Air-Combat Strategy Using Deep Q-Learning

Xiaoteng Ma
CFINS, Dept. Automation
Tsinghua University
Beijing, China
ma-xt17@mails.tsinghua.edu.cn

Li Xia
CFINS, Dept. Automation
Tsinghua University
Beijing, China
xial@tsinghua.edu.cn

Qianchuan Zhao
CFINS, Dept. Automation
Tsinghua University
Beijing, China
zhaoqc@tsinghua.edu.cn

***Abstract*—Unmanned aircraft systems (UAS) are essential components in the future air-combat. Due to high dynamics and randomness of the aircrafts, traditional methods are difficult to solve the optimal control strategy. The characteristics of reinforcement learning (RL) match the difficulty of this problem. In this paper, we build an air-combat game environment and train the agent with deep Q-learning (DQN). Despite of increasing probability of loses slightly, our method performs much better than other algorithms in the simulations. Compared with the searching based methods, like Minimax and MCTS, the policy trained by DQN can take specific tactics with a long-term view of the game. Result shows that a large number of simulations and carefully designed features and reward are the essential points of DQN.**

***Index Terms*—air-combat, UAS, DQN, RL**

I. Introduction

Unmanned systems have shown the potential in a variety of dangerous tasks, whether in commercial sector or military. However, there are still a lot of challenges in decision problems of unmanned systems with complex dynamic. Unmanned aircraft systems are dangerous and costly systems which have complex dynamics. There are mainly two ways to control the unmanned aircrafts (UAs). One way is to polit the aircraft remotely, which does not fully use the strengths of UAS and has a bad performance according to time delay. The other way is to manipulate the UAs with artificial intelligence (AI). In 2016, University of Cincinnati proposed a method to train an AI system for air-combat with Genetic Fuzzy Tree methodology (GFT) [1], [2]. They built a system called ALPHA which beat Colonel (retired) Gene Lee in a simulation environment, showing the possibility of replacing the export polit with AI.

Despite the success of ALPHA in building a AI system for UAS, the method they used relies exclusively on the expert experience to build fuzzy rules. In an ideal control system for UAs, the export experience will be helpful but should not be in the dominant position. Reinforcement learning (RL) gives us another view for air-combat problems, which could train the agent by trial and error. Approximate dynamic programming (ADP) was used to find air-combat strategy by McGrew [3], [4], which is a kind of RL algorithms. They designed some basic features with experience of pilots, only used for the basic state value function. With the high development of deep learning, RL algorithms are already successfully used in driverless technology, Robotics, video games, etc. In 2013, DeepMind first proposed deep Q-learning (DQN) which outperformed six Atari 2600 games than human experts [5], [6]. Later in 2016, they built an AI system for the game of Go called AlphaGo, which beat former Go world champion Li Shishi and became a hit [7]. The key point of the success of AlphaGo is combining deep neural network with Monte Carlo tree search (MCTS). AlphaGo zero, an updated version of AlphaGo, even beat the current world champion Ke Jie with 3:0 [8]. It should be noted that AlphaGo zero trained itself by self-play without any human experience.

In recent years, great progress had been made to improve the performance of DQN. Double DQN points out the overestimation problem and solves it with two same structure Q-networks [9]. Prioritized experience replay uses a sum tree structure to sample transitions from replay buffer based on temporal difference error (TD-error), which accelerates training a lot [10]. For a big discrete action space, dueling network estimates value and advantage function separately in spite of Q-factor [11]. There are also more papers improving DQN based algorithms in other views, like [12]–[14]. Combining all the methods and tricks above together, DeepMind proposed an algorithm called Rainbow, which could be seen as the state of art. Besides, different from value function estimation based algorithms, there is another cluster of methods called policy gradient, which also has a large impact on RL, like [15]–[18].

In this paper, we apply DQN to solve an air-combat problem and find an optimal strategy without polit experience. We build a 2D air-combat game simulation environment with simplified dynamics, and compare our methods with two practical methods, Minimax and MCTS. The results show that our method, which has abilities both for short distance precise operating and long-term planning, has the best performance in experiments. The essential point of the success of DQN in air-combat problem is a large number of simulations. Carefully designed features and reward also help a lot in the training process, which are discussed detailedly in the paper.

