IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 11, 2017, accepted November 21, 2017, date of publication November 27, 2017,
date of current version December 22, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2777827

System Design Perspective for Human-Level Agents Using Deep Reinforcement Learning: A Survey

NGOC DUY NGUYEN, THANH NGUYEN, AND SAEID NAHAVANDI, (Senior Member, IEEE)
Institute for Intelligent Systems Research and Innovation, Deakin University, Waurn Ponds Campus, Geelong, VIC 3216, Australia
Corresponding author: Ngoc Duy Nguyen (duy.nguyen@deakin.edu.au)

ABSTRACT Reinforcement learning (RL) has distinguished itself as a prominent learning method to augment the efficacy of autonomous systems. Recent advances in deep learning studies have complemented existing RL methods and led to a crucial breakthrough in the effort of applying RL to automation and robotics. Artificial agents based on deep RL can take selective and intelligent actions comparable with those of a human to maximize the feedback reward from the interactive environment. In this paper, we survey recent developments in the literature regarding deep RL methods for building human-level agents. As a result, prominent studies that involve modeling every aspect of a human-level agent will be examined. We also provide an overview of constructing a framework for prospective autonomous systems. Moreover, various toolkits and frameworks are suggested to facilitate the development of deep RL methods. Finally, we open a discussion that potentially raises a range of future research directions in deep RL.

INDEX TERMS Deep learning, human-level agents, reinforcement learning, robotics, survey, system design.

I. INTRODUCTION

By mimicking human behaviors, researchers have adopted a learning method that has a high impact on foundations of artificial intelligence study. This approach, called *reinforcement learning* (RL), focuses on examining actions that gain a maximal value of long-term reward from the environment [1]. Additionally, RL utilizes a trial-and-error learning process to achieve its goals. This unique feature has been confirmed to be an advanced approach to building a human-level agent [2]. For instance, in 1992, Mahadevan and Connell built a robot based on RL named OBELIX that learned how to push boxes [3]. In 1996, the Sarcos humanoid DB was constructed by Schaal to learn the pole-balancing task [4]. Lin *et al.* [5] proposed an RL method to control dynamic walking of a robot without prior knowledge of the environment. Recently, Müelling *et al.* [6] employed RL to train a robot to play table tennis and Riedmiller *et al.* [7] applied a batch RL to prepare crucial skills for a soccer-playing robot.

Fig. 1 illustrates a traditional RL problem, a pole-balancing task. The goal of this task is to exert a reasonable amount of force to the cart along the track so that it balances the pole from falling. Each *state* of the system involves dynamics information such as the position, velocity of the cart as well as angle and angular velocity of the pole. For every micro time-step t, there are two possible *actions* impacting the cart: moving to the right and moving to the left. At the same time, we may give -1 as a *reward* if the pole falls and 0 in the other cases.

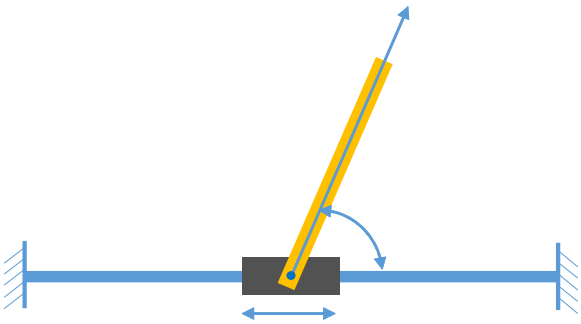

FIGURE 1. A pole-balancing task based on RL.

Previous applications of RL can be categorized into three independent research fields until the reconciliation to modern RL in the late 1980s [1]. The first research area was rooted in the psychology and neuroscience of animal learning by conducting a series of trial-and-error experiences [8], [9]. The second study focused on the optimal control problem.

Richard Bellman proposed a method to solve this issue by introducing the *Bellman equation* and the discrete model of optimal control problem, named the *Markovian Decision Processes* (MDPs) [10]. At that time, *dynamic programming* identified itself as the only way to solve the *curse of dimensionality*, where computations increase dramatically with the number of variables. Last but not least, the third study concerns *temporal-difference* learning, which originated the well-known *Q-learning* method. This method has a great impact on establishing a standard approach towards RL. In summary, all of these concepts together contributed to building the fundamental elements of contemporary RL. Although RL has become a common method for building an autonomous agent, the shortcomings of traditional learning approaches restrain it from dealing with complex problems. Recent integrations of deep learning methods with RL have improved the performance of existing RL methods considerably.

Deep learning is a subset of machine learning that effectively employs neural networks to learn on multiple levels, each corresponding with different levels of abstraction [11]. The recent success of various applications shows that deep learning outperforms traditional approaches in terms of accuracy and efficiency. This feat involves the first-time usage of deep learning and *Convolutional Neural Network* (CNN) in the ImageNet competition that ultimately beat the record of traditional approaches in computer vision [12], [13]. Lately, deep learning has been applied widely to various research fields including image recognition, video classification, audiograms, language processing, video games, and robotics [14]–[18].

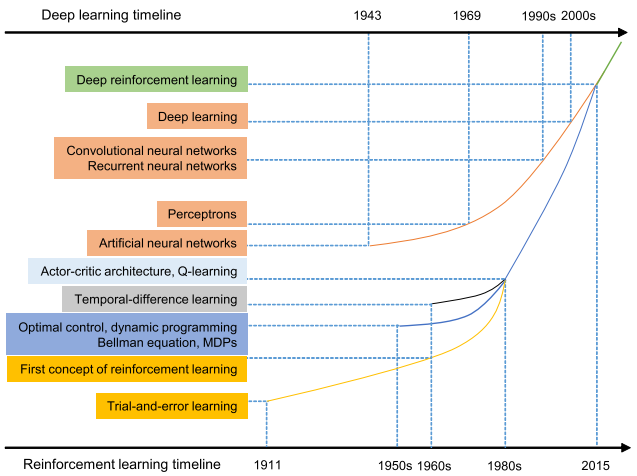

FIGURE 2. Milestones of deep reinforcement learning.

Fig. 2 summarizes the milestones of deep RL. Before the advent of deep learning, Monte Carlo and Q-learning were two fundamental methods used in RL [19]. However, these methods restrict RL to solving complex problems due to the inherent constraints of computer memory space. By addressing an approximate approach, deep learning solves these intractable problems effectively. In fact, Mnih *et al.* incorporated deep learning with RL to create a human-level agent that is virtually unbeatable in a series of 49 video Atari games [20], [21]. To extend the success of deep RL, Google's DeepMind subsidiary created AlphaGo, a program that beat one of the best professional Go players of all time, South Korea's Lee Sadol, in 2016 [22]. At the time of this paper, AlphaGo even leads 2-0 against the highest-ranking Go player, Ke Jie from China. Additionally, Google, Uber, and Tesla are hastening the research on deep RL to design the next generation of intelligent self-driving cars.

In summary, this research focuses on examining deep RL approaches that have a significant impact on building distinct aspects of human-level agents. Our work brings up the following key contributions. First, we provide an overview of state-of-the-art deep RL achievements in recent years. Under the system design perspective, we present not only the general of RL methodology but also a concise explanation of modern deep RL and its applications. Second, we separate contemporary deep RL studies into different categories that affect various aspects of modeling a human-level agent. The latest toolkits and framework libraries that can be used to develop deep RL approaches are introduced. Third, we design a high-level system architecture to describe the usage of recent breakthroughs in constructing a human-level agent. Finally, we open a discussion related to deep RL and then inherently raise different directions of future studies.

