A Study on the Impact of Obstacle Size on Training Models Based on DQN and DDQN

Siyu Lu[1], *Ye Tao*[2], *Junwei Zeng*[3,*], *Qihuan Zuo*[4],

[1]College of Science, Mathematics and Technology, Wenzhou-Kean University, 325006, China
[2]Waterford Institute, Nanjing University of Information Science and Technology, Nanjing, 210044, China
[3]School of Life Health Information Science and Engineering, Chongqing University of Posts and Telecommunications, 400065, China
[4]School of science, Southwest Petroleum University, 610500, China

Abstract. This paper presents a comparative analysis of Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) algorithms in a simulated car racing environment, focusing on how variations in obstacle density affect each algorithm's learning and trajectory planning, with the goal of enhancing adaptability and safety in autonomous driving systems. In the Gymnasium car racing environment, each episode generates a unique track where obstacles of varying sizes, colours, and quantities are introduced to test the agent's adaptability. Collisions result in immediate penalties and termination of the episode, while avoiding obstacles grants rewards. The model applies penalties to encourage fast completion and minimize the number of frames used. Various parameters such as obstacle size and complexity influence the agent's performance, promoting efficient learning and policy optimization using both DQN and DDQN algorithms under different configurations. After our experiment, comparing with DQN, we found that DDQN could be a better algorithm in car racing scenarios as in some extreme environments, DQN would not very accurately estimate future reward, therefore, agent could not make better decisions because of the misguided evaluation. While DDQN introduces another network for evaluation, which could better decrease overestimation and explore more possible actions to improve training performance. In this experiment, we evaluated both DQN and DDQN algorithms for obstacle avoidance in reinforcement learning, revealing DDQN's superior performance, especially in complex environment. Initially, DQN and DDQN performed similarly with smaller obstacles, but DDQN gained higher rewards as training progressed. In the case of large obstacles, DDQN always outperforms DQN, which proves the enhancement of strategy optimization. Despite these advantages, DDQN also has training fluctuations that indicate areas for future research, such as enhancing stability, exploring multi-agent learning, and adapting to more complex environments.

* Corresponding author: 2020212040@stu.cqupt.edu.cn

1 Introduction

In recent years, the field of autonomous driving has seen tremendous advancements, with significant contributions from artificial intelligence and machine learning. As the world moves towards smarter cities and autonomous vehicles, ensuring the safety and efficiency of these systems becomes increasingly critical. Autonomous driving systems must navigate complex environments, encountering unpredictable obstacles and varying conditions, driving research into reinforcement learning algorithms capable of adapting to dynamic environments.

Among the various AI techniques, reinforcement learning has emerged as a powerful tool for training autonomous agents to make sequential decisions. One of the most notable methods is the Deep Q-Network, which has demonstrated impressive results in applications like game playing and robotic control [1]. However, while DQN offers robust learning capabilities, it faces limitations in complex scenarios such as environments with frequent changes, like varying numbers of obstacles on a racetrack [2].

The effectiveness of DQN is hindered by the overestimation of action values, particularly in high-variability environments [3]. To address this, the Double Deep Q-Network was introduced to reduce the overestimation bias by decoupling the action selection and evaluation processes [4]. This modification has shown promise in improving learning stability, especially where the agent must adapt to frequent changes [5].

In car racing simulations, which reflect broader autonomous driving challenges, the ability of an algorithm to adapt to varying obstacle densities is crucial. A car's trajectory planning not only determines its success on the track but also correlates with the safety and efficiency of real-world driving models. Therefore, analyzing how different reinforcement learning algorithms respond to changing obstacle density provides insights for developing more robust autonomous systems [6].

This paper focuses on a comparative analysis of DQN and DDQN in car racing, emphasizing how variations in the number of obstacles affect learning and trajectory planning. The study uses a simulated environment with systematic changes in obstacle density to examine each algorithm's adaptability.

Methodologically, this research trains car racing models using both DQN and DDQN. The DQN approach utilizes techniques like experience replay and target networks, while DDQN separates action selection from evaluation to mitigate overestimation. The key components include model construction, environment setup, and detailed result analysis.

