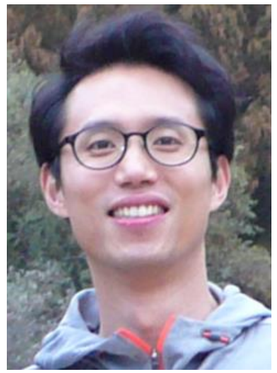
PMI Lab
Pattern Recognition and Machine Intelligence

Hankuk University of Foreign Studies

Introduction To Deep Reinforcement Learning-Part I



Prof. Jae Young Choi

Pattern Recognition and Machine Intelligence Lab. (PMI)

Hankuk University of Foreign Studies

Reinforcement Learning

- Reinforcement Learning: Solving sequential decision-making problems

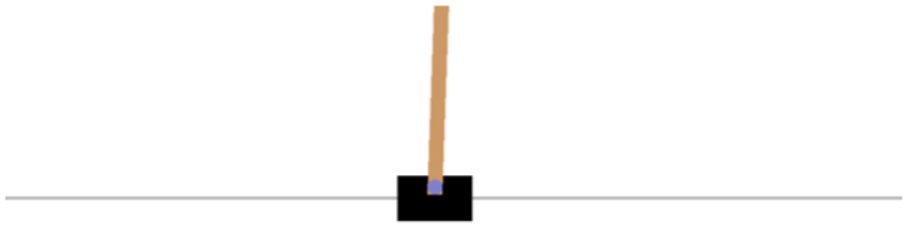


Figure 1.1 `CartPole-v0` is a simple toy environment. The objective is to balance a pole for 200 time steps by controlling the left-right motion of a cart.

When completing *State*→ *Action*→*Reward*, one-time step has passed

Reinforcement Learning Control Loop

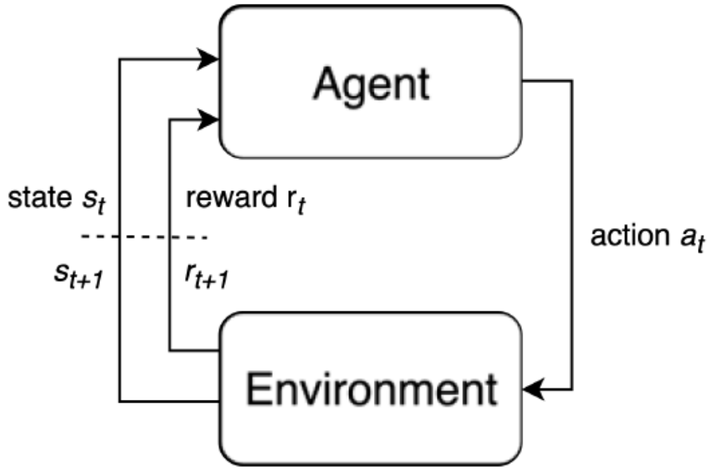
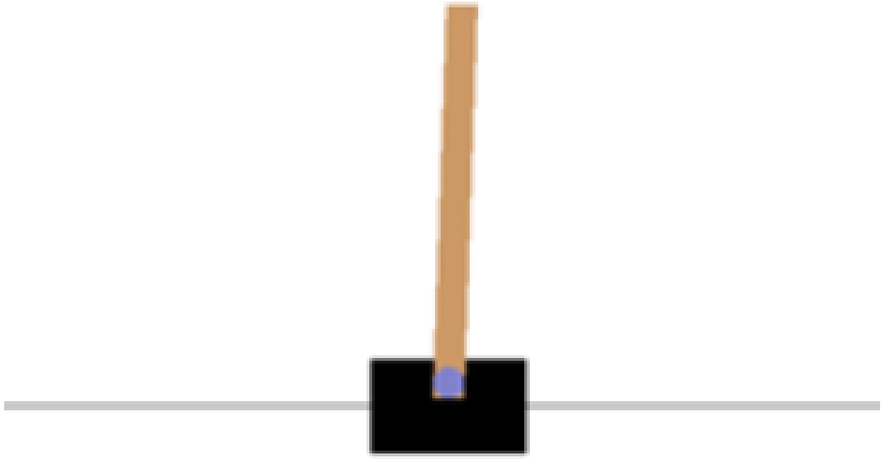