II. PRELIMINARY

Interactive learning attracts considerable attention because of its analogy to human learning behaviors. As a result, interactive RL establishes a comprehensive learning framework for the advance of human-level agents and autonomous systems. Before examining key breakthroughs of contemporary RL, we review fundamental concepts of RL as well as the related learning schemes. Specifically, subsection II-A summarizes the concept of RL and its target problem domain, i.e., MDP. Subsection II-B describes the Bellman equation, which is the core of all derivations in RL. Finally, subsection II-C and subsection II-D outline two fundamental learning methods in RL.

A. REINFORCEMENT LEARNING AND MARKOV DECISION PROCESS

RL is a tabula rasa learning, interacting with the stochastic environment to earn the best long-term reward, i.e., RL seeks an optimal policy π^* that is a mapping function Γ_{π^*} from each possible state s of the agent to its selective action a so as to maximize accrued long-term reward r. Intuitively, Γ_π is a set of probabilities from every transition s to s' by following a under the policy π as below:

$$\Gamma_\pi = \{\Pr(s \xmapsto{a} s' \mid \pi) : \forall a \in \Lambda_\pi(s)\}, \tag{1}$$

where $\Lambda_\pi(s)$ is the action space of s under the policy π.

The agent often interacts with the environment in a discrete time-step manner. For example, in each time-step t

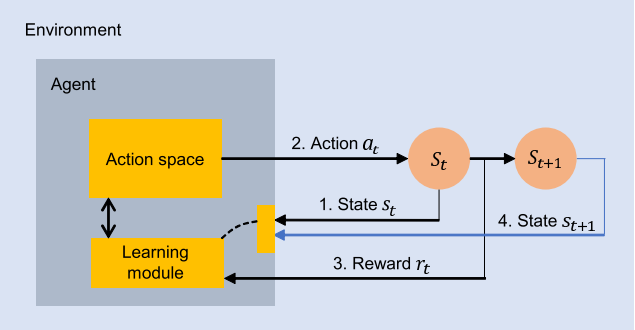

FIGURE 3. The relationship between the agent and the environment in RL.

$(t = 0, 1, 2, \ldots)$, the agent observes a state s_t of the environment and selects an action a_t from its action space $\Lambda(s_t)$. The agent then obtains the reward r_{t+1} from the environment. Finally, the current state of the environment turns to s_{t+1}. As a result, RL produces the sequence of states, actions, and rewards, respectively: $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, \ldots$. If the number of time steps is finite, like a play of a game, we call the RL problem an *episodic* task. Fig. 3 summarizes the relationships between these measures. Mathematically, the goal of RL is to maximize the *discounted return* $\Re_t$ at each time-step t, as described in [1]:

$$\Re_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \tag{2}$$

where γ denotes the *discounted rate* and $0 \leq \gamma \leq 1$.

A state signal has a *Markov property* if every next state s_{t+1} and next reward r_{t+1} depends only on the current state s_t and its accompanying action a_t, regardless of the history. An RL problem is an MDP if its state signal has the Markov property. For example, the pole-balancing task introduced in section I is an MDP because the state of the task, including position and velocity of the cart as well as the angle and angular velocity of the pole, is sufficient to foretell the future operations of the system. In summary, MDP is the crucial problem domain in RL.

B. THE BELLMAN EQUATION

To describe the RL problem in a mathematical way, we review the concept of *value function*. Value function is a function of state that is used to evaluate a state or a specific action from a state. From (2), we have the *state-value function* under the policy π as follows:

$$V_\pi(s) = \mathbf{E} \bullet \left\{ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s, \pi \right\}, \tag{3}$$

where $\mathbf{E}\bullet$ terms an expected function. Similar to (3), the *action-value function* expresses the expected return from a state s by following an action a under the policy π as below:

$$Q_\pi(s, a) = \mathbf{E} \bullet \left\{ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s, a_t = a, \pi \right\}. \tag{4}$$

Based on (1), (3), and (4), we can infer the relationship between the current state s with its next state s' as described in the following equations:

$$V_\pi(s) = \sum_{a,s'} \Pr(s \xrightarrow{a} s' \mid \pi) \left(\overline{R}^a_{ss'} + \gamma V_\pi(s') \right), \tag{5}$$

$$Q_\pi(s, a) = \sum_{s'} \Pr(s \xrightarrow{a} s' \mid a, \pi) \left(\overline{R}^a_{ss'} + \gamma V_\pi(s') \right), \tag{6}$$

where $\overline{R}^a_{ss'}$ indicates the expected reward from s to s' by following a. Equations (5) and (6) denote Bellman equations for V_π and Q_π, respectively. Therefore, the RL problem becomes finding an optimal policy π^* such that:

$$\begin{cases} V_{\pi^*}(s) = \max\limits_{\pi} V_\pi(s) \\ Q_{\pi^*}(s, a) = \max\limits_{\pi} Q_\pi(s, a), \end{cases}$$

for every state s and action a. Therefore, the Bellman equations for V and Q under the optimal policy π^* are rewritten as below [1]:

$$V^*(s) = \max_a \sum_{s'} \Pr(s \xrightarrow{a} s') \left(\overline{R}^a_{ss'} + \gamma V^*(s') \right),$$

$$Q^*(s, a) = \sum_{s'} \Pr(s \xrightarrow{a} s') \left(\overline{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right) \tag{7}$$

In summary, Bellman equation is the core component of derivative learning methods in RL. In the two subsequent subsections, we review two fundamental learning methods in RL: *Monte Carlo* and *Q-learning*.

C. MONTE CARLO METHOD

Monte Carlo (MC) method is learning by rolling out *experiences* (samples) of the task and averaging the returns from these samples to approximate the value functions. Asymptotically, these average values converge to the value function and therefore can determine the optimal policy for the task. For instance, to approximate the value of a state s under the policy π, $V_\pi(s)$, we roll out a series of episodes passing through s: $e_1, e_2, e_3, \ldots$. This action yields a series of corresponding return values from s: $r_1, r_2, r_3, \ldots$. By averaging all the return values, we can approximate the state-value function $V_\pi(s)$. The benefit of using MC learning is that it does not require a complete knowledge of the environment's dynamics and therefore can be used in both *on-line* and *off-line* learning efficiently. On-line learning is learning by interacting directly with the environment whereas off-line learning is learning in the simulated environment. Similarly, we can use the MC method to estimate the action-value function $Q_\pi(s, a)$.

To find the optimal policy π^*, it is necessary to process through a *policy iteration* that involves two interleaved processes: *policy evaluation* (PE) and *policy improvement* (PI).

