Multi-Agent Reinforcement Learning for Football

Akshaj Kumar Veldanda
New York University
akv275@nyu.edu

Kawshik Kannan
New York University
kk4161@nyu.edu

Abstract

Reinforcement Learning has been widely successful in single-agent settings and deterministic environment, but the real world is stochastic and often needs two ore more agents to cooperate or compete against each other to solve certain tasks. This requires policies that can handle a mixture of behaviour and dynamically decide which is more important depending on the situation. We focus our work in a setting that requires agents to compete against one another while cooperating with its own team to score high rewards. We believe through self play, the agents would evolve through an automatic curriculum induced during game-play to display interesting strategies in football. We explore different challenges like non-stationarity and credit-assignments and follow previous work to adopt single agent Proximal policy optimization for multi-agent settings.

1 Introduction

Reinforcement Learning has made great progress in many applications from games [22,23] to robotics [24]. Particularly , single agent Reinforcement learning has proven to be successful in playing the games of Go [25,26], real-time strategy games [21,27], robotic control [28,29] etc. While most algorithms focus on static environments, the real world is stochastic and often requires two or more agents to cooperate with or compete against each other to solve different tasks.

Agents can either learnt to cooperate with other agents or compete with them to solve tasks quicker and better. However its very difficult to find situations which are either purely cooperative or competitive in the real world. For example, car racing may look like purely competitive environments, however it still requires the cars to cooperate in order to avoid crashing during corners or while overtaking where one car should accommodate and slow down while the other car overtakes it. Similarly, take a situation when cars move through traffic, each driver has a different state of urgency which makes the environment stochastic in nature. While each car has to move through traffic obeying all the rules and maintaining a safe distance between each other through cooperation, the states of emergency induces a sense of competition among the cars in order to reach their destinations faster. Hence we see that most situations requires a mixture of cooperation and competition to perform tasks successfully. We focus on such an setting, namely football where agents have to explicitly learn to compete against the other team while cooperate with other agents in their own team.

It is interesting to explore which kind of policy is preferred at each moment of the game. Specifically we explore self-play which has been shown to be a useful training paradigm [14,23] for multi-agent settings. It provides an automatic-curriculum which helps the agents learn slowly but steadily while exploring the pitfalls in its own set of strategies and overcoming it to become better. After analysis of previous work, its restrictive to specify constraints inducing competition or cooperation in an algorithm explicitly. Instead, we use self play to observe emergent behaviour in multi-agent football. We implement Proximal Policy Optimization (PPO) [31], Soft Actor Critic [32] for learning stochastic policies and Deep Determinisitc policy gradients (DDPG) [33] for simple multi-agent settings for sanity checks. We found that the hyperparameters required for Proximal policy optimization were relatively easier to tune and hence we focus on this for our subsequent work.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Moreover we opt the asynchronous PPO version which uses multiple workers to collection experience which significantly speeds up training. Particularly we extend this work by adding the Decentralized-Execution, Centralized-Training strategy to handle the non-stationarity in multi-agent settings and call it CPPO. Following previous work [16], we further extend this work to include the counter factual baseline to calculate the advantage function in PPO and call it CPPO-CFB. Then we present the maximum, minimum and mean value of rewards for each setting with and without self-play and observe emergent behaviour in all cases.

2 Related Work

Multi-Agent Reinforcement Learning (MARL) is a long studied problem [1] and has recently re-emerged due to the advent of Deep learning and the advances in single-agent Reinforcement Learning (RL) techniques.

MARL is a long studied problem . Topics within MARL are diverse, ranging from learning communication between cooperative agents [2], [3] to algorithms for optimal play in competitive settings [4].

The simplest approach to problems in Multi-agent settings is independently learning agents [2]. This class of algorithms assume that each agent learns independently and assumes that the other agents are part of the environment. However each agent's policy changes during training, resulting in a non-stationary environment preventing the naive application of experience replay. This naive application does not perform well in practice [5].

The nature of interaction between agents can either be cooperative, competitive, or both, but many algorithms are designed only for a particular nature of interaction. Most studied are cooperative settings, with strategies such as optimistic and hysteretic Q function updates [6, 7, 28], which assume that the actions of other agents are made to improve collective reward. Another approach is to indirectly arrive at cooperation via sharing of policy parameters [9], but this requires homogeneous agent capabilities. These algorithms are generally not applicable in competitive or mixed settings.

