Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 2129393, 6 pages
https://doi.org/10.1155/2018/2129393

Research Article

Ensemble Network Architecture for Deep Reinforcement Learning

Xi-liang Chen, Lei Cao, Chen-xi Li, Zhi-xiong Xu, and Jun Lai

Institute of Command Information System, PLA University of Science and Technology, No. 1, Hai Fu Road, Guang Hua Road, Qin Huai District, Nanjing City, Jiangsu Province 210007, China

Correspondence should be addressed to Xi-liang Chen; 383618393@qq.com

Received 8 September 2017; Revised 10 February 2018; Accepted 20 February 2018; Published 5 April 2018

Academic Editor: Jian G. Zhou

The popular deep Q learning algorithm is known to be instability because of the Q-value's shake and overestimation action values under certain conditions. These issues tend to adversely affect their performance. In this paper, we develop the ensemble network architecture for deep reinforcement learning which is based on value function approximation. The temporal ensemble stabilizes the training process by reducing the variance of target approximation error and the ensemble of target values reduces the overestimate and makes better performance by estimating more accurate Q-value. Our results show that this architecture leads to statistically significant better value evaluation and more stable and better performance on several classical control tasks at OpenAI Gym environment.

1. Introduction

Reinforcement learning (RL) algorithms [1, 2] are very suitable for learning to control an agent by letting it interact with an environment. In recent years, deep neural networks (DNN) have been introduced into reinforcement learning, and they have achieved a great success on the value function approximation. The first deep Q-network (DQN) algorithm which successfully combines a powerful nonlinear function approximation technique known as DNN together with the Q-learning algorithm was proposed by Mnih et al. [3]. In this paper, experience replay mechanism was proposed. Following the DQN work, a variety of solutions have been proposed to stabilize the algorithms [3–9]. The deep Q-networks classes have achieved unprecedented success in challenging domains such as Atari 2600 and some other games.

Although DQN algorithms have been successful in solving many problems because of their powerful function approximation ability and strong generalization between similar state inputs, they are still poor in solving some issues. Two reasons for this are as follows: (a) the randomness of the sampling is likely to lead to serious shock and (b) these systematic errors might cause instability, poor performance, and sometimes divergence of learning. In order to address these issues, the averaged target DQN (ADQN) [10] algorithm is implemented to construct target values by combining target Q-networks continuously with a single learning network, and the Bootstrapped DQN [11] algorithm is proposed to get more efficient exploration and better performance with the use of several Q-networks learning in parallel. Although these algorithms do reduce the overestimate, they do not evaluate the importance of the past learned networks. Besides, high variance in target values combined with the max operator still exists.

There are some ensemble algorithms [4, 12] solving this issue in reinforcement learning, but these existing algorithms are not compatible with nonlinearly parameterized value functions.

In this paper, we propose the ensemble algorithm as a solution to this problem. In order to enhance learning speed and final performance, we combine multiple reinforcement learning algorithms in a single agent with several ensemble algorithms to determine the actions or action probabilities. In supervised learning, ensemble algorithms such as bagging, boosting, and mixtures of experts [13] are often used for learning and combining multiple classifiers. But in RL, ensemble algorithms are used for representing and learning the value function.

Based on an agent integrated with multiple reinforcement learning algorithms, multiple value functions are learned at the same time. The ensembles combine the policies derived from the value functions in a final policy for the agent. The majority voting (MV), the rank voting (RV), the Boltzmann multiplication (BM), and the Boltzmann addition (BA) are used to combine RL algorithms. While these methods are costly in deep reinforcement learning (DRL) algorithms, we combine different DRL algorithms that learn separate value functions and policies. Therefore in our ensemble approaches we combine the different policies derived from the update targets learned by deep Q-networks, deep Sarsa networks, double deep Q-networks, and other DRL algorithms. As a consequence, this leads to reduced overestimations, more stable learning process, and improved performance.

