Hierarchical Deep Reinforcement Learning for Continuous Action Control

Author:
Yang, Z; Merrick, K; Abbass, H

Publication details:
IEEE Transactions on Neural Networks and Learning Systems
v. 29
Chapter No. 11
Medium: Print-Electronic
pp. 5174 - 5184
2162-237X (ISSN); 2162-2388 (ISSN)

Publication Date:
2018

Publisher DOI:
https://doi.org/10.1109/TNNLS.2018.2805379

Hierarchical Deep Reinforcement Learning for Continuous Action Control

Zhaoyang Yang, Kathryn Merrick, *Senior Member, IEEE*, Hussein A. Abbass, *Senior Member, IEEE*, and Lianwen Jin, *Member, IEEE*

***Abstract*—Robotic control in a continuous action space has long been a challenging topic. This is especially true when controlling robots to solve compound tasks, as both basic skills and compound skills need to be learned. In this paper, we propose a hierarchical deep reinforcement learning algorithm to learn basic skills and compound skills simultaneously. In the proposed algorithm, compound skills and basic skills are learned by two levels of hierarchy. In the first level of hierarchy, each basic skill is handled by its own actor, overseen by a shared basic critic. Then in the second level of hierarchy, compound skills are learned by a meta critic by reusing basic skills. The proposed algorithm was evaluated on a Pioneer 3AT robot in three different navigation scenarios with fully observable tasks. The simulations were built in Gazebo 2 in a ROS Indigo environment. The results show that the proposed algorithm can learn both high performance basic skills and compound skills through the same learning process. The compound skills learned outperform those learned by a discrete action space deep reinforcement learning algorithm.**

***Index Terms*—Continuous control, deep learning, hierarchical learning, reinforcement learning**

I. Introduction

REINFORCEMENT learning [1] is a kind of algorithm that permits an agent or robot to learn from trial-and-error and reward during interaction with its environment.

Classical TD-learning [2] algorithms such as Q-learning [3], and SARSA [4] are well-known reinforcement learning algorithms that can learn by trial-and-error and even when the reward feedback is infrequent or delayed until the end of a learning episode. What is more, many works introduced linear function approximations to enhance the generalization ability of algorithms [5, 6] to handle more complex environments.

Recently, deep learning [7] algorithms have achieved record-breaking performance in several applications and research topics, such as computer vision [8, 9], semantic analysis [10, 11] and others. With hundreds of thousands of auto-learned parameters in the model, deep neural networks have shown unprecedented feature extraction and generalization capabilities. Available choices for network architectures such as convolutional neural networks (CNN) [12] and long-short term memory (LSTM) networks [13] further help deep learning in applications with different requirements, such as applications that require considering past states to make decisions and applications that only have access to image data. These successes inspired interest in combining reinforcement learning with deep learning to further improve the performance of the agents.

Although it is generally believed that non-linear approximations like deep neural networks are not suitable for reinforcement learning because of the correlations between data and the possible sparsity of supervision signals in reinforcement learning scenarios [14], recent advances in reinforcement learning have addressed these challenges and brought deep reinforcement learning great success. Some deep reinforcement learning agents have recently outperformed humans in playing Atari games [15] and Go games [16].

However, unlike games or other decision making processes that contain only a limited number of legal actions, robotic control usually involves action choices in a continuous action space. Moreover, learning agents also need to consider many physical factors to keep the robot moving smoothly. The problem becomes even more difficult when trying to solve compound tasks where the agent needs to learn both basic skills and compound skills at the same time.

In this paper, in order to address the challenges mentioned above, we propose a novel hierarchical deep reinforcement learning algorithm based on the Deep Deterministic Gradient Descent algorithm [17]. The proposed algorithm makes use of observations from both sensor data and a first-person view camera images to learn basic skills and compound skills simultaneously. We call this algorithm h-DDPG.

The proposed algorithm comprises two levels of hierarchy. In the first level of hierarchy, multiple basic skills, each handled by its own actor, are learned simultaneously. This is achieved by adapting a multi-task deep reinforcement learning algorithm we developed in previous work [18]. Multi-layer perceptron convolutional (mlpconv) layers [19] are used in this hierarchy to reduce the number of parameters needed for learning

Manuscript received for review on May 26, 2017. This work was supported by the Australian Research Council under Grant DP160102037.

Z. Yang is with the School of Engineering and Information Technology, University of New South Wales, Canberra, Australia, and also with the College of Electronic and Information Engineering, South China University of Technology, Guangzhou, China (e-mail: yangzhaoyang6@126.com).

K. Merrick and H. Abbass are with the School of Engineering and Information Technology, University of New South Wales, Canberra, Australia (e-mail: K.Merrick@adfa.edu.au and h.abbass@adfa.edu.au).

L.Jin is with the School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China (e-mail: lianwen.jin@gmail.com).

multiple skills. The second level of hierarchy is responsible for learning compound skills. These compound skills are made up of a series of basic skills learned in the first level of hierarchy.

To test the proposed h-DDPG algorithm, we built three different scenarios in Gazebo 2 in a ROS Indigo environment. Each scenario has a different compound task that the robot will try to solve. Our simulations were conducted with a Pioneer 3AT robot with range sensors and a camera set on the front of it. Results show that the proposed algorithm can successfully learn high performance basic skills and compound skills simultaneously. Results also show that its performance in solving compound tasks in the three scenarios outperform discrete action deep reinforcement learning algorithm.

In summary, the main contribution of this paper is a hierarchical deep reinforcement learning algorithm that can:

1) Learn multiple basic skills at the same time,
2) Learn compound skills to solve compound tasks by reusing basic skills,
3) Handle the above two kinds of skill learning within the same process.

The rest of the paper is organized as follows. A brief literature review and necessary background will first be presented in the next section. Then in Section III, we will describe the structure of the proposed h-DDPG algorithm in detail, followed by the details of its learning process in Section IV. Simulation settings and experimental results will be reported in Section V. Section VI is the conclusion.

II. Preliminaries

A. Related Work

Deep reinforcement learning has attracted considerable attention in recent years due to its potential to learn highly generalized representations in complex environments. A first breakthrough was the deep Q network (DQN) algorithm [14]. Many works [20-22] emerged after DQN's success in achieving human level performance in playing Atari games.

Various work has also been done on solving continuous control tasks with deep reinforcement learning. The actor-critic architecture [23] is generally chosen as a baseline to build deep reinforcement learning algorithms with deterministic policy gradients [17, 24] or stochastic policy gradients [25]. Moreover, Trust Region Policy Optimization (TRPO) [26] achieved continuous control in a similar way to natural policy gradient. Some work has also been done to integrate model-based methods to accelerate learning in continuous action spaces [27]. However, while all these works achieved very good performance in learning basic locomotion skills, few of them can solve compound tasks efficiently [28]. Although some experiments show that algorithms can solve some compound tasks [24], this is mainly owing to the use of a multi-threaded parallel learning scheme that makes the exploration more balanced and increases the chance of collecting rewards during exploration. In this paper, we focus on the case where only one robot is available.

Various work on hierarchical reinforcement learning exists [29][30], including work considering deep architectures [31] [32] . However, different from our work, work in [31] focuses on discrete action spaces with embedded DQN structure. Work in [32] is focuses on finding the best hierarchical structure of the tasks with clustering methods, and is mainly concerned with how to decompose tasks.