Competitive games have long served as a testbed for learning. Early systems mastered Backgammon [10], Checkers [12], and Chess [11]. Self-play was shown to be a powerful algorithm for learning skills within high-dimensional continuous environments [13] and a method for automatically generating curricula [14].

Most relevant to this work are recent approaches that propose an actor-critic framework consisting of centralized training with decentralized execution, Lowe et al [15] and Foerster et al [16]. Lowe et al. (2017) investigate the challenges of multi-agent learning in mixed reward environments [17].
Similarly Foerster et al. (2018) introduce a centralized critic for cooperative settings with shared rewards. Their method incorporates a "counterfactual baseline" for calculating the advantage function which allows for complex multi-agent credit assignment.

Our contribution is closely connected to these methods and investigates the influence of each component towards the final rewards scored by the agent. We also explore self play for training as it uses a natural curriculum on the same episode instead of using different scenarios (easy to hard) during training for sequential learning and show the emergent strategies during the training process.

3 Method

3.1 Background

Multi-Agent RL addresses sequential decision making where there's more than one agent involved. In particular, both the evolution of the state of the system and the rewards received by each agent are influenced by the joint actions of all agents.

3.1.1 Markov Games

In this work we consider the multi-agent extension of Markov Decision Processes (MDPs) called Markov games (or Stochastic games). A Markov game is defined by a tuple $(\mathcal{N}, \mathcal{S}^i_{i\in\mathcal{N}}, \mathcal{A}^i_{i\in\mathcal{N}}, \mathcal{P}, \mathcal{R}^i_{i\in\mathcal{N}})$ where $\mathcal{N}$ = {1,...,N} for N > 1 agents, $\mathcal{S}^i$ denotes the state observed by

each agent i, $\mathcal{A}^i$ denotes the action of each agent i, let $\mathcal{A} = \mathcal{A}^1$ x $\mathcal{A}^2$ x ... x $\mathcal{A}^N$ denotes the joint action of all agents, then $\mathcal{P} : \mathcal{S}$ x $\mathcal{A} \rightarrow \mathcal{S}$ denotes the transition probability for a given state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ given a joint action $\mathcal{A}$; $\mathcal{R}^i : \mathcal{S}$ x $\mathcal{A}$ x $\mathcal{S} \rightarrow \mathbb{R}$ is the immediate reward received by agent i for a transition from (s,a) $\rightarrow$ s'; $\gamma \in [0, 1]$ is the discount factor. Each agent uses a policy π_i to get an action $\mathcal{A}_i \sim \pi_i$ at a time step t given a state s_t. As a consequence, the value function $\mathcal{V}^i : \mathcal{S} \rightarrow \mathbb{R}$ becomes a function of the joint policy $\pi(S, A) = \prod_{i=1}^{N} \pi(a_i|s_i)$,

$$\mathcal{V}^i_{\pi_i, \pi_{-i}}(s) := \mathbb{E}\left[\sum_{t>0} \gamma_t \mathcal{R}^i(s_t, a_t, s_{t+1}) \Big| a_t \sim \pi(.|s_t), s_0 = s\right]$$

where -i represents the indices of all agents in $\mathcal{N}$ except agent i. The solution to such a Markov Game based setup deviates from that of MDP since the optimal performance of each agent is controlled by the choices of all agents in the game.

We focus on the cooperative setting where each agent in the team optimizes for a team reward based on the goals scores. This setting is also referred to as multi-agent MDPs (MMDPs) and Markov Team games. The global optimum for such a setting is called a Nash Equilibrium which is defined for a markov game as,

$$\mathcal{V}^i_{\pi^*_i, \pi^*_{-i}}(s) \geq \mathcal{V}^i_{\pi_i, \pi^*_{-i}}(s), \exists \pi_i$$

where π^* denotes an equilibrium point from which none of the agents have any incentive to deviate. In other words π^*_i is the best response by an agent i given a state s_i.

Finally we also experiment with the mixed setting where there are two zero sum teams of cooperative agents competing against each other [19,20,21] .