2. Related Work

2.1. Reinforcement Learning. Reinforcement learning is a machine learning method that allows the system to interact with and learn from the environment to maximize cumulative return rewards. Assume that the standard reinforcement learning setting where an agent interacts with the environment ε. We can describe this process with Markov Decision Processes (MDP) [2, 9]. It can be specified as a tuple (S, A, π, R, γ). At each step t, the agent receives a state s_t, and select an action a_t from the set of legal actions A according to the policy π, where π is a policy mapping sequences to actions. The action is passed to the environment E. In addition, the agent receives the next state s_{t+1} and a reward signal r_t. This process continues until the agent reaches a terminal state.

The agent seeks to maximize the expected discounted return, where we define the future discounted return at time t as $R_t = \sum_{k=0}^{\infty} \gamma^k r^{t+k}$ with discount factor $\gamma \in (0, 1]$. The goal of the RL agent is to learn a policy which makes the future discounted return maximize. For an agent behaving according to a stochastic policy π, the value of the state-action pair can be defined as follows: $Q^{\pi}(s, a) = E\{R_t \mid s_t = s, a_t = a, \pi\}$. The optimal action-value function Q satisfies the Bellman equation $Q^*(s, a) = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$.

The reinforcement learning algorithms estimate the action value function by iteratively updating the Bellman equation $Q^*(s, a) = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$. When $t \to \infty$, the algorithm makes Q-value function converge to the optimal action value function [1]. If the optimal Q-function Q^* is known, the agent can select optimal actions by selecting the action with the maximal value in a state: $\pi^* = \text{argmax}_a Q^*(s, a)$.

2.2. Target Deep Q Learning. RL agents update their model parameters while they observe a stream of transitions like $(s_t, a_t, r_{t+1}, s_{t+1})$. They discard the incoming data after a single update. There are two issues with this method. The first one is that there are strong correlations among the incoming data, which may break the assumption of many popular stochastic gradient-based algorithms. Secondly, the minor changes in the Q function may result in a huge change in the policy, which makes the algorithm difficult to converge [7, 9, 14, 15].

As for the deep Q-networks algorithms proposed in (Mnih et al., 2013), two aspects are improved. On the one hand, the action value function is approximated by the DNN, DQN uses the DNN with a parameter θ to approximate the value function, $Q(s, a; \theta) \approx Q^*(s, a; \theta)$; on the other hand, the experience replay mechanism is adopted. The algorithm learns from sampled transitions from an experience buffer, rather than learning fully online. This mechanism makes it possible to break the temporal correlations by mixing more and less recent experience for updating and training. This model free reinforcement learning algorithm solves the problem of "model disaster" and uses the generalized approximation method of the value function to solve the problem of "dimension disaster."

The convergence issue was mentioned in 2015 by Schaul et al. [14]. The above Q-learning update rules can be directly implemented in a neural network. DQN uses the DNN with parameters θ to approximate the value function$_{\circ}$. The parameter θ updates from transition $(s_t, a_t, r_{t+1}, s_{t+1})$ are given by the following [11]:

$$\theta_{t+1} \longleftarrow \theta_t + \alpha \left(y_t^{\mathrm{Q}} - Q\left(s_t, a_t; \theta_t\right) \right) \nabla_{\theta} Q\left(s_t, a_t; \theta_t\right), \quad (1)$$

with $y_i^{\mathrm{DQN}} = r_t + \gamma \max_a \widetilde{Q}(s_{t+1}, a; \theta_t^-)$

The update targets for Sarsa can be described as follows:

$$y_i^{\mathrm{Sarsa}} = r_t + \gamma \widetilde{Q}\left(s_{t+1}, a_{t+1}; \theta_t^-\right), \quad (2)$$

where α is the scalar learning rate. θ^- are target network parameters which are fixed to $\theta^- = \theta^t$. In case the squared error is taken as a loss function $L_i(\theta_i) = E_{s' \sim \varepsilon}(y_i - Q(s, a; \theta_i))^2$.