Most recent work on deep hierarchical reinforcement learning can be found in [33] and [34]. Work in [33] extends policy gradient methods to the option framework [35], which allows auto-decomposition of tasks in the forms of options. Work in [34] used feudal reinforcement learning methods to further improve the performance of top level hierarchy (which they called the Manager) in controlling lower level hierarchy (which they called the Worker).

Different from our work, these two works can learn a hierarchical agent without giving additional reward functions for sub-goals and can fit in learning with different base deep reinforcement learning algorithms. However, compared to our work, these two works are learning in two time scales, which means the low-level hierarchy should take control for a certain period. This is different from our work as both levels of hierarchy in our proposed algorithm learn in the same time scale, which allows more instant control of low-level actors. What is more, although our method needs explicitly defined rewards for both levels of hierarchy, the basic tasks learned at the lowest-level of hierarchy are general basic movement skills, are non-task specific, and can potentially be transferred to any other compound tasks.

Another related topic is intrinsically motivated learning which also involves reuse of basic skills [36]. Some recent work [37] has successfully achieved an intrinsically motivated agent by replacing reward functions with a maximization of the mutual information during learning.

B. Background

In this paper, we consider a standard reinforcement learning setup, where the agent is interacting with the environment $\boldsymbol{E}$ in discrete timesteps. In each timestep t, the agent receives a state $s_t \in \boldsymbol{S}$ from the environment, and chooses and executes an action $a_t \in \boldsymbol{A}$ according to the current policy $\pi: \boldsymbol{S} \rightarrow \boldsymbol{A}$. Then the agent will receive a reward r_t for taking a_t and transition to the next state s_{t+1}, where the process starts again.

The goal of reinforcement learning is to learn a policy π that can maximize reward. This can be achieved by maximizing the expected future return for each timestep. The expected future return is defined as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

where T is the total number of timesteps taken and $\gamma \in [0,1]$ is the discounted factor that indicates to what extent future rewards are being considered.

Note that the policy π may be stochastic in some cases. However, we are considering deterministic policies, where action a_t only depends on s_t and $\pi(s_t)$. Also, we assume that all environments in this paper are fully observable, which

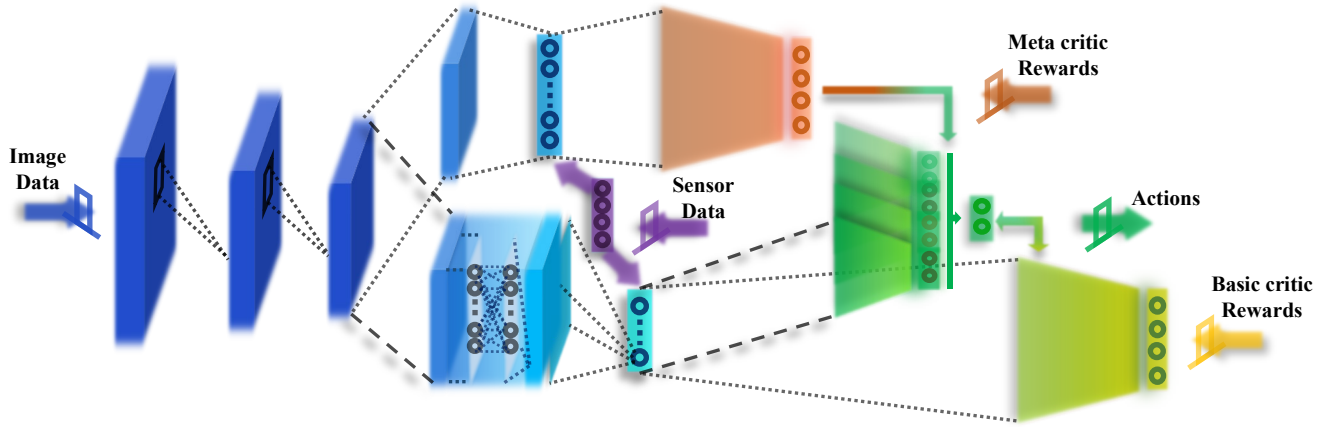

Fig. 1. An overview of h-DDPG architecture. The trapeziums represent fully connected layers. These fully connected layers are layers from meta critic, actors and basic critic respectively from top to bottom. The square-dotted lines are connections with back-propagation while the dashed lines are not.

means s_t can fully describe the current state.

The state-action values (also known as Q values) are central to reinforcement learning algorithms. They are estimations of the expected future returns for given state-action pairs:

$$Q^{\pi}(s_t, a_t) = E[R_t|s_t, a_t] \tag{2}$$

Deep reinforcement learning algorithms aim to use deep neural network (θ^Q) to approximate this value function. This was not achieved until two major techniques, replay memory and target network ($\theta^{Q'}$), were applied in DQN [14].

In order to extend deep reinforcement learning to continuous action spaces, an algorithm called DDPG [17] was proposed, which has an actor-critic architecture. Unlike value function based algorithms such as DQN, DDPG uses two separate networks to approximate the critic (value) function (θ^Q) and actor (policy) function (θ^{π}) (Each network also has its own target network $\theta^{Q'}$ and $\theta^{\pi'}$). With the additional actor function, DDPG can learn more sophisticated policies to handle continuous action spaces. For the critic network, the parameters are optimized by minimizing the loss:

$$L(\theta^Q) = (Q(s_t, a_t|\theta^Q) - y_t)^2 \tag{3}$$

where

$$y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta^{\pi'})|\theta^{Q'}) \tag{4}$$

and the update equation is

$$\theta^Q \leftarrow \theta^Q - \mu_Q \cdot \nabla_{\theta^Q} L(\theta^Q) \tag{5}$$

where μ_Q is the learning rate and the symbol ∇ donates a gradient calculation.

After the critic network has been updated, the actor network will be updated by inferring gradients from the critic network:

$$\theta^{\pi} \leftarrow \theta^{\pi} - \mu_{\pi} \cdot \nabla_a Q(s_t, , \pi(s_t|\theta^{\pi})|\theta^Q) \cdot \nabla_{\theta^{\pi}} \pi(s_t|\theta^{\pi}) \tag{6}$$

where μ_{π} is the learning rate. A full derivation of (6) can be found in [38].

The mlpconv layer is a new network layer proposed in [19]. In traditional convolutional layers, feature maps are activated with an activation function $F(\cdot)$ as:

$$f_{i,j,k} = F(\omega_k^T x_{i,j} + b_k) \tag{7}$$

where (i, j) indexes the feature pixels in channel k and ω_k and b_k are the weight and bias of the convolutional kernel of channel k.

Generally, good abstractions are highly non-linear functions of the input. While activation functions such as ReLU can add some non-linearity to the network, the network may still not be expressive enough. As a result, we need to make the network deeper or wider. However, this may make the network hard to train, both in a computational perspective and in a gradient propagation perspective. Therefore, in order to enhance the non-linearity of traditional convolutional layers, mlpconv layer forms a recombination of features across different channels with a multi-perceptron nature. Then the recombined feature outputs of the mlpconv layer becomes:

$$f_{i,j,k_n}^n = F\left(\omega_{k_n}^{n\,T} f_{i,j}^{n-1} + b_{k_n}\right) \tag{8}$$

Where n is the number of perceptron layers used and $f_{i,j}^0 = x_{i,j}$. For both the original paper and this paper $n = 2$.

