Multisource Transfer Double DQN Based on Actor Learning

Jie Pan, Xuesong Wang, *Member, IEEE*, Yuhu Cheng, *Member, IEEE*, and Qiang Yu

***Abstract*— Deep reinforcement learning (RL) comprehensively uses the psychological mechanisms of "trial and error" and "reward and punishment" in RL as well as powerful feature expression and nonlinear mapping in deep learning. Currently, it plays an essential role in the fields of artificial intelligence and machine learning. Since an RL agent needs to constantly interact with its surroundings, the deep Q network (DQN) is inevitably faced with the need to learn numerous network parameters, which results in low learning efficiency. In this paper, a multisource transfer double DQN (MTDDQN) based on actor learning is proposed. The transfer learning technique is integrated with deep RL to make the RL agent collect, summarize, and transfer action knowledge, including policy mimic and feature regression, to the training of related tasks. There exists action overestimation in DQN, i.e., the lower probability limit of action corresponding to the maximum Q value is nonzero. Therefore, the transfer network is trained by using double DQN to eliminate the error accumulation caused by action overestimation. In addition, to avoid negative transfer, i.e., to ensure strong correlations between source and target tasks, a multisource transfer learning mechanism is applied. The Atari2600 game is tested on the arcade learning environment platform to evaluate the feasibility and performance of MTDDQN by comparing it with some mainstream approaches, such as DQN and double DQN. Experiments prove that MTDDQN achieves not only human-like actor learning transfer capability, but also the desired learning efficiency and testing accuracy on target task.**

***Index Terms*— Actor learning, Atari2600 game, double deep Q network (DQN), multisource transfer.**

I. Introduction

DEEP reinforcement learning (RL) plays a significant role in the development of artificial intelligence by comprehensively using the psychological and neurological mechanisms of RL [1]–[3] and the powerful feature representation capability of deep neural networks. In recent years, the progress reports on RL agents, such as AlphaGo and Master, have had tremendous social influence, which has set off a boom in the field of machine learning and artificial intelligence. The milestone of these studies is the deep Q network (DQN) proposed by Google DeepMind [4], which breakouts the learning mechanism of value function approximation and policy search based on shallow structure in the traditional RL methods [5]. By means of multilayer deep convolutional neural networks, an end-to-end mapping from high-dimensional input space to Q value space is realized to imitate the human brain's activity. The human-level control precision has been achieved in the game data set of Atari2600. Inspired by the advantages of DQN, researchers have developed various applications, such as optimization hyper-parameters [6], character segmentation of license plate images [7], traffic signal timing [8], automatic bridge bidding [9], financial signal processing [10], autonomous vehicle control [11], language understanding for text-based games [12], terrain-adaptive locomotion [13], saccadic-based object visual search [14], and natural language processing [15], [16]. These studies show that DQN has the capabilities of hierarchical feature extraction and accurate Q value approximation in various high-dimensional sensing environments.

Besides the exploration of different background applications, more researchers have focused on structural optimization of DQN, performance gains of Q value approximation and efficient training for RL agents [17]. Ong *et al.* [18] proposed a distributed DQN that can efficiently train a complicated RL agent on multiple machines in a cluster. The DistBelief frame is applied and developed in this network. By using an asynchronous gradient updating policy for different RL agents, the training progress of DQN is greatly accelerated. Compared with the synchronous policy agents of linear efficiency, the asynchronous agents consume less time for training. To further improve efficiency, Taylor *et al.* [19] proposed a hierarchical structure of tasks, and transferred knowledge from the source to target tasks. The hierarchical structure accelerates the training of RL agents for multiple target tasks. However, it is found that the algorithm is unstable intrinsically if RL and deep neural networks are simply combined with each other. Since there is a strong correlation among time-series data obtained by the RL agent, the continuous observation and updating of online learning will create instability for the agent. The offline data storage and experience replay can greatly reduce algorithmic instability, i.e., the batch processing and random sampling in the subsequent time steps are able to eliminate the strong correlation between time-series data for each network updating. Numerous improved algorithms were proposed based on these considerations [20]–[23].

Manuscript received April 26, 2017; revised October 10, 2017, December 24, 2017, and February 8, 2018; accepted February 10, 2018. Date of publication March 9, 2018; date of current version May 15, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61772532, Grant 61472424, and Grant 51607180 and in part by the Natural Science Foundation of Jiangsu Province under Grant BK20150203. *(Corresponding author: Xuesong Wang.)*

J. Pan, X. Wang, and Y. Cheng are with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China (e-mail: panjie1616@126.com; wangxuesongcumt@163.com; chengyuhu@163.com).

Q. Yu is with the School of Electrical and Power Engineering, China University of Mining and Technology, Xuzhou 221116, China (e-mail: yuqiangcumt@163.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2018.2806087

However, although the resampling method can ensure the algorithmic stability, RL is confined to off-policy learning [24], [25]. In addition, the offline data storage requires plenty of memory space, and the computational complexity is high at each iteration step. To overcome these problems, Mnih *et al.* [26] proposed an asynchronous algorithm for deep RL. The asynchronous advantage actor-critic algorithm can not only sharply reduce the hardware requirements of multiple GPUs, but also achieve desired effects on 2-D and 3-D game testing scenarios.

In terms of the DQN structure, only the state-action values (also known as Q values) are estimated, and the state values that have nothing to do with actions are not considered. The estimate of all state-action values will result in poor generalization of actions when the agent is in an environment that is not influenced by actions. To overcome this problem, Wang *et al.* [27] added a dueling architecture to the output layer of deep network, which can represent state-dependent action advantage function and state values. The superiority of the architecture is that the generalization of the algorithm is greatly improved. Besides, the transfer between different networks is easy to be achieved since the architecture does not depend on the algorithm. Different from local optimization of structures, double DQN offers an improvement of a global network updating. When estimating state-action values by DQN, Hasselt *et al.* [22] found that in the unbiased estimation, the lower probability limit of the maximum Q value does not equal to zero, which may cause action overestimation, thereby affecting policy choice and agent's performance. To eliminate the influence of action overestimation, a symmetrical updating mechanism of the dual network was adopted by Hasselt *et al.* [22]. In the structure, one network is used for action estimation and decision making, while the other is to update Q values. Compared with the above single RL agent, the multiagent system is much more complex. Since there are competitive relationships between RL agents, the opponent modeling is necessary. Thus, both environments and opponent factors need to be considered in the decision-making process. Therefore, He *et al.* [28] incorporated the opponent's behavior into the input of deep RL network. This unsupervised mechanism based on environmental rewards makes it easier for RL agent to learn competitive policies against opponents.

The above research work revolves around a single task for an RL agent. The agent needs to constantly interact with the environment to make decisions. Thus, the process takes a long time. The powerful feature representation and function approximation of deep neural networks often mean a high cost of computation. If knowledge from related tasks, such as weights, value functions, and policies, can be transferred to target tasks, the learning difficulties can be sharply reduced. Many researchers have tried to resolve this problem by three approaches: model transfer, sample transfer, and feature transfer. Model transfer is the simplest and most direct approach. Fachantidis *et al.* [29] transferred the state transition and reward function models of source tasks to target tasks. In this way, they realize the transfer from 2-D to 3-D in the mountain car task. However, the algorithm performance will be affected by the model dependence between related tasks. Compared with model transfer, sample transfer is more general. It is reported in [30] that even if there are obvious model differences between source and target tasks, the sample transfer from source tasks can still help the target task reduce requirements for samples.

