Learning in Multi-Agent Games

David Zeng
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
`dzeng@andrew.cmu.edu`

George Lu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
`gclu@andrew.cmu.edu`

Abstract

Despite strong advances in reinforcement learning based agents in perfect information games such as Go, reinforcement learning has had difficulty in imperfect information games such as Poker. Multi-agent reinforcement learning poses many challenges for learning and this is especially true in imperfect information games. In this paper, we experiment with a method that combines a game-theoretic approach with reinforcement learning, Neural Fictitious Self-Play (NFSP), and apply it to imperfect information games with more than two agents. We also experiment with different variants of experience replay and adapt them to NFSP. We find that NFSP succeeds at learning in games with more than two agents.

1 Introduction

Multi-agent reinforcement learning focuses on learning when there are multiple agents interacting in an environment.

1.1 Problem

Multi-agent reinforcement learning can be studied in the context of imperfect information games. Many card games, such as poker, are imperfect information games. In these games, agents do not know the entire state of the game. For example, they might not know the cards of other players, or they might not know some hidden actions of other players. Thus, in contrast to perfect information games, the optimal strategy will be stochastic rather than deterministic.

In the imperfect information game setting, the goal is to design agents whose play approaches a Nash equilibrium, a set of strategies where no individual agent will benefit from changing their strategy. This can be thought of as the optimal solution to the game. In the case of symmetric zero-sum games, Nash equilibrium guarantees that there is no opponent strategy that is able to exploit, or get expected positive reward against our agent.

1.2 Related Works

Learning imperfect information games has been studied by Heinrich et al. in the context of poker in [4] and [3]. They use a game-theoretic approach.

Modifying replay memory as a method of increasing training stability for multi-agent reinforcement learning has been studied by Foerester et al. in [1]. A different approach to increase stability by He et al works on opponent modeling [2]. Finally, another approach adapts policy gradient methods to consider the policies of other agents [5].

2 Background

In this section, we discuss terms from game theory and how we model games for multi agent reinforcement learning.

Extensive Form Games Extensive-form games are a common model for games with multiple agents. In particular, they are able to model imperfect-information games. Each player only sees their own information state, which may correspond to many different game states. Each player's strategy can only depend on their information state.

For each agent i, let their strategy be π^i. We can then define a strategy profile $(\pi^1, \cdots, \pi^n)$ to be a set of strategies, one for each agent. For each agent, we often care about their best response against the strategies of all opponent agents. Let π^{-i} be the strategy profile π, but only including the strategies for all opponent agents. For each π^{-i}, there exists a best response strategy for agent π'.

Finally, a Nash equilibrium is defined as a strategy profile π such that where each agent's strategy π^i is also their best response strategy for π^{-i}.

Fictitious Play Fictitious play is a method of learning games using self-play. At each step, each agent will consider their opponent's past actions and compute the best response against their opponent's average strategy. This repeats and given enough time, the average strategies of agents under fictitious play has been shown to converge to a Nash Equilibrium under certain conditions.

Multiple Agent Reinforcement Learning and Games In the standard single agent reinforcement learning setting, the environment is modeled as a Markov Decision Process (MDP). In multi-agent reinforcement learning with independently learning agents, we wish to model the environment in a way such that the environment each agent experiences can be viewed as an MDP that is a function of other agent's strategies.

First, we consider games where all agents take turns simultaneously. We can model a game G, using a tuple $G = (S, Z, A, P, r, O, n, \gamma)$. S, A are the sets of states and actions. Z are the sets of states each agent can actually observe, which is used in the case of imperfect information games. The transition function P maps $S \times A^n \times S \to [0, 1]$. An action set $(a_1, \cdots, a_n) \in A^n$ is the set of actions taken by each agent. Since each agent has a different reward function, $r : S \times A^n \times [n] \to \mathbb{R}$ determines the reward for each agent. Finally, in the case of imperfect information games, $O : S \times [n] \to Z$ governs what agents will see, so $z = O(s, i)$ represents what agent i will see when the game is at state $s \in S$. Once the strategies of other agents are fixed, G can be viewed as an MDP by each agent, so single agent reinforcement learning techniques may be applied. We can view sequential turn games as special cases of simultaneous turn games.

