Multi-Agent Adversarial Deep Reinforcement Learning

Robbie Sadre, Trevor Chan, Yixian Wang, Ibrahim Ahmed, Yash Bhartia

Abstract

This research project details concepts from multi-agent reinforcement learning, putting differing Deep Reinforcement Learning algorithms against one another to evaluate their respective strengths and weaknesses. Using the Atari game of PONG as a multi-agent setting, an agent modeled by the classic Deep Q Learning Network algorithm competes against the Dueling Deep Q Network algorithm to determine which algorithm learns more quickly and adaptively adjusts it routine to seize a victory.

1. Introduction

In the last decade, we have seen huge improvements in the area of machine learning. This is heavily due to the utilization of computationally powerful GPU technology being re-purposed so that deep learning algorithms can flourish. This has led to strong development and usage of convolution neural networks geared towards computer vision tasks that have incredible performances. Supervised Machine learning is the most dominant group of machine learning/computer vision algorithms. However, another brand of machine learning has gained popularity in the last 5 years as well– Reinforcement Learning.

To illustrate the idea of reinforcement learning, we can offer a metaphor. Suppose that Bob's mother points to a hot stove and tells him "if you touch this you will burn your hand, and it will hurt." This way Bob learns not to touch the hot stove. However, suppose Bob's mother is not around, so he does not know any better. So never having touched a hot stove and being the curious toddler he is, he touches it, and learns from experience that a hot stove will burn his hand and hurt him. The first example of how Bob learns can be seen as supervised learning, and the second example is reinforcement learning. Reinforcement learning is to learn the best actions to take in a specific state based on previous experiences of reward and punishment. The data one receives is unique every time as whatever you learn depends on the scenario that one comes across while learning. That being said, while training we go over multiple iterations that allow us to improve our learning conditions.

Traditional reinforcement learning algorithms are usually based on storing Q values for each specific state. A Q value, is the expected discounted future rewards of taking a specific action in a certain state. In other words, it is a quantifiable way to compare how much reward will result from taking one action over another in a specific state. For example, in some state STATE we have two possible actions A1 and A2. The Q values are represented as Q(STATE, A1) and Q(STATE, A2). If one of these values is greater than the other, it means that that particular action is much more likely to lead to higher rewards over time.

Traditionally Q values are stored in a table. For every state, there are a certain number of actions, and for each action there is a space for a q value in the table. for example if we have 10 states and 2 actions per state then we have 20 spaces in the q table. This method can work for simple Grid-world games with few potential states, but not even for simple old Atari games where even a 320x320 pixel image display game can have a huge number of states. This is where Deep Reinforcement Learning Comes in.

Deep reinforcement learning combines the concepts of a CNN and traditional Tabular Q learning. In the case of Deep Q Learning, the way it works is it will use a Neural network to act as a Q value function approximation. Essentially, an image or multiple images that are the pixel image frames of the game environment are input into the CNN. These represent the state. The output of the CNN will be the Q values approximated by the CNN to model this learning agent Q table. The policy then decides which of the actions to take based on the Q values (usually it's just the greatest q value is chosen as the best action) and then that action is enacted. Additionally, what will happen, is that information is collected. Information like states, actions taken, and reward signals that result from these sequences. Then this information is used every few frames to train the network into developing more accurate Q values thus slowly converging the agent to a better policy.[5]

Another area of research in reinforcement learning is in multi-agent reinforcement learning[10][9][8]. This is basically the case for when two RL agents must either compete or collaborate on separate or joined tasks. This has been done for tabular Q learning in grid-world environments, for example[8], which assessed the performance differences that were observed when an agent using a Q learning algorithm was pit against a an agent using a Min Max Q learn-

ing algorithm. There have been more recent experiments in Multi Agent Deep RL such as [8][1][]. The main focus of these papers is to assess the collaborative/competitive nature of DQN learning algorithms when the reward signal/game rules are adjusted to reflect different ultimate goals. In these papers the same algorithm is used for each agent in the environment, however.