Inspired by the psychology of knowledge reconstruction, Taylor and Stone [31] pointed out that feature transfer can be applied to related RL tasks, even if there exist different feature representations in the source and target tasks. Since a value function directly affects the decision making of an RL agent, an action transfer based on value function was proposed in [32]. Specifically, the RL agent was not pretrained according to some artificially generated tasks but was instead applied to different related tasks to construct value function mapping relationships between different feature representations. Since these related tasks may have different state-action spaces, feature transfer has a wide range of applications. Ammar *et al.* [33] noted that in terms of intertask transfer, the key factor is the construction of mappings, which comprehensively uses sparse coding, sparse mapping, and sparse feature representation of Gaussian process to obtain high-dimensional sparse spaces of source tasks. In addition, the integration method of the least-square policy iteration and the fitted Q-iteration is superior to those conventional Markov decision process (MDP) mapping methods. Regarding feature mapping, Libeau *et al.* [34] studied the application of virtual climbing robots in the field of transfer RL. The state-action space of RL agent was abstracted as "perceptions" free from tasks, which is deemed as a high-level feature transfer across tasks. Similar to states perceptions, Croonenborghs *et al.* [35] viewed feature transfer as a related "option," which intrinsically belongs to the set of "macro actions" that RL agent uses to achieve high-level skills. The hierarchical abstraction facilitates to realize inductive transfer across tasks. All of the above methods are examples of transfer RL using a single RL agent. To further extend to multiagent systems, both cooperation and competition between RL agents should be considered in the process of transfer. It is extremely complicated and difficult to implement. Da Silva and Costa [36] proposed a framework for a solution to this problem, in which the RL agent needs to perform transfer learning from the following two aspects: 1) knowledge extraction from source tasks and 2) knowledge transfer from experienced RL agents. In addition, bias transfer and Q value reuse were adopted for multiagent learning [37].

Although transfer RL is not a new research area, few studies focus on deep RL algorithms, such as transfer DQN. Compared with traditional back propagation networks, radial basis function networks, and MDP, a deep neural network has more layers, more parameters, and longer training time. To accelerate the process of learning, it is necessary to study the method of transfer learning in DQN. If the errors from action estimation cannot be theoretically eliminated, the transfer process of DQN may accumulate these errors to cause the poor performance of the algorithm. Therefore, a double DQN is adopted in this paper to remove the influence of action overestimation. Inspired by the modes of action evaluation and policy iteration in actor-critic learning, we propose a multisource transfer double DQN (MTDDQN) algorithm based on actor learning, which can reduce the influence of negative transfer existing in single-source transfer. First, expert networks are trained on multiple source tasks and thus expert policies are obtained. Then, the generated policy mimic loss is used to train the multisource transfer network (MTN) that contains each piece of expert policy information. The key aspect of MTDDQN lies in actor learning, which includes two parts: policy mimic with network correction capability and feature regression with output layer optimization ability. The proposed MTDDQN algorithm is tested using the Atari2600 game on the platform of arcade learning environment. Compared with several mainstream algorithms, results show that MTDDQN can not only accelerate the learning speed of the target task, but also achieve the desired accuracy of testing.

This paper is structured as follows. The related actor-critic learning and double DQN algorithms are introduced in Section II. The proposed MTDDQN is explained, including system framework and actor learning of target network (TN) in Section III. The algorithm convergence is analyzed in Section IV. The experimental results on the Atari2600 game are reported in Section V, followed by conclusions in Section VI.

II. Related Work

A. Actor-Critic Learning

RL is essentially a method of dynamic programming in a complex environment [38]. To overcome the curse of dimensionality, a parameterized expression of continuous Q value is used as a value function of RL [39]. Usually, the state-action value function $Q^{\pi}(s,a)$ and the state value function $V^{\pi}(s)$ are specified as

$$Q^{\pi}(s,a) = E_{\tau}\left\{\sum_{t=0}^{\infty}\gamma^{t}r_{t}|s_0 = s, a_0 = a\right\} \tag{1}$$

$$V^{\pi}(s) = E_{\tau}\left\{\sum_{t=0}^{\infty}\gamma^{t}r_{t}|s_0 = s\right\} \tag{2}$$

where s and a are the state and action of an RL agent, respectively; π is the policy; r_t is the reward at time t; τ is the state trajectory; and $\gamma \in (0,1)$ denotes the discount factor.

To make $Q^{\pi}(s,a)$ and $V^{\pi}(s)$ approximate the ideal functions $Q^{*}(s,a)$ and $V^{*}(s)$, respectively, the parameterized loss function needs to be minimized by

$$\begin{aligned} J(\theta) &= E\left\{(1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}r_{t}|\theta\right\} \\ &= \int_{S}d^{\pi}(s)\int_{A}\pi(a|s)r(s,a)dsda \end{aligned} \tag{3}$$

where $d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}p(s_t = s)$ denotes the discount state distribution.

The loss function $J(\theta)$ can be computed by its gradient $\nabla_{\theta}J(\theta)$, which can also be used to measure the progress of policy iteration. The most commonly used method is to adopt the natural policy gradient by $\tilde{\nabla}_{\theta}J(\theta) = F^{-1}(\theta)\nabla_{\theta}J(\theta)$, where $F(\theta)$ is the Fisher information matrix. The action evaluation of critic comprehensively considers two factors: the natural policy gradient and the parameterized value function of linear regression. Let

$$\begin{aligned} \nabla_{\theta}J(\theta) = \bigg[\int_{S}d^{\pi}(s)\int_{A}\pi(a|s)\nabla_{\theta}\log\pi(s|a) \\ \times\nabla_{\theta}\log\pi(s|a)^{T}dads\bigg]w = F(\theta)w \end{aligned} \tag{4}$$

where w is the critic parameter. Then

$$F(\theta,s) = \int_{A}\pi(a|s)\nabla_{\theta}\log\pi(a|s)\nabla_{\theta}\log\pi(a|s)^{T}da. \tag{5}$$

By (5), the policies are evaluated to further guide parameter updating of value functions.

B. Double DQN

DQN enables the mapping from the agent's state space $\mathbf{R}^{m}$ to the Q value space $\mathbf{R}^{n}$ [22]. The objective function for network weight updating is defined as

$$Y_{t}^{\mathrm{DQN}} \equiv r_{t+1} + \gamma\max_{a}Q\big(s_{t+1},a;\theta_{t}^{-}\big) \tag{6}$$

where θ_{t}^{-} is the duplicated value of weight θ of online DQN at every τ-step, i.e., there exists $\theta_{t}^{-} = \theta_{t}$ at the time of duplication and θ_{t}^{-} remains unchanged at other times. Therefore, the objective function can be further transformed into

$$Y_{t}^{\mathrm{DQN}} \equiv r_{t+1} + \gamma Q(s_{t+1},\arg\max_{a}Q(s_{t+1},a;\theta_{t});\theta_{t}). \tag{7}$$

It is noted by Hasselt *et al.* [22] that the use of the same network weight θ for both policy decision and Q value approximation leads to a nonzero lower limit of $\max_{a}Q(s,a)$

$$\max_{a}Q(s,a) \geq V^{*}(s) + \sqrt{\frac{C}{m-1}} \tag{8}$$

where C is the variance of the state value and m is the number of optional actions at state s. The overestimation of $\max_{a}Q(s,a)$ will result in errors during the process of network training, which will eventually affect the learning performance of RL agent.

Therefore, the idea of double Q learning is applied to network updating of DQN, which means that the original weights θ_t are still applied when DQN is deciding an action, and a second set of weights θ_{t}' is used to approximate the objective function. In this approach, the lower limit of the estimated maximum Q value can be guaranteed zero. When $Q_{t}'(s,a_1) = V^{*}(s)$ is satisfied, the error of Q value is zero for arbitrary $Q_{t}'(s,a_i)(i > 1)$. That is, the objective function of double DQN is

$$Y_{t}^{\mathrm{DoubleDQN}} \equiv r_{t+1} + \gamma Q\big(s_{t+1},\arg\max_{a}Q(s_{t+1},a;\theta_{t});\theta_{t}'\big). \tag{9}$$

