Reinforcement Learning

Bùi Tiến Lên

2022

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

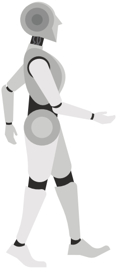
Notation

symbol	meaning
$a, b, c, N \ldots$	scalar number
$\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{x}, \boldsymbol{y} \ldots$	column vector
$\boldsymbol{X}, \boldsymbol{Y} \ldots$	matrix
$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	set of integer numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{R}^D$	set of vectors
$\mathcal{D}, \mathcal{X}, \mathcal{Y}, \ldots$	set
$\mathcal{A}$	algorithm

symbol	meaning
$A, X, Y, S \ldots$	random variable
$\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S} \ldots$	multivariate random variable
$a, x, y, s \ldots$	realization
$\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s} \ldots$	realization
p, pr, P, Pr	probability

Reinforcement Learning

Agent and Environment



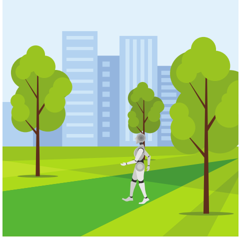
Agent

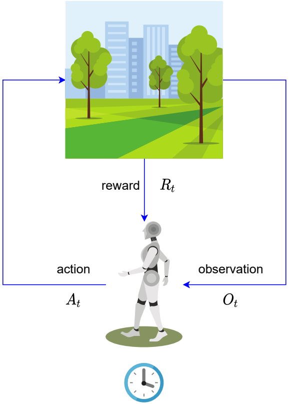
Reward

Environment

- **Agent**: takes actions.
- **Environment**: the world in which the agent exists and operates.
- **Reward**: feedback that measures the success or failure of the agent's action.

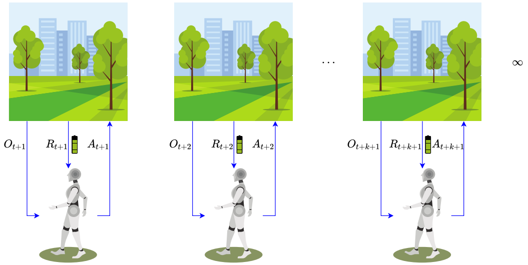
Agent and Environment (cont.)





- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Agent and Environment (cont.)



- Total reward obtained from time t

$$G_t = \sum_{k=0}^{\infty} R_{t+k+1} = R_{t+1} + R_{t+2} + \cdots \tag{1}$$

- Discounted total reward with discount factor γ

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \cdots \tag{2}$$

Reinforcement Learning

Concept 1

Reinforcement learning (RL) is concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/effectors

Characteristics of Reinforcement Learning

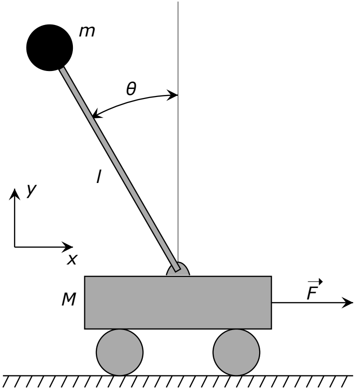
What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

The Reinforcement Learning Problem

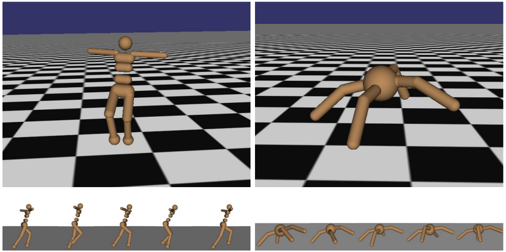
- Problems
- State

Cart-Pole Problem



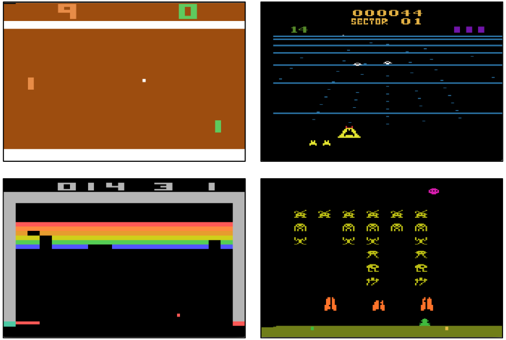
- **Objective**: Balance a pole on top of a movable cart
- **State**: angle, angular speed, position, horizontal velocity
- **Action**: horizontal force applied on the cart
- **Reward**: 1 at each time step if the pole is upright