In this work, what we aim to do is to experiment whether a specific Deep RL algorithm may triumph over another in an adversarial environment. We will be testing the classic Deep Q Learning Network Algorithm vs the Dueling Deep Q Network algorithm in the classic Atari game PONG. In this research we hope to identify any weaknesses or strengths that the algorithms may have in a multi agent environment where two separate algorithms are used to model each player. We will accomplish this with two types of tests. We will first pretrain two agents modeled by the different algorithms in isolation as done in the original papers against a built in simply modeled opponent. We will take these two models and then have them compete in the multi-agent Pong environment. For our second test we will have two agents with the differing algorithms learn to play each other from scratch to identify if either algorithm tends to learn faster than the other in the multi-agent environment.

1.1. Background

In reinforcement learning, there are three key components: state, action, and reward. The state describes the current situation. Action is what an agent can do in each state. Reward is what the agent receives after it takes an action. For the example of a robot try to learning how to walk, the state is the position of robot's two legs. The action will be taking steps within a certain distance. And the reward will be keep balanced (positive) or failing (negative). The robot repeatedly tries different step many times and finally learns to walk. The policy function one uses in reinforcement learning is decided from a pool of large possibilities. Different learning methods have different grading policies and these policies are not like alphas. These cost functions use the win or loss conditions to decided whether to increase the score or not. An illustration of the process can be found in Figure 1.

1.1.1 Q Learning

Q-learning is a commonly used reinforcement learning method. It is a value iteration method to get the optimal result, and the formula is below:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma maxQ(s_1,a_1)] \quad (1)$$

In the formula, α is learning rate, which is a number between 0 and 1. It is a weight given to the new information versus the old information. Q(s,a) is the long term reward in memory, and it is updated by the summation of the current reward r, with all future rewards in the next state, $Q(s_1, a_1)$. γ is the discount factor between 0 and 1, meaning future rewards are not as valuable as the reward now.

1.1.2 DQN

The drawback of Q learning is that it requires a Q value table to store every Q value for every state, which is impossible when the states are very large based on the current storage capability. Deep reinforcement learning solves this problem. We know that deep neural networks are good at representing high dimensional data, so we employ a deep reinforcement learning algorithm called DQN, which is simply a combination of deep neural network and reinforcement learning. Deep Q Network(DQN) [5] was first proposed by DeepMind, a company owned by Google in 2013. It is a traditional Q learning algorithm but replaces the Q-value table with a convolutional neural network. To calculate the gradient score and decide on the policy DQN looks at the score attained only on winnings or losing. That too the policy extracts the score only if the victor made a move in the previous move. Furthermore, the score translates to all previous stage as a percentage of the original until it reaches a beneficiary score of 0.

Considering tasks where an agent interacts with an environment through a series of observations, actions, and rewards, the goal of the algorithm is to maximize cumulative rewards through selecting actions. In other words, DQN uses a deep CNN to approximate the action-value function

$$Q*(s,a) = max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi]$$

where $Q*$ is the maximal sum of rewards r_t discounted by γ at each time step t, achievable by some reward policy $\pi = P(a|s)$, after making an observation s and taking an action a. When learning Q-learning updates are applied on samples $(s,a,r,s') \approx U(D)$, drawn uniformly at random from pool of stored samples. Each update at iteration i minimizes the following loss function

$$L_i(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(D)[(r + \gamma max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i))^2]}$$

where γ is again the discount factor, θ_i are the parameters of the network at iteration i, and θ_i^- are the parameters used to compute the policy at iteration i. θ_i^- are only updated with θ_i at every constant C steps and are held fixed in between each of the individual updates [6].