3.1.2 Challenges

A standard paradigm in reinforcement learning involves a single agent interacting with a stationary environment, processing sensory inputs and selecting an action to maximize rewards. This is a powerful and broadly applicable paradigm, but, one key limitation of it is that it assumes the learning agent is the only agent in the world. So, a natural extension to this paradigm is the multi-agent paradigm in which each agent acknowledges that the environment with which it interacts consists of other agents who are also perceiving inputs and selecting actions to maximize reward. The main challenges involved in multi-agent reinforcement learning are:

- The non-stationarity between independent agents
- The exponential increase in state and action space
- Credit assignment problem

The real-world is built with multi-agent systems everywhere. So, if we want to built intelligent agents that can act in the real world and solve real world problems, we should tackle the above challenges in multi-agent systems. Some of the techniques proposed in the past are:

- Centralized training - Decentralized execution: An actor-critic method that uses separate centralized critics for each agent which take in all other agents' actions and observations as input, while training policies that are conditioned only on local information. This practice reduces the non-stationarity of multi-agent environments, as considering the actions of other agents to be part of the environment makes the state transition dynamics stable from the perspective of one agent. In practice, these ideas greatly stabilize learning, due to reduced variance in the value function estimates.
  - Centralized training: Each agent has access to local observations. These observations can contain several things including relative positions of other agents. During learning, all agents are guided by a centralized module or critic. Even though each agent only has local information and local policies to train, there is an entity overlooking the entire system of agents, advising them on how to update their policies. This reduces the effect of non-stationarity. All agents learn with the help of a module with global information.
  - Decentralized execution: During testing, the centralized module is removed, leaving only the agents, their policies, and local observations. This reduces the detriments of increasing state and action space because joint policies are never explicitly learned.

Instead, we hope that the central module has given enough information to guide local policy training such that it is optimal for the entire system once test time comes around.

- Counter factual baseline: Marginalizes a single agent's actions while keeping others fixed. This method uses an advantage function only encourages actions that directly influence an agent's rewards. This practice aides the credit assignment problem.

$$A_i(o, a) = Q_i^{\psi}(o, a) - b(o, a_{\setminus i}), \tag{1}$$

where

$$b(o, a_{\setminus i}) = \mathbb{E}_{a_i \sim \pi_i(o_i)}[Q_i^{\psi}(o, (a_i, a_{\setminus i}))] = \Sigma_{a_i^{'} \in A_i} \pi(a_i^{'}|o_i) Q_i(o, (a_i^{'}, a_{\setminus i}))) \tag{2}$$

3.2 Algorithms

Actor Critic methods represent the policy function independent of the value function. The policy network is called as *actor*, which is used to select an action. The value function is known as *critic*, because it criticizes the actions selected by the actor. In most cases, the critic is a state value function and after each action selection, the critic evaluates the new state by determining if things have gone better or worse than expected. That evaluation is the TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{3}$$

This error can be used to evaluate the action just selected, the action a_t taken in state s_t. If the TD error is positive, it suggests that the tendency to select should be strengthened for the future, whereas if the TD error is negative, it suggests the tendency should be weakened.

Policy optimization methods Policy gradient methods compute the gradients of policy network w.r.t policy parameters and then use gradient ascent to update the policy network.

- **Trust Region Policy Optimization(TRPO)** Instead of directly using policy gradients, there is a category of policy iteration algorithms that guarantee monotonic improvement . Trust Region Policy Optimization(TRPO) is one of the most widely used baseline for policy optimization. Using TRPO, it has been proved that if the new policy is close to the old policy in terms of the KL divergence, there is a lower bound of the long-term rewards of the new policy.
- **PPO** is a modified version of TRPO which mainly uses clipped probability ratio of old policy to new policy in the objective function. The modified objective avoids excessively large policy updates.

$$l^{PPO}(\theta; s, a) = min\{\frac{\pi_\theta(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s, a), \texttt{clip}(\frac{\pi_\theta(a|s)}{\pi_{old}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{old}}(s, a)\} \tag{4}$$

  PPO is empirically more sample efficient than TRPO, and is one of the frequently used baselines in RL tasks.