Robot Locomotion



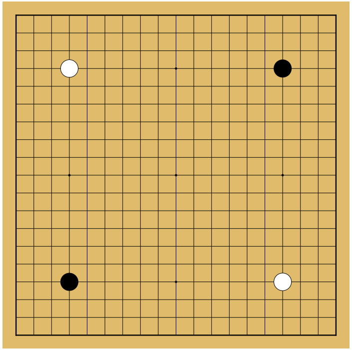
- **Objective**: Make the robot move forward
- **State**: Angle, position, velocity of all joints
- **Action**: Torques applied on joints
- **Reward**: 1 at each time step upright + forward movement

Atari Games



- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game screen
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

Go



- **Objective**: Win the game!
- **State**: Position of all pieces
- **Action**: Where to put the next piece down
- **Reward**: On last turn: 1 if you won, 0 if you lost

History and State

Concept 2

The **history** is the sequence of observations, actions, rewards, i.e. all observable variables up to time t

$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t \tag{3}$$

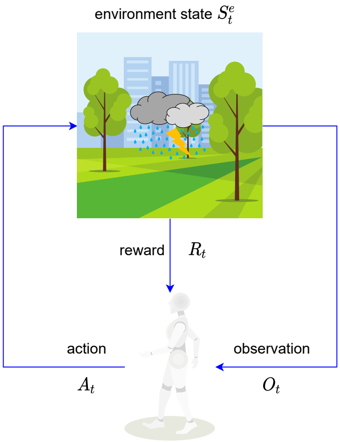
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards

Concept 3

State is the information used to determine what happens next. Formally, state is a function of the history

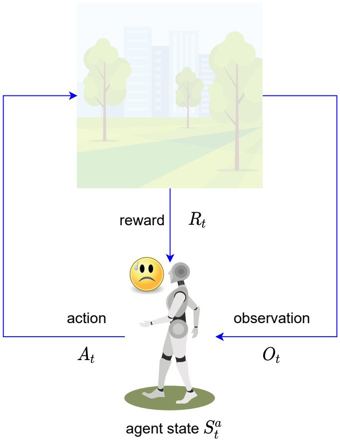
$$S_t = f(H_t) \tag{4}$$

Environment State



- The **environment state** S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward
- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

Agent State



- The agent state S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t) \tag{5}$$

Information State

An **information state** (a.k.a. **Markov state**) contains all useful information from the history.

Concept 4

A state S_t is **Markov** if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t] \tag{6}$$

- "The future is independent of the past given the present"

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty} \tag{7}$$

- Once the state is known, the history may be thrown away; i.e., the state is a sufficient statistic of the future

Markov Assumption

Concept 5

- The environment state S_t^e is Markov
- The history H_t is Markov

Fully Observable Environments

Concept 6

Full observability: agent **directly** observes environment state

$$O_t = S_t^a = S_t^e \tag{8}$$

- Formally, this is a **Markov decision process** (MDP)

Partially Observable Environments

Concept 7

Partial observability: agent **indirectly** observes environment; now agent state is different from environment state. Agent must construct its own state representation S_t^a , e.g.

- Complete history: $S_t^a = H_t$
- Beliefs of environment state: $S_t^a = (P[S_t^e = s^1], ..., P[S_t^e = s^n])$
- Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

- A robot with camera vision isn't told its absolute location
- A trading agent only observes current prices
- A poker playing agent only observes public cards
- Formally this is a **partially observable Markov decision process** (POMDP)

Inside An RL Agent

Major Components of an RL Agent

An RL agent may include one or more of these components:

- **Agent state**
- **Policy**: agent's behaviour function
- **Value function**: how good is each state and/or action
- **Model**: agent's representation of the environment