The primary goal of this research is to compare the trajectory outcomes of DQN and DDQN under different obstacle densities. By identifying the more effective algorithm, the study contributes to the development of safer and more efficient autonomous driving systems.

2 Algorithm

2.1 DQN Algorithm

The goal of the DQN algorithm is to approximate the action-value function $Q(s,a)$, which represents the expected cumulative reward when starting from state s, taking action a, and following an optimal policy thereafter [7]. The update rule of the action-value function follows the standard Q-learning update, given as:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{1}$$

Where: $Q(s,a)$ is the current estimate of the action-value function for state s and action a. α is the learning rate, which controls how much new information updates the current estimate. r is the immediate reward received after taking action a in state s. γ is the discount factor, which determines the importance of future rewards, with $0 \le \gamma \le 1$. s' is the next state observed after performing action a. $\max_{a'} Q(s',a')$ is the highest estimated action-value for the next state s', over all possible actions a' [8].

To stabilize training, DQN incorporates two essential techniques:

1. Experience Replay: The agent stores transitions (s,a,r,s') in a replay buffer, and during training, mini-batches of transitions are sampled randomly. This technique reduces correlations between consecutive experiences and increases data efficiency, leading to more stable learning [9].

2. Target Network: A separate target network is used to calculate the target values $r + \gamma \max_{a'} Q(s',a')$, which improves the stability of the Q-value updates. The weights of the target network are periodically synchronized with the main network to prevent rapid fluctuations in the Q-value estimates [9].

By utilizing these mechanisms, DQN is able to learn effective control strategies in complex environments, such as car racing simulations, where obstacle avoidance and trajectory optimization are crucial to success.

DDQN Algorithm

This paper also explains the applicability of DDQN (Double DQN) algorithm in obstacle avoidance tasks of reinforcement learning carts and compares its significant advantages over DQN (Deep Q-Network). First, DDQN reduces overestimation bias in DQN by separating action selection and target value calculation. In DQN, using the same network to select actions and estimate the target q value easily leads to overestimation of some actions, while DDQN effectively alleviates this problem by using online network to select actions and target network to calculate the q value. This modification enables the car to more accurately evaluate the benefits of each action during obstacle avoidance, thereby reducing the risk of selecting unsafe actions [10].

$$y = r + \gamma Q_{\theta^-}(s', argmax Q_{\theta}(s', a)) \quad (2)$$

In addition, DDQN provides higher stability during the training process. DQN is prone to fluctuations during the training process due to the over-estimation problem, making it difficult for the model to converge. By providing a more accurate estimation of Q-values, DDQN facilitates smoother adaptation to dynamic obstacle environments, thus improving the cart's strategic learning effectiveness [11].

In terms of decision quality, the accurate valuation of DDQN enables the cart to distinguish the true value of different actions more effectively. Compared to the suboptimal choices caused by DQN that may overestimate certain actions, DDQN enables the cart to make more reasonable action choices when facing obstacles, thus improving the efficiency and safety of task completion.

Finally, DDQN can help models avoid local optimal solutions, because DDQN can reduce the impact of overestimation, encourage the model to explore more possibilities, avoid limiting itself to suboptimal behaviour, and enable the car to perform more flexibly in more complex and dynamic obstacle environments. Therefore, by reducing overestimation, enhancing stability, improving decision quality, and facilitating strategy exploration, DDQN

algorithm shows strong applicability in trolley obstacle avoidance reinforcement learning task. And significantly improves the cart's obstacle avoidance ability and training effect compared to DQN [12].

3 Methods

3.1 Preprocessing

In the Gymnasium car racing environment, each episode generates a random track, ensuring that the agent learns to adapt to various track configurations rather than simply memorizing a single layout. An episode ends either when the car successfully passed all track tiles or when it leaves the playfield, incurring a penalty of -100 and resulting in the termination of the episode. The primary objective for the agent is to finish the track as quickly as possible. To encourage efficient completion, a penalty of -0.1 is applied for each frame, motivating the agent to minimize the number of frames required to finish the track. Additionally, the agent receives a reward of $\frac{1000}{N}$ for each track tile successfully passed, where N is the total number of track tiles. This reward structure incentivizes the agent to cover all track tiles while spending less time.

