DISTRIBUTED DEEP POLICY GRADIENT FOR COMPETITIVE ADVERSARIAL ENVIRONMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

This work considers the problem of cooperative learners in partially observable, stochastic environment, receiving feedback in the form of joint reward. The paper presents a flexible multi-agent competitive environment for online training and direct policy performance comparison. This forms a formal problem of a multi-agent Reinforcement Learning (RL) under partial observability, where the goal is to maximize the score performance measured in a direct confrontation. To address the complexity of the problem we propose a distributed deep stochastic policy gradient with individual observations, experience replay, policy transfer, and self-play.

1 INTRODUCTION

The recent development of Artificial Intelligence (AI) and Reinforcement Learning (RL) in particular demonstrate that AI agents are capable of learning to complete a variety of tasks using a reward feedback. It was shown that RL agents utilizing deep Q-learning outperform humans in many tasks such as Atari Games (Mnih et al., 2013; 2015; Guo et al., 2014; He et al., 2016). Furthermore, advanced layered solutions involving Alpha-Beta search, Reinforcement Learning, Monte-Carlo Tree Search, and Learning from Demonstration have been successfully applied to very challenging problems that previously were claimed intractable in the nearest future. This includes defeating the human world champion in the game of Go (Silver et al., 2017) and the worlds best players in DotA 2 (Bansal et al., 2017). However, real-world problems remain to be far more challenging than RL is currently capable of.

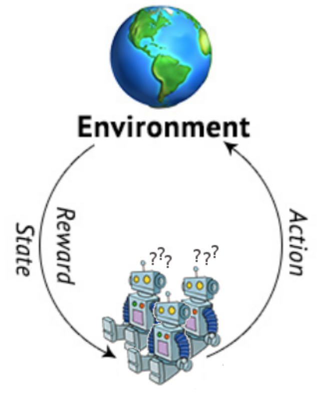

Figure 1: Mutli-agent Reinforcement Learning Framework

Many real-life tasks involve partial observability and multi-agent planning. Traditional RL approaches such as Q-Learning and policy-based methods are poorly suited to multi-agent problems. Actions performed by third agents are usually observed as transition noise that makes the learning very unstable (Nowé et al., 2012; Omidshafiei et al., 2017). Policy gradient methods, usually exhibit very high variance when coordination of multiple agents is required (Lowe et al., 2017). The complex interaction between the agents makes learning difficult due to the agent's perception of the environment as non-stationary and partially observable. Nevertheless, multi-agent systems are finding applications in high demand problems including resource allocation, robotic swarms, distributed control, collaborative decision making, real-time strategy (RTS) and real robots. But they are a substantially more complex task for online learning algorithms and often require multi-layer solutions.

This work considers the problem of cooperative learners in partially observable, stochastic environment, receiving feedback in the form of joint reward. The paper presents a flexible multi-agent competitive environment for online learning and policy performance comparison. We also introduce the multi-agent RL solution under partial observability problem, where the goal is to maximize the score performance measured in a direct confrontation.

2 Capture the Flag Environment

To throw together different control algorithms, we developed a complex dynamic environment we called Capture the Flag (CtF) that models a real combat scenario and mimics multi-player computer games. We made an effort to simplify the environment so it runs fast and requires minimum graphics, computation, and the number of dependencies. The simulation is open-source and works under OpenAI GYM software package (Brockman et al., 2016) that is a well-respected framework for training AI algorithms. In addition to the mentioned above, the CtF introduces the ability to run multiple AI policies playing against each other. This allows comparing not only the performance of the algorithms but also serves as a gladiator pit where the resulted policies can be evaluated in a direct confrontation. All these arguments make the CtF the perfect environment to train and test various competitive AI algorithms.