PE is the process of estimating the value function, while PI is the process of seeking the optimal policy. For instance, given a policy π_0, we use the MC method to estimate Q_{π_0}. We then find a policy π_1, which yields Q_{π_1} so that Q_{π_1} is better than Q_{π_0} ($Q_{\pi_1} \geq Q_{\pi_0}$). This process is iterative until we find the optimal solution π^* and Q^*:

$$(\pi_0 \asymp Q_{\pi_0}) \rightarrow (\pi_1 \asymp Q_{\pi_1}) \rightarrow \ldots \rightarrow (\pi^* \asymp Q^*).$$

One of the naive approaches to achieving the PI is the usage of *exploration*. To yield a better policy π' from π, we select an arbitrary state s in π and "explore" an action a' from s so that $a' \notin \Lambda_\pi(s)$, we then estimate $Q(s, a')$ and compare it to $Q(s, a)$. If $Q(s, a') \geq Q(s, a)$, we form the new policy π' by adding a' into π. Conceptually, the process of integrating the exploration in the learning policy is called *on-policy* control. One of the common on-policy approaches used in PI is ϵ-*greedy*. The probability of selecting an action a from a state s in ϵ-greedy is calculated using the following formula:

$$\Pr(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|\Lambda(s)|}, & \text{if } a = \arg\max\limits_{a'} Q(s, a') \\ \dfrac{\epsilon}{|\Lambda(s)|}, & \text{otherwise} \end{cases} \quad (8)$$

where $|\Lambda(s)|$ denotes number of possible actions from s and $0 < \epsilon \leq 1$.

As opposed to on-policy control, *off-policy* control separates the exploration from the learning policy by using a different policy, called *behavior policy*. Behavior policy π' is used to generate samples and explore the action space while the learning policy π is free to select a greedy action in a deterministic manner. However, to ensure the convergence of the algorithm, π' is selected so that every action taken in π must occur at least once in π'. Off-policy MC method inspired the well-known artificial computer program, AlphaGo, which shall be discussed in section III. Fig. 4 summarizes the differences between on-policy control and off-policy control in the MC method.

D. Q-LEARNING

Before discussing the concept of Q-learning, we briefly summarize the broader set of learning, *temporal-difference* (TD) learning. Like MC method, TD learning does not require a complete knowledge of the environment but rather derives its information from experiences. However, while MC needs to wait to finish the episode before updating, TD waits only until the next time-step $t+1$ to update the estimated value function of current time t. The following equation illustrates an update rule for the simplest case of TD (originated from (5)):

$$V(s_t) \odot \alpha V(s_t) + \beta[r_{t+1} + \gamma V(s_{t+1})], \quad (9)$$

where α and β are parameters such that $0 \leq \alpha, \beta \leq 1, \beta \neq 0$, and $\alpha + \beta = 1$; $\odot$ denotes the update rule for $V(s_t)$. The update rule (9) is used to approximate the value function $V(s)$.

Similar to the MC method, two control approaches are used to obtain the optimal policy: on-policy TD control known as *Sarsa* and off-policy TD control known as Q-learning.

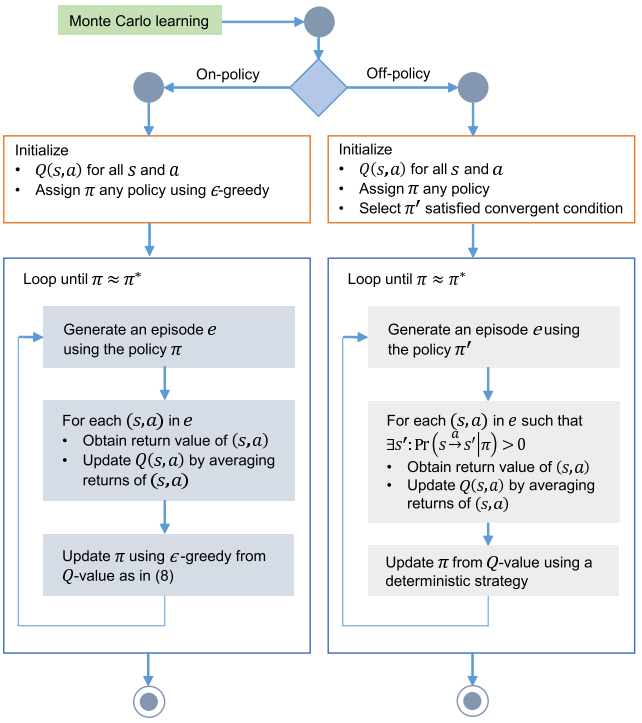

FIGURE 4. An example of using on-policy versus off-policy control in MC learning.

In Sarsa, instead of estimating the state-value function as in (9), we estimate the action-value function by using the following rule:

$$Q(s_t, a_t) \odot \alpha Q(s_t, a_t) + \beta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})]. \quad (10)$$

Sarsa uses (10) to estimate the Q-value function (PE) and uses ϵ-greedy to improve the policy (PI).

As opposed to Sarsa, Q-learning uses the optimal Bellman equation (7) to approximate Q^* directly. This approach dramatically shortens the convergence time of the algorithm to find the optimal solution. In summary, Q-learning leverages the following update rule to approximate Q^*:

$$Q(s_t, a_t) \odot \alpha Q(s_t, a_t) + \beta[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)]. \quad (11)$$

Fig. 5 illustrates the differences between Sarsa and Q-learning in the TD learning method.

The last subset method of the TD approaches reviewed in this section is the *actor-critic* (AC) method [23]. AC separates the policy π from the value function by constructing two independent memory structures. One structure (*actor*) selects actions from π and feeds them into another structure (*critic*) for evaluation. The critic uses the following error measurement formula to decide the frequency of using an action a_t:

$$\begin{cases} \Delta_t = \delta[r_{t+1} + \gamma V(s_{t+1})] - \delta' V(s_t) \\ \Delta_t > 0 \longrightarrow [Pr(s_t \overset{a_t}{\longmapsto} s_{t+1}) \oplus |\Delta_t|], \\ \Delta_t < 0 \longrightarrow [Pr(s_t \overset{a_t}{\longmapsto} s_{t+1}) \ominus |\Delta_t|] \end{cases}$$

where δ and δ' are adjustment parameters; $a \oplus b$ denotes adding to a a quantity that is proportional to b; and $a \ominus b$

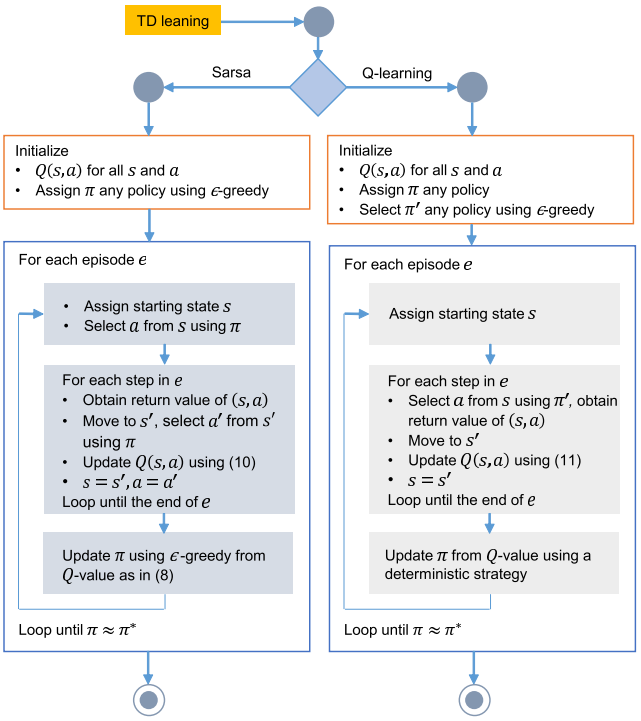

FIGURE 5. An example of using Sarsa versus Q-learning in TD learning.

denotes subtracting from a a quantity that is proportional to b. The benefit of using AC structure is the ability to deal with continuous problem domains that allow to learn a task in an asynchronous manner. In summary, the concept of RL and its core element, Bellman equation, have been reviewed as well as two fundamental learning approaches in RL. In the next section, a survey of recent breakthroughs will be carried out, which describe how to combine deep learning with traditional RL methods such as MC or Q-learning to create an agent with human-level skills.

