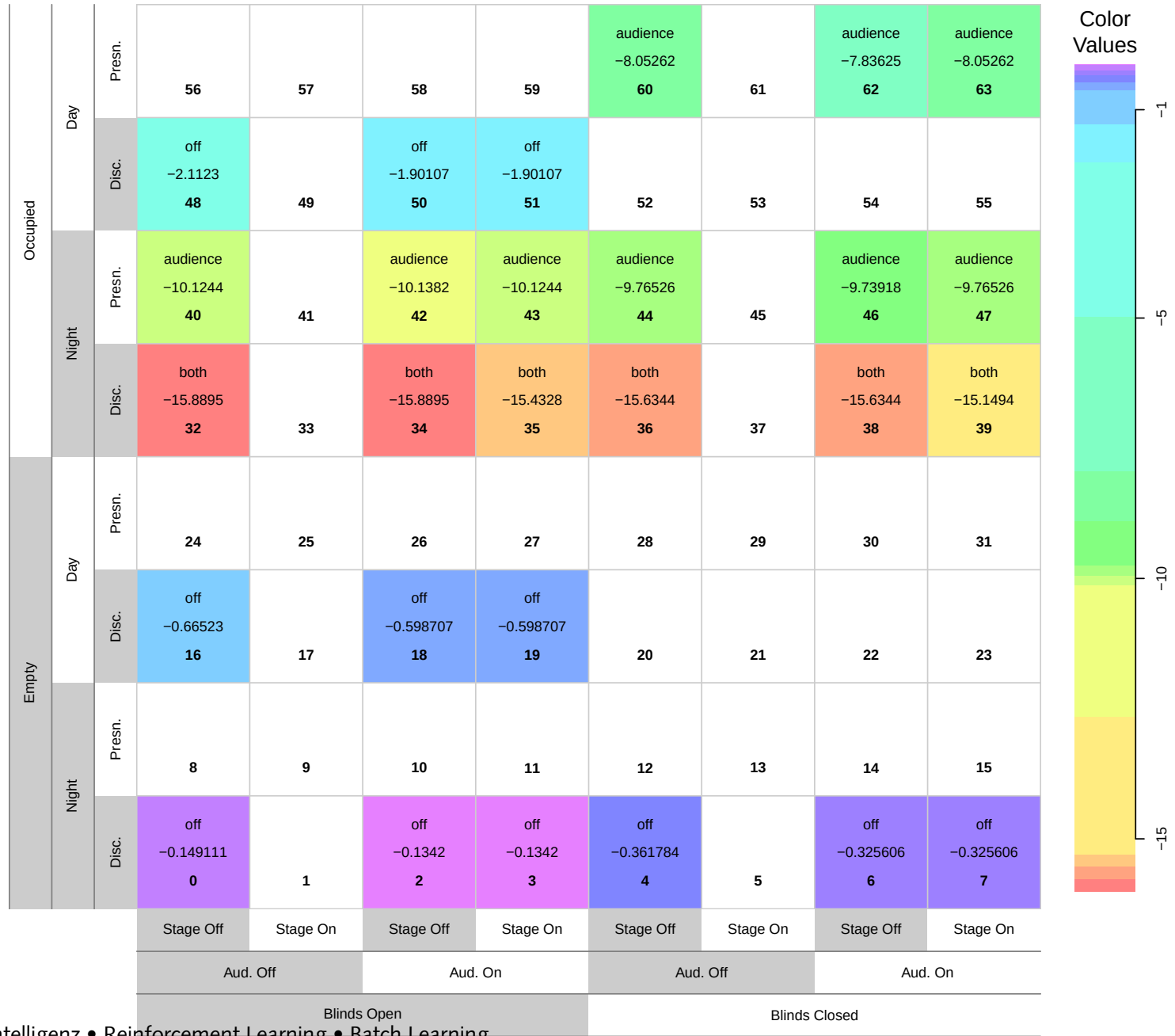
Note: variance plots not to scale, scale of T axis is logarithmic

$$d_{Pol}(\pi^T, \pi^\star) = \sum_s [\pi^T(s) \neq \pi^\star(s)] \tag{7}$$

$$d_{Euc}(V^T, V^\star) = \sqrt{\sum_s \left(V^T(s) - V^\star(s)\right)^2}. \tag{8}$$

It takes 181 simulation days, corresponding to 260640 experiences to reach the region of convergence, even for such a simple model.