Independent Learning One common approach to multi-agent reinforcement learning is independent learning. In independent learning, agents play against against each other and the strategies of each agent are trained separately. Independent learning is used in both cooperate and competitive multi-agent environments. However, the major downside to this is that the environments of each agent are no longer fixed, leading to training instability. In addition, as the number of agents increases, the instability also increases. Combating training instability is a focus of current research.

3 Methods

3.1 Environment

We implement two environments.

The first is a generalized version of rock paper scissors (RPS), where the reward structure is unknown to all agents. Each agent acts simultaneously and in the event that one agent wins against another agent, they receive r_i reward and the opposing agent receives $-r_i$, where r_i is a randomized constant that depending on the move i. Rewards are summed over all pairs of agents. While generalized RPS is not an imperfect information game, it contains elements of imperfect information such as unknown rewards and the dependence of agent's best response strategies on the strategies of other agents. As a result, the environment is a simple but effective test for NFSP.

The second is Leduc hold'em, generalized to more than 2 agents. Leduc hold'em a simplified version of poker played with a deck of cards. Each player receives a single card, after which a round of betting will occur, where players sequentially choose to call, raise, or fold. Afterwards, another card is revealed publicly before another round of betting occurs. Afterwards, the winner(s) are determined to anyone whose card's rank matches that of the revealed card, and failing that, whoever has the highest cards. Each player's reward is their net change in 'money' - the difference between the amount they won against the amount they bet. Traditionally, Leduc hold'em is played with 2 agents on a six card deck (3 ranks, 2 suits). With our multiplayer version, we modify this to be played with a deck of fifteen cards (5 ranks, 3 suits) played with five players. Bet sizes are fixed and we limit each round to have at most 3 bets. We encode the state information of Leduc Hold'em via a $2 \times 4 \times 5 \times 3 + 5 + 5$ 0/1 tensor, with 2 rounds, 5 players, potentially 4 turns per round, and 3 actions, while the latter two addends represent the river (if revealed), and the player's hidden card.

3.2 NFSP

We first describe our base implementation of NFSP, which is based on the NFSP from [4] with slight modifications to support multiple agents and different replay memories. At a high level, NFSP is an extension of fictitious play using neural networks as approximators. For each agent, an average policy network is trained to imitate their own average behavior. Simultaneously, a value network is trained to find the best response against their opponent's average behavior.

We train independent NFSP agents that play against each other. We use two separate replay memories for each network, $\mathcal{M}_{RL}$ and $\mathcal{M}_{SL}$. The value network is trained to learn a $Q(s, a|\theta)$, using deep Q-learning. Meanwhile, the average policy network is trained to approximate $\Pi(s, a|\theta^{\pi})$. $\Pi(s, a|\theta^{\pi})$ should approximate its average behavior. We use supervised learning to train Π. During self-play, we use a mixture of both networks to choose each agent's actions. The choice between which network to use on any given episode is randomized, and controlled by a η, referred to as the anticipatory parameter.

Algorithm 1 NFSP

```
 1: function RUN AGENT($\gamma$)
 2:     Initialize $\mathcal{M}_{RL}$, $\mathcal{M}_{SL}$
 3:     Initialize average policy network $\Pi(s, a|\theta^{\Pi})$ and value network $Q(s, a|\theta^{Q})$.
 4:     Initialize target network with parameters $\theta'^{Q} = \theta^{Q}$.
 5:     for each episode do
 6:         Set policy $\sigma$ to $\varepsilon$-greedy$(Q)$ with probability $\eta$ and $\Pi$ with probability $1 - \eta$
 7:         Observe initial state $s_1$.
 8:         for $t = 1, T$ do
 9:             Sample action $a_t$ from $\sigma$
10:             Send action $a_t$ to game and observe $s_{t+1}, r_{t+1}$
11:             if $\sigma = \varepsilon$-greedy$(Q)$ then
12:                 Store $(s_t, a_t)$ in $\mathcal{M}_{SL}$
13:             Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{M}_{RL}$
14:             if $t$ is a mini-batch update step then
15:                 Update $\theta^{\Pi}$ with cross-entropy loss, drawing from $\mathcal{M}_{SL}$
16:                 Update $\theta^{Q}$ with MSE loss on the q-learning update using the target network,
    drawing from $\mathcal{M}_{RL}$
17:             if $t$ is a target network update step then update $\theta^{Q'} = \theta^{Q}$
```

3.3 Architecture and Training

For our value network, we use a network with a single 30-unit hidden layer using ReLU activation. Our final output layer uses linear activation.