The rest of this paper is organized as follows. In Sec.II, we introduce an air-combat game we are concerned about, and discuss definitions of state, action and dynamics detailedly. In Sec.III, we propose a deep Q-learning algorithm to find an optimal air-combat strategy. Features and reward are developed in this part. In the last section, we show the results of simulation compared to different algorithms, analyse the strengths and weaknesses of our method.

978-1-7281-1312-8/18/$31.00 ©2018 IEEE

II. Problem Formulation

In this section, we introduce our air-combat game environment for simulation. First, the system states, actions and the goal of game are described. Next, the dynamics of aircrafts are discussed, followed by the baseline strategy we use for imaginary enemy. This work mainly references the work of McGrew [4], but has many improvements in details.

A. States

The state of the plane s is defined by the position, velocity, heading and bank angle.

$$s = [x, y, v, \psi, \phi] \tag{1}$$

where x, y are position variables of the aircraft, which have no limits in the $x - y$ plane. v is velocity variable which is limited between the speed range of the aircraft model. The heading variable is noted as ψ and allowed to change between $[-\pi, \pi]$. The aircraft turning capacity depends largely on the back angle, which is noted by ϕ here. The limitation of v and psi will be shown in the next part, where system dynamics is described.

The game environment considers 1v1 situation, where subscripts r and b denote ourself and the opponent respectively, meaning red and blue. To extract the relative feature of the state, the terms aspect angle AA, antenna train angle ATA and the distance R are used to describe the relative geometry of the state. AA and ATA are the specific terms of air-combat, which in this paper define the angles between the line of sight (IOS) and the heading directions of two planes respectively (see Fig. 1).

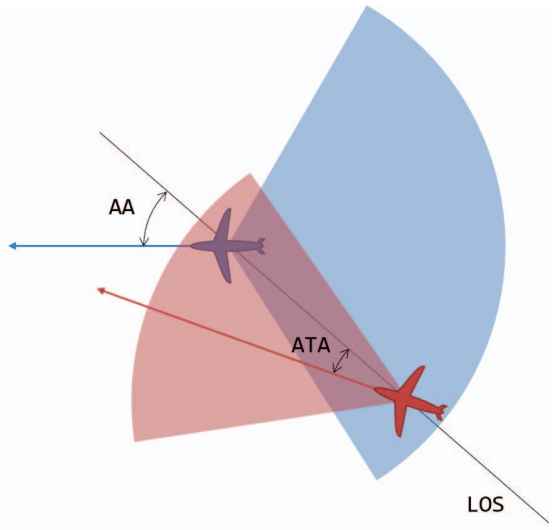


Fig. 1. The definition of AA and ATA. Assume that our plane is in red and the blue is the enemy. Two sector areas show the risk zone and the goal zone of the red.

B. Actions

The action is defined as follow,

$$a = [w, u] \tag{2}$$

where w is control action to the bank angle, which decides how much the plane can turn. The acceleration is noted by u, used to control the velocity of the plane.

Algorithms based on searching need discrete action space, which is also helpful in Q-learning. We define five valid actions for each plane to simplify the strategy, which represent turning left, turning right, speed up, speed down, maintaining bank angle and speed respectively. All the algorithms discussed later will be designed based on these actions. They are

$$\begin{aligned} a \in \{&[0, -40^\circ/s], [0, 40^\circ/s], \\ &[0.3m/s^2, 0], [-0.2m/s^2, 0], [0, 0]\} \end{aligned} \tag{3}$$

C. Goal

The goal of the player is to reach and maintain a advantage position behind the opponent, which is seen as a sufficient condition to launch a missile in the real combat. To quantify the goal, we define a reward function for every system state (see algorithm 1). If the player's plane flies into the goal zone of the enemy, it will get a positive reward at that time step. On the contrary, it will receive a negative reward when the opponent makes it to see its tail. When the two plane get too closer to each other, we will give a penalty to avoid collision. In summary, there could be four possible results in a time limited episode: win, lose, collision and tie.

