Praktikum Mobile und Verteilte Systeme

Decision Making in Autonomous Systems

Prof. Dr. Claudia Linnhoff-Popien
André Ebert, Thomy Phan, Robert Müller, Steffen Illium
http://www.mobile.ifi.lmu.de

WiSe 2018/19
Outline

• An Introduction to Autonomous Systems
  – Motivation, Definition and Challenges
  – Artificial Intelligence

• Decision Making in Autonomous Systems
  – Markov Decision Processes
  – Planning
  – Reinforcement Learning

• Applications and Further Challenges
  – Deep Reinforcement Learning
  – Combining Planning and Reinforcement Learning
  – Further Challenges
Short Recap
Artificial Intelligence

- Learn
  - Machine Learning
  - Reinforcement Learning
- Think
  - Pattern Recognition
  - Scheduling
- Act
  - Decision Making
  - Social Interactivity
  - Multi-Agent Systems
- Planning
Artificial Intelligence

Learn --> Think

- Pattern Recognition
- Machine Learning
- Reinforcement Learning
- Decision Making
- Multi-Agent Systems

Act --> Think
- Planning
- Social Interactivity

Learn --> Act
- Planning
- Social Interactivity
- Multi-Agent Systems
Outline

• An Introduction to Autonomous Systems
  – Motivation, Definition and Challenges
  – Artificial Intelligence

• Decision Making in Autonomous Systems
  – Markov Decision Processes
  – Planning
  – Reinforcement Learning

• Applications and Further Challenges
  – Deep Reinforcement Learning
  – Combining Planning and Reinforcement Learning
  – Further Challenges
→ Decision Making Problem
Decision Making

agent

action

reward, observation

environment
Decision Making

- **Goal:** Select actions to solve a complex task
- Focus on **Sequential Decision Making:** Time matters!
- Actions have possible consequences in the long run
- Feedback/Reward may be delayed
Why Decision Making?

- High Complexity (autonomous systems instead of hand-coded solutions)
- Risk and Safety (autonomous systems instead of living beings)
- Dynamics (adaptation to changes)

**Examples:**

- *Industry 4.0*
- *Search and Rescue*

Markov Decision Processes
Definition

- A Markov Decision Process (MDP) is defined as $M = \langle S, A, P, R \rangle$:
  - $S$ is a (finite) set of states
  - $A$ is a (finite) set of actions
  - $P(s_{t+1}|s_t, a_t) \in [0, 1]$ is the probability for reaching $s_{t+1} \in S$ when executing $a_t \in A$ in $s_t \in S$
  - $R(s_t, a_t) \in \mathbb{R}$ is a reward function
Markov Decision Processes

- MDPs formally describe environments for Sequential Decision Making
- All states $s_t \in S$ are Markov such that
  $$\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \ldots, s_t)$$ (no history of past states required)
- Assumes full observability of the state
- States and actions may be discrete or continuous
- Many problems can be formulated as MDPs!
Examples

• Board Games
  – Board positions represent states $s_t \in S$
  – Moves represent actions $a_t \in A$
  – Game dynamics (opponents, rules, etc.) represent $\mathcal{P}(s_{t+1}|s_t, a_t)$
  – $\mathcal{R}(s_t, a_t)$ defines when a game is won, lost or a draw.

• Navigation Task
  – Agent‘s and other object‘s positions represent the state $s_t \in S$
  – Agent moves represent actions $a_t \in A$
  – Environment dynamics represent $\mathcal{P}(s_{t+1}|s_t, a_t)$
  – $\mathcal{R}(s_t, a_t)$ defines when a target is reached
Policies and Value Functions

• A **policy** $\pi: S \rightarrow A$ represents the behavioural strategy of an agent
  – Policies may also be stochastic $\pi(a_t | s_t) \in [0,1]$

• The **return** of a state $s_t \in S$ for a horizon $h$ given a policy $\pi$ is the cumulative (discounted) future reward ($h$ may be infinite!):
  $$G_t = \sum_{k=0}^{h-1} \gamma^k R(s_{t+k}, \pi(s_{t+k})), \gamma \in [0,1]$$

• The **value** of a state $s_t \in S$ is the expected return of $s_t$ for a horizon $h \in \mathbb{N}$ given a policy $\pi$:
  $$V^\pi(s_t) = \mathbb{E}[G_t | s_t]$$

• The **action value** of a state $s_t \in S$ and action $a_t \in A$ is the expected return of executing $a_t$ in $s_t$ for a horizon $h \in \mathbb{N}$ given a policy $\pi$:
  $$Q^\pi(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$$
Policies and Value Functions

