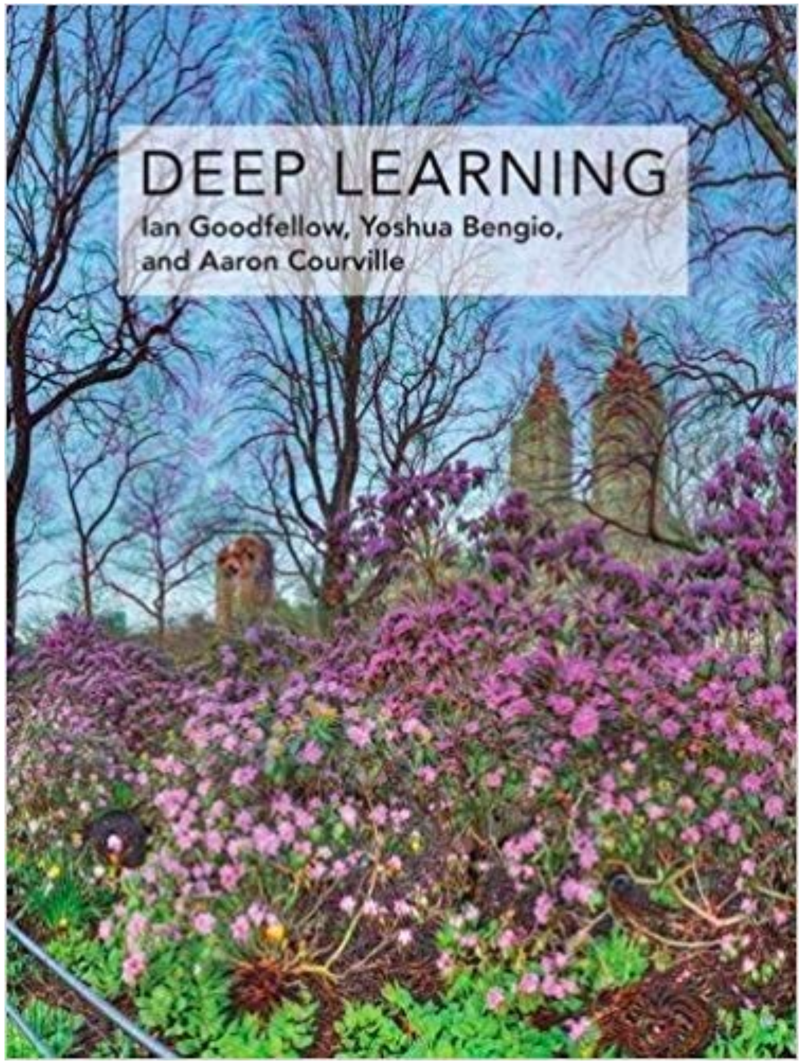
國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Deep Learning
深度學習
Fall 2023

Reinforcement Learning

Prof. Chia-Han Lee
李佳翰 教授



Reinforcement learning

- K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26-38, Nov. 2017.

- The essence of reinforcement learning (RL) is learning through interaction. An RL agent interacts with its environment and, upon observing the consequences of its actions, can learn to alter its own behavior in response to rewards received.

Reinforcement learning

- The agent interacts with the environment by taking an action $\mathbf{a}_t$ in state $\boldsymbol{s}_t$ at time step t. When the agent takes an action, the environment and the agent transition to a new state, $\boldsymbol{s}_{t+1}$, based on the current state and the chosen action.

- The state is a sufficient statistic of the environment and thereby comprises all the necessary information for the agent to take the best action, which can include parts of the agent such as the position of its actuators/sensors.

- Every time the environment transitions to a new state, it also provides a scalar reward r_{t+1} to the agent as feedback. The best sequence of actions is determined by the rewards provided by the environment.

Reinforcement learning

- The goal of an agent is to learn a policy π that maximizes the expected return. Given a state, a policy returns an action to perform; an optimal policy is any policy that maximizes the expected return in the environment.

- The challenge in RL is that the agent needs to learn about the consequences of actions in the environment by trial and error, as a model of the state transition dynamics is not available to the agent. Every interaction with the environment yields information, which the agent uses to update its knowledge.

