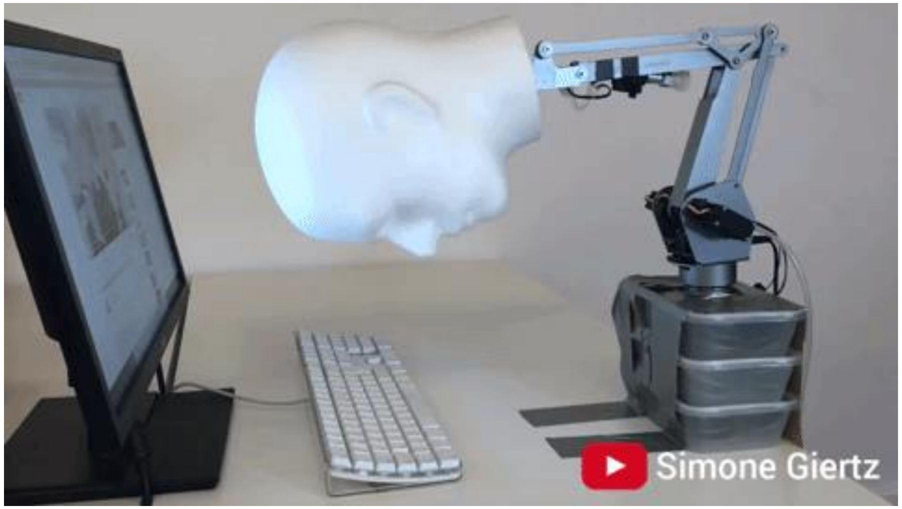
REINFORCEMENT LEARNING

ECS170 Spring 2018
Josh McCoy, @deftjams

Simple Agents



function TABLE-DRIVEN-AGENT(*percept*) **returns** an action
  persistent: *percepts*, a sequence, initially empty
        table, a table of actions, indexed by percept sequences, initially fully specified

  append *percept* to the end of *percepts*
  action ← LOOKUP(*percepts*, *table*)
  return *action*

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Reflex Agent

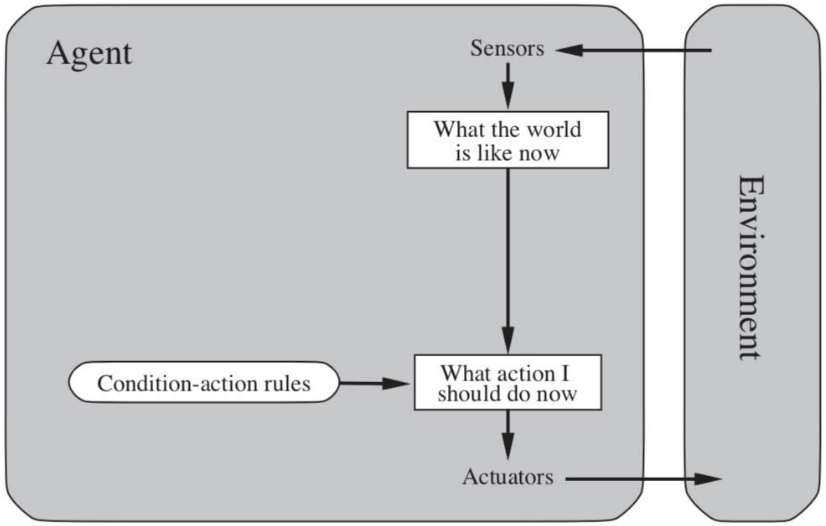


Figure 2.9 Schematic diagram of a simple reflex agent.