To increase the complexity of the car racing environment, this study introduces modifications by randomly generating obstacles on the track. The size, number, and colour of these obstacles can be adjusted through specific parameters, making the complexity of the environment configurable and allowing for the evaluation of the agent's adaptability under different conditions. In this modified environment, a collision between the car and an obstacle result in the immediate termination of the episode, with a penalty of -100 applied as negative feedback, emphasizing the importance of avoiding obstacles. Additionally, the environment generates a message stating "collision detected with obstacle," providing immediate feedback to both the agent and observers, clarifying the reason for the episode's termination.

In this study, to ensure that the agent has sufficient exploration during the early stages of training and can make decisions that avoid obstacles while maximizing future rewards, the discount factor (gamma) and exploration rate decay (epsilon decay) were set to 0.95 and 0.9999925, respectively. Additionally, to prevent the agent from becoming entirely exploitative and prematurely converging to suboptimal solutions, the minimum exploration rate was set to 0.05.

3.2 Visual recognition

OpenAI car racing environment produces a top-down 96x96 RGB image containing much information like car and obstacle position, racetrack configuration to represent different states. We made some modifications on the produced images for later training process. Firstly, to better reduce computational demands, they will be transformed from RGB to grayscale colour and the resolution is compressed into 84x84 pixels. To timely track the car's movement and surroundings, every continuous frame is stacked with other three former frames. Therefore, the state in this environment can be represented by every continuous four frames. they serve as input in the later Q-network.

3.3 DQN

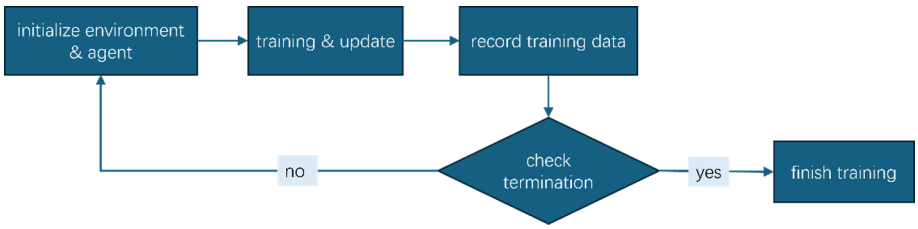

Fig. 1. The overall update flowchart

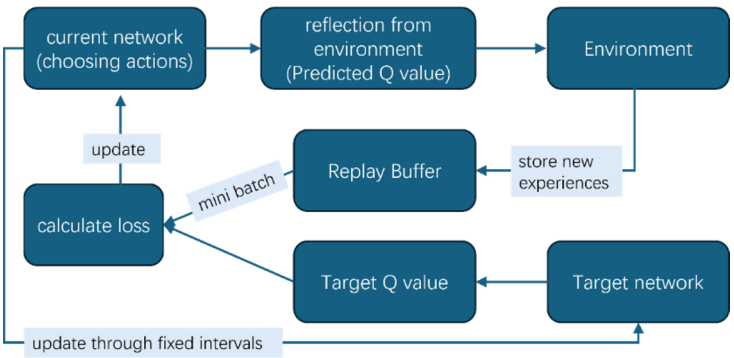

Fig. 2. Procedures of updating networks in DQN

Figure 1 has demonstrated the flowchart of the overall training procedures in the algorithm. DQN algorithm starts from the initialization phase. DQN agent initializes two main neural networks: an updating network (for training) and a frozen network (for calculating target Q value). Moreover, a series of training parameters are set, like epsilon, gamma, decay rate, replay buffer size and so forth. It is shown in figure 2.

When a new episode begins, environment would be reset into original stage. Agent would choose actions according to its current state derived from the visual recognition. This choosing pattern follows ϵ-greedy policy for randomly selecting actions as exploration. This action could be applied into current environment. Also, environment could respond an experience including new state, corresponding reward or penalty and whether this episode terminates. Agent would store this experience to replay buffer for latter training.

After every time step, agent would check if it met specific condition for model updating. Once it meets, agent would randomly select experience in a given batch size (it is set to 32 in our experiment) from the replay buffer to update its Q-network. The updating process is achieved by calculating the loss between target Q value and current Q value. Target Q value is calculated by frozen network and accurate Q value is derived from updating Q-network. What's more, the parameters of updating network would be updated every 1000 timesteps to frozen Q-network to assure the stability.

