Chapter 1
Introduction

Abstract Reinforcement Learning is widely used in various industries, including business, IT, Pharmacy, government, etc. Along with the development and thriving of WWW and other important computer techniques, such as IOT and neural networks, the breadth and depth of Reinforcement Learning have improved significantly. In this chapter, we highlight the evolution of Reinforcement Learning. Discuss the use of Reinforcement Learning in our daily life and beyond. Finally, we briefly discuss advanced topics in the area and leave the details to be elaborated on in later chapters.

1.1 Evolution of Reinforcement Learning

Reinforcement learning is a comparatively independent branch of machine learning where an agent learns to conduct through experience by interacting with an environment and receiving rewards for its actions. Reinforcement learning nowadays dates back to the early last century. From the early 19th century to the middle of the previous century, trial and error, which originated from the psychology of animal learning, dominated in the area and, as the earliest work in artificial intelligence, led to the revival of reinforcement learning in the early 1980s. In the meantime, 'optimal control' originated from the control theory and came into the area in the late 1950s. The works focused on optimization problems which aimed to design a solution to minimize or maximize the overall reward of a sequence of interactive activities to achieve a predefined goal. The details of the evolution of early reinforcement learning are elaborated in [21].

Modern reinforcement learning integrates advances in neural techniques such as deep learning and software and hardware improvements such as large-scale distributed computation network. The innovations greatly improve the system's efficiency and accuracy, which made the large-scale RL applications realistic.

In the following subsections, we introduce several main RL branches, each addressing different aspects and difficulties in the area, including meta reinforcment learning (MRL), hierarchical reinforcement learning (HRL), multi-task reinforcement learning (MTRL) and deep reinforcement learning (DRL). The mathematical fundations and details about RL and its subfields are presented in the next chapter.

1.1.1 Basic Reinforcement Learning

Basic reinforcement learning addresses the problem of finding the optimum interactive actions to achieve predefined goals, which are usually long-term, and the related issues about agent-environment interactions, reward estimation of interactive actions, value functions, and agent policies. Most reinforcement learning works assume the agent-environment interactions follow the MDP (Markov Decision Processes), because of the mathematic completeness of such frameworks. More works on non-MDP stood out recently to expand the scope of reinforcement learning [9], [13] and [15].

1.1.1.1 Main Components

The main components in a reinforcement learning problem include environment and agent with auxiliary components defining the two and their interactions, such as action, reward, observation, state, and policy. Figure. 1.1 shows the simplified diagram of an RL problem.

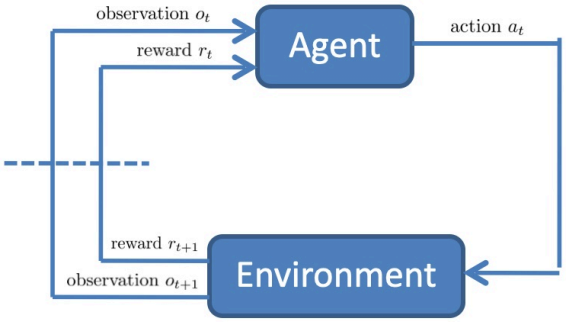


Fig. 1.1: Simplified diagram of reinforcment learning.

Environment

Environment refers to the universe with which the agents interact. It can be either physical or abstract or both. In computational reinforcement learning, the environment is typically formulated as an MDP because agent behaviors are assumed to be Markovian. Rewards, observations, and environment states measure the way of agent-environment interactions. We assume the environment of an RL problem is a singleton for all elements related to the problem. The following gives examples of environment in various problems:

- Board Game: the environment is everything in the game, including the entire board, the game pieces, and your opponents.
- Robotic Systems: the environment is the physical world where the robot is trained to conduct certain tasks. It also includes other entities that affect the actions of

the robot, such as other robots in the same physical world, utilities the robot can use to complete the tasks, etc.
- Web Applications: the environment is the Internet, including web servers and network infrastructure on which the agent works. Users interact with the application on the Internet. The environment can be huge and includes millions of different network components. Usually simplified web environment models that are tested by various web applications are used to make the learning process realistic and efficient.
- Finance Trading Systems: the environment is the financial market the agent is trading in. And for some complex training systems, also includes everything that influences the market, such as the supply and demand of trade, political policies, country-specific conditions like GDP.

