Chapter 1
Introduction

Abstract Reinforcement Learning is type of machine learning where an agent learns by interacting with its environment. It is widely used in various industries, including game playing, finance, healthcare, 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 from early stage to very recent developments. Then, we we briefly discuss advanced topics in the area and leave the details to be elaborated on in later chapters. Finally, we introduce the applications of Reinforcement Learning in our daily life and beyond.

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 [50].

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 [16], [32] and [37].

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.

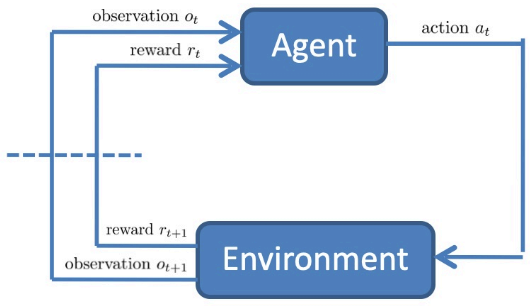


Fig. 1.1: Simplified diagram of reinforcment learning.

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 [48] which was released by OpenAI, Android Env [56] and MuJoCo [55] which are developed by DeepMind.

Agent

In a reinforcement learning problem, an agent is the main body interacting with the environment [26] 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 the fundamental signal that guides an agent's learning in reinforcement learning. They are numerical values that indicate the desirability or undesirability of a particular outcome or action. The agent's goal in reinforcement learning is usually to maximize its cumulative reward over time. 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.

Traditionally, reward is a numerical value and the higher rewards indicate more desirable outimes. Reward is the direct force that drives the agents' learning process. When a reward is received, it can be immeidate or delayed. Some environments produce rewards frequently and immeidately after an agent acts while some other environments may have sparse rewards, meaning rewards are infrequent. Correctly extract the rewards from observations on the environmental feedback is important to the learning performance. 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.

In other words, designing effective reward functions is crucial for successful reinforcement learning. Poorly designed rewards can lead to unintended behaviors or suboptimal performance. Here are some considerations when we design the reward mechanism:

1. Rewards should be aligned with the desired behavior or objective.
2. Rewards should be clear and unambiguous to avoid confusion for the agent about the goodness of its behaviors.
3. The appropriate level of reward frequency depends on the specific task. Dense rewards can be helpful or early learning, while sparse rewards can encourage exploration.
4. Intermediate rewards on agent's behaviors can be used to guide the agent towards the final goal.

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

- Board Game: points earned for wining the board game or reaching a predefined goal.

- Robotic Systems: positive rewards are rewarded to an agent for completing tasks like picking up objects or navigating obstacles.
- Web Applications: cash rewards can be granted when a particular webpage on review displays items in a favorable way that the average time users spend on the webpage increases significantly.
- Finance Trading Systems: rewards can be profits or losses from trading decisions.

Observation

Observation is the information about the environment, including that about the agents in the environment, that an agent receives from its environment or collects intentionally from the environment. They provide information about the current state of the environment and help the agent make decisions. 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.

Depending on the observability of the system and the agents, there are full state observations and partial state observations. For full state observations, the agent has complete information about the environment's state. In simple environments or simulations, the observations are often fully observable. For partial state observations, the agent only has access to a subset of the environment's state. This is more realistic for real-world scenarios where information is limited or noisy.

Observations are important to the decision making, learning and exploration of agents. Observations are used by the agent to select actions based on its current knowledge of the environment. In the meantime, observations are crucial for the agent to learn from its experiences and improve its behavior over time. Observations can also help the agent explore the environment and discover new opportunities.

Observations may or may not be relevant to the upcoming reward, e.g. about the environmental influence of a past action. They can be noisy or uncertain, making it difficult for the agent to accurately perceive the environment. They can be either in a determined format like numerical numbers or in an unstructured form such as an image of the agent or environment in a certain state. For the later, observations are usually high-dimensional, preprocessing on the observations are needed and can be challenging to extract useful information about the states related.

The dealing of observations is not straight-forward. The agent may need to learn a representation of the environment's state from the observations. This can be done using techniques like feature engineering or deep learning. If the agent has multiple sensors, it may need to combine the information from different sources to get a more complete picture of the environment through techniques of sensor fusion. Because observations are usually noisy and uncertain, the agent can incorpoerate uncertainty into its decision-making process to account for the potential errors in its observations.