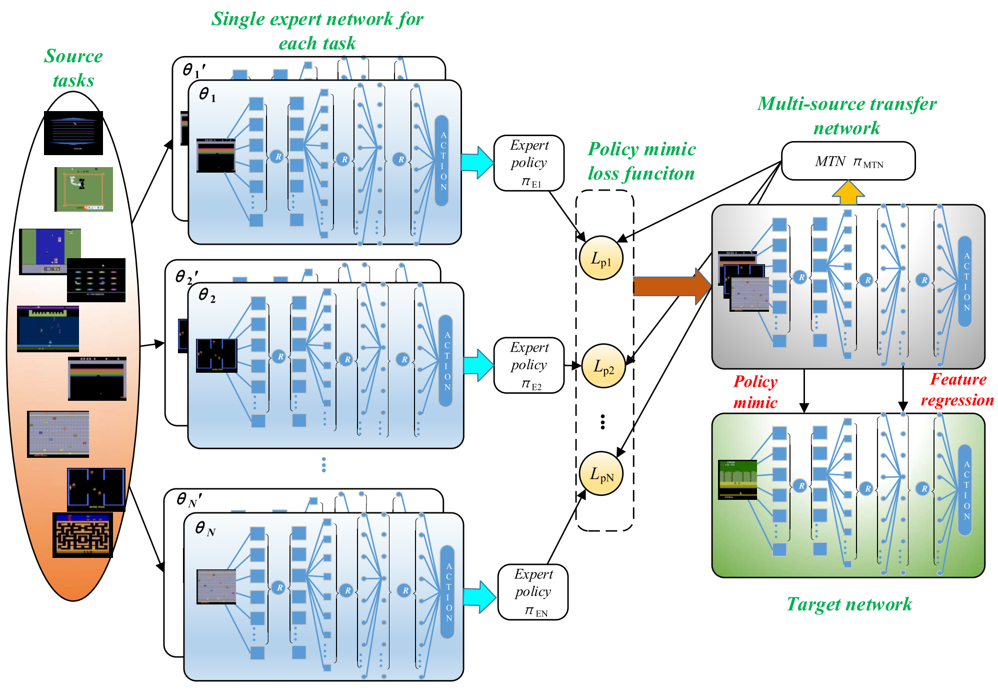


Fig. 1. Multisource transfer DQN based on actor learning.

III. Multisource Transfer Double DQN Based on Actor Learning

A. System Framework

The system framework of the proposed algorithm is illustrated in Fig. 1, which is divided into two parts: training of MTN and actor learning of TN.

One of difficulties in transfer learning is to choose a reasonable transfer source [40]. If only a single transfer source is considered, strong correlations between the source and target tasks must be guaranteed; otherwise, negative transfer will probably occur. Therefore, in this paper, the expert network E_i that is trained according to each single-source task is not directly applied. Instead, an MTN is jointly trained by using all source tasks. In addition, since double DQN can eliminate error accumulation caused by action overestimation, it is used to approximate each expert network. A double DQN has two networks that have identical structure and different network parameters θ and θ'. The network corresponding to θ makes decision, while the other ensures a reasonable Q value estimation. To guarantee the training efficiency, network parameters θ and θ' follow a symmetric updating rule. MTN can acquire integrated decision-making information from each expert network and select actions according to target domain information, thereby avoiding errors due to random action selection.

The difference between network parameters θ and θ' makes it impossible to perform direct transfer of Q values, even under the identical expert network structure. Therefore, in this paper, MTN is trained using a policy mimic mechanism, i.e., to ensure the policy loss between expert policy and MTN policy is small. The Gibbs distribution of state-action value $Q(s,a)$ is applied to convert expert network E_i into policy network $\pi_i(a|s)$. Different from the ε-greedy policy, the above policy choice mechanism takes comprehensive consideration of the effects on Q value of each action. Thus, the degree of policy similarity between networks is easy to determine. The policy of each expert network is denoted by π_{Ei} and expressed as follows:

$$\pi_{\mathrm{Ei}}(a|s) = Z_i^{-1} e^{\frac{1}{T} Q_{\mathrm{Ei}}(s,a)}, \quad Z_i = \sum\nolimits_{a' \in \mathbf{A}_{\mathrm{Ei}}} e^{\frac{1}{T} Q_{\mathrm{Ei}}(s,a')} \tag{10}$$

where T is the temperature parameter and $\mathbf{A}_{\mathrm{Ei}}$ is the action space of expert network E_i. It is noted that policy π_{Ei} is estimated by expert network $E_i(\theta')$ instead of $E_i(\theta)$. Similarly, the MTN used for Q value approximation is also converted into a policy network

$$\begin{aligned} \pi_{\mathrm{MTN}}(a|s) &= Z_{\mathrm{MTN}}^{-1} e^{\frac{1}{T} Q_{\mathrm{MTN}}(s,a)} \\ Z_{\mathrm{MTN}} &= \sum\nolimits_{a' \in A_{\mathrm{MTN}}} e^{\frac{1}{T} Q_{\mathrm{MTN}}(s,a')}. \end{aligned} \tag{11}$$

Then, the policy mimic loss L_{pi} between the expert network and MTN is measured by cross-entropy

$$L_{\mathrm{pi}}(\theta_i') = \sum\nolimits_{a \in A_{\mathrm{Ei}}} \pi_{\mathrm{Ei}}(a|s;\theta_i') \log \pi_{\mathrm{MTN}}(a|s). \tag{12}$$

This cross-entropy ensures that similar expert policy and MTN policy leads to a lower loss $L_{\mathrm{pi}}(\theta_i')$. In this way, multiple expert networks are used to train MTN. This may make MTN include more policy information. Furthermore, MTN is trained and transferred to the TN by actor learning. Details are given in Section III-B.

B. Actor Learning of Target Network

By minimizing the policy mimic loss between the expert network and MTN, as a learner network, MTN acquires comprehensive decision information from each expert network. For the TN, MTN becomes a supervisor and instructs

TN to quickly have the ability to initialize the optimal policy. To achieve this, the TN must perform actor learning on MTN, which includes two parts: policy mimic and feature regression. The policy mimic process of TN is similar to that of the MTN training. Through (10)–(12), the policy networks of MTN and TN are obtained and the policy mimic loss is calculated using cross-entropy. Policy mimic facilitates the construction of TN for overall weights, while the output of Q values is realized by feature regression.

Algorithm 1 MTDDQN

```
**Inputs:** Source task $\Gamma = \{S_1, S_2, \ldots, S_N\}$ and target task.
  1) **Initialization:** Source network parameters $\theta_1, \theta_2, \ldots, \theta_N$ and $\theta'_1, \theta'_2, \ldots, \theta'_N$, feature regression parameter $\theta_h$, policy network temperature parameter $T$, balancing coefficients $\alpha$ and $\beta$, multi-source transfer network parameter $\theta_{\mathrm{MTN}}$ and target network parameter $\theta_{\mathrm{TN}}$.
  2) **Multi-source transfer network training**
     a) Train each double DQN expert network $E_1, E_2, \ldots, E_N$ using source tasks $S_1, S_2, \ldots, S_N$.
     b) Determine policy $\pi_{\mathrm{E1}}, \pi_{\mathrm{E2}}, \ldots, \pi_{\mathrm{EN}}$, and MTN policy $\pi_{\mathrm{MTN}}$ according to (10)-(11) respectively.
     c) Calculate policy mimic loss $L_{\mathrm{pi}}(\theta'_i)$ between $E_i$ and MTN using (12).
  3) **Actor learning of target network**
     a) Determine policy $\pi_{\mathrm{MTN}}$ of MTN and policy $\pi_{\mathrm{TN}}$ of target network, according to (10) and (11) respectively.
     b) Calculate policy mimic loss of target network $L_p(\theta_{\mathrm{MTN}})$ according to (12).
     c) Calculate feature regression loss between MTN and TN $L_f(\theta_h)$ according to (13).
     d) Calculate actor learning loss $L_{\mathrm{AL}}(\theta_h, \theta_{\mathrm{MTN}})$ according to (14).
     e) Update network according to (15)-(16).
  4) **Target network correction:**
     Carry out fine-tuning of TN using target task to achieve the optimal learning effects.
**Output:** Target network.
```

Set the last fully connected layer of MTN as the feature extraction layer, then the output of the last fully connected layer of MTN is the extracted feature f_{MTN}. Thus, a feature regression network is constructed as $h(f_{\mathrm{MTN}}) \to f^*_{\mathrm{TN}}$, where f^*_{TN} is the ideal feature. Then, for a given state s, the feature regression loss between MTN and TN is

$$L_f(\theta_h) = ||h(f_{\mathrm{MTN}}(s), \theta_h) - f_{\mathrm{TN}}(s)||_2 \tag{13}$$

where θ_h is the parameter of feature regression network and $||\cdot||_2$ is the L2-norm. In the case of ideal regression, i.e., $f^*_{\mathrm{TN}} = f_{\mathrm{TN}}$, the feature regression loss $L_f(\theta_h) = 0$. Both policy mimic loss L_p and feature regression loss L_f are comprehensively considered, the actor learning loss is expressed as

$$L_{\mathrm{AL}}(\theta_h, \theta_{\mathrm{MTN}}) = \alpha L_p(\theta_{\mathrm{MTN}}) + \beta L_f(\theta_h) \tag{14}$$

where α and β are balancing coefficients of policy mimic and feature regression, respectively. Generally, we set $\alpha/\beta \gg 1$, which means that the policy mimic in MTN plays a leading role. Feature regression can accelerate network training but spoils the transfer effect to a certain extent. Therefore, β is usually set to a small value.