Example: Light Control: Simulation Study: $V^\star$ and $\pi^\star$



Space Complexity

- Space complexity of batch learning: $O(N^2 M)$ where $N = |\hat{\mathcal{S}}|$ and $M = |\hat{\mathcal{A}}|$.
- Space complexity of Q is $O(NM)$
- Is there a strategy for directly estimating Q, without having to build up an explicit estimate of the model?

Q Learning

Consider the Bellman optimality equation for $Q^\star$:

$$Q^\star(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^\star(s',a') \right)$$

Now consider an experience $(s, a, r, \tilde{s}')$. Using this tuple, we can first build very coarse approximations of $\mathcal{P}^a_{ss'} = p(s' \mid s, a)$ and $\mathcal{R}^a_{ss'}$ by defining

$$\mathcal{P}^a_{ss'} \approx [s' = \tilde{s}'] \tag{9}$$

$$\mathcal{R}^a_{s\tilde{s}'} \approx r \tag{10}$$

entering this into the equations above gives

$$Q^\star(s,a) \approx \sum_{s'} [s' = \tilde{s}'] \big(r + \gamma \max_{a'} Q^\star(s',a') \big) \tag{11}$$

Q Learning

$$Q^\star(s,a) \approx \sum_{s'} [s' = \tilde{s}'] \big(r + \gamma \max_{a'} Q^\star(s', a')\big)$$
$$= r + \gamma \max_{a'} Q^\star(\tilde{s}', a') \tag{12}$$

Where this quantity is sometimes known as *target*.
Due to approximations, this target value may contain errors. But imagine, $Q_{i-1}(s,a)$ is a previous approximation. From this approximation and the target created from experience (s,a,r,s') we can create an updated approximation by using a weighted average of both values:

$$Q_{i+1}(s,a) \leftarrow (1-\alpha) Q_i(s,a) + \alpha \big(r + \gamma \max_{a'} Q_i(s', a')\big) \tag{13}$$

Q Learning

Definition 1 (Q Learning)

$$Q_{i+1}(s,a) \leftarrow (1-\alpha)Q_i(s,a) + \alpha\big(r + \gamma \max_{a'} Q_i(s',a')\big) \tag{14}$$

Sometimes, this is written as

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha\big(r + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)\big) \tag{15}$$

This version emphasizes the idea that the new action value is the old value plus a move of step size α in the direction of the *difference* between the *old* value and the target created by the *new experience* (i.e., we move the action value gradually to the target using the temporal difference).

Q Learning

Definition 2 (Learning rate)

In equations (14) and (15), the value α, where $0 \leq \alpha \leq 1$ is the *learning rate*.

Proposition 3 (Convergence of Q learning)

If α is appropriately adjusted over time such that $\lim_{t \to \infty} \alpha = 0$, the Q learning algorithm will converge to $Q^\star$.

Note 4

Finding the right adjustment schedule is problem dependent. Fortunately, $\pi^\star$ will (usually) be discovered long before $Q^\star$ has been reached.

Action Selection

- Q-learning has the interesting property that it will converge to the true optimal policy (and the true optimal action value function), regardless of policy, as long as the policy will try every action infinitely often in any state.
- However, in order to optimize reward, it seems advisable to start using actions with high value as soon as possible. This is know as the *exploration–exploitation dilemma*: if one starts to greedily use the currently optimal moves, action options leading to maybe even better results will never be explored.

Epsilon-greedy Action Selection

One simple solution is to *always* reserve a probability of ϵ (e.g., $\epsilon = 0.01$) for experimentally trying a non-optimal action. $\pi(s, a)$ can in this case be defined by:

$$\pi(s,a) = \begin{cases} 1-\epsilon & \text{if } a \text{ optimal} \\ \epsilon/(n-1) & \text{otherwise} \end{cases} \tag{16}$$

Where $n > 1$ is the number of actions available in state s (if $n = 1$ there is obviously nothing to explore).

Other RL Algorithms

- Explicitly maintaining the Q-Function as a table becomes infeasible for problems with large (or continuous) state or action spaces
  - Can use a discretization of states and/or actions
  - Or, use a function approximator for Q, e.g. a deep neural network ("deep Q learning")
- Q-Learning is just one, simple example of reinforcement learning algorithms, other algorithms differ w.r.t.
  - Number of steps taken before updating Q
  - Policy used for generating experiences ("on-policy" vs. "off-policy" RL)
  - Learning Q vs. directly learning π
  - ...

Summary

- RL algorithms allow an agent to learn taking actions in an unknown environment by learning from experience
- Challenging due to large state and action spaces, sparse rewards
- Active and exciting field of research
- We regularly offer team projects and master theses on applying RL to solve real-world problems