Most reinforcement learning software, including gym and tf-agent, provide popular environments and functionalities to develop customized ones. Popular commercial software of reinforcement learning environments include Neural MMO [20] which was released by OpenAI, Android Env [24] and MuJoCo [23] which are developed by DeepMind.

Agent

In a reinforcement learning problem, an agent is the main body interacting with the environment [12] autonomously or controlled. An agent can usually observe the environment, take action, and receive feedback or reward.

The following gives examples of agents in the same set of problems as we discussed before:

- Board Game: the agent is the player of the board game.
- Robotic Systems: an intelligent robot who is capable of conducting a particular task, usually in a restricted or designed environment.
- Web Applications: a software instance on the web, which may further control other intermediate objects, to complete predefined web tasks. For instance, in a CDS (Content Distribution System), an agent can be a recommender instance that tells the webpage controller the optimum set of visual contents, retrieved from the CDS database, to display.
- Finance Trading Systems: a software instance that make trade decisions and order execution to maximize the trading profit.

Action

In an RL system, actions define the things an agent can do in the reinforcement learning system. An agent is associated with a set of actions that are usually predefined. The actions of an agent can be discrete, continuous, or hybrid. A continuous action set can assign a numeric value to a particular action to identify a particular action. For instance, in a wheel control system, the action of the wheel controller is a turn of a particular angle.

The following gives examples of actions in the same set of problems as we discussed before:

- Board Game: the actions are usually predefined movements that a player can make on a particular game piece on the game board. The set of actions are different for different board games. Chess, or instance, an action can be moving a pawn forward by one step.
- Robotic Systems: actions a robot can do depend on the task to be trained. For a house-keeping robot, the actions may include moving in multiple directions like forward, backward, right, left and diagonally; cleaning the dust within its reach, picking up a trash, etc.
- Web Applications: actions a web application can do depends on the specific application. Web navigation, for instance, the actions may include recommend a list of webpages to watch, display a webpage from its clicked link, move backwards and navigate to a previous webpage, etc.
- Finance Trading Systems: actions can be place and cancel various trading orders.

Reward

A reward measures the goodness of a action or a group of related actions at a particular time or step to a goal. Rewards are usually discounted over time to take into account the time effect on the influence of a passed action to the final goal. Particularly, a discounted reward is measured by the original reward multiplied by a discounted factor.

Reward is the direct force that drives the agents' learning process. Correctly extract the rewards from observations on the environmental feedback is important to the learning convergence. If the reward mechanism is not modeled accurately or noisily, or even slightly off the course from the primaty goal, there is a chance that training will diverge or go in a wrong direction.

Observation

Observation is the information about the environment, including that about the agents in the environment, that an agent receives from the environment or collected intentionally from the environment. An observation usually captures the information or partial information about an agent (or a group of agents in multi-agent reinforcement learning) and the environment at a particular time.

Observations may or may not be relevant to the upcoming reward, e.g. about the environmental influence of a past action. They can be either in a determined format like numerical numbers or in a unstructured form such as an image of the agent or environment in a certain state. For the later, preprocessing on the observations are needed to extract useful information about the states related.

The following gives examples of observation in the same set of problems as we discussed before:

- Board Game: observations are what you see about the current board positions. The perception of the opponent's mood helps increase the chance to win.
- Robotic Systems: observations are the vision images of the scene and utilities. Information about action rewards and robot states is usually extracted from these observations.
- Web Applications: observations can be users' feedback on the fitness of the webpages such as the average time users spend on a particular web page.
- Finance Trading Systems: observations can be the price trends of various stocks and alternatives trading in the market.

State

A state encodes the important observations that define the status of an agent at a particular time in the environment. The states are evaluated and assigned a value to measure their importance to obtain a higher intermediate reward or achieve the final goal.