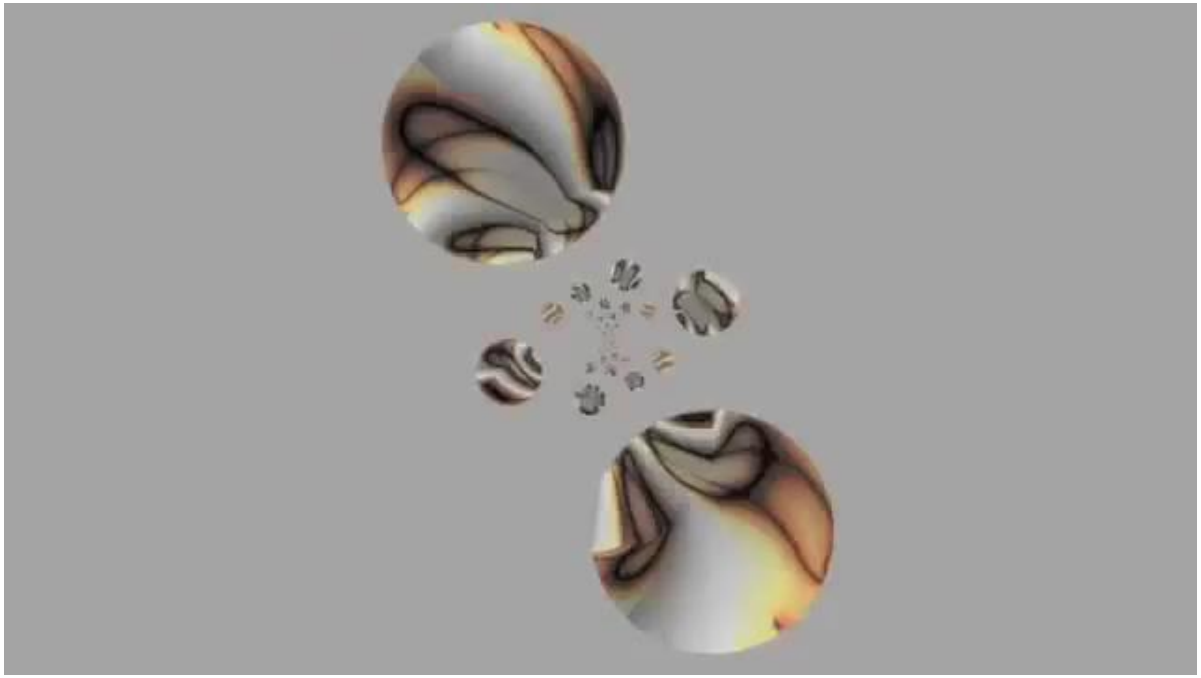
function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

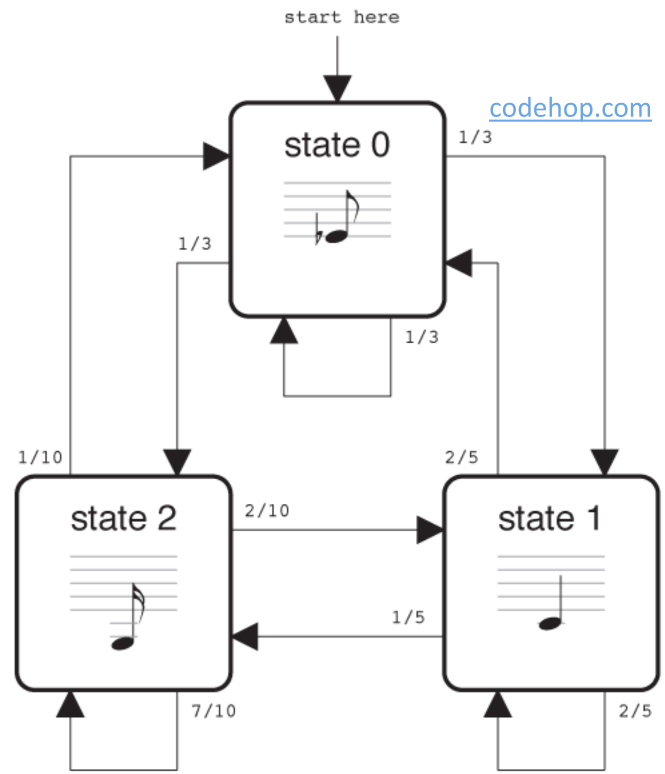
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

Markov Models

[Emmy Vivaldi, 2012](#)