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

States represent the current situation or condition of an agent within its environment. They are the basis for decision-making and learning in reinforcement learning. 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.

States can be represented in various ways including discrete states, continuous states and high-dimensional states. Discrete states can be represented as a finite set of possible values. Continuous states in reinforcement learning refer to states that can take on any value within a given range, rather than being limited to a discrete set of possibilities. This is in contrast to discrete states, which have a finite number of possible values. Continuous states can be represented as a real-valued vector. The advanatages of using continuous states include real-world complexity, richness of representation, and flexibility. High-dimensional states may have complex representations such as images or raw sensor data. States can be fully obsered or partially observed, even in a same environment.

They can be deterministic where next state is fully determined by the current state and action; or stochastic where there is some randomness in the state of transition.

States are important to decision making, learning, value estimation, and to defining the status of the agents as well as that of the environment. In reinforcement learning problems that are modeled as MDPs, the agent's actions are typically chosen based on the current state. States are used to associate actions with rewards, which may be used to update the agent's policy. The value of a state represents the expected future reward that can be obtained from that state.

Challenges on dealing with states include high-dimensional states, partial observability, and continuous states. Dealing with high-dimensional states can be computationally expensive and require efficient representation and learning techniques. When states are partially observed, the agent must infer the underlying state from limited information.

Representation and learning from continuous states can be more challenging than discrete states. To handle continuous state spaces, reinforcement learning agents often use function approximation techniques to generalize knowledge from a limited set of experiences. Common techniques include neural networks, linear function approximation, and kernel methods. Continuous state spaces can make exploration more challenging, as there are infinitely many possible states to explore. Techniques like ϵ-greedy exploration or Boltzmann exploration can be used to balance exploration and exploitation. Dealing with continuous state spaces can be computationally expensive, especially when using complex function approximators. Techniques like experience replay and prioritized experience replay can help to improve sample efficiency.

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. In other words, a policy is 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. The quality of a policy is typically evaluated based on the expected cumulative reward it generates. 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.

A policy can be deterministic or stochastic. A deterministic policy always selects the same action for a given state. A stochastic policy selects actions probabilistically, allowing for exploration and exploitation. Policies can be represented in various ways such as lookup tables, neural networks, and parameterized functions. Lookup tables are usually used for discrete states and actions. Neural networks are frequently used for complex state spaces. Parameterized functions are used for continuous state spaces.

One of the popular goals of reinforcement learning is to find an optimal policy that maximizes expected reward. This frequently involves iteratively improving the policy based on experience. Common techniques for policy optimization include policy gradient methods that directly optimize the policy parameters to maximize expected reward, Q-learning that learns a Q-value function that estimates the expected future reward for taking a particular action in a given state, and deep Q-networks that combine deep learning with Q-learning to handle complex state spaces.

Generally, there three types of policies: greedy policy, ϵ-greedy policy, and Boltzmann policy. A greedy policy always selects the action with the highest estimated value. An ϵ-greedy policy selects a random action with probability of ϵ, and selects the greedy action otherwise. A Boltzmann policy selects actions probabilistically based on their estimated values and a temperature parameter.

Challenges in policy optimization include exploration and exploitation tradeoff, credit assignment, and overfitting. Exploration and exploitation tradeoff deals with the problem of balancing the need to explore new actions to discover better policies with the desire to exploit known good actions. Credit assignment determines which actions and states contributed to the observed rewards. Overfitting the policy to the training data can lead to poor generalization, careful design of algorithms and learning processes to avoid overfitting is important. Adding regularization terms to the policy optimization objective can help prevent overfitting. For complex tasks, the policies can be complex and hard to learn. A task can be broken down into subtasks and using hierarchical policies to coordinate them. Transfer learning can also be used to leverage knowledge from previous tasks and subtasks to accelerate learning in the task.

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 that it is trying to optimize, given its policy or observed behavior. In other words, it's like reverse engineering the goal of an agent based on its actions. The promising of inverse reinforcement learning 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.

Similar to reinforcement learning, the key components of inverse reinforcement learning include agent, environment, policy, and reward function. The inverse reinforcement leraning process invovles observation, inference and evaluation. Observation collects data on the agent's behavior, including its actions and the resulting states. Inference uses machine learning algorithms to infer the reward function that best explains the observed behavior. Evaluation refers to evaluate the inferred reward function to ensure it accurately predicts the agent's behavior.