The following gives examples of state in the same set of problems as we discussed before:

- Board Game: a state can describe the current status of the board game like the current deployment of pieces on the board.
- Robotic Systems: a state can be the robot's current position in the scene which partially determines the next optimum action the robot will take.
- Web Applications: a state can be the average view time of a web page during a period, it affects the possibility that the web page is recommended again immediately after the period.
- Finance Trading Systems: a state can be the current stock price of a target stock to sell or buy.

Policy

A policy is a particular strategy an agent takes to complete a task or achieve a goal, or in other words an agent's way of behave. Formally, a policy is a mapping that maps each state and action pair to the probability of taking the action in the state. An agent can apply multiple policies to achieve the same goal. For instance, in financial trading, a trader can buy and sell different stocks, to achieve a profit goal while minimizing the number of transactions. The ones selected are sually optimal or suboptimal for exploitation and random or ϵ greedy for exploration.

The following gives examples of policy in the same set of problems as we discussed before:

- Board Game: a policy is a player's strategy to beat his/her opponent. In a chess game, for instance, a policy can be a progressive one that coerces the opponent into creating an isolated queen's pawn and centers the play around this structural weakness in the opponent's pawns. Or, it can be a defensive one that ensures the king's safety.

- Robotic Systems: a policy can be a preset routine on the robot to complete the assigned task.
- Web Applications: for a particular web task, a policy can be a designed software algorithm to complete the web task.
- Finance Trading Systems: a policy can be a portfolio strategy to maximize the trader's overall profit over time.

1.1.2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) is the problem of inferring the reward function of an agent, given its policy or observed behavior. The promising of IRL lies in the possibility of training intelligent agents, who can learn to conduct and are capable of influence others, using recorded data in performing other related tasks. It's intuitive that the rewards learned by IRL are more accurate and can be better generalized than rewards by manual specification.

Figure 1.2 gives an example of applying rewards learned via IRL.

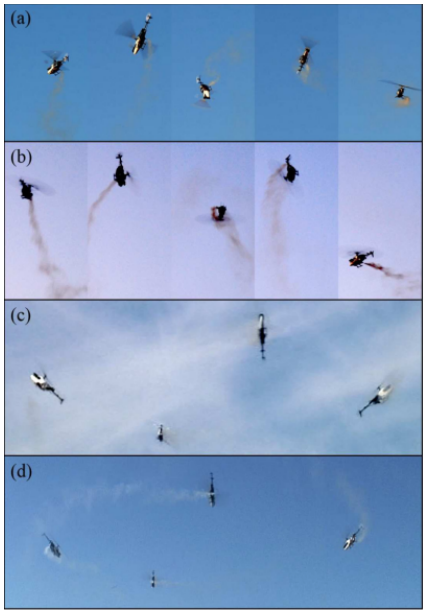


Fig. 1.2: Sample simulation of complex helicopter maneuver using a reward function learned from expert pilots through IRL. The image is copied from [1].

IRL has a wide scope of applications in the problems where reward function is unknown or partially known. Especially for problems with complex rewarding mechanisms that are difficult to specify manually in the areas of automation, animation, and economics for instance. IRL Learning of computational models of human and animal behaviors are discussed in [17], [4] and

The learing approaches can be categorized into two types, according to how the input experiences are utilized. One type of approches focus on the learning of policies that can reproduce the input behaviors, and bypass the learning of reward function or learning the reward function as an intermediate step. Another type of approches focus on the approximation of the reward function from the input data. While the latter approches allow a better generalization of the task at hand, sometimes the learned reward function cannot fully reproduce the demonstrated behaviors.

IRL is also closely related to other RL-related areas where learning a reward function is needed. For instance, the applications of IRL in apprenticeship learning have enhanced the performance of learning algorithms. The reward functions learned from experts are used to teach juniors to perform the same or similar tasks. Optimal and sub-optimal policies generated from the learned reward functions are usually adopted in the teaching process. The real-world problems include helicopter flight control, socially adaptive navigation, boat sailing, and learning driving styles [3].