- The policy $\pi$ describes **what** an agent should do in a certain state.
  - Recommendation of actions

- The state value function $V^\pi$ describes **how well** an agent will perform in a **certain state** (with policy $\pi$).
  - State evaluation

- The action value function $Q^\pi$ describes **how well** agent will perform when executing a **certain action** in a certain state (with subsequent policy $\pi$).
  - Action evaluation
Policies and Value Functions

• **Goal:** Find an *optimal policy* $\pi^*$ which maximizes the expected return for each state:

$$\pi^* = \arg\max_\pi V^\pi(s_t), \forall s_t \in S$$

• The *optimal value function* is defined by:

$$V^*(s_t) = V^{\pi^*}(s_t) = \max_\pi V^\pi(s_t)$$

$$Q^*(s_t, a_t) = Q^{\pi^*}(s_t, a_t) = \max_\pi Q^\pi(s_t, a_t)$$

• When $V^*$ or $Q^*$ is known, $\pi^*$ can be derived.
→ Planning
Planning

- **Goal:** Find an (near-)optimal policy to solve complex problems

- Use (heuristic) lookahead search on a **given model** of the problem
  - Model can be defined by rules (e.g. physical laws, game rules, etc.) or statistical approximations (e.g. using machine learning)
  - Represented by $P(s_{t+1}|s_t, a_t)$ in the context of MDPs
Global Planning

- Global Planning considers the entire state space $\mathcal{S}$ to approximate $\pi^*$
- Produces for each state $s_t \in \mathcal{S}$ a mapping to actions $a_t \in \mathcal{A}$
- Typically performed offline (before deploying the agent)
Example: Value Iteration

- Starting with an initial guess $V^0$
- Refine approximation $V^n$ for each state $s_t \in S$ according to

$$V^{n+1}(s_t) = \max_{a_t \in A}\{R(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} \mathcal{P}(s_{t+1} | s_t, a_t)V^n(s_{t+1})\}$$

- Based on Bellman's "principle of optimality"
- Converges to the optimal value function $V^*$

- **Optimal policy** $\pi^*$ can be derived from $V^*$

$$\pi^*(s_t) = \arg\max_{a_t \in A}\{R(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} \mathcal{P}(s_{t+1} | s_t, a_t)V^*(s_{t+1})\}$$

- Requires explicit model $\mathcal{P}(s_{t+1} | s_t, a_t)$!
Example: Value Iteration

• Simple navigation problem:
  – 3 possible positions / states in $S$
  – Possible actions $\mathcal{A}$: move left/right
  – State transitions are deterministic
  – Reward $\mathcal{R}(s_t, a_t)$:
    • +1, if flag is reached (terminal state)
    • -1, if agent bumped against wall
    • 0 otherwise
  – Discount factor $\gamma = 0.95$

• Optimal value (function) map is $V^*$:

\[
\begin{array}{ccc}
0.9025 & 0.95 & 1 \\
\gamma^2(+1) & \gamma^1(+1) & \gamma^0(+1)
\end{array}
\]
Local Planning

- Local Planning only considers the **current state** $s_t \in S$ (and possible future states) to approximate $\pi^*(s_t)$
- Can be performed **online** (interleaving planning and execution)

- Typically constrained **time** and **computation budget** (overall performance depends on available resources)
Example: Monte Carlo Planning

- Explicit model $\mathcal{P}(s_{t+1}|s_t, a_t)$ usually unknown for many MDPs
- **Monte Carlo Planning**: uses a generative model $\hat{\mathcal{M}} \approx \mathcal{M}$
  - $\hat{\mathcal{M}}$ provides a sample $\langle s_{t+1}, r_t \rangle \sim \hat{\mathcal{M}}(s_t, a_t)$ for $s_t \in S$ and $a_t \in \mathcal{A}$
  - Can be used as black box simulator to approximate $V^*$ or $Q^*$
  - Approximate planning via statistical sampling
  - Requires minimal domain knowledge ($\hat{\mathcal{M}}$ can be easily replaced)

- **Example**: Monte Carlo Tree Search

![Monte Carlo Tree Search Diagram]

Selection  \hspace{2cm} Expansion  \hspace{2cm} Evaluation/Simulation  \hspace{2cm} Backup

Prof. Dr. C. Linnhoff-Popien, André Ebert, Thomy Phan, Robert Müller, Steffen Illium - Praktikum Mobile und Verteilte Systeme
WiSe 2018/19, Decision Making in Autonomous Systems
→ Reinforcement Learning
Reinforcement Learning