Challenges in inverse reinforcement learning include complexity of reward functions, multiple reward functions, noise and uncertainty. Reward function can be complex and difficult to represent. On the other hand, there may be multiple reward functions that could explain the observed behavior, making it difficult to identify the correct one. Furthermore, real-world data is often noisy and uncertain, which can make it challenging to accurately infer the reward function.

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.

Popular inverse reinforcement learning algorithms include maximum likelihood estimation (MLE), generative adversarial networks (GANs), and bayesian inverse reinforcement learning. MLE in inverse reinforcement learning finds the reward function that maximizes the likelihood of the observed behavior. GANs uses a generative model to generate samples of the agent's behavior and a discriminator to distinguish between real and generated behavior. Bayesian IRL uses a Bayesian approach to model the uncertainty in the reward function and infer a posterior distribution.

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 [44], [4].

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].

There are also numerous real-world applications of inverse reinforcement learning, including human behavior understanding that studies human preferences and intensions, robot imitation that teaches robots new skills by observing human demonstrations, game AI that creates challenging opponents in video games, and autonomous vehicles that learns driving behaviors from human drivers.

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

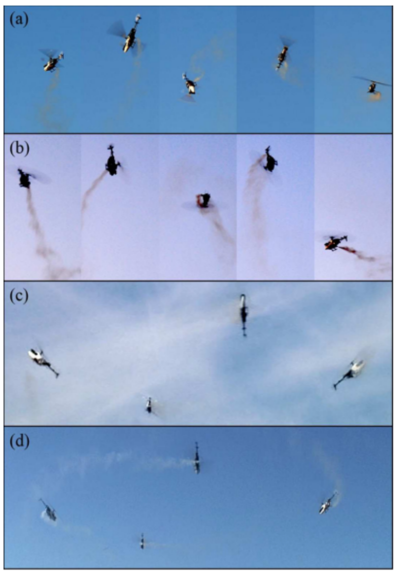


Fig. 1.2: Sample simulation of complex helicopter maneuver using a reward function learned from expert pilots through IRL. The image is copied from [2].

1.1.3 Meta Reinforcement Learning

Meta reinforcement learning is a learning paradigm that aims to equip agents with the ability to learn new tasks quickly and efficiently. Instead of focusing on a single task, meta reinforcement learning usually focuses on learning how to learn, allowing agents to adapt to new environments and challenges. Meta reinforcement learning targets at effective and efficient learning through meta learning. The key components in meta reinforcement learning can include meta learning that addresses the problems of learning to learn, task distribution that can be represented as a set of related tasks that the agent will encounter, meta learner which is a model that learns to learn tasks from the task distribution, inner loop which refers to the process of training an agent on a specific task, and outer loop which is the process of updating the meta learner based the performance of the inner loop agents.

This branch of reinforcement learning can be further divided into three categories: meta-parameter reinforcment learning, which deals with automatic meta learning of model hyperparameters; meta-data learning which improves data efficiency through meta learning on multiple tasks and is closely related to multi-task reinforcement learning; meta-task learning which learns to learn.

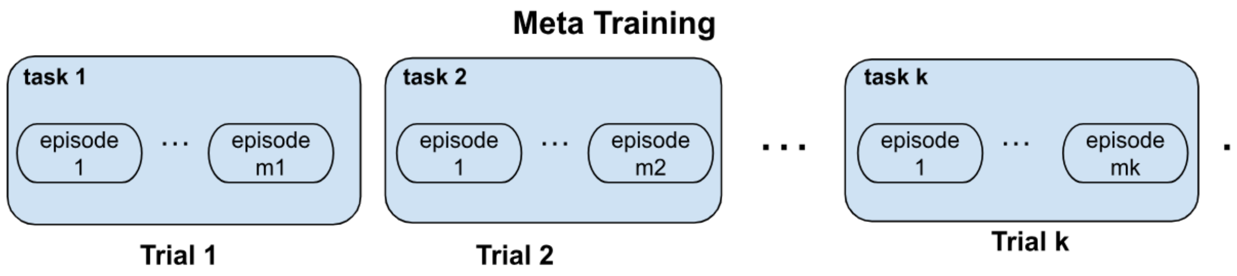


Fig. 1.3: A sample meta-training process of an meta reinforcement learning problem with k tasks. The learning process is carried out in N trials, with training trials on the k tasks repeatedly.