In general, experience replay can reduce the amount of experience required to learn and replace it with more computation and more memory, which are often cheaper resources than the RL agent's interactions with its environment [14].

2.3. Double Deep Q Learning. In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can therefore lead to overoptimistic value estimates (van Hasselt, 2010). To mitigate this problem, the update targets value of double Q-learning error can then be written as follows:

$$y_i^{\mathrm{DDQN}} = r_t + \gamma \widetilde{Q}\left(s_{t+1}, \arg\max_{a'} \widetilde{Q}\left(s_{t+1}, a'; \theta_t\right); \theta_t^- \right). \quad (3)$$

DDQN is the same as for DQN [8], but with the target y_i^{DQN} replaced with y_i^{DDQN}.

3. Ensemble Methods for Deep Reinforcement Learning

As DQN classes use DNNs to approximate the value function, it has strong generalization ability between similar state inputs. The generalization can cause divergence in the case

of repeated bootstrapped temporal difference updates. So we can solve this issue by integrating different versions of the target network.

In contrast to a single classifier, ensemble algorithms in a system have been shown to be more effective. They can lead to a higher accuracy. Bagging, boosting, and Ada Boosting are methods to train multiple classifiers. But in RL, ensemble algorithms are used for representing and learning the value function. They are combined by major voting, Rank Voting, Boltzmann Multiplication, mixture model, and other ensemble methods. If the errors of the single classifiers are not strongly correlated, this can significantly improve the classification accuracy.

3.1. Temporal Ensemble. As described in Section 2.2, the DQN classes of deep reinforcement learning algorithms use a target network with parameters θ^- copied from θ^t every C steps. Temporal Ensemble method is suitable for the algorithms which use a target network for updating and training. Temporal ensemble uses the previous K learned networks to produce the value estimate and builds up $K \in N$ complete networks with K distinct memory buffers. The recent Q-value function is trained according to its own target network $Q(s, a; \theta_t)$. So each one of Q-value functions $Q_1, Q_2, \ldots, Q_k$ represents temporally extended estimate of Q-value function.

Note that the more recent target network is likely to be more accurate at the beginning of the training and the accuracy of the target networks is increasing as the training goes on. So we denote a learning rate parameter $\lambda \in (0, 1]$ here for target network. The weight of ith target network is $w_i = \lambda^{i-1} / \sum_{i=1}^{N} \lambda^{i-1}$.

So the learned Q-value function by temporal ensemble can be described as follows:

$$Q^T(s, a; \theta) = \sum_{i=1}^{N} \left(\frac{\lambda^{i-1} Q_i(s, a; \theta_i)}{\sum_{i=1}^{N} \lambda^{i-1}} \right). \tag{4}$$

As $\lim_{\lambda \longrightarrow 1} \lambda^{i-1} / \sum_{i=1}^{N} \lambda^{i-1} = 1/N$, we can see that the target networks have the same weights when λ equals 1. This formula indicates that the closer the target networks are, the greater the target networks' weight is. As target networks become more accurate, their weights become equal. The loss function remains the same as in DQN and so does the parameter update equation:

$$\begin{aligned} y_i^T &= r_t + \gamma \max_{a'} \sum_{i=1}^{N} w_i Q^T \left(s', a'; \theta^T \right), \\ \theta_{t+1} &\longleftarrow \theta_t + \alpha \left(y_t^T - Q(s_t, a_t; \theta_t) \right) \nabla_\theta Q(s_t, a_t; \theta_t). \end{aligned} \tag{5}$$

In every iteration, the parameters of the oldest ones are removed from the target network buffer and the newest ones are added to the buffer. Note that the Q-value functions are inaccurate at the beginning of training. So the parameter λ may be a function of time and even the state space.