Algorithm 1 Goal Reward Function $g(s)$

Input: state s

Calculate AA, ATA, R
if $R < 0.1m$ **then**
  $g = -10$
else if $0.1m < R < 2m$ **then**
  if $|AA| < 60^\circ$ and $|ATA| < 30^\circ$ **then**
    $g = 1$
  else if $|ATA| > 120^\circ$ and $|AA| > 150^\circ$ **then**
    $g = -1$
  end if
else
  $g = 0$
end if

Output: g

D. Dynamics

The simulation is developed based on the dynamics of a model micro-UA. After taking the actions, each plane updates its state according to the transition function respectively (see algorithm 2). Each action repeats N time for a time step δt, where $N = 5$ and $\delta t = 0.05s$ in this paper. To be clear, there will be a 0.25s time interval between two decisions. Detailedly, the velocity v is changed by the acceleration u. Note that considering speed up and speed down of planes have different mechanism, we limit u between $-0.2m/s^2$ and $0.3m/s^2$. The bank angle ϕ is changed by w, which means $\dot{\phi}$ in physics. $|w|$ is fixed to $40^\circ/s$, and the action decides which the direction to turn. The turn rate $\dot{\psi}$ is updated by the new bank angle,

based on the aerodynamics. This feature makes it impossible to change the direction of planes sharply, and forces the agent to make a long-term plan.

Algorithm 2 State Transition Function $f(s, a)$

Input: state of plane $s = [x, y, v, \psi, \phi]$, action $a = [w, u]$.
Input: acceleration of gravity g.
Input: the action repeat times N, the time step δt.
1: **for** $i = 1$ to N **do**
2: $\phi = chip(\phi + w\delta t, -\phi_{max}, \phi_{max})$
3: $v = chip(v + u\delta t, v_{min}, v_{max})$
4: $\dot{\psi} = \frac{g}{v} tan(\phi)$
5: $\psi = \psi + \dot{\psi}\delta t$
6: $x = x + vsin(\psi)\delta t$
7: $y = y + vcos(\psi)\delta t$
8: **end for**
Output: $s = [x, y, v, \psi, \phi]$

E. Enemy strategy

The strategy of the imaginary enemy is the Minimax algorithm, which is a traditional and robust method for games [19]. The main idea of Minimax, coming from game theory, is that taking a best decision under assumption of the most aggressive enemy. Decisions made by Minimax are reasonable, but require a remarkable amount of searching for the future cases. In this paper, we realize a recursive algorithm, traversal search all the situations in next N steps and return a predefined score (see algorithm 3). The score function considers the relative angles and distance, and returns a score in $[-1, 1]$.

Algorithm 3 Score Function $S(s)$

Input: state s , desired distance R_d
1: Calculate AA, ATA, R
2: $S_A = 1 - \frac{|AA|}{180^\circ} - \frac{|ATA|}{180^\circ}$
3: $S_R = e^{\frac{-|R-R_d|}{180^\circ k}}$
4: $S = S_A S_R$
Output: S

III. Methodology

In this section, the details of our implementation of deep Q-learning to the air-combat game are discussed. We introduce the tricks used in the training process, including the dueling network structure, priority experience replay, etc. Features for the network input are developed, and rewards are well designed to solve this problem.

A. Deep Q-Learning

The air-combat problem can be formulated as a Markov decision problem (MDP) in math. Q-learning is a model free algorithm to solve the MDP problem, which means decision-making process is independent from the system dynamics. According to the curse of dimensionality, deep neural network is used in the Q-learning, which method is called DQN. The key point of DQN is using a big transition buffer and two same structure networks, evaluation and target network, to break the strong correlations between consecutive samples.