The correct settings of several meta-parameters are crucial to the success of reinforcement learning. Meta-parameter reinforcement learning addresses the problems of how to set and adjust these meta-parameters to enhance the learning performance [46]. 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 1.3, 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.

The second direction of MRL, meta-data reinforcement learning, focuses on data distribution learning, which utilizes cross environment samples to generalize the learning, we call it as meta-data reinforcement learning. The two main algorithm categories are few-shot meta reinforcement learning and many-shot meta reinforcement learning [7]. Fig. 1.3 shows a sample process of meta-data learning problem. Different from semi-supervised meta-data reinforcement learning, which usually depends on handcrafted task distributions, unsupervised meta-data reinforcement learning adaptively learns those distributions along the learning of meta-models. One main stream of meta-data reinforcement learning aims to learn exploration policies that solve challenging tasks with insufficient learning data like sparse or delayed rewards [18]. 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.

The third direction of MRL, meta-task reinforcement learning, focuses on how to learn from learning. It combines the power of multi-task learning and meta-learning and are usually unsupervised or semi-supervised, so that the meta-models are flexible enough to govern multiple different tasks which have a common base where meta-learning is effective exploration. This direction of meta reinforcement learning is largely unexplored and can reveal its power in the future of reinforcement learning study. It's worth to notice that meta reinforcment learning, especially meta-data reinforcement learning and meta-task reinforcement learning, may be referred to as multi-task reinforcement learning when it handles multi-task learning problems.

We leave the details of such algorithms in the sections of multi-task reinforcement learning.

Challenges in meta reinforcement learning include nosiy task distribution, learning overfitting and generalization, large computational cost, and credit assignment. We list these challenges in more details as below:

- Task Distribution: defining a stuitable task distribution can be challenging, due to incomplete information about the tasks to be solved such as limited availability of the tasks. Determining the similarity between tasks within the distribution is crucial for effective meta-learning, but it can be subjective and diffcult to quantify.
- Overffiting and Generalization: finding the right balance betwen exploring new tasks and exploiting known good strategies is essential for effective meta-learning. Meta learner can overfit to the training tasks, especially when some tasks are too similar to each other. This also limits its ability to generalize to new tasks.
- Computational Cost: Handling many tasks simultaneously, meta learner can also be computationally expensive, espeically for complex tasks with many-shot agent experiences. Meta reinforcement learning often involves training multiple agents on individual tasks within the inner loop, which can be computationally expensive.
- Credit assignment: determining which aspects of the meta-learner's training contributed to successful task adaptation can be difficult. In some cases, tasks may require hierarchical structures, which can introduce additional complexity and challenges in credit assignment.

Real-world applications of meta reinforcement learning face additional challenges: on one hand, obtaining sufficient data for a diverse set of tasks can be challenging in real-world applications. On the other hand, real-world environments are often noisy and uncertain, which can make it difficult for meta-learners to learn effective strategies.

Popular meta reinforcement learning algorithms include model-agnostic meta learning (MAML), learning to optimize (L2O), and meta reinforcement learning with memory. MAML updates the meta learner by finding initial parameters that allow agents to quickly adapt to new tasks. L2O learns an optimization algorithm that can be applied to different tasks. Meta reinforcement learning with memory incorporates memory into the meta learner to enable it to remember past experiences and adapt more effectively.

There are a wide range of applications of meta reinforcement learning in related machine learning fields. Few-shot learning utilizes meta reinforcement learning to learn new tasks with limited data. Transfer learning utilizes meta reinforcement learning to transfer knowledge from one task to another. Adaptive control utilizes it to adapt to changing environments and disturbances. Personalized recommendation systems use it to learn to personalize recommendations for individual users.

1.1.4 Hierarchical Reinforcement Learning

Reinforcement learning 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) is a framework that decomposes complex tasks into simpler subtasks, making it easier for agents to learn and solve them. By organizing tasks hierarchically, 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.

The key components of hierarchical reinforcement learning include high-level policy, low-level policy, decomposition, and coordination. High-level policies make decisions about abstract goals and subgoals. Low-level policies execute specific actions to achieve subgoals. Decomposition is the process of breaking down a complex task into smaller, more manageable subtasks. Coordination refers to the mechanism for cooerdinating the high-level and low-level policies.

