Analysis of Emergent Behavior in Multi Agent Environments using Deep Reinforcement Learning

Stefanie Anna Baby Ling Li Ashwini Pokle

Abstract

Reinforcement learning can provide a robust and natural means for agents to learn how to coordinate their action choices in multi agent systems. We examine some of the factors that can influence the dynamics of the learning process in such a setting. We also analyze the effectiveness of few multi-agent reinforcement learning algorithms and present the results of extending popular single agent algorithms to multi-agent game environments. We also report some of the collective behavior that arose via interactions of individual agents to yield distinct targeted results at the group level.

1. Introduction

Multi agent reinforcement learning is increasingly being used in different domains like robotics, distributed control, communication as well as games. In general, decision-making in multi-agent settings is complex due to the exponential growth of computational size with increasing number of agents. A lot of recent research work (Palmer et al., 2017) (Foerster et al., 2016) (Chu & Ye, 2017) (Lowe et al., 2017) has focused on using Deep reinforcement learning to effectively tackle this problem. Challenges like lack of Markov state assumption, exponential computation complexity of joint state-action space, full/partial observability of agents hinders convergence of multi-agent algorithms. However, given the right environment and an algorithm, agents discover and devise interesting strategies on their own through exploration and collective action. They use world knowledge to evolve distinct behaviors such as co-operation/ coordination that is well beyond the interplay of individual interactions. (Bansal et al., 2017)

The objective of this project is to analyze the evolution of complex behaviors in a multi-agent game setting and evaluate the effectiveness of some multi-agent reinforcement learning algorithms. We worked on three multi-agent environments for our experiments - Pursuit (MAgent) (Zheng et al., 2017), Battle (MAgent) and Gathering (custom environment).

We trained agents on these environments using parameter sharing DQN (Mnih et al., 2015) (Gupta et al., 2017), duelling DDQN (Van Hasselt et al., 2016), ACKTR (Wu et al., 2017) and Proximal Policy Gradients (Schulman et al., 2017). In addition, we also tried heuristics like training with expert demonstrations, prioritized experience replay and exploration bonuses through Context Tree Switching (CTS) (Ostrovski et al., 2017). We obtained good results when we trained agents with parameter sharing DQN, duelling DDQN, DRQN and Proximal Policy Gradients. We were unable to train agents when we used ACKTR and exploration bonuses. We present a detailed explanation and analysis of our results in section 4.

2. Related Work

Traditional single-agent algorithms cannot be directly applied to multi-agent cases primarily due to the possibility of divergence and instability of direct extensions. For instance, policy gradient methods that typically have high variance in gradient estimates get severely affected when we have multiple agents. It can be shown that the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents with policy gradient methods (Lowe et al., 2017). Robustness to decision making process of other agents is an important criterion in this setting as suggested by (Hu & Wellman, 1998). Several more studies have looked at multi-agent games for collective strategies, especially through cooperation (Lazaridou et al., 2016; Gupta et al., 2017), for example using optimistic Q learning and parameter sharing (Gupta et al., 2017). Other works have researched on issues like coordination of multiple agents (Le et al., 2017).

3. Approach

3.1. Deep Q Network

Deep Q Network (DQN) (Mnih et al., 2015) is a temporal-difference method that uses a multi-layered neural network to approximate state, action value function.

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

DQN relies on experience replay dataset D_t which stores

agent's experiences $e_t = s_t, a_t, r_t, s_{t+1}$ to reduce correlations between observations. The learning update at each iterations uses a loss function based on temporal difference update.

$L_i(\theta_i) =$

$E_{(s,a,r,s')\sim D}[(r + \gamma max'_a Q(s', a', \theta_i^-) - Q(s, a, \theta))^2]$

where parameters θ_i and θ_i^- are parameters of Q network and target Q network respectively at iteration i. In partially observable domains where only observations o_t are available at time t instead of the entire state st, the experience takes the form $e_t = (o_t, a_t, r_t, o_{t+1})$.

3.2. Double Q-learning

Standard Q-learning and DQN, uses the max operator over actions which means we use the same value both to select and to evaluate an action. This gives rise to the famed maximization bias where we more likely select overestimated values and end up with overoptimistic value estimates. Double Q-learning (Van Hasselt et al., 2016) prevents this by decoupling the selection from the evaluation.