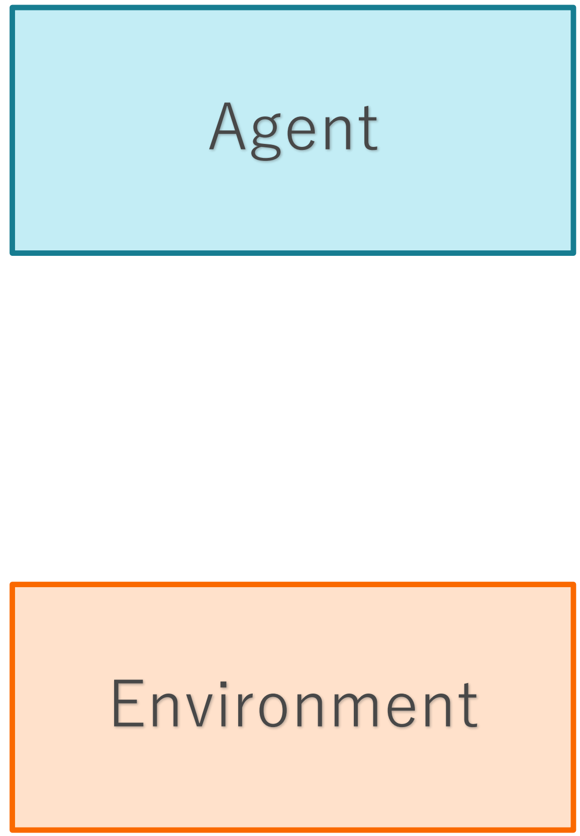
$$q(x \to x') \qquad \pi_t(x)$$

MDPs

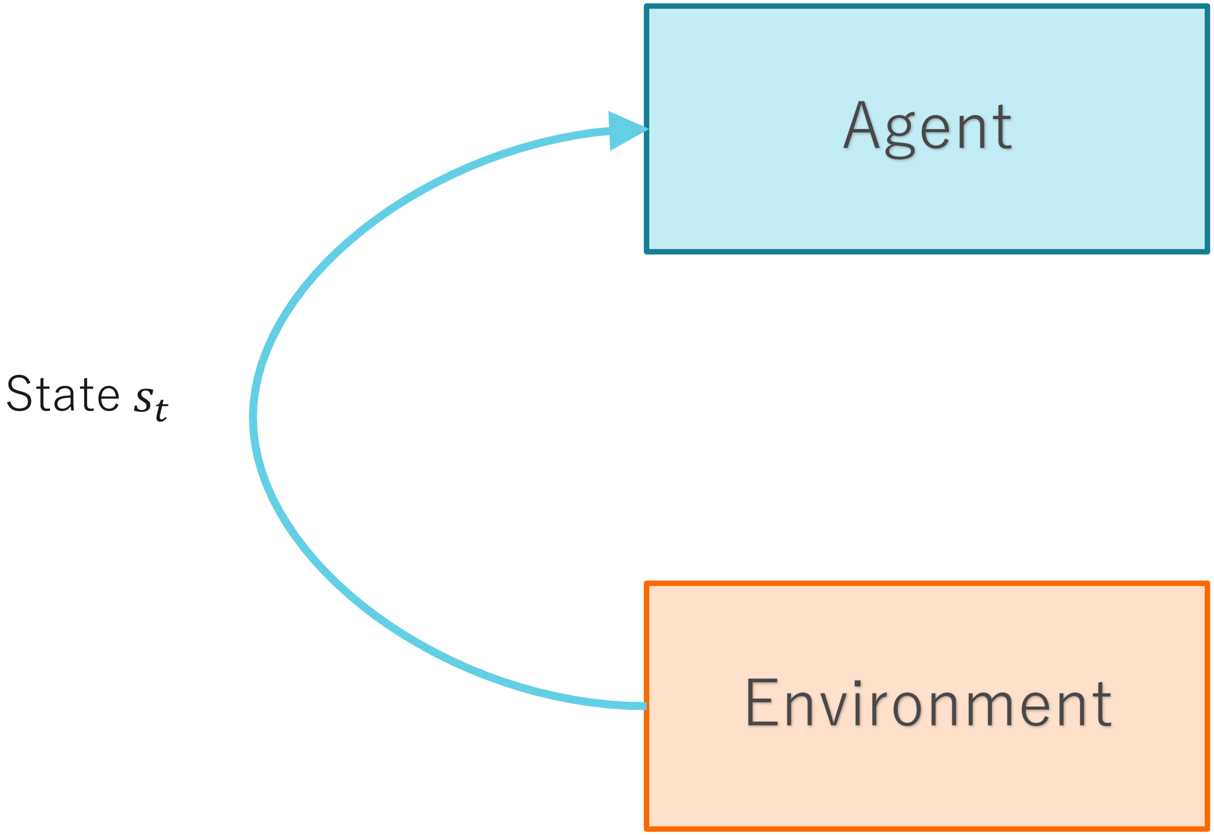
- Markov Decision Process
- Optimal policy that maximizes expected total reward.
- RL approximates an unknown MDP.