Figure 1.2 The reinforcement learning control loop

- Feedback control loop
- (s_t, a_t, r_t) is called "experience"
- Control loop can repeat forever or terminate by reaching terminal state or a maximum time step $t = T$
- Time horizon from t=0 to termination : "Episode"
- $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), \ldots$ sequence of experiences over an episode : "Trajectory"

Examples of RL Environments

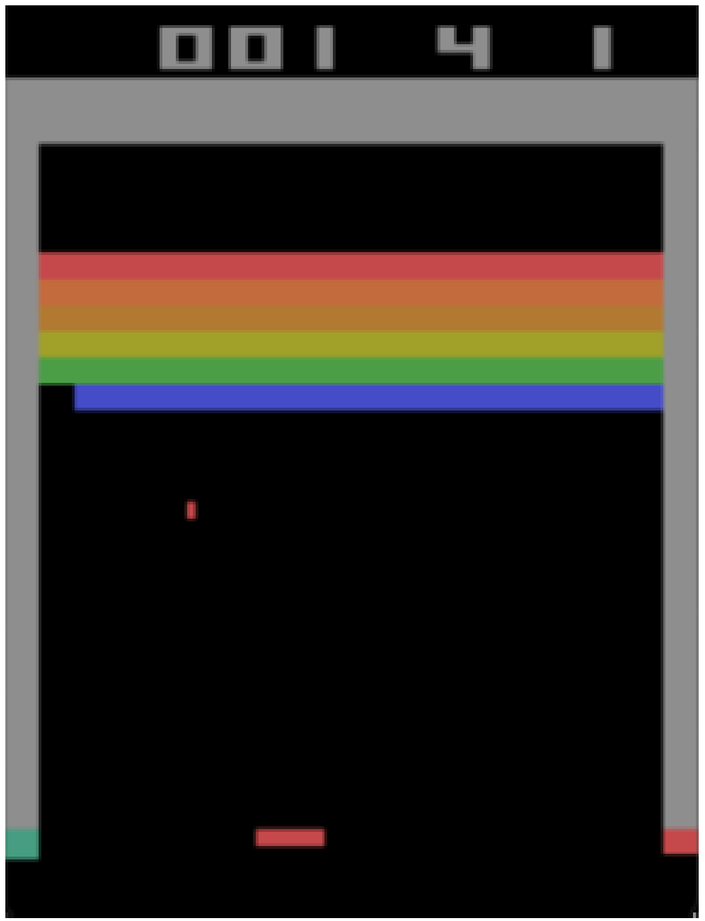
- CartPole



1. **Objective:** Keep the pole upright for 200 time steps.
2. **State:** An array of length 4 which represents: [cart position, cart velocity, pole angle, pole angular velocity]. For example, [−0.034, 0.032, −0.031, 0.036].
3. **Action:** An integer, either 0 to move the cart a fixed distance to the left, or 1 to move the cart a fixed distance to the right.
4. **Reward:** +1 for every time step the pole remains upright.
5. **Termination:** When the pole falls over (greater than 12 degrees from vertical), or when the cart moves out of the screen, or when the maximum time step of 200 is reached.

Examples of RL Environments

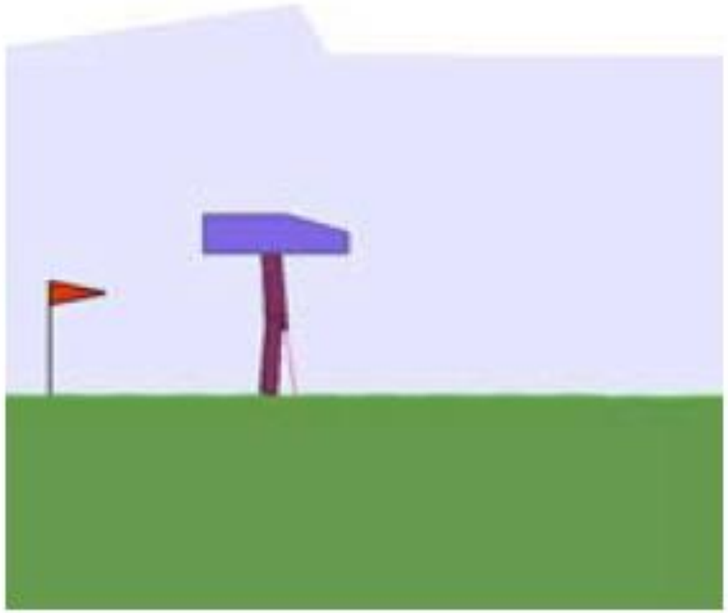
- Atari Breakout



1. **Objective:** Maximize the game score.
2. **State:** An RGB digital image with resolution 160 × 210 pixels—that is, what we see on the game screen.
3. **Action:** An integer from the set {0, 1, 2, 3} which maps to the game controller actions {no-action, launch the ball, move right, move left}.
4. **Reward:** The game score difference between consecutive states.
5. **Termination:** When all game lives are lost.