1.1.3 Multi-agent Reinforcement Learning

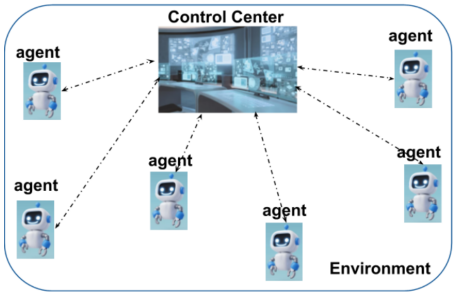
Multiagent reinforcement learning studies the problems where multiple agents are involved in a common environment. The types of agents may be different. The interactions among the agents are generally modeled under the framework of game theory. Agents can share observations, episodes, value functions, and/or policies. In an environment with pure cooperative agent-agent relationships, agents cooperate to perform certain tasks or produce certain outcomes; while in an environment with pure competitive agent-agent relationships, agents compete with each other to perform certain tasks or produce certain outcomes.

The learning convergence of the former results in a win-win situation where all agents are beneficiaries. The learning convergence of the latter results in a win-lose situation where only one agent perceives the outcome as positive while the learning divergence may lead to a no-win situation. And usually only part of the agents (may or may not be in groups) perceive the outcomes as positive. In between the two types of environments, both cooperative and competitive agent-agent relationships can exist. A comprehensive survey on MARL can be found in [6]

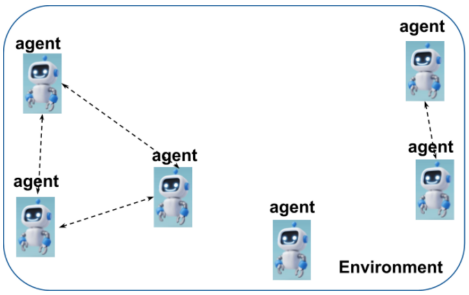
In a MARL problem, the agent-agent communications can be centralized, distributed, or hybrid as demonstrated in Figure 1.3.

Multi-agent RL can be used to address problems of learning to do complex tasks in a variety of domain, including automation, robotics, web applications and economics. In such tasks, predefined agents behaviors are too simplified to handle the complex and uncertain agent-agent and agent-environment interactions. Instead, adaptation of dynamic behaviors using learning is required to complete the tasks.

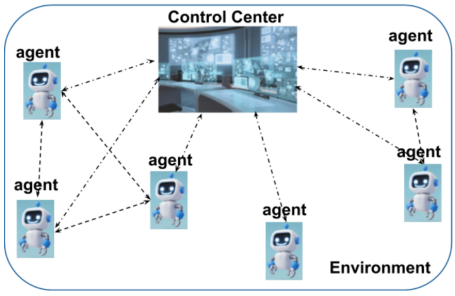
Advantages



(a) A sample MARL system with a centralized agent-agent communication/interaction structure. In such a setting, a centralized controller is in charge of information process, exchange, and distribution among agents. For instance, all actions, rewards, and observations may be collected from the agents involved, and resulting policies are distributed to individual agents.



(b) A sample MARL system with a decentralized/distributed agent-agent communication/interaction structure. In such a setting, agents are fully distributed, they only exchange information and interact directly with each other as necessary.



(c) A sample MARL system with a Hybrid agent-agent interaction structure. In such as setting, agents can communicate and interact with each other either directly, through a centralized controller shared among the group, or both.

Fig. 1.3: Sample MARL systems with three different agent-agent interaction structures.

By modeling multiple agents simultaneously, speed-up can be easily realized by software and hardware parallel computation, especially for a setting of a decentralized structure of agents for the same task or set of tasks.

Multi-agent RL benefits also from sharing agents' experiences through communication, teaching, and monitoring. The experience sharing further makes the learning process robust, in the sense of rate of learning convergence thanks to the redundancy from multiple agents learning the same or similar tasks. For instance, when one or more agents fail to learn a certain task, other agents with better knowledge about the same or similar task can take over the remaining tasks. And the failed agents can catch up using shared knowledge from the other agents.

Challenges

The out-performance of MARL over Single-agent RL is demonstrated by existing works. However, the benefits of MARL mentioned above require additional preconditions and theoretical foundations which are usually missing in the literature, although relaxing them can improve the performance of MARL.