1.1.3 Dueling DQN

There are many variants of DQN, but one of the most influential is Dueling DQN [10]. It is a modification of the

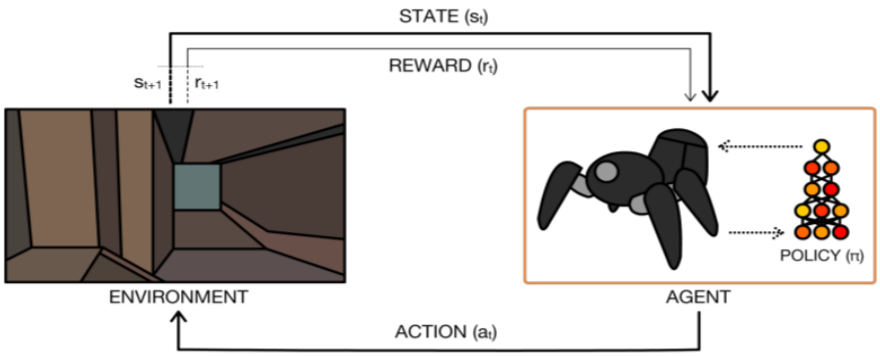


Figure 1. The perception-action-learning loop. At time t, the agent receives a state s_t from the environment. Using a reward policy, the agent then selects the best action a_t to perform. Once it executes this action, it receives a reward r_{t+1} and a following state s_{t+1}. It uses these to learn and improve its policy.

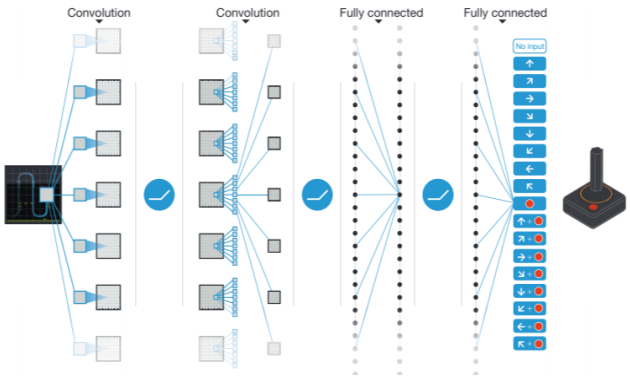


Figure 2. DQN architecture

architecture of DQN, which compute the state value and advantage value separately and combines them together to get the final Q value, instead of computing the Q value by using the state value and advantage value together. The method is simple but provides a decent result.

2. Related Works

There have been many algorithms that have been developed over the years that utilize Deep Reinforcement Learning. The most well known of which is the DQN[5], an algorithm developed by DeepMinds, a company owned by Google. Several algorithms since then have been developed such as A3C [4] TRPO [7] and Double-q-learning [3]. Most of these works highlight the improve performance of an agent interacting with a simulated environment. This is one aspect of the work. As far as adversarial reinforcement learning in multi agent environments, there has been research that studies the long term effects and results of training separate algorithms to compete for opposing goals in a zero sum game like Q-learning agents playing grid-world soccer[9].

Other works utilizing multi agent settings have traditionally focused on team play and collaborative works between agents, powering each agents by the same training algorithm. Some have trained agents in different ways, but ultimately used the training to help agents collaborate in team settings. They show how competitive and collaborative behavioral trends emerge as an agent's training style reflects their ability to work with or against other such agents [8]. Tampuu et. al tune reward parameters to illustrate the 'trained' emergence of such trends. They aim for optimality in keeping the game ball active for as long as possible, gearing the adaptive behaviors of agents to accomplish this task. The goal of these researchers was to study the "decentralized learning of multiagent systems living in highly complex environments [8]." To accomplish such a trained goal, the researchers advise their created agents in 'fully competitive' or 'fully cooperative' schemes, where an agent must score more than the opponent in the first scenario. In the second setting, loosing the ball penalizes both players, making the setting fully cooperative. To collect the in-game statistics, the research team collected quantitative measurements such as average paddle-bounces per point, average wall-bounces per paddle-bounce, and average serving-time per point from all agents. Such statistics allowed for recordings of emerging behavioral trends and patterns. In their discussion, Tampuu et. al state that a more complex game than Pong, such as Warlord, could be used and studied to apply the same principles

Similar works like that of Diallo et. al focus on tuning reward parameters to encourage multi-agent collaboration, once again focused more on team work that inter-agent competition [2]. What Diallo et. al accomplish is a a collective understanding between agents, where each agents jointly learns to divide their area of responsibility towards a shared goal. By tuning the reward mechanism to train

agents for collaboration, the researchers provide proof that a balanced division of work for individual team members allows for effective cooperative behaviors. Their work and resulting conclusion provides ample evidence that multi-agent concurrent Deep Q Networks can "smoothly converge even if the environment is non-stationary" [2]. They also add that despite the determined options for agent-to-agent team work, some drawbacks of their methods lie in states where both agents face a ball in each of their responsible area. To fix this broken mechanism, the authors state that control scenarios need to be put in place to facilitate better communication [2].