For our policy network, we also use a single 30-unit hidden layer with ReLU activation. Our final output layer is a softmax layer.

For both, we train with an SGD optimizer with no momentum.

Hyperparameters For the RPS environment, we used the following hyperparameters. We set the anticipatory parameter $\eta = 0.1$. We set the learning rate for the value network to 0.1 and policy network to 0.005. We use a buffer size of 1000 for $\mathcal{M}_{RL}$ and 50000 for $\mathcal{M}_{SL}$. The buffer size is relatively small due to the lack of meaningful state in RPS. We set ϵ to 0.5 initially and have it decay with number of iterations. We update the target network every 100 iterations and perform a batch update on both networks every 128 iterations.

For the Leduc Holdem environment, we increase the buffer sizes to 10000 and 500000 for $\mathcal{M}_{RL}$ and $\mathcal{M}_{SL}$ respectively. The increase in buffer size is due to the added complexity and state space of the game.

3.4 Replay Memory

NFSP in [4] requires two different replay memories. The first, $\mathcal{M}_{RL}$ is the replay memory used to train the value network. This replay memory consists of the full transition tuple $(s_t, a_t, r_{t+1}, s_{t+1})$.

The second, $\mathcal{M}_{SL}$ is used to train the average policy network. $\mathcal{M}_{SL}$ contains (s_t, a_t) pairs for when the action taken was following the best response policy, as estimated by the value network.

In [4], Heinrich et al. makes $\mathcal{M}_{RL}$ a circular buffer while using reservoir sampling for $\mathcal{M}_{SL}$. From a theoretical standpoint, we should be careful not to introduce bias in the sampling method used to train the average policy network. Reservoir sampling works well because it takes a uniformly random sample of past transitions. Thus, we primarily focus on modifying $\mathcal{M}_{RL}$.

Importance Sampling We use the importance sampling technique from [1] for the RL replay memory. As the size of the buffer for $\mathcal{M}_{RL}$ increases, the transitions stored in memory become increasingly outdated. Specifically, with multiple agents, the environment as seen by one agent changes as the behavior of its opponents changes. For some agent i, consider a transition (s, a, r, s') augmented with the states and actions of all agents $(S = (s_1, s_2, \cdots, s_n), A = (a_1, a_2, \cdots a_n))$. We want to weight the probability of sampling that transition with the ratio of probabilities of other agents taking A given S at the time the transition was added to replay memory to the probability at the time of sampling.

Formally, let $\pi_{-i}^{t_c}(a_{-i}, s) = \prod_{j \neq i} \pi_i^{t_c}(a_j, s)$. This is the probability of opponent agents taking their actions at the time of the transition. Similarly, we can define $\pi_{-i}^{t_s}(a_{-i}, s)$, the probability of opponent agents taking those actions at the time of sampling. We want to weight the probability of sampling the transition by $\frac{\pi_{-i}^{t_s}(a_{-i},s)}{\pi_{-i}^{t_c}(a_{-i},s)}$. Equivalently, we can redefined our standard mse loss function to be a weighted mse loss:

$$\mathcal{L}(\theta^Q) = \frac{\pi_{-i}^{t_s}(a_{-i}, s)}{\pi_{-i}^{t_c}(a_{-i}, s)} \left(r + \max_{a'} Q(s', a'|\theta^{Q'}) - Q(s, a|\theta^Q)\right)^2$$

One complication with implementing importance sampling in NFSP is that agents actions are sampled from two different networks. Since one network is a value-network, where actions are chosen greedily, this could lead to very large variance in importance ratios. As a result, when we calculate importance ratios, we only take the average policy network into account. For small enough η, this is a suitable and more stable approximation for the importance ratio.

