IoT·인공지능·빅데이터 개론 및 실습

Imitation Learning

서울대학교 전기정보공학부
오성회

Contents

Contents

1 Markov Decision Process (MDP)

- **Markov decision process**: a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards
  - Set of states $\mathcal{S}$
  - Set of actions $Actions(s)$ in each state $s \in \mathcal{S}$
  - Transition model $P(s'|s, a)$
  - Reward function $R(s)$
- **Policy** π: $\mathcal{S} \rightarrow \{a \mid a \in Actions(s), s \in \mathcal{S}\}$ such that $\pi(s) \in Actions(s)$ for all s. $\pi(s)$ is the action recommended by the policy π for state s.
- **Optimal policy** π^*: a policy with the highest expected utility (expected sum of rewards).

1 Markov Decision Process (MDP)

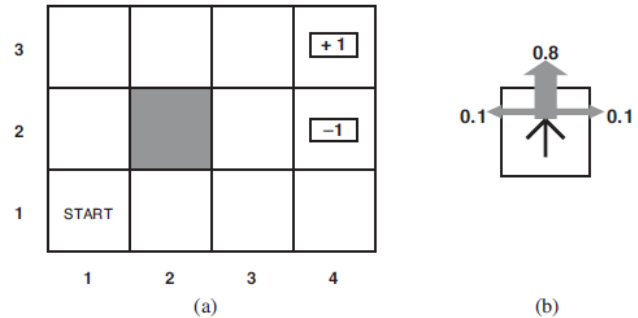


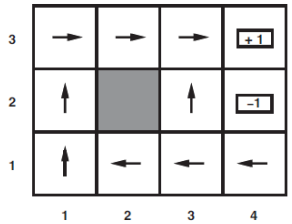
Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

- $Actions(s) = A(s) = [Up, Down, Left, Right]$
- For a deterministic case, a solution is $[Up, Up, Right, Right, Right]$.
- For the transition model given above, the probability of reaching the goal state at $(4, 3)$ using the action sequence $[Up, Up, Right, Right, Right]$ is only $0.8^5 = 0.32768$.

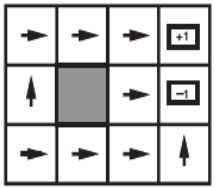
(Source: Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig, Prentice Hall, 2010.)

Example: Optimal Policy

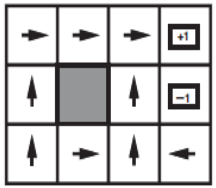
Optimal policy when R(s) = -0.04



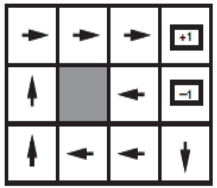
Optimal policies for different reward functions



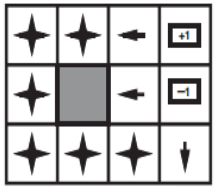
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

Utilities over Time

- Two types of problems
  - **Finite horizon problem**: the decision problem terminates at a fixed time T
  - **Infinite horizon problem**: there is no time limit
- With a finite horizon, the optimal action in a given state could change over time. Optimal policy for a finite horizon is nonstationary.
- Optimal policy for an infinite horizon is stationary.

Rewards under Stationarity

- **Additive rewards**:

$$V([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots$$

- **Discounted rewards**: ($\gamma \in [0, 1]$ is a discount factor)

$$V([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

- To avoid an infinite utility value for an infinite horizon problem:

  1. **Discounted utility**. Suppose that $|R(s)| \leq R_{max}$ for all s. Then, for $\gamma < 1$,

$$V([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1 - \gamma}.$$

  2. If the agent is guaranteed to reach a terminal state (such policy is called a **proper policy**).
  3. **Average reward**.