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. Benefits of hierarchical reinforcement learning include scalability, efficiency, and reusability. It can handle complex tasks that would be difficult to solve with a single-level policy. By decomposing tasks into smaller subtasks, hierarchical reinforcement learning can reduce the search space and improve learning efficiency. Futhermore, subgoals and low-level policies can be reused across different tasks.

Fig. 1.4 shows an example diagram of an HRL problem.

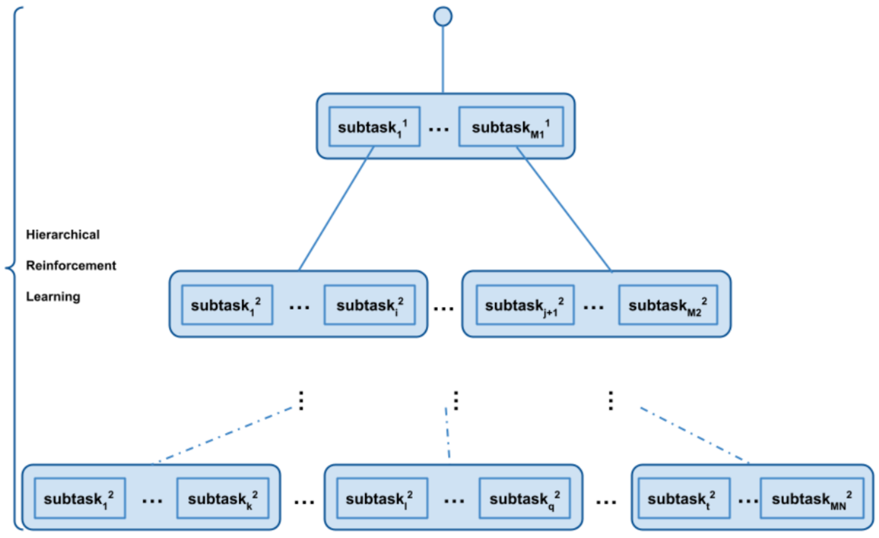

Hierarchical
Reinforcement
Learning
subtask$_1^1$ … subtask$_{M1}^1$
subtask$_1^2$ … subtask$_j^2$ … subtask$_{j+1}^2$ … subtask$_{M2}^2$
subtask$_1^2$ … subtask$_k^2$ … subtask$_l^2$ … subtask$_q^2$ … subtask$_t^2$ … subtask$_{MN}^2$

Fig. 1.4: 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.

The hierarchical structure of hierarchical reinforcement learning 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 [41] applies HAM constraints on the state exploration and thus reduces the invalid search. In [52], 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.

Hierarchical reinforcement learning faces several challenges, including credit assignment problem, exploration-exploitation trade-off, difficulties in subgoal definition, proper coordination between levels, scalability problems, weak transfer learning, and uncertainty handling. We list these challenges in more details as below:

- Credit Assignment Problem: the problem is two-folds: on one side, complex interactions between hierarchical levels make it hard to isolate the effects of individual actions. On the other side, difficulty in determining which actions at different levels contribute to the final reward.
- Exploration-exploitation Trade-off: similar to other reinforcement learning branches, balancing exploration of new actions and exploitation of known good ones is crucial. Hierarchical structures can exacerbate this problem due to the increased complexity of the search space.
- Subgoal definition: it is essential for effective hierarchical reinforcement learning. Subgoals must be relevant to the overall goal and achievable within the given context.
- Coordination between Levels: it is critical to ensure smooth coordination between hierarchical levels. Misalignment or conflicts between levels can lead to suboptimal performance.
- Scalability: hierarchical reinforcement learning can become computationally expensive as the number of hierarchical levels and actions increases. Efficient algorithms and representations are necessary to scale hierarchical reinforcement learning to complex tasks.

- Transfer Learning: tansferring knowlege from one task to another can be challenging in hierarhical reinforcement learning. The hierarchical structure and specific subgoals may not be directly applicable to new tasks.
- Uncertainty Handling: dealing with uncertainty in the environment is crucial for hierarchical reinforcement learning. Robustness to noise and unforeseen events is essential for reliable performance.