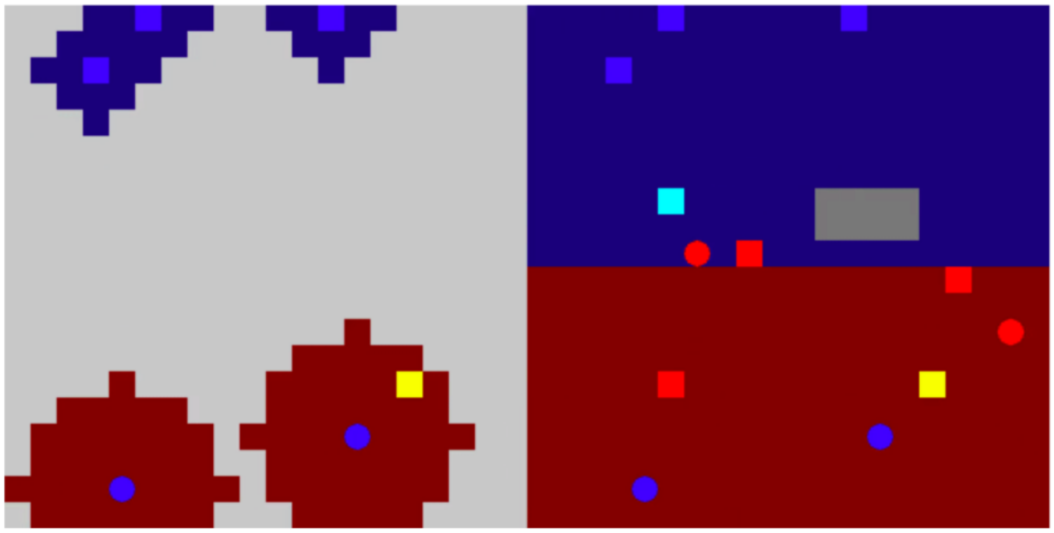

Figure 2: The proposed simulation is a gladiator pit for algorithms. Python-based environment Capture the Flag (CtF) is designed to throw together various AI algorithms or a human. Two teams confront each other and the goal is to capture the other team's flag or destroy the enemy units. Two types of units exist in each team and they have different abilities. The observation is limited by the fog of war shown on the left and provides the information available to the team. The true state of the environment shown on the right and reflects all the teams.

The package includes sample solutions that may challenge other project investigators. These solutions carry simple script-based heuristic behaviors that would help the researchers to train and test their adaptive AI algorithms. These algorithms may serve as building bricks for advanced adaptive solutions. For instance, we utilized the heuristic behaviors during the training process of the proposed RL solution. The policy-based RL algorithm described in this paper is also provided to the general audience. This algorithm may serve as a separate distributed solution or may be extended to

Type	Speed	Obs. Range	Attack Range	Map Code	Qty
UGV	1	3	2	2 or 4	4
UAV	3	5	0	3 or 5	2

Table 1: Type of units available to the team.

Hidden Space	-1
Blue Team Field	0
Red Team Field	1
Blue Team UGV	2
Blue Team UAV	3
Red Team UGV	4
Red Team UAV	5
Blue Team Flag	6
Red Team Flag	7
Obstacle	8

Table 2: Codes of the objects available in the observation.

form a hierarchical solution with a centralized planner. The layered solution currently is the topic of ongoing research and currently is under testing.

2.1 Environment Description

The CtF simulates a grid world of a finite size where two teams compete against each other trying to capture the other team's flag in conditions of partial information and visibility. The goal of each team is to survive, destroy the competitor or capture their flag.

Each team is represented by a number of units of a different type - unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs). Every unit carries certain properties such as an observation range, attack range, speed and different abilities as demonstrated in Table 1.

UGVs serve as a main actor of the game, they are allowed to defend their flag, capture the enemy's flag, fight the enemy UGVs, but they are limited in movements to the space that is free from obstacles and other units. UAVs serve as an observer of the system. They are allowed to move everywhere, they have a larger observation range and speed. The UAVs are invulnerable, but cannot attack or capture the flag.

All the objects in the environment are projected on the 2D map that is separated into two pieces: the blue team territory and the red team territory. These areas are equivalent and serve as parameter modifiers affecting the interaction of the units. For instance, the initial location of each unit is a uniform random draw limited by its team territory. Additionally, the territory affects the outcome of the local confrontation of the units by favoring those that belong to the same color.

