Neurocomputing 408 (2020) 157–168

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Accelerating deep reinforcement learning model for game strategy

Yifan Li [a], Yuchun Fang [a,*], Zahid Akhtar [b]

[a] School of Computer Engineering and Science, Shanghai University, China
[b] Department of Computer Science, University of Memphis, USA.

ARTICLE INFO

Article history:
Received 1 February 2019
Revised 9 May 2019
Accepted 11 June 2019
Available online 14 March 2020

Keywords:
Deep reinforcement learning
Convolutional neural network
Depthwise separable convolution
Binary weight network

ABSTRACT

In recent years, deep reinforcement learning has achieved impressing accuracies in games compared with traditional methods. Prior schemes utilized Convolutional Neural Networks (CNNs) or Long Short-Term Memory networks (LSTMs) to improve the performances of the agents. In this paper, we consider the issue from a different perspective when the training and inference of deep reinforcement learning are required to be performed with limited computing resources. Mainly, we propose two efficient neural network architectures of deep reinforcement learning: Light-Q-Network (LQN) and Binary-Q-Network (BQN). In LQN, The depth-wise separable CNNs are utilized in memory and computation saving. While, in BQN, the weights of convolutional layers are binary that help in shortening the training time and reduce memory consumption. We evaluate our approach on Atari 2600 domain and StarCraft II mini-games. The results demonstratethe efficiency of the proposed architectures. Though performances of agents in most games are still super-human, the proposed methods advance the agent from sub to super-human performance in particular games. Also, we empirically find that non-standard convolution and non-full-precision networks do not affect agent learning game strategy.

1. Introduction

The DeepMind team combined deep learning with perceptual capabilities and reinforcement learning with decision-making capabilities, and proposed deep reinforcement learning [1], forming a new research direction in the field of artificial intelligence.

Deep learning [2] originates from the artificial neural network. It is inspired by the hierarchical network structure of inferential data onto the human brain. Researchers proposed multi-layer perceptron and back propagation algorithm [3], which lays the basis of learning rules of deep learning. With the development of data and computing devices, deep learning has made great progress in the fields of image analysis [4,5], video analysis [6], natural language processing [7,8], and speech recognition [9,10]. The basic idea of deep learning is to process data through multilayered network structures and nonlinear transformations to form abstract high-level features and expose distributed data representations [11]. Therefore, deep learning focuses on the perception and expression of data.

Reinforcement learning is inspired by the organism's ability to respond to environmental stress and effective adaptation. It uses trial and error mechanisms to interact with the surrounding environment and uses learning strategies that maximize cumulative rewards to learn optimal strategies [12], which is similar to adaptive fuzzy control [13,14]. It has been applied for robot control [15], games [16,17], simulation [18], industrial manufacturing [19], optimization and scheduling [20,21] and other fields. The goal of reinforcement learning is to obtain maximum cumulative rewards. In order to achieve this goal, on the one hand, it needs to "explore". It explores the environment during the learning process, fully grasp the environmental information, then find a status with higher reward, and on the other hand, it needs to "use". It uses the historical experience that has been learned to select the highest-reward action and shift the entire system to a better state. Therefore, reinforcement learning focuses on strategies for learning to solve problems.

Traditional reinforcement learning is limited to small action space and sample space, and these spaces are generally discrete. However, more complex and more realistic tasks often have a large state space and continuous action space. When the input data are images and sounds, the data samples often have a high dimension, and the traditional reinforcement learning is difficult to deal with. Deep learning has a strong ability to perceive expression. Reinforcement learning has strong decision-making ability, but its ability is limited in perception about high dimension data. Therefore, the combination of the two has complementary advantages and provides ideas about solving the cognitive and decision-making problems in complex environments. In many challenging areas, the DeepMind team constructed and implemented deep reinforcement

* Corresponding author.
E-mail address: ycfang@shu.edu.cn (Y. Fang).

https://doi.org/10.1016/j.neucom.2019.06.110
0925-2312/

learning models at the human expert level [1,22–24]. These models constructed the knowledge system and learned the environment directly from the original input signal, without any related domain knowledge. Deep reinforcement learning is a very versatile end-to-end sensing and control system. At present, deep reinforcement learning has been applied in games [1,22,25], machine vision [26,27], robot control [28–30], parameter optimization [31,32] and other fields.

Although the previous models of deep reinforcement learning perform well on expensive GPU-based machines, they are usually not suitable for small devices such as embedded electronic devices, which limit the application of reinforcement learning in the real world. For example, the most famous agents of reinforcement learning: AlphaGo Zero and AlphaGo Master played with 4 TPUs; AlphaGo Fan and AlphaGo Lee were played over 176 GPUs and 48 TPUs [24]. These agents overtax the limited storage and compute capabilities of smaller devices.

In order to solve the inference problem of agents on small devices, we propose two simple approaches in this paper, called Light-Q-Network (LQN) and Binary-Q-Network (BQN), which are respectively inspired by the MobileNets [33] and Binary-Weight-Networks [34]. In our previous work [35], we discover the depthwise separable CNN can shorten the training time of the agent on Starcraft II Reinforcement Learning Environment [36]. It is very attractive to explore the possibility of running agents on small devices. As an extension to our previous work [35], the main contributions of this paper are as follows:

- We discover that the depthwise separable convolution has no degradation effect in game performance and can be generally applicable to the deep reinforcement learning environment.
- We propose a neural network architecture of deep reinforcement learning with binary weights. The weights of convolutional and fully connected layers are binary values in the network, which can enable agents to maintain superhuman level game performance while generating different game strategies.
- We analyze the contributions of the models. We find that the proposed accelerating methods can keep the effective learning of game strategies and result in several different but equally effective and feasible game strategies.
- By experiments on the Atari 2600 domain, we demonstrate our methods can obtain similar game performances compared to standard methods.

1.1. Background

For convenient description, we first summarize the mathematical background of deep reinforcement learning, such as Markov Decision Processes and deep Q-learning.

Reinforcement Learning and Markov Decision Processes A Markov Process or Markov Chain is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$, where $\mathcal{S}$ is a finite set of states of the environment sensed by the agent and $\mathcal{P}$ is a state transition probability matrix from s to s'. Assuming that the expectation of reward from state s to s' is $\mathcal{R}$ and a Markov reward process can be defined by a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\gamma \in [0, 1]$ is a discount factor, and $\mathcal{R}$ denotes rewards that the environment feeds a reward back to the agent at the time of state transition. After adding action set $\mathcal{A}$ taken by the agent, Markov reward process becomes Markov decision process, denoted as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. A policy π is the probability of an agent taking action in a given state.

Given an Markov decision process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π, the state sequence $s_1, s_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$ and the state-reward sequence $(s_1, r_2), (s_2, r_3), \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$.

The state-value function $V_{\pi}(s)$ of Markov decision processes is the expectation of return starting from state s, following policy π. It is defined as:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | \mathcal{S}_t = s] \tag{1}$$

where the G_t is the total discounted reward from step t: $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \Sigma_{i=0}^{\infty} \gamma^i R_{t+i+1}$.

The action-value function $Q_{\pi}(s, a)$ of Markov decision processes is the expectation of return starting from state s, taking action a and following policy π. It is defined as:

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | \mathcal{S}_t = s, \mathcal{A}_t = a] \tag{2}$$

Deep Q-learning The deep neural networks are used in Deep Reinforcement Learning as function approximators. Deep Q-Network [37] is a typical example. The Deep Q-Network optimizes the loss function Eq. (3) with the gradient in Eq. (4).

$$L^{Q}(\theta) = (R + \gamma \underset{a'}{max} Q(s', a'; \theta') - Q(s, a; \theta))^2 \tag{3}$$

$$\nabla L^{Q}(\theta) = 2(R + \gamma \underset{a'}{max} Q(s', a'; \theta') - Q(s, a; \theta)) \tag{4}$$

where (s, a, r, s') is sampled from the replay buffer randomly, θ is optimized by the on-line agent, and θ' is the fixed parameters of the off-line network. The DQN agent acts with a ϵ-greedy policy that is used to select the action preferentially according to the action-value function Q, or take random action with probability ϵ.

2. Related work

The efficiency of sampling in deep reinforcement learning is extremely low, which leads to the long training time of agents. Several methods have been proposed to solve efficient training and inference in deep reinforcement learning by designing improved control and algorithm.

Asynchronous advantage actor-critic The Asynchronous Advantage Actor-Critic (A3C) is proposed in [23]. It can run multiple agents in parallel. Each agent has its own copy of the environment. All the agents use their own samples when updating the policy. Different agents may encounter different states and transitions to avoid data correlation. A3C has greatly shortened the learning time of deep reinforcement learning by making full use of hardware resources. It is a policy-based online learning algorithm. The gradient of the loss of the A3C policy which learns with n-step frames is shown in Eq. (5)

$$\begin{aligned} \nabla_{\theta} L^{\pi}(\theta) = -\mathbb{E}^{\pi}\Big[& \Sigma_{i=0}^{n} \nabla_{\theta} \log(\pi(a_{t+i}|s_{t+i}; \theta))(G_t(i) - V(s_{t+i}; \theta)) \\ & + \beta \Sigma_{i=0}^{n} \nabla_{\theta} H(\pi(\cdot|s_{t+i}; \theta)) \\ & - \eta \Sigma_{i=0}^{n} (G_t(i) - V(s_{t+i}; \theta)) \nabla_{\theta} V(s_{t+i}; \theta) \Big] \end{aligned} \tag{5}$$

where $H(\pi(\cdot|s_{t+i}; \theta))$ is the entropy of the policy π and β is a hyperparameter, which controls the strength of the entropy term. $G_t(i)$ means the rewards obtained by executing the policy π in state s_{t+i}: $G_t(i) = \Sigma_{j=i}^{n} \gamma^{j-i} r_{t+j} + \gamma^{n-i} V(s_{t+n}; \theta)$, where $r_t + j$ is the reward at step $t + j$ and $V(s; \theta)$ is the value of agent's estimate by state s.

Proximal policy optimization Proximal Policy Optimization (PPO) [38] is based on the Trust Region Policy Optimization (TRPO) [39]. The TRPO modified the policy gradient methods. When maximizing the objective function, a constraint is needed to limit the update of the policy so that the KL distance between the old and new policy does not exceed a certain threshold. As shown in Eq. (6)

$$\underset{\theta}{maximize}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \quad subject\ to$$
$$\hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(\cdot|s), \pi_\theta(\cdot|s)]] \le \delta \tag{6}$$

where $\hat{A}_t$ represents the advantage of an action, s means the state, a means the action and $\pi_{\theta_{old}}$ means the policy with old parameters.

The PPO achieves a balance among ease of implementation, sampling complexity, and effort required for debugging. PPO uses $r_t(\theta)$ to represent the ratio of probability distribution $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. So the objective function of PPO is as in Eq. (7).

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{7}$$

where ϵ is a hyperparameter.

When $\hat{A}_t > 0$, the choice of the current action is better, but the range of the policy's update should not be too large. If the ratio is greater than $1+\epsilon$, there is an upper bound value for the value of the objective function. When $\hat{A}_t < 0$, it means that the current action is not well chosen. Similarly, the policy of choosing this action needs to be reduced, but the reduction should not be too large. When the ratio is less than $1-\epsilon$, it is clipped to make the value of objective function great than or equal the lower bound value. It reduces training time from the optimization method of the objective function.

Unsupervised reinforcement and auxiliary learning Unsupervised reinforcement and auxiliary learning (UNREAL) [40] based on the A3C is another way to train deep reinforcement learning tasks. It trains multiple auxiliary tasks while training the A3C model. The multiple tasks complement each other to accomplish the goals. It includes two kinds of auxiliary tasks, auxiliary control tasks and auxiliary reward tasks.

The auxiliary control of UNREAL includes pixel change tasks and network feature tasks. In pixel change tasks, agents learn a separate policy to maximize the changes of pixels in each cell of n*n non-overlapping grids. And agents learn a separate policy to maximize the activation of each unit in a particular hidden layer in network features task. In auxiliary reward tasks, UNREAL introduces a reward prediction task to learn values that focus on immediate rewards. The task predicts the beginning of immediate rewards with historical frames. This task involves processing a sequence of observations and requiring agents to predict subsequent rewards.

The loss of UNREAL policy is:

$$L(\theta) = L_{A3C} + \alpha L_{VR} + \beta L_Q + \eta L_{RP} \tag{8}$$

where the L_{A3C} is the loss function proposed in [23], and L_Q is the same as in Eq. (3). $L_{VR} = (G_t - V(s_t;\theta))^2$, and $L_{RP} = (R_t - V(s_t;\theta))^2$. α, β and η are the scale factors.

The L_{A3C} is optimized by on-policy, while the value function loss L_{VR}, the auxiliary control loss L_Q and the auxiliary reward loss L_{RP} are optimized by off-policy using replayed data. The UNREAL accelerates learning speed and improves performance by setting multiple auxiliary tasks. Auxiliary tasks and target tasks are trained simultaneously by sharing weights.

Hierarchical critics Hierarchical Critics method is proposed in Hierarchical Critic Assignment for Multi-agent Reinforcement Learning [41]. It introduces multiple cooperative critics in two-level hierarchies. Agents not only receive low-level details but also obtain global information from high-level, which improves performance and speeds up the learning process. Hierarchical Critics method increases the perception of each agent in the global environment through a global critic. It shows outstanding performance in Multi-Agent Reinforcement Learning field.

Table 1
The architecture of Light-Q-Network.

Type / Stride	Kernel shape	Input shape
Conv dw / s4	8 * 8 * input channel dw	84 * 84 * input channel
Conv / s4	1 * 1 * input channel * 32	20 * 20 * input channel
Conv dw / s2	4 * 4 * 32 dw	20 * 20 * 32
Conv / s1	1 * 1 * 32 * 64	9 * 9 * 64
Conv dw / s1	3 * 3 * 64 dw	9 * 9 * 64
Conv / s1	1 * 1 * 64 * 64	7 * 7 * 64

3. The network architecture

Considering the structure of the convolution kernel, we represent two neural network architectures with fewer parameters. In LQN, the convolutional layers are depthwise separable convolutional layers, whereas in BQN, the weights of the network are binary.

3.1. Light-Q-Network

Depthwise separable convolutions in MobileNets [33] can significantly reduce the computational cost of convolution operations. A CNN serves to observe the environment in the deep reinforcement learning model. We introduce depthwise separable convolutions into deep reinforcement learning, and test the speed of agents of the LQN model.

Depthwise separable convolution decomposes the standard convolution operation into a depthwise convolution operation and a pointwise convolution operation. Depthwise convolution executes convolution operations on each channel of the features separately. It does not involve convolution operations among channels. Pointwise convolution is used to combine the features among channels.

The standard convolution is directly performed to all input data. The computational complexity can be calculated as in Eq. (9).

$$H_k * W_k * N_c * N_k * H_f * W_f. \tag{9}$$

In depthwise separable convolution, the computational complexity of depthwise convolution is as in Eq. (10)

$$H_k * W_k * N_c * H_f * W_f. \tag{10}$$

In depthwise separable convolution, the computational complexity of pointwise convolution is as in Eq. (11)

$$N_c * N_k * H_f * W_f. \tag{11}$$

The ratio of the depth separable convolution to the standard convolution is calculated in Eq. (12)

$$\frac{H_k * W_k * N_c * H_f * W_f + N_c * N_k * H_f * W_f}{H_k * W_k * N_c * N_k * H_f * W_f} = \frac{1}{N_k} + \frac{1}{H_k * W_k} \tag{12}$$

where H_k and W_k are the height and width of convolution kernels, N_c is the number of input channels, and N_k is the number of convolution kernels. H_f and W_f are the height and width of the input feature maps.

The network architecture of the convolution part is shown in Table 1 and the architecture of LQN is shown in Fig. 1. Each convolution layer is followed by a ReLU activation function. We remove the batch normalization between the convolution layer and the activation function.

3.2. Binary-Q-Network

Another efficient method of the neural network is binary-weight-network [34]. In the binary-weight-network, all the values of networks weights are replaced by binary values. CNNs with binary weights are much smaller than equivalent networks with

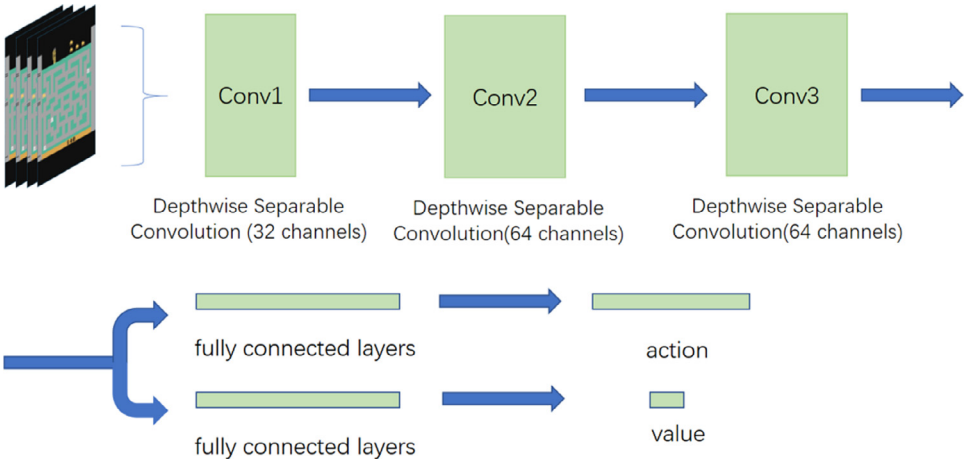


Fig. 1. The architecture of Light-Q-Network.

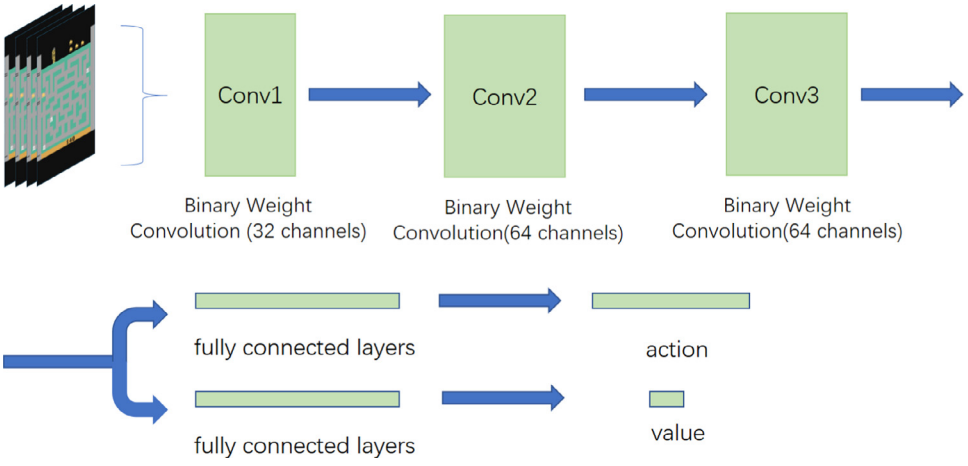


Fig. 2. The architecture of Binary-Q-Network.

Table 2
The architecture of Binary-Q-Network.

Type / Stride	Kernel shape	Input shape
Binary Conv / s4	8 * 8 * input channel * 32	84 * 84 * input channel
Binary Conv / s2	4 * 4 * 32 * 64	20 * 20 * 32
Binary Conv / s1	3 * 3 * 64 * 64	9 * 9 * 64

single-precision weights (about 32 times). In addition, when the weight value is binary, convolution can be calculated only by addition and subtraction (without multiplication), which can increase the operation speed by about two times. We introduce binary weight convolutions into deep reinforcement learning and test the speed of the agents in BQN. The convolution operation can be approximated by Eq. (13).

$$X * W \approx \alpha(X \oplus B) \tag{13}$$

where $\oplus$ represents a convolutional operation without multiplication. X is input. B is binary weights. $B \in \{-1, +1\}$ and α is a scaling factor. The optimal B is $B^* = sign(W)$ and optimal α is $\frac{1}{n}\|W\|_{\ell 1}$

The BQN contains binary convolutional layers, as shown in Table 2. The architecture of BQN is shown in Fig. 2. Each convolution layer is followed by a ReLU activation function. The binary operation of the fully connected layer is only used at the inference stage.

4. Experiments

We evaluate the performance of LQN and BQN agents on Atari games [42] and StarCraft II mini games [36]. We compare our methods with state-of-the-art DQN baselines: Rainbow [43]. We train our models on 10 Atari games and 3 StarCraft II mini games with 15M frames. In each case, we use the same hyper-parameters. We use Adam [44] optimizer with an initial learning rate of 0.0000625, and the epsilon of the optimizer is 0.00015. For each environment, the number of consecutive states processed is 4. We also use a multi-step return parameter of 3, and a discount factor of the reward of 0.99. In our model, the discretized size of value distribution is 51. The minimum value distribution support is -10, and the maximum value distribution support is 10. The initial standard deviation of noisy linear layers is 0.1. For the replay buffer, the capacity is 1,000,000, and the frequency of sampling from memory is 4. We also use a prioritized experience replay exponent of 0.5, and we set the initial importance sampling weight 0.4. In the training phase, the number of steps after which to update the target network is 32,000 and the batch size is 32.

We compare the performance of agents using the human-normalized score function in Eq. (14).

$$100\% * \frac{Score_{Agent} - Score_{Random}}{Score_{Human} - Score_{Random}} \tag{14}$$

where human and random scores are the same as in [45]. We compare the performance of our agents with the performance of base-

Table 3
Human-normalized performance in Atari game.

Games	Rainbow [43]	Rainbow our implementation	A3C [23]	PPO [38]	LQN	BQN
Assault	2689.78%	2432.26%	1010.87%	914.07%	**1765.82%**	796.44%
Bank Heist	181.86%	95.52%	129.37%	**171.39%**	106.35%	107.70%
Breakout	1443.75%	1398.96%	**2361.81%**	948.26%	1390.97%	1336.81%
Chopper Command	240.89%	176.06%	94.42%	41.13%	**155.23%**	**95.47%**
Defender	330.27%	5125.88%	339.30%	-	**5347.20%**	**5442.04%**
Demon Attack	6104.41%	5265.54%	**6221.14%**	617.20%	4655.23%	1141.32%
Free Way	132.94%	132.94%	0.00%	127.06%	**132.94%**	**127.84%**
Kangaroo	488.95%	501.11%	1.41%	331.10%	**497.75%**	**471.61%**
Pong	117.85%	118.13%	74.50%	117.28%	**118.13%**	**118.13%**
Space Invaders	1572.84%	1889.90%	**1024.69%**	52.25%	150.06%	84.63%
Mean	1295.65%	1542.82%	1125.75%	368.86%	**1431.97%**	972.20%
Median	409.61%	341.45%	234.33%	251.24%	**326.49%**	**299.72%**

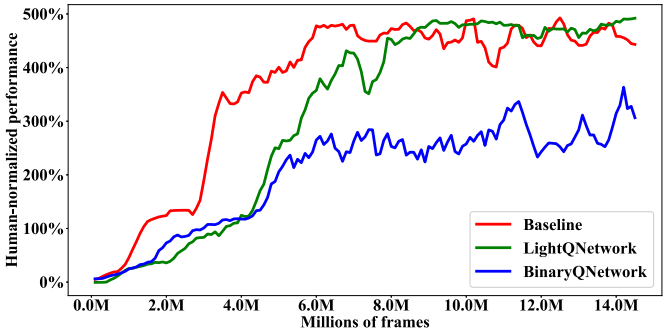


Fig. 3. Median human-normalized performance across Atari games. We compare our agents (green and blue) to Rainbow DQN (red) baseline. Note that the LQN nearly matches the rainbow at 8M frames. The curves are smoothed with a moving average over 6 points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

line by the baseline-normalized function in Eq. (15).

$$100\% * \frac{Score_{Agent} - Score_{Random}}{Score_{Baseline} - Score_{Random}} \tag{15}$$

We compare the inference speed of agents using the score function in Eq. (16).

$$100\% * \frac{Time_{Agent}}{Time_{Baseline}} \tag{16}$$

4.1. Atari games

The Atari 2600 was a popular American game console in the 1980s. The Arcade Learning Environment [42] is a software framework designed to make it easier to develop game agents. It includes a total of 49 independent games, such as Breakout, Galaxy Invaders and more.

In Fig. 3, we compare the performance of LQN and BQN to the corresponding performance of Rainbow baseline. The results are measured in terms of the median human normalized score across games.

The game score is the average performance of each game. The normalized scores of human in Atari are shown in Table 3, and the base-normalized scores are shown in Table 4 (Please refer to Table 8 in the Appendix for concrete scores). .As can be seen, the performance of most of our agents is still a super-human level for the majority of games on Atari. Particularly, in some games, the LQN gets similar game performance (e.g., see the results for Freeway in Fig. 4(b) and Pong in Fig. 4(a)). But our BQN does not perform well in some games (e.g., see the results for Demon Attack in Fig. 4(c) and Chopper Command in Fig. 4(d)). LQN and BQN have

Table 4
Baseline-normalized performance in Atari game.

Games	A3C [23]	PPO [38]	LQN	BQN
Assault	41.56%	37.58%	**72.60%**	32.74%
Bank Heist	135.43%	**179.43%**	111.33%	112.75%
Breakout	**168.83%**	67.78%	99.43%	95.56%
Chopper Command	53.63%	23.36%	**88.17%**	54.23%
Defender	6.62%	-	**104.32%**	**106.17%**
Demon Attack	**118.15%**	11.72%	88.41%	21.68%
Free Way	0.00%	95.58%	**100.00%**	96.17%
Kangaroo	0.28%	66.07%	**99.33%**	94.11%
Pong	63.07%	99.28%	**100.00%**	**100.00%**
Space Invaders	**563.66%**	28.74%	82.55%	46.55%
Mean	**115.12%**	67.73%	94.61%	76.00%
Median	58.35%	66.90%	**99.38%**	94.84%

good performance in some games. These games have some common characteristics. In these games, the values of the rewards are bounded and the rewards are easy to obtain. We also discover that all of the games in which our model underperformed are shooting games, with more negative feedback than positive.

Please refer to Fig. 8 in the Appendix for additional games. We evaluate LQN, BQN and Rainbow for 100K frames on 10 games.

4.2. StarCraft II

StarCraft II is a real-time strategy (RTS) game from Blizzard Entertainment. StarCraft II has millions of players and has a regular professional league. The environment has a huge search space. The double match of StarCraft II has a maximum of 400 units. If we use a 128px*128px map, the size of the search space is about 10^{1685} regardless of other details. In the mini game environment, there are 523 non-spatial actions. When the environment is set to 32*32 pixels, there are 1024 spatial actions and the size of the total action space is 535,552. The environment is complicated in several aspects. Because the game has fog of war, it is a game with incomplete information state. Besides, the rewards are severely delayed in the environment, but the environment requires the model to make an immediate decision.

In the minigame under the Starcraft II reinforcement learning environment, we select three maps to train deep reinforcement learning model. The human-normalized performance is shown for comparison in Table 5 (Please refer to Table 9 in the Appendix for the concrete scores). We evaluate Light-A3C, Binary-A3C and A3C for 10K frames on 3 games. The deep neural network structures of A3C and DQN are different. In order to make our method adapt to A3C, we make some modifications. The Q-value part of our model is divided into the advantage part and the value part, which correspond to the actor part and the critic part in A3C and PPO respectively.

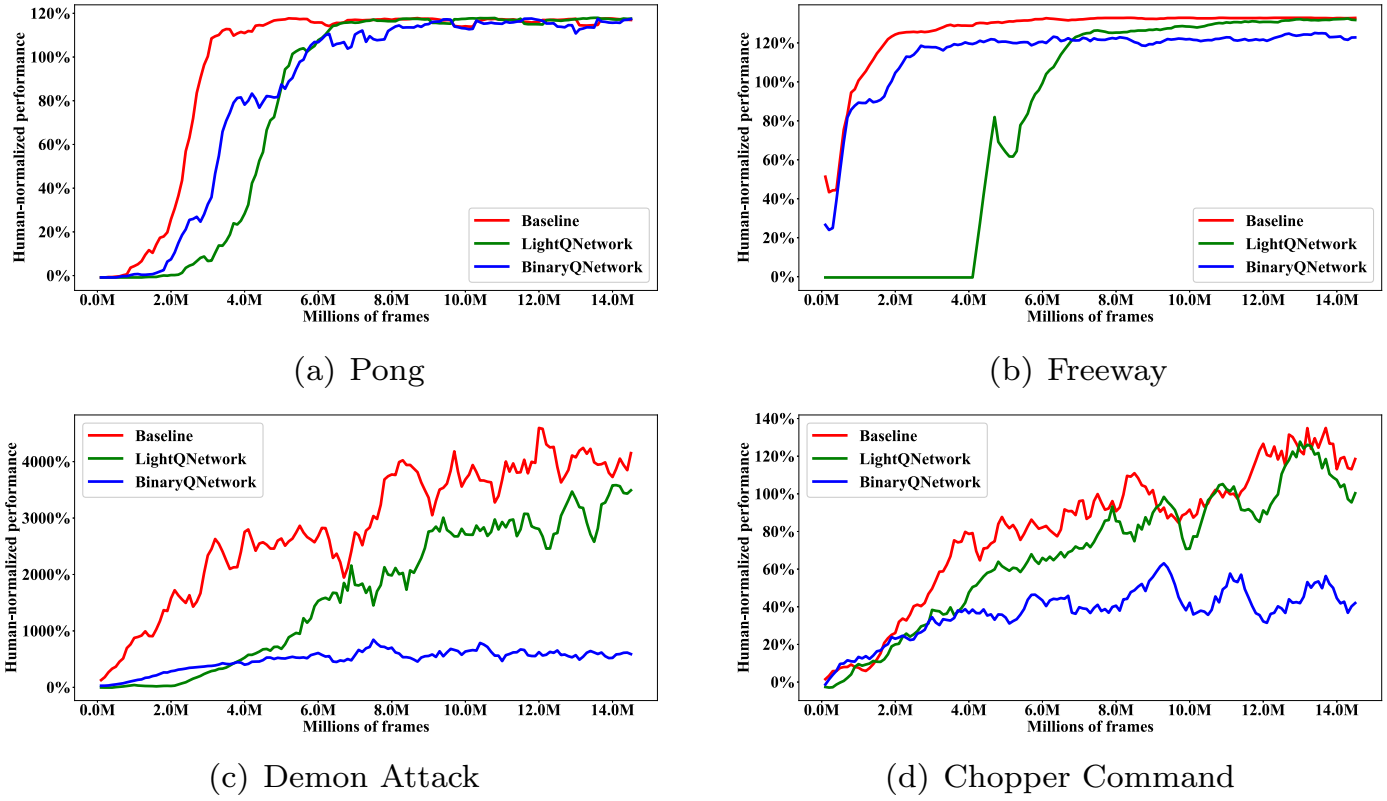


Fig. 4. Human-normalized performance of example Atari games. We compare our agents (green and blue) to Rainbow DQN (red) baseline. The curves are smoothed with a moving average over 6 points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 5
Human-normalized performance in StarCraft II minigame.

Games	A3C [36]	Our Light-A3C	Our Binary-A3C
DefeatRoaches	438%	403%	361%
FindAndDefeatZerglings	114%	110%	104%
DefeatZerglingsAndBanelings	33%	31%	28%

4.3. Game strategy

In addition to game performance, we also consider the game strategy adopted by the agent in the game. We believe that the strategy learned by the agent is more important than the simple game score, because agents can get very high scores by cheating in some games. For instance, two rewards are set in game CoastRunners. One is for completing the game, and the other is for collecting the scoring goals in the environment. Finally, the agent finds an isolated lagoon and collects the target score repeatedly. In such a case, the game is not over, but the agent can score more [46].

In the Atari games, our agents have learned the game strategies. In game Breakout in Fig. 5, Rainbow 5(a), LQN 5(b) and BQN 5(c) can learn to break somewhere to get higher scores. In game Freeway, Rainbow 5(d), LQN 5(e) and BQN 5(f) can learn to cross the road and avoid car crashes to get high scores even when there are many cars on the road. In game Kangaroo, Rainbow 5(g), LQN 5(h) and BQN 5(i) can learn some skills to get high scores. Rainbow and LQN can learn to get high scores by killing enemies at the lower right corner of the game, while BQN can learn to get high scores at the lower left corner of the game by avoiding enemy attacks and picking up props.

In StarCraft mini game, all of the agents can learn the game strategies to achieve better performance. In the game "Defeat Roaches", the agent can learn the strategy of "siege", in which the marines first kill an enemy unit and kill one by one, as shown in Fig. 6(a) and Fig. 6(b). In the game "Find And Defeat Zerglings", the agent can learn the strategy of "searching", in which the marines explore unknown positions in clockwise order, as shown in Fig. 6(c) and Fig. 6(d). In the game "Defeat Zerglings And Banelings", the agent can learn the strategy of "attacking by order", in which the marines first kill enemy units with higher damage ability, and then kill other enemy units, as shown in Fig. 6(e) and Fig. 6(f).

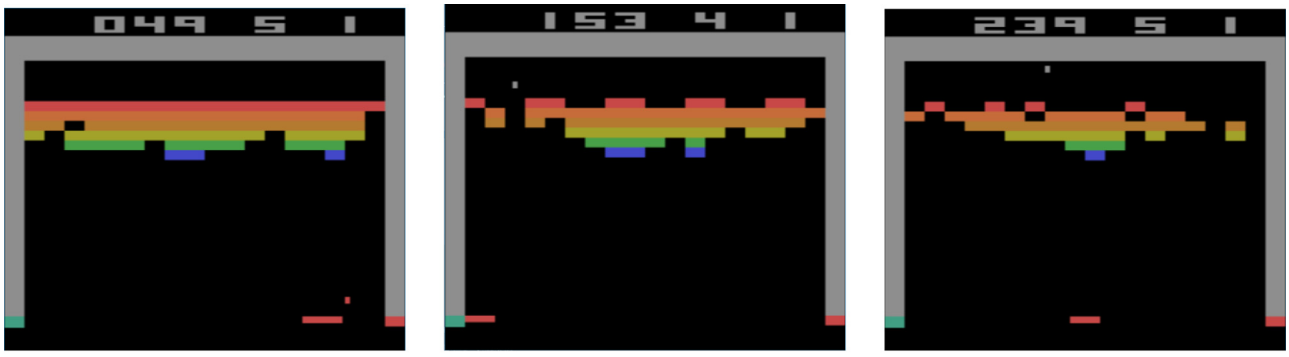
4.4. Efficiency analysis

In the convolution operation, the memory usage of the model weight parameter is calculated as $K^{2*}C_{in}{}^{*}C_{out}$, where K is the kernel size of the convolutional layer and C is the number of channels. Fig. 7 shows the required memory for three architectures which is normalized by the baseline. In Table 6, we compare the scores and training time with several other methods.

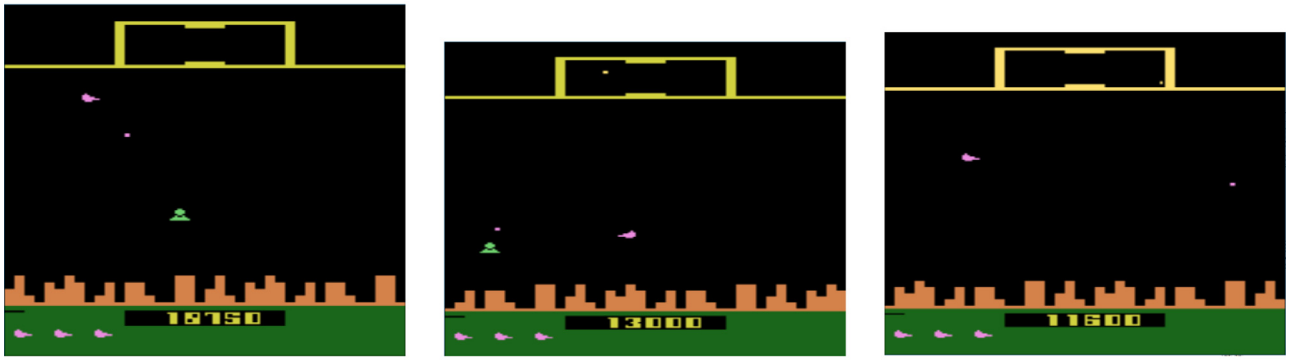
For our method, there are four fully-connected layers with huge parameter quantities behind the convolutional layers. The advantage of our methods is limited on GPU platforms (such as 1080Ti). According to the theory of the roofline model [47], the calculation of standard convolution operation is difficult on the platform with low computational intensity, and our model can also perform well on these platforms. Especially, if the computing platform has an optimization mechanism in bit operations, BQN can perform better.

4.5. Ablation studies

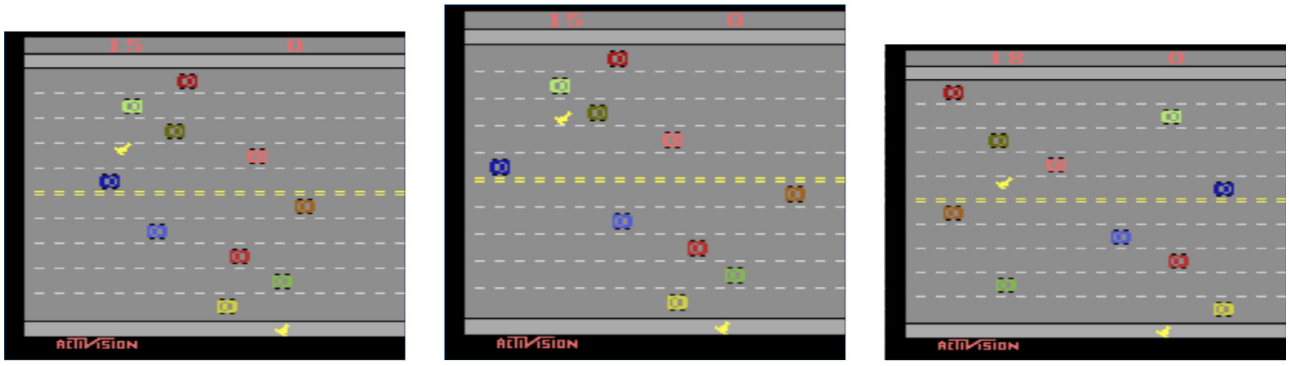
There are some key differences between our convolutional layers and the previous convolutional layers used in classification. For



(a) Breakout (Rainbow) (b) Breakout (LQN) (c) Breakout (BQN)



(d) Defender (Rainbow) (e) Defender (LQN) (f) Defender (BQN)



(g) Freeway (Rainbow) (h) Freeway (LQN) (i) Freeway (BQN)

(j) Kangaroo (Rainbow) (k) Kangaroo (LQN) (l) Kangaroo (BQN)

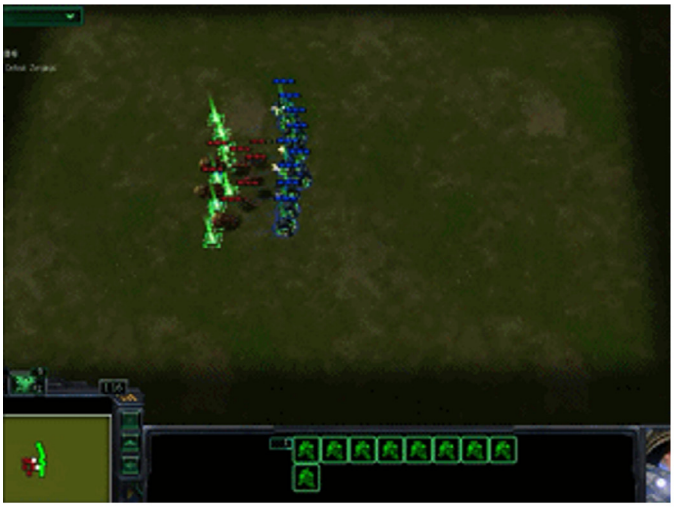
Fig. 5. Atari game strategy.

(a) Defeat Roaches 1

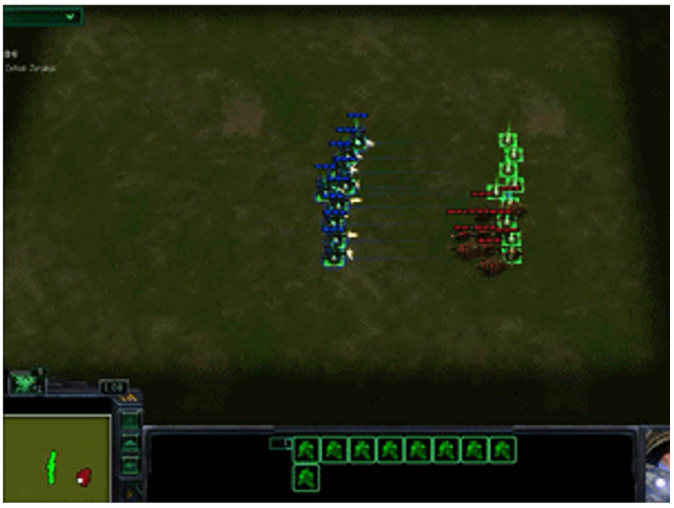
(b) Defeat Roaches 2

(c) Find and Defeat Zerglings 1

(d) Find and Defeat Zerglings 2



(e) Defeat Zerglings and Banelings 1



(f) Defeat Zerglings and Banelings 2

Fig. 6. StarCraft II mini game strategy.

Table 6
Training time on Atari games.

Methods	Mean performance	Median performance	Average training time
Baseline(15M Frames)	1542.82%	341.45%	3d 13h 28m
A3C(40M Frames) [36]	1125.75%	234.33%	4d
PPO(40M Frames)	368.86%	251.24%	-
LQN(15M Frames)	1431.97%	326.49%	2d 15h 57m
BQN(15M Frames)	972.20%	299.72%	2d 34m

Table 7
Ablation studies on Atari with baseline-normalized performance.

(a) LQN

Convolution block structure	Mean	Median	Average training time (15M Frames)
With BN	93.32%	96.17%	4d 1h 5m
Without BN	**94.61%**	**99.38%**	**4d 57m**

(b) BQN

Active Function	Mean	Median	Average Training Time (15M Frames)
Binary-Active	74.27%	93.49%	**3d 22h 56m**
ReLU	**76.00%**	**94.84%**	3d 23h 2m

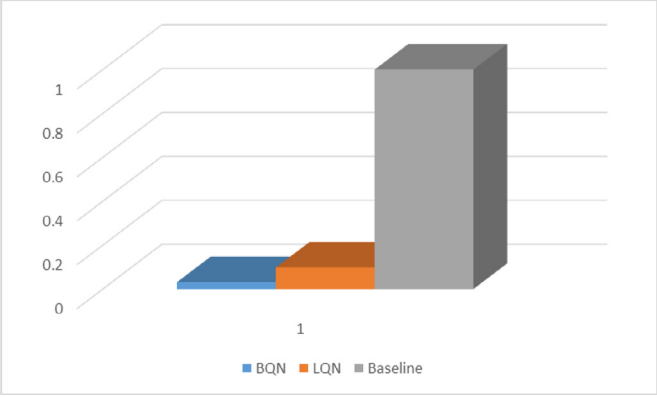


Fig. 7. Required memory of three different CNN architectures.

LQN, we do not use the batch normalization layer between the convolution layer and the activation function. For BQN, we use the ReLU active function instead of Binary-Active proposed in [34]. In Table 7(a), we compare the performance of a separable convolution network with or without batch normalization operation. In Table 7(b), we compare the performance of two different active functions.

4.6. Discussion

LQN and BQN show their respective advantages in the experiments. The game performance of both is above 100% human-level or close to 100% human level. However, BQN shows surprising performance in game strategy, which is different from game strategy of LQN, but has similar game scores. The LQN compresses the neural network from the structure, and its calculation method is still full precision. The BQN compresses the weights of the neural network without changing the structure of the convolutional network. For low computational intensity devices, the two models have different applications. LQN is more suitable for devices with general architecture. BQN has a good effect on special devices whose bit operations are optimized.