Popular hierarchical reinforcement learning approaches include Options, Subgoal Decomposition, and Hierarchical Q-learning. An option is a temporally extended action that can be executed over multiple time steps. The high-level policy selects options, and the low-level policies execute them. Subgoal Decomposition decomposes the task into subgoals, and the high-level policy plans a sequence of subgoals. Hierarchical Q-learning is a variant of Q-learning that learns Q-values for both high-level and low-level actions.

1.1.5 Multi-Task Reinforcement Learning

Multi-task reinforcement learning is a machine learning paradigm where a single agent learns to perform multiple tasks simultaneously or sequentially. 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 [60]. Multi-task reinforcement learning solves a more general problem of efficient learning on multiple tasks simultaneously or under the same learning framework. It is closely related to multi-task learning in the general machine learning domain. For multi-task reinforcement learning 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. Multi-task reinforcement learning aims to improve generalization, efficiency, and sample complexity by leveraging knowledge gained from one task to aid learning others.

Key benefits of multi-task reinforcement learning include improved generalization, increased efficiency, enhanced sample complexity, and enhanced transfer learning. We list these benefits in more details as below:

- Improved Generalization: by training on a diverse set of tasks, the agent can learn more generalizable representations and strategies, making it more adaptable to new and unseen tasks.
- Faster Learning: multi-task reinforcement learning can accelerate learning by leveraging knowledge acquired from related tasks. This can be particularly useful when data for individual tasks is limited.
- Efficient Resource Utilization: multi-task reinforcement learning can also improve sample efficiency by allowing agents to learn from a wider variety of experiences. Particularly, sharing parameters and experiences across multiple tasks can reduce the overall computational cost and data requirements.
- Enhanced Transfer Learning: multi-task reinforcement learning can facilitate the transfer of knowledge between related tasks, enabling agents to learn new skills more quickly.

Besides challenges that are common to reinforcement learning problems such as computational overhead, task-specific features, exploration-exploitation trade-off. Multi-task reinforcement learning faces its own challenges with respect to task interference, catastropic forgetting, task prioritization, and transferability. We list these challenges in more details as below:

- Task Interference: learning multiple tasks simultaneously can lead to interference, where knowledge from one task hinders learning in another. In other words, when tasks are conflicting or share similar states, the agent may struggle to balance their objectives, leading to suboptimal performance.
- Catastrophic Forgetting: As the agent learns new tasks, it may forget knowledge from previously learned tasks, especially when the tasks are dissimilar.

- Task Prioritization: determining the appropriate balance between learning different tasks can be challenging.
- Transferability: ensuring that knowledge learned in one task is relevant and transferable to others is crucial.
- Reward Sparsity: in many real-world scenarios, rewards are sparse, making it difficult for the agent to learn effective policies for multiple tasks simultaneously.

Multiple approaches and techniques are frequently used in reinforcement learning: transfer learning-based approaches, representation learning of shared part of value functions, deep neural network-based approaches, and their combinations. In special cases, tasks can be grouped into sub-groups hierarchically, and the learning problem is solved similarly to those in HRL, where subtasks originated from the same subtask at the higher level share or partially share the same set of learning parameters. Works have been conducted on integrating deep presentation learning into MRL to learn the shared value and policy parameters among tasks efficiently. A multi-task neural network is developed for dynamic malware classification in [23]. Actor-Mimic (AM) [40] is another multi-task reinforcement learning approach to training intelligent agents who are capable of acting in multiple environments and transferring the knowledge learned from past experiences to new situations. These agents can conduct multi-tasks simultaneously and generalize their accumulated knowledge to new domains.