For the classification problems discussed in [19], the final output can be obtained by applying a global average pooling on the output of the last mlpconv layer. This saves the parameters needed for fully-connected layers in traditional CNN.

In the proposed h-DDPG algorithm, we will use a mlpconv layer and global average pooling to generate abstractions from image data to reduce the number of parameters needed for multiple basic skills in the first hierarchy of the algorithm.

III. H-DDPG Architecture

In this paper, we propose a novel hierarchical deep reinforcement learning algorithm that can learn compound skills by reusing basic skills it learns during the same process. We define basic tasks to be tasks that can be achieved by choosing actions in a single 'pattern' learned in response to a basic reward function. Basic tasks are fundamentally non task-specific and rather are tied to the physical capabilities of the robot, such as rotating a wheel or bending a joint. Let P_g be the pattern (action set) of the basic task g. Then at any time t during execution of a basic task:

$$a_t \in P_g \tag{9}$$

Conversely, compound tasks are defined to be tasks that can only be achieved by combining different patterns. So here, the chosen actions can be an action from any pattern in a basic skill set:

$$a_t \in \{P_1, P_2, \dots, P_G\} \quad (10)$$

where G is the number of basic skills the agent has. These basic skills are combined to achieve a compound task.

The proposed algorithm has two levels of hierarchy, corresponding to compound and basic skills, which is achieved with a duel-critic, multi-actor architecture. The duel-critic comprises a basic critic in the first level of hierarchy, which is responsible for training multiple actors that learn different basic skills, and a meta critic in the second level of hierarchy that learns to reuse actors to solve compound tasks. An overview of the proposed h-DDPG algorithm is shown in Fig. 1. In the next few sections, we will use subscript letters b and m to distinguish basic critic components and meta critic components in equations.

A. Basic Critic

In order to achieve a level of hierarchy that learns multiple skills simultaneously, we need to expand the original actor-critic architecture [17, 23] to a single critic, multi-actor architecture in this level of hierarchy. We achieve this by adapting a multi-task deep reinforcement learning algorithm we developed in previous work [18]. Specifically, we kept the basic concepts of the algorithm in [18] and made some changes on its network architecture to fit it in the proposed h-DDPG algorithm.

As a result, in this level of hierarchy, we will have a multi-actor network, with each network learning a different skill. One more benefit we can receive from adapting this multi-task algorithm [18] is that it ensures that the update of different actors are independent from each other. This ensures that the learning of basic skills with different requirements and movement patterns do not interfere each other. We also demonstrate in Section V that this protects high performance actors from being impacted by poor performing actors.

B. Meta Critic

Different from the basic critic, the meta critic focuses on learning compound skills to solve compound tasks. The meta critic can access a set of basic skills provided by actors in the first level of hierarchy. The goal of the meta critic is to choose a basic skill that will help it solve the attempted compound tasks in each timestep. Therefore, similar to discrete action scenarios, the meta critic will choose the basic skill with the highest value from a given set of basic skills.

This can be achieved by bootstrapping estimation of Q values of each basic skill. Thus, the meta critic will be a network with G output neurons that give estimations of Q values of all available actors. Then the network will be optimized by minimizing the loss function:

$$L(\theta^{Q_m}) = (Q_m(s_t, \pi_g(s_t|\theta^{\pi_g})|\theta^{Q_m}) - y_{m,t})^2 \quad (11)$$

where the supervision signal is:

$$y_{m,t} = r_{m,t} + \gamma_m \max_{\pi_g} Q_m(s_{t+1}, \pi_g(s_{t+1}|\theta^{\pi_g})|\theta^{Q'_m}) \quad (12)$$

where π_g is the policy of the basic skill g.

Note that π_g is dynamic, as it will change as learning goes on. Also note that as learning of the basic critic and actors happens in the same process as the meta critic learns, there is no guarantee that all actors have good performance at their corresponding basic skills. However, the way that meta critic understands the actors is to understand the transitions from s_t to s_{t+1} after a particular actor has been chosen. The meta critic does not know what basic skills the actors are assigned to learn prior to the training starting. So as training goes on, actors that are providing poor actions to the compound task will rarely be chosen by the meta critic.

C. Hierarchies of Abstractions

In the proposed algorithm, we include sensor data after the convolutional part of the networks and concatenate it with a feature vector of the image data. In addition, we extract two levels of image feature abstractions to keep the whole hierarchical architecture consistent and concise.

Specifically, for the meta critic network that needs a thorough understanding of the environment to infer proper choice of basic skills to solve compound tasks, image features are a long vector flattened from the feature maps of the last convolutional layer. This vector is then fed into the rest of the fully-connected layers. In this way, every pixel in the feature maps of the last convolutional layer will contribute to the final decision of the meta critic. Abstractions at this level of hierarchy are intended to give a more detailed description of the states so that the critic could learn and make decisions according to full observations of the environment.

For the basic critic as well as the actors that focus on basic skill learning, image features are a much shorter vector rendered from a global max pooling. This is achieved by applying mlpconv layer operations on the feature maps of the last convolutional layer. Each reconstructed feature map from the mlpconv layer will then be averaged globally to form an element in the abstraction vector. Abstractions at this level of hierarchy are intended to give a less detailed description of the states so that the critic can learn basic skills more easily.

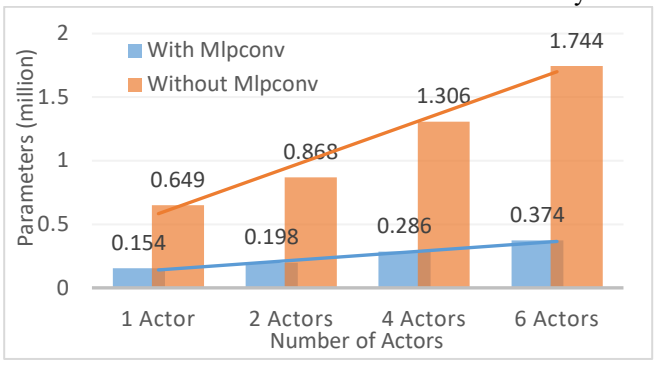

Fig. 2. Comparison of parameters needed with or without mlpconv layers.

Moreover, the implementation of mlpconv layer in the basic critic helps reduce the number of parameters needed significantly compared with using a traditional convolutional layer. As shown in Fig. 2, the proposed critic architecture reduces the number of parameters by 75% for a single actor. The reduction will become more significant as the number of actors increases.

IV. H-DDPG LEARNING

The whole learning process of the proposed h-DDPG algorithm follows an ordering of hierarchy priority. In addition, the co-existence of two levels of hierarchy in the same architecture demands adjustments in several aspects of the algorithm.

A. Rewards and Punishments

In this paper, we are considering scenarios where all reward functions are pre-defined. Specifically, two kinds of rewards are necessary: reward for the meta critic, which can only be received when the final compound tasks are achieved, and reward for the basic critic, which can be received as soon as the action chosen by the actor is achieving its corresponding basic skill, that is, satisfying (9). As the frequency of receiving these two kinds of rewards is different, different values may be chosen. Having suitable reward values for the meta critic is especially important as it is much more sparse than the one for the basic critic. We investigate this in Section V.