Examples of RL Environments

- BipedalWalker



1. **Objective:** Walk to the right without falling.
2. **State:** An array of length 24 which represents: [hull angle, hull angular velocity, x-velocity, y-velocity, hip 1 joint angle, hip 1 joint speed, knee 1 joint angle, knee 1 joint speed, leg 1 ground contact, hip 2 joint angle, hip 2 joint speed, knee 2 joint angle, knee 2 joint speed, leg 2 ground contact, . . ., 10 lidar readings]. For example, [2.745e−03, 1.180e−05, −1.539e−03, −1.600e−02, . . ., 7.091e−01, 8.859e−01, 1.000e+00, 1.000e+00].
3. **Action:** A vector of four floating point numbers in the interval [−1.0, 1.0] which represents: [hip 1 torque and velocity, knee 1 torque and velocity, hip 2 torque and velocity, knee 2 torque and velocity]. For example, [0.097, 0.430, 0.205, 0.089].
4. **Reward:** Reward for moving forward to the right, up to a maximum of +300. −100 if the robot falls. Additionally, there is a small negative reward (movement cost) at every time step, proportional to the absolute torque applied.
5. **Termination:** When the robot body touches the ground or reaches the goal on the right side, or after the maximum time step of 1600.

States, Actions, Rewards

$s_t \in \mathcal{S}$ is the state, $\mathcal{S}$ is the state space.

$a_t \in \mathcal{A}$ is the action, $\mathcal{A}$ is the action space.

$r_t = \mathcal{R}(s_t, a_t, s_{t+1})$ is the reward, $\mathcal{R}$ is the reward function

- State space, action space, and reward function are specified by the environment
- (s,a,r) is the basic unit of information describing a reinforcement learning system

RL as Markov Decision Process (MDP)

❖ General formulation

$$s_{t+1} \sim P\big(s_{t+1} \,|\, (s_0, a_0), (s_1, a_1), \ldots, (s_t, a_t)\big)$$

Next state s_{t+1} is sampled from a probability distribution P conditioned on the entire history

Challenging to model a transition function in this form if episodes last for many time steps

RL as Markov Decision Process (MDP)

❖ Markov Property

$$s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$$

Next state s_{t+1} only depends on the previous state s_t and action a_t

Current state and action at current time step contain sufficient information to fully determine the transition probability for the next state

RL as Markov Decision Process (MDP)

❖ MDP formulation of a reinforcement Learning

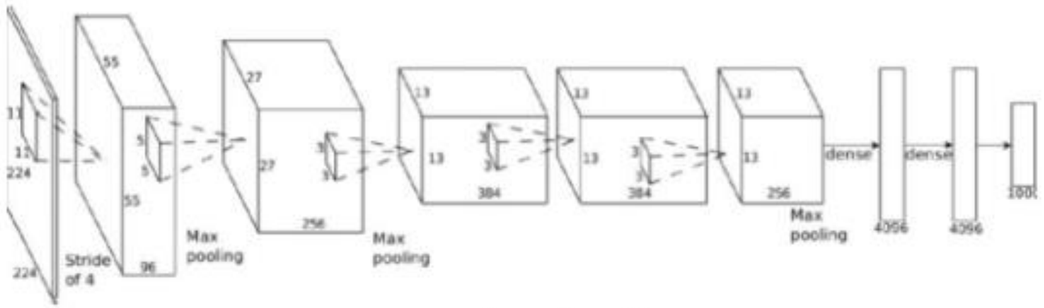
An MDP is defined by a 4-tuple $\mathcal{S}, \mathcal{A}, P(.), \mathcal{R}(.)$

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of actions.
- $P(s_{t+1} \mid s_t, a_t)$ is the state transition function of the environment.
- $\mathcal{R}(s_t, a_t, s_{t+1})$ is the reward function of the environment.

Objective of Reinforcement Learning

❖ Reward '*r*'

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

which action is better or worse?