Prioritized Replay We also implement Prioritized Replay as specified in [6]. This methodology seeks to solve the problem that experiences have varying levels of redundancy/quality, and that an agent can learn more effectively from some transitions than others. In addition, some transitions may not be immediately relevant to the agent, but might become so later. Prioritized replay seeks to fix this by weighting the probability of sampling a transition from replay memory by the magnitude of their TD error.

How this is implemented is that along with every transition $(S_{t-1}, A_{t-1}, R_t, S_t)$, we store each a parameter p_t which measures the absolute value of the TD-error of the term as a value representing the 'surprise' of seeing that term. We calculate this as

$$p_t = |R_t + \gamma \cdot Q_{target}(S_j, \texttt{argmax}_a Q(S_j, a|\theta^Q)) - Q(S_{j-1}, A_{j-1}|\theta^Q)|$$

When sampling from the replay, we sample each transition with probability

$$\frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where α is a hyperparameter such where greater α weighs increases sampling of more surprising values (observe $\alpha = 0$ is equivalent to normal experience replay).

3.5 Evaluation

We use two primary evaluation methods. The first is to measure performance against some fixed agent. The second is create an estimate of exploitability, or the performance of the best-response strategy against the fixed strategies of our trained agents.

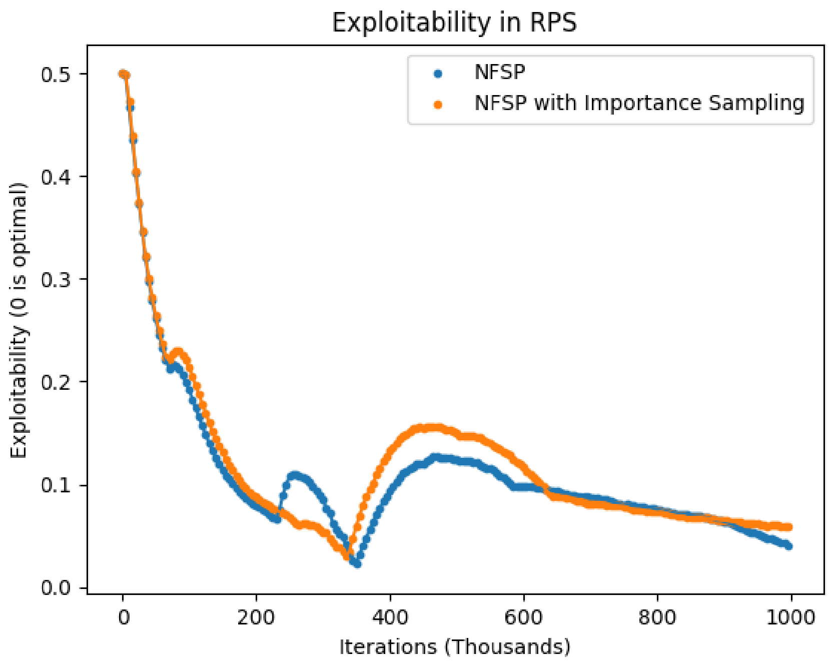
In the case of generalized rock-paper-scissors, we can explicitly calculate the optimal strategies against our trained agents. Formally, we can choose an arbitrary i, and calculate the best response strategy against π_{-i}, where π_{-i} is defined by the average policy network of each agent. We can then evaluate the performance of that best response. Thus, we have exact measure of exploitability.