3. Technical Approach

Our experiments require the use of a gaming environment that has the capabilities of having two separate players in the Atari PONG game. Unfortunately, the commonly used Library for RL gaming environments does not have a multiplayer mode for which we can have two separate agents control the two paddles. Therefore we developed our own multiplayer Pong game environment in Pygame. Using a Pong Game Class we developed a function that takes in 2 separate commands form the players (each of the two can be up down or don't move), and then this action is repeated for 4 frames in the gaming environment (as was done in [3]) and those four frames are returned and used as inputs for the neural networks to determine the next set of actions for the agents to take. The next step was to train separately a DQN and Dueling DQN in their own individual single against environments against simple opponents modeled by if statements in order to ensure that our models would learn over time at least individually (to confirm functionality and correct implementation). We then took these pretrained models and had them play each other in a multi agent environment in order to assess which learned a better policy/q value approximator. Our final test was to train the agents in a multi-agent environment from scratch with no pretraining what so ever. This was to assess which algorithm might learn faster against an adversarial opponent in a multi agent environment.

4. Experiment

One major change we found necessary to this traditional algorithm was to change the way the input was formated. Traditionally, the input to the CNN are the most recent 4 frames. We found this to have very slow convergence for the Q Values that, not only in terms of the number of training steps it would take, but also the amount time the individual training steps ran for. Using 4 frames as input, we saw that the a single agent network required more than 500k training steps to reach any substantial performance improvement. A much faster way to process the data was to have the input to the network be a weighted difference between the current frame and the previous frame. This offered adequate state information for training of the networks.

Once the gaming environment was completed, we then took incremental steps towards our final multi agent environment. The next step was to train a regular DQN versus a built in player to get an idea of how many training steps we can expect are required to gain substantial experience to perform well in a game. We graphed the rewards over time to analyze this single agent DQN. We basically accrued rewards for a certain amount of time and then gained an idea for the rewards accumulated/time. Convergence to a close to optimal policy was essentially identified as whenever the rewards/ time plateued and continued at a constant rate.

Once our models pretrained separately against simple opponents modeled by if/else statements, we had them compete in the multi agent environment using their learned Q values.

For our final test we did a multi agent training session from scratch, where a DQN and Dueling DQN trained against each other without any pretraining or knowledge going in. All training sessions were given 500,000 training steps.

One issue that we needed to face was how to identify when the agents reached close to optimal policies. In the single agent version this is as simple as getting to a point where rewards over time becomes consistent because it would start out negative and level off at some positive value. However, for the multi agent version, because Pong is a 0 sum game, and both agents should do equally poor in the beginning having no experience or knowledge whatsoever, their rewards would each be 0 to start off with, because they would constantly be scoring on each other. In fact, given that one possibility for the outcome is that each player learns at the same rate and reaches an equal level of skill at the game, it is possible that the rewards over time remain equal for the entire game. We were correct in this assumption. So in order to deal with this we needed to create a new metric to analyze when the players reached their optimal policies.