By the loss function L_{AL}, actor learning of TN can be further realized. To avoid overestimation of the maximum Q value, we set the state-action outputs of double DQN as $Q^A(s, \theta)$ and $Q^B(s, \theta')$, and update any of the two networks at random. If network A is updated, then

$$\begin{aligned} Q^A_{t+1}(s, a) = {} & Q^A_t(s, a|\theta) + \delta(s, a) \\ & \times \left[r + \gamma Q^B_t(s', a'|\theta') - Q^A_t(s, a|\theta)\right] \end{aligned} \tag{15}$$

else

$$\begin{aligned} Q^B_{t+1}(s, a) = {} & Q^B_t(s, a|\theta') + \delta(s, a) \\ & \times \left[r + \gamma Q^A_t(s', a'|\theta) - Q^B_t(s, a|\theta')\right] \end{aligned} \tag{16}$$

where r is the reward and $\delta(s, a)$ is the variable learning rate. The value of $\delta(s, a)$ is set as $1/t$ in the current state, while the other states are set to a constant value of 0.6.

In summary, the detailed steps of the proposed MTDDQN are summarized as Algorithm 1.

IV. Algorithm Convergence Analysis

The deep expert network E_i, which is trained on source task S_i, can approximate value function Q^*_i and learn policy π_{Ei}. When π_{Ei} is obtained from E_i using ε-greedy, it intrinsically depends on network parameter θ_i. Let the total reward $R_{\mathrm{MTN}} = E(\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_0, \pi)$ for MTN and the true value function $Q^*_{\mathrm{MTN}}(s, a)$, the derivative of R_{MTN} with respect to policy mimic loss is

$$\frac{\partial R_{\mathrm{MTN}}}{\partial L_{\mathrm{pi}}} = \sum_s Y^\pi(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s, a)}{\partial L_{\mathrm{pi}}} Q^*_{\mathrm{MTN}}(s, a) \tag{17}$$

where $Y^\pi(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s | s_0, \pi)$ is the discount transfer probability.

Let the Q value approximation of MTN be $Q_{\mathrm{MTN}}(s, a) : S \times A \to \mathrm{R}$. Then, the learning rule of Q value approximation under policy mimic is written as

$$\begin{aligned} \Delta L_{\mathrm{pi}} & \propto \frac{\partial}{\partial L_{\mathrm{pi}}} [\hat{Q}^\pi(s_t, a_t) - Q_{\mathrm{MTN}}(s_t, a_t)]^2 \\ & \propto [\hat{Q}^\pi(s_t, a_t) - Q_{\mathrm{MTN}}(s_t, a_t)] \frac{\partial Q_{\mathrm{MTN}}(s_t, a_t)}{\partial L_{\mathrm{pi}}} \end{aligned} \tag{18}$$

where $\hat{Q}^\pi(s_t, a_t)$ is the unbiased estimation of Q value under policy π. If the above process converges to a local optimal solution, then

$$\begin{aligned} \sum_s Y^\pi(s) \sum_a \pi_{\mathrm{MTN}}(s, a) [\hat{Q}^\pi(s, a) - Q_{\mathrm{MTN}}(s_t, a_t)] & \\ \times \frac{\partial Q_{\mathrm{MTN}}(s_t, a_t)}{\partial L_{\mathrm{pi}}} = 0. & \end{aligned} \tag{19}$$

Since $\partial Q_{\mathrm{MTN}}(s, a)/\partial L_{\mathrm{pi}} = \pi_{\mathrm{MTN}} \cdot \partial \pi_{\mathrm{MTN}}(s, a)/\partial \theta$, (19) is rewritten as

$$\sum_s Y^\pi(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s, a)}{\partial L_{\mathrm{pi}}} [\hat{Q}^\pi(s, a) - Q_{\mathrm{MTN}}(s, a)] = 0. \tag{20}$$

From (20), there is an orthogonal relationship between the estimation error of Q value under policy mimic and the partial derivative of policy π_{MTN} with respect to policy mimic loss. Moreover, when (20) is satisfied, $\hat{Q}^{\pi}(s,a) \approx Q^*_{\mathrm{MTN}}(s,a)$. By utilizing (17), (20) can be written as

$$\begin{aligned}\frac{\partial R_{\mathrm{MTN}}}{\partial L_{\mathrm{pi}}} &= \sum_s Y^{\pi}(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s,a)}{\partial L_{\mathrm{pi}}} Q^*_{\mathrm{MTN}}(s,a) \\ &\quad - \sum_s Y^{\pi}(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s,a)}{\partial L_{\mathrm{pi}}} \\ &\quad \times [\hat{Q}^{\pi}(s,a) - Q_{\mathrm{MTN}}(s,a)] \\ &= \sum_s Y^{\pi}(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s,a)}{\partial L_{\mathrm{pi}}} \\ &\quad \times \left[Q^*_{\mathrm{MTN}}(s,a) - \hat{Q}^{\pi}(s,a) + Q_{\mathrm{MTN}}(s,a)\right] \\ &\approx \sum_s Y^{\pi}(s) \sum_a \frac{\partial \pi_{\mathrm{MTN}}(s,a)}{\partial L_{\mathrm{pi}}} Q_{\mathrm{MTN}}(s,a). \end{aligned} \tag{21}$$

Therefore, $Q_{\mathrm{MTN}}(s,a) \rightarrow Q^*_{\mathrm{MTN}}(s,a)$.

For an arbitrary MTN policy π_{MTN} and an arbitrary TN π_{TN}, there are network parameters θ_{MTN} and θ_{TN}. By the two parameters, the policy network is updated to $\Phi(\hat{Q}_{\theta_{\mathrm{MTN}}})$ and $\Phi(\hat{Q}_{\theta_{\mathrm{TN}}})$. According to the Lipschtiz Law of Continuity, we have

$$\begin{aligned}&||\Phi(\hat{Q}_{\mathrm{MTN}}) - \Phi(\hat{Q}_{\mathrm{TN}})|| \leq c_1 ||\hat{Q}_{\mathrm{MTN}} - \hat{Q}_{\mathrm{TN}}|| \\ &\quad = c_1 ||H\theta_{\mathrm{MTN}} - H\theta_{\mathrm{TN}}|| \leq c_1 ||H|| \cdot ||\theta_{\mathrm{MTN}} - \theta_{\mathrm{TN}}|| \end{aligned} \tag{22}$$

where H is the linear approximation of unbiased estimate $\hat{Q}$ in terms of network parameter, $\hat{Q}_{\mathrm{MTN}} = H\theta_{\mathrm{MTN}}$, $\hat{Q}_{\mathrm{TN}} = H\theta_{\mathrm{TN}}$, and $||\cdot||$ is the Euclidean norm.

For the TN of actor learning, suppose $P_{\theta_{t-1}}$ is the state-action prediction probability of the $(t-1)$th iterative output of the softmax layer, Π_{MTN} is the policy matrix, and $D_{\pi}(\cdot)$ is a diagonal matrix. Network parameter updating by the gradient descent method is adopted according to

$$\Delta\theta_t = -\alpha_t [H^T D_{\pi}(P_{\theta_{t-1}} - \Pi_{\mathrm{MTN}}) + \beta_{\theta}\theta_{t-1}] \tag{23}$$

where α_t is the learning factor and β_{θ} is the decay factor. The network is continuously updated until reaching the steady state $\Delta\theta_{\mathrm{MTN}} = 0$ and $\Delta\theta_{\mathrm{TN}} = 0$ with policies π_{MTN} and π_{TN}. Then, according to (23)

$$\begin{aligned}&||\theta_{\mathrm{MTN}} - \theta_{\mathrm{TN}}|| \\ &= ||H^T D_{\pi_{\mathrm{MTN}}}(P_{\theta_{\mathrm{MTN}}} - \Pi_{\mathrm{MTN}}) \\ &\qquad - H^T D_{\pi_{\mathrm{TN}}}(P_{\theta_{\mathrm{TN}}} - \Pi_{\mathrm{MTN}})||/\beta_{\theta} \\ &= ||H^T|| \cdot ||\Pi_{\mathrm{MTN}}(D_{\pi_{\mathrm{MTN}}} - D_{\pi_{\mathrm{TN}}}) + D_{\pi_{\mathrm{MTN}}}(P_{\theta_{\mathrm{MTN}}} - P_{\theta_{\mathrm{TN}}}) \\ &\qquad + P_{\theta_{\mathrm{TN}}}(D_{\pi_{\mathrm{TN}}} - D_{\pi_{\mathrm{MTN}}})||/\beta_{\theta} \\ &\leq ||H^T|| \cdot (||\Pi_{\mathrm{MTN}}|| \cdot ||D_{\pi_{\mathrm{MTN}}} - D_{\pi_{\mathrm{TN}}}|| \\ &\qquad + ||D_{\pi_{\mathrm{MTN}}}|| \cdot ||P_{\theta_{\mathrm{MTN}}} - P_{\theta_{\mathrm{TN}}}|| \\ &\qquad + ||P_{\theta_{\mathrm{TN}}}|| \cdot ||D_{\pi_{\mathrm{MTN}}} - D_{\pi_{\mathrm{TN}}}||)/\beta_{\theta}. \end{aligned} \tag{24}$$