Note that learning with the multi-task learning architecture [18] in the first level of hierarchy requires a reward vector in which each element represents whether the action is what is desired for the corresponding actor. This means that whichever actor is providing the action, the action will always be evaluated by reward functions of all available actors. As a result, for every action taken, the agent will receive a reward for the meta critic and a reward vector for the basic critic.

Similar to the rewards, two kinds of punishments, one for the meta critic and one for the basic critic, are introduced in the algorithm. Only the component that has caused undesired actions will receive punishments. In our case, in order to make the basic critic and actors focus on learning basic skills, we only punish the basic critic when the robot turns over, as making the robot move stably is a prerequisite of having good basic skills. Note that actions that could cause the robot to turn over are undesired actions for all actors, regardless of what basic skill the actor is assigned to learn. Therefore, the punishment is universal to all actors. The meta critic will receive a punishment when the robot crashes into obstacles as avoiding collision is considered as a part of the compound skills. What is more, we punish the meta critic with a small value every step before the episode terminates. This is mainly to push the meta critic to find the shortest solution to the task. We use this reward and punishment structure for all experiments in this paper.

B. Replay Memory and Batch Sampling

As a consolidated system, all levels of hierarchy in this algorithm share a single replay memory. However, we stored a label in each transition to record which actor made it. This is because when sampling transitions from the replay memory, a balance among transitions made by different actors is required. Specifically, the final batch of transitions will always consist of the same number of transitions from different actors. This sampling strategy makes the sampling more controllable and ensures that the critics can see transitions of different actors evenly.

Note that as required by the multi-task learning architecture [18] in the first level of hierarchy, when calculating supervision signals for the basic critic, only one target actor network will be used. In this work, we select target networks iteratively. This is feasible owing to the actor-unspecific nature of the basic critic during self-updating as explained in the original work [18].

C. Exploration

Exploration is a critical aspect of all reinforcement learning algorithms. For deep reinforcement learning, exploration needs to be balanced to prevent the agent getting stuck in a local optimum.

In the proposed algorithm, the exploration of the meta critic is governed by a $\epsilon - greedy$ policy while the exploration of the basic critic is governed by an Ornstein-Uhlenbeck process [39]. What is more, the value of ϵ will be annealed throughout the training process to allow more exploitation. Similarly, we change the intensity of the Ornstein-Uhlenbeck process according to the performance of actors' intermediate training according to:

$$\sigma_g \leftarrow \max\{\sigma_{min}, (1-p_g)\sigma_{init}\} \tag{13}$$

Where σ is the parameter that controls the intensity of the process and subscript $init$ and min denotes its initial and minimum value respectively. p_g is the latest measured performance of actor g, which is measured as the reward per action the actor can collect.

D. Learning Process

The learning process follows an ordering of hierarchy priority. The component that has a higher level of hierarchy will be updated prior to components with lower level of hierarchy. Each training iteration will start right after an action has been executed in a timestep of exploration. The whole algorithm is summarized in Algorithm 1.

Algorithm 1 H-DDPG

Input: maximum training episode E_{max}, maximum steps in each episode S_{max}, mini-batch size M, replay memory **P.**
Initialization: randomly initialize networks weights θ^{Q_b}, θ^{Q_m}, θ^{π_1}, ... , θ^{π_G} and target networks weights $\theta^{Q_b'} \leftarrow \theta^{Q_b}$, $\theta^{Q_m'} \leftarrow \theta^{Q_m}$, $\theta^{\pi_g'} \leftarrow \theta^{\pi_g}$.
  while episode < E_{max}
    Initialize random noise *N* for exploration
    Get initial state s_1
    while step <S_{max} **and** episode **not** terminated
      Get Q values of actors using meta critic
      Select actor i according to ϵ − greedy policy
      Select action a_t using selected actor and add *N*
      Execute a_t and get rewards $r_{b,t}$, $r_{m,t}$ and next state s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in **P**
      Randomly sample $\frac{M}{G}$ transitions of each actor from **P** and make up a mini-batch with M transitions
      Update θ^{Q_m} according to (11) and (12) and update $\theta^{Q_m'}$
      Update θ^{Q_b} and $\theta^{Q_b'}$ according to [18]
      for g **in** G:
        Update θ^{π_g} and $\theta^{\pi_g'}$ according to [18]
    end
  end

All target network updates in the algorithm follow a soft update process. With a soft update factor φ, the soft update process can be expressed as:

$$\theta'_{t+1} \leftarrow (1-\varphi)\theta'_t + \varphi\theta_t \tag{14}$$

Note that, we also applied a parameter-sharing scheme on

convolutional layers across different networks. Specifically, all

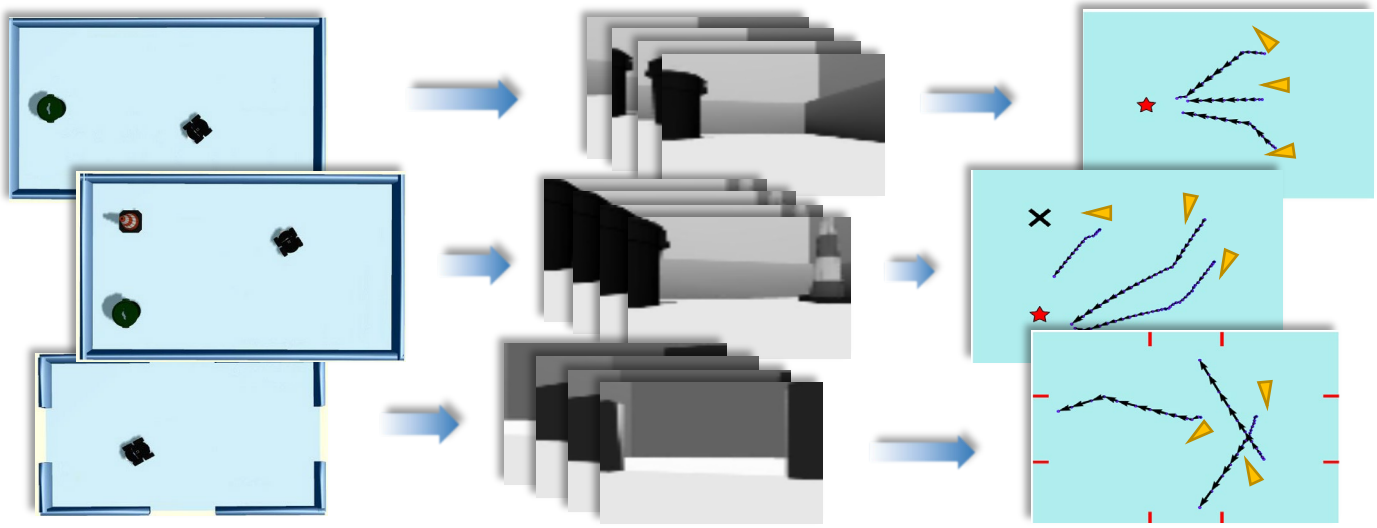

Fig. 3. An introduction of the three simulation scenarios. They are, from top to bottom, the approaching object scenario (Scenario 1), the approaching specific target scenario (Scenario 2) and the doorway scenario (Scenario 3). Pictures from left to right in each row are top-down views of scenarios, image captures of the camera and the samples of solutions made by the agent. In the samples of solution, the yellow triangles indicate initial orientation, and stars indicate target objects.

traditional convolutional layers will be updated by the meta critic and fixed when updating other networks. This is inspired by previous work [40] that indicates that less complex tasks (in our case, the basic tasks) can learn from abstractions extracted by a more complex task (in our case, the compound tasks). Particularly in our application, this can also make training more efficient as the shared convolutional part will not be updated too frequently. We also implement a similar schedule as the annealing intensity of the Ornstein-Uhlenbeck process on the base learning rate of the actor networks. We use Adam [41] with l_2 regularization for updating networks.