Optimal Policy and Value Function

- Value function (by executing π starting from s):

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

- Optimal policy starting from s:

$$\pi_s^* = \arg\max_{\pi} V^{\pi}(s).$$

- **Fact**: The optimal policy is *independent* of the starting state for a discounted infinite horizon problem. Optimal policy: π^*
- True value (or utility) of state s: $V^{\pi^*}(s) =: V(s)$. (cf. $R(s)$), (a.k.a. reward-to-go or cost-to-go)
- Using the principle of maximum expected utility

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$$

Value Iteration

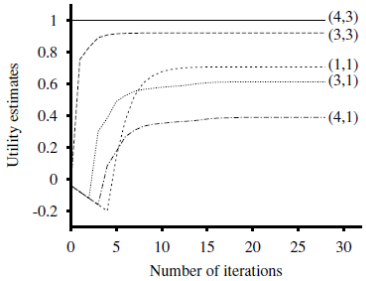
Bellman equation:

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$$

Value Iteration Algorithm:

- Repeat until convergence
  - For all state s

$$V(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$$



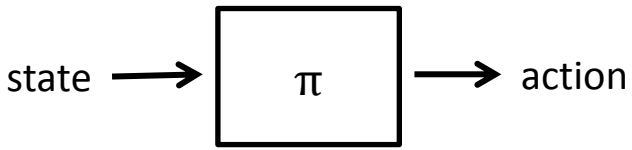
3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

2 Learning from Demonstration

- Difficulties in learning complex tasks
  - Designing rewards
  - Unknown dynamics ($P(s'|s,a)$)
  - (Large state and action spaces)
- Alternative solution:
  - Learning from expert's demonstrations
- Imitation learning (or LfD) methods:
  - Behavior cloning (BC)
  - Inverse reinforcement learning (IRL)

3 Behavior Cloning (BC)

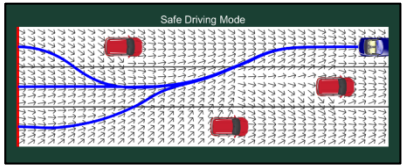
- Demonstrations: $\mathcal{D} = \{\tau_1, \ldots, \tau_N\}$
  - $\tau_i = \{(s_{i0}, a_{i0}), (s_{i1}, a_{i1}), \ldots, (s_{iT}, a_{iT})\}$
- **Behavior cloning**:
  - Learns a policy from $\mathcal{D}$ using supervised learning
  - **Classification**: problems with a finite number of actions
  - **Regression**: problems with an infinite number of actions
  - **Limitations**: (1) training samples are correlated; (2) can perform poorly for states not in the training set



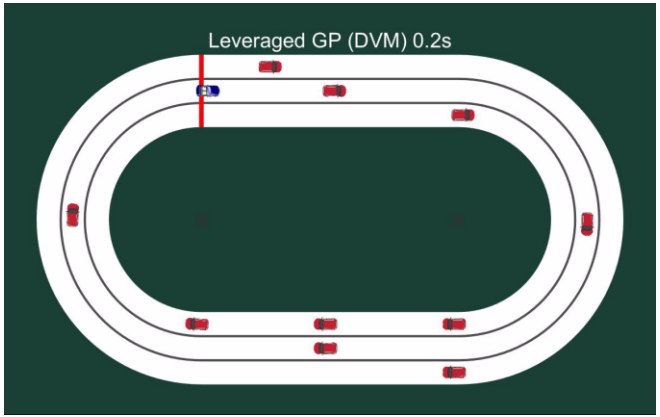
Autonomous Driving with GPR

- Action: heading angle (1).
- State: Distances to the closest cars in left, center, and right lanes in the front (3); Distances to the closest cars in left, center, and right lanes in the back (3); and Vehicle's distance from the center of the lane (1)