There exist constants c_2 and c_3 such that $||D_{\pi_{\mathrm{MTN}}} - D_{\pi_{\mathrm{TN}}}|| \leq c_2||\pi_{\mathrm{MTN}} - \pi_{\mathrm{TN}}||$ and $||P_{\theta_{\mathrm{MTN}}} - P_{\theta_{\mathrm{TN}}}|| \leq c_3||\pi_{\mathrm{MTN}} - \pi_{\mathrm{TN}}||$. Therefore, $||\theta_{\mathrm{MTN}} - \theta_{\mathrm{TN}}|| \leq c_4||\pi_{\mathrm{MTN}} - \pi_{\mathrm{TN}}||$ and (22) can be rewritten as

$$||\Phi(\hat{Q}_{\mathrm{MTN}}) - \Phi(\hat{Q}_{\mathrm{TN}})|| \leq c||\pi_{\mathrm{MTN}} - \pi_{\mathrm{TN}}|| \tag{25}$$

where $c = c_1 c_4 ||H|| \in (0,1)$ is a constant. Therefore, for policies of continuous change, TN will converge to fixed network parameters θ^*_{TN} with probability 1.

V. Experiments

The mainstream Atari2600 game data set is adopted to test the proposed MTDDQN algorithm. As a second-generation console released by Atari Corporation in 1977, Atari2600 has been popular worldwide for decades, and over 500 games have been developed. Categories include shooting, fighting, puzzle, sports, and adventure. And many of them set up new game modes [21]. However, due to the hardware limitations in the early period, such as 1.19MHz CPU, 2- to 4-kB ROM, 160×210 pixels, 128 colors, and the panel of 18 action inputs, the difficulty levels of various games are relatively low and balanced. In tests on different cross-domain games, RL agents can achieve a human-like intelligence level, while avoiding test imbalance due to excessively high difficulty level. The adopted platform of arcade learning environment provides an interface for all kinds of Atari2600 games. It not only offers image input for the Q value network of the RL agent, but also realizes real-time interaction between the RL agent and the environment by providing action feedback to the game.

A. Experimental Settings

The structure and parameters of each expert network E_i are set according to [4], i.e., three convolution layers and two full-connected layers. Each convolution layer is followed by a ReLU layer. In addition, the $84 \times 84 \times 4$-size images are taken as input of the expert networks, instead of the original $160 \times 210 \times 3$-size images. Specific parameter settings are as follows: with standard double DQN training by expert network E_i, number of training periods epochs =1000, number of steps in each period step = 200 000, learning interval learning interval = 1, discount factor $\gamma = 0.99$, learning rate $\mathrm{lr} = 2.5 \times 10^{-4}$, greedy policy exploration rate $\varepsilon = 0.05$, parameters $\theta_1, \theta_2, \ldots, \theta_N$ and $\theta'_1, \theta'_2, \ldots, \theta'_N$ of each network is randomly initialized, and bias $b = 0.1$. The collection of E_i is the transfer source, which is defined as $\Gamma =$ {Freeway, Enduro, Hero, Ice Hockey, Phoenix, Pong, Qbert, River Raid, Krull, Skiing, Pitfall, Robotank, Zaxxon, Bank Heist, Berzerk, Asterix, Breakout, Alien, Boxing, Amidar} or a subcollection. Based on each E_i from MTN, the number of training steps of policy mimic is steps_MTN = 2000 000; step_MTN $\ll$ *step_E$_i$* = 200 000 000 to avoid overfitting of E_i by MTN. The structure settings of MTN and TN are as follows: $8 \times 8 \times 4 \times 256 - 4 \rightarrow 4 \times 4 \times 256 \times 512 - 2 \rightarrow 3 \times 3 \times 512 \times 512 - 1 \rightarrow 3 \times 3 \times 512 \times 512 - 1 \rightarrow 2048$ full connections $\rightarrow$1024 full connections, respectively. The balancing coefficients in the TN actor learning stage are $\alpha = \beta = 0.5$.

B. Analysis of Transfer Learning Effects

Experimental results of MTDDQN under step = 1000 000 and 4000 000 training steps are shown in Fig. 2. It can be seen from Fig. 2 that the player needs to hold the moving ball by controlling the "short panel" under the screen to hit each layer

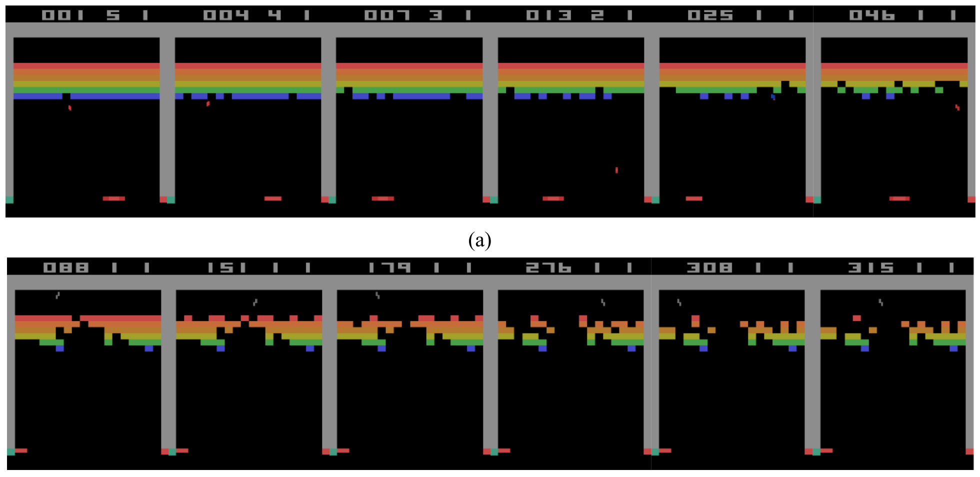


Fig. 2. MTDDQN policy learning at different training steps (Breakout game). (a) Step = 1000 000. (b) Step = 4000 000.

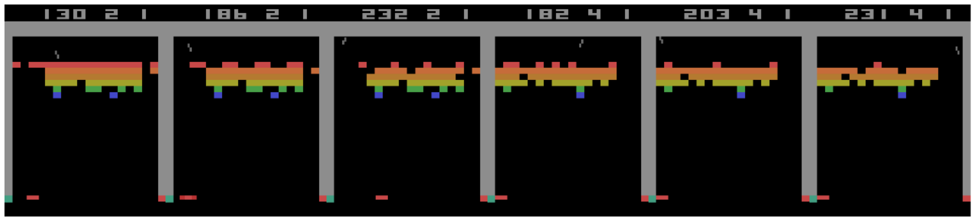


Fig. 3. Breakdown policies of two different zones (Breakout game).