$r(\mathbf{s}, \mathbf{a})$: reward function

tells us which states and actions are better

$\mathbf{s}$, $\mathbf{a}$, $r(\mathbf{s}, \mathbf{a})$, and $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ define Markov decision process

high reward

low reward

Objective of Reinforcement Learning

❖ Return $R(\tau)$

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$$

$\tau = (s_0, a_0, r_0), \ldots, (s_T, a_T, r_T)$: Trajectory from an episode

$\gamma \in [0, 1]$: Discount factor

Objective of Reinforcement Learning

❖ Objective $J(\tau)$

$$J(\tau) = \mathbb{E}_{\tau \sim \pi}[R(\tau)] = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

Return $R(\tau)$ is the sum of discounted rewards $\gamma^t r_t$ over all time steps $t = 0, \ldots, T$

The objective $J(\tau)$ is the return averaged over many episodes. The expectation accounts for stochasticity in the actions and the environment—that is, in repeated runs, the return may not always end up the same. Maximizing the objective is the same as maximizing the return.

Objective of Reinforcement Learning

❖ Discount factor $\gamma \in [0, 1]$

Small discount factor : “Shortsighted”

$$R(\tau)_{\gamma=0} = \sum_{t=0}^{T} \gamma^t r_t = r_0$$

Large discount factor : “Farsighted”

$$R(\tau)_{\gamma=1} = \sum_{t=0}^{T} \gamma^t r_t = \sum_{t=0}^{T} r_t$$

Expressing RL as MDP Control Loop

❖ Algorithm 1.1 expresses the interaction between an agent and an environment over many episodes and time steps

Algorithm 1.1 MDP control loop

```
Given an env (environment) and an agent:
for episode = 0, ..., MAX_EPISODE do
    state = env.reset()
    agent.reset()
    for t = 0, ..., T do
        action = agent.act(state)
        state, reward = env.step(action)
        agent.update(action, state, reward)
        if env.done() then
            break
        end if
    end for
end for
```

For each episode

Maximum time step T

Produce an action given a state

Environment produces next state and reward given the action

Encapsulating an agent's learning algorithm

Expressing RL as MDP Control Loop

- Algorithm 1.1 expresses the interaction between an agent and an environment over many episodes and time steps

Algorithm 1.1 MDP control loop

```
Given an env (environment) and an agent:
for episode = 0, ..., MAX_EPISODE do
    state = env.reset()
    agent.reset()
    for t = 0, ..., T do
        action = agent.act(state)
        state, reward = env.step(action)
        agent.update(action, state, reward)
        if env.done() then
            break
        end if
    end for
end for
```

MDP control loop collects data and performs learning internally to maximize the objective

This algorithm is generic to all reinforcement learning problems as it defines as consistent interface between an agent and an environment

Learnable Functions in Reinforcement Learning

- Three primary functions to learn in reinforcement learning

1. A policy, π, which maps state to action: $a \sim \pi(s)$

   A policy can be stochastic. That is, it may probabilistically output different actions for the same state. We can write this as $\pi(a \mid s)$ to denote the probability of an action a given a state s. An action sampled from a policy is written as $a \sim \pi(s)$.

2. A value function, $V^{\pi}(s)$ or $Q^{\pi}(s, a)$, to estimate the expected return $\mathbb{E}_{\tau}[R(\tau)]$

   The value functions provide information about the objective. They help an agent understand how good the states and available actions are in terms of the expected future return. They come in two forms—the $V^{\pi}(s)$ and $Q^{\pi}(s, a)$ functions.

3. The environment model,[3] $P(s' \mid s, a)$

   The transition function $P(s' \mid s, a)$ provides information about the environment

Learnable Functions in Reinforcement Learning

❖ Value Functions

$$V^{\pi}(s) = \mathbb{E}_{s_0=s,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$$

- Value function evaluates how good or bad a state is
- V^{π} measures the expected return from being in state *s*, assuming that the agent continues to act according to its current policy π
- It is a forward-looking measure, since all rewards received before state *s* are ignored.
- Value function V^{π} always depends on a particular policy π