III. LITERATURE REVIEW

Although RL has a significant impact on building a general framework for autonomous systems, traditional RL methods still have shortcomings that need to be mitigated [20], [24].

First, RL methods only work efficiently on discrete and finite MDPs where the number of states is limited, i.e., it is infeasible to implement RL in practical systems that deal with a mix of complicated tasks and chaotic environments. These practical problems are usually non-Markov, continuous, and have an unbounded action space. Second, RL requires a comprehensive knowledge of the observed environment in order to analyze the reward signal function. This collected information directly concerns the learning process of the RL regarding agility and efficiency. Finally, RL is known to be unstable when using a nonlinear approximator to estimate the value functions. One of the main reasons for this instability is the correlation between the updates of the value function using the Bellman equation. As a result, a minor update of the value function may enormously impact the output policy and thus make the RL problem divergent.

A. DEEP REINFORCEMENT LEARNING

In 2015, Mnih *et al.* [20] from Google's Deepmind advanced modern RL by introducing the concept of *Deep Q-Network* (DQN). DQN connects the dots between deep neural networks with RL by subduing the intractable problems in traditional RL methods. Particularly, DQN leverages CNNs to analyze input images and use these CNNs to approximate Q-value function. In other words, the goal of DQN is to minimize the loss function of a CNN as below:

$$L(\theta) = \mathbf{E} \bullet \left\{ \left(\overbrace{r + \gamma \max_{a'} Q(s', a'|\theta')}^{target} - \underbrace{Q(s, a|\theta)}_{output} \right)^2 \right\} \quad (12)$$

where θ and θ' represent the parameters of the current estimation Q-network and the target network, respectively. The target network is used to estimate the Q-value in the next state to reduce the correlations between Q-value updates and thus make the Q-network output's distribution stationary. To make the Q-learning further stable, the authors introduce the concept of *experience replay*, i.e., all experiences in the form $e = (s, a, r, s')$ are stored in the memory and sampled uniformly in a random manner. The loss function (12) has a similar form to the Q-learning update rule (10) and thus can be used to estimate directly Q^* as well as the optimal policy π^*. As a result, DQN creates a human-level agent that outperforms the best RL method so far in the test series of 49 classic Atari games [21], [25]. Fig. 6 summarizes the system architecture used in DQN.

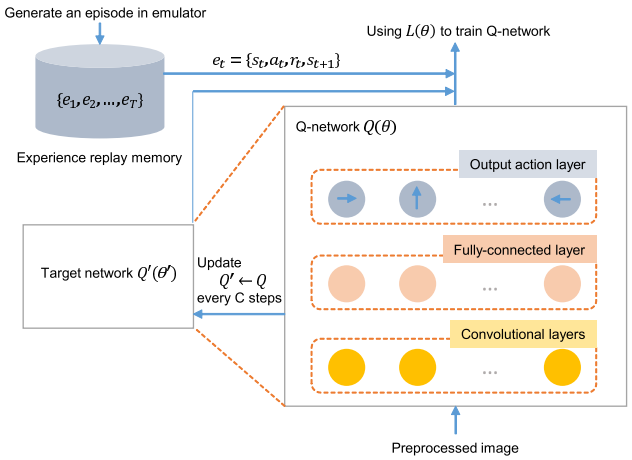

FIGURE 6. The system architecture used in DQN.

The second feat of deep RL is noteworthy by the historic triumph of AlphaGo versus the best professional Go player, China's Ke Jie, in May 2017. Prior to AlphaGo's victory, Deep Blue developed by IBM was the only computer program to defeat Garry Kasparov in the World Chess Champion in 1997. The underlying algorithm behind Deep Blue is based on a traditional search tree and a robust evaluation function [26]. However, Go is more complex than Chess because Go's search tree space is multiple times that of Chess. Therefore,

traversing all possible cases of Go's search tree to find the best play is infeasible. The mechanism underlying AlphaGo's success was initially a trade secret.

In 2016, Google Deepmind decided to reveal the secret recipe behind AlphaGo [22]. AlphaGo employs the *Monte Carlo Tree Search* algorithm (MCTS) [27]–[29] and deep learning with a CNN to estimate the best move. MCTS simulates an immense number of episodes to estimate the optimal value of each node in the game search tree. The training process of AlphaGo involves two separate policy networks p_π, p_δ and a value network v_θ. At first, the MC method generates the fast policy p_π. Then, the second network p_δ is trained under the supervision of professional human players. Afterward, p_δ plays itself to improve the performance and is used to train v_θ. Finally, the entire algorithm is based on a combination of MCTSs, with the value function of the leaf node s_l being as follows [22]:

$$V(s_l) = \lambda v_\theta(s_l) + (1 - \lambda) z_\pi(s_l),$$

where λ denotes weight parameter and $z_\pi(s_l)$ is the simulation output using p_π from s_l.

The idea of play-self originates from the success of *TD-Gammon* created by IBM in 1992 [30], [31]. TD-Gammon at that time was the only program that was skilled enough to play backgammon. However, by applying deep learning with RL, AlphaGo achieves superhuman level in the AI game challenge.

B. MULTITASK DEEP REINFORCEMENT LEARNING

One of the common defects of a neural network is that it is able to learn at most one task at a time. This problem, known as *catastrophic forgetting*, especially concerns continual learning of neural networks and RL [32], [33]. Particularly, this phenomenon occurs in the continuing learning process when task B is trained after task A. The knowledge of task A is instantly erased to acquire new information from task B.

A direct solution for this problem is to select a smart configuration of the neural network with a regularization strategy [34]–[36] or increase "progressively" the capacity of the network so as to learn multiple tasks at once [37], [38]. However, these methods limit the number of tasks the network can learn at the same time.

Another solution is using a process of *policy distillation* or *knowledge transfer* [39]–[41]. This strategy learns each problem domain individually and afterward transfers all the knowledge to a single multitask network. However, this solution encounters a side effect known as negative transfer. Negative transfer is the phenomenon when the network performs well in each problem domain but fails to operate in the multitask scenario. In 2017, Yin and Pan [41] designed a multitask policy architecture, *Hierarchical Prioritized Experience Replay* (HPER), which utilizes high-level features of a task to subdue the effect of negative transfer. To reduce the enormous data when attaining knowledge from each problem domain, HPER selects only important information from the experience replay memory.

To further learn more tasks with minimal computations and without changing the network configuration, Google's Deepmind [42] again proposed the algorithm, *Elastic Weight Consolidation* (EWC), which emulates the concept of synaptic consolidation in neuroscience [43]. EWC utilizes the property of a neural network that: given a task A, there is always a set of configurations of the network, μ, which can yield the same output performance [44]. Therefore, there exists a solution of task B, μ_B^*, which is close to the solution of task A, μ_A^*. When learning task B, the goal is to restrict the important parameters of task A inside the low error region of task A, centered by μ_A^*. This fact turns out to minimize the following loss function [42]:

$$\Xi(\mu) = \Xi_B(\mu) + \sum_i \frac{\rho}{2} F_i(\mu_i - \mu_{A,i}^*)^2,$$

where ρ expresses the importance between task A and task B, $\Xi_B(\mu)$ is the loss function for task B only, i denotes each parameter, and F_i is a Fisher information matrix that selectively finds important parameters in task A. By using DQN, EWC succeeds in training an agent to learn multiple Atari games in the RL context. Fig. 7 summarizes the basic idea of EWC.