Double Q-learning learns two value functions with weights θ and θ^- by assigning each experience ran- domly to update one of the two value functions. In every update, one set of weights greedily determines our policy and the other to determine action value. The Q-learning error can now be written as

$L_i(\theta_i) = E_{(s,a,r,s')\sim D}(r+$

$\gamma Q(s', argmax_{a'} Q(s', a', \theta_{i-1}^-), \theta_{i-1}) - Q(s, a, \theta_i))^2$

3.3. Prioritized Experience Replay

Sampling experiences transitions uniformly from a replay memory doesnt give any more significance to interesting observations, but simply replays observations at the same frequency that they were originally experienced. Learning can be done more efficiently if we replay important transitions more frequently. This is the objective of prioritized experience replay in Deep Q-Networks. Transitions have varying levels of importance - they may be surprising, redundant, task-relevant or not immediately useful to the agent. We can measure how surprising or unexpected a transition is based on the magnitude of its TD error δ that measures how far the value is from its next-step bootstrap estimate.

Prioritized DQN frequently replays those transitions with high expected learning progress as measured by δ. However this greedy prioritization focuses on a small subset of the experience and can cause loss of diversity. So we instead do stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling, with the probability of sampling transition i given by:

$P(i) = \frac{p_i^\alpha}{\Sigma_k p_k^\alpha}$

Here $p_i = |\delta_i| + \epsilon$ is the priority of transition i based on its TD error δ_i and α determines how much prioritization is used. Because prioritized replay changes the distribution in an uncontrolled fashion, it changes the solution that the estimates converge to. To correct this bias, we use importance sampling weights $w_i = (\frac{1}{N}\frac{1}{P(i)})^\beta$

and update the Q-learning equation using $w_i\delta_i$ instead of δ_i Here N is the size of replay memory and the exponent β is the importance-sampling correction. We annealed β from 0.5 to 1 towards using linear decay since the unbiased nature of the updates is most important only near convergence towards the end of training.

3.4. ACKTR

ACKTR is a sample efficient natural gradient algorithm for actor-critic methods. In order to minimise the non-convex loss function, we typically use gradient descent which depends on the parameterization θ of the model. This is not favorable because model parameters is an arbitrary choice and we do not want it to affect the optimization trajectory. Other policy gradient methods like the method of natural gradients that follows the direction of steepest descent is independent of θ, but also intractable. This is because it uses as its underlying metric the Fisher metric which is based not on the choice of coordinates but rather on the surface (manifold). Kronecker-factored approximated curvature (K-FAC) on the other hand is an approximation to natural gradient that is scalable and ACKTR extends this for actor-critic algorithms.

3.5. Proximal Policy Optimization

Proximal Policy Optimization is a new family of policy gradient methods that optimises a surrogate objective function using stochastic gradient ascent. The objective is to minimize the clipped loss:

$L^{CLIP} = E[min(r_t(\theta)\hat{A}_t], clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the policy parameters after and before the update, and $\hat{A}_t$ is the estimate of the advantage function at time t.

The motivation for clipping the probability ratio and taking the minimum is to get a lower bound on the unclipped objective scheme (which does conservative policy iteration). Without a clip/ check, maximizing the objective would lead to an excessively large policy update. With the clipped probability ratios, we form a pessimistic estimate of the performance of policy, ignoring only the change in probability ratio when it would make the objective improve. This improves the overall stability of the method and gave good convergence in our multi-agent environments.

4. Experiments and Results

We conducted our experiments on the following environments provided by MAgent platform (Zheng et al., 2017):

- Pursuit - This is a two-dimensional partially observable environment has a predator prey setting. There are N good agents (blue) that are faster and want to avoid being hit by adversaries (red). Adversaries are slower and want to hit good agents. Random obstacles like walls block the way. Predators have a circle of visibility of radius 5 and prey have visibility of radius 4. Prey are 1.5 times more faster than predators. Predators get a reward of -0.5 if they attack another predator. This environment is both competitive and cooperative. The adversaries need to learn to cooperate to catch prey. Each time the adversarial agents collide with a prey, the adversaries are rewarded while the prey is penalized.