DDPG is a policy gradient method for deterministic policy. In DDPG, the actor directly maps states to actions instead of outputting a probability distribution on discrete action space. DDPG learning algorithm can be broken down into the following parts:

- **Replay Buffer:** For optimization tasks in ML, the training data should be independent. But, the data is not independent when we optimize a sequential decision in an on-policy way. So, to overcome this issue, we store them in a replay buffer and take random batches for training.
- **Actor Network Update:** For the policy function, our objective is to maximize the expected return. The policy function is differentiable, so we take the derivative of the loss function w.r.t policy parameters and update the actor network.

$$\Delta_{\theta^\mu} J(\theta) = \Delta_a Q(s, a) \Delta_{\theta^\mu} \mu(s|\theta^\mu) \tag{5}$$

- **Critic Network Update:** The value network is updated similar to Q-learning. The loss function to optimize is

$$\texttt{Loss} = \frac{1}{N}\Sigma_i(y_i - Q(s_i, a_i|\theta^Q))^2 \tag{6}$$

where

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}, \theta^{\mu^{'}}|\theta^{Q^{'}})) \tag{7}$$

- **Exploration:** In DDPG, the Ornstein-Uhlenbeck process is used to add noise to the action output to encourage the agent for having exploratory behaviour.

Soft actor critic (SAC): In SAC, we seek to maximize the entropy of the policy besides maximizing the lifetime rewards. We want to maximize the entropy in our policy to encourage exploration. In other words, the policy learns to assign equal probabilities to actions that have same or nearly equal Q-values. SAC makes use of three networks

- **State value function:** Across all the states in the replay buffer, we seek to minimize the mean squared error between the prediction of our value network and the expected prediction of the Q function plus the entropy of the policy function.
- **Soft Q-function:** For all (state, action) pairs in the replay buffer, we want to minimize the mean squared error between the prediction of our Q function and the immediate (one time-step) reward plus the discounted expected Value of the next state.
- **Policy function:** We solve the following objective function which tries to make the distribution of our policy function look similar to distribution of the exponentiation of our normalized (with Z) Q Function in terms of KL divergence .

$$J_\pi = \mathbb{E}_{s_t \sim \mathcal{D}}[D_{KL}(\pi_\phi(\cdot|s_t))||\frac{exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)}] \tag{8}$$

CPPO We extend the PPO algorithm by including a centralized training strategy, decentralized execution strategy to deal with the non-stationarity problem. We call this algorithm as CPPO (or centralized-PPO)Although independent learner do perform well empirically, they do not have convergence guarantees in many cases. Hence in order to be theoritically complete, we unify the critic function by taking in the global state and actions of agents as input to produce the q value for that state-action pair.

$$Q_i^\pi(s_i, a_i) \Rightarrow Q_i^\pi(S, A)$$

where S is the global state and A is the set of all actions (A1,A2,A3.....A_n). This strategy considers the other agents when it calculates the q value for a particular agent. This allows each agent to learn together and act in a way that is beneficial for every body involved. As stated before, during execution, each agent executes their policy independently but implicitly in favour of high team rewards.

CPPO-CFB We further extend the CPPO algorithm by using the counter factual baseline for calculating the advantage function. The rewards in the google footbal environment depend a lot on the spare scoring reward. However its difficult to understand which agent's actions contributed to success. If we don't do this, the algorithm may fail to penalize bad actions due to good final rewards or penalize good actions when the final rewards are low. This leads to instability during training and creates problems for convergence. In order to overcome this, previous work [16] have tried to marginalize the agent's action from the centralized critic to find the advantage of a particular action over others for a particular agent. Particularly, in CPPO, we replace the generalized advantage function by the counterfactual baseline which tries to address this problem.

$$A_i(o, a) = Q_i(o, a) - \mathbb{E}_{a_i \sim \pi_i(o_i)}[Q_i(o, (a_i, a_{\setminus i}]$$
$$A_i(o, a) = Q_i(o, a) - \sum_{a_i \in A} \pi(a_i|o_i) Q_i(o, (a_i, a_{\setminus i}))$$

where A denotes the set of all actions, A_i the advantage function for agent i, o is the observation, a is the action, a_i is the action of agent i, $a_{\setminus i}$ is the action of all agents except agent i. This is very simple to implement for discrete actions and does not need multiple forward passes. This can be done in a single forward pass such that the output of critic function now produces the output of the q value for each action value.