The initial version of DQN has some problems and several tricks are proposed to improve its performance. Q-learning is a boosting algorithm, which takes the maximal value over actions of its output every step. Due to the maximizing, using single network to estimate the scores will cause the overestimation problem. Double DQN uses two networks, which are just evaluation and target network in the initial version, to solve the problem [9]. When there are many actions in the Q-network, it is hard to learn all the actions scores accurately according to the big variance of state value estimation over actions. Dueling network addresses the problem and solve skillfully it with a two-head network architecture [11]. The network has two estimators for state value and action advantage separately, without imposing any change to the underlying algorithms. Though the original DQN uses a big transition memory to accelerate the training process, it is still less data-efficient. Prioritized experience replay [10] proposes sampling the transitions according to TD-error and giving more computing resource to important transitions, which improves data-efficiency greatly. They also develop a sum-tree structure which makes the batch sampling process efficient.

In this paper, we combine all the tricks above together, making it more suitable for air-combat problem (see algorithm 4). We use the same method as Double DQN to train the network and a dueling network architecture for multiple actions. We also use a sum-tree structure for memory buffer and prioritized experience replay for sampling transitions. Note that we use a soft target replacing method, which changes the parameters of target network little by little otherwise directly sets the parameters same as the evaluation one. It is a trick first used in DDPG [12], proved to be helpful in the experiments.

B. Features and Reward

Different from inputting the system state directly into the network, we design the features which reflect the relative information of the cases sharing the score. We choose the relative geometry of the state $[R, ATA, AA]$ as the basic features. However, these terms can not tell the information about velocity and bank angle. And the rates of ATA and AA are also important for the judgement of pilots, which infer the degree of enemy's aggression. Considering angles will not be continuous at $-\pi$ and π, we use $[cos(ATA), sin(ATA), cos(AA), sin(AA)]$ as parts of the feature rather than inputting them directly. The full definition of the network input x is shown as follow,

$$x = [R, cos(ATA), sin(ATA), cos(AA), sin(AA), \dot{ATA}, \dot{AA}, v_r, v_b, \psi_r, \psi_b] \tag{4}$$

Reward is one of the key points of reinforcement learning, which largely determines the performance of algorithm. In the air-combat case, it is intuitive to use the goal reward function as the reward. However, the goal reward function does not

Algorithm 4 Deep Q-learning Algorithm

Input: maximal episode number K, maximal step of each episode T, episode replay buffer capacity N, exponents α and β

Initialize the Q network $Q(s, a|\theta)$ with weights θ randomly
Initialize the target network $Q'(s, a|\theta')$ with weights θ' randomly
Initialize replay buffer H using random policy with maximal priority
for $episode = 1 : K$ **do**
  Reset the system and receive initial state s_1
  for $t = 1 : T$ **do**
    Choose action a_t according to $\epsilon - greedy$ policy
    Execute action a_t, observe r_t, s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in H
    Sample a random mini batch of N transitions (s_i, a_i, r_i, s_{i+1}) from H where $i \sim P(i) = p_i^\alpha / \sum_i p_i^\alpha$
    Calculate the importance sampling weight $w_i = (N \cdot P(i)^{-\beta})/max_i w_i$
    Set $a_{i+1} = argmax_a Q(s_{i+1}, a|\theta)$
    if Terminal **then**
      Set TD-error $\delta_i = r_i - Q(s_i, a_i|\theta)$
    else
      Set TD-error $\delta_i = r_i + \gamma Q'(s_{i+1}, a_{i+1}|\theta') - Q(s_i, a_i|\theta)$
    end if
    Update θ with loss $L = \frac{1}{N}\sum_i w_i {\delta_i}^2$
    Update transition priority $p_i = |\delta_i|$
    Update $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
  end for
end for

work well in practice. In a stalemate, there is no feedback from the goal reward function. If two planes hover in circle, no one could get to the goal zone, the algorithm will lead the network stuck in the suboptimal point. Thus, in this paper we use a reward combining the goal reward function and the score function for Minimax together. The former provides the main information about current performance and the latter gives small guidance about where to go in hovering.

$$r = (1-c)g + cS \tag{5}$$

C. Network structure

The network architecture mainly reference dueling network, which has two branches to estimate state value and actions advantage separately (see Fig. 2). We use a full connection hidden layer to transform the input features, following another two branches of full connection layers to process the high level features for state value and actions advantage. In the last layer we add outputs of state value branch and the normalized actions advantage together as the Q-factor. The first hidden layer has 100 nodes, and the second layers have 30 nodes respectively. Tanh activations are applied for all the layers.