The computational complexity of MARL is approximately exponential in the number of agents, considering the different agent behaviors and agent-agent interactions that add their variables to the state and action spaces. This makes the curse of dimensionality more difficult to break than in single-agent RL.

The additional challenges of MARL, besides those of traditional RL such as the dimensionality curse, possible irregular policy drifting along time, and balance between exploration and exploitation, including the difficulty in modeling the diversified agent-agent (e.g. cooperative vs competitive vs hybrid) and agent-environment interactions, the non-stationary nature of the system due to those dynamic interactions.

The joint-learning mechanism in MARL, which models the cooperation between agents comes from the fact that the behaviors of any agent depend on both the environment and other agents in the same environment which may or may not interact with the agent directly. Proper design of when and how to model such coordination can take time and effort. For instance, in competitive settings, cooperation may also be beneficial for self-interested agents, e.g. in a situation where the lack of collaboration negatively affects the agents involved. In cooperative and hybrid settings, the inherent conflicts between self-interested agents and the cooperation goal make it difficult to specify or learn the coordination policies.

1.1.4 Meta Reinforcement Learning

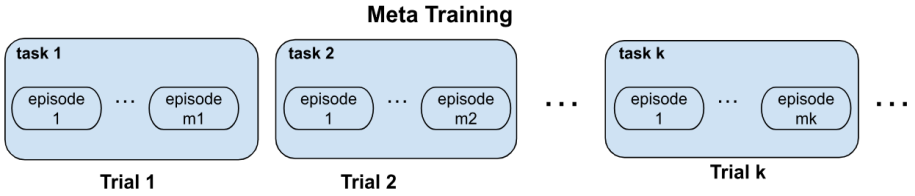


Fig. 1.4: A sample meta-training process of an MRL problem with k tasks. The learning process is carried out in N trials, with training trials on the k tasks repeatedly.

The correct settings of several meta-parameters are crucial to the success of reinforcement learning. Meta-reinforcement learning addresses the problems of how to set and adjust these meta-parameters to enhance the learning performance [18]. Meta reinforcement learning can involve multi-tasks sharing the same set of meta parameters. Then, the training or learning process addresses how to tune and adjust

the meta parameters to handle a new task efficiently. Particularly, as shown in Figure 1.4, in the meta-training process, multiple trials are carried out with each trial performing a certain task and tuning a particular set of meta parameters. multiple episodes are usually handled in each trial. A recent survey on meta-reinforcement learning can be found in [5].

One direction of MRL aims to learn exploration policies that solve challenging tasks with insufficient learning data like sparse or delayed rewards [10]. The works can be divided into two main categories, gradient descent and policy based learning approaches. The former approaches tune the policy model parameters with gradient descent for new tasks. The latter approaches learn a policy network which can be finetuned for new tasks.

1.1.5 Hierarchical Reinforcement Learning

RL is cursed by the high dimensionality of the state space and/or the action space. That is the number of learning parameters grows exponentially with the increase of the size of the state space and/or of the size of the action space. Hierarchical reinforcement learning (HRL), in which multiple layers of policies are trained to perform decision-making and control, has being promising to solve such complex RL problems with a high-dimensional state and/or action space.