Policy

Concept 8

A **policy** is the agent's behaviour. It is a map from state to action

- Deterministic policy:

$$a = \pi(s) \tag{9}$$

- Stochastic policy:

$$\pi(a \mid s) = P[A_t = a \mid S_t = s] \tag{10}$$

Value Function

Concept 9

Value function is a prediction of future reward

- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s] \tag{11}$$

Model

Concept 10

A model predicts what the environment will do next

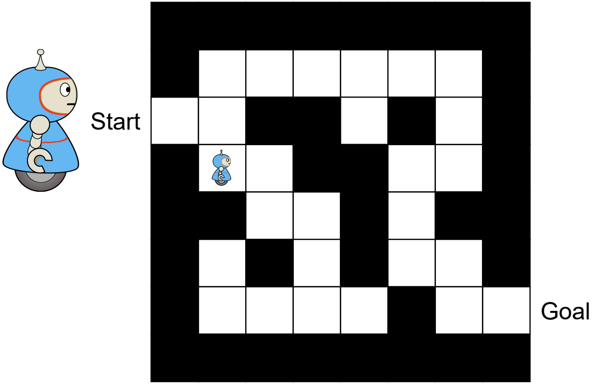
- $\mathcal{P}$ predicts the next state

$$\mathcal{P}^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{12}$$

- $\mathcal{R}$ predicts the next (immediate) reward

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \tag{13}$$

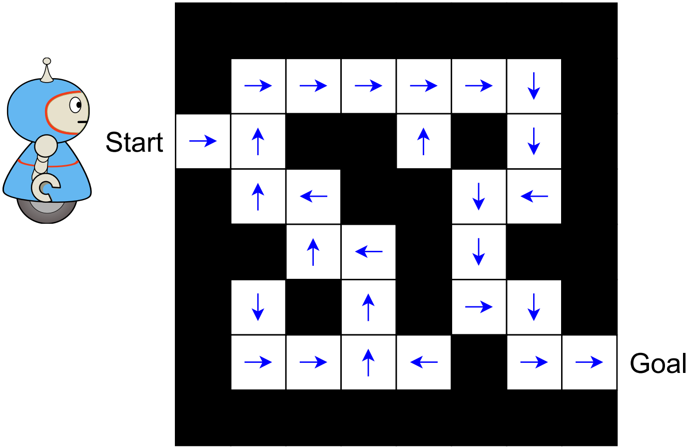
Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

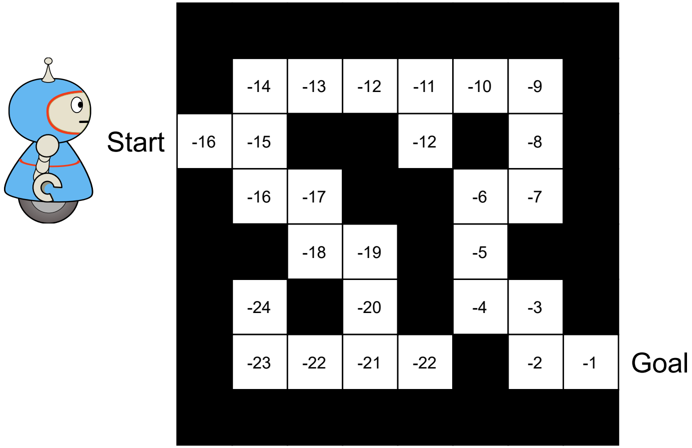
Maze Example: Policy

- Arrows represent policy $\pi(s)$ for each state s



Maze Example: Value Function

- Numbers represent value $v^{\pi}(s)$ of each state s



Maze Example: Model

- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect
- Grid layout represents transition model $P^a_{ss'}$
- Numbers represent immediate reward R^a_s from each state s (same for all a)

Categorizing RL agents

• Value Based
  • No Policy (Implicit)
  • Value Function
• Policy Based
  • Policy
  • No Value Function
• Actor Critic
  • Policy
  • Value Function
	• Model Free
	  • Policy and/or Value Function
	  • No Model
	• Model Based
	  • Policy and/or Value Function
	  • Model