TABLE I
COMPARISON OF SCORES AT DIFFERENT TIME STEPS (BREAKOUT)

Algorithm	Step=1000000	Step=2000000	Step=4000000	Step=10000000
DQN	13.40	22.20	43.10	119.20
double DQN	17.10	24.30	51.90	126.90
MTDDQN	**49.70**	**129.60**	**312.10**	**335.20**
TE	190.64%	433.33%	501.35%	164.14%

of blocks above and obtains scores that gradually increase from the bottom to the top layers. When step = 1000 000, the RL agent is in the exploration stage and cannot execute any effective policy, other than to merely catch the moving ball. As shown in Fig. 2(a), all the stricken blocks in a complete game are confined to the first three layers. The hit points are completely random and lack strategy, and the game ends with a low score of 46. When step = 4000 000, as shown in Fig. 2(b), the RL agent has mastered the advanced strike policy of "block layer breakdown," in which broken balls can directly hit blocks in the top layer to obtain higher score. Mastery of this policy enables far higher score than the random hits. Experimental results support this conclusion, and the breakdown zone is not limited to block layers intervals; it also includes both the left and right sides. As shown in Fig. 3, the first three panels show breakdowns at the left side, while the last three show breakdowns at the right side.

It is predicted that for each game, there exist policies of different levels of complexity. Such policies are developed from players' experience and intuition, while the learning of RL agent policy not only involves artificial interference, but also is unsupervised. The RL agent does not know the operational principles of games. Instead, it summarizes the corresponding policy using rewards by interacting with environment. Hence, the intelligence and learning efficiency of RL agent are determined by whether a corresponding policy can be quickly obtained. Tables I–IV show the comparison of scores obtained by DQN, double DQN, and MTDDQN at different time steps on Breakout, Enduro, Crazy Climber, and Beam Rider games. In addition, the transfer efficiency (TE) of MTDDQN, which is compared with that of double DQN, is given. TE reflects the efficiency improvement of a transfer learning algorithm compared with the original nontransfer algorithm, which is modeled as

$$\mathrm{TE} = (\mathrm{Score}_{\mathrm{MTDDQN}} - \mathrm{Score}_{\mathrm{DoubleDQN}})/\mathrm{Score}_{\mathrm{DoubleDQN}}. \tag{26}$$

The following can be seen from Tables I–IV.

TABLE II
COMPARISON OF SCORES AT DIFFERENT TIME STEPS (ENDURO)

Algorithm	Step=1000000	Step=2000000	Step=4000000	Step=10000000
DQN	16.30	33.70	69.90	198.77
double DQN	15.40	41.20	72.30	215.50
MTDDQN	**101.30**	**134.50**	**199.70**	**257.42**
TE	557.79%	226.46%	176.21%	19.45%

TABLE III
COMPARISON OF SCORES AT DIFFERENT TIME STEPS (CRAZY CLIMBER)

Algorithm	Step=1000000	Step=2000000	Step=4000000	Step=10000000
DQN	15672.00	22375.33	30185.00	99524.33
double DQN	14896.00	25412.66	37452.00	87129.00
MTDDQN	**40125.33**	**54254.00**	**65441.00**	**100255.00**
TE	169.37%	113.49%	74.73%	15.07%

TABLE IV
COMPARISON OF SCORES AT DIFFERENT TIME STEPS (BEAM RIDER)

Algorithm	Steps=1000000	Step=2000000	Step=4000000	Step=10000000
DQN	524.71	1051.35	2103.65	4995.93
double DQN	612.04	997.34	2256.31	5634.00
MTDDQN	**1806.32**	**2574.31**	**4681.20**	**7025.90**
TE	195.13%	158.12%	107.47%	24.71%

1) With the increase of time step, scores of all algorithms gradually increase.
2) Scores of MTDDQN on all of four games are the highest at any time step, which verifies that transfer learning technique can facilitate MTDDQN to find the optimal policy within a relatively short period of time steps. Therefore, the learning efficiency of MTDDQN is the highest.
3) As time step increases, the TE of MTDDQN tends to decrease.

This is because more exploration on target tasks is achieved by DQN and double DQN with the increase of time step, which will improve the policy learning capacities of DQN and double DQN. In addition, knowledge transferred from source tasks gradually tend to be saturated, i.e., MTDDQN cannot acquire more valuable information from source tasks to further improve its policy learning capacity at the late learning stage. Therefore, if the learning time is long enough, compared with DQN and double DQN, the superiority of MTDDQN is not obvious on learning accuracy. But the learning efficiency of MTDDQN is always the highest among the three algorithms.

C. Analysis of Actor Learning Effects

To obtain more direct and precise descriptions of the policy learning processes of DQN, double DQN, and MTDDQN, Fig. 4 shows the average Q value curves for Breakout, Enduro, Gopher, and River Raid games. Different from reward, the Q value directly reflects ultimate learning effect. That is, the Q value distribution determines which policy the RL agent adopts. The following can be seen from Fig. 4.

1) Initially, the RL agent is at the policy exploration stage; thus, the average Q values of all algorithms are relatively low.
2) Although the average Q values of the three algorithms converge to identical or similar values, the policy mimic learning mechanism of multisource transfer ensures that the RL agent of MTDDQN obtains a stable policy network at an earlier stage.
3) At the initial exploration stage, MTDDQN has a faster rise and higher stability values. This indicates that faster policy exploration can be realized via multisource transfer learning.
4) A larger approximated Q value indicates that a more stable policy and a higher expected reward can be obtained. DQN, compared with double DQN, explores faster. However, due to action overestimation, its ultimate expected reward is inferior to that of the other two deep RL algorithms.

Fig. 4 shows that the algorithm is a convergence process, which is more important than learning efficiency and accuracy. Based on analysis in Section IV, for an arbitrary MTN policy π_{MTN} and TN policy π_{TN}, there exist network parameters θ_{MTN} and θ_{TN}. Hence, there exists a constant c_4 satisfying $||\theta_{\text{MTN}} - \theta_{\text{TN}}|| \leq c_4||\pi_{\text{MTN}} - \pi_{\text{TN}}||$. Therefore, theoretically, the TN for actor learning converges to a fixed policy π^*

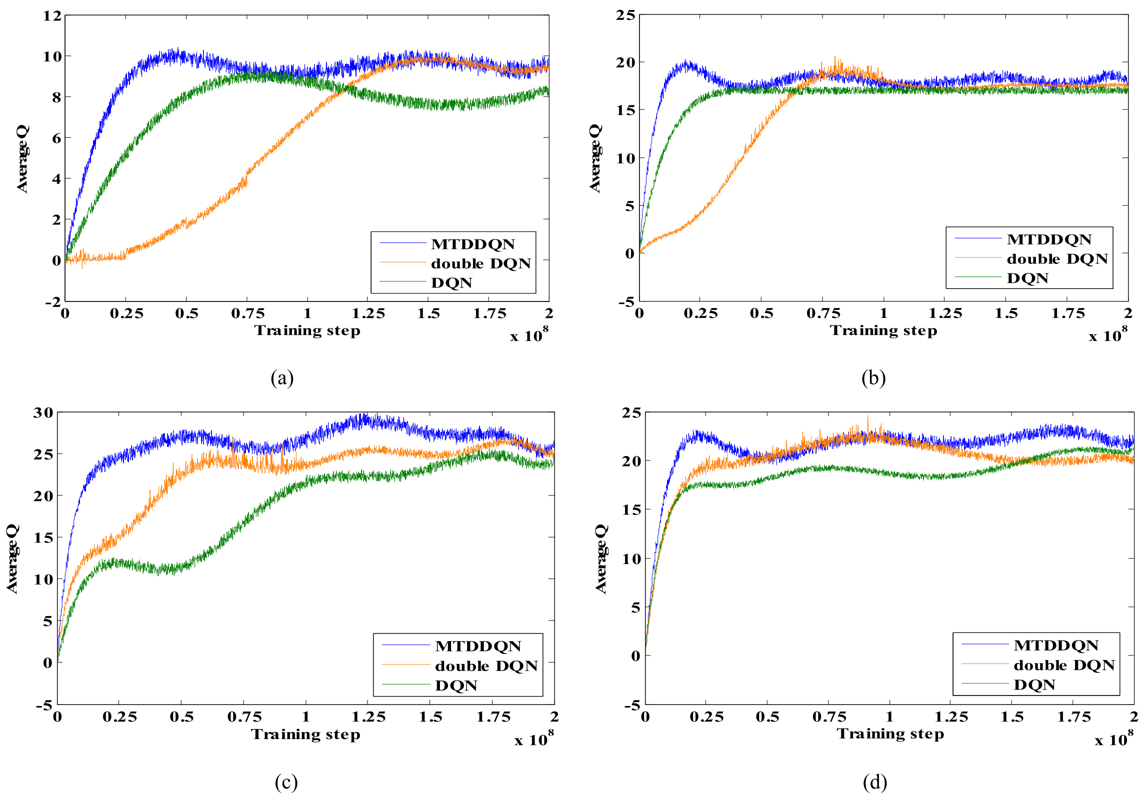


Fig. 4. Average Q value curves on different games. (a) Breakout game. (b) Enduro game. (c) Gopher game. (d) River raid game.

with probability 1 and obtains a fixed network parameter θ^*_{TN}. This conclusion is verified in Fig. 4.