Training set

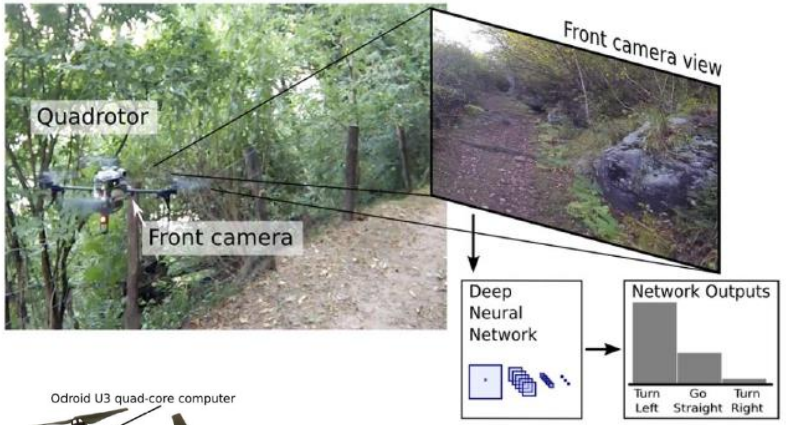


Test (Track Driving)



3 Behavior Cloning (BC)

Flying with Deep Learning



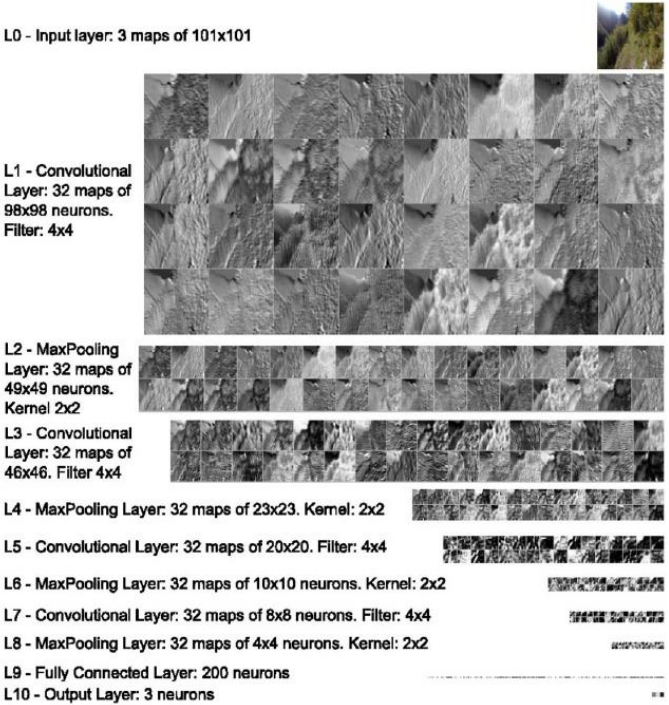
Classification problem:

- Input: Image
- Output: Turn Left, Go Straight, Turn Right

Alessandro Giusti, Jérôme Guzzi, Dan C. Ciresan, Fang-Lin He, Juan P. Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, and Luca M. Gambardella. "**A machine learning approach to visual perception of forest trails for mobile robots**." IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 661-667, Dec., 2015.

Flying with Deep Learning

CNN Architecture



3 Behavior Cloning (BC)

Flying with Deep Learning

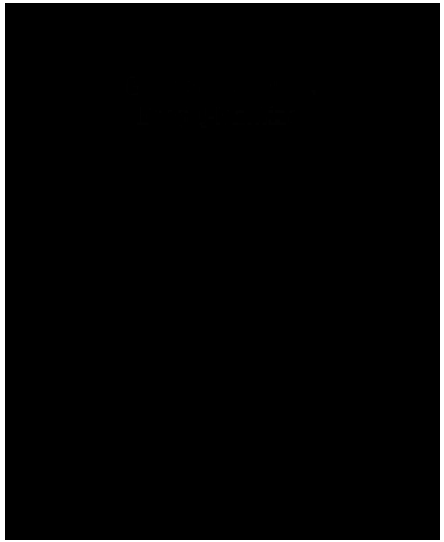
Contents

1 Reinforcement Learning

- **Markov decision processes (MDPs)**
  - Complete model is known
- **Reinforcement learning (RL)**
  - Use observed rewards to learn an optimal policy for the environment.
  - Complete model is not known.
- **Methods**
  - Q-learning: Q-function (or action-value function)
  - Actor-Critic
  - Policy gradient
  - and many more …