Learnable Functions in Reinforcement Learning

❖ Value Functions- Simple Example

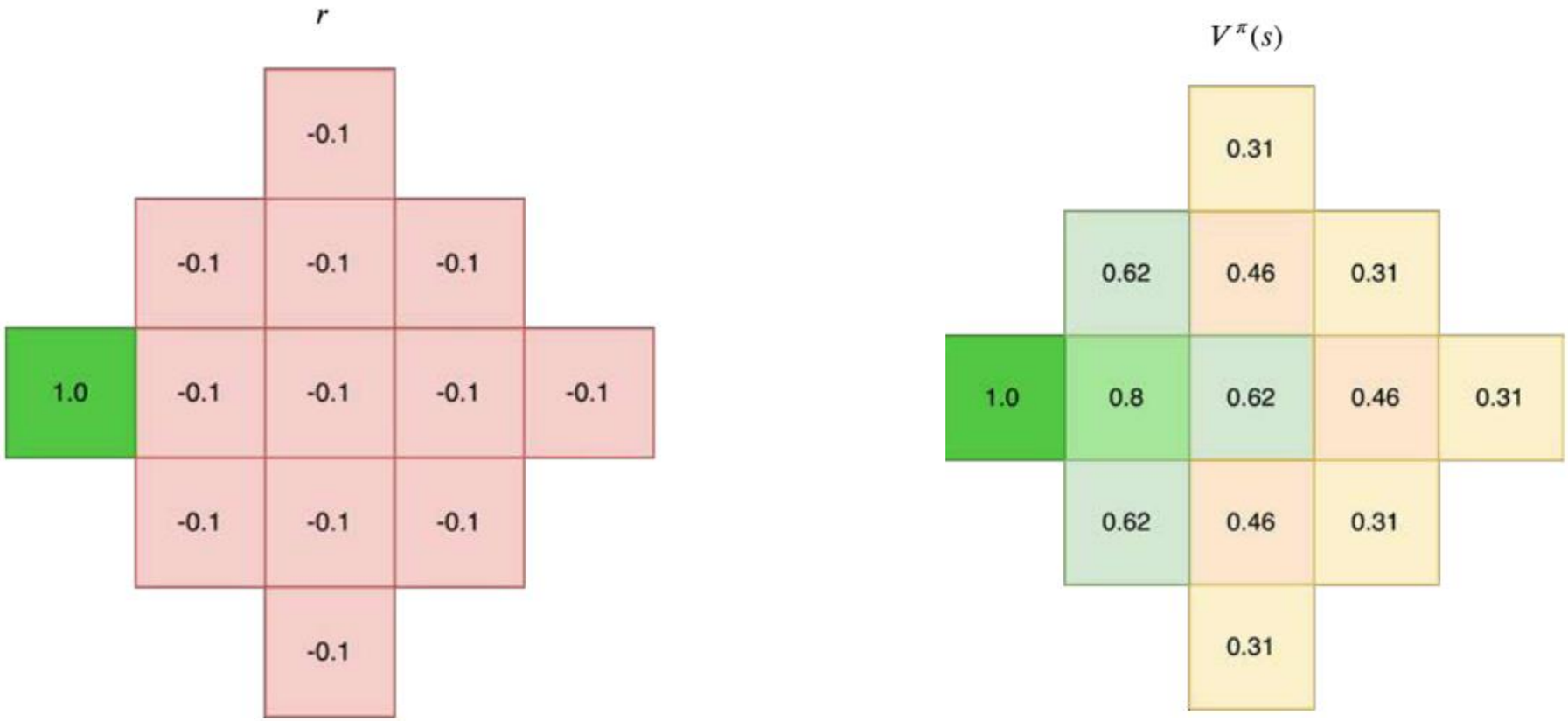


Figure 1.4 Rewards r and values $V^{\pi}(s)$ for each state s in a simple grid-world environment. The value of a state is calculated from the rewards using Equation 1.10 with $\gamma = 0.9$ while using a policy π that always takes the shortest path to the goal state with $r = +1$.

Learnable Functions in Reinforcement Learning

❖ *Q*-value function

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0=s, a_0=a, \tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

- Q-value function evaluates how good or bad a state-action pair is
- It measures the expected return from taking action *a* in state *s*
- Assumption: Agent continues to act according to its current policy π
- Q-value is measured from the current state *s* to the end of an episode

Deep Reinforcement Learning Algorithms

- Using deep neural network as the function approximation method
- Four major families of deep RL