V. Experiments and Results

A. Experiment Setup

We tested the proposed algorithm in simulations conducted in Gazebo 2 in a ROS Indigo environment. We built three scenarios with different tasks in a walled space. We have tried to make them as observable as possible when building them:

1) *Approaching Object*
   The walled space is mostly obstacle-free with only a single target object. The robot's task in this scenario is to approach the target object, starting from a random position and orientation, without crashing into walls.
2) *Approaching Specific Target*
   The walled space is filled with two different objects, one target and one decoy. The robot's task in this scenario is to approach the target object, starting from a random position and orientation, without confusing it with the decoy or crashing into walls. To achieve this, the robot has to distinguish between the objects and apply different strategies to them, which makes this task more difficult than the task in scenario 1.
3) *Doorway Escape*
   The walled space is obstacle-free with four doorways one on each side of the space. The robot's task in this scenario is to go through one of the doorways, starting from a random position and orientation, without crashing into walls. This task is even more difficult than tasks in scenario 1 and 2 as the robot has to avoid collision with the sides of a door when going through it.

Top-down screenshots of these three scenarios are shown in Fig. 3. In all three scenarios, the agent has to learn the same four basic skills by solving four basic tasks we assigned to it. The patterns of these four basic tasks are wheel rotations for going forward, going backward, turning left and turning right.

In each simulation, a single Pioneer 3AT robot is spawned in the environment. The robot is equipped with a camera on the front of it that will give it first-person vision of the environment. Moreover, range sensors give distance readings in four directions (front, back, left and right). We bundled the most recent four frames of the camera as well as the sensor data in the last frame to form a state. Frames captured by the camera will be converted to 64×64 grey scale images before being fed into networks. The action of the agent is to set the speed of the wheels on both sides of the robot, which are real values in a continuous space output by the actor networks. These actions are executed in continuous time space, which means the length of time an action will be executed depends on the processing time needed before the next action has been decided.

As described in Section IV, we applied a parameter sharing learning scheme among different networks in the algorithm. The shared convolutional part consists of three layers. The first layer has 32 kernels with size 8×8 and stride 4, followed by the second layer which has 64 kernels with size 4×4 and stride 2. The last convolutional layer has 64 kernels with size 3×3 and stride 1. For the meta critic, the fully connected layer part consists of two layers with 512 nodes and 256 nodes respectively, while for the basic critic, the two fully connected layers both consists of 300 nodes. Moreover, the basic critic includes a mlpconv layer operation before the fully-connected part and also includes the actions in its second fully-connected layer. All fully-connected parts of the actor networks consist of

two layers each, with 200 and 150 nodes respectively. In all

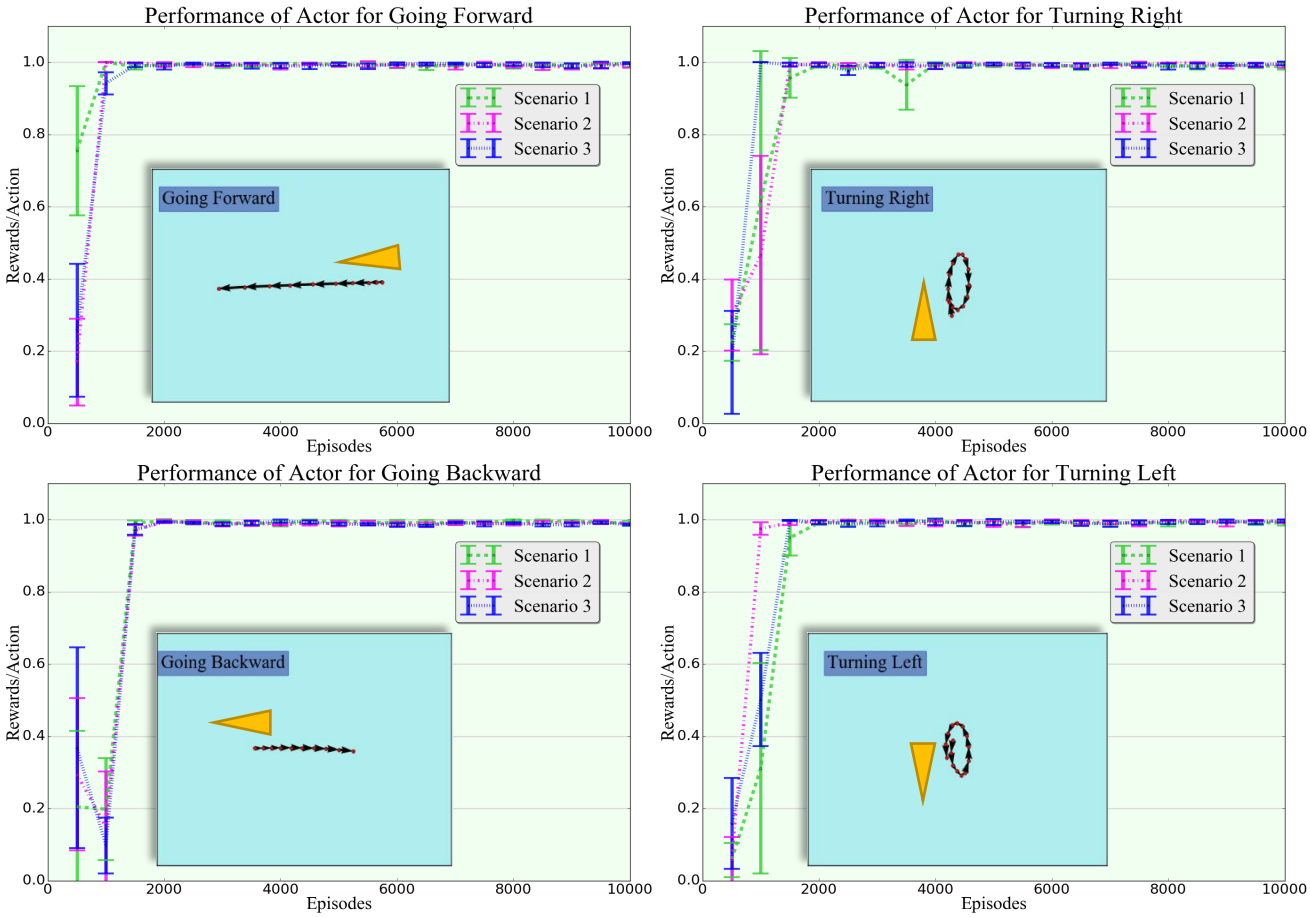

Fig. 4. The performance of each actor in the three scenarios. The pictures at the bottom-right of each curve are samples of moving trajectories of the actor. The yellow triangles indicate the initial orientation.