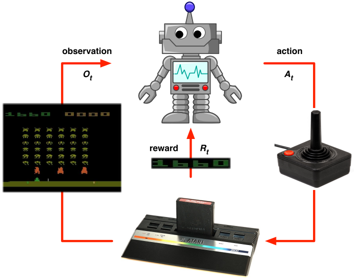
Problems within Reinforcement Learning

Learning and Planning

Two fundamental problems in sequential decision making

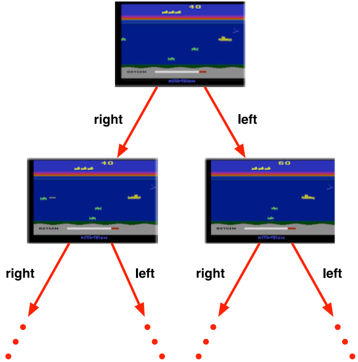
- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy

Atari Example: Reinforcement Learning



- Rules of the game are **unknown**
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Atari Example: Planning



- Rules of the game are **known**
- Can query emulator
  - perfect model inside agent's brain
- If I take action a from state s:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search

Exploration and Exploitation

- Reinforcement learning is like **trial-and-error learning**

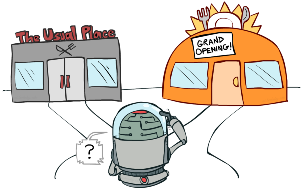
  The agent should discover a good policy
  from its experiences of the environment
  without losing too much reward along the way

- **Exploration** finds more information about the environment
- **Exploitation** exploits known information to maximise reward
- It is usually **important** to explore **as well as** exploit

Exploration and exploitation (cont.)

	Exploitation	**Exploration**
	go with the best strategy found so far	take a new action with unknown consequences
Pros	Maximize reward as reflected in the current utility estimates Avoid bad stuff	Get a more accurate model of the environment Discover higher-reward states than the ones found so far
Cons	Might also prevent you from discovering the true optimal strategy	When you're exploring, you're not maximizing your utility Something bad might happen

Examples



	Exploitation	Exploration
Restaurant Selection	Go to your favourite restaurant	Try a new restaurant
Online Banner Advertisements	Show the most successful advert	Show a different advert
Oil Drilling	Drill at the best known location	Drill at a new location
Game Playing	Play the move you believe is best	Play an experimental move

Prediction and Control

- Prediction: evaluate the future
  - Given a policy
- Control: optimise the future
  - Find the best policy

Agent’s Learning Task

- Bellman Equation
- Dynamic Programming
- Q-Learning
- Deep Q-Learning

Value Functions

Concept 11 (How good is a state?)

The state-value function at state s, is the expected cumulative reward from following the policy π from state s

$$v^\pi(s) = \mathbb{E}[G_t \mid S_t = s] \tag{14}$$

Concept 12 (How good is a state-action pair?)

The action-value Q function at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy π

$$q^\pi(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a] \tag{15}$$

Optimal Value Functions

Concept 13

- The optimal state-value function $v^*(s)$ is the maximum value function over all policies

$$v^*(s) = \max_{\pi} v^{\pi}(s) \tag{16}$$

- The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

$$q^*(s, a) = \max_{\pi} q^{\pi}(s, a) \tag{17}$$

Estimating Value Functions

- Estimating v^π or q^π is called **policy evaluation** or, simply, **prediction**
- Estimating v^* or q^* is sometimes called **control**, because these can be used for **policy optimization**

Optimal Policy

Concept 14

A partial ordering over policies

$$\pi \geq \pi' \text{ if } v^{\pi}(s) \geq v^{\pi'}(s), \forall s \tag{18}$$

Theorem 1

For any Markov Decision Process

- *There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v^{\pi^*}(s) = v^*(s)$*
- *All optimal policies achieve the optimal action-value function, $q^{\pi^*}(s, a) = q^*(s, a)$*

Agent's Learning Task

- **Agent's Goal**: Find the **optimal policy** π^* that maximize the expected sum of rewards .