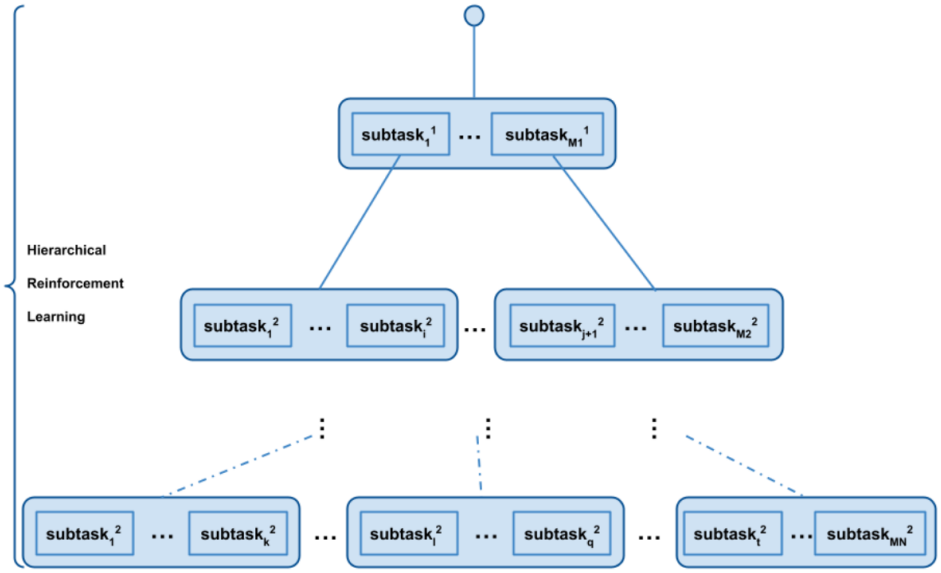


Fig. 1.5: A sample hierarchical reinforcement learning process with N level of policy learning. In particular, the learning task is divided into N levels, each level has M_i subtasks, The number of subtasks in level i is the summation of subtasks of the tasks in the previous level i-1, with possible duplicated substasks across subtask groups and levels.

Generally, the learning task is divided into a set of hierarchically structured subtasks. Each subtask is associated with a set of actions and states from the original action and state spaces. A particular subtask may be traced back from multiple parent subtasks in the immediate higher level. 1.5 shows an example diagram of an HRL problem.

The hierarchical structure of HRL can be based on policy or value function. The value-based HRL organizes the subtasks hierarchically with constraints on only actions/subtasks to be conducted on each node. The reduction of problem complexity is limited, in the sense that subtasks may share a large portion of overlapped policies.

The policy-based HRL hierarchically structures MDP policies by directly restricting the class of policies feasible at a particular state and after. The hierarchical abstract machine approach [16] applies HAM constraints on the state exploration and thus reduces the invalid search. In [22], a particular policy-based approach is proposed. It introduced the concept of option to illustrate the framework neatly. An option consists of a policy and a termination condition. A set of states are predefined or learned to associate with options to be selected. If an option is chosen at a particular state, the actions followed are conducted according to the related policy until the option termination condition is met.

1.1.6 Multi-Task Reinforcement Learning

In most specific domains, such as robotic manipulation and locomotion, many individual tasks share a common structure that can be reused to acquire related tasks more efficiently. For example, most robotic manipulation tasks involve grasping or moving objects in the workspace [26]. Multi-task reinforcement learning solves a more general problem of efficient learning on multiple tasks simultaneously or under the same learning framework. For MRL problems, multiple tasks usually share or partially share common structures that can be parameterized jointly to some extent. The learned policy must maximize the weighted average of expected returns across all tasks.

In special cases, tasks can be grouped into sub-groups hierarchically, and the learning problem is solved similarly to HRL, where subtasks originated from the same subtask at the higher level share or partially share the same set of learning parameters. Survey [25] discussed the integration of deep presentation learning into MRL to learn the shared value and policy parameters among tasks efficiently.

1.2 Deep Reinforcement Learning

Deep reinforcement learning uses deep neural networks to represent one or more of value function, policy, model under the RL framework. A combination of multiple schemes is designed to utilize their unique advantages. Figure 1.6 shows a training architecture with both policy and value networks.

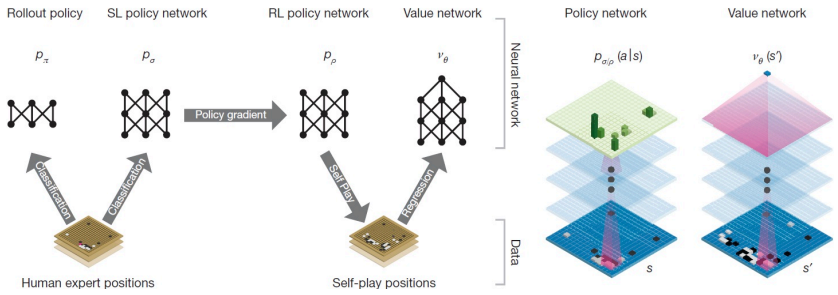


Fig. 1.6: A sample training architecture which learns policy and value function using deep neural networks. The picture is originated from [19].