3.2. Ensemble of Target Values. The traditional ensemble reinforcement learning algorithms maintain multiple tabular algorithms in memory space [4, 16], and majority voting, rank voting, Boltzmann addition, and so forth are used to combine these tabular algorithms. But deep reinforcement learning uses neutral networks as function approximators. The use of multiple neural networks is very computationally expensive and inefficient. In contrast to previous researches, we combine different DRL algorithms that learn separate value functions and policies. Therefore in our ensemble approaches we combine the different policies derived from the update targets learned by deep Q-networks, deep Sarsa networks, double deep Q-networks, and other DRL algorithms as follows:

$$\begin{aligned} y_i^{\text{DQN}} &= r_t + \gamma \max_a \widetilde{Q}(s_{t+1}, a; \theta_t^-), \\ y_i^{\text{Sarsa}} &= r_t + \gamma \widetilde{Q}(s_{t+1}, a_{t+1}; \theta_t^-), \\ y_i^{\text{DDQN}} &= r_t + \gamma \widetilde{Q}\left(s_{t+1}, \operatorname*{argmax}_{a'} \widetilde{Q}(s_{t+1}, a_{t+1}; \theta_t); \theta_t^- \right). \end{aligned} \tag{6}$$

Besides these update targets formula, other algorithms based on value function approximators can be also used to combine. The update targets according to the algorithm k at time t will be denoted by $y_t = \sum_{i=1}^{k} \beta_i y_t^i$.

The loss function remains the same as in DQN and so does the parameter update equation:

$$\theta_{t+1} \longleftarrow \theta_t + \alpha \left(y_t^E - Q(s_t, a_t; \theta_t) \right) \nabla_\theta Q(s_t, a_t; \theta_t). \tag{7}$$

3.3. The Ensemble Network Architecture. The temporal and target values ensemble algorithm (TEDQN) is an integrated architecture of the value-based DRL algorithms. As shown in Sections 3.1 and 3.2, the ensemble network architecture has two parts to avoid divergence and improve performance.

The architecture of our ensemble algorithm is shown in Figure 1; these two parts are combined together by evaluated network.

The temporal ensemble stabilizes the training process by reducing the variance of target approximation error [10]. Besides, the ensemble of target values reduces the overestimate and makes better performance by estimating more accurate Q-value. The temporal and target values ensemble algorithm are given by Algorithm 1.

As the ensemble network architecture shares the same input-output interface with standard Q-networks and target networks, we can recycle all learning algorithms with Q-networks to train the ensemble architecture.

4. Experiments

4.1. Experimental Setup. So far, we have carried out our experiments on several classical control and Box2D environments on OpenAI Gym: CartPole-v0, MountainCar-v0, and LunarLander-v2 [15]. We use the same network architecture, learning algorithms, and hyperparameters for all these environments.

We trained the algorithms using 10,000 episodes and used the Adaptive Moment Estimation (Adam) algorithm to minimize the loss with learning rate $\mu = 0.00001$ and set

(1) Initialize action-value network Q with random weights θ
(2) Initialize the target neural network buffer $(Q_i)_{i=1}^{L}$
(3) For episode 1, M do
(4) For $t = 1, T$ do
(5) With probability ε select a random action a_t, otherwise
$a_t = \text{argmax}_a Q(s_t, a; \theta)$
(6) Execute action a_t in environment and observe reward r_t
and next state s_{t+1}, and store transition (s_t, a_t, r_t, s_{t+1}) in D
(7) Sample random *minibatch* of transition (s_t, a_t, r_t, s_{t+1}) from D
(8) set $w_i = \lambda^{i-1} / \sum_{i=1}^{N} \lambda^{i-1}$
(9) Ensemble Q-learner $\widetilde{Q}(s, a; \theta) = \sum_{i=1}^{N} w_i Q_i(s, a; \theta_i)$
(10) set $y_i^{\text{DQN}} = r_t + \gamma \max_a \widetilde{Q}(s_{t+1}, a; \theta_t^-)$
(11) set $y_i^{\text{Sarsa}} = r_t + \gamma \widetilde{Q}(s_{t+1}, a_{t+1}; \theta_t^-)$
(12) set $y_i^{\text{DDQN}} = r_t + \gamma \widetilde{Q}(s_{t+1}, \text{argmax}_{a'} \widetilde{Q}(s_{t+1}, a_{t+1}; \theta_t); \theta_t^-)$
(13) Set $y_i = \{r_j$, if episode terminates at step $j+1$; $\sum_{i=1}^{k} \beta_i y_t^i$, otherwise$\}$
(14) $\theta_i = \underset{\theta}{\text{argmin}}\, E\left[\left(y_{(s,a)}^i - Q(s, a; \theta)\right)^2\right]$
(15) Every C steps reset $\widetilde{Q} = Q$
(16) End for
(17) End for