(a) Real video frame (b) Super Minimap frame

Figure 1: Example of observation

(a) Corner (b) Kickoff (c) Yellow Card

Figure 2: Examples of various scenarios in google research football environment

4 Experiments and Results

4.1 Setup

4.1.1 Environment

We use the Google Research Football environment, a highly optimized stochastic game engine that simulates the game of football in all our experiments. It provided state based and image based observations for use and multiple scenarios (shown in figure 2a, 2b, 2c) like penalty shoot, corner play, 3 vs 1 keeper, 5 vs 5 etc with flexibility to define custom scenarios as well. We opted to use state based Reinforcement Learning owing to the complexity of the task. The authors [18] introduce a set of wrappers (Super minimap, float 115 etc) for the raw state based observations. We use the Super minimap state wrapper because its very simple and contains inherent spatial information covering the entire play area. It has a resolution of 72 x 96 with four channels representing the players on the left team, players on the right team, the position of the ball and finally the position of the active player in case of multi agent settings. We stack four observations together along the channels to include a history of observations. Although the ball can move in the z direction, the super minimap does not contain this part of the observation as everything is represented in 2D. An example observation is shown in figure 1a and 1b. The red dots represent players from one team, the green dots represent the player from the other team while the blue dot represents the ball.

Specifically, we experiment with two scenarios, "academy goal close" where a player who is pretty close to the goal post has to learn to kick the ball into the goal post as shown in figure 3a. We use this environment to check the correctness of our algorithm implementations. The other main scenario we deal with is the 3 vs 3 (shown in figure 3b) setting, a custom scenario based on the predefined 5 vs 5 scenario in the environment. There are 3 players in each team where 2 players excluding the goal keeper are controlled by agents following a policy π and the players switch sides every 2 episodes which allows the policy to be general enough to control any player. The goal keepers are inbuilt bots.

(a) empty goal close

(b) 3 vs 3

Figure 3: Scenarios used in this study

Top	Bottom	Left	Right
Top-Left	Top-Right	Bottom-Left	Bottom-Right
Short Pass	High Pass	Long Pass	Shot
Do-Nothing	Sliding	Dribble	Stop-Dribble
Sprint	Stop-Moving	Stop-Sprint	—

Figure 4: Set of all actions for each agent

The duration of each episode is 400 steps with the complete gameplay including corners, goal kicks, offside etc.

Each agent has the ability to take 19 different actions making them very similar to real world scenarios as shown in Figure 4].

There are two different types of rewards available, scoring (+1 for scoring a goal, -1 if conceding a goal) and checkpoints (starts at 0.1 and goes upto 1.0 as the player goes towards the goal keeper). The checkpoint rewards are used for exploration as it tends to be difficult for the agents to learn with sparse rewards. The checkpoint rewards are automatically awarded if the agent scores a goal before it reaches the end. This allows different individual rewards for each agent during gameplay while the final rewards remain constant for every episode.

4.1.2 Model Architecture

We started out with a simple MLP architecture on the float 115 observation containing 115 float values describing a bunch of information about the players, ball and other things that are part of the gameplay. However this did not work out for us. Instead we switched to the Super mini map style observations as described above and used a CNN based backbone with 3 convolutional layers consisting of 32, 64 and 64 filters with sizes 8,4 and 3 and strides 4,2,1 respectively. This backbone is shared by both the actor and the critic for initial processing. The Actor network then processes this information with 2 more fully connected layers (256, 19) to finally output one of the 19 discrete actions. In case of an independent learner, the critic takes in the local observation denoting the active player and the action of that agent to give a Q value for that agent. Whereas in case of centralized training, the actions of other agents are considered together with the current agent's actions with a global state (which we consider it as the first 3 channels of the local observation - which is constant for every agent and excludes the information of the active player. To allow gradients to flow back from the critic network to the actor network through the discrete actions predicted, we use the gumbel

softmax relaxation technique to make it differentiable. Given a discrete action a_t, the i.i.d standard gumbel distribution g_t is added to the action distribution to simulate the sampling operation. Then a softmax is applied with a temperature T to simulate the argmax operator. We used a temperature of T=1 to not introduce any noise at this stage. However higher temperature can allow more exploration during learning. It is described in equation (9) where τ denotes the inverse temperature.

$$a'_t = \text{softmax}(\tau * (a_t + g_t)) \tag{9}$$

$$\text{where,} \tag{10}$$

$$g_t = -\log(\log \text{U}), \tag{11}$$

$$\text{U} \sim \text{Uniform}(0, 1) \tag{12}$$

The critic network shares a similar CNN backbone except that the input has 3 channels instead of 4 because its the global state and uses embeddings to encode the discrete actions to a higher dimension and encodes the output from the CNN and the actions together to produce a Q value for that state-action pair.