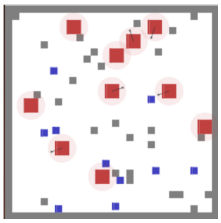

Figure 1. Initial configuration of "Pursuit" game.

- Battle - This is a two-dimensional partially observable environment in which two groups of N equally strong agents battle against each other. Each agent has a circle of visibility of radius 6. The attack range for each agent is a circle of radius 1.5. Agents get a reward of 5 if they kill an agent of opposite group. If an agent attacks a member of the same group, it gets reward of -0.1. Similar to Pursuit, this environment is both competitive and cooperative in which all agents in the same group should learn to cooperate to compete against the opposite group.

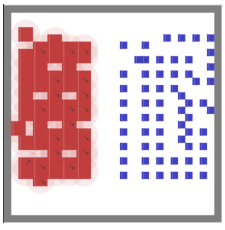

Figure 2. Initial configuration of "Battle" game.

In addition, we also performed few experiments on a custom environment inspired by "Gathering" environment in (Leibo et al., 2017). In this environment, there are two agents which compete for food. The agents can shoot each other and make the opponent disappear for t timesteps. We can also make the food scarce or abundant by controlling the number of timesteps for which it disappears. We used both fully observable and partially observable setting for our experiments in this environment.

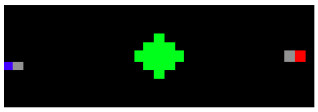

Figure 3. Initial configuration of "Gathering" game.

4.1. Results of experiments with DQN and its variants

1. **Setting**: Fully-observable "Gathering" environment with two agents. Each agent has complete information of the environment and has an independent DQN that is updated simultaneously after taking an action and observing a reward.

   Results: We observed that the DQN diverges in this setting. Loss for one of the agents increases exponentially, whereas loss for the other agent remains low. We realized that the issue with this is that each agent independently updates its policy as learning progresses. The environment appears non-stationary from the point of view of any one agent, violating the Markov assumptions required for convergence of Q-learning. This also prevents the naive application of experience replay.

2. **Setting**: Partially observable environment for "Pursuit" game with parameter-sharing Double DQN (Gupta et al., 2017) and duelling DQN. We used varying number of agents 5, 10 and 20 for this experiment. These agents were trained on a map of size 30X30. Initial positions of agents and walls were random. We used Adam Optimizer with learning rate 1e-4 and e-greedy strategy with piece-wise decay of epsilon from 1 to 0.2 in 600 epochs and from 0.2 to 0.05 over next 200 epochs. The size of experience replay buffer was set at 2^{22}.

   Results: We observed that the predators learned to cooperate and form complex structures like enclosures to trap the prey. An example of this strategy is shown in Figure 4 where the predators collectively act to corner and capture the prey. We also observed that the prey learn strategies to successfully escape the enclosures created by predator during initial stages of trap formation.

   Comparative analysis of performance of DQN and its variants

   From Figures 7 and 8 we can see that loss for predators first increases and then decreases. This trend might

be because predators receive negative rewards if they attack each other, therefore resulting in large values of loss initially. Eventually, they figure out a global strategy to capture prey and therefore loss decreases. We observe that the agents trained with DDQN and DRQN quickly learn the right behavior to cooperate and attack prey compared to the agents trained with DQN. In case of prey, we observe that loss keep increasing with epochs. This is because as time passes more prey particles will get captured by the predators.

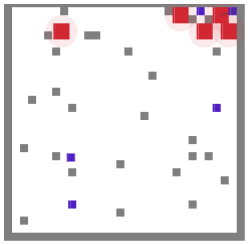

Figure 4. An example of predators (red particles) forming enclosures to successfully trap multiple prey