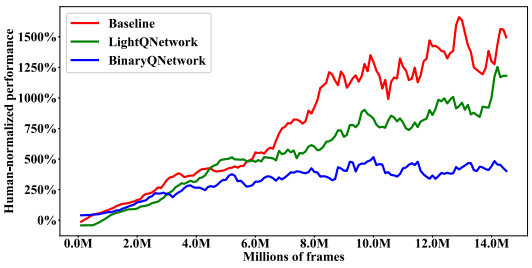
5. Conclusions

We introduce simple, efficient, and accurate neural networks for reinforcement learning. The proposed methods show significant performance and quick inference across many Atari games and StarCraft II mini games. In particular, compared with several CNN models used in deep reinforcement learning, we observe that the inference times of LQN and BQN are shortened. The LQN has no degradation effect on game strategy, except slightly reducing the score of game. Although the performances of the BQN are unsatisfactory, agents can still learn game strategy and can even result in different strategies with similar performances. The proposed methods can be easily extended to reinforcement learning algorithms beyond the baselines considered in this paper. They can also be applied to any deep reinforcement learning algorithms, such as PPO, UNREAL etc. The proposed methods prove the possibility of training or running the inference of deep reinforcement learning agents on small devices in real-time.

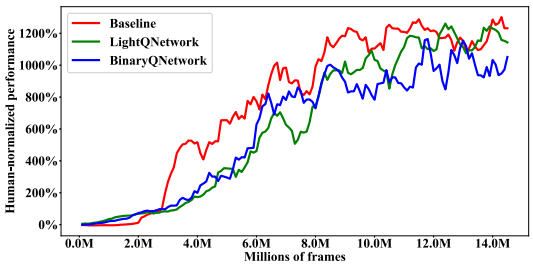
Acknowledgments

The work is supported by the National Natural Science Foundation of China under Grant No.: 61976132, 61170155 and the National Natural Science Foundation of Shanghai under Grant No.: 19ZR1419200. We appreciate the High Performance Computing Center of Shanghai University, and Shanghai Engineering Research Center of Intelligent Computing System (No. 19DZ2252600) for providing the computing resources.

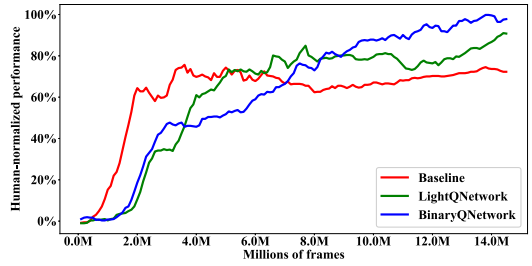
Appendix



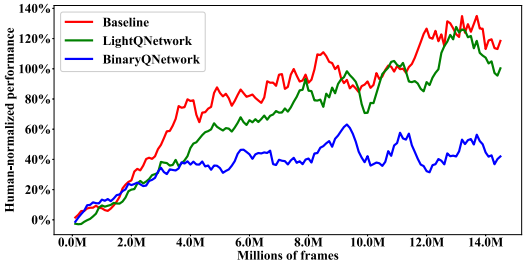
(a) Assault



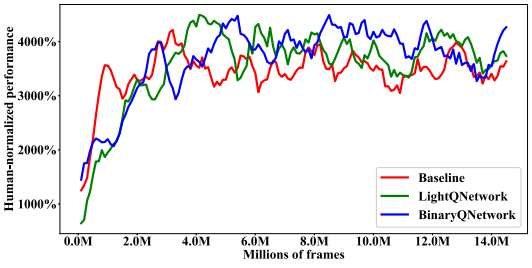
(b) Breakout



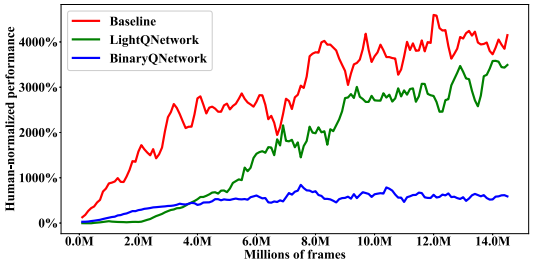
(c) Bank Heist



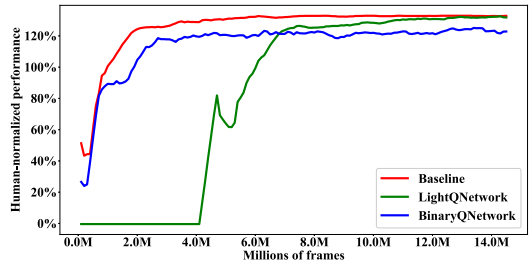
(d) Chopper Command



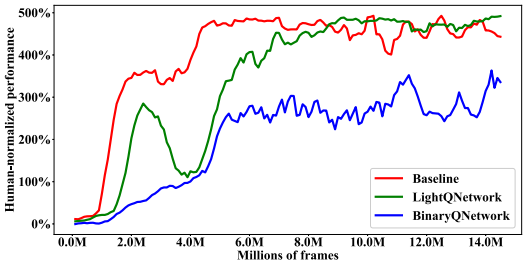
(e) Defender



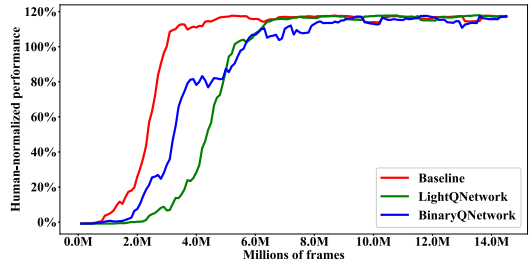
(f) Demon Attack



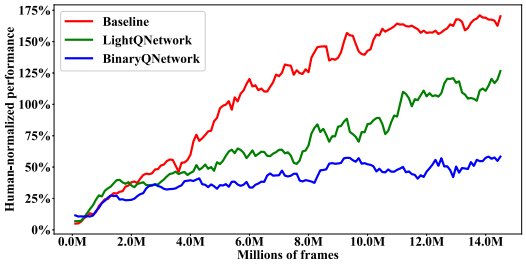
(g) Freeway



(h) Kangaroo



(i) Pong



(j) Space Invaders

Fig. 8. Human-normalized performance for 10 Atari games. We compare our agents (green and blue) with those of Rainbow (red) baseline. All curves are smoothed with a moving average over 6 points. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 8
Performance in Atari game.

Games	Random	Human	Rainbow[43]	Rainbow our implementation	A3C [23]	PPO [38]	Our LQN	Our BQN
Assault	222.4	742	14,198.5	12,860.4	5,474.9	4,971.9	**9, 397.6**	4,360.7
Bank Heist	14.2	753.1	1,358.0	720.0	970.1	**1, 280.6**	800.0	810.0
Breakout	1.7	30.5	417.5	404.6	**681.9**	274.8	402.3	386.7
Chopper Command	811.0	7,387.8	16,654.0	12,390.0	7,021.0	3,516.3	**11, 020.0**	7,090.0
Defender	2,874.5	18,688.9	55,105.0	813,501.0	56,533.0	-	848,502.0	**863, 501.0**
Demon Attack	152.1	1,971.0	111,185.2	95,927.0	**113, 308.4**	11,378.4	84,826.0	20,911.5
Free Way	0.1	25.6	34.0	34.0	0.1	32.5	**34.0**	32.7
Kangaroo	52.0	3,035.0	14,637.5	15,000.0	94.0	9,928.7	**14, 900**	14,120
Pong	-20.7	14.6	20.9	21.0	5.6	20.7	**21**	**21**
Space Invaders	148	1,668.7	18,789	2,912.5	**15, 730.5**	942.5	2,430.0	1,435.0

Table 9
Performance in StarCraft II minigame.

Games	Human	Grandmaster	A3C[36]	Our Light-A3C	Our Binary-A3C
DefeatRoaches	81	363	355	323	293
FindAndDefeatZerglings	49	61	56	54	51
DefeatZerglingsAndBanelings	757	848	251	236	217

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with deep reinforcement learning, arXiv:/1312.5602 (2013).
[2] K. Yu, L. Jia, Y. Chen, W. Xu, Deep learning: yesterday, today, and tomorrow, J. Comput. Res. Dev. 50 (9) (2013) 1799–1804.
[3] Rumelhart, E. David, Hinton, E. Geoffrey, Williams, J. Ronald, Learning representations by back-propagating errors, 1986.
[4] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.
[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.
[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, arXiv:/1406.1078 (2014).
[8] Z. Yang, D.-p. Tao, S.-y. Zhang, L.-w. Jin, Similar handwritten chinese character recognition based on deep neural networks with big data, J. Commun. 35 (9) (2014) 184–189.
[9] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: Acoustics Speech Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 6645–6649.
[10] Y. Li, J. Zhang, D. Pan, D. Hu, A study of speech recognition based on RNN-RBM language model, J. Comput. Res. Dev. 51 (9) (2014) 1936–1944.
[11] Z.-J. Sun, L. Xue, Y.-m. Xu, Z. Wang, Overview of deep learning, Jisuanji Yingyong Yanjiu 29 (8) (2012) 2806–2810.
[12] R.S. Suton, A.G. Barto, Reinforcement learning: An introduction. A bradford book, 2002.
[13] K. Sun, S. Mou, J. Qiu, T. Wang, H. Gao, Adaptive fuzzy control for nontriangular structural stochastic switched nonlinear systems with full state constraints, IEEE Transactions on Fuzzy Systems (2018), doi:10.1109/TFUZZ.2018.2883374. 1–1
[14] J. Qiu, K. Sun, T. Wang, H. Gao, Observer-based fuzzy adaptive event-triggered control for pure-feedback nonlinear systems with prescribed performance, IEEE Transactions on Fuzzy Systems (2019), doi:10.1109/TFUZZ.2019.2895560. 1–1
[15] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey, Int. J. Rob. Res. 32 (11) (2013) 1238–1274.
[16] G. Tesauro, Td-gammon: A self-teaching Backgammon program, in: Applications of Neural Networks, Springer, 1995, pp. 267–285.
[17] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: European Conference on Machine Learning, Springer, 2006, pp. 282–293.
[18] F. Qi-Ming, L. Quan, W. Hui, X. Fei, Y. Jun, L. Jiao, A novel off policy q (λ) algorithm based on linear function approximation, Chin. J. Comput. 37 (3) (2014) 677–686.
[19] Y. Gao, R.-Y. Zhou, H. Wang, Z.-X. Cao, Study on an average reward reinforcement learning algorithm, Chin. J. Comput. 30 (8) (2007) 1372.
[20] Y. Wei, M. Zhao, A reinforcement learning-based approach to dynamic job-shop scheduling, Acta Autom. Sin. 31 (5) (2005) 765.
[21] E. Ipek, O. Mutlu, J.F. Martínez, R. Caruana, Self-optimizing memory controllers: A reinforcement learning approach, in: ACM SIGARCH Computer Architecture News, 36, IEEE Computer Society, 2008, pp. 39–50.
[22] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[23] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937.
[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Mastering the game of go without human knowledge, Nature 550 (7676) (2017) 354.
[25] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.
[26] J. Oh, X. Guo, H. Lee, R.L. Lewis, S. Singh, Action-conditional video prediction using deep networks in Atari games, in: Advances in Neural Information Processing Systems, 2015, pp. 2863–2871.
[27] J.C. Caicedo, S. Lazebnik, Active object localization with deep reinforcement learning, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2488–2496.
[28] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv:/1509.02971 (2015).
[29] Y. Duan, X. Chen, R. Houthooft, J. Schulman, P. Abbeel, Benchmarking deep reinforcement learning for continuous control, in: International Conference on Machine Learning, 2016, pp. 1329–1338.
[30] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, Continuous deep q-learning with model-based acceleration, in: International Conference on Machine Learning, 2016, pp. 2829–2838.
[31] S. Hansen, Using deep q-learning to control optimization hyperparameters, arXiv:/1602.04062 (2016).
[32] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. De Freitas, Learning to learn by gradient descent by gradient descent, in: Advances in Neural Information Processing Systems, 2016, pp. 3981–3989.
[33] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: efficient convolutional neural networks for mobile vision applications, arXiv:/1704.04861 (2017).
[34] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision, Springer, 2016, pp. 525–542.
[35] Y. Li, Y. Fang, Accelerating spatio-temporal deep reinforcement learning model for game strategy, in: International Conference on Neural Information Processing, Springer, 2018, pp. 303–312.
[36] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A.S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, et al., Starcraft ii: a new challenge for reinforcement learning, arXiv:/1708.04782 (2017).
[37] M. Volodymyr, K. Koray, S. David, R. Andrei A, V. Joel, B. Marc G, G. Alex, R. Martin, F. Andreas K, O. Georg, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017.
[39] J. Schulman, S. Levine, P. Moritz, M.I. Jordan, P. Abbeel, Trust region policy optimization, Comput. Sci. (2015) 1889–1897.
[40] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, Reinforcement learning with unsupervised auxiliary tasks, 2016.
[41] Z. Cao, C. Lin, Hierarchical critics assignment for multi-agent reinforcement learning, CoRR (2019) abs/1902.03079.

[42] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (1) (2013) 253–279.
[43] M. Hessel, J. Modayil, H.V. Hasselt, T. Schaul, G. Ostrovski, W. Dabney, H. Dan, B. Piot, M. Azar, D. Silver, Rainbow: Combining improvements in deep reinforcement learning, 2017.
[44] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:/1412.6980 (2014).
[45] Z. Wang, N.D. Freitas, M. Lanctot, Dueling network architectures for deep reinforcement learning, 2015, pp. 1995–2003.
[46] Dario, Jack, Faulty reward functions in the wild, (https://blog.openai.com/faulty-reward-functions).
[47] G. Ofenbeck, R. Steinmann, V. Caparros, D.G. Spampinato, M. Püschel, Applying the roofline model, in: 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 76–85, doi:10.1109/ISPASS.2014.6844463.

Yifan Li, received B.S. degree in Shanghai University, Shanghai, China, in 2018. He is now a graduate student at Shanghai University, Shanghai, China. His research interests include machine learning, reinforcement learning and pattern recognition.

Yuchun Fang, Associate Professor. She gained her Ph.D. from the Institute of Automation, Chinese Academy of Sciences in 2003. From 2003 to 2004, she worked as a post-doctoral researcher at the France National Research Institute on Information and Automation (INRIA). Since 2005, she has worked at the School of Computer Engineering and Sciences, Shanghai University. She is a member of IEEE, ACM, and CCF (Chinese Computer Federation). Her current research interests include multimedia, pattern recognition, machine learning and image processing.

Zahid Akhtar, Research assistant professor in the Department of Computer Science at the University of Memphis, USA. Prior to joining the University of Memphis, he was a Postdoctoral Fellow at INRS-EMT, University of Quebec (Canada), University of Udine (Italy), Bahcesehir University (Turkey), and University of Cagliari (Italy), respectively. His research interests include computer vision, pattern recognition, and machine/deep learning with applications in various fields, including biometrics, affective computing, security systems, and multimedia quality assessment. Dr. Akhtar received a Ph.D. in electronic and computer engineering from the University of Cagliari (Italy). He is a member of the IEEE Signal Processing Society and Association for Computing Machinery.