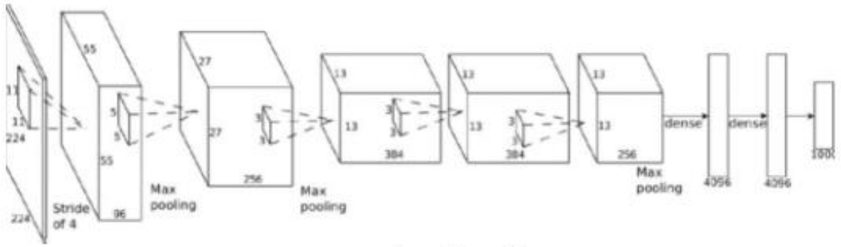
Policy-based : learning policy

Value-based : learning value functions

Model-based : learning model

Combined method : learning more than one of the functions

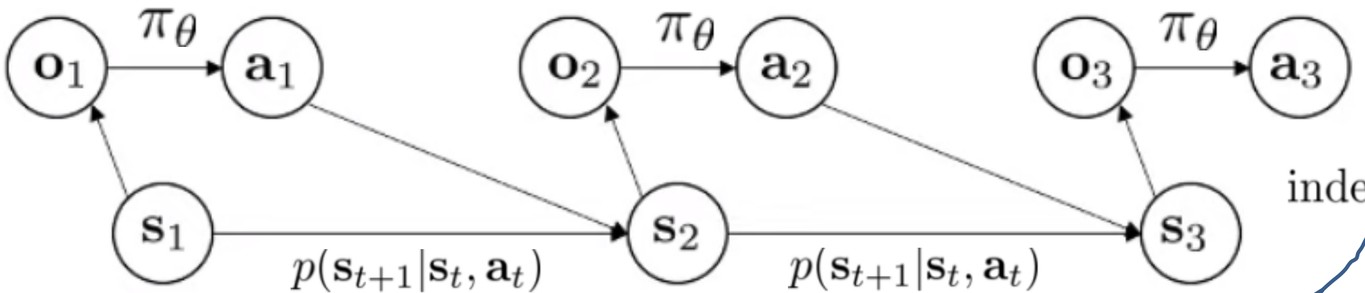
Deep Reinforcement Learning Algorithms



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



Markov property

independent of $\mathbf{s}_{t-1}$

Deep Reinforcement Learning Algorithms

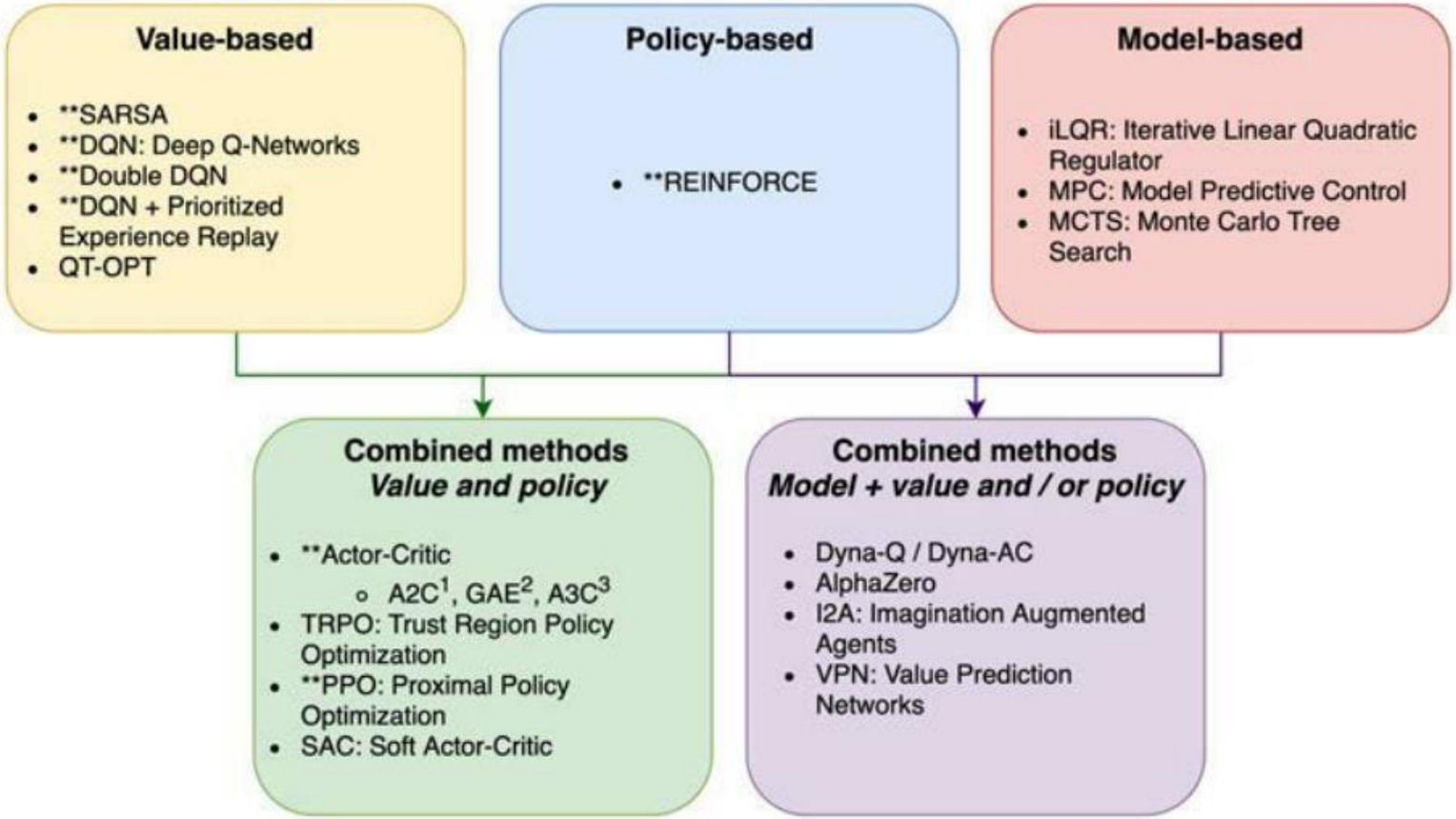


Figure 1.5 Deep reinforcement learning algorithm families

Deep Reinforcement Learning Algorithms

❖ Policy-Based Algorithms

- Good policies should generate actions which produce trajectories that maximize an agent's objective $J(\tau) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$