The original map forms a complete state of the dynamical system, while the teams are provided with a partially observable state. The observation of each team is limited to the areas surrounding the friendly units while the rest of the map remains hidden. This is often called a fog of war and demonstrated in Figure 2. All visible regions are superposed to a single observation provided as a matrix of numbers representing the objects in accordance with the Table 2.

2.2 Problem Taxonomy

The selection of hyperparameters of environment allows to gradually increase the complexity of the problem. In this paper, we will be talking about the most sophisticated case as the most ambitious goal. With that in mind, the formal taxonomy of the problem becomes a partially observable multi-agent non-homomorphic planning Markov decision process (MDP) with simultaneous stochastic transitions and a stochastic reward function.

$$\text{State}_{\text{env}} \in \mathbb{S}_{\text{True}}^{20\times 20}, \text{ where } \mathbb{S}_{\text{True}} = \{0, 1..8\} \tag{1}$$

$$\text{Obs}_{\text{env}} \in \mathbb{Z}^{20\times 20}, \text{ where } \mathbb{Z} = \{-1, 0, 1..8\} \tag{2}$$

$$\text{Action}_{\text{env}} \in \mathbb{A}^{N}, \text{ where } \mathbb{A} = \{0, 1, 2, 3, 4\} \tag{3}$$

$$\text{Reward}_{\text{env}} = f(N_{\text{blue}}, N_{\text{red}}, N_{\text{flag}}), \in \mathbb{R}, \text{ where } \mathbb{R} = \{-100.. + 100\} \tag{4}$$

By default, the map is limited to 20×20 grid world, where each team controls 4 UGVs and 2 UAVs. These two types of units have different abilities what adds a non-homomorphism to the problem. The fully observable map shown in Eq. 1 is available to spectators for demonstration purpose only and represents the complete state space of the environment. The observation space available to the control algorithm is represented by a 2D vector of integers as shown in Eq. 2. As shown in Eq. 3, action vector is limited to the choice of five discrete actions per each unit and applied simultaneously to all the units including the enemy units. This adds a transition noise to the problem in contrast to such games as Chess and Go.

The reward system shown in Eq. 4 represents the current score of the game that is a step function of a number of units alive and flags available. For instance, once the friendly unit is lost, the score of the game changes to -10 and remains unchanged till the next score change happens. Once the enemy flag captured, it terminates the game and returns the score $+100$. Therefore, the reward is clamped in range $[-100..100]$.

3 Multi-Agent Deep Reinforcement Learning

The solution that we proposed in this paper is based on the Policy Gradient method (Williams, 1992; Sutton et al., 1998). It has been demonstrated that Policy-based methods outperform Q-Learning due to optimizing the expected reward directly and learning the explicit policy (Karpathy, 2016; Schulman et al., 2015). It follows the idea of a score function estimator shown in Eq. 9 that we use to update the model.

To address the complexity of the problem we created a distributed deep stochastic policy gradient. To make the training converging, we also utilized the following techniques: first-person view observation, experience replay, policy transfer and self-play.

$$\begin{aligned}
\nabla_\theta E_{s\sim\pi(s;\theta)}[R(s)] &= \nabla_\theta \sum_s \pi(s)R(s) && (5)\\
&= \sum_s \nabla_\theta \pi(s)R(s) && (6)\\
&= \sum_s \pi(s)\frac{\nabla_\theta \pi(s)}{\pi(s)} f(s) && (7)\\
&= \sum_s \pi(s)\nabla_\theta \log \pi(s)R(x) && (8)\\
&= E_s[R(s)\nabla_\theta \log \pi(s)] && (9)
\end{aligned}$$

Distributed policy gradient proposed in this paper utilizes a stochastic policy gradient algorithm REINFORCE (Williams, 1992) modified to fit multi-agent needs. The detailed algorithm is shown in Alg. 1. Each agent carries its own policy that defines the action choice with respect to the agent's current observation. Simulation of all the agents generates a game trajectory τ from initial global state S_0 to termination state S_T. This trajectory is decoupled into agent-centered trajectories