*MDPs covered in detail by lecture around this Jupyter notebook:
https://github.com/aimacode/aima-python/blob/master/mdp_apps.ipynb

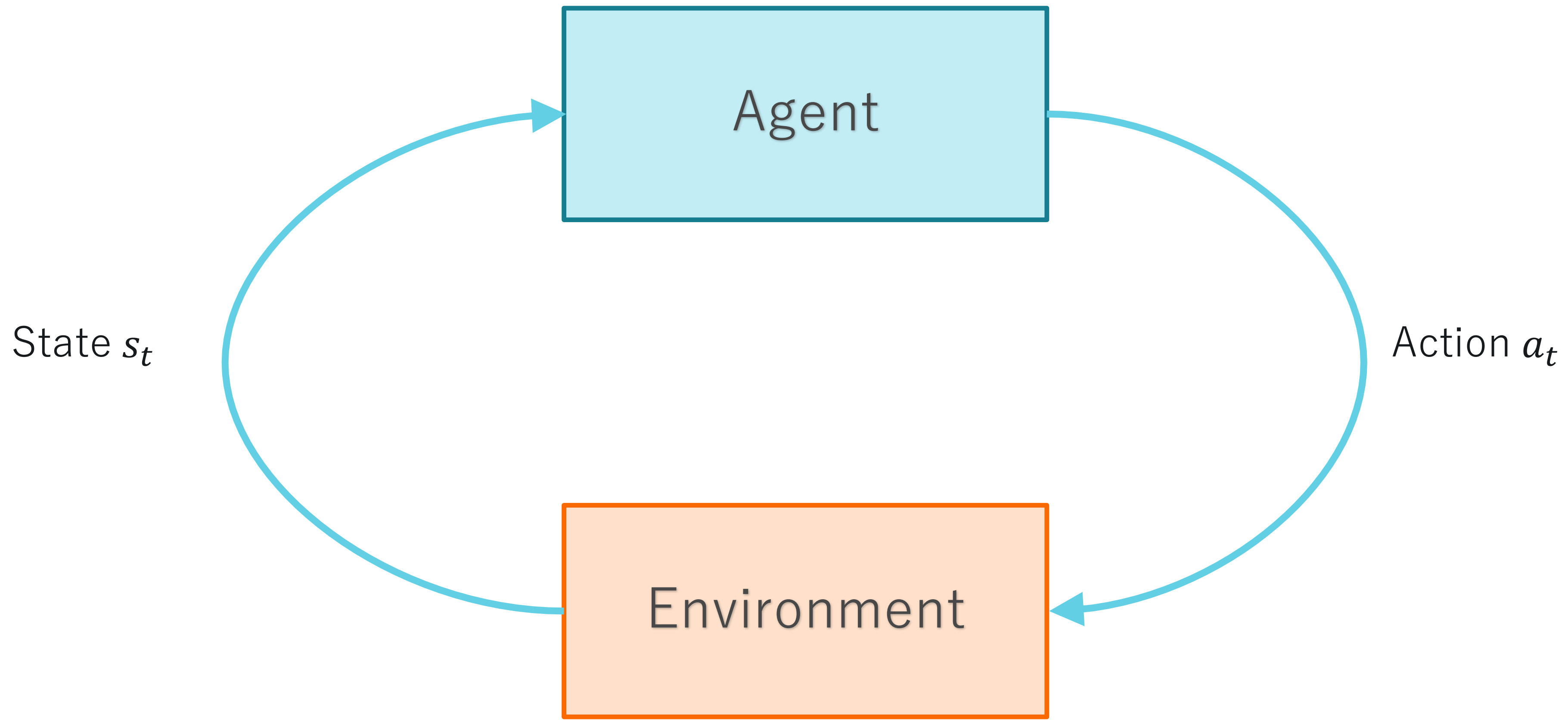
Basic Reinforcement Learning Loop



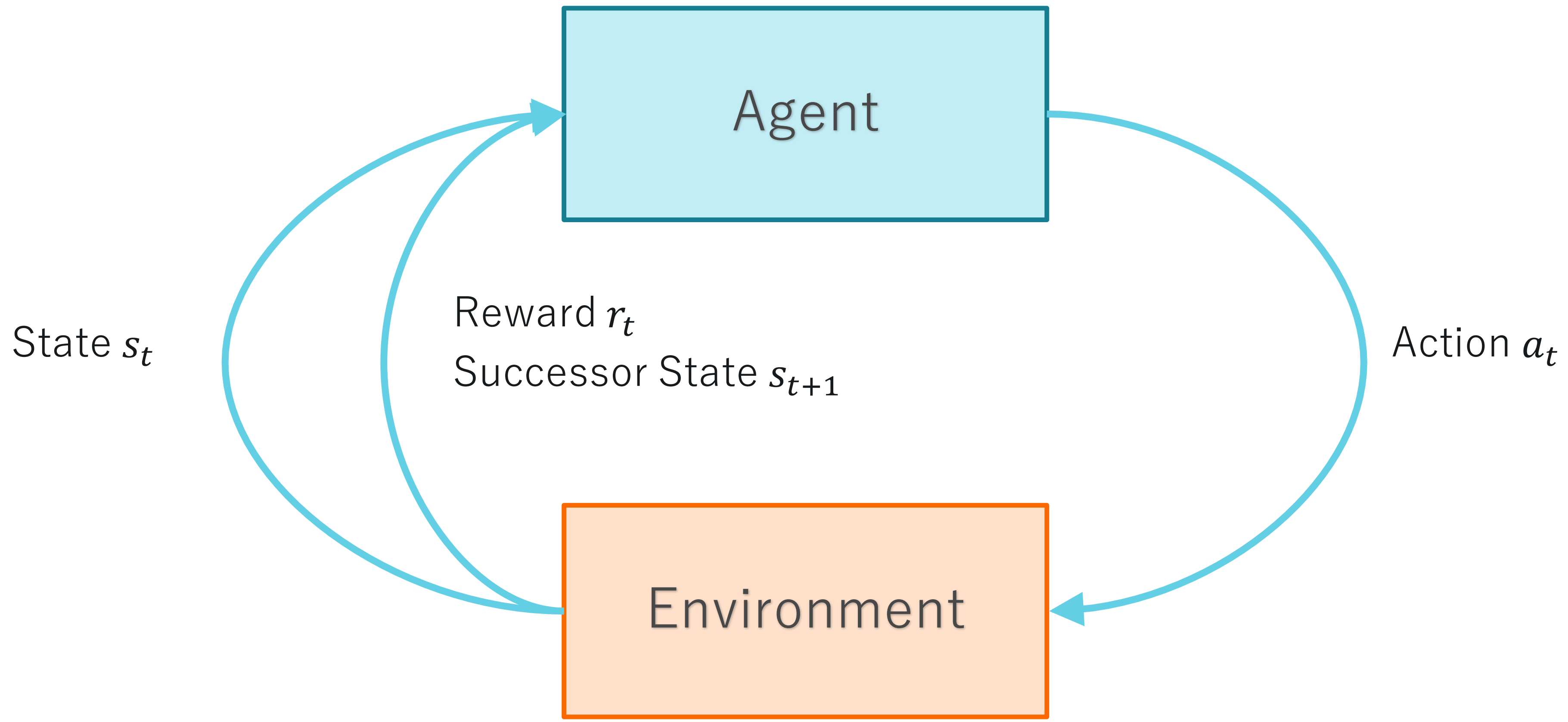
Basic Reinforcement Learning Loop