Algorithm 1: The temporal and target values ensemble algorithm.

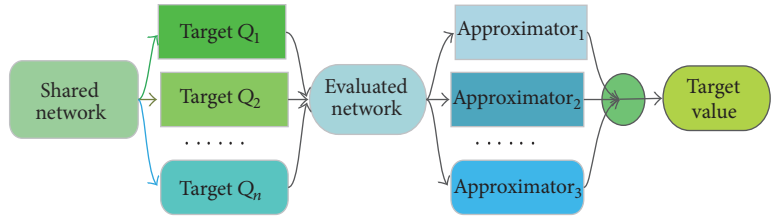

Figure 1: The architecture of the ensemble algorithm.

the batch size to 32. The summary of the configuration is provided below. The target network updated each 300 steps. The behavior policy during training was ε-greedy with ε annealed linearly from 1 to 0.01 over the first five thousands steps and fixed at 0.01 thereafter. We used a replay memory of ten thousands most recent transitions.

We independently executed each method 10 times, respectively, on every task. For each running time, the learned policy will be tested 100 times without exploration noise or prior knowledge by every 100 training episodes to calculate the average scores. We report the mean and standard deviation of the convergence episodes and the scores of the best policy.

4.2. Results and Analysis. We consider three baseline algorithms that use target network and value function approximation, namely, the version of the DQN algorithm from the Nature paper [8], DSN that reduce over estimation [17], and DDQN that substantially improved the state-of-the-art by reducing the overestimation bias with double Q-learning [9].

Using this 10 no-ops performance measure, it is clear that the ensemble network does substantially better than a single network. For comparison we also show results for DQN, DSN, and DDQN. Figure 2 shows the improvement of the ensemble network over the baseline single network of DQN, DSN, and DDQN. Again, we see that the improvements are often very dramatic.

The results in Table 1 show that algorithms we presented can successfully train neural network controllers on the classical control domain on OpenAI Gym. A detailed comparison shows that there are several games in which TE DQN greatly improves upon DQN, DSN, and DDQN. Noteworthy examples include CartPole-v0 (performance has been improved by 13.6%, 79.5%, and 7.8%, and variance has been reduced by 100%, 100%, and 100%), MountainCar-v0 (performance has been improved by 26.7%, 21.2%, and 24.8%, and variance has been reduced by 31.6%, 77.9%, and 8.4%), and LunarLander-v2 (performance has been improved by 28.3%, 32.8%, and 50.5%, and variance has been reduced by 19.2%, 46.4%, and 50.5%).