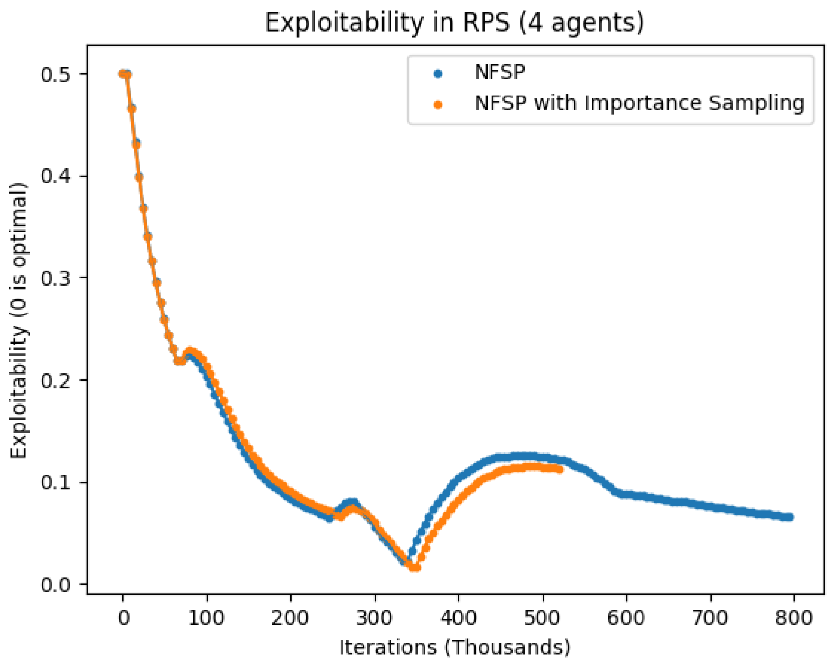
For the case of Leduc poker, this is slightly more challenging, especially with multiple (> 2) agents. We train agents against each other but test their performance against fixed strategy static agents. While this evaluation metric is not as exact as exploitability, improved performance against several different fixed strategy agents would demonstrate that the learned policies of our agents are robust against a few different strategies.

4 Results

4.1 RPS

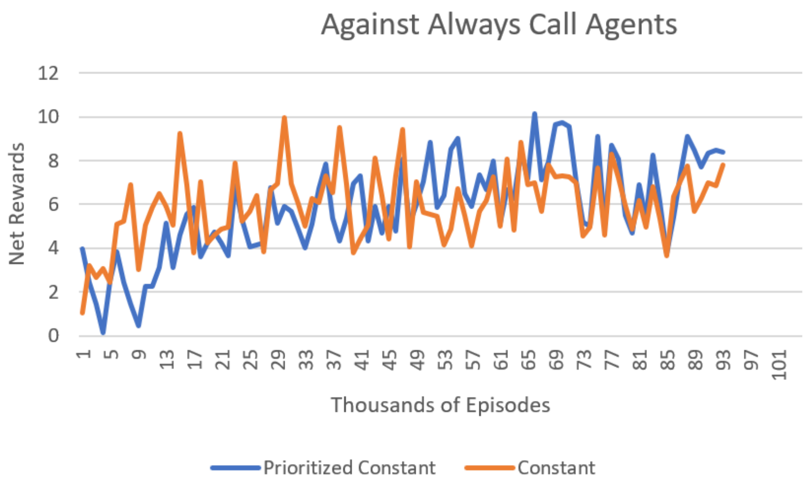
We first train with 2 agents for 1000000 iterations. We get the following results. Both NFSP appear to converge to a Nash Equilibrium, with importance sampling not appearing to have a significant impact. We also train with 4 agents and achieve similar training behavior. The reason for the "bouncing" behavior in exploitability is that the policy network is overshooting the optimal strategy. However, the policy network should be more stable as time progresses, and as shown in the graph, will eventually converge.