4.1.3 Hyperparameters

We focused most of our experiments on PPO and its variants (CPPO and CPPO-CFB). We implemented the asynchronous version of PPO where multiple workers collect experience and then they are pooled together and used for updating the agents. We tie the weights of the actor networks of each agent and the critic networks of each agent to setup a simpler pipeline in terms of the number of learnable parameters involved which can allow early convergence of the learning process. For PPO, the models were trained for 4 epochs on the collected experience tuples. The target networks were updated after this at each step. This setup was not stable and required clipping the gradient values at 0.5 and 10.0 for the actor and the critic respectively. The PPO algorithm uses a clipped objective with the clip value being 0.27. It uses a generalized advantage function with entropy component and the value loss component weighted by the coefficients 0.01 and 0.5 respectively. A batch size of 256 was used for each update in each epoch. A learning rate of 1e-4 turned out to be optimal for learning. Each worker runs for 128 steps across multiple episodes with a total set of 128 * n tuples collected for n workers after interaction. We used a maximum of 16 workers owing to the resource constraints. The agents showed a positive trend in learning after 3 days of GPU time on a single p40 GPU.

4.2 Results and Analysis

4.2.1 Single-agent RL

As a baseline, we trained an agent using PPO algorithm to learn the following tasks:

- **Empty Goal Close:** We observe that the agent learns to score the goal on every test episode after training on 250,000 frames. In this case, the reward function is scoring.
- **3 vs 1 with keeper:** We observe that the agent learns to score a goal with an average goal difference of 0.57 over 100 episodes after 5M training steps. In this case, the reward function is scoring.

4.2.2 Multi-agent RL

We conduct a multi-agent RL study on google football 3 vs 3 scenario using scoring and checkpoint rewards. In this setting, the agents plays against built-in AI and the performance is evaluated after training for 25M steps. As a starting point, we extended the PPO to multi-agent RL 3 vs 3 scenario and notice that the average goal difference is -2.57. But, with centralized training using PPO algorithm, we note that the average goal difference improves to 0.87. This improvement is mainly due to reduced non-stationarity between independent agents. Finally, we experimented the 3 vs 3 scenario using centralized training with counter factual baseline to address the counter factual baseline. The average goal difference with CPPO-CFB is 1.28. Although a counterfactual baseline should help improve the situation over the CPPO counterpart, we did not observe this. We believe that this is because the Table 1 shows the minimum, maximum and mean rewards for each of the three algorithmic setting.

Method	Min Reward	Max Reward	Mean Reward over 10 episodes
MA-PPO	-12	0	- 2.57
MA-CPPO	-2	4	0.87
MA-CPPO-CFB	-1	6	1.28
Self-Play-CPPO	-3	4	1.12

Table 1: Performance of multi-agent RL trained against built-in AI on 3 vs 3 scenario evaluated with different algorithms

4.2.3 Emergent Strategies

We implement multi-agent RL with self play using the CPPO algorithm to observe emergent strategies. Here, the agents in the both the teams are controlled by the training algorithm and learn to compete against each other. We notice that the average goal difference is 1.12 with self play compared to 0.87 without self play. One interesting behaviour we found was scoring self goals. We believe this is because of our implementation of self play where the actions of the opposition team were also used in the critic function. Future work could address this issue by opting for more complex critic settings where they have different ways to include the behaviour of oppostion teams so that it doesn't influence the actions of the agent. Otherwise, our agents learn to pass well and guard the ball well by taking it away from an opponent nearing them or blocking their opponents by turning away from them.

4.2.4 Training challenges

We observed that the multi-agent RL takes too many episodes (3 days of training time) in the beginning before learning to get non positive rewards. Moreover, if the critic weights are not tied, then training with multiple networks for each agent takes a lot more time to converge. This prevented us from exploring this prospect further. Different policy network weights for each agent may allow very different policy behaviours for each agent which may be beneficial for complex team behaviours. We also faced issues with computational resources; we wanted to do more experiments to study hyper-parameter sensitivity, but each experiment took 3 to 4 days of time before showing some positive behaviour. This work has helped us understand the complexity of training multi-agent training algorithms.