5. Conclusion

We introduced a new learning architecture, making temporal extension and the ensemble of target values for deep Q learning algorithms, while sharing a common learning module. The new ensemble architecture, in combination with some algorithmic improvements, leads to dramatic improvements

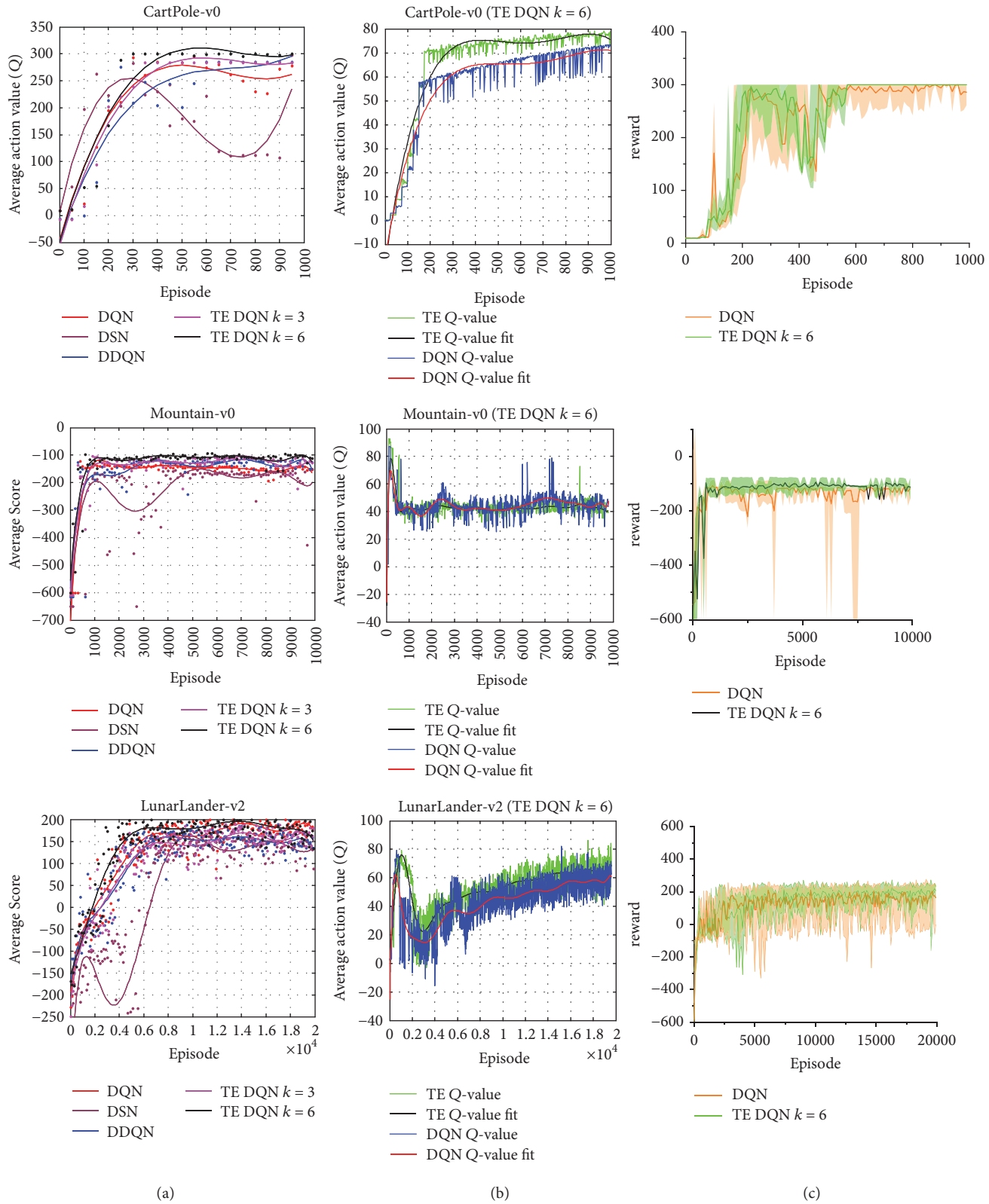

FIGURE 2: Training curves tracking the agent's average score and average predicted action-value. (a) Performance comparison of all algorithms in terms of the average reward on each task. (b) Average predicted action-value on a held-out set of states on each task. Each point on the curve is the average of the action-value Q computed over the held-out set of states. (c) The performance of DQN and TEDQN on each task. The darker line shows the average scores of each algorithm, and the orange shaded area shows the two extreme values of DQN and the green shaded area shows TE DQN.