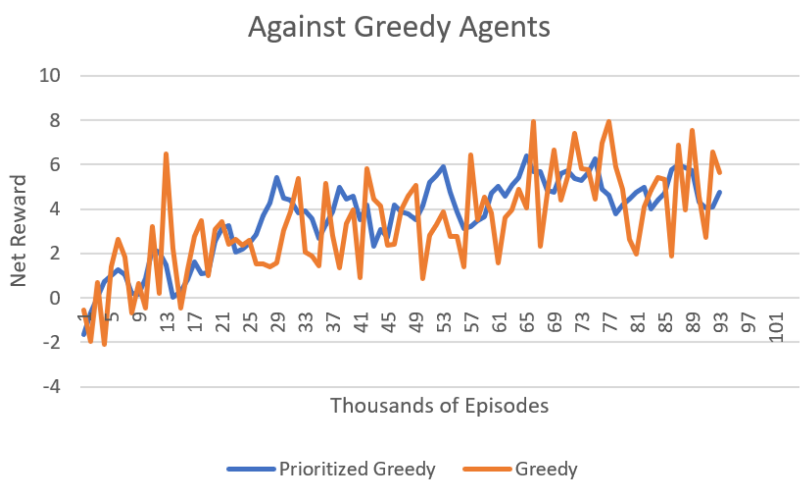


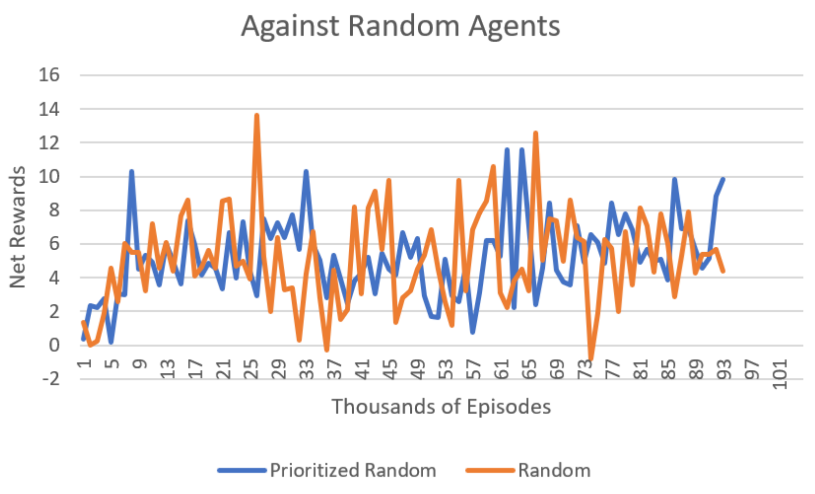


4.2 Leduc Poker

We trained the group of five agents for 100000 episodes ($\tilde{4}00000$ iterations per agent). Every thousand episodes, we evaluated our agents against three fixed strategies - always calling, raising high cards and folding low cards (referred to below as greedy) and random agents (who choose to raise, call, or fold uniformly at random)







We can see that NFSP very quickly goes above zero, which means it is outperforming the fixed-strategy agents, and in the case of greedy and always call agents, seems to show an overall trend of improvement (albeit with high variance that may simply be a product of the game) as our agent's strategies become more developed. One might note that this does not seems to be the case for play against random agents. This could be attributed to the fact that even in a nash equilibrium means that each agent is employing their best response strategy given that opponent strategies are fixed, but that does not mean that it is still the best response if other agents change their behavior.

We also see that prioritized replay does not seem to have had a noticeable effect on training. This could be due to the fact that while it theoretically picks instances which more quickly trains the state network, this in fact skews the distribution of experiences towards unlikely (surprising) events, which is does not necessarily maximize expected value.

5 Discussion

Our results suggest that NFSP can be adapted to environments with more than 2 agents without much additional work. In addition, preliminary testing suggests different replay systems may not improve performance. In Section 3.4, we discussed why changing replay memory for $\mathcal{M}_{SL}$ was not theoretically sound, but it was unclear whether changing replay memory for $\mathcal{M}_{RL}$ could improve performance. Our results suggest that changing $\mathcal{M}_{RL}$ did not have a impact on performance. In addition to the ideas above, another possible reason for the lack of impact is that while changing $\mathcal{M}_{RL}$ might impact the learning behavior of the value network, this impact may not translate into a significant impact on the learning behavior of the average policy network.

5.1 Future Work

Having shown the effectiveness of NFSP on simple environments with more than 2 agents, we would like to experiment on more complex environments. Techniques such as importance sampling might also be more beneficial when used in more complex environments.

6 Code Appendix

The environment and implementation can be found at: Github Repo

References

[1] Jakob Foerster, Nantas Nardelli, Greg Farquhar, Phil Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *ICML 2017: Proceedings of the Thirty-Fourth International Conference on Machine Learning*, June 2017.

[2] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé, III. Opponent modeling in deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1804–1813. JMLR.org, 2016.

[3] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 805–813. JMLR.org, 2015.

[4] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.

[5] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[6] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.