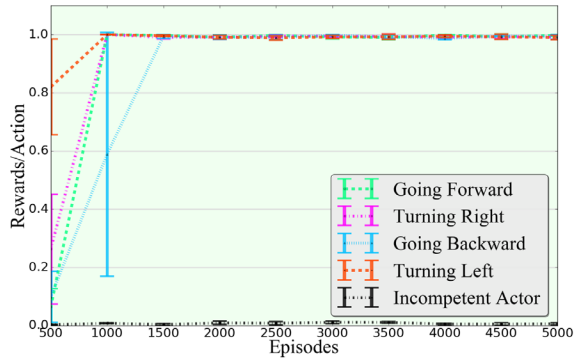

Fig. 5. Performance of actors in Scenario 1 when training with an incompetent actor.

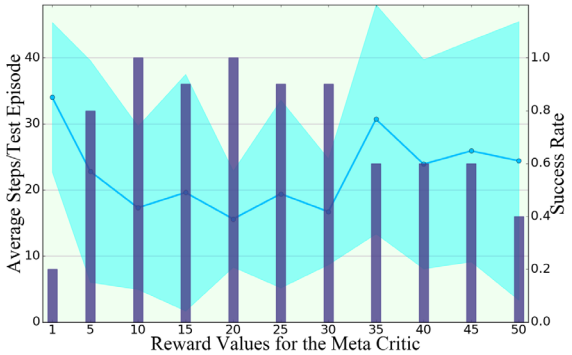

Fig. 6. Performance comparison at 5,000 episode between training with different meta critic reward values in Scenario 1.

networks, range sensor data is included before the first fully-connected layer. All the networks were built and trained in TensorFlow [42]. The value of other parameters introduced in the paper can be found in the Appendix.

In each experiment, agents were trained for 10,000 episodes, which consists of around 200,000 update iterations. We tested the performance of both the meta critic and the basic critic at several points during training. Each time we test the model, we run 10 independent episodes and initialize the robot at a random position and orientation. Then we calculated the success rate of the agent on solving the task in these 10 episodes.

We conducted experiments to test the performance of the basic and meta critic as well as to compare the proposed algorithm with other algorithms. The results are discussed below.

B. Basic Critic Performance

We first examined the performance of the basic critic by testing the performance of the actors during training. Results are shown in Fig. 4. We can see that all four actors achieved very good performance after training for around 1500 episodes. After the actors reached their best performance, the performance remains stable until the end of training. Samples

of movement trajectories of each actor shown in Fig. 4 also demonstrate that all actors are achieving good basic skills that

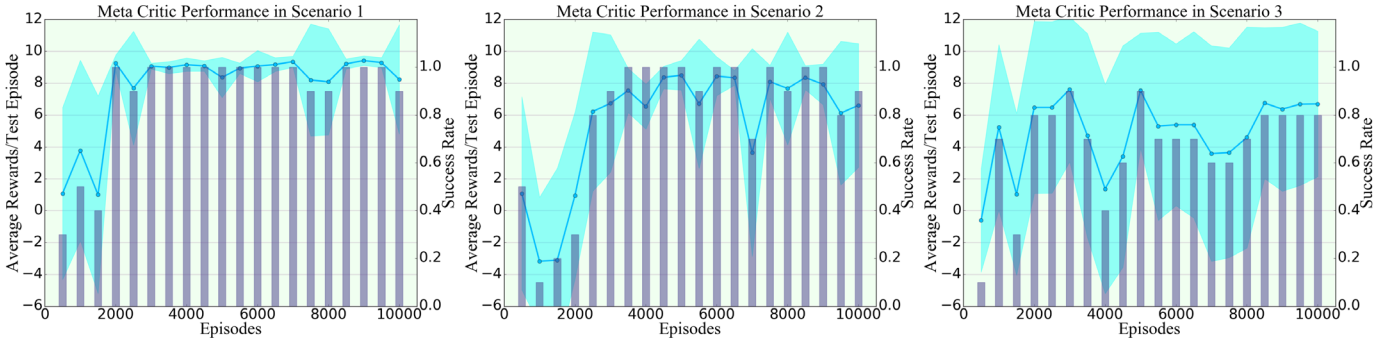

Fig. 7. The performance of the meta critic. In each curve, the bars are success rate of finishing the task. The curves are average rewards in each test episode and the shadows indicate the standard deviation.

TABLE I
COMPARISON BETWEEN BEST PERFORMANCES OF DIFFERENT ALGORITHMS

	Scenario 1		Scenario 2		Scenario 3	
	Average reward	Success rate	Average reward	Success rate	Average reward	Success rate
DQN [15]	5.34 ± 5.17	80%	3.91 ± 5.78	70%	2.03 ± 5.38	50%
DDPG [17]	-0.97 ± 6.21	30%	-3.45 ± 4.05	10%	-4.03 ± 3.80	10%
h-DDPG	**9.88 ± 0.74**	**100%**	**8.86 ± 0.90**	**100%**	**7.73 ± 2.01**	**90%**

the meta critic could rely on. High performance basic skills provided by the basic critic and actors would help the meta critic to understand the movement patterns of each actor better and learn how to reuse them to achieve the final compound task.

In addition, we conducted a set of experiments in Scenario 1 to investigate the impact of an incompetent actor on the performance of other actors. In order to do so, we included an actor with a meaningless reward function. This actor received rewards randomly, so it would never learn. The results of the training are shown in Fig. 5. We can see that, the basic critic can still learn several high performance actors even when one of the actors has bad performance. This adds to the robustness of the proposed algorithm as individual failed actors will not interfere with the learning of other actors.

C. Meta Critic Performance

We first conducted a set of experiments in Scenario 1 to find an appropriate value of the reward for the meta critic. We compared the performance of the algorithm after 5000 episodes of training. The results are shown in Fig. 6. We can see that the performance is poor when the reward value for the meta critic is close to the reward for the basic critic. The agent generally failed to learn when the reward value was 1. The performance improves when the value increases and tops when reward value gets to 10, as the average steps in one episode becomes fewer and success rate reached 100%.

After this, the performance remains high in a range of reward values. However, it starts to drop when the value gets to 35. The performance gets worse when the value gets higher. This may be because of the unstable gradient updates caused by big loss values. This also demonstrates that a reward value between 10 and 30 is most suitable for the meta critic in learning compound skills in our scenarios. We then choose the reward value to be 10 and fixed it for the rest of the experiments.

The results of the final performance of the proposed h-DDPG algorithm in all three scenarios are shown in Fig. 7. We can see that, the agent started to find solutions to the tasks after training for around 2000 episodes. We can see from Fig. 4 that most actors achieve stable basic skills at around 1500 episodes. This means the meta critic actually learned better compound skills for the tasks right after stable basic skills became available. This also explains the instability of the performance before 2000 episodes as it would be hard for the meta critic to infer the basic skills of unstable actors.

For Scenario 1 and 2, the agent successfully achieved the final goal in more than 90% of test cases with random initialization conducted during training. This statistic is lower for Scenario 3, which is at around 80%. We observed that most failures in Scenario 3 were caused by collisions with the sides of a door when the robot tried to go through it. This may be caused by the fact that when the robot is near the door, it becomes harder to infer the orientation and position as what it can sense from camera and range sensors there is extremely similar (it is blank outside of the door). Sometimes, we observed the robot tried to solve the task by reversing out of the doors. This may be the way the agent used to infer orientation and position near the door.

In all three scenarios, the agent was able to solve the tasks within around 18 action steps. Note that this is highly relevant to the position and orientation of the random initialization, which also partially causes the high deviation in Fig. 7. We can see from samples of task solving trajectories given in Fig. 3 that the agent was actually solving the tasks with near optimal solutions from different initializations. The overall success rate of the proposed algorithm in solving the tasks is 87.6%.

D. Comparisons with Other Algorithms

We compared the proposed algorithm with two well-known deep reinforcement learning algorithms: DQN [15] and DDPG [17]. Both algorithms are one-thread training based, so it is suitable for comparison with the proposed h-DDPG algorithm. The first one is a discrete action algorithm, so we fixed the speed of the wheels in each basic skill so that the agent can get access to the four basic skills we used in our algorithm with quality equal to the best quality skills learned by our h-DDPG actors. The speed values for each of the basic skills were chosen via a grid search around the values frequently chosen by the

corresponding actors. We used the network architecture proposed in their original papers and did a grid search for hyper-parameters such as learning rate and discount factor centred around the values in the original papers. We finally fixed these hyper-parameters at their original values as we found they provide better performance. We also tuned the reward values for more stable learning and unified them to the values chosen for h-DDPG when reporting the results. We compared the best performance averaged from 10 test episodes obtained by learning with these three algorithms in all three scenarios in TABLE I.

We can see that, the proposed h-DDPG outperformed DQN and DDPG in solving tasks in all three scenarios. DDPG frequently failed to solve the tasks, and the success rates are very low. This implies it may be hard to learn compound skills without knowing any basic skills.

Although DQN can solve all three tasks, it is less capable to do so compared to h-DDPG as the success rates are lower. Moreover, it took more action steps to solve the tasks compared to h-DDPG. We observed that when using DQN, it took much more effort to keep the robot moving smoothly. This is mainly caused by the abrupt changes in speed when changing from one skill to another, as actions are executed in continuous time space and the length of time an action will be executed may vary. This may have influenced the performance of DQN. In contrast, when using h-DDPG, the robot was able to move more smoothly owing to the actors that could adjust the speed when performing basic skills to avoid sharp changes in speed. This is also one of the advantages of h-DDPG for handling robot control in a continuous action space.

These comparisons show that by introducing the hierarchical architecture and decomposing compound and basic skill learning, the proposed algorithm can not only learn better compound skills compared to other continuous action control algorithms, but also achieve smoother movement to support more stable compound skill learning compared to discrete action control algorithms.

VI. Conclusion

In this paper, we proposed a novel hierarchical deep reinforcement learning algorithm called h-DDPG. The proposed algorithm is made up of two levels of hierarchy. It can learn both basic skills and compound skills in a continuous action space simultaneously. The first level of hierarchy comprises a single basic critic with multiple actors, each handling a particular basic skill. This basic critic trains multiple actors at the same time. The second level of hierarchy contains a meta critic. This meta critic learns compound skills by reusing basic skills in the first level of hierarchy to solve compound tasks. The overall learning process of h-DDPG shares a single replay memory and skills in both hierarchies are learned within the same process.

In order to test the proposed algorithm, we built three scenarios with different compound tasks in Gazebo 2 in a ROS Indigo environment. A simulated Pioneer 3AT robot with front view camera is used in these simulations. The tasks in these three scenarios are designed to examine the agent capability in observing and distinguishing objects and moving accurately.

The results show that the proposed algorithm successfully learns both high performance basic skills and compound skills. In total, it successfully solved the tasks with a rate of 87.6% among all test cases with random position and orientation initialization in different scenarios. Results also show that in cases that some of the actors fail to learn, other actors can still learn high performance basic skills. In comparison with other algorithms, the proposed h-DDPG outperforms other one-thread training based algorithms while also achieving comparable performance against other discrete action based algorithms in solving compound tasks.

However, the proposed algorithm also has three main drawbacks. Firstly, we found that the agent needs sufficient free space in the environment to explore the continuous action space to find the movement patterns for basic skills. Secondly, the algorithm is confined to solve tasks in fully observable scenarios and may fail in partially observable tasks. Lastly, the algorithm requires predefined reward functions for all levels of hierarchy. Nonetheless, the capability of the proposed algorithm at solving fully observable compound tasks and providing smoother movement in a continuous action space should still make it a competitive algorithm compared to other deep reinforcement learning algorithms.

Future work could focus on three different aspects to improve the proposed algorithm. Firstly, technical improvements on the replay memory would help reduce the free space needed for learning basic skills as transitions in the replay memory may have different contribution to the learning, and those that are beneficial should be highlighted. This may also help make the whole learning process faster as compound skills are based on the basic skills. Secondly, the introduction of new levels of hierarchy can enable the algorithm to use multi-level hierarchy to decompose complex tasks in a more detailed way, which may give more potential to solve partially observable tasks. Thirdly, a combination of intrinsic motivations with the proposed algorithm could help eliminate the need of predefined reward functions. This could further add to the generalization and flexibility of the algorithm.

Appendix

TABLE II
Value of Hyper-Parameters

Description	Symbol (if has)	Value
Meta critic base learning rate	μ_{Q_m}	0.0025
Meta critic discount factor	γ_m	0.99
Basic critic base learning rate	μ_{Q_b}	0.001
Basic critic discount factor	γ_b	0.9
Basic critic reward	r_b	1
Actors base learning rate	μ_{π_g}	0.0001
l_2 penalty	/	0.01
Mini-batch size	M	64
Initial actor noise intensity	σ_{init}	0.12
Minimum actor noise intensity	σ_{min}	0.021
Initial ϵ	/	1
Minimum ϵ	/	0.1
Soft update factor	φ	0.001

REFERENCES

[1] R. S. Sutton and A. G. Barto, "*Reinforcement Learning: An Introduction,"* vol. 1. Cambridge: MIT Press, 1998.

[2] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning,* vol. 3, pp. 9-44, 1988.

[3] C. J. C. H. Watkins and P. Dayan, "Q-Learning," *Machine Learning,* vol. 8, pp. 279-292, 1992.

[4] G. A. Rummery and M. Niranjan, "*On-line Q Learning using Connectionist Systems."* Cambridge, England: University of Cambridge, Department of Engineering, 1994.

[5] M. Grounds and D. Kudenko, "*Parallel reinforcement learning with linear function approximation,"* vol. 4865. Berlin, Heidelberg: Springer, 2008.

[6] G. Konidaris, S. Osentoski, and P. Thomas, "Value Function Approximation in Reinforcement Learning using the Fourier Basis," in *Proc. AAAI*, 2011, pp. 380-385.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature,* vol. 521, pp. 436–444, 2015.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Proc. Adv. NIPS*, 2015, pp. 91-99.