Table 1: The columns present the average performance of DQN, DSN, DDQN, EDQN, and TE-DQN after 10000 episodes, using ε-greedy policy with $\varepsilon = 0.0001$ after 10000 steps. The standard variation represents the variability over seven independent trials. Average performance improved with the number of averaged networks.

Task (AVG score, Std.)	CartPole-v0	MountainCar-v0	LunarLander-v2
DQN	(264.9, 21.7)	(−148.2, 17.4)	(159.3, 16.7)
DSN	(167.1, 61.6)	(−137.7, 53.9)	(153.9, 25.2)
Double DQN	(278.2, 31.8)	(−144.2, 16.8)	(135.8, 11.8)
TE DQN $K = 3$	(299.1, 1.3)	(−115.6, 21.4)	(186.9, 19.1)
TE DQN $K = 6$	(300, 0)	(−108.4, 11.9)	(204.4, 13.5)

over existing approaches for deep RL in the challenging classical control issues. In practice, this ensemble architecture can be very convenient to integrate the RL methods based on the approximate value function.

Although the ensemble algorithms are superior to a single reinforcement learning algorithm, it is noted that the computational complexity is higher. The experiments also show that the temporal ensemble makes the training process more stable, and the ensemble of a variety of algorithms makes the estimation of the Q-value more accurate. The combination of the two ways enables the training to achieve a stable convergence. This is due to the fact that ensembles improve independent algorithms most if the algorithms predictions are less correlated. So that the output of the Q-network based on the choice of action can achieve balance between exploration and exploitation.

In fact, the independence of the ensemble algorithms and their elements is very important on the performance for ensemble algorithms. In further works, we want to analyze the role of each algorithm and each Q-network in different stages, so as to further enhance the performance of the ensemble algorithm.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] S. Mozer and M. Hasselmo, "Reinforcement learning: an introduction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 16, no. 1, pp. 285-286, 2005.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[3] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing Atari with deep reinforcement learning [EB/OL]," https://arxiv.org/abs/1312.5602.

[4] M. A. Wiering and H. van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 4, pp. 930–936, 2008.

[5] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *Journal of Machine Learning Research (JMLR)*, vol. 7, pp. 877–917, 2006.

[6] P. Preux, S. Girgin, and M. Loth, "Feature discovery in approximate dynamic programming," in *Proceedings of the 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2009*, pp. 109–116, April 2009.

[7] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-Free reinforcement learning with continuous action in practice," in *Proceedings of the 2012 American Control Conference, ACC 2012*, pp. 2177–2182, June 2012.

[8] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 2094–2100, February 2016.

[10] O. Anschel, N. Baram, N. Shimkin et al., "Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning [EB/OL]," https://arxiv.org/abs/1611.01929.

[11] I. Osband, C. Blundell, A. Pritzel et al., "Deep Exploration via Bootstrapped DQN [EB/OL]," https://arxiv.org/abs/1602.04621.

[12] S. Faußer and F. Schwenker, "Ensemble Methods for Reinforcement Learning with Function Approximation," in *Multiple Classifier Systems*, pp. 56–65, Springer, Berlin, Germany, 2011.

[13] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[14] T. Schaul, J. Quan, I. Antonoglou et al., "Prioritized Experience Replay [EB/OL]," https://arxiv.org/abs/1511.05952.

[15] I. Zamora, N. G. Lopez, V. M. Vilches et al., "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo [EB/OL]," https://arxiv.org/abs/1608.05742.

[16] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research (JMLR)*, vol. 6, no. 2, pp. 503–556, 2005.

[17] M. Ganger, E. Duryea, and W. Hu, "Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning," *Journal of Data Analysis and Information Processing*, vol. 04, no. 04, pp. 159–176, 2016.