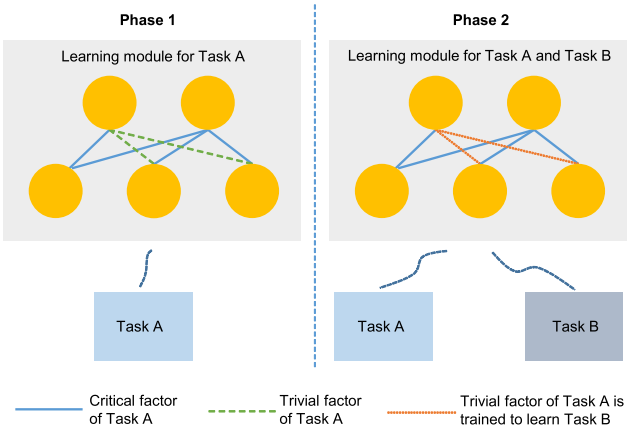

FIGURE 7. Training a network to learn multitask using EWC idea.

C. MULTIAGENT DEEP REINFORCEMENT LEARNING

Deep RL has a clear value in an extensive array of multiagent systems (MASs) such as a soccer team of robots, multiplayer online games, cooperative robots in the production chain, and autonomous military systems like unmanned aerial vehicles, surveillance, and spacecraft [7], [45], [46].

In MASs, each agent not only learns to operate independently but also cooperates with others to achieve the best joint reward. The direct strategy is to apply independent Q-learning in each agent and consider other agents as a part of the environment [47]. However, this approach limits the number of agents because it is computationally expensive to train every agent in the system. In 2016, Kraemer and Banerjee [48]

proposed a centralized approach, where a group of agents can be guided at the same time by using a centralized algorithm via an open communication channel. After the training, the agents are allowed to communicate over a limited bandwidth channel (to preserve energy), and thus can operate freely in a decentralized manner. Afterward, Foerster *et al.* [49] extended the centralized approach by developing two novel control schemes for an MAS. Instead of using an independent Q-learning, each agent needs to learn two RL problems at the same time: a goal-directed problem and a communication problem. In the goal-directed task, agents are required to select an action that may yield a high potential of long-term reward in the partially observed environment. At the same time, they must decide upon a suitable communication action to cooperate with each other in the MAS. However, this approach, known as *Reinforced Inter-Agent Learning* (RIAL), only shares parameters among agents to utilize centralized training. An improved version of RIAL, *Differentiable Inter-Agent Learning* (DIAL), enables the feedback from a communication channel by sending nonverbal cues to indicate the level of interest. In this way, DIAL ultimately trains the MAS in a centralized way. However, RIAL and DIAL only work in a discrete communication channel.

As opposed to RIAL and DIAL, Sukhbaatar and Fergus [50] introduced a novel model named *Communication Neural Net* (CommNet) that was used to support agents to cooperate via a continuous communication channel. The authors modeled each agent as a deep feed-forward neural network, which can access a shared communication channel C. In operation, each agent receives a summed transmission data (continuous vector) from other agents via C. In this way, CommNet becomes a general framework that can combine with an RL method to train a set of agents to communicate in a backpropagation manner. The model is shown to be versatile and can be used with any number and any kind of agents in the partially observed environment.

Another notable work regarding multiagent RL was presented in [51]. In that paper, He *et al.* approached multiagent problems from a different perspective. Instead of constructing a direct communication model like CommNet, the authors modeled an opponent agent used to compete in the joint reward. In this way, the effectiveness of the training agent can be increased without knowledge of the problem domain.

D. ASYNCHRONOUS DEEP REINFORCEMENT LEARNING

Another issue of DQN is the lengthy training period. For instance, training an Atari game may require 14–15 days using a single GPU with DQN. In 2015, Nair *et al.* [52] introduced the *General Reinforcement Learning Architecture* (Gorila) framework that enables DQN to operate in a distributed manner. Specifically, Gorila uses the AC architecture (Fig. 8) as mentioned in subsection II-D to facilitate the training process asynchronously. In Gorila, there are two main components: learning processes and central parameter servers. In each learning process, an actor operates in its own

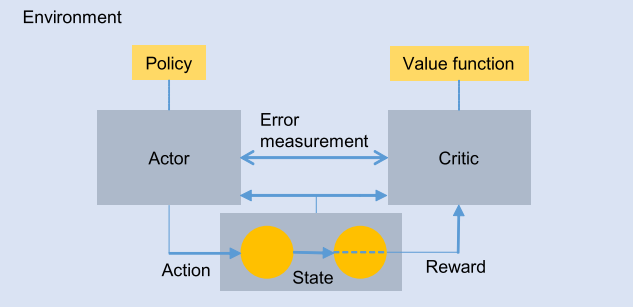

FIGURE 8. The AC architecture for asynchronous training process to reduce DQN training time.

copy model of the environment, and a critic uses the experience replay memory to compute the loss function $L(\theta)$ (12). The central parameter server uses distributed gradients $L_i(\theta)$ from each learning process i to update its model replica. Finally, the learning process receives updated parameters from the central server in a fixed time interval. The simulation showed that a 6-day training with 100 parallel actors in Gorila surpasses 12–14 days of training with a single GPU in 41 out of 49 games in the Atari domain. Although Gorila provides a significant improvement in the training process, it requires significant resource allocation.

Recently, Schaul *et al.* [53] introduced a simple strategy to improve the learning process. The authors suggested a prioritized scheme to promote critical transitions from the experience replay memory, i.e., meaningful experiences are sampled more frequently than trivial ones. Additionally, Wang *et al.* [54] used the prioritized experience replay to propose a duel network that calculates state and state-action function in parallel. This scheme has proven to provide better policy evaluation compared to DQN in the Atari domain.

In 2016, Mnih *et al.* [55] proposed a lightweight framework for asynchronous deep RL methods, named *Asynchronous Advantage Actor-Critic* (A3C). A3C surpasses the prioritized experience replay method and the dual network scheme regarding reduction of training time. As opposed to Gorila, A3C scheme allows concurrent learning in a single multi-core CPU and hence preserves allocated resources. Specifically, A3C enables agents to work in various environments at the same time by parallelizing multiple actors-learners in the AC architecture. Finally, A3C expresses a tendency to select an action via an advantage function that is the error measurement between the model's prediction quality of the action taken with its actual value. A3C also creates the decorrelation between experiences and eventually makes the learning outcome stable and convergent. The simulation showed that A3C can train an agent to learn Atari Breakout in less than 12 hours compared to 3–4 days of using DQN alone.

E. OTHER LIMITATIONS OF DQN AND RECENT SOLUTIONS

1) NON-MARKOV MODEL

DQN can solve MDP problems under the assumption that the next state $s' = s_{t+1}$ replies solely on the current state s_t and its corresponding action a_t regardless of the history.