The game scores of MTDDQN, double DQN, and DQN are shown in Fig. 5, which actually are the average reward from RL agent over all epochs. Compared with Q value, the reward can reflect the learning effect of RL agent at different stages more directly. At the exploration stage, when step is within 0–2×10^7, the rewards for Breakout, Enduro, Gopher, and River Raid games increase with policy search. In this stage, the exploration advantage of MTDDQN can be clearly observed. At the policy optimization stage, when step is within $2 \times 10^7 - 1 \times 10^8$, the rewards of MTDDQN and double DQN increase steadily, while those of DQN fall into local optimum to different extents, which show the fluctuation of policy learning, especially on Breakout and Gopher games. Due to action overestimation of DQN, at the policy steady stage, when step is within $1 \times 10^8 - 2 \times 10^8$, DQN yields worse effect compared with MTDDQN and double DQN. Moreover, it is found that the scores have large instability even at the policy steady stage, such that action selection under the same policy for each task are sensitive and a tiny change can lead to a large difference in reward. However, the average rewards of all algorithms are relatively stable, and MTDDQN is slightly better than DQN and double DQN.

The learning effects for 10 groups of target tasks under different balancing coefficients are shown in Fig. 6, after training with step $= 10\,000\,000$. To set all scores in one identical coordinate for convenience of comparison, the scores of policy mimic, feature regression, and actor learning are all divided by that of double DQN, respectively; thus, the relative score of task learning is given in percentages. The balancing coefficients for each target task are: 1) $\alpha = 1$ and $\beta = 0$, which corresponds to policy mimic; 2) $\alpha = 0$ and $\beta = 1$, which corresponds to feature regression; and 3) $\alpha = 0.5$ and $\beta = 0.5$, which corresponds to actor learning. The following conclusions are drawn.

1) Under reasonable balancing coefficients, except for Robotank, the obtained relative scores are over 100%, which means the effect is superior to that of the baseline double DQN method. For Robotank, the score is below 80%, with obvious negative transfer.
2) Compared with policy learning and feature regression, actor learning by jointly considering two factors can ensure optimal result with higher probability. Six out of 10 games, Wizard of War, Road Runner, Robotank, Star Gunner, Kangaroo, and Space Invaders, demonstrate advantages of actor learning.
3) Policy mimic behaves better on most tasks than feature regression.

According to (14), the error between MTN and TN is described from different aspects by the three methods: policy mimic, feature regression, and actor learning. The policy mimic adopts mimic loss across networks to ensure that TN approximates MTN, while feature regression more depends on the similarity degree of the network output. Therefore, policy mimic achieves better learning performance than feature regression.

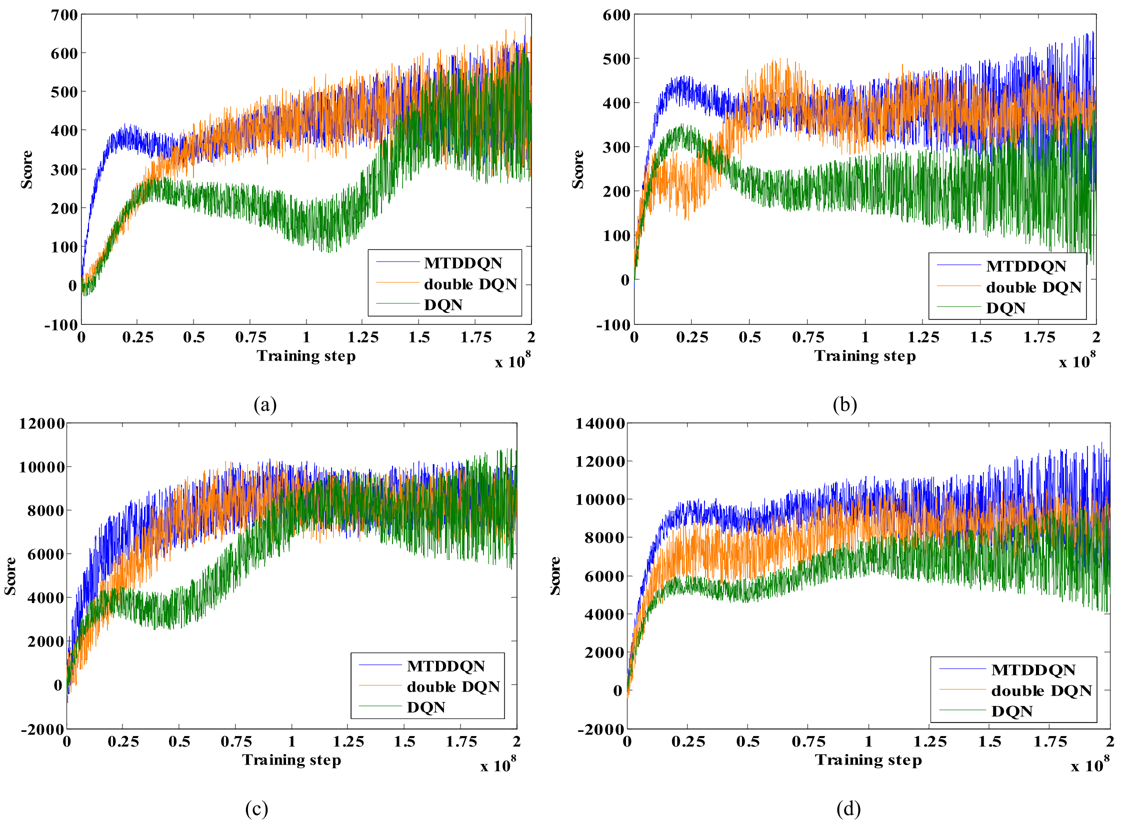


Fig. 5. Score curves on different games. (a) Breakout game. (b) Enduro game. (c) Gopher game. (d) River raid game.

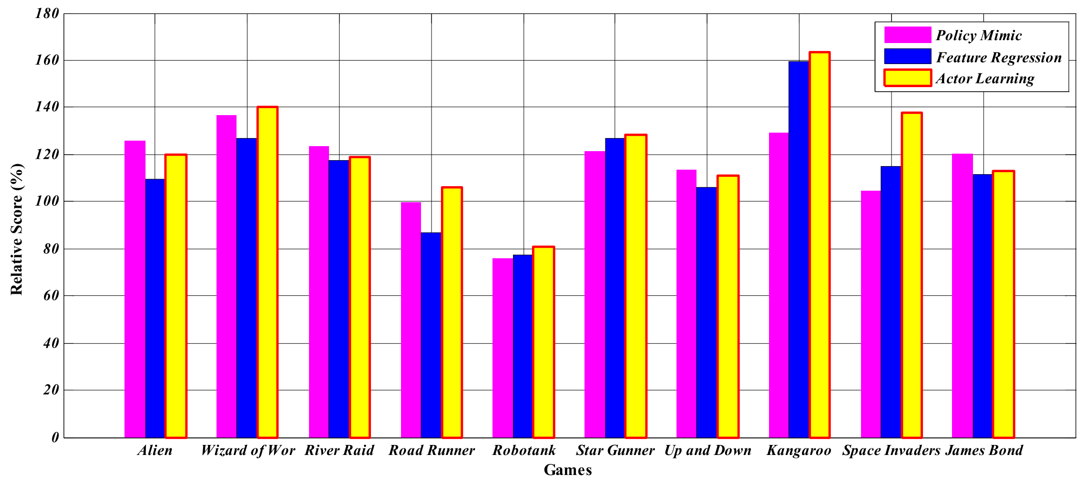


Fig. 6. Actor learning effect under different balancing coefficients.