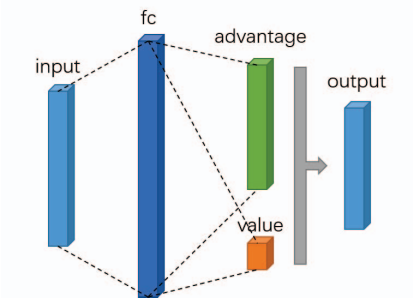


Fig. 2. Q network structure

IV. Experiments

A. Training

We train the DQN against 3 steps Minimax strategy. The implementation details are as follow. We simulate 20000 episodes. Each episode is limited to 50s, which means there is at most 200 steps in a single episode. Initial state for each episode is sample randomly in the state space. The learning rate is set to 0.001 and the discounted factor γ is set to 0.99. The batch size is set to 128. The capacity of the transition memory buffer is set to $2^{15} = 32768$. The exponents α and β in priority experience replay are set to 0.6 and 0.4 respectively. The epsilon decreases linearly from 1.0 to 0.1 in the beginning 10000 steps. After 10000 steps, the epsilon is fixed to 0.1 for further exploration. We choose Adam as optimizer [20]. Momentum and gamma of the optimizer are set to 0.9 and 0.99 respectively. We implement our DQN based on the public deep learning platform PyTorch. The computing server has 8 Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz and a NVIDIA Tesla P100-PCIE GPU. The moving average output and the performance of DQN are shown in Fig. 3.

B. Testing

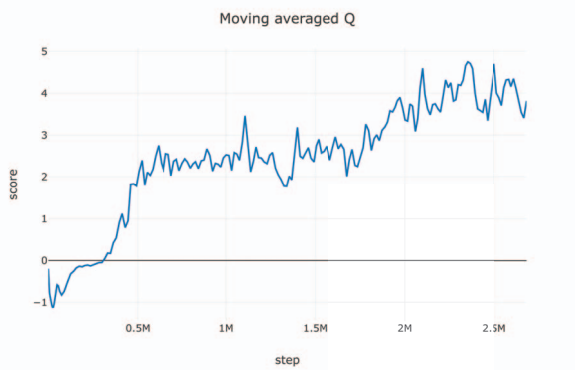
To test the performance of our method, we use a 3 steps Minimax strategy as the opponent policy. The planes are totally same in the experiments, which means the starts of the simulations are essential. We compare our method with Minimax and MCTS, starting from 2048 different initial states. The start states are sampled in R, AA, ATA evenly, where $v = 2m/s$ and $\psi = 0°$.

TABLE I
Result Comparison

Algorithm	Win	Lose	Tie	Collision
Minimax	3	3	1004	1038
MCTS	192	8	813	1035
DQN	759	92	153	1044

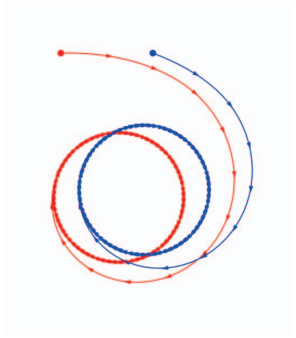
The result shows that DQN is much better than other searching based algorithms (see table I). The policy of DQN can win more than third of simulations in 2048 different starts. However, the cost of better performance to win is bigger risk to

(a) Performance

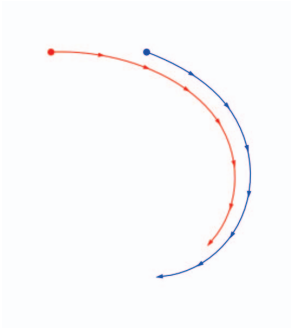


(b) Estimated Q

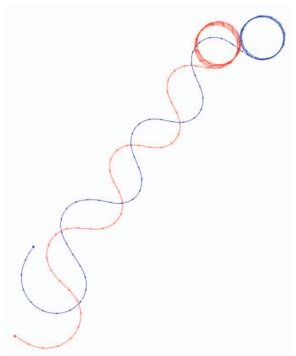
Fig. 3. Performance (a) and estimated Q during training (b).