- "REINFORCE" is the most well-known policy-based algorithm that forms the foundation of many subsequent algorithms
- Directly optimizing the objective function
- One disadvantage is high variance and sample-inefficient

Deep Reinforcement Learning Algorithms

❖ Value-Based Algorithms

- An agent learns either $V^{\pi}(s)$ or $Q^{\pi}(s,a)$
- Using learned value function to evaluate (*s*,*a*) pairs and generate a policy
- SARSA(State-Action-Reward-State-Action)
- Deep Q-Network (DQN), Double DQN, DQN with Prioritized Experience Replay (PER)
- More sample-efficient than policy-based algorithm

Deep Reinforcement Learning Algorithms

❖ Model-Based Algorithms

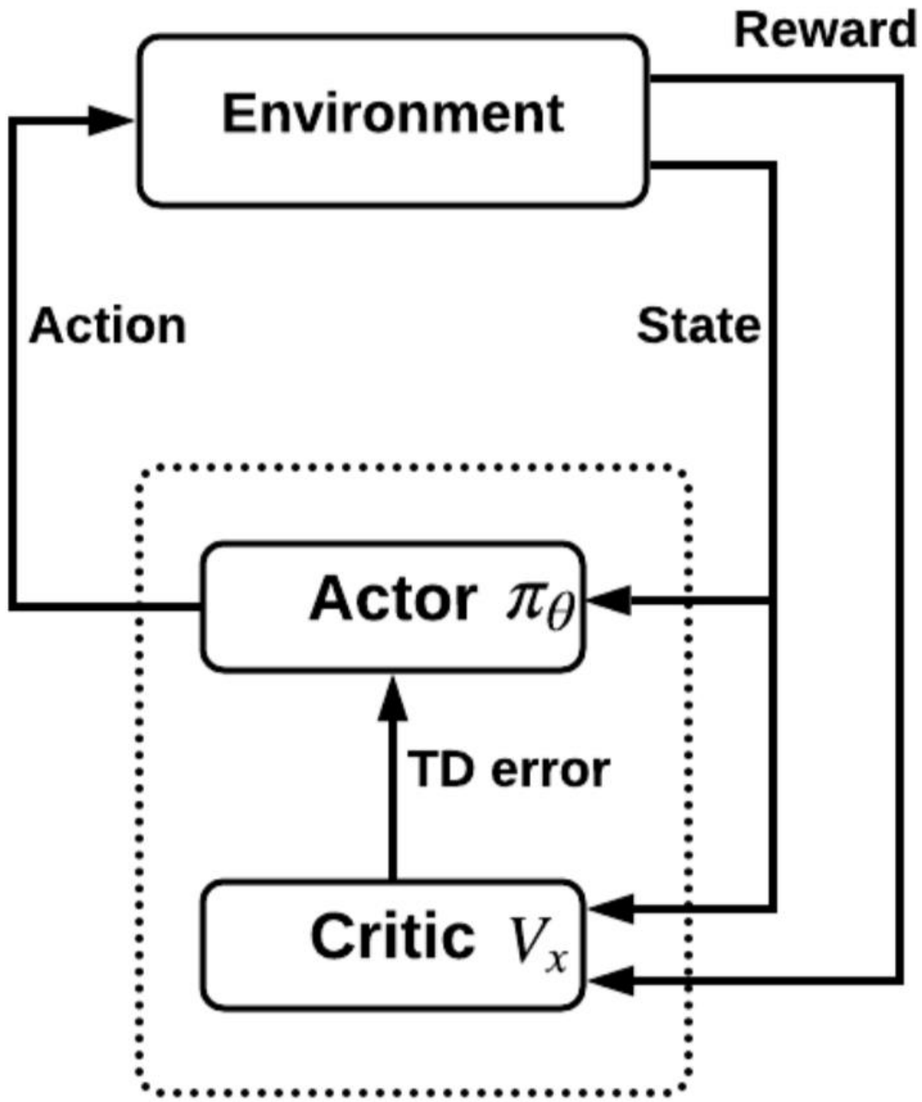
- Learning a model of an environment's transition dynamics $P(s' \mid s, a)$