Deep Q-Network (DQN), 2015

- Playing Atari games
- Input: Game screen shots
- Output: Control (left, right, shoot, …)
- CNN, Q-learning

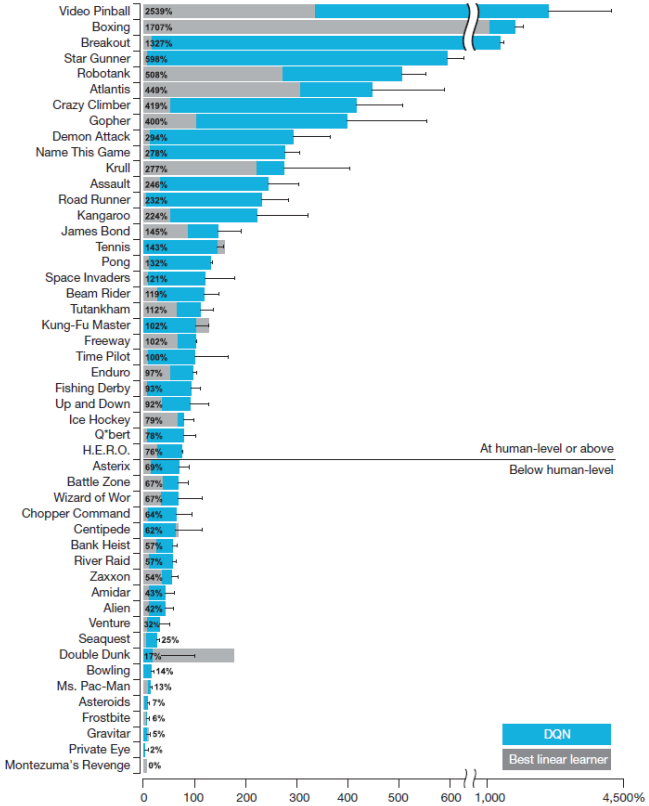


Breakout

Space Invaders

1 Reinforcement Learning

Deep Q-Network (DQN), 2015



AlphaGo, 2016

- Possible board positions of Go: 10^{170}
  - cf. Chess: 10^{47}
- Monte Carlo tree search
- Deep neural networks:
  - Value network
  - Policy network
- Reinforcement learning
- Trained from
  - 30 million human moves
  - Playing against itself
- 1,202 CPUs, 176 GPUs

Game	Date	Black	White	Result	Moves
1	9 March 2016	Lee Sedol	**AlphaGo**	Lee Sedol resigned	186 Game 1
2	10 March 2016	**AlphaGo**	Lee Sedol	Lee Sedol resigned	211 Game 2
3	12 March 2016	Lee Sedol	**AlphaGo**	Lee Sedol resigned	176 Game 3
4	13 March 2016	AlphaGo	**Lee Sedol**	AlphaGo resigned	180 Game 4
5	15 March 2016	Lee Sedol[note 1]	**AlphaGo**	Lee Sedol resigned	280 Game 5
Result: **AlphaGo 4 – 1 Lee Sedol**					

❷ Inverse Reinforcement Learning

- Markov decision process (MDP):
  $\boldsymbol{M} = \{\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{T}, \gamma, \boldsymbol{r}\}$
  - Goal: find the optimal policy $\boldsymbol{\pi}$ which maximizes the sum of $\boldsymbol{r}$
- Inverse Reinforcement Learning (IRL):
  - $\boldsymbol{M}/\boldsymbol{r}$ with $\boldsymbol{\mathcal{D}} = \{\tau_i\}$
  - Goal: find the underlying reward $\boldsymbol{r}$, which best explains $\boldsymbol{\mathcal{D}}$
  - Assumption: $\boldsymbol{\mathcal{D}}$ comes from an expert (optimal policy)