Parameter	Value
γ	0.98
α	0.0001
Episode length	100
Update freq	10
Experience buffer	50000
Batch size	2000

Table 3: Training hyperparameters

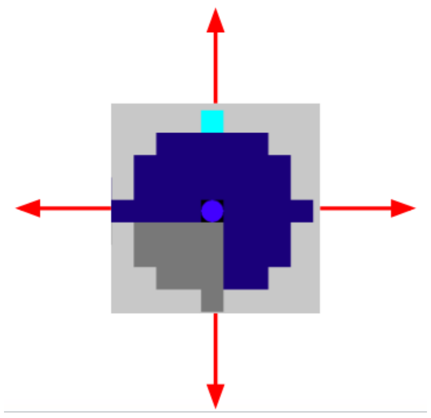

Figure 3: First-person observation. Global observation is split into individual agent observations and shifted to put the controlled agent into the center.

$\tau_1, \tau_2 ... \tau_N$ and then to their states $s_1, s_2 ... s_N$. The individual trajectories populate the experience replay buffer. The parameters of the algorithm are shown in Table 3.

```
**Data:** CtF simulation
**Result:** Optimal policy model $\theta^*$
Initialize $\theta \in \mathbb{R}^{\mathbb{D}}$;
**while** $k < N_{episodes}$ **do**
    Generate trajectory $\tau = \text{CtF}(\pi(\theta))$;
    Split $s_1, s_2 ... s_{\text{agents}} = \tau(t)$;
    Populate buffer $B$ with $\{s, R(s_1, s_2 ... s_{\text{agents}})\}$;
    **if** $k = K_{update}$ **then**
        Sample buffer $s, R, t \sim B(U(0, N_{\text{buffer}}))$;
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
        $\theta \leftarrow \theta + \alpha \gamma^t G \nabla \log(\pi(a_t | s_t; \theta))$;
    **end**
**end**
```

Algorithm 1: Multi-agent SPG with experience replay

The experience replay is a fairly common technique in deep RL that allows uniform parameter updates and, as result, a quicker convergence (Mnih et al., 2013). The replay buffer has been modified to accommodate the experience from multiple units at a time. We utilized a global reward as a metric of success for the policy that would allow the agents to learn to sacrifice themselves for a bigger reward. This final reward at the end of each experiment has been discounted by time and populated the experience buffer that was later uniformly sampled to update the policy model.

The first-person observations, in contrast to global observations, are rarely used in RL but make the total sense in the case of the CtF problem. The observations provided by the game don't draw a difference in the units. All the units of similar type look the same that makes it difficult to recognize which unit corresponds to which action input. The other reason to use first-person observations that we want to train self-sufficient agents that work in any number from 1 to N units and can be easily

Experiment	N Enemies	Observation	Pre-training
1	0	Full	No
2	4	Full	1
3	4	Part.	2
4	4	Part. + UAV	2

Table 4: Training procedure, experiments description

scalable. The first-person learning puts the controlled agent into the center of observation by shifting the map as shown in Figure 3. Moreover, it allows projecting observations of other friendly units onto the decoupled observations. This allows a convergence to a mutual (team-based) policy even with no the coordination between the units.

In order to gradually approach to the optimal policy, this work proposed to use a policy transfer approach where the policy was moved from a simple to a complex task. We split the training into several steps with respect to complexity and run the training separately one by one. The optimal policy has been transferred from task to task and adapted to the changes in the environment. As demonstrated in Table 4, we started with the case where the map is fully seen, no enemies. This trained the agents to approach and capture the flag. Then, we rerun the training with the enemy team. This time the agents learned to carefully approach the flag and defend their flag. Next, we trained on the partially observable scenario.

The initial training (in experiments 1 and 2) does not require a sophisticated opponent and may converge to optimal policies which are different from our target policy. Anticipating the effect, we introduced a Self Play technique that proved its efficiency in the game of Go (Silver et al., 2017). After the policy update, we upgraded the policy of the opponent team with the current best model. This constantly challenges the algorithm and pushes the best policy further.