5 Conclusion

In this work, we focus on a RL setting that requires agents to compete against one another while cooperating with its own team to score high rewards. We adopt self play to achieve this and address different challenges like non-stationarity and credit assignment problems that arise in multi-agent reinforcement learning. We observe that the average goal difference of 1.12 with self play compared to 0.87 without self play.

Future work could train base policies in simpler environments and apply transfer learning or use them as policy primitives to handle complex multi-agent settings. Hierarchical Reinforcement Learning is one other venue which could be very beneficial where the each hierarchy can either provide high level actions to all agents of the same team or high level actions to defenders, attackers, midfielders etc separately and the base policies could be implemented as policy primitives to reduce complexity during training. Another important prospect is to create a policy function which takes into account other agent's observations indirectly perhaps using self-attention which could be beneficial in environment with sparse team rewards. We believe this is an important prospect which has a lot of potential and could open new venues for research in the future.

References

[1] Bus¸oniu, L., Babuska, R., and De Schutter, B. Multi-agent ˘ reinforcement learning: An overview. In Innovations in multi-agent systems and applications-1, pp. 183–221. Springer, 2010.

[2] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In Proceedings of the tenth international conference on machine learning, pages 330–337, 1993.

[3] Fischer, F., Rovatsos, M., and Weiss, G. Hierarchical reinforcement learning in communication-mediated multiagent coordination. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3, pp. 1334–1335. IEEE Computer Society, 2004.

[4] Littman, M. L. Markov games as a framework for multiagent reinforcement learning. In Machine Learning Proceedings 1994, pp. 157–163. Elsevier, 1994.

[5] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. The Knowledge Engineering Review, 27(01):1–31, 2012.

[6] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In In Proceedings of the Seventeenth International Conference on Machine Learning, pages 535–542. Morgan Kaufmann, 2000.

[7] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, pages 64–69. IEEE, 2007.

[8] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. CoRR, abs/1703.06182, 2017.

[9] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. 2017

[10] Tesauro, G. TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural computation 6, 215–219 (1994).

[11] Campbell, M., Hoane Jr., A. J. & Hsu, F.-h. Deep Blue. Artif. Intell. 134, 57–83. issn: 0004-3702 (Jan. 2002).

[12] Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P. & Szafron, D. A world championship caliber checkers program. Artificial Intelligence 53, 273–289. issn: 0004-3702 (1992).

[13] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I. & Mordatch, I. Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748 (2017).

[14] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A. & Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. arXiv preprint arXiv:1703.05407 (2017).

[15] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In Advances in Neural Information Processing Systems, pp. 6382–6393, 2017.

[16] Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In AAAI Conference on Artificial Intelligence, 2018

[17] Bus¸oniu, L., Babuska, R., and De Schutter, B. Multi-agent ˘ reinforcement learning: An overview. In Innovations in multi-agent systems and applications-1, pp. 183–221. Springer, 2010.

[18] Kurach, Karol, et al. "Google research football: A novel reinforcement learning environment." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 04. 2020.

[19] Lagoudakis, M. G. and Parr, R. (2003). Learning in zero-sum team Markov games using factored value functions. In Advances in Neural Information Processing Systems.

[20] Zhang, K., Yang, Z., Liu, H., Zhang, T. and Bas¸ar, T. (2018b). Finite-sample analyses for fully decentralized multi-agent reinforcement learning. arXiv preprint arXiv:1812.02783.

[21] OpenAI (2018). Openai five. https://blog.openai.com/openai-five/.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484 – 489, 2016.

[24] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. arXiv preprint arXiv:1504.00702, 2015.

[25] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529 484–489.

[26] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017). Mastering the game of Go without human knowledge. Nature, 550 354.

[27] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Wu, Y., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D. and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.

[28] Kober, J., Bagnell, J. A. and Peters, J. (2013). Reinforcement learning in robotics: A survey. International Journal of Robotics Research, 32 1238–1274.

[29] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In International Conference on Learning Representations.

[30] Schulman, John, et al. "Proximal policy optimization algorithms. arXiv 2017." arXiv preprint arXiv:1707.06347.

[31] Haarnoja, Tuomas, et al. "Soft actor-critic algorithms and applications." arXiv preprint arXiv:1812.05905 (2018).

[32] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).

[33] Stooke, Adam, and Pieter Abbeel. "Accelerated methods for deep reinforcement learning." arXiv preprint arXiv:1803.02811 (2018).