VI. Conclusion

The proposed MTDDQN algorithm offers a new approach for fast training and learning of deep RL agents in various environments and tasks. An RL agent can extract transferrable knowledge from multiple experienced source tasks to accelerate the training and learning of target tasks. Compared with DQN and double DQN, the proposed MTDDQN achieves the desired learning efficiency, i.e., fewer training time steps are necessary when a suboptimal policy is obtained. More specifically, MTDDQN has the following main advantages.

1) Training of expert networks allows MTN to combine the cross-domain policies of different tasks. The learned polices are then viewed as transfer sources so that negative transfer can be avoided.
2) Unlike sample transfer and model transfer, the deep network for Q value approximation is converted into a policy network. By the policy mimic loss, the similarity degrees between networks are measured to realize policy approximation from the learner to the expert level.
3) The actor learning of TN includes both policy mimic and feature regression. The effect of policy transfer

is strengthened, and the relationship between policy transfer and feature transfer is optimized by adjusting balancing coefficients.

REFERENCES

[1] D. Zhao and Y. Zhu, "MEC-a near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Trans. Neural Netw. Learn. Sys.*, vol. 26, no. 2, pp. 346–356, Feb. 2015.

[2] B. Piot, M. Geist, and O. Pietquin, "Bridging the gap between imitation learning and inverse reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 8, pp. 1814–1826, Aug. 2017.

[3] J. Li, H. Modares, T. Chai, F. L. Lewis, and L. Xie, "Off-policy reinforcement learning for synchronization in multiagent graphical games," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2434–2445, Oct. 2017.

[4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[5] Y. Zhu, D. Zhao, and X. Li, "Iterative adaptive dynamic programming for solving unknown nonlinear zero-sum game based on online data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 714–725, Mar. 2017.

[6] S. Hansen. (2016). "Using deep Q-learning to control optimization hyperparameters." [Online]. Available: https://arxiv.org/abs/1602.04062

[7] F. Abtahi, Z. Zhu, and A. M. Burry, "A deep reinforcement learning approach to character segmentation of license plate images," in *Proc. IAPR Int. Conf. Mach. Vis. Appl.*, Jul. 2015, pp. 539–542.

[8] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA J. Automat. Sinica*, vol. 3, no. 3, pp. 247–254, Apr. 2016.

[9] C. K. Yeh and H. T. Lin, "Automatic bridge bidding using deep reinforcement learning," in *Proc. 22nd Eur. Conf. Artif. Intell.*, Aug. 2016, pp. 1362–1369.

[10] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[11] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in *Proc. Int. Symp. Comput. Intell. Des.*, Jan. 2017, pp. 198–201.

[12] K. Narasimhan, T. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," in *Proc. Conf. Empir. Methods Nature Lang. Process.*, Sep. 2015, pp. 1–11.

[13] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 81–94, Jul. 2016.

[14] T. Kornuta and K. Rocki, "Utilization of deep reinforcement learning for saccadic-based object visual search," *Adv. Intell. Sys. Comput.*, vol. 550, pp. 565–574, 2017.

[15] J. He *et al.*, "Deep reinforcement learning with a natural language action space," in *Proc. Annu. Meet. Assoc. Comput. Linguist.*, Aug. 2016, pp. 1621–1630.

[16] H. Cuayáhuitl, *SimpleDS: A Simple Deep Reinforcement Learning Dialogue System* (Lecture Notes in Electrical Engineering). Saariselka, Finland: Springer-Verlag, Jan. 2016, pp. 109–118.

[17] X. Xu, Z. Huang, L. Zuo, and H. He, "Manifold-based reinforcement learning via locally linear reconstruction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 4, pp. 934–947, Apr. 2017.

[18] H. Y. Ong, K. Chavez, and A. Hong. (2015). "Distributed deep Q-learning." [Online]. Available: https://arxiv.org/abs/1508.04186

[19] M. E. Taylor, G. Kuhlmann, and P. Stone, "Accelerating search with transferred heuristics," in *Proc. ICAPS Workshop AI Planning Learn.*, 2007.

[20] M. Riedmiller, "Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.*, Oct. 2005, pp. 317–328.

[21] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning," in *Proc. Adv. Neural Inf. Proces. Syst.*, Dec. 2014, pp. 3338–3346.

[22] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, Feb. 2016, pp. 2094–2100.

[23] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. (2015). "High-dimensional continuous control using generalized advantage estimation." [Online]. Available: https://arxiv.org/abs/1506.02438

[24] D. Zhao, Q. Zhang, D. Wang, and Y. Zhu, "Experience replay for optimal control of nonzero-sum game systems with unknown dynamics," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 854–865, Mar. 2016.

[25] D. Zhao, Z. Xia, and D. Wang, "Model-free optimal control for affine nonlinear systems with convergence analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 4, pp. 1461–1468, Oct. 2015.

[26] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1928–1937.

[27] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2016, pp. 2939–2947.

[28] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé, III, "Opponent modeling in deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1804–1813.

[29] A. Fachantidis, I. Partalas, G. Tsoumakas, and I. Vlahavas, "Transferring models in hybrid reinforcement learning agents," in *Proc. IFIP Adv. Inf. Commun. Technol.*, Sep. 2011, pp. 162–171.

[30] A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning," in *Proc. 25th Int. Conf. Mach. Learn.*, Jul. 2008, pp. 544–551.

[31] M. E. Taylor and P. Stone, "Representation transfer for reinforcement learning," In *Proc. AAAI Fall Symp. Tech. Rep.*, Nov. 2007, pp. 78–85.

[32] M. E. Taylor, P. Stone, and Y. Liu, "Value functions for RL-based behavior transfer: A comparative study," in *Proc. Nat. Conf. Artif. Intell.*, Jul. 2005, pp. 880–885.

[33] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss, "Reinforcement learning transfer via sparse coding," in *Proc. 11th Int. Conf. Auton. Agents Multiagent Syst.*, Jun. 2012, pp. 383–390.

[34] B. Libeau, A. Micaelli, and O. Sigaud, "Transfer of knowledge for a climbing virtual human: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 2119–2124.

[35] T. Croonenborghs, K. Driessens, and M. Bruynooghe, "Learning relational options for inductive transfer in relational reinforcement learning," in *Proc. Lect. Notes Comput. Sci.*, Jun. 2007, pp. 88–97.

[36] F. L. da Silva and A. H. R. Costa, "Transfer learning for multiagent reinforcement learning systems," in *Proc. 25th IJCAI Int. Joint Conf. Artif. Intell.*, Jul. 2016, pp. 3982–3983.

[37] G. Boutsioukis, I. Partalas, and I. Vlahavas, "Transfer learning in multi-agent reinforcement learning domains," in *Proc. Lect. Notes Comput. Sci.*, Sep. 2011, pp. 249–260.

[38] L. Dong, X. Zhong, C. Sun, and H. He, "Event-triggered adaptive dynamic programming for continuous-time systems with control constraints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 8, pp. 1941–1952, Aug. 2017.

[39] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, Nov. 2009.

[40] L. Yang, L. Jing, J. Yu, and M. K. Ng, "Learning transferred weights from co-occurrence data for heterogeneous transfer learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 11, pp. 2187–2200, Nov. 2016.

Jie Pan received the Ph.D. degree from the China University of Mining and Technology, Xuzhou, China, in 2014.

He is currently a Lecturer with the School of Information and Control Engineering, China University of Mining and Technology. His current research interests include transfer learning, image processing, and deep learning.

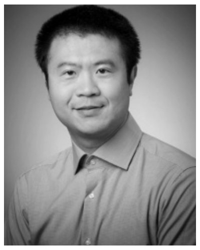
Xuesong Wang (M'15) received the Ph.D. degree from the China University of Mining and Technology, Xuzhou, China, in 2002.

She is currently a Professor with the School of Information and Control Engineering, China University of Mining and Technology. Her current research interests include machine learning, bioinformatics, and artificial intelligence.

Dr. Wang is serving as an Associate Editor of IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS and IEEE ACCESS.

Yuhu Cheng (M'15) received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2005.

He is currently a Professor with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou, China. His current research interests include machine learning, transfer learning, and intelligent system.



Qiang Yu received the Ph.D. degree from the University of Bundeswehr Muenchen, Munich, Germany, in 2012.

He is currently an Associate Professor with the School of Electrical and Power Engineering, China University of Mining and Technology, Xuzhou, China. His current research interests include intelligent learning and systems.