Basic Reinforcement Learning Loop



Basic Reinforcement Learning Loop

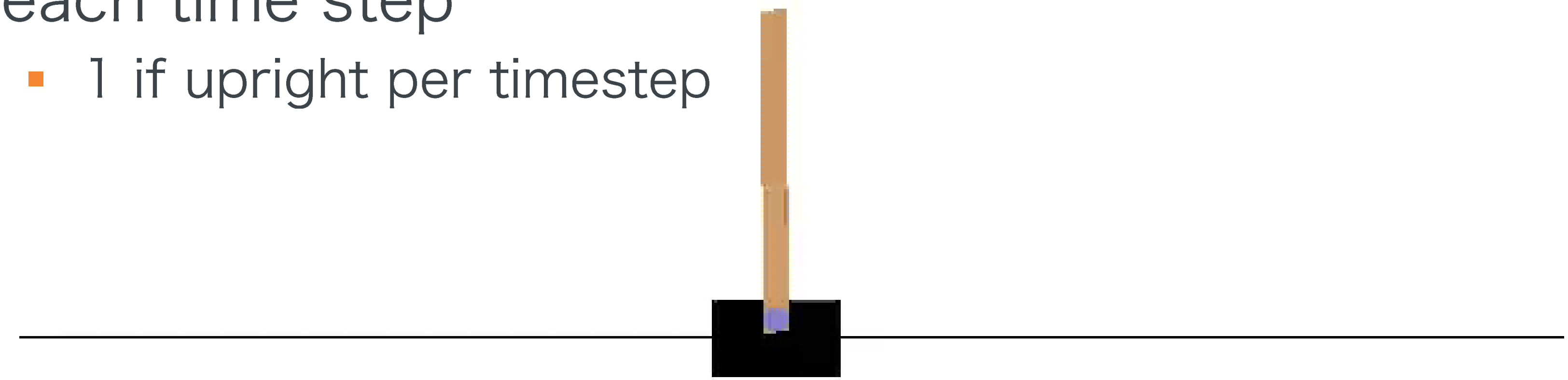


Goal: use observed rewards to learn an optimal (or nearly optimal) policy for the environment

p. 830

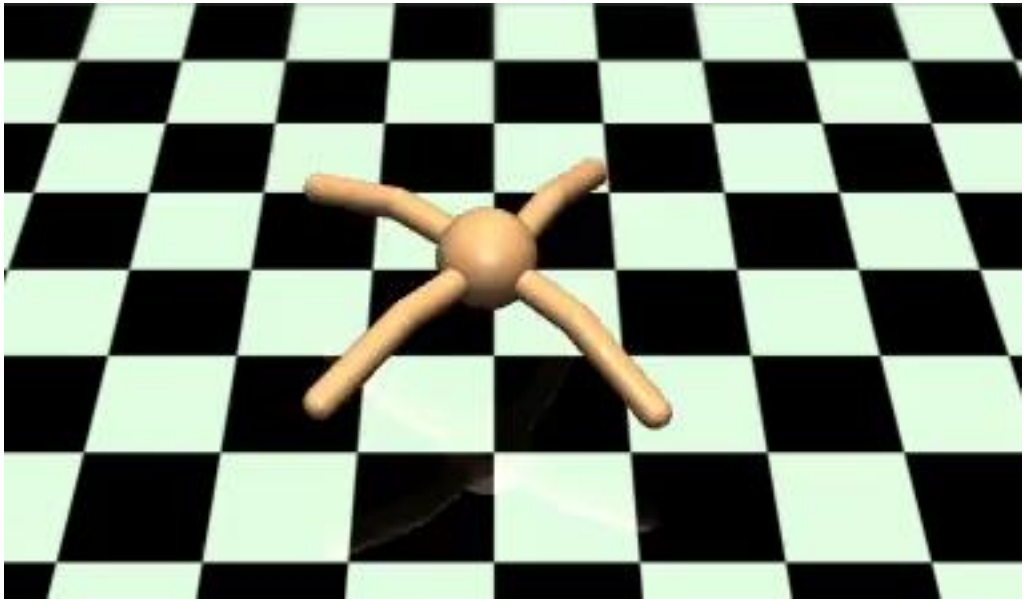
Problems: Cart-Pole

- Balance pole on a cart.
- State: position of cart, angle of pole, velocity
- Actions: cart movement
  - Front, back, none
  - Discrete or continuous
- Reward: how close the poll is to being upright at each time step
  - 1 if upright per timestep