We looked at real ping pong players. When two people are very bad at ping pong, the rounds may not last long, because they will not be able to keep the ball in volley for very long. However, for two very good players, a single round will last a very long time because they can keep the ball in volley for much longer. Suppose two unskilled players played non stop for 10 minutes, and two skilled players played non stop for 10 minutes. The former pair would likely have many short lived rounds of the game, and the later would likely have few long lasting rounds of the game. So in order to measure the learning of the multi agent players, we simply accumulated how many rounds were played in total in the last x number of time steps, and waited until

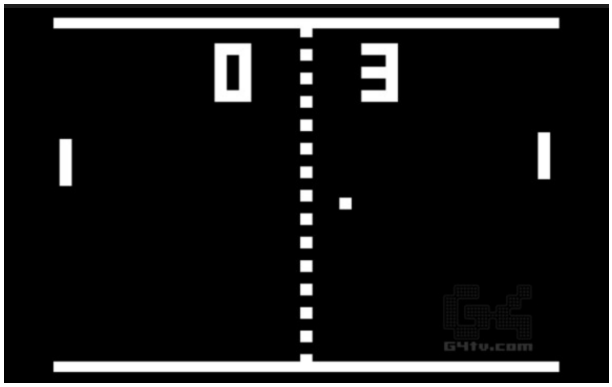


Figure 3. The game Pong

this number stagnated. This method of measuring proved to be very useful and actually was successful at determining when the multi agent players reached near optimal levels of skill.

A video demo of the multi agent training (without pretraining the separate agents) can be found at: https://www.youtube.com/watch?v=g9svjZxpug0

4.1. Setup

All code was written in Python 3.6 using the Tensorflow framework. The Pygame library was also used to help simulate the Pong game environment. Experiments were conducted on a NVIDIA GEFORCE GTX 1070.

4.2. Results

Figure 4 showing a single line is the rewards over time of a single agent DQN training session. Each point value on the graph is the rewards gained in the last 5000 training steps. Because actions were chosen every 4 frames, and the network was trained every 4 actions, this is essentially the number of rewards gained in the last 80000 frames. We will refer to this as 5000 training steps for future references. As expected, the rewards over time starts out negative, steadily increases, and then levels off. The leveling off occurs when the network reaches accurate q values for the optimal policy.

Figure 4 shows the rewards of the DQN and Dueling DQN in the multi agent environment after initially being pretrained in their individual single agent environments. As can be seen, there is some oscillation for each in the rewards over time. This makes sense intuitively. Whatever algorithm is losing at a point in time will after a while learn a policy to overcome its adversary, and then its adversary will then adapt to the new policy of the other player. Since learning a new policy takes time, this results in oscillation.

Figures 5 shows the rewards over time and number of games every 5000 training steps over time (two separate trials). As predicted, the number of games every 5000 training steps would decrease as the performance of the players improved and eventually level off once the individual players reached near optimal policies respectively. It seems that in the case for both the pretrained and trained from scratch training sessions, the steady state rewards over time (the rewards over time after being trained for a long time) tend to be higher on average for the DQN (shown in orange on figures . This is interesting, since Dueling DQNs have shown to have higher performance in Pong in other works [8]. This data suggests that in an adversarial Environment DQNs actually Triumph over the Dueling DQN Architecture.

Something to note about the first trial for the multi-agent training (with no pretraining, figure 5 top) is that up until around 150,000 training steps, the Dueling DQN had a significant advantage over the DQN. This led to the possibility that the Dueling DQN may in fact be superior during earlier steps of training. However, after repeating the experiment in a separate trial (Figure 5 Bottom) we saw that something else happened this time. The DQN actually triumphed over the Dueling DQN in the beginning as well as the end of the training. This leads to the possibility that any significant performance difference in the beginning of the training of the multi-agents is likely due to the random nature of the variable initialization, and may not be something that can be predicted or accounted for. Further trials would need to be run in order to confirm this, which time did not permit us.

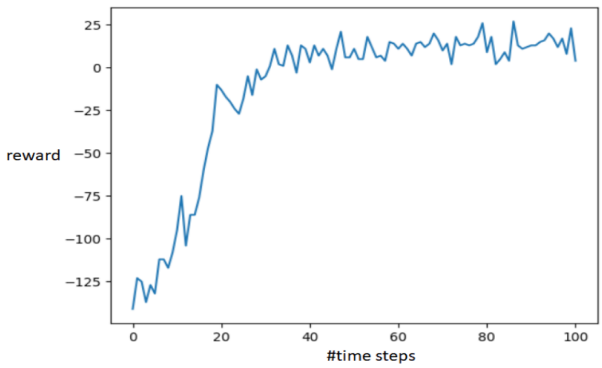


Figure 4. Results of training the left Pong paddle modeled by a Dueling DQN for 500,000 timesteps