In this research, the policy model has been created using TensorFlow package (Abadi et al., 2016). The design of the neural network model is demonstrated in Table 5 and represented by a deep convolutional network with a dense output layer operating a softmax activation function. Our model takes a 2D state vector size of 20×20 and returns a probability vector size of 5. These probabilities correspond to the actions that the agent prefers at the given input state.

Layer	Size
Input	20×20
CNN	$[5 \times 5] \times 32$
CNN	$[3 \times 3] \times 32$
Dense	400
Output	5

Table 5: Model structure

We understand that POMDP nature of the problem requires from us to use the temporal information from the environment. However, we deliberately chose to not utilize Deep Recurrent Q-Networks (DRQNs) Hausknecht & Stone (2015); Omidshafiei et al. (2017) due to the complexity of the task. The temporal component would significantly increase the complexity of the model and resulted policies. On the other side, we wanted to test an approximate solution using an observer design. As mentioned above, the UAVs are designed to provide a safe exploration and increase the observability of the problem. The assumption is that with an accurate design of the policy for the observer we can turn the problem into an approximation of a fully observable problem. The limitation in the number of UAVs is making the task challenging, especially when learning the policies of UAVs together with other units.

4 Results

We first evaluated the performance in the fully observable case with no enemies. As demonstrated in Figure 4, the policy converged to the optimal policy. With no enemies, the agents learned to navigate

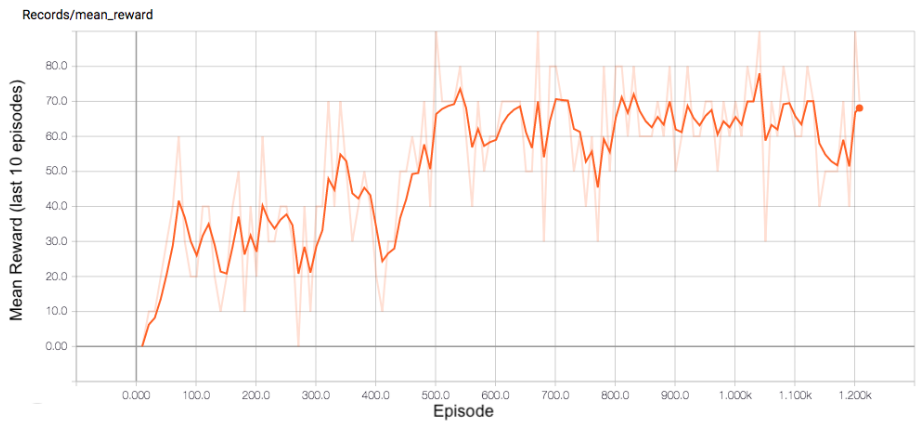

Figure 4: Performance of distributed policy learning with no competitors and fully observable environment.

through the world, avoid obstacles and capture the other team's flag. This navigation problem was an essential step for training the further steps.

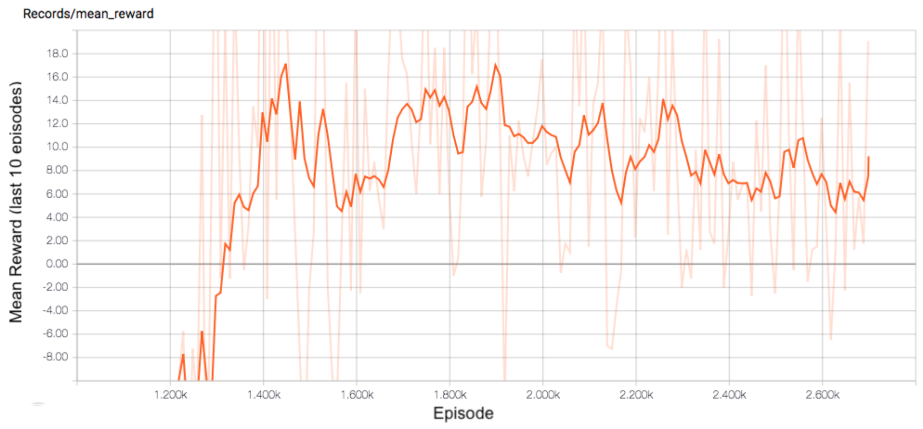

Figure 5: Performance of distributed policy learning with default number of competitors and fully observable environment.