Problems: Robot Locomotion

- Forward movement.
- State: robot position, joint angles
- Actions: change joint angles
- Reward: more for each time step with
  - Forward movement
  - Upright



Problems: Go

- Win.
- State: game board
- Action: place stone
- Reward: 1 for win, 0 for loss
  - Other approaches?

Problems: games

- Wins, points or speed run.
- State: game states
- Action: player actions
- Reward: score, better proxies for winning (#of units, etc.)
  - Other approaches?

Reinforcement Learning Areas

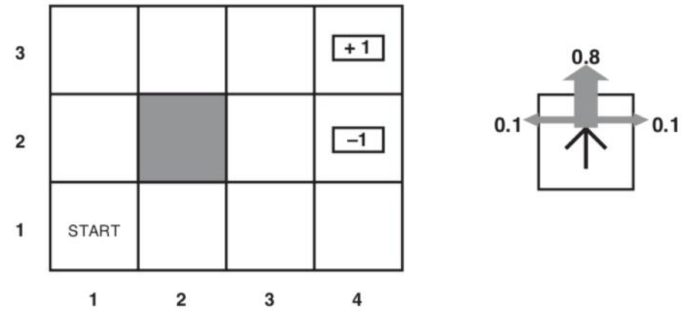
- **Passive learning:** fixed policy; learns utility of states.
- **Active learning:** also learns policy.
- **Reflex Agents:** maps states directly to actions.
- **Utility-based agent:** Utility function over states for action selection.
- **Q-learning**: learns action-utility function (Q-function) that gives the expected utility for an action in a state.

Formalization of RL

- Utility function of a state: $U(s)$
  - Numeric value often with 1 as best.
- Probability that a state and an action will result in a specific subsequent state
  - $P(s'|s,a)$
- Discount factor γ between 0 and 1.
  - Low means driven by current rewards.
  - High means driven by future rewards.
- optimal policy that maximizes utility: π^*

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

Toy Problem



(a) (b)

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.

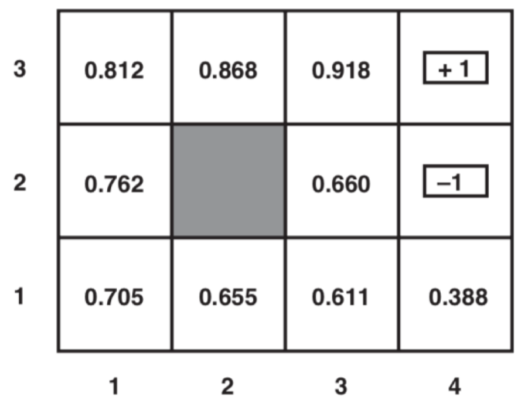


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

Temporal Difference Learning

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

function PASSIVE-TD-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Figure 21.4 A passive reinforcement learning agent that learns utility estimates using temporal differences. The step-size function $\alpha(n)$ is chosen to ensure convergence, as described in the text.

Q-learning

Q-value:

$$U(s) = max\ Q(s, a)$$

- Value of doing a in s.