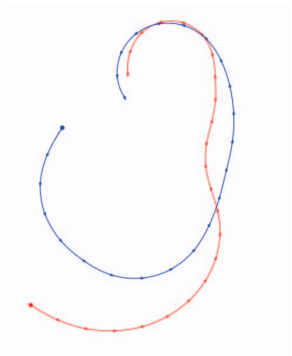
(a) Minimax with offensive start



(b) DQN with offensive start



(c) Minimax with defensive start



(d) DQN with defensive start

Fig. 4. Flight trajectories with specific starts. The red plane are controlled by the tested policy. The blue plane is the imaginary enemy operated by Minimax policy.

lose the game. On the other hand, DQN avoids more collisions, which are both bad in Minimax and MCTS.

We analyse the decisions of different policies and draw the fight trajectories (see Fig. 4). The policy trained by DQN has a stronger ability to plan for a long time. With a offensive start, both Minimax and DQN win the game. However, Minimax goes greedily, hovering for a long time before catching the goal zone of enemy. On the contrary, DQN bits the tail of the enemy, controls the rhythm well and maintains the advantage. In the case of defensive start, Minimax also swerve sharply to get rid of the opponent and twists together in the end. DQN turns in a proper angle at the very beginning, and obtains a neutral position naturally. It is very smart that the agent goes across the opponent's trajectory, following speeding down to get an advantage position, which can be seen as a tactic.

In summary, the policy trained by DQN has a long-term view with the specific tactics in some situations. Despite of increasing the risk of loses, DQN has a significant performance improvement for air-combat problems. However, there are no free lunch in the air-combat problems. The success of DQN is based on a large number of simulations for exploration. Features and reward are also important in the training process. Good features help the agent fitting the scores quickly, while carefully designed reward guides the exploration in stalemates.

V. Conclusion

This paper introduces a method to find a air-combat strategy with deep Q-learning. A 2-D air-combat game environment with aircraft model dynamics is built for training and testing the algorithm. Features and reward are developed carefully as the input and feedback of the neural network. In the 2048 experiments with random starts, our method wins more than 700 episodes against 3 steps Minimax strategy. Compared with the searching based algorithms, the policy trained by DQN has a long-term view of the game, and can take specific tactics in some cases. Despite of increasing the risk sightly, the performance of DQN is much better than former methods.

VI. Acknowledgement

This work was supported by the National Natural Science Foundation of China (No. 61425027 and No. 61573206).

References

[1] N. D. Ernest, "Genetic fuzzy trees for intelligent control of unmanned combat aerial vehicles," *Dissertations & Theses - Gradworks*, 2015.

[2] N. Ernest, D. Carroll, C. Schumacher, M. Clark, K. Cohen, and G. Lee, "Genetic fuzzy based artificial intelligence for unmanned combat aerial vehicle control in simulated air combat missions," *J Def Manag*, vol. 6, no. 144, pp. 2167–0374, 2016.

[3] J. S. Mcgrew, "Real-time maneuvering decisions for autonomous air combat," *Master's Thesis Mit*, p. 138, 2008.

[4] J. S. Mcgrew, J. P. How, B. Williams, and N. Roy, "Air-combat strategy using approximate dynamic programming," *Journal of Guidance Control & Dynamics*, vol. 33, no. 5, pp. 1641–1654, 2010.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Computer Science*, 2013.
[6] 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*, vol. 518, no. 7540, p. 529, 2015.
[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, d. D. G. Van, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of go with deep neural networks and tree search." *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, vol. 2. Phoenix, AZ, 2016, p. 5.
[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *Computer Science*, 2015.
[11] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," pp. 1995–2003, 2015.
[12] 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," *Computer Science*, vol. 8, no. 6, p. A187, 2015.
[13] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," 2017.
[14] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, and O. Pietquin, "Noisy networks for exploration," 2017.
[15] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *Computer Science*, pp. 1889–1897, 2015.
[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.
[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.
[18] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.
[19] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computer Science*, 2014.