Haibo He
Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island, Kingston, RI, USA

Xiangnan Zhong
Department of Electrical Engineering, University of North Texas, Denton, TX, USA

Learning Without External Reward

Abstract

In the traditional reinforcement learning paradigm, a reward signal is applied to define the goal of the task. Usually, the reward signal is a "hand-crafted" numerical value or a pre-defined function: it tells the agent how good or bad a specific action is. However, we believe there exist situations in which the environment cannot directly provide such a reward signal to the agent. Therefore, the question is whether an agent can still learn without the external reward signal or not. To this end, this article develops a self-learning approach which enables the agent to adaptively develop an internal reward signal based on a given ultimate goal, without requiring an explicit external reward signal from the environment. In this article, we aim to convey the self-learning idea in a broad sense, which could be used in a wide range of existing reinforcement learning and adaptive dynamic programming algorithms and architectures. We describe the idealized forms of this method mathematically, and also demonstrate its effectiveness through a triple-link inverted pendulum case study.

I. Introduction

When we first think about the nature of learning, we probably start with the idea that we learn by interacting with the environment [1]–[3]. For instance, when we try to hold a conversation with others, we need to decide what to say based on the people we are talking to as well as the conversational context. Over the past several decades, many researchers have explored computational approaches to learn from active interactions with the environment, such as reinforcement learning (RL) and adaptive dynamic programming (ADP). Imagine we hope to train a monkey to learn the result of "1+1". At first, we present two cards with number "1" on them to the monkey: If it picks a card with the number "2" on it from a box, we present a banana as a reward. In this way, although the monkey does not know the exact meaning of math, it knows that a banana can be given when it picks the appropriate card. Therefore, the banana reward plays an important role in the learning process. In general, the key element of RL is defined by the reward signal, which is given by the environment [1], [4]–[6]. In order to achieve goals, the agent chooses a set of actions that maximize the expected total rewards it receives over time. Therefore, RL achieves goals by defining the interaction between an agent and its environment in terms of states, actions, and rewards [1], [7]. Recently, the development of deep RL [8]–[10] has attracted increasing attention, especially for the level of intelligence it has achieved.

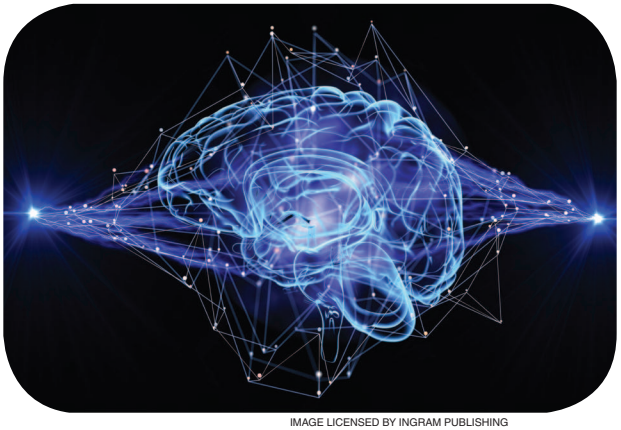

IMAGE LICENSED BY INGRAM PUBLISHING

So far, many RL/ADP designs focus on how to calculate and maximize the cumulative rewards [11]–[15]. Usually, it is assumed that the agent knows what the immediate reward is or how the immediate reward is computed as a function of the actions and states in which they are taken [16]. There are several approaches in the literature to define such a reward signal. For instance, a typical approach is to use a binary signal, e.g., using a "0" or "−1" to represent "success" or "failure" of an action [17], or a semi-binary reward signal, e.g., using "0, −0.4, −1" as a more informative representation [18]. Another way to define the reward signal is to use a quadratic function based on the system states and actions [19]–[22]. This type of

Digital Object Identifier 10.1109/MCI.2018.2840727
Date of publication: 18 July 2018

Corresponding Author: Haibo He (Email: haibohe@uri.edu)

1556-603X/18©2018IEEE

reward signal is more adaptive and very popular in system stabilization. Recently, a goal representation heuristic dynamic programming (GrHDP) method [23], [24] has been proposed in the literature. By introducing an internal reward signal, this method can automatically and adaptively provide information to the agent instead of hand-crafting. Generally, all these existing approaches require the environment to define the reward signal with *a priori* knowledge or domain expertise. This can be seen as an external teacher or trainer who provides the reward value or function for each action performed by the agent. However, what if this kind of teacher is unavailable or he/she cannot provide such a direct feedback to the agent for some reasons? Can the agent learn by itself in this case under the RL framework?

In this article, by considering the relationship between the goal and the system states and actions, we propose a method that enables the agent to learn only with the ultimate goal but no explicit external reward signals. To this end, the key contribution of this work is the development of a self-learning approach based on the ultimate goal, rather than obtaining the external supervision (reward signals) directly from the environment. We further develop the computational formulation of this self-learning idea based on a specific ADP design, and validate its performance and effectiveness based on a triple-link inverted pendulum case study. We would like to note that this article focuses on the self-learning of an agent's own goals, in contrast to observational learning or apprenticeship learning about other's goals such as in inverse RL [25].

II. The Key Idea: Self-Learning Design

In contrast to the traditional RL/ADP design, in which a specific reward signal passes from the environment to the agent to tell the effects ("good" or "bad"), our proposed self-learning ADP method enables the agent to establish the reward signal itself, which is called the *internal reward signal* in this article. The comparison of the agent-environment interaction in traditional and self-learning ADP design is described in Fig. 1. We can observe that instead of receiving the immediate reward signal from the environment (Fig. 1(a)), the agent in the self-learning ADP method, estimates an internal reward $s(t)$ to help achieve the goal (Fig. 1(b)). Hence, the communications between the environment and the agent at each time step are only the states and actions, which is fundamentally different to the existing RL and ADP methods.

Note that, in the traditional RL/ADP design, the use of the reward signal is to define the agent's goal of a task. However, in the self-learning ADP method, the reward signal is unavailable in the interaction. Since the reward signal reflects the evaluation of an action's effect, which is always paired with the goal, the agent in this learning process needs to learn what the reward signal is according to the ultimate goal. The effect of the action is compared to the goal within a common reference frame in order to assess the achievement [26]. The agent then learns how "good" or "bad" the action is at each time step by itself via the guidance from the ultimate goal. After that, based on the estimated internal reward signal and the system state, the agent generates the control action. This is to say, in order to achieve the ultimate goal, instead of learning to make the decision directly, the agent needs first to learn what the best reward signal is to represent the information upon which to base a certain action.

More specifically, the interaction between the agent and the environment happens sequentially, in discrete time steps. At each time step t, the agent selects an action $a(t)$ according to the representation of the environment $x(t)$. In consequence, the agent finds itself in a new state $x(t+1)$ and then estimates the corresponding internal reward signal $s(t+1)$ at next time step. For example, when we train a battery-charged robot to collect trash, the robot decides at each time step whether it should move forward to search more trash or find its way back to its battery charger. Its decision is based on its position and speed. As a consequence of its action and state, the robot estimates the reward signal $s(t) = f(x(t), a(t))$. Initially, the robot randomly assigns a value $s(0)$ since no prior knowledge is available about what to do. However, after trial-and-error learning, we want the robot to learn how to represent an action in a given state.

Let us stipulate that the agent's goal is to maximize the reward it estimates in the long term. If the reward sequence after time step t is as $s(t+1), s(t+2), s(t+3), \ldots$, then the value function can be described as

$$\begin{aligned} V(t) &= s(t) + \gamma s(t+1) + \gamma^2 s(t+2) \\ &\quad + \gamma^3 s(t+3) + \cdots \\ &= s(t) + \gamma V(t+1) \end{aligned} \tag{1}$$

where $0 < \gamma < 1$ is the discount factor. The discount factor determines if an immediate reward is more valuable than the rewards received in the far future. If $\gamma = 0$, the value function is equal to the

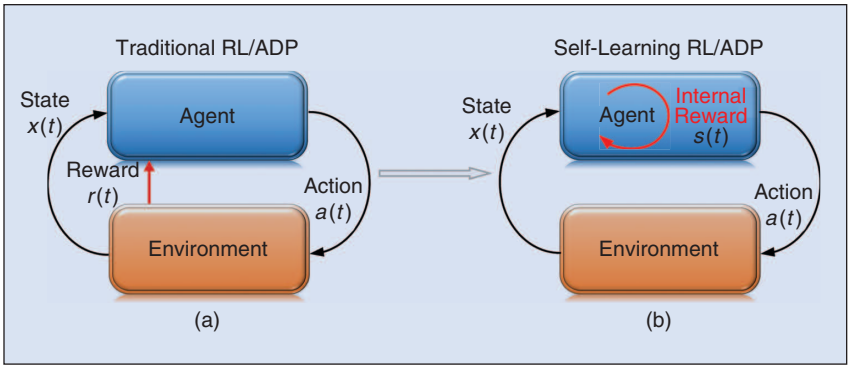

FIGURE 1 The conceptual diagram of agent-environment interaction: (a) Traditional RL/ADP design: the external reinforcement signal $r(t)$ passing from the environment to the agent; (b) Self-learning RL/ADP design: no external reinforcement signal needed during this process, and the agent estimating an internal reward signal $s(t)$ itself.

In contrast to the traditional RL/ADP design, in which a specific reward signal passes from the environment to the agent to tell the effects ("good" or "bad"), our proposed self-learning ADP method enables the agent to establish the reward signal itself, which is called the internal reward signal in this article.

immediate reward and the agent's goal becomes to maximize the immediate reward. This may reduce the possibility of accessing future rewards. When γ approaches 1, future rewards will increase their influence in selecting actions, therefore the agent becomes more farsighted [1]. We note here that $s(t)$ is the estimated internal reward signal, rather than the external reward signal as in the classic RL/ADP literature [1], [12], [13], [18], [27].

In this way, the optimal value function can be defined as

$$V^*(t) = \max_{a(t),s(t)} \{s^*(t) + \gamma V^*(t+1)\}. \quad (2)$$

Here $s^*(t)$ is the optimal internal reward signal that the agent learns to represent how good or bad an action is in a given state. Our objective is to ensure this signal can guide the agent to achieve the goal, i.e.,

$$s^*(t) = \operatorname{argmax}_{s(t)} \{V^*(t)\}. \quad (3)$$

Then, according to Bellman's optimality principle [28], the control action $a(t)$ can be given as

$$a^*(t) = \operatorname{argmax}_{a(t)} \{s^*(t) + \gamma V^*(t+1)\}. \quad (4)$$

Note that since the internal reward signal $s(t)$ is estimated by the agent, it is random at first during the training process. However, after trial-and-error learning, the agent can learn how to represent the internal reward signal based on the ultimate goal and update it accordingly. In this way, the internal reward signal is automatically decided within the agent in this approach. In Fig. 2, we use the neural network technique as an example to illustrate this idea. In different applications, one can choose any type of function approximators for implementation depending on the scenario and/or design. The internal reward signal estimation will be achieved by goal network with inputs $x(t)$, $a(t)$ and an output $s(t)$. The value function $V(t)$ will be generated by the critic network to adjust the updating process. Furthermore, the action network will provide the control action. Therefore, we can observe that instead of obtaining an explicit external reward signal directly from the environment, the agent in the proposed method self-learns an internal reward signal to guide its action based on the ultimate goal. This clearly distinguishes self-learning ADP from the existing ADP methods: (i) No external supervision is needed in this fully autonomous learning. The basic principle becomes finding an internal reward function which makes the agent achieve the ultimate goal. (ii) Communication burden is reduced since the interaction between

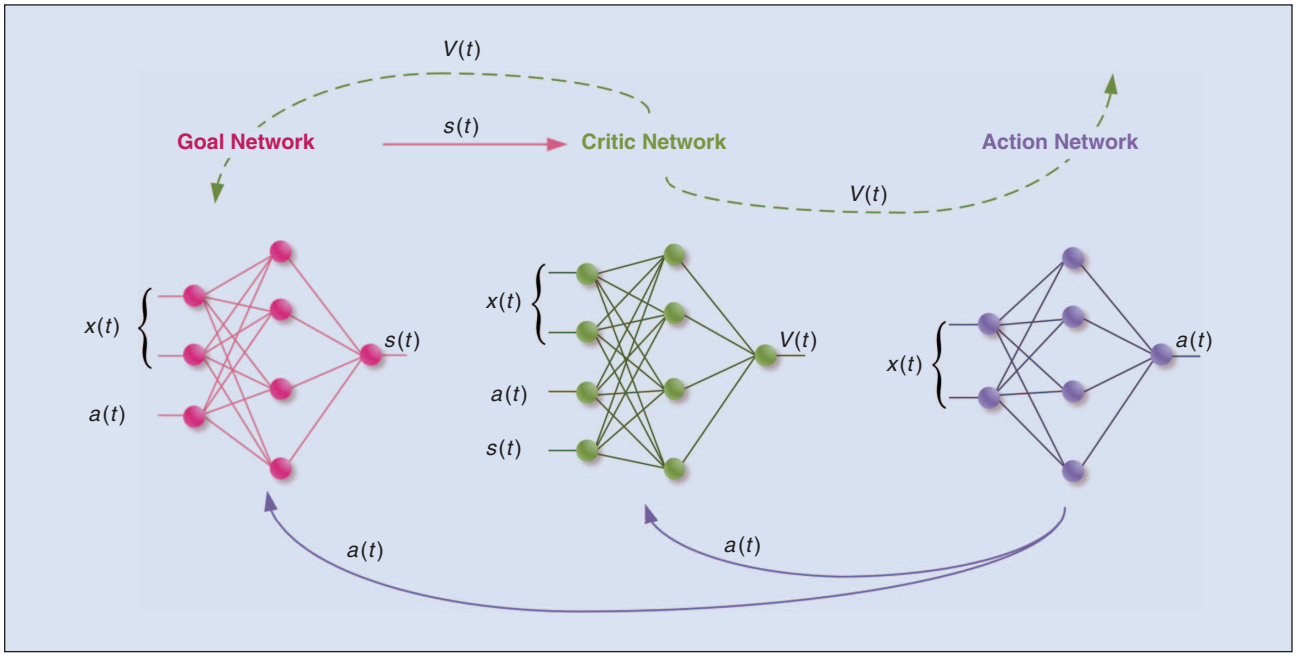

FIGURE 2 Neural network training process. The goal network takes the system state $x(t)$ and control action $a(t)$ as its input, and outputs an estimated internal reinforcement signal $s(t)$. The critic network similarly applies the neural network structure, but takes an additional input $s(t)$ and outputs the value function $V(t)$. Moreover, the action network uses a similar neural network structure with input $x(t)$ and output $a(t)$.

the agent and the environment at each time step only becomes the states and actions. This is important when the communication bandwidth or power resources are constrained.

In order to achieve the ultimate goal, instead of learning to make the decision directly, the agent needs first to learn what the best reward signal is to represent the information upon which to base a certain action.

III. Learning Architecture and Implementation

The key of implementing self-learning ADP design is to estimate Eqs. (2), (3), and (4). In our current research, we use three neural networks to design the *goal network, action network,* and *critic network,* as shown in Fig. 3. Here, we assume that the solution of the task exists and it is unique. The mathematical analysis of these three neural networks is provided as follows.

A. Goal Network

The goal network estimates the internal reward signal $s(t)$ in Eq.(3) based on the ultimate goal U_c. Since the internal reward signal is a response of the external environment, we set the inputs of the goal network as the state $x(t)$ and action $a(t)$. With these information, the goal network provides an estimation of the internal reward signal in order to optimize the value function $V(t)$. Therefore, the error can be described as:

$$e_g(t) = V(t) - U_c$$
$$E_g(t) = \frac{1}{2}e_g^2(t) \quad (5)$$

where U_c is the ultimate goal. The value of U_c is critical in the design and it could be variant in different applications. We apply backpropagation to train the goal network and update the goal network weights $\omega_g(t)$ as following:

$$\omega_g(t+1) = \omega_g(t) - \beta_g\left(\frac{\partial E_g(t)}{\partial \omega_g(t)}\right) \quad (6)$$

where β_g is a learning rate of the goal network. Followed by the chain backpropagation rule, we further have

$$\frac{\partial E_g(t)}{\partial \omega_g(t)} = \frac{\partial E_g(t)}{\partial V(t)}\frac{\partial V(t)}{\partial s(t)}\frac{\partial s(t)}{\partial \omega_g(t)}. \quad (7)$$

B. Critic Network

The value function $V(t)$ in Eq. (2) is estimated by the critic network. To closely connect the critic network with the goal network, we set the estimated internal reward signal $s(t)$ as one of the inputs to the critic network. Therefore, the inputs of the critic network include the state $x(t)$, the action $a(t)$, and the internal reward signal $s(t)$. The critic network aims to minimize the following error function over time:

$$e_c(t) = \gamma V(t) - [V(t-1) - s(t-1)]$$
$$E_c(t) = \frac{1}{2}e_c^2(t). \quad (8)$$

Note that, although both $V(t)$ and $V(t-1)$ depend on critic network's weights ω_c, we do not account for the dependence of $V(t-1)$ on weights ω_c when minimizing the error in Eq. (8) [16]. Therefore, the updating scheme of the critic network can be defined as:

$$\omega_c(t) = \omega_c(t) - \beta_c\left(\frac{\partial E_c(t)}{\partial \omega_c(t)}\right) \quad (9)$$

where β_c is the learning rate of the critic network. Chain backpropagation rule can be applied to further calculate $\frac{\partial E_c(t)}{\partial \omega_c(t)}$:

$$\frac{\partial E_c(t)}{\partial \omega_c(t)} = \frac{\partial E_c(t)}{\partial V(t)}\frac{\partial V(t)}{\partial \omega_c(t)}. \quad (10)$$

C. Action Network

At any time instant, action network provides the control action $a(t)$ for the system based on the system states $x(t)$. Therefore, we define the error function as $e_a(t) = V(t) - U_c$ and $E_a(t) = \frac{1}{2}e_a^2(t)$. The weights updating rule can be defined as:

$$\omega_a(t+1) = \omega_a(t) - \beta_a\left(\frac{\partial E_a(t)}{\partial \omega_a(t)}\right) \quad (11)$$

where β_a is the learning rate of the action network. With the chain backpropagation rule, we have:

$$\frac{\partial E_a(t)}{\partial \omega_a(t)} = \frac{\partial E_a(t)}{\partial V(t)}\frac{\partial V(t)}{\partial a(t)}\frac{\partial a(t)}{\partial \omega_a(t)}. \quad (12)$$

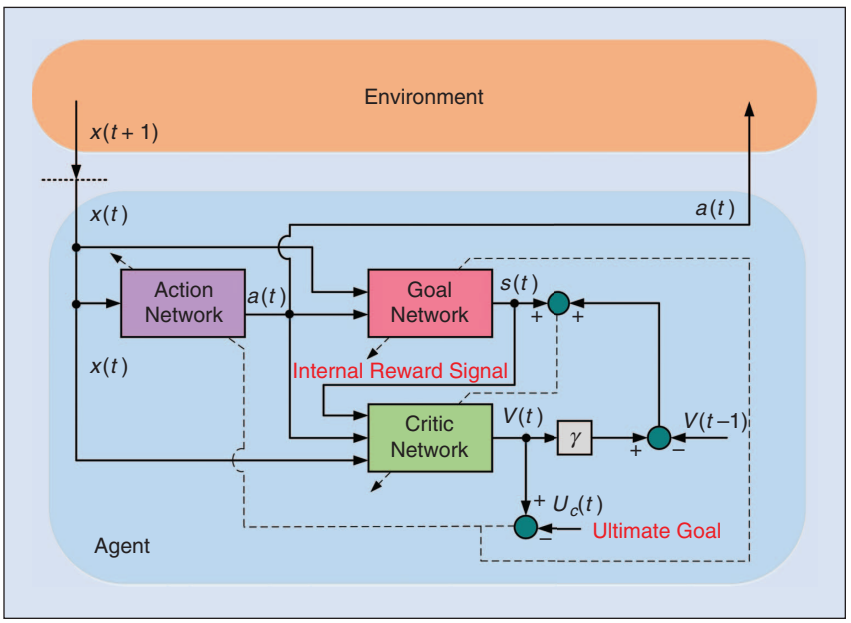

FIGURE 3 Implementation architecture of self-learning adaptive dynamic programming design: three neural networks are established as the action network, the critic network and the goal network.

The algorithm for the proposed self-learning ADP approach is given in [Algorithm 1], where step 1 setups the variables and ingredients of three neural networks, and then steps 2-26 describe the online training process.

Algorithm 1 **Self-learning adaptive dynamic programming.**

1: $a \leftarrow f_a(x_a, \omega_a)$, control action selection;
f_a: the action network;
x_a: inputs of action network, $x_a = [x]$;
ω_a: weights in action network;
a: control action;
$s \leftarrow f_g(x_g, \omega_g)$, internal reward signal choosing;
f_g: the goal network;
x_g: input of goal network, $x_g = [x, a]$;
ω_g: weights in goal network;
s: internal reward signal;
$V \leftarrow f_c(x_c, \omega_c)$, value function mapping;
f_c: the critic network;
x_c: input of critic network, $x_c = [x, a, s]$;
ω_c: weights in critic network;
V: value function;
N_a, N_g, and N_c: internal cycles of the action network, goal network, and critic network, respectively;
T_a, T_g, and T_c: internal training error thresholds for the action network, goal network, and critic network, respectively;
2: **for** 1 to MaxRun **do**
3: Initialize $\omega_a(0), \omega_g(0), \omega_c(0)$;
4: $x(t) \leftarrow System(x(t-1), a(t-1))$; // execute action and obtain current state
5: // online training of goal network
6: **while** $(E_g(t) > T_g$ & $cyc < N_g)$ **do**
7: $\omega_g(t) = \omega_g(t) + \Delta\omega_g(t)$ via (6) and (7); // update the weights recursively
8: $s(t) \leftarrow f_g(x_g(t), \omega_g(t))$;
9: $e_g(t) = V(t) - U_c, E_g(t) = \frac{1}{2}e_g^2(t)$;
10: **end while**
11: // online training of critic network
12: **while** $(E_c(t) > T_c$ & $cyc < N_c)$ **do**
13: $\omega_c(t) = \omega_c(t) + \Delta\omega_c(t)$ via (9) and (10);
14: $s(t) \leftarrow f_g(x_g(t), \omega_g(t))$; $V(t) \leftarrow f_c(x_c(t), \omega_c(t))$;
15: $e_c(t) = \gamma V(t) - [V(t-1) - s(t-1)]$, $E_c(t) = \frac{1}{2}e_c^2(t)$;
16: **end while**
17: // online training of action network
18: **while** $(E_a(t) > T_a$ & $cyc < N_a)$ **do**
19: $\omega_a(t) = \omega_a(t) + \Delta\omega_a(t)$ via (11) and (12);
20: $a(t) \leftarrow f_a(x_a(t), \omega_a(t))$; $s(t) \leftarrow f_g(x_g(t), \omega_g(t))$; $V(t) \leftarrow f_c(x_c(t), \omega_c(t))$;
21: $e_a(t) = V(t) - U_c, E_a(t) = \frac{1}{2}e_a^2(t)$;
22: **end while**
23: $\omega_a(t+1) = \omega_a(t)$;
24: $\omega_g(t+1) = \omega_g(t)$;
25: $\omega_c(t+1) = \omega_c(t)$; // update weights through each trial
26: **end for**

IV. Results and Discussions

We consider a triple-link inverted pendulum case here. As seen in Fig. 4, this pendulum includes three poles connected by three links. This model is highly unstable and exhibits non-negligible nonlinearities. Therefore, this benchmark is frequently used to evaluate the performance of new control strategies. The system model and system parameters are identical as those in [18]. In this task, our goal is to balance the inverted pendulum under the following constraints: (1) the cart track should be within 1.0 m to both sides from the center point; and (2) each link angle should be within the range of $[-20°, 20°]$ with respect to the vertical axis. For these two conditions, if either one fails or both fail, we consider that the current controller fails to accomplish the task.

In our study, the triple-link inverted pendulum is the environment and the designed controller, which produces the control action, is the agent. The control unit $a(t)$ (in voltage) is converted into force by an analog amplifier (with gain $K_s = 24.7125N/V$) to the DC servo motor. Each link here only rotates in a vertical plane. At each time step t, the controller receives an eight-dimension vector which represents the state, i.e., the position of the cart on the track (x), the vertical angles of three links $(\theta_1, \theta_2, \theta_3)$, the cart velocity $(\dot{x})$, the angular velocities of three links $(\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3)$. On this basis, the controller chooses an action. Note that the sign and the number of the action denote direction and magnitude of the force, respectively. Then one time step later, as a result of the action, the controller receives a new state vector and estimates the internal reward signal according to the current state and the produced action. Therefore, during this training process, there is no external reward signal transmitting from the environment to the agent.

To test the performance of the self-learning ADP approach, one hundred runs were conducted. We set the initial cart position at the center of the track and its velocity to be zero. For different runs, the initial values of three angles and angular velocity of the triple links were chosen randomly within the range of $[-1°, 1°]$ and $[-0.5, 0.5]$ rad/s, respectively. The maximum number of trials in each run was 3000. A run is considered successful if the controller can balance the triple-link inverted

pendulum within the assigned number of trials. The structures of goal, critic, and action networks established for this case refer to Fig. 2 with the number of neurons in each layer as 9-14-1, (i.e., the neural network has 9 input nodes, 14 hidden nodes, and 1 output node), 10-16-1, and 8-14-1, respectively. Since the optimal equilibrium of all states are near zero, we define the mathematical representation of the ultimate goal, in this example, as $U_c = 0$.

Our simulation demonstrated that 92% of the runs resulted in a successful balance. The average number of trials to success was 1071.6. Table 1 shows the comparative results of our method with respect to those reported in [18] and [23].

Table 1 shows that the performance of the proposed self-learning ADP method is not as good as the performance of the existing ADP methods with pre-defined reward signal. This is because the agent in our approach needs to learn what the reward signal is. However, the key observation from this research is that, the learning process of the proposed method can be accomplished without the explicit external reward directly from the environment. Instead, this reward will be automatically and adaptively learned and developed by the goal network according to the ultimate goal. This is the key fundamental contribution of this article.

To further examine the performance of the self-learning ADP method, a typical trajectory for each of the state variables for the task is shown in Fig. 5, including (a) the position of the cart on the track; (b)–(d) the vertical angle of the 1st, 2nd and 3rd links of the pendulum, respectively; (e) the velocity of the cart; and (f)–(h) the angular velocity of the 1st, 2nd and 3rd links of the pendulum, respectively. We observe that the cart position on the track and all the joint angles of the links are balanced within a small range of the balance point. This indicates that the proposed method can effectively control the system to achieve desired performance and the controller can estimate the effect of an action during the learning process automatically.

The learning process of the proposed method can be accomplished without the explicit external reward directly from the environment. Instead, this reward will be automatically and adaptively learned and developed by the goal network according to the ultimate goal.

V. Summary and Conclusion

We have designed a self-learning method without the explicit external reward signal directly given by the environment. Comparing with the traditional RL/ADP methods in the literature, the key contribution of our approach is that we introduce a new goal network to automatically and adaptively develop an internal reward signal based on the ultimate goal to facilitate the self-learning process. Therefore, instead of receiving an explicit reward signal directly from the external environment, the agent itself can learn an internal reward signal $s(t)$ by the goal network according to the ultimate goal U_c and guide itself to accomplish the task. This also means in our approach, only two interaction elements, state $x(t)$ and action $a(t)$, are required at each time step during the learning process. From simulations, we observe that the success rate of the designed self-learning method is lower than that of the traditional ADP methods. This is because the agent in the proposed method needs to learn how to

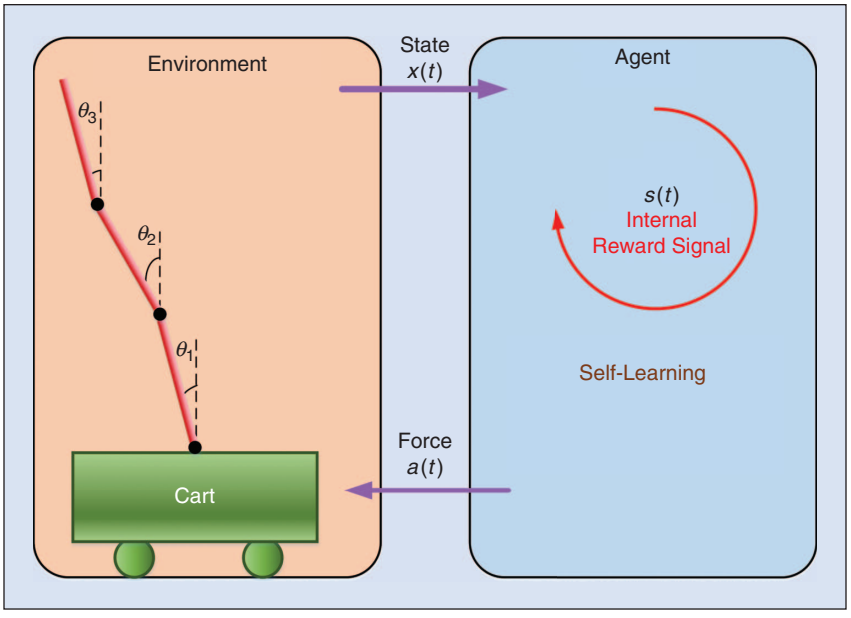

FIGURE 4 Triple-link inverted pendulum case and its interaction with the agent.

TABLE 1 Comparison with existing ADP learning algorithms.

	SELF-LEARNING ADP	TRADITIONAL ADP [18]	GOAL REPRESENTATION ADP [23]
SUCCESS RATE	92%	97%	99%
NUMBER OF TRAILS	1071.6	1194	571.4
NEED EXTERNAL REWARDS	NO	YES	YES

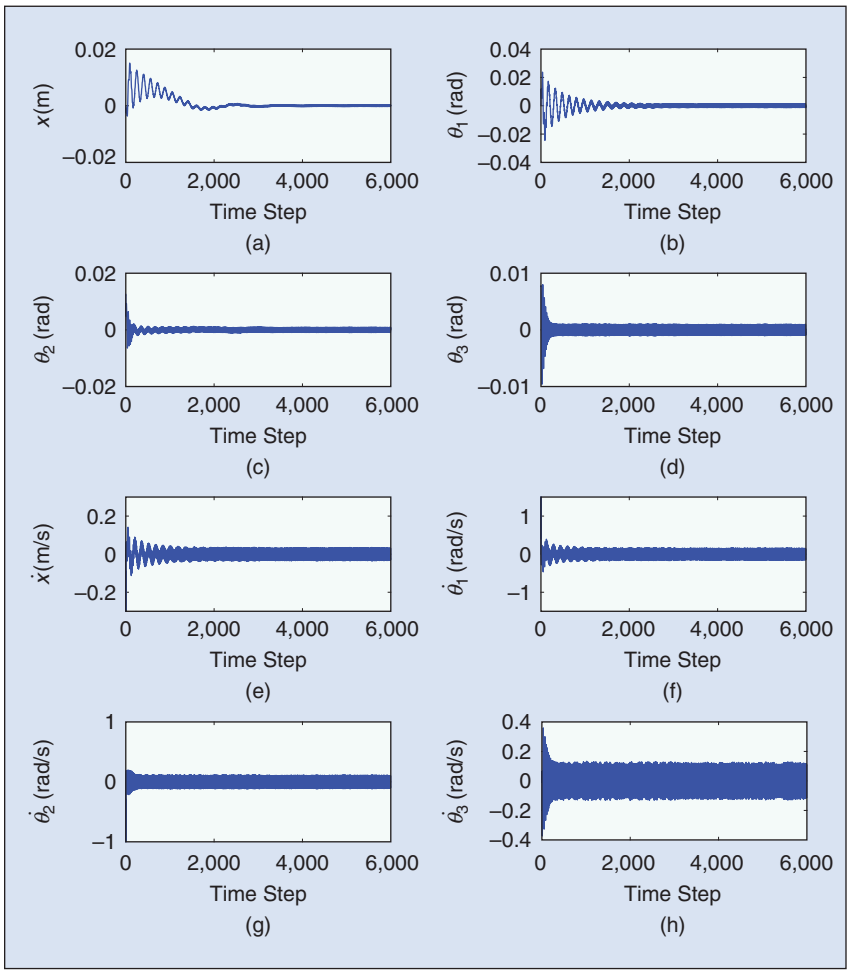

FIGURE 5 Typical trajectory of a successful trial on the triple-link inverted pendulum balancing task during the learning process: (a) the position of the cart, (b) the vertical angle of the 1st link joint to the cart, (c) the vertical angle of the 2nd link joint to the 1st link, (d) the vertical angle of the 3rd link joint to the 2nd link, (e) the cart velocity, (f) the angular velocity of the 1st link joint to the cart, (g) the angular velocity of the 2nd link joint to the 1st link, (h) the angular velocity of the 3rd link joint to the 2nd link.

represent the reward signals by himself based on the ultimate goal, rather than is given explicitly by an external teacher in the existing methods. The implications of self-learning with no external supervision available at each time step during the learning process is significant, resulting in potential far-reaching applications across a wide range of fields.

Acknowledgment

This work was supported in part by the National Science Foundation (NSF) under CMMI 1526835 and ECCS 1731672.

References

[1] A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] R. A. Brooks, "Intelligence without reason," in *Proc. 12th Int. Joint Conf. Artificial Intelligence*, Sydney, NSW, Australia, Aug. 1991, pp. 569–595.

[3] R. Pfeifer and C. Scheier, *Understanding Intelligence*. Cambridge, MA: MIT Press, 1999.

[4] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 39–47, Apr. 2009.

[5] B. Bathellier, S. P. Tee, C. Hrovat, and S. Rumpel, "A multiplicative reinforcement learning model capturing learning dynamics and interindividual variability in mice," *Proc. Natl. Acad. Sci. USA*, vol. 110, no. 49, pp. 19950–19955, Nov. 2013.

[6] P. W. Glimcher, "Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis," *Proc. Natl. Acad. Sci. USA*, vol. 108, no. Suppl 3, pp. 15647–15654, Mar. 2011.

[7] D. Zhao, Z. Xia, and Q. Zhang, "Model-free optimal control based intelligent cruise control with hardware-in-the-loop demonstration [research frontier]," *IEEE Comput. Intell. Mag.*, vol. 12, no. 2, pp. 56–69, Apr. 2017.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, T. G. G. van den Driessche, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[11] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995.

[12] P. J. Werbos, "Intelligence in the brain: A theory of how it works and how to build it," *Neural Netw.*, vol. 22, no. 3, pp. 200–212, Apr. 2009.

[13] P. J. Werbos, "ADP: The key direction for future research in intelligent control and understanding brain intelligence," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 4, pp. 898–900, Aug. 2008.

[14] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Syst. Mag.*, vol. 32, no. 6, pp. 76–105, Dec. 2012.

[15] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York: Wiley, 2007.

[16] D.V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 997–1007, Sept. 1997.

[17] F. Liu, J. Sun, J. Si, W. Guo, and S. Mei, "A boundedness result for the direct heuristic dynamic programming," *Neural Netw.*, vol. 32, pp. 229–235, Aug. 2012.

[18] J. Si and Y.-T. Wang, "Online learning control by association and reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 264–276, Mar. 2001.

[19] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear HJB solution using approximate dynamic programming: Convergence proof," *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 4, pp. 943–949, June 2008.

[20] F. L. Lewis and K. G. Vamvoudakis, "Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using measured output data," *IEEE Trans. Syst., Man, Cybern. A*, vol. 41, no. 1, pp. 14–25, Feb. 2011.

[21] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming," *IEEE Trans. Autom. Sci. Eng. (from July 2004)*, vol. 9, no. 3, pp. 628–634, July 2012.

[22] X. Zhong, H. He, H. Zhang, and Z. Wang, "Optimal control for unknown discrete-time nonlinear Markov jump systems using adaptive dynamic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2141–2155, Mar. 2014.

[23] H. He, Z. Ni, and J. Fu, "A three-network architecture for on-line learning and optimization based on adaptive dynamic programming," *Neurocomputing*, vol. 78, no. 1, pp. 3–13, Feb. 2012.

[24] Z. Ni, H. He, J. Wen, and X. Xu, "Goal representation heuristic dynamic programming on maze navigation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, pp. 2038–2050, Dec. 2013.

[25] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. 17th Int. Conf. Machine Learning*, San Francisco, CA, July 2000, pp. 663–670.

[26] M. Rolf and M. Asada, "Where do goals come from? A generic approach to autonomous goal-system development," arXiv Preprint, arXiv:1410.5557, Oct. 2014.

[27] X. Zhong, H. He, D. Wang, and Z. Ni, "Model-free adaptive control for unknown nonlinear zero-sum differential game," *IEEE Trans. Cybern.*, vol. 48, no. 5, pp. 1633–1646, May 2018.

[28] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.