Deep Q-networks (DQNs) and their variations and extensions including N-step DQN, Double DQN, Dueling DQN, and Categorical DQN [12] are frequently used for value approximation. These networks are usually composed of multiple convolutional layers for feature representation, followed by task layers such as fully connected layers, pooling layers, and a scoring layer to output the estimated Q values. The inputs are usually raw or processed images and other data types such as data points and videos that represent the underlying states and observations. The outputs are the estimated state and/or (state, action) values. The majority of existing works use supervised learning with backpropagation to train the network to generate desired network values. But any general training methods for neural networks such as Levenberg-Marquardt algorithm adapted for neural networks can be used [14] to obtain the optimum network parameters.

1.3 Applications of Reinforcement Learning

Reinforcement learning has a wide range of real-world applications. From small-sized local physical systems, which may serve a single human, to large-scale software systems, which serve millions and even billions of users globally [8].

Physical systems that are backed with RL modules and chips can range in size from a small drone [2] to a single data center that serves a local community, to a group of data centers connected virtually to serve a larger region. The complexity of these systems usually increases with the size of the system. But in complexity, for a standalone system, they can range from a one-dimensional thermostat [11] to a self-driving car. In cost, it can range, for a standalone system, from several dollars for a calculator to millions of dollars for a spaceship.

Intelligent software systems that are engined by RL algorithms and frameworks range from on-device controllers for individual smartphones to million-user and even larger recommendation systems [7] which are usually web-based. These software systems can optimize the battery profile of a single device or schedule millions of software jobs over a distributed global computer network. The codebase might be a

simple kernel module with thousands of lines for standalone systems to millions of lines of code for large-scale systems.

Most real-world RL systems include both RL software for cost efficiency and better scalability and RL hardware for performance and local controllability. In contrast designing an RL simulator for a simulated perfect environment with deterministic system dynamics, where the agents and environments are fully visible and no or little consequence of bad actions, difficulties in the development of these real-world systems include inherent latencies, system noise, non-stationarities of the agent behaviors, large state and action spaces, and large training and deployment costs due to the complexity of the system and/or the high requirement of performance accuracy and efficiency.

Reinforcment learning is also largely used in system simulations, e.g. to evaluate effects or consequences of certain behavioral changes of agents, environment changes, and policy changes, by simulating the agents behaviors under those changes.

1.4 Summary

In this chapter, we present the basic concepts in reinforcement learning, highlight the evolution, and introduce several main branches including multi-agent reinforcement learning, inverse reinforcement learning, meta reinforcement learning, hierarchical reinforcement learning, multi-task reinforcement learning and deep reinforcement learning. Then, we briefly discuss the real-world applications of reinforcement learning with opportunities and challenges.

In the next chapter, mathematical foundations of reinforcement learning and deep learning are summarized.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006.

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[3] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[4] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[5] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

[6] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[7] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[8] G Dulac-Arnold, N Levine, DJ Mankowitz, and etc. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110:2419–2468, 2021.

[9] Maor Gaon and Ronen Brafman. Reinforcement learning with non-markovian rewards. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3980–3987, 2020.

[10] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31, 2018.

[11] Todd Andrew Hester, Evan Jarman Fisher, and Piyush Khandelwal. Predictively controlling an environmental control system, January 16 2018. US Patent 9,869,484.

[12] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[13] IA Luchnikov, SV Vintskevich, DA Grigoriev, and SN Filippov. Machine learning non-markovian quantum dynamics. *Physical review letters*, 124(14): 140502, 2020.

[14] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*, pages 105–116. Springer, 2006.

[15] Daniel Neider, Jean-Raphael Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. Advice-guided reinforcement learning in a non-markovian environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9073–9080, 2021.

[16] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10, 1997.

[17] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103, 1998.

[18] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[20] Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[22] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

[24] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

[25] Nelson Vithayathil Varghese and Qusay H Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

[26] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1095. PMLR, 2020.