Markov decision process

- Formally, RL can be described as a Markov decision process (MDP), which consists of
  - a set of states $\mathcal{S}$, plus a distribution of starting states $p(\mathbf{s}_0)$
  - a set of actions $\mathcal{A}$
  - the transition dynamics $\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that map a state-action pair at time t onto a distribution of states at time $t+1$
  - an immediate/instantaneous reward function $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$
  - a discount factor $\gamma \in [0,1]$, where lower values place more emphasis on immediate rewards.

Markov decision process

- In general, the policy π is a mapping from states to a probability distribution over actions π: $\mathcal{S} \rightarrow p(\mathcal{A} = \mathbf{a} | \mathcal{S})$. If the MDP is episodic, i.e., the state is reset after each episode of length T, then the sequence of states, actions, and rewards in an episode constitutes a trajectory or rollout of the policy.

- Every rollout of a policy accumulates rewards from the environment, resulting in the return $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The goal of RL is to find an optimal policy, π^*, that achieves the maximum expected return from all states:

$$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}[R \mid \pi]. \tag{1}$$

Markov decision process

- A key concept underlying RL is the Markov property—only the current state affects the next state, or, in other words, the future is conditionally independent of the past given the present state. This means that any decisions made at $\mathbf{s}_t$ can be based solely on $\mathbf{s}_{t-1}$, rather than $\{\mathbf{s}_0, \mathbf{s}_1, \cdots, \mathbf{s}_{t-1}\}$.

- Although this assumption is held by the majority of RL algorithms, it is somewhat unrealistic, as it requires the states to be fully observable.

- A generalization of MDPs are partially observable MDPs (POMDPs), in which the agent receives an observation $\mathbf{o}_t \in \Omega$, where the distribution of the observation $p(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{a}_t)$ is dependent on the current state and the previous action.

Reinforcement learning algorithms

- There are two main approaches to solving RL problems: methods based on value functions and methods based on policy search.

- There is also a hybrid actor-critic approach that employs both value functions and policy search.

Value functions

- Value function methods are based on estimating the value (expected return) of being in a given state. The state-value function $V^{\pi}(\mathbf{s})$ is the expected return when starting in state $\mathbf{s}$ and following π subsequently:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}\left[R \middle| \mathbf{s}, \pi\right]. \tag{2}$$

- The optimal policy, π^*, has a corresponding state-value function $V^*(\mathbf{s})$, and vice versa; the optimal state-value function can be defined as

$$V^*(\mathbf{s}) = \max_{\pi} V^{\pi}(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S}. \tag{3}$$

- If we had $V^*(\mathbf{s})$ available, the optimal policy could be retrieved by choosing among all actions available at $\mathbf{s}_t$ and picking the action $\mathbf{a}$ that maximizes $\mathbb{E}_{\mathbf{s}_{t+1}\sim\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a})}[V^*(\mathbf{s}_{t+1})]$.

Value functions

- In the RL setting, the transition dynamics $\mathcal{T}$ are unavailable. Therefore, we construct another function, the state-action value or quality function $Q^{\pi}(\mathbf{s},\mathbf{a})$, which is similar to V^{π}, except that the initial action $\mathbf{a}$ is provided and π is only followed from the succeeding state onward:

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \mathbb{E}[R \mid \mathbf{s},\mathbf{a},\pi]. \quad (4)$$

- The best policy, given $Q^{\pi}(\mathbf{s},\mathbf{a})$, can be found by choosing $\mathbf{a}$ greedily at every state: $\mathrm{argmax}_{\mathbf{a}} Q^{\pi}(\mathbf{s},\mathbf{a})$. Under this policy, we can also define $V^{\pi}(\mathbf{s})$ by maximizing $Q^{\pi}(\mathbf{s},\mathbf{a})$: $V^{\pi}(\mathbf{s}) = \max_{\mathbf{a}} Q^{\pi}(\mathbf{s},\mathbf{a})$.

Dynamic programming

- To actually learn Q^{π}, we exploit the Markov property and define the function as a Bellman equation, which has the following recursive form:

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}}[r_{t+1} + \gamma Q^{\pi}(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))]. \qquad (5)$$

- This means that Q^{π} can be improved by bootstrapping, i.e., we can use the current values of our estimate of Q^{π} to improve our estimate. This is the foundation of Q-learning and the state-action-reward-state-action (SARSA) algorithm:

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta, \qquad (6)$$

where α is the learning rate and $\delta = Y - Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ the temporal difference (TD) error; here, Y is a target as in a standard regression problem.