[9] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep Networks with Stochastic Depth," in *ECCV*, 2016, pp. 646-661.

[10] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani*, et al.*, "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing," in *Proc. ICML*, 2016, pp. 1378-1387.

[11] Z. Liu, X. Li, P. Luo, C. C. Loy, and X. Tang, "Semantic Image Segmentation via Deep Parsing Network," in *ICCV*, 2015, pp. 1377-1385.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Network," in *Proc. Adv. NIPS*, 2012, pp. 1097-1105.

[13] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *ICASSP*, 2013, pp. 6645-6649.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra*, et al.*, "Playing Atari with Deep Reinforcement Learning," presented at the NIPS, 2013.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare*, et al.*, "Human-level Control through Deep reinforcement Learning," *Nature,* vol. 518, pp. 529-533, 2015.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche*, et al.*, "Mastering the Game of Go with Deep Neural Networks and TreeSearch," *Nature,* vol. 529, pp. 484-489, 2016.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa*, et al.*, "Continuous Control with Deep Reinforcement Learning," in *ICLR*, 2016.

[18] Z. Yang, K. Merrick, H. Abbass, and L. Jin, "Multi-Task Deep Reinforcement Learning for Continuous Action Control," in *Proc. IJCAI*, 2017, pp. 3301-3307.

[19] M. Lin, Q. Chen, and S. Yan. "Network in Network." *arXiv preprint arXiv:1312.4400*, 2013.

[20] Z. Wang, T. Schaul, M. Hessel, H. v. Hasselt, M. Lanctot, and N. d. Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *Proc. ICML*, 2016, pp. 1995-2003.

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *ICLR*, 2016.

[22] G. Dulac-Arnold, R. Evans, H. v. Hasselt, P. Sunehag, T. Lillicrap, J. Hunt*, et al.*, "Deep Reinforcement Learning in Large Discrete Action Spaces," presented at the ICML Abstraction in Reinforcement Learning Workshop, 2016.

[23] J. Peters, S. Vijayakumar, and S. Schaal, "Natural Actor-Critic," in *ECML*, 2005, pp. 280-291.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap*, et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proc. ICML*, 2016, pp. 1928-1937.

[25] N. Heess, G. Wayne, D. Silver, T. Lillicrap, Y. Tassa, and T. Erez, "Learning Continuous Control Policies by Stochastic Value Gradients," in *Proc. Adv. NIPS*, 2015.

[26] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proc. ICML*, 2015, pp. 1889-1897.

[27] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," in *Proc. ICML*, 2016, pp. 2829-2838.

[28] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," in *Proc. ICML*, 2016, pp. 1329-1338.

[29] A. G. Barto and S. Mahadevan, "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dynamic Systems,* vol. 13, pp. 341-379, 2003.

[30] N. Dethlefs and H. Cuayahuitl, "Hierarchical Reinforcement Learning for Situated Natural Language Generation," *Natural Language Engineering,* vol. 21, pp. 391-435, 2015.

[31] 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. NIPS*, 2016, pp. 3675-3683.

[32] R. Krishnamurthy, A. Lakshminarayanan, P. Kumar, and B. Ravindran, "Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks," presented at the ICML Abstraction in Reinforcement Learning Workshop, 2016.

[33] P. L. Bacon, J. Harb, and D. Precup, "The Option-Critic Architecture," in *Proc. AAAI*, 2017, pp. 1726-1734.

[34] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver*, et al.* "FeUdal Networks for Hierarchical Reinforcement Learning." *arXiv preprint arXiv:1703.01161*, 2017.

[35] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial intelligence,* vol. 112, pp. 181-211, 1999.

[36] G. Baldassarre and M. Mirolli, "*Intrinsically Motivated Learning in Natural and Artificial Systems."* Berlin, Heidelberg: Springer, 2013.

[37] S. Mohamed and D. J. Rezende, "Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning," in *Proc. Adv. NIPS*, 2015, pp. 2125-2133.

[38] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proc. ICML*, 2014, pp. 387-395.

[39] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Review,* vol. 36, p. 823, 1930.

[40] Z. Feng, Z. Yang, L. Jin, S. Huang, and J. Sun, "Robust Shared Feature Learning for Script and Handwritten/Machine-Printed Identification," *Pattern Recognition Letters,* vol. 100, pp. 6-13, 2017.

[41] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," presented at the ICLR, 2015.

[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro*, et al.* "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." *arXiv preprint arXiv:1603.04467*, 2016.

Zhaoyang Yang received a bachelor of engineering degree at School of Electronic and Information Engineering at South China University of Technology (SCUT), Guangzhou, China, in 2015. He is currently pursuing the master degree at SCUT, while also doing a duel master degree program at University of New South Wales, Canberra, Australia. His current research interests include deep learning, computer vision, and robotics.

Kathryn Merrick has a bachelor of Computer Science and Technology (Advanced, Honours I, University Medal), University of Sydney, NSW, Australia, 2002; PhD (computer science), National ICT Australia and University of Sydney, NSW, Australia, 2007. She is an Associate Professor in information technology at the University of New South Wales, Canberra, ACT. Her research lies in the areas of autonomous mental development and computational motivation, with applications in virtual characters, developmental robotics and intelligent environments. Her research is principally concerned with the development of algorithms for self-motivated learning agents. She is co-author of two books and over seventy refereed conference and journal papers. She is an associate editor of the IEEE Trans. on Cognitive and Developmental Systems, Chair of the IEEE Technical Committee on Cognitive and Developmental Systems (2017) and Chair of the ACT Chapter of the IEEE Computational Intelligence Society (2017).

Hussein Abbass is a Professor of Information Technology at the University of New South Wales at the Australian Defence Force Academy (UNSW@ ADFA) in Canberra, Australia. He is a fellow of the Australian Computer Society (FACS), a fellow of the Operational Research Society (FORS, UK); a fellow of the Australian Institute of Management (AFAIM), and the Vice-president for Technical Activities (2016-2017) for the IEEE Computational Intelligence Society. He is an associate Editor of the IEEE Trans. On Evolutionary Computation, IEEE Trans. on Cybernetics, IEEE Trans. on Cognitive and Developmental Systems, IEEE Computational Intelligence Magazine, and four other journals. His current research contributes to trusted autonomy with an aim to design next generation trusted artificial intelligence systems that seamlessly integrate humans and machines. His work fuses artificial intelligence, big data, cognitive science, operations research, and robotics.

Lianwen Jin received the B.S. degree from the University of Science and Technology of China, Anhui, China, and the Ph.D. degree from the South China University of Technology, Guangzhou, China, in 1991 and 1996, respectively. He is currently a Professor with the School of Electronic and Information Engineering, South China University of Technology. He is the author of more than 150 scientific papers. Dr. Jin was a recipient of the award of New Century Excellent Talent Program of MOE in 2006 and the Guangdong Pearl River Distinguished Professor Award in 2011. His research interests include image processing, machine learning, document analysis and recognition, computer vision and intelligent systems. He is member of the IEEE Computer Society, IEEE Signal Processing Society, IEEE System Man and Cybernetics Society, and IEEE Computational Intelligence Society.