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[G_t\right] \tag{19}$$

Optimal Q-function

- q^* encodes the optimal policy, an optimal policy can be found by maximising over $q^*(s, a)$

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q^*(s, a) \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

$$\pi^*(s) = \arg\max_a q^*(s, a) \tag{21}$$

- If we know $q^*(s, a)$, we immediately have the optimal policy

Bellman equations

- Two Bellman equations for policy π

$$
\begin{aligned}
v^{\pi}(s) &= \mathbb{E}[R_{t+1} + \gamma v^{\pi}(S_{t+1}) \mid S_t = s] \\
q^{\pi}(s, a) &= \mathbb{E}[R_{t+1} + \gamma q^{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]
\end{aligned} \tag{22}
$$

- Two Bellman optimality equations

$$
\begin{aligned}
v^{*}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v^{*}(S_{t+1}) \mid S_t = s, A_t = a] \\
q^{*}(s, a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q^{*}(S_{t+1}, a') \mid S_t = s, A_t = a]
\end{aligned} \tag{23}
$$

Bellman equations (cont.)

- There are equivalences between state and action values

$$v^{\pi}(s) = \sum_{a} \pi(a \mid s) q^{\pi}(s, a) \tag{24}$$

$$v^{*}(s) = \max_{a} q^{*}(s, a) \tag{25}$$

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear and there is no closed form solution (in general)
- Many iterative solution methods
- Using models/dynamic programming
  - Value iteration
  - Policy iteration
- Using samples
  - Monte Carlo
  - Q-learning
  - Sarsa

What is Dynamic Programming?

Concept 15

Dynamic Programming is a method for solving complex problems by breaking them down into subproblems

- Solve the subproblems
- Combine solutions to subproblem

- All such methods consist of two important parts: **policy evaluation** and **policy improvement**

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - Principle of optimality applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused

Q-Learning for Deterministic Worlds

For each (s, a) **initialize** table entry $Q(s, a) \leftarrow 0$
Observe current state s
Do forever:

- **Select** an action a and execute it
- **Receive** immediate reward r and **observe** the new state s'
- **Update** the table entry for $Q(s, a)$ as follows

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \tag{26}$$

$$s \leftarrow s'$$

Q-Learning Theorem

Notice if rewards non-negative, then

$$(\forall s, a, n)\ 0 \leq Q_n(s, a) \leq Q_{n+1}(s, a) \leq Q^*(s, a)$$

Theorem 2

Q converges to Q^*.

Proof

Define a full interval to be an interval during which each (s, a) is visited. During each full interval the largest error in Q table is reduced by factor of γ

Let Q_n be table after n updates, and Δ_n be the maximum error in Q_n; that is

$$\Delta_n = \max_{s,a} |Q_n(s, a) - Q^*(s, a)|$$

Q-Learning Theorem (cont.)

For any table entry $Q_n(s, a)$ updated on iteration $n+1$, the error in the revised estimate $Q_{n+1}(s, a)$ is

$$
\begin{aligned}
|Q_{n+1}(s,a) - Q^*(s,a)| &= |(r + \gamma \max_{a'} Q_n(s',a')) - (r + \gamma \max_{a'} Q^*(s',a'))| \\
&= \gamma |\max_{a'} Q_n(s',a') - \max_{a'} Q^*(s',a')| \\
&\leq \gamma \max_{a'} |Q_n(s',a') - Q^*(s',a')| \\
&\leq \gamma \max_{s'',a'} |Q_n(s'',a') - Q^*(s'',a')| \\
|Q_{n+1}(s,a) - Q^*(s,a)| &\leq \gamma \Delta_n
\end{aligned}
$$

■

Q-Learning for Nondeterministic Worlds

For each (s, a) **initialize** table entry $Q(s, a) \leftarrow 0$
Iterate over $t = 1, 2, \ldots$
  Update the table entry for $Q(s, a)$ as follows

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \underbrace{\alpha}_{\text{learning rate}} \overbrace{[\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a)}_{\text{new value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}]}^{\text{temporal difference}} \quad (27)$$

- Q converges to Q^* [Watkins and Dayan, 1992]

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

Lê, B. and Tô, V. (2014).
Cở sở trí tuệ nhân tạo.
Nhà xuất bản Khoa học và Kỹ thuật.

Russell, S. and Norvig, P. (2021).
Artificial intelligence: a modern approach.
Pearson Education Limited.