Dynamic programming

- SARSA, an on-policy learning algorithm, is used to improve the estimate of Q^{π} by using transitions generated by the behavioral policy (the policy derived from Q^{π}), which results in setting $Y = r_t + \gamma Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$.

- Q-learning is off-policy, as Q^{π} is instead updated by transitions that were not necessarily generated by the derived policy. Instead, Q-learning uses $Y = r_t + \gamma \max_{\mathbf{a}} Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a})$, which directly approximates Q^{*}.

Dynamic programming

- To find Q^* from an arbitrary Q^π, we use generalized policy iteration, where policy iteration consists of policy evaluation and policy improvement.

- Policy evaluation improves the estimate of the value function, which can be achieved by minimizing TD errors from trajectories experienced by following the policy.

- As the estimate improves, the policy can be improved by choosing actions greedily based on the updated value function.

- Instead of performing these steps separately to convergence (as in policy iteration), generalized policy iteration allows for interleaving the steps, such that progress can be made more rapidly.

Sampling

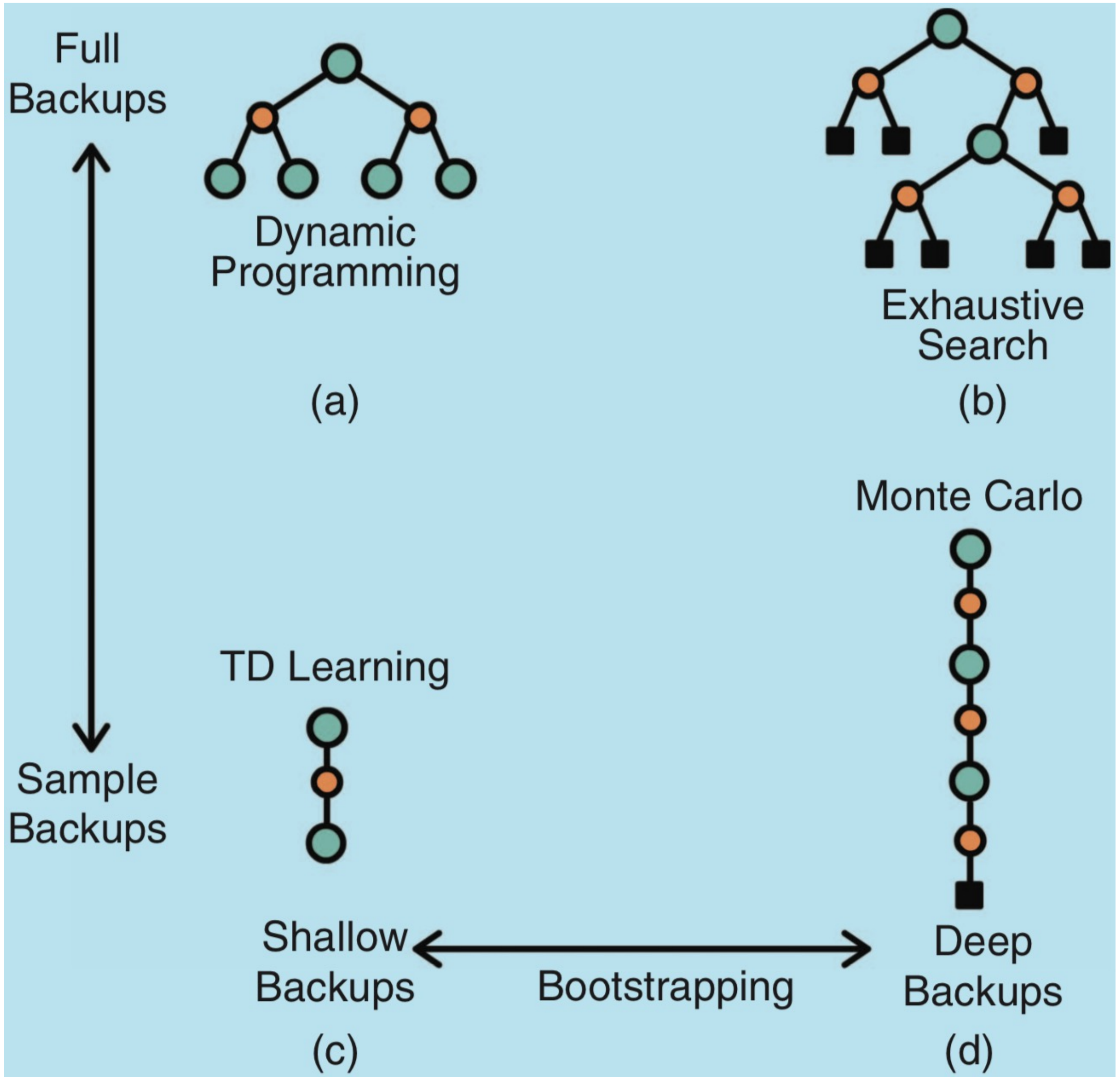
- Instead of bootstrapping value functions using dynamic programming methods, Monte Carlo methods estimate the expected return