This assumption restricts the generality of DQN. Therefore, it is intractable to solve a complex game when the next state not only depends on the current state but also upon a history of states, seen in games such as Pong or Double Dunk. These games are undoubtedly *Partially Observable MDPs* (POMDPs). In 2015, Hausknecht and Stone [56] introduced the *Deep Recurrent Q-Network* (DRQN) by adding a recurrent layer to DQN. Although DRQN is running on one frame at each time step, it can estimate the information requirements underlying the system states through time. Therefore, DRQN extends DQN by estimating the number of history frames needed for POMDP games. More recently, Sorokin *et al.* [57] extended DRQN by integrating ''attention'' mechanisms into DRQN in order to highlight important parts in the learning process. This approach is called *Deep Attention Recurrent Q-Network* (DARQN). DARQN was shown to promote high performance on the game Seaquest when compared against results of DQN and DRQN.

2) CONTINUOUS RL PROBLEM

DQN cannot be applied to a problem with continuous action space because it relies chiefly on the Q-learning mechanism. In 2015, Lillicrap *et al.* [58] employed the AC architecture to present an off-policy algorithm that can operate in a continuous action space. This approach, called *Deep Deterministic Policy Gradient* (DDPG), uses a parameterized actor function to map states to actions deterministically, while keeping DQN learning on the critic side. The mechanism is applied successfully to a range of continuous RL problems such as the pole-balancing task, legged locomotion, skillful manipulation, and car driving.

3) OVERFITTING

Overfitting has been shown to occur in DQN. In 2015, Hasselt *et al.* [59] suggested a double DQN scheme, which surpassed the performance of DQN in the Atari domain. Double DQN separates the selection from evaluation, i.e., it learns two value functions at the same time and hence results in two sets of parameters (θ_1 and θ_2), where θ_1 specifies greedy policy and θ_2 determines the value function. In summary, the target function can be formulated as [59]:

$$T_t = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a \mid \theta_1) \mid \theta_2).$$

More recently, Mnih *et al.* [55] introduced the A3C mechanism (subsection III-D) that even outperforms double DQN in terms of training time and stability.

4) INTRINSIC MOTIVATION

Another challenge in RL is working with a complicated environment where feedback is sparse. This problem leads to poor exploration and causes vulnerable behaviors of agents. To handle such environments, the RL problem is divided into hierarchical (tree) subtasks so that the parent subtask has higher abstraction than the child subtask. This study was originated from the concept of *Hierarchical Reinforcement Learning* [60]–[63]. In hierarchical RL, the agent needs to learn different levels of temporal abstraction to explore the environment efficiently. At the same time, the study on *intrinsic motivation* [64] focuses on finding a natural and good intrinsic reward function that encourages self-motivation when facing with the selection of generic actions. In 2016, Kulkarni *et al.* [65] combined hierarchical deep RL with intrinsic motivation to aid agents in learning within the complex environment. This approach can work efficiently in a sparse and delayed feedback environment like the Atari game Montezuma's Revenge.

F. REINFORCEMENT LEARNING EVALUATION FRAMEWORKS

In this subsection, the four latest framework libraries that can be used to develop deep RL algorithms are reviewed. Particularly, analysis of the libraries is carried out in different aspects so as to provide a proper selection of the RL framework when working with a specific problem domain. Table 1 summarizes advantages and disadvantages of these libraries.

TABLE 1. Overview of RL frameworks.

Framework	Pros	Cons
OpenAI Gym	• Python library • Compatible with TensorFlow and Theano • Easy to integrate • Various environments • Robot simulation • MAS	○ Immature library ○ Lack of generality ○ Lack of monitoring tools
Burlap	• Java library • General purpose • Various built-in RL algorithms • Pre-made domains • RL-Glue interface • MAS • Visualization tools	○ Java library ○ Not support deep RL directly ○ Low-level integration
RL-Glue	• Language independence • Reuse elements • Multiplatform	○ Spotty documentation ○ Small user community
ALE	• Atari domain • Support Python • Multiplatform • Visualization tools	○ Only for Atari domain ○ Support only 61 games

1) OpenAI GYM

OpenAI Gym [66] is the most robust toolkit to monitor and compare RL algorithms. It provides a friendly Python interface and is compatible with third-party numerical libraries such as TensorFlow, Theano, Keras, and Scikit-learn [67]–[70]. Succinctly, OpenAI Gym can compare performance of RL agents in different environmental simulations. Recently, the release of *Roboschool* (integrated

in OpenAI Gym) [71] provides a new environment for robot simulation in an MAS context.

2) THE BROWN-UMBC REINFORCEMENT LEARNING AND PLANNING (BURLAP)

Burlap [72] is a Java library used to develop an RL algorithm for single and multiagent systems. As opposed to OpenAI Gym, Burlap establishes a general framework so that users can define a problem domain themselves. It provides a wide array of built-in RL algorithms, premade domains, and visualization tools.

3) RL-GLUE

RL-Glue [73] facilitates a common interface that connects different pieces of RL programs together even if they are in different programming languages. In this way, agents can be reused as well as environments that have been written by other developers in the RL research community.

4) THE ARCADE LEARNING ENVIRONMENT (ALE)

ALE [21] provides an environment framework for 61 out of 2600 classic Atari games. ALE can be used to compare the performance of RL agents in the Atari domain. It supports Python interface and has a set of visualization tools.

G. SYSTEM ARCHITECTURE FOR A HUMAN-LEVEL AGENT

Before the conclusion of this review, we propose a high-level *System Architecture for a Human-level Agent* (SAHA) by combining recent breakthroughs in different aspects as illustrated in Fig. 9. The goal is to build an agent that can 1) communicate with other agents in MASs, 2) learn and work in multitask scenarios, and 3) only requires a short period of training. In SAHA, we use the CommNet model in our communication module to provide generality when dealing with various types of problem domains. To deal with different kinds of agents, we add a common interface, as well as an encoder, and a decoder to interpret transmission data among agents via a shared communication channel. The core operation of the system is in the controller, which includes a multilayer neural network (DQN, DRQN, etc.). The specific architecture of the network is dependent upon the problem domain. Inputs of the neural network involve continuous vector data (summed transmission data from other agents) from the communication module and a task-specific data from the task specification module. We add EWC and A3C architecture to enable multitask learning and asynchronous learning respectively, to reduce training time. Finally, the online/offline control can be used to change the learning mode of the agent. In summary, the proposed system design describes the usage of recent breakthroughs to collectively establish a practical human-level agent that can adapt with various kinds of problem domains.

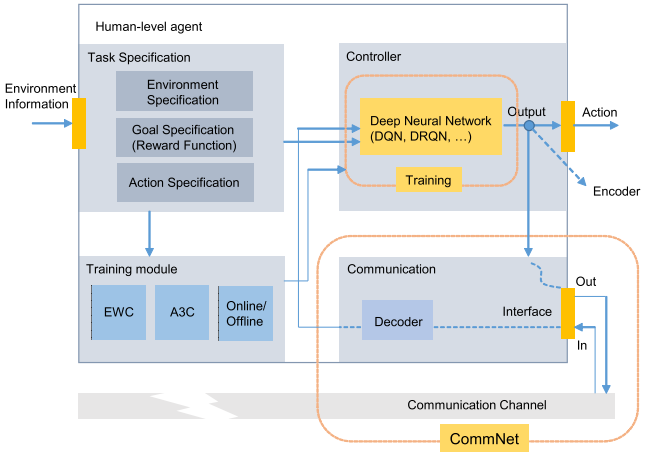

FIGURE 9. The proposed high-level system architecture for a human-level agent (SAHA).