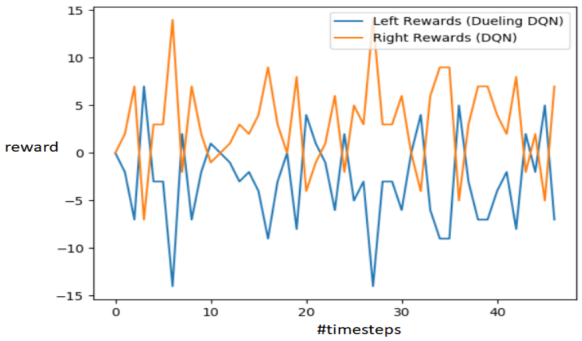
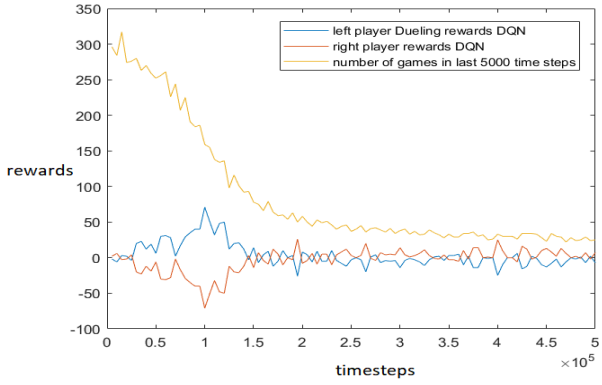
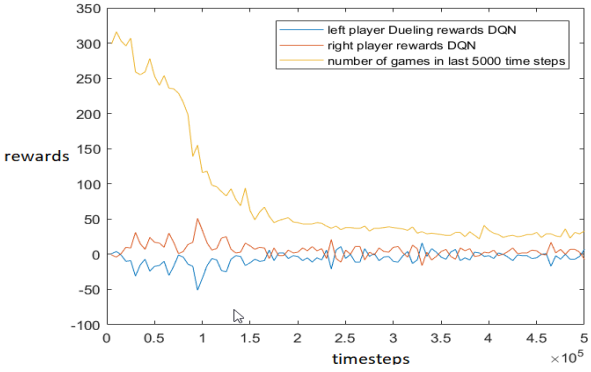


Figure 5. The reward differentials between DQN and Dueling DQN after 500,000 timesteps





5. Conclusion

This paper offered a new method of comparing different Deep Reinforcement Learning Algorithms. By deploying multiple agents with different algorithms to compete and learn form each other over time, we introduce a new metric by which to compare the strength of algorithms in a different type of environment. This may prove to be very important in applications where algorithms will need to compete with each other towards opposing goals. We applied this concept to compare the performance of the classic DQN architecture with the Dueling DQN architecture in an adversarial setting, finding that the DQN actually is stronger in this particular application.

This work can be extended into improving multiple algorithm training and comparison techniques. Multiple algorithms in the same environment can be used to work together and against. Additionally, different algorithms can be used to model different agents in other multi player markov games as well. This research has plenty of room for future expansion.

There are endless possibilities. These algorithms act differently while working together and differently when these algorithms are working against each other. Yes this is just the beginning of a whole new training and comparison techniques. Often we see the same algorithm working with itself and working against itself. This novel attempt to see how different algorithms work in the same environment together has been a success and should be tried in other practices too.

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[2] E. A. O. Diallo, A. Sugiyama, and T. Sugawara. Learning to coordinate with deep reinforcement learning in doubles pong game. *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 14–19, 2017.

[3] H. V. Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] e. a. Mnih V., Kavukcuoglu K. Human-level control through deep reinforcement learning. pages 1–2.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[8] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[9] W. Uther and M. Veloso. Adversarial reinforcement learning. Technical report, Technical report, Carnegie Mellon University, 1997. Unpublished, 1997.

[10] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.