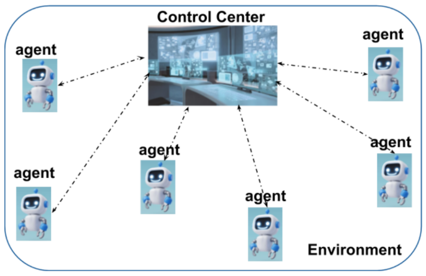
More recently, a more significant part of research efforts conducted within the area of multi-task reinforcement learning is about the integration of neural network techniques into area. The representative works include Policy Distillation (PD) [59], DISTRAL [54], IMPALA [15] and PopArt [20]. Policy Distillation [59] leverages the concept of distillation in transfer and apprenticeship learning towards multi-task deep reinforcement learning. DISTRAL [54] was designed as a framework for simultaneous learning of multiple tasks, with the focus on building a scheme for distilling the centroid policy of common behaviors among agents to individual learners in multi-task reinforcement learning. Instead of direct parameter sharing among agents of the multiple tasks, the key idea of DISTRAL was to learn a shared policy that captures common behavior across tasks which may be governed by heterogeneous policies through distilling common experiences from task-specific policies. Then the distilled policy is regularized to develop specialized policies to handle individual tasks more effectively. IMPALA [15] was designed for efficient single-agent multi-task reinforcement learning to handle the increased amount of data and training time required, with the ability to scale to multiple machines without sacrificing data efficiency and resource utilization. PopArt [20] was designed, based on IMPALA model architecture of multiple convolutional neural network layers, with the integration of other neural network techniques such as word embedding through recurrent neural network of long-short term memory (LSTM) type. The main objective of PopArt is to minimize distraction and stabilize learning.

Multi-task learning is increasingly relevant in real-world scenarios where agents need to be capable of handling a variety of tasks, such as self-driving cars or robots that perform multiple functions.

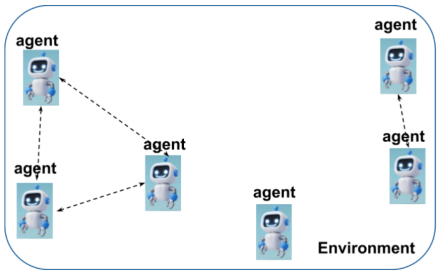
1.1.6 Multi-agent Reinforcement Learning

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.

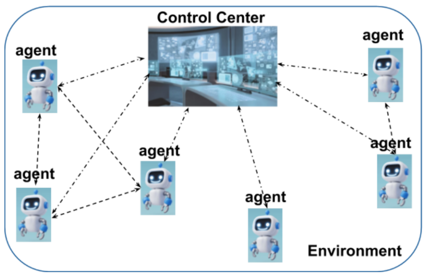
The learning convergence of the former results in a win-win situation where all agents are beneficiaries. The learning convergence of the latter results in a win-lose situation where only one agent perceives the outcome as positive while the learning divergence may lead to a no-win situation. And usually 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 multi-agent reinforcement learning can be found in [11]



(a) A sample multi-agent reinforcement learning 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 multi-agent reinforcement learning 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 multi-agent reinforcement learning 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. 1.5: Sample multi-agent reinforcement learning systems with three different agent-agent interaction structures.

In a multi-agent reinforcement learning problem, the agent-agent communications can be centralized, distributed, or hybrid as demonstrated in Figure 1.5.

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

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

The computational complexity of multi-agent reinforcement learning 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 multi-agent reinforcement learning, 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 multi-agent reinforcement learning, 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.

1.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a subfield of reinforcement learning that combines deep learning techniques with reinforcement learning algorithms. It allows agents to learn complex tasks in high-dimensional state and action spaces. In other words, 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. The challenges in deep reinforcement learning include those challenges in traditional reinforcement learning and the ones in deep learning and new ones in the integration of the two areas of techniques. Figure 1.6 shows a training architecture with both policy and value networks.

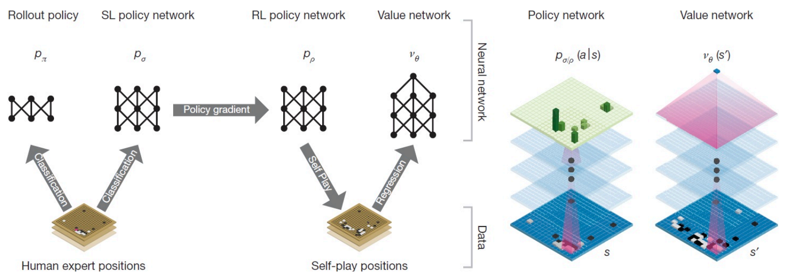


Fig. 1.6: A sample training architecture which learns policy and value function using deep neural networks. The picture is originated from [47].