TABLE 2. Summary of key reviewed papers.

Problems	Deep RL	Multitask deep RL	Multiagent deep RL	Asynchronous deep RL
Classic Atari	[20][59]	[38]–[42]	[47]	[52]–[55]
POMDP Atari	[56][57]			
Sparse Atari	[65]			
Go	[22]			
Switch Riddle			[49]	
MNIST Games			[49]	
Lever Pulling			[50]	
Traffic Junction			[50]	
Combat Task			[50]	
Soccer			[51]	
Quiz Bowl			[51]	
TORCS Racing				[55]
MuJuCo				[55]
Labyrinth				[55]
Cartpole	[58]			
Gripper	[58]			
Cheetah	[58]			
Discrete SDP*	[65]			

(*) SDP stands for Stochastic Decision Process

IV. DISCUSSIONS AND CONCLUSIONS

In this paper, we have examined recent representative deep RL breakthroughs that actively contribute to every aspect of modeling a human-level agent. Table 2 summaries key deep RL studies and their corresponding applications. The combination of RL and deep learning indeed yields excellent results and marks the evolution of modern RL in the advances of prospective autonomous systems. Deep RL capability has attracted great interests among the research community. Although deep RL and its variants provide a certain level of success within the field of artificial intelligence, it still has limitations that we need to discuss and alleviate.

To some extent, deep RL takes part in solving the dilemma of dimensionality by using deep learning to approximate an RL problem with high-dimensional input. However, the complete proof of this dilemma is still an open question.

Therefore, to understand the limitations of deep RL, as well as to find proof of this problem, we analyze the complexity of the enigma ''the curse of dimensionality.''

In the real world, a human receives an enormous amount of input data via the five-sense system: seeing, hearing, smelling, tasting, and touching. These five-dimensional data then transfer information to the brain in order to analyze and control the body to react with the environment. The action is selected based on our self-motivation (intrinsic motivation), the knowledge learned from other experienced sources (knowledge transfer), self-experience (RL), and critical thinking. Humans then combine all the information to manage an assigned work (single or multiple tasks) efficiently. Humans also communicate with each other to finish a task cooperatively (MAS). Therefore, what is the clue here? Three important factors that make up a human-level agent that should be discussed: multitask learning, multiagent system, and critical thinking.

Firstly, recent research on multitask learning using RL is quite sparse. One characteristic of the neural network is the ability to learn at most one task at a time. This limitation restricts the RL method from learning multiple tasks at the same time. Moreover, the input information from each problem domain when combined together yields an enormous amount of data that are unsuitable for current RL methods. Therefore, it is essential to conduct an extensive research into this problem.

Secondly, the agent needs to communicate and cooperate with other agents to finish a common task. The current approach examines multiagent systems as a single RL problem by considering partner agents as part of the environment or constructing a separate communication channel among agents. These approaches, however, inherently inflate the data space balloon and makes the multiagent problem intractable.

Finally, the third characteristic of a prospective agent discussed here is critical thinking. Very few studies have considered this ability for an AI agent. This problem is difficult because a large amount of data is required to synthesize in order for the agent to ''think'' like a human. This ability is critical because it distinguishes the intelligence of a human. Can we create an agent that can think and be creative? Recent advances in hierarchical RL, intrinsic motivation, meta-learning [74] (ability to adjust itself to adapt with the environment) and inverse RL [75] (a process of inferring reward function via demonstrations) are the first steps in the progress of building a creative agent.