- **Goal:** Find an (near-)optimal policy to solve complex problems

- Learn via trial-error from (real) experience:
  - Model $\mathcal{P}(s_{t+1} | s_t, a_t)$ is unknown
  - **Experience samples** $e_t = (s_t, a_t, r_t, s_{t+1})$ are generated by interacting with a (real or simulated) environment
  - To obtain sufficient experience samples one has to balance between **exploration** and **exploitation** of actions
Model-Free Reinforcement Learning

• Learn policy and/or value function **directly** from experience samples
• Experience samples $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$ are generated by interacting with a (real or simulated) environment
• Approximate $\pi^*, V^*$ and/or $Q^*$ with experience samples

• **Example:** Temporal Difference (TD) Learning
  – Recap: Value Iteration – iteratively refine guess $V^n$ by using a model

\[
V^{n+1}(s_t) = \max_{a_t \in \mathcal{A}} \{r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1} | s_t, a_t) V^n(s_{t+1})\}
\]

  – TD-Learning: regression on TD targets $y_t$ generated from experience

\[
y_t = \hat{V}^{n+1}(s_t) = r_t + \gamma \hat{V}^n(s_{t+1})
\]
Example: Q-Learning

• Deriving a policy $\hat{\pi}$ from $\hat{V}$ still requires a model:

$$\hat{\pi}(s_t) = \arg\max_{a_t \in \mathcal{A}} \left\{ r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) \hat{V}(s_{t+1}) \right\}$$

• **Solution:** approximate $Q^*$ instead!
  
  – Regression on TD targets $y_t$ generated with $e_t = (s_t, a_t, r_t, s_{t+1})$

$$y_t = \hat{Q}^{n+1}(s_t, a_t) = r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} \hat{Q}^n(s_{t+1}, a_{t+1})$$

  – Derive a policy $\hat{\pi}$ from $\hat{Q}$

$$\hat{\pi}(s_t) = \arg\max_{a_t \in \mathcal{A}} (\hat{Q}(s_t, a_t))$$

  – **Challenge:** obtain sufficient samples $e_t = (s_t, a_t, r_t, s_{t+1})$ by addressing the exploration-exploitation dilemma.
Example: Q-Learning

- Simple navigation problem:
  - 3 possible positions / states in $S$
  - Possible actions $A$: move left/right
  - State transitions are deterministic
  - Reward $R(s_t, a_t)$:
    - +1, if flag is reached (terminal state)
    - -1, if agent bumped against wall
    - 0 otherwise
  - Discount factor $\gamma = 0.95$
- Optimal Q-values $Q^*$ for given state $s_t$ (see above):

  $$Q^*(s_t, a_t = "move left") = R(s_t, a_t) + \gamma V^*(s_{t+1}) = -0.142625$$

  $$Q^*(s_t, a_t = "move right") = R(s_t, a_t) + \gamma V^*(s_{t+1}) = 0.9025$$
Model-Based Reinforcement Learning

- Learn a model \( \hat{M} \approx M \) from experience samples
- Experience samples \( e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle \) are obtained from interaction with the real environment
- Approximate \( P(s_{t+1} | s_t, a_t) \) (and \( R(s_t, a_t) \)) with **supervised learning**
  - Learn \( \langle s_t, a_t \rangle \rightarrow s_{t+1} \) via density estimation
  - Learn \( \langle s_t, a_t \rangle \rightarrow r_t \) via regression

- Learned model \( \hat{M} \) can be used for:
  - Planning
  - Model-free reinforcement learning
Case Study: Rockclimbing
How would you try to climb up a rock wall?

• Trial-and-Fall

• Watch how others do it

• Plan own moves based on wall

• Some combination

What is the most intelligent way to you?
How would you try to climb up a rock wall?

• Trial-and-Fall
  – (On-Policy) Reinforcement Learning

• Watch how others do it
  – (Off-Policy) Reinforcement Learning

• Plan own moves based on wall
  – Offline Planning (plan all moves before climbing)
  – Online Planning (plan next moves while climbing)

• Some combination
  – Use own experience or experience of others to plan ahead

What is the most intelligent way to you?
Outline

• **An Introduction to Autonomous Systems**
  – Motivation, Definition and Challenges
  – Artificial Intelligence

• **Decision Making in Autonomous Systems**
  – Markov Decision Processes
  – Planning
  – Reinforcement Learning

• **Applications and Further Challenges**
  – Deep Reinforcement Learning
  – Combining Planning and Reinforcement Learning
  – Further Challenges
Thank you!