The key components of deep reinforcement learning include deep neural networks, reinforcement learning algorithms, and experience replay. Deep neural networks are used to represent the state space, action space, and Q-values or policies. Popular reinforcement learning algorithms include Q-learning, policy gradients, or actor-critic methods. Deep Q-Networks (DQN) is a classic DRL algorithm that uses a deep neural network to approximate the Q-value function. Policy Gradients is a family of algorithms that directly optimize the policy function using gradient methods. Actor-Critic Methods combine a policy function (actor) with a value function (critic) to improve learning. Asynchronous Advantage Actor-Critic (A3C) is a parallel implementation of actor-critic methods that can handle complex environments. Experience replay is a technique that stores experiences in a buffer and samples them randomly to train the neural network, improving statbility and efficiency.

Deep Q-networks (DQNs) and their variations and extensions including N-step DQN, Double DQN, Dueling DQN, and Categorical DQN [26] 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 [36] 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 [14].

Physical systems that are backed with reinforcement learning modules and chips can range in size from a small drone [1] 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 [21] 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 [12] 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 reinforcement learning systems include both reinforcement leran-ing 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.

Below we list the popular real-world applications of reinforcement learning:

- Robotics and Automation
  - Robot Manipulation: RL algorithms can train robots to perform complex tasks like grasping, assembly, and manipulation.
  - Autonomous Vehicles: Self-driving cars can use RL to learn optimal driving strategies, considering factors like traffic, weather, and road conditions. Industrial Automation: RL can optimize processes in factories, warehouses, and other industrial settings, improving efficiency and productivity.
- Game Playing
  - Video Games: RL agents have achieved superhuman performance in games like Go, StarCraft II, and Dota 2.
  - Board Games: RL can be used to develop strategies for games like chess, poker, and backgammon.
- Healthcare
  - Drug Discovery: RL can accelerate the process of drug discovery by optimizing molecular structures for desired properties.
  - Personalized Medicine: RL can be used to develop personalized treatment plans based on individual patient data.
- Finance

  - Algorithmic Trading: RL algorithms can be used to make automated trading decisions based on market data.
  - Risk Management: RL can help in optimizing investment portfolios and managing risk.

- Natural Language Processing
  - Machine Translation: RL can improve the quality of machine translation systems by learning from large datasets of parallel text.
  - Dialogue Systems: RL can be used to train chatbots and virtual assistants to engage in more natural and informative conversations.
- Energy Management
  - Smart Grids: RL can optimize energy distribution and consumption in smart grids, reducing costs and improving efficiency.
  - Renewable Energy Integration: RL can help integrate renewable energy sources like solar and wind power into the grid.
- Other Applications
  - Recommendation Systems: RL can personalize recommendations for products, movies, music, and other content.
  - Supply Chain Optimization: RL can optimize supply chain operations, improving inventory management and delivery efficiency.

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

[2] 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.

[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] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13:341–379, 2003.

[6] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[7] 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.

[8] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[9] Léon Bottou. Online algorithms and stochastic approximations. *Online learning in neural networks*, 1998.

[10] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[11] 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.

[12] 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.

[13] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[14] 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.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[16] 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.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[18] 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.

[19] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

[20] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

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

[22] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.

[23] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 399–418. Springer, 2016.

[24] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *ICLR*, 2019.

[25] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[26] 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.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Frank L Lewis and Kyriakos G Vamvoudakis. Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using measured output data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):14–25, 2010.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[30] Robert Loftin, Aadirupa Saha, Sam Devlin, and Katja Hofmann. Strategically efficient exploration in competitive multi-agent reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 1587–1596. PMLR, 2021.

[31] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

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

[33] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] 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.

[37] 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.

[38] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, volume 1, page 2, 2000.

[39] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[40] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

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

[42] Pascal Poupart, Aarti Malhotra, Pei Pei, Kee-Eung Kim, Bongseok Goh, and Michael Bowling. Approximate linear programming for constrained partially observable markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[43] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine learning: ECML 2005: 16th European conference on machine learning, Porto, Portugal, October 3-7, 2005. proceedings 16*, pages 317–328. Springer, 2005.

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

[45] Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

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

[47] 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.

[48] 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.

[49] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

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

[51] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[52] 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.

[53] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[54] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[55] 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.

[56] 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.

[57] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30-1, 2016.

[58] Wikipedia contributors. Stochastic gradient descent — Wikipedia, the free encyclopedia, 2024. [Online].

[59] Haiyan Yin and Sinno Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[60] 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.

[61] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[62] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.