Upon finishing the required training episode, six types of data in each episode including date, time, reward, length, loss and epsilon are recorded into log and ultimate model parameters are saved into files for further evaluation.

3.4 DDQN

DDQN utilizes the same network architecture and experience generation process as DQN; however, the primary distinction lies in the method of calculating the target Q-value (TD target). In DDQN, the agent initially employs the Q-network to compute the Q-values for all

potential actions in the current state and subsequently selects the action with the highest Q-value for the next state. The target Q-network is then used to evaluate the Q-value of this selected action in the subsequent state, serving as the target value. This decoupling of action selection and evaluation differs from the DQN algorithm, which uses the same network for both selecting and evaluating actions when determining the target value. This method mitigates the risk of Q-value overestimation, thereby enhancing the stability and performance of model training.

4 Results

4.1 DQN vs DDQN

When the obstacle size is 5/scale (Fig. 3.), the performance of both DQN and DDQN is similar during the early stages of training (approximately the first 600 episodes). However, after that, the reward value of DDQN grows more rapidly, surpassing and consistently remaining above that of DQN. Throughout the training process, the performance of both DQN and DDQN steadily improves, with DDQN overall demonstrating superior performance compared to DQN.

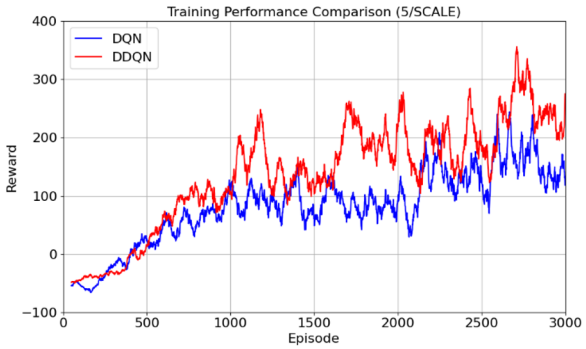

Fig. 3. Training Performance Comparison (5/SCALE)

When the obstacle size is 10/scale (Fig. 4.), the reward value of DQN and DDQN are nearly identical. Although DDQN exhibits better performance during the first 1000 episodes, the reward value of DQN begins to converge with that of DDQN, particularly after 1500 episodes, where the performance of both algorithms becomes increasingly similar. The overall trend for both is characterized by a gradual increase in performance.

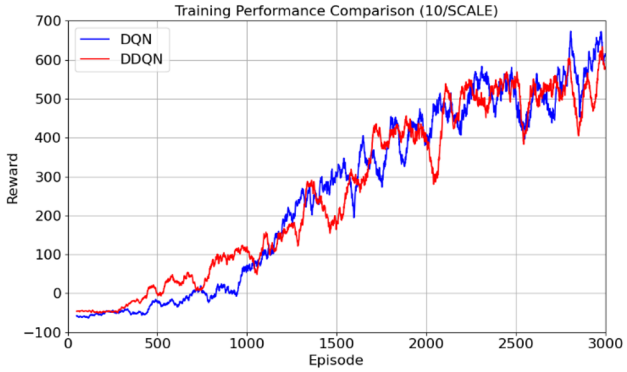

Fig. 4. Training Performance Comparison (10/SCALE)

When the obstacle size is 15/scale (Fig. 5.), DDQN still performs better than DQN. Despite both algorithms showing similar performance before 1000 episodes, the reward value of DDQN becomes higher after 1000 episodes, and DDQN consistently achieves higher rewards than DQN.

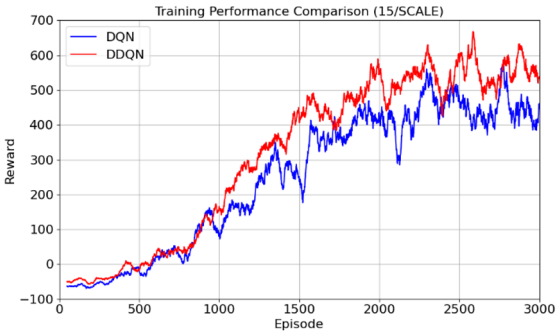

Fig. 5. Training Performance Comparison (15/SCALE)

With the obstacle size is 5/scale, the obstacles are relatively small, making the environment simpler and allowing the agent to avoid obstacles more easily. Both DQN and DDQN can quickly learn effective strategies, resulting in similar performance during the early stages of training. Due to the advantage of the double Q-network in DDQN, it is more effective at optimizing the policy in the later stages, leading to better overall performance.

When the obstacle size is 10/scale, the increased size of the obstacles introduces a higher level of complexity to the environment, making it more challenging for the agent to avoid obstacles. In this case, however, DQN is still able to effectively learn obstacle-avoidance strategies, resulting in performance that is comparable to DDQN. During the first 1000 episodes, DDQN's rewards are higher than DQN's, indicating that DDQN estimates Q-values more accurately in the early stages, thereby avoiding unnecessary exploration. As training progresses, DQN gradually reduces its overestimation error in the policy, ultimately achieving performance like that of DDQN.

As the obstacle size becomes 15/scale, the environment becomes more complex, requiring more efficient strategies to avoid obstacles and placing higher demands on the accuracy of the Q-network. In this more challenging environment, although the rewards of both algorithms continue to improve, DDQN achieves higher rewards in the later stages by mitigating the overestimation of Q-values.

In addition, the fluctuation is more evident when agent is training by DDQN algorithm. Several factors would contribute to this result. Firstly, this might be due to the introduction of its dual-network approach, especially when the networks tend to find a balance in the early training stage. Moreover, due to the presence of two networks, DDQN might be more sensitive to the change of environment. The sensitivity can be seen from the quick response of environment change, leading to significant fluctuation.

4.2 SIZE

In the comparison of the performance of DQN model and DDQN model in different obstacle size, we found that in all scenarios, the reward is growing with fluctuation. Moreover, the gap between some scenarios tends to be more significant.

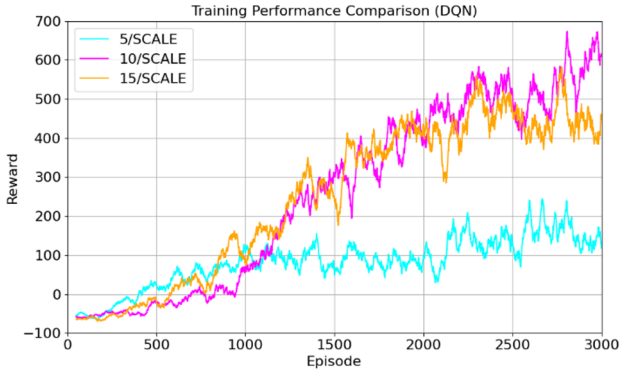

Fig. 6. Training Performance Comparison (DQN)

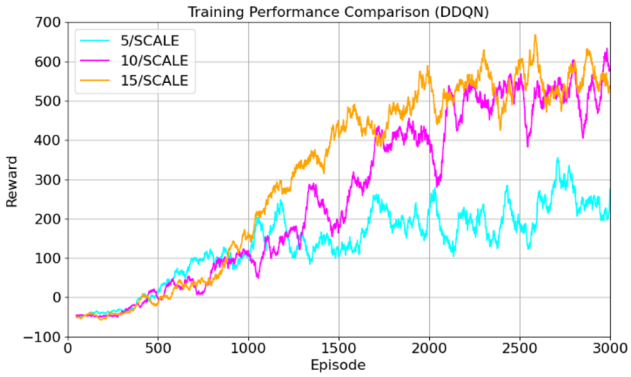

Fig. 7. Training Performance Comparison (DDQN)

It is shown in figure 6 and figure 7. When the obstacle size is 5/scale, the performance of both algorithm is worst. As in first 1,000 episode, the difference is not that obvious. This might be the agent performance is not so related to recognizing obstacles. In the rest of 2,000 episodes, there is only slightly increase and mostly maintained in range from 100 to 200. This mean that after training, the policy it learnt cannot support it for a whole race. From the relative stable performance, we assume that the lack of effective recognition would attribute to the bad result.

When the obstacle size is 10/scale, the training result is better than the obstacle is smaller. After training, car can finish a whole race, but its velocity is not that high.

When the obstacle size is 15/scale, the training result is almost like that in 10/scale. And the performance can be a little different. In DQN algorithm, 10/scale can be better training environment, while in DDQN, 15/scale is more advantageous. This can be seen that DDQN is more reliable in some extreme environments. DDQN through its separate network effectively reduce the problem of overestimation when it is in a complex environment. Therefore, DDQN could be an optimal algorithm in a more complex scenario.

5 Conclusion

In this experiment, DQN and DDQN algorithms are used to train the obstacle avoidance task of the car in reinforcement learning, and the results show that the two algorithms show significant differences in the environment of different obstacle sizes.

When the obstacle size is 5/scale, both DQN and DDQN perform similarly in the early stages of training, but DDQN achieves higher rewards later due to better optimization. With an obstacle size of 10/scale, DDQN has a slight advantage initially, but both algorithms converge to similar performance after 1500 rounds. When the obstacle size is 15/scale, DDQN significantly outperforms DQN, demonstrating faster and more stable strategy optimization in complex environments. Overall, DDQN shows superior performance in later stages and in complex environments, while both algorithms perform comparably in simpler settings.

Although this study shows that DDQN algorithm is superior to DQN in complex environments, there are still some unsolved problems and directions for further research in the future.

The current experiment focuses on obstacle size variations but can be extended to dynamic environments with more complexity, such as increasing the number of obstacles to test DDQN's adaptability. Despite DDQN's improved performance over DQN, it shows fluctuations during training, likely due to the dual network's sensitivity. Future research could enhance DDQN's stability, explore multi-agent cooperative learning, and optimize strategies under a multi-agent framework. Although DDQN outperforms DQN, future studies on algorithm refinement, environment complexity, and collaboration could further validate its performance in complex reinforcement learning tasks.

Authors Contribution

All the authors contributed equally and their names were listed in alphabetical order.

References

1. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, Human-level control through deep reinforcement learning. Nature **518**, 529 (2015)
2. M. P. Ronecker, Y. Zhu. Deep Q-Network Based Decision Making for Autonomous Driving, in Proceedings of the 2019 International Conference on Robotics and Automation Sciences (ICRAS), Wuhan, China, June 1-3, June 1-3 (2019), 154-160
3. H. van Hasselt, A. Guez, D. Silver. Deep reinforcement learning with double Q-learning. Proc. AAAI Conf. Artif. Intell. **30**, 2094 (2016)
4. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, arXiv **1509**, 02971 (2015)
5. Z. Li, X. Hou. Mixing Update Q-value for Deep Reinforcement Learning, in Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, July 14-19, July 14-19 (2019), 1-6
6. C. Zhang, C. Peng, M. Lin, Z. Du, and C. Wu, Double DQN Reinforcement Learning-Based Computational Offloading and Resource Allocation for MEC, in Proceedings of the International Conference on Mobile Networks and Management, Madrid, Spain, June 14-16, June 14-16 (2023), B205
7. V. T. Ha, T. N. Tu, N. T. Dung, T. L. Mien, C. T. Thuy, Deep Q-Network (DQN) Approach for Automatic Vehicles Applied in the Intelligent Transportation System (ITS), in Proceedings of the 2023 International Conference on System Science and Engineering (ICSSE), (2023), 527

8. S. Li, Y. Ji, J. Liu, Z. Bai, J. Hu, Global Path Planning of Unmanned Surface Vehicles Based on Deep Q Network, in Proceedings of the 2024 43rd Chinese Control Conference (CCC), (2024), 27
9. X. Zhang, L. W. Rogowski, M. J. Kim, 3D Micromanipulation of Particle Swarm Using a Hexapole Magnetic Tweezer, in Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), (2019)1581
10. A. Radford et al., Learning Transferable Visual Models From Natural Language Supervision, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, (2021)8748
11. Z. Wang et al., Dueling Network Architectures for Deep Reinforcement Learning, in Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, **48**, 1995 (2016)
12. M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. Proc. AAAI Conf. Artif. Intell. **32**, 3215 (2018)