With introduction of a competitor team, the performance of the policy drastically dropped to the negative score as shown in Figure 5. This was caused by the optimistic policy from step one that was trying to explore and capture the flag. The policy eventually learned the presence of the enemy units and adapted itself to perform a safer exploration and more conservative planning.

With added partial-observability, the policy demonstrated a very short peak in the performance that is shown in Figure 6. We explain this peak with that the agents stopped seeing the enemies and turned to use the policy previously learned in Step 1. This policy was over-optimistic and allowed reaching the higher score at the beginning due to less fear of enemies. However, on average, it quickly returned to the average of fully observable case with enemies that was more conservative. The next improvement of the performance was related to the fact that policy learned to use UAVs which previously have been unused. In partially observable case, the designing the observer helps to improve the performance of the system. The UAVs in this simulation serve as observers, giving some information that agents previously did not have access to. However, the lack of observers and temporal logic in the model caused the agents to forget the previously seen enemies.

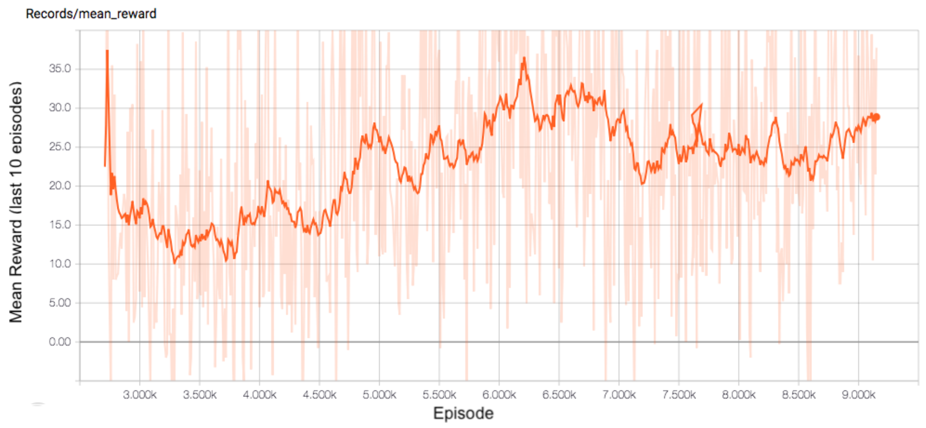

Figure 6: Performance of distributed policy learning with default number of competitors and partially observable environment.

5 Conclusion

In this project, we presented the new environment specifically designed for deep multi-agent learners. This environment allows to directly compare the performance of the policies and online training. The complex environment allows to design multi-layered solutions and throw them against each other.

We also demonstrated that the complex multi-agent POMDP problem can be efficiently solved using the distributed policy gradient method. The deep learning agent can learn a conservative policy in case of the limited information and even design the policy for its own observers to address the partial observability problem. The work also demonstrated, that the limitations in observers don't allow to turn the POMDP problem into fully observable and work only as an approximation. Addressing the temporal component by utilizing RNN networks or introducing a belief state into the input state would allow to further improve the performance.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pp. 3338–3346, 2014.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

Frank S He, Yang Liu, Alexander G Schwing, and Jian Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*, 2016.

Andrej Karpathy. Deep reinforcement learning: Pong from pixels. *url: http://karpathy. github. io/2016/05/31/rl*, 2016.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6379–6390, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

A Nowé, P Vrancx, YM De Hauwere, M Wiering, and M van Otterlo. Reinforcement learning: state-of-the-art. *Game Theory and Multi-agent Reinforcement Learning*, pp. 441–470, 2012.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. *arXiv preprint arXiv:1703.06182*, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.