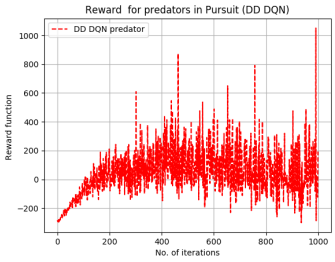

Figure 5. Rewards earned by predator over epochs in DDQN

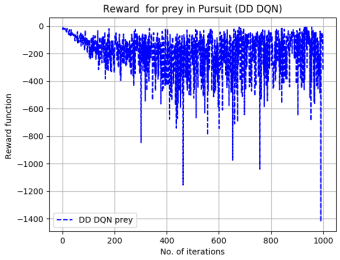

Figure 6. Rewards earned by prey over epochs in DDQN

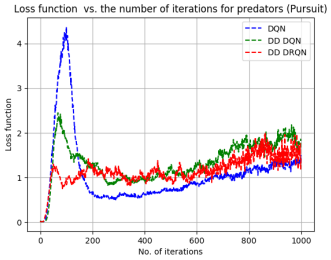

Figure 7. Loss over epochs for predators

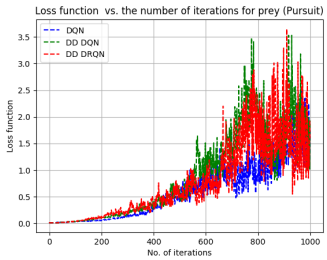

Figure 8. Loss over epochs for prey

3. **Setting**: Partially observable environment for ”Battle” game with parameter-sharing Double DQN and duelling DDQN. We used varying number of agents 5, 10 and 20 for this experiment. These agents were trained on a map of size 30X30. The values of other hyper-parameters were same as the corresponding values used in ”Pursuit”.

   Results: We observed that the particles learned interesting cooperative strategies to successfully capture and kill the particles of the opponent team. Two examples of global strategies are shown in Figure 9 where the attacking particles collectively act to corner and kill the opponents. From the corresponding reward plots, we can see that both types of particles were able to learn to kill opponents, but red particles managed to kill more blue particles through a global strategy and hence get higher total reward of around 200. Figures 15 and 14 show variation of loss and rewards for battle with increasing number of agents. As expected, rewards start with large negative values with increase in number of agents. This is observed because the agents are yet to figure out a collective global strategy. However, with time the agents learn to cooperate and attack the opponents and therefore loss decreases and rewards increase. With increase in number of agents, more complex attack formations are possible. Therefore we observe increase in total reward with increasing number of agents.

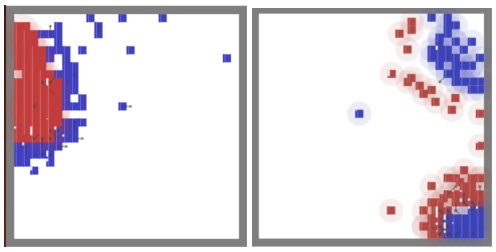

Figure 9. Examples of strategies learned by particles to trap the opponents in "Battle" game.

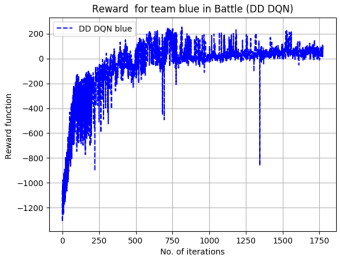

Figure 10. Rewards over epochs for blue particles in "Battle" game

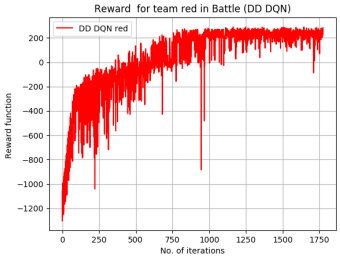

Figure 11. Rewards over epochs for red particles in "Battle" game

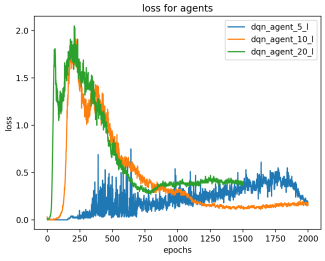

Figure 12. Comparison of loss for 5, 10 and 20 agents in "Battle" game

Figure 13. Comparison of rewards for 5, 10 and 20 agents in "Battle" game

4. **Setting**: Partially-observable "Gathering" environment with two agents. Each agent has an independent DQN that is updated simultaneously after taking an action and observing a reward. We used Adam optimizer and a replay buffer of size 2^{22}. We use e-greedy strategy with epsilon annealed uniformly over 1000 iterations from 1 to 0.1. Each agent can observe a square of size 10X10 around it.

   Results: We observed that the DQN converges in this case and both the agents learn a strategy to eat food. Partial observability helps with the Markov state assumption, and therefore aids DQN to converge. The results are in line with what was observed for Pursuit and Battle.

4.2. Results of experiments with duelling DDQN with prioritized replay

Setting: Partially observable environment for "Pursuit" game with parameter-sharing duelling Double DQN and prioritized experience replay. We trained both the algorithms on 10 agents. We used replay buffer of size 2^20. The parameter α was set to 0.7 and parameter beta was uniformly increased from 0.5 to 1 over 1000 steps.

Results: We obtain better rewards with prioritized experience replay. This might be because prioritization helps to sample those experiences that aid the predators to quickly learn better strategies and obtain higher rewards.

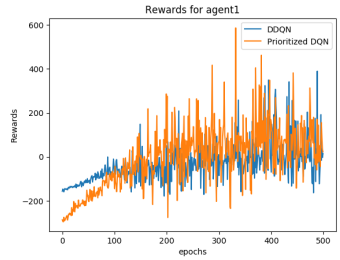

Figure 14. Comparison of rewards obtained by predator particles for duelling DDQN with and without prioritized experience replay in "Pursuit" game

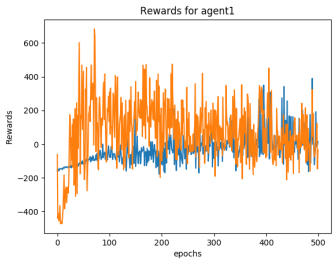

Figure 15. Comparison of rewards obtained by predators in "Pursuit" when duelling DDQN is pretrained with expert demonstrations.

4.3. Results of experiments of pre-training duelling DDQN with expert demonstrations

Setting: Inspired by the recently proposed work by (Hester et al., 2018), we decided to initialize experience replay buffer of duelling DDQN with expert experiences. To fetch expert experiences, we trained a duelling DQN for 2000 epochs and assumed it to be an expert agent. We then initialized the replay buffer of a new duelling DDQN with rollouts obtained from running this expert agent in the environment and filled the replay buffer of this new DQN with the resulting experiences. The values of other hyperparameters for duelling DDQN were same as the corresponding values used in previous experiments.

Results: We expected the transfer learning to work in this case. We observed that the predators initially learn very well when presented with expert demonstrations. However, over time the performance of training with expert demonstrations is similar to that of the normal duelling DDQN. We observe negative transfer learning. We hypothesize that this might be due to one of the two reasons. First, since size of the replay buffer is large 2^{22}, not all experiences stored in it might be optimal expert demonstrations. A large number of these experiences are very likely to be suboptimal. Second, since we are storing collective experiences of all the agents in a single replay buffer, transfer learning might not be very easy in the multi-agent scenario.

4.4. Results of experiments with ACKTR

Setting: Pursuit game with 10 agents. All the agents that belong to same class share a single actor and a single critic. This setting, though not ideal, was imposed due to the restriction of MAgent environment. The hyperparameters - Value coefficient was set to 0.1 and entropy coefficient was set to 0.08. We used KFAC Optimizer with learning rate of 1e-3 and batch size of 64.

Results: Though ACKTR is supposed to be sample efficient, we did not observe any cooperative behavior being learned by our agents. We think that this might be mostly due to our use of a single actor as opposed to multiple actors.

4.5. Results of experiments with DQN with CTS exploration bonus

Setting: Pursuit game with similar configuration as the previous experiments. However, each agent was initialized with its own CTS density model i.e. though network parameters are shared, each agent has it's own independent CTS density model. This choice was mostly due to the fact that CTS computes exploration bonuses on the basis of pixels. As this environment is partially observable, the frames observed by each individual agent is different and therefore each agent should receive a different exploration bonus. We set the value of η to 0.9 and β to 0.05. As our observation space is small, we used density model with 5 bins as opposed to 8 bins originally used in the paper by (Bellemare et al., 2016).

Results: Providing CTS exploration bonuses did not help agents to learn a cooperative global behavior. CTS has performed extremely well in fully observable environments with a single agent, however it might not be suitable for use in multiagent environments, as there are additional factors like interactions between agents that are not accounted by CTS while computing exploration bonus.

4.6. Results of experiments of with Proximal Policy Optimization

Setting: Fully-observable "Gathering" environment with two PPO agents. Episode length was fixed at 500 timesteps. We used Adam optimizer with learning rate of 1e-4 and batch size of 1000. We experimented with two policy network architectures for PPO. First architecture was four fully connected layers followed by softmax over four possible actions. Second architecture was similar to the DQN architecture used in (Mnih et al., 2015).

Results: The network initially did not learn useful behavior when implemented with a simple MLP architecture. However, it was able to learn useful policy with CNN based architecture. We experimented with two scenarios for this experiment. In the first case we created scarcity of food by making the food disappear permanently once its eaten. In the second case we made the food reappear after 10 timesteps after it is eaten. In both the cases, an agent can shoot the other agent to make it disappear for 250 timesteps. We observed that in the first case, agent learns to shoot the other agent to get more rewards, whereas in the second case, agents do not shoot each other as there is infinite supply of food. We have plotted minimum life span of agent in Figure 16

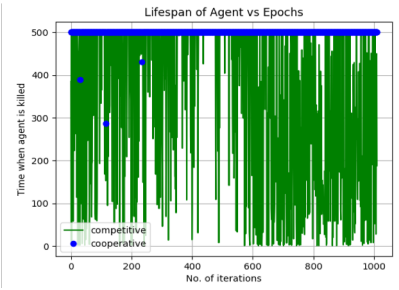

Figure 16. Plot of minimum lifespan of agents in Gathering environment when there is extreme scarcity of food (blue) and when there is infinite supply for episode of length 500.

5. Conclusion

Although we tried several different methods based on policy gradient, count based technique and gradient approximation, not everything converged and/or improved learning. However, several of the simpler algorithms did show convergence and resulted in collective behavior among the agents. We believe with better optimization strategies fine tuned for multi-agent games, our agents will be able to learn and depict more complex swarm behavior.

Demonstrations of collective behavior in Battle and Pursuit can be accessed in the link below [1]

References

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

Bellemare, Marc, Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Remi. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Chu, Xiangxiang and Ye, Hangjun. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

Foerster, Jakob, Assael, Ioannis Alexandros, de Freitas, Nando, and Whiteson, Shimon. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

[1] https://drive.google.com/drive/folders/1VW1AY1ddYvVk7nautDXPnV0-4Xa0uSqV?usp=sharing

Gupta, Jayesh K, Egorov, Maxim, and Kochenderfer, Mykel. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 66–83. Springer, 2017.

Hester, Todd, Vecerik, Matej, Pietquin, Olivier, Lanctot, Marc, Schaul, Tom, Piot, Bilal, Sendonaris, Andrew, Dulac-Arnold, Gabriel, Osband, Ian, Agapiou, John, et al. Deep q-learning from demonstrations. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.

Hu, Junling and Wellman, Michael P. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the second international conference on Autonomous agents*, pp. 239–246. ACM, 1998.

Lazaridou, Angeliki, Peysakhovich, Alexander, and Baroni, Marco. Multi-agent cooperation and the emergence of (natural) language. *arXiv preprint arXiv:1612.07182*, 2016.

Le, Hoang M, Yue, Yisong, and Carr, Peter. Coordinated multi-agent imitation learning. *arXiv preprint arXiv:1703.03121*, 2017.

Leibo, Joel Z, Zambaldi, Vinicius, Lanctot, Marc, Marecki, Janusz, and Graepel, Thore. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pp. 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

Lowe, Ryan, Wu, Yi, Tamar, Aviv, Harb, Jean, Abbeel, OpenAI Pieter, and Mordatch, Igor. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6382–6393, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Ostrovski, Georg, Bellemare, Marc G, Oord, Aaron van den, and Munos, Rémi. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

Palmer, Gregory, Tuyls, Karl, Bloembergen, Daan, and Savani, Rahul. Lenient multi-agent deep reinforcement learning. *arXiv preprint arXiv:1707.04402*, 2017.

Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pp. 2094–2100, 2016.

Wu, Yuhuai, Mansimov, Elman, Grosse, Roger B, Liao, Shun, and Ba, Jimmy. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5285–5294, 2017.

Zheng, Lianmin, Yang, Jiacheng, Cai, Han, Zhang, Weinan, Wang, Jun, and Yu, Yong. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *arXiv preprint arXiv:1712.00600*, 2017.