All of these three factors along with deep RL will represent a complete solution to the curse of dimensionality. These important factors paint the last layer to complete the RL picture.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction*. Cambridge, MA, USA: MIT Press, 2012.
[2] J. Kober, J. A. Bagnell, and J. Peters, ''Reinforcement learning in robotics: A survey,'' *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.
[3] S. Mahadevan and J. Connell, ''Automatic programming of behavior-based robots using reinforcement learning,'' *Artif. Intell.*, vol. 55, nos. 2–3, pp. 311–365, Jun. 1992.
[4] S. Schaal, ''Learning from demonstration,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 1997, pp. 1040–1046.
[5] J.-L. Lin, K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, ''Gait balance and acceleration of a biped robot based on Q-learning,'' *IEEE Access*, vol. 4, pp. 2439–2449, 2016.
[6] K. Mülling, J. Kober, O. Kroemer, and J. Peters, ''Learning to select and generalize striking movements in robot table tennis,'' *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 263–279, 2013.
[7] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, ''Reinforcement learning for robot soccer,'' *Auton. Robots*, vol. 27, no. 1, pp. 55–73, 2009.
[8] E. L. Thorndike, ''Animal intelligence: An experimental study of the associate processes in animals,'' *Amer. Psychol.*, vol. 53, no. 10, pp. 1125–1127, 1998.
[9] W. Schultz, P. Dayan, and P. R. Montague, ''A neural substrate of prediction and reward,'' *Science*, vol. 275, no. 5306, pp. 1593–1599, 1997.
[10] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 2010.
[11] L. Deng and D. Yu, ''Deep learning: Methods and applications,'' *Found. Trends Signal Process.*, vol. 7, nos. 3–4, pp. 197–387, Jun. 2014.
[12] L. Fei-Fei, J. Deng, and K. Li, ''ImageNet: Constructing a large-scale image database,'' *J. Vis.*, vol. 9, no. 8, p. 1037, 2009.
[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ''ImageNet classification with deep convolutional neural networks,'' in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1097–1105.
[14] K. Simonyan and A. Zisserman. (Sep. 2014). ''Very deep convolutional networks for large-scale image recognition.'' [Online]. Available: https://arxiv.org/abs/1409.1556
[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, ''Large-scale video classification with convolutional neural networks,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1725–1732.
[16] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn, ''Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition,'' in *Proc. IEEE Conf. Acoust. Speech Signal Process.*, Mar. 2012, pp. 4277–4280.
[17] M.-T. Luong, M. Kayser, and C. D. Manning, ''Deep neural language models for machine translation,'' in *Proc. Conf. Comput. Nat. Lang. Learn.*, Jul. 2015, pp. 305–309.
[18] H. A. Pierson and M. S. Gashler, ''Deep learning in robotics: A review of recent research,'' *Adv. Robot.*, vol. 31, no. 16, pp. 821–835, 2017.
[19] C. J. C. H. Watkins and P. Dayan, ''Q-learning,'' *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[20] V. Mnih *et al.*, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[21] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, ''The arcade learning environment: An evaluation platform for general agents,'' *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, May 2013.
[22] D. Silver *et al.*, ''Mastering the game of Go with deep neural networks and tree search,'' *Nature*, vol. 529, no. 7578, pp. 484–489, Jan. 2016.
[23] V. R. Konda and J. N. Tsitsiklis, ''Actor-critic algorithms,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.
[24] J. N. Tsitsiklis and B. Van Roy, ''An analysis of temporal-difference learning with function approximation,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 1997, pp. 1075–1081.
[25] M. G. Bellemare, J. Veness, and M. Bowling, ''Investigating contingency awareness using Atari 2600 games,'' in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2012, pp. 864–871.
[26] M. Campbell, A. J. Hoane, Jr., and F.-H. Hsu, ''Deep blue,'' *Artif. Intell.*, vol. 134, nos. 1–2, pp. 57–83, Jan. 2002.
[27] R. Coulom, ''Efficient selectivity and backup operators in Monte-Carlo tree search,'' in *Proc. Int. Conf. Comput. Games*, 2006, pp. 72–83.
[28] C. B. Browne *et al.*, ''A survey of Monte Carlo tree search methods,'' *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[29] L. Kocsis and C. Szepesvári, ''Bandit based Monte-Carlo planning,'' in *Proc. Eur. Conf. Mach. Learn.*, vol. 6. 2006, pp. 282–293.
[30] G. Tesauro and G. R. Galperin, ''On-line policy improvement using Monte-Carlo search,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 1997, pp. 1068–1074.
[31] G. Tesauro, ''Temporal difference learning and TD-Gammon,'' *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[32] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends Cognit. Sci.*, vol. 3, no. 4, pp. 128–135, 1999.
[33] D. Kumaran, D. Hassabis, and J. L. McClelland, "What learning systems do intelligent agents need? Complementary learning systems theory updated," *Trends Cognit. Sci.*, vol. 20, no. 7, pp. 512–534, 2016.
[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[35] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, no. 2, pp. 219–269, Mar. 1995.
[36] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. (Dec. 2013). "An empirical investigation of catastrophic forgetting in gradient-based neural networks." [Online]. Available: https://arxiv.org/abs/1312.6211
[37] S. Thrun and L. Pratt, *Learning to Learn*. Boston, MA, USA: Kluwer, 1998.
[38] A. A. Rusu *et al.* (2016.). "Progressive neural networks." [Online]. Available: https://arxiv.org/abs/1606.04671
[39] A. A. Rusu *et al.* (Nov. 2015). "Policy distillation." [Online]. Available: https://arxiv.org/abs/1511.06295
[40] E. Parisotto, J. L. Ba, and R. Salakhutdinov. (2015). "Actor-mimic: Deep multitask and transfer reinforcement learning." [Online]. Available: https://arxiv.org/abs/1511.06342
[41] H. Yin and S. J. Pan, "Knowledge transfer for deep reinforcement learning with hierarchical experience replay," in *Proc. AAAI Conf. Artif. Intell.*, Jan. 2017, pp. 1640–1646.
[42] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," in *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 3, pp. 3521–3526, 2017.
[43] C. Clopath, "Synaptic consolidation: An approach to long-term learning," *Cognit. Neurodyn.*, vol. 6, no. 3, pp. 251–257, Jun. 2012.
[44] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input-output map," *J. Neural Netw.*, vol. 5, no. 4, pp. 589–593, Jul./Aug. 1992.
[45] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Trans. Autom. Control*, vol. 51, no. 3, pp. 401–420, Mar. 2006.
[46] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 1994, pp. 157–163.
[47] A. Tampuu *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, p. e0172395, Apr. 2017.
[48] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, May 2016.
[49] J. Foerster, Y. A. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2137–2145.
[50] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2244–2252.
[51] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé, III, "Opponent modeling in deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1804–1813.
[52] A. Nair *et al.* (2015). "Massively parallel methods for deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1507.04296
[53] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. (2015). "Prioritized experience replay." [Online]. Available: https://arxiv.org/abs/1511.05952
[54] Z. Wang *et al.* (2015). "Dueling network architectures for deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1511.06581
[55] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[56] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Symp. Seq. Decis. Mak. Intell. Agents*, Nov. 2015. [Online]. Available: https://www.cs.utexas.edu/~pstone/Papers/bib2html/b2hd-SDMIA15-Hausknecht.html
[57] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva. (2015). "Deep attention recurrent Q-network." [Online]. Available: https://arxiv.org/abs/1512.01693
[58] T. P. Lillicrap *et al.* (2015). "Continuous control with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1509.02971
[59] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[60] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, Oct. 2003.
[61] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, nos. 1–2, pp. 181–211, Aug. 1999.
[62] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored MDPs," *J. Artif. Intell. Res.*, vol. 19, pp. 399–468, Jul./Dec. 2003. [Online]. Available: http://www.jair.org/contents.html
[63] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.
[64] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2005, pp. 1281–1288.
[65] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.
[66] G. Brockman *et al.* (2016). "OpenAI gym." [Online]. Available: https://arxiv.org/abs/1606.01540
[67] M. Abadi *et al.* (2016). "TensorFlow: Large-scale machine learning on heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1603.04467
[68] J. Bergstra *et al.*, "Theano: Deep learning on GPUs with Python," in *Proc. BigLearn Workshop NIPS*, 2011.
[69] F. Chollet. (2015). *Keras*. [Online]. Available: http://keras.io
[70] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[71] O. Klimov and J. Schulman. (2017). *Roboschool*. [Online]. Available: https://blog.openai.com/roboschool/
[72] J. MacGlashan. (2016). *Brown-UMBC Reinforcement Learning and Planning (BURLAP)*. [Online]. Available: http://burlap.cs.brown.edu/
[73] B. Tanner and A. White, "RL-glue: Language-independent software for reinforcement-learning experiments," *J. Mach. Learn. Res.*, vol. 10, pp. 2133–2136, Sep. 2009.
[74] A. Makmal, A. A. Melnikov, V. Dunjko, and H. J. Briegel, "Meta-learning within projective simulation," *IEEE Access*, vol. 4, pp. 2110–2122, 2016.
[75] P. Abbeel and A. Y. Ng, "Inverse reinforcement learning," in *Encyclopedia of Machine Learning*. New York, NY, USA: Springer, 2011, pp. 554–558.

NGOC DUY NGUYEN received the M.S. degree in computer engineering from Sungkyunkwan University, Seoul, South Korea, in 2011. He is currently pursuing the Ph.D. degree with the Institute for Intelligent Systems Research and Innovation, Deakin University, Australia.

From 2011 to 2016, he was a Project Manager and a Researcher with Iritech, Inc., Seoul, where he was involved in the research and development of world-leading biometrics and recognition systems. His research interest involves machine learning, optimization problems, and system design.

Dr. Nguyen received the Best Thesis Award funded by the Department of Information and Communication Engineering, Sungkyunkwan University.

THANH NGUYEN received the Ph.D. degree in mathematics and statistics from Monash University, Australia, in 2013. He is currently a Research Fellow with the Institute for Intelligent Systems Research and Innovation, Deakin University, Australia.

He was a Visiting Scholar with the Computer Science Department, Stanford University, CA, USA in 2015. He has published various peer-reviewed papers in the field of computational and artificial intelligence. His current research interests include applied statistics and machine learning. He was a recipient of an Alfred Deakin Post-Doctoral Research Fellowship in 2016.

SAEID NAHAVANDI (M'91–SM'07) received the Ph.D. degree from Durham University, U.K., in 1991. He is an Alfred Deakin Professor, the Pro Vice-Chancellor (Defence Technologies), the Chair of Engineering, and the Director of the Institute for Intelligent Systems Research and Innovation, Deakin University.

His research interests include the modeling of complex systems, robotics, and haptics. He has published over 600 papers in various international journals and conferences. He is a fellow of the Engineers Australia and Institution of Engineering and Technology.

He is the Co-Editor-in-Chief for the IEEE SYSTEMS JOURNAL, an Associate Editor for the IEEE/ASME TRANSACTIONS ON MECHATRONICS and the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS, and an Editorial Board Member of the IEEE ACCESS.