- Advantage: A perfect model endows an agent with foresight
  → It can play out scenario and understand the consequence of its action without having to actually act in an environment.

- Disadvantage: For most problems, models are hard to find
  → Many environments are stochastic and their transition dynamics are not known

Deep Reinforcement Learning Algorithms

- Combined M
  - Actor-Critic al
    → Learning b
    → The policy
    → During trai
    informative
    from the e



es the actions
provide a more
d of using rewards

Deep Reinforcement Learning Algorithms

❖ On-Policy

	Value Based	Policy Based	Actor-Critic
On-Policy	• Monte Carlo Learning (MC) • TD(0) • SARSA • Expected SARSA • n-Step TD/SARSA • TD(λ)	• REINFORCE • REINFORCE with Advantage	• A3C • A2C • TRPO • PPO
Off-Policy	• Q-Learning • DQN • Double DQN • Dueling DQN		• DDPG • TD3 • SAC • IMPALA

- Requiring much more memory to store the data

Deep Reinforcement Learning Algorithms

❖ Summary

- Policy-based, value-based, model-based, or combined methods: Which of the three primary reinforcement learning functions an algorithm learns

- Model-based or model-free: Whether an algorithm uses a model of an environment's transition dynamics

- On-policy or off-policy: Whether an algorithm learns with data gathered using just the current policy

Deep Learning for Reinforcement Learning

❖ Training Procedure (as Reinforcement Learning Aspect)

- No network input *x* and no correct outputs *y* are given in advance
- Instead, these values are obtained through **agent interaction** with an environment
  → from the states and rewards agents observe
- Data exchange between agent and environment is interactive
- To generate data for training, an agent has to **experience** every time step
- Data collection and training cycle runs repeatedly with MDP control loop

Deep Learning for Reinforcement Learning

❖ Training Procedure (as Supervised Learning Aspect)

1. Sample a random batch (x, y) from dataset, where the batch size is significantly smaller than the total size of the dataset.

2. Compute a forward pass with the network using the inputs x to produce predicted outputs, $\hat{y} = f(x; \theta)$.

3. Compute the loss $L(\hat{y}, y)$ using $\hat{y}$ predicted by the network and y from the sampled batch.

4. Calculate the gradient (partial derivative) of the loss $\nabla_\theta L$ with respect to the parameters of the network. Modern neural network libraries such as PyTorch [114] or TensorFlow [1] handle this automatically using the backpropagation [117] algorithm (a.k.a. "autograd").

5. Use an optimizer to update the network parameters using the gradient. For example, a *stochastic gradient descent* (SGD) optimizer makes the following update: $\theta \leftarrow \theta - \alpha \nabla_\theta L$, where α is a scalar learning rate. However, there are many other optimization techniques available in neural network libraries.