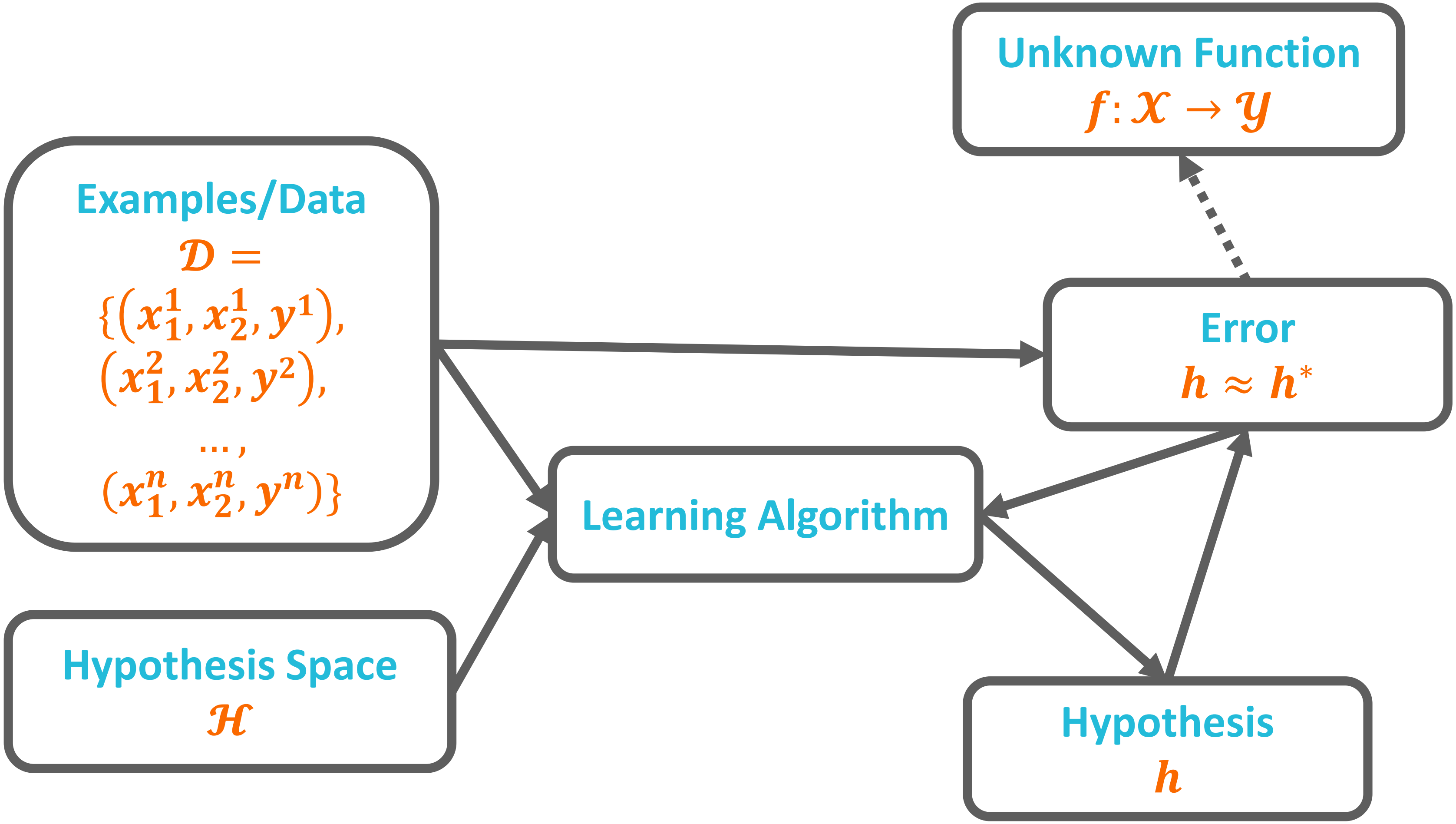
$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a') \ .$$

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s,a] \leftarrow Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s',a'], N_{sa}[s',a']), r'$
  return a

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(s,a)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

Process


Unknown Function
$f: \mathcal{X} \rightarrow \mathcal{Y}$
Examples/Data
$\mathcal{D} = \{(x_1^1, x_2^1, y^1), (x_1^2, x_2^2, y^2), \ldots, (x_1^n, x_2^n, y^n)\}$
Error
$h \approx h^*$
Learning Algorithm
Hypothesis Space
$\mathcal{H}$
Hypothesis
h

- OpenAI Gym
  - https://gym.openai.com/
- Cart-pole Q-learning implementation
  - https://dev.to/n1try/cartpole-with-q-learning---first-experiences-with-openai-gym
- Reinforcement Learning Course
  - Dave Silver
  - https://www.youtube.com/playlist?list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT