CS7642 PROJECT 3 REPORT – FOOTBALL

Dixon Domfeh
(dnkwantabisa3@gatech.edu)
4e100281799ba23b6373dac29722d889aa8ea1d0 (git hash)

***Abstract*—In this report I will present a full account of implementing and evaluating two multi-agent reinforcement algorithms in the competitive Google Research Football environment. After much deliberation, I chose *multi-agent soft actor critic* (mSAC) and deep deterministic policy gradient (MADDPG) deep reinforcement learning techniques. I also present my justification of choosing the algorithms.**

I. Introduction

Most of the algorithms studied in the course so far (TD, SARSA and Q-learning) do not naturally extend to the multi-agent case without modifications. In single-agent case of reinforcement learning (RL), other actors (agents) within the environment as considered as part of the dynamics of environment in which the agent operates. Although such an assumption makes it feasible to solve certain problems in RL, it does not work well on complex real-life problems where there is both cooperation and competition. Specifically, the simultaneous learning of multi-agent raises the issues of non-stationarity which has been addressed in the literature by clever paradigms of learning. For methods such as Q-learning, non-stationary presents learning stability challenges which prevents the use of experience replay that are important for stabilizing deep Q-learning. Policy gradients on the other hand usually exhibit high variance in multi-agent environments. Additionally, in cooperative and competitive environments the issue of credit assignment is prevalent. In multi-agent cooperative games, individual agent actions contribute to a final reward. Hence, there is a need to identify the contribution of each agent towards the total reward. However, there have been advances in solving the multi-agent problems in recent literature. One of such paradigms is centralized learning and decentralized execution (CTDE) (see for example, Foerster et. al, 2018). Several ablations of the CTDE concept of learning have been explored (see example Haarnoja et. al, 2018, Yuan et. al, 2021, Foerster et. al, 2018) for both on-policy and off-policy reinforcement learning algorithms. In this report I explore the two of such algorithms – multi-agent deep deterministic policy gradient (MADDPG) and multi-agent soft-actor-critic (mSAC) – to improve of the learning capabilities of the agents in the Google football environment. I argue that MADDPG is a good method for improvement agents' performance because it considers that competitive as well as the cooperative nature of the Google football environment. mSAC also works equally well in a multi-agent environment.

II. Methodology

A. *Naïve deep Deterministic Policy Gradient (DDPG)*

We motivate our discussion of deterministic policy gradient by first considering a simple case of a single-agent and introduce a few notations. DDPG borrows a lot of tricks from deep Q-Networks (DQN) which make use of an action-value function for a policy $\boldsymbol{\pi}$ such that $Q^{\boldsymbol{\pi}}(s,a) = \mathbb{E}[R|s^t = s, a^t = a]$. DDPG can be seen as a DQN for continuous action space. DDPG uses a replay buffer to train action-value function as in DQN in an off-policy manner. It also uses a target network to stabilize learning. The main difference between DDPG and DQN is that while DQN uses a target Q-function to take greedy actions, DDPG utilizes a target deterministic policy function that is trained to approximate that greedy action. To refresh our memory, the objective value function of DQN is

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathfrak{D})}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta_i)\right)^2\right]. \quad (1)$$

For DDPG, the same objective in equation (1) can be used with a slight change in notation as

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathfrak{D})}\left[\left(r + \gamma\, Q(s',\mu(s';\phi^-);\theta^-) - Q(s,a;\theta_i)\right)^2\right] \quad (2)$$

So instead of taking the argmax according to the Q-function in equation (1), we learn a policy $\boldsymbol{\mu}$ for the deterministic greedy action for the state in question from the target network. The objective of the deterministic policy can be learned by maximizing the expression below

$$J_i(\theta_i) = \mathbb{E}_{s\sim\mathcal{U}(\mathfrak{D})}[Q(s,\mu(s;\phi);\theta)] \quad (3)$$

where we query the policy (ϕ) for the best action in a particular state and then query the Q-function for the Q-value using states samples from the replay buffer.

Since the off-policy methods such as DDPG train a deterministic policy, they can only achieve exploration by injection some random noise (typically Gaussian) to their actions vectors going into the environment.

B. Multi-agent Deep Deterministic Policy Gradient (MADDPG)

The MADDPG is an extension of the DDPG with a centralized or shared critic algorithm which was introduced by Lowe et al, 2020. It is built on the CTDE learning paradigm to specifically address the weaknesses of Q-learning methods (see discussion in section (I) in multi-agent settings. The authors specifically considered actor-critic methods that use action policies of other agents to successfully learn policies that require complex multi-agent cooperation. MADDPG trains an ensemble of policies for each agent which leads to robust multi-agent policies. Lowe et al. (20120) also show that MADDPG works well in cooperative and competitive games, hence my reason for choosing it to improve the agents in the football environment. In MADDPG, agents learn policies using local information or observation (i.e., based on the agents' field of vision) at execution time. In other words, after training is complete only local actors are used at execution time. Cooperative behavior is encouraged at execution time while competitive behavior is encouraged at training time.

In MADDPG, the actor-critic policy gradient methods are extended to include extra information about the policies of other agents for the critic while the actor only acts using local information (Lowe et al, 2020). Perhaps the most desirable feature of MADDPG is that since centralized critic functions learn about the policies of other agents, they can learn to approximate models of other online agents well and use that information effectively in the learning procedure. It is worth mentioning some of the similarities and difference between MADDPG and a close learning method – COMA. COMA (Foerster et. al, 2018) addresses the credit assignment problem in multi-agent RL by using a counterfactual advantage function. Value functions are used to solve the credit assignment problem. However, in multi-agent cooperative games, individual agent actions contribute to a final reward. Hence, there is a need to identify the contribution of each agent towards the total reward. COMA uses single centralized critics for all agents to calculate the advantage function per agent through marginalization while keeping the other agents fixed, whereas the MADDPG method uses a centralized critic for each agent. This allows for varying reward functions suitable for competitive scenarios. COMA combines recurrent policies with feed-forward critics whereas MADDPG considers explicit communication channels between agents. Figure 1 below presents an overview of the multiagent actor-critic with decentralized actor and centralized critic approach.

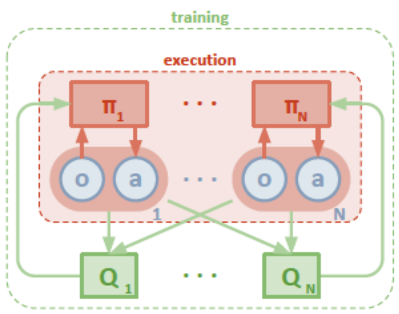

Figure 1: Overview of the actor-critic approach used in MADDPG. Source: Lowe et al., 2020.

Let us consider a game of N agents with policies parameterized by $\boldsymbol{\theta} = \{\theta_1, .., \theta_N\}$ with a set of policies $\boldsymbol{\pi} = \{\pi_1, .., \pi_N\}$. Then we can rewrite equation (3) to include the gradient of the expected return for agent i as

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\mu}, a_i \sim \boldsymbol{\pi}_i}\left[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathrm{x}, a_1, .., a_N)\right] \quad (4).$$

In equation (4), $Q_i^{\boldsymbol{\pi}}(\mathrm{x}, a_1, .., a_N)$ is the centralized action-value function which takes inputs from all other agents in addition to some state information, x. Because the $Q_i^{\boldsymbol{\pi}}$ is learned separately by each agent we can expect arbitrary reward structures including conflicting rewards which is desirable in competitive games (Lowe et al., 2020). Consequently, the centralized action-value function can be updated similarly as in equation (2) using the expression

$$L_i(\theta_i) = \mathbb{E}_{(x,a,r,x')\sim \mathcal{U}(\mathfrak{D})}\left[\left(Q_i^{\boldsymbol{\mu}}(\mathrm{x}, a_1, .., a_N) - y\right)^2\right]$$
$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}}(\mathrm{x}, a_1, .., a_N)|_{a_j^{'} = \boldsymbol{\mu}_j^{'}(o_j)}$$

where $\boldsymbol{\mu}' = \left\{\boldsymbol{\mu}_{\theta_1'}, \dots, \boldsymbol{\mu}_{\theta_N'}\right\}$ is the target policies.

In MADDPG, if we know that actions taken by all other agents then the environment is stationary (addressing the issue of non-stationarity in multi-agent RL) even though the policies change at the individual agent level. In other words, $P(s'|s, a_1, .., a_N, \boldsymbol{\pi}_1, .., \boldsymbol{\pi}_N) = P(s'|s, a_1, .., a_N) = P(s'|s, a_1, .., a_N, \boldsymbol{\pi}_1{}', .., \boldsymbol{\pi}'_N) \; \forall \, \boldsymbol{\pi}_i \neq \boldsymbol{\pi}'_{\boldsymbol{i}}$. Knowing other agents' policy is in practice is difficult. Instead, we can approximate these policies of each agent i, by $\hat{\boldsymbol{\mu}}_{\phi_i^j}$ (where the ϕ is a set of parameters for the approximation). See Lowe et al. (2020) for more details on the how the approximate policy is learned.

C. Multi-agent Soft Actor-Critic (mSAC)

I also considered an another off-policy algorithm – mSAC – for policy improvement in the football environment. I reason that since MADDPG trains a deterministic policy with no explicit exploration except the addition of Gaussian noise to the action-space; it would be beneficial to also test algorithms that are still off-policy but directly encourages exploration behavior. In its single-agent form, SAC is quite stable and has achieved state-of-the-art performance on most continuous action space problems. SAC is a hybrid of two RL paradigms to address some of the challenges of model-free deep RL (see Haarnoja et. al, 2018). One is the *off-policy* paradigm which trains a deterministic policy using experiences generated by a behavior policy different from the policy to be optimized. The second paradigm is *on-policy* method that trains a stochastic policy which introduces randomness, and hence, exploration. The SAC algorithm is an *off-policy* algorithm, but it trains a stochastic policy. The most important characteristic of SAC is that the objective of the agent is to maximize a joint total expected reward and total entropy. This mechanism naturally encourages exploration behaviors that are diverse while still maximizing the expected return. More formally, the agent is tasked to maximize the objective function below

$$q_\pi(s,a) = \mathbb{E}_{r,s'\sim P(s,a),a'\sim\pi(s')}\left[r + \gamma\left(q_\pi(s',a') + \alpha\mathcal{H}\left(\pi(\cdot\,|s')\right)\right)\right]. \quad (5)$$

In equation (5) we take the expectation of the immediate reward, the discounted value of the next action-state pair plus the entropy of the next state. The term α, is a tunable parameter which determines the importance of the entropy term and therefore controls the stochasticity of the policy. Notice that the conventional objective function for RL in the above equation can be recovered in the limit as $\alpha \to 0$.

The multi-agent SAC uses the same underlining objectives from the single agent case but with modifications that incorporate CTDE. Yuan et. al (2021) recently introduced a decomposed SAC for cooperative multi-agent RL. I argue that this approach is quite suitable to improve the agents in the Google Research Football environment. Yuan et. al (2021) evaluate the performance of several ablations of their method, but in this report, we specifically use mSAC since the other variants are not available in the ray RL library.

D. Adversary Agent Using PPO

Finally, I introduce the adversary or opponent agent that the two algorithms described in the preceding section is going to compete against – Proximal Policy Optimization (PPO). In contrast to my choice of algorithms for policy improvements, PPO is an *on-policy* algorithm. Each agent of the opposing team uses an untrained a PPO algorithm without cooperative among team members. It would be expected that a well-coordinated and cooperative algorithm such as MADDPG and mSAC would outperform PPO.

The PPO algorithm is an improvement of plain policy gradient algorithms such as A2C. PPO introduces clipping which restricts the optimization steps. Clipping puts a limit of the policy objective so that at each learning step, the policy is only allowed to wander within a tolerable space (hence reduced variance). In plain policy gradient, small changes in parameter space often leads to big differences in performance, hence the reason to use small learning rates. Clipping stabilizes performance. PPO is sample-efficient even though it is an *on-policy* algorithm. It creates a large mini batch (replay buffer) to sample from on each optimization step. Stochasticity is still maintained since in each mini batch the samples are not the same.

III. EXPERIMENTS & RESULTS

In this section, I present the experiments conducted and some analysis on the results.

A. Google Research Football environment

The Google football environment is a physics-based 3D football simulation. Although, originally designed as11-by-11-player soccer game with full standard rules, in used a reduced version -- 3-by-3 players including a goalkeeper in this report. The action space consists of 19 actions for example move right, short pass, long pass, left, right etc.

B. Experiments

I first started with the MADDPG. The MADDPG presented in section II works for a continuous actions space which is not compatible with Google football environment with discrete action space. I had to modify the environment to take discrete actions. The policy network for all agents is parameterized by two-layer ReLU MLP with 64 hidden units per layer. Table 1 below give a summarize of the some of the final hyperparameters used. We maintained most of the default configurations in the ray RL library.

Table 1: Hyper-parameters for MADDPG

Parameter Name	**Value**
Replay buffer capacity	1e6
Actor learning rate	3e-4
Critic learning rate	1e-7
Training batch size	1024
Learning starts	1024×5
Stopping criteria	1e6 steps

For the mSAC implementation, I utilized a 256×256 fully connected MLP for both the target and policy networks with a ReLU activation. Table 2 give a summary of the hyperparameters used.

Table 2: Hyper-parameters for mSAC

Parameter Name	**Value**
Discount factor	0.99
Actor learning rate	3e-4
Critic learning rate	3e-4
Training batch size	1024
Learning starts	1500

Stopping criteria	1e6 steps
Soft target Q update	5e-3

The PPO implementation was ran as given without any changes to parameter inputs.

C. Results and Discussion

The results obtained from my experiment were not as expected. Both MADDPG and mSAC surprisingly performed poorly against the PPO. I hypothesize that the hyperparameter tuning could be the cause. I spent several days manually selecting and varying hyperparameter value in the hopes of getting better results but to no avail. Apparently, there are ways of automatic tuning in the ray RL library which I didn't have time to apply due to time constraints. Notwithstanding the bad performance on my experiments, I proceed to compare some the results below. Since the goal of the football game is to score the number of goals (in the form of rewards) and ultimately win the game, I compared the algorithms based on mean rewards and maximum rewards per episode. My idea was to observe the convergence stability of the rewards during training. In Figure 2 (a)-(c) I show the maximum and mean rewards for MADDPG, PPO and mSAC respectively.

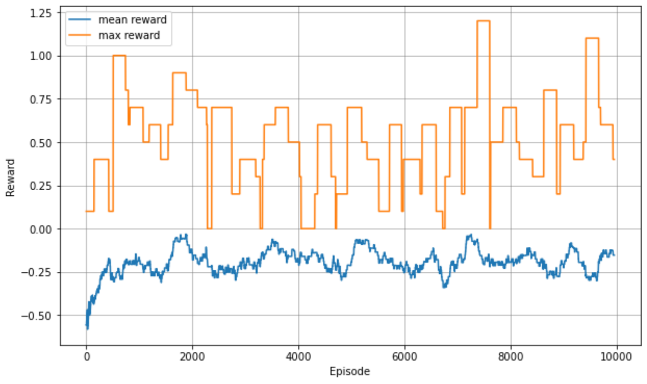

Figure 2(a): Reward per episode for MADDPG

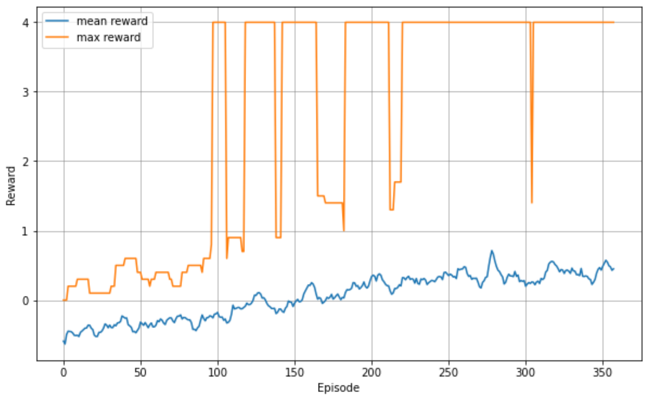

Figure 2(b): Reward per episode for PPO

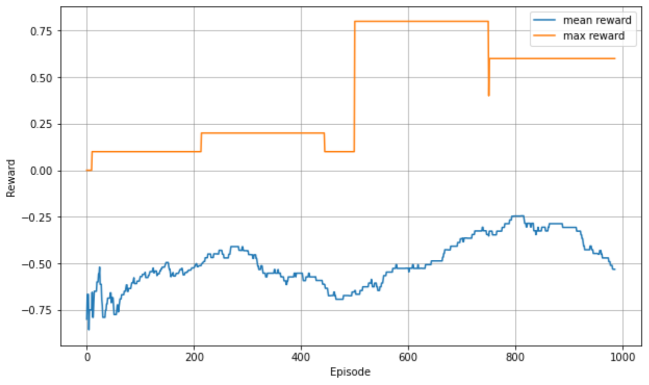

Figure 2(c): Reward per episode for mSAC

From the preceding plots, we can observe that the mean rewards for MADDPG and mSAC tend to be unstable with higher variance. The opposite effect was what I was hoping to find because of the learning improvements I had discussed in section II of this report. The maximum rewards are also not stable for my two algorithms. My results show that the algorithms could benefit from enhancement of hyperparameters. The PPO algorithm on the other hand show a consistent increase in mean rewards as learning progresses. Also, the maximum reward for PPO is relatively stable with few occasional drops. I also suspect that I could has trained my models for much longer barring any hardware bottlenecks, my selected algorithms would have at lease achieved comparable performance with PPO is not more.

Finally, I show the win rate for all three algorithms in Figure 3 below.

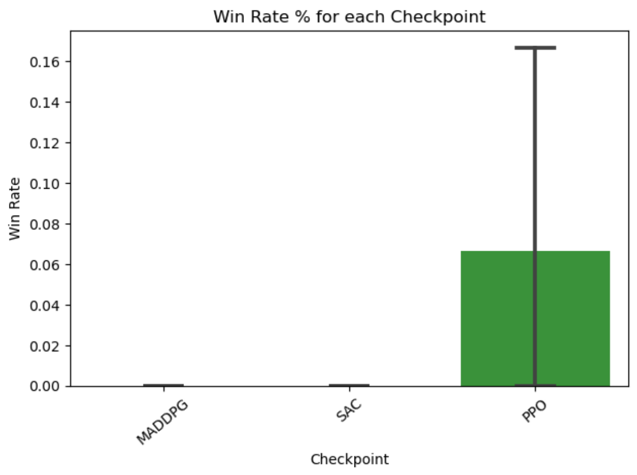

Figure 3: Win rates best checkpoints across all 3 algorithms.

The PPO is evidently the winning algorithm from the plots in Figure 3.

IV. CHALLENGES

This project has been riddled with challenges and problems right from the onset. First, I spent several days on trying to install the RLDM package on my Google cloud account without success. Training on my personal laptop was not possible due to low hardware capabilities. I finally had to run my experiment on a friend's customized desktop machine which had 1 GPU. Even with 16 GB of memory, I always ran out memory while training and the entire process shuts down. I will then have to manually change some of the hyperparameters and resume training. Eventually, I had to reduce the number of time steps and number of episodes to get a working result which was suboptimal (as evinced in the plots). Tuning hyperparameters manually was challenging and time-consuming. For four days my algorithms where consistently getting a negative loss until I realized that the learning rate hyperparameter was too high.

V. CONCLUSIONS

Thus far the report has presented a full account of the algorithms proposed to improve agents' learning capabilities in a competitive and cooperative environment like the Google Football. Although the results have not been as expected, due to certain limitations already discussed, I have gained a good theoretical understanding in multi-agent RL. Multi-agent RL concepts are promising but in practice the solution space is vast, so it remains constrained to few institutions who have a deep understanding of their inner workings like DeepMind. However, I will continue to do research in RL regarding its application in financial decision-making. On hindsight, there are a few things I wish I wish I could have tested which are beyond the scope of this project such as inverse reinforcement learning and imitation learning. I will be looking at them in my free time.

REFERENCES

[1] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (pp. 1861-1870). PMLR.

[2] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018, April). Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1).

[3] Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized POMDPs*. Springer.

[4] Pu, Y., Wang, S., Yang, R., Yao, X., & Li, B. (2021). Decomposed Soft Actor-Critic Method for Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2104.06655*.

[5] Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., & Mordatch, I. (2020). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems, 30.*