$$V^{\pi}(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi]. \qquad (2)$$

from a state by averaging the return from multiple rollouts of a policy.

- Another major value-function-based method relies on learning the advantage function $A^{\pi}(\mathbf{s}, \mathbf{a})$. A^{π} represents a relative advantage of actions through the simple relationship $A^{\pi} = Q^{\pi} - V^{\pi}$. It is easier to learn that one action has better consequences than another than it is to learn the actual return from taking the action.

Sampling


Full Backups
Sample Backups
Dynamic Programming
(a)
Exhaustive Search
(b)
TD Learning
Monte Carlo
Shallow Backups
Bootstrapping
Deep Backups
(c)
(d)

Policy search

- Policy search methods directly search for an optimal policy π^*.

- Typically, a parameterized policy π_θ is chosen, whose parameters are updated to maximize the expected return $\mathbb{E}[R|\theta]$ using either gradient-based or gradient-free optimization.

- Neural networks that encode policies have been successfully trained using both gradient-free and gradient-based methods. Gradient-free optimization can effectively cover low-dimensional parameter spaces, but gradient-based training remains the method of choice for most DRL algorithms, being more sample efficient when policies possess a large number of parameters.

Policy search

- With gradient-free methods, finding better policies requires a heuristic search across a predefined class of models. Methods such as evolution strategies essentially perform hill climbing in a subspace of policies, while more complex methods, such as compressed network search, impose additional inductive biases.

- Gradients can provide a strong learning signal as to how to improve a parameterized policy. However, to compute the expected return

$$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}[R \mid \pi]. \qquad (1)$$

we need to average over plausible trajectories induced by the current policy parameterization. This averaging requires either deterministic approximations (e.g., linearization) or stochastic approximations via sampling.

Planning and learning

- Given a model of the environment, it is possible to use dynamic programming over all possible actions, sample trajectories for heuristic search (as was done by AlphaGo), or even perform an exhaustive search.

- Model-free RL methods learn directly from interactions with the environment, but model-based RL methods can simulate transitions using the learned model, resulting in increased sample efficiency. This is important in domains where interaction with the environment is expensive.

- However, learning a model introduces extra complexities, and may suffer from model errors, which in turn affects the learned policy.

The rise of deep RL

- Many of the successes in DRL have been based on scaling up prior work in RL to high-dimensional problems. This is due to the learning of low-dimensional feature representations and the powerful function approximation properties of neural networks. By means of representation learning, DRL can deal efficiently with the curse of dimensionality.

- For instance, convolutional neural networks (CNNs) can be used as components of RL agents, allowing them to learn directly from raw, high-dimensional visual inputs.

- In general, DRL is based on training deep neural networks to approximate the optimal policy π^* and/or the optimal value functions V^*, Q^*, and A^*.

Function approximation and DQN

- The deep Q-network (DQN) was the first RL algorithm that was demonstrated to work directly from raw visual inputs and on a wide variety of environments.

- The DQN, consisted of convolutional layers followed by fully connected layers, outputs $Q^{\pi}(\mathbf{s},\cdot)$ for all action values in a discrete set of actions. This not only enables the best action, $\mathrm{argmax}_{\mathbf{a}} Q^{\pi}(\mathbf{s},\mathbf{a})$, to be chosen after a single forward pass of the network, but also allows the network to more easily encode action-independent knowledge in the lower, convolutional layers.

- The strength of the DQN lies in its ability to compactly represent both high-dimensional observations and the Q-function using deep neural networks.