Master Soccer with Multi-Agent Reinforcement Learning and Self-Play

Wei-Cheng Tseng (A072045) [1] **Po-Han Chi (A072032)** [1] **Jaw-Yuan Chang (A072126)** [1] **Jing-Yo Yen (A072129)** [1]
Hsuan-Po Liu (0760201) [2]

Abstract

Motivated by the future scenarios that applications with multiple agents are currently discussed in the reinforcement learning tasks, we implement four different methods to compare the behavior discrepancy in an environment with multiple agents constructed by Unity. In this project, single-agent methods such as Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) are implemented. Moreover, in order to compare single-agent methods to methods proposed specifically for tasks with multiple agents, we implement two multi-agent methods such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) and Actor-Attention-Critic for Multi-Agent Reinforcement Learning (MAAC). Experimental results show the win rate of each method against each other. Under such a result, a surprising conclusion can be obtained that not every multi-agent method outperforms single-agent methods under environments with multiple agents as the researches of multi-agent methods claimed. Moreover, since reward setting has always been an important issue in reinforcement learning, we also adjust the reward setting in this task in order to observe the variation difference for the performance. Our source code is released at [this repo].

1. Introduction

Recently, reinforcement learning (RL) achieves exciting progress in several domains such as robotics, telecommunications, and game playing. With an arising complexity in these domains, these tasks have become much more difficult than before to solved by preprogrammed agent behaviors. Although the agents in a system can assign with behaviors designed in advance, they often need to learn new behaviors online, such that the performance of the system can gradually improves. Therefore, in order to avoid this issue, agents must employ new strategies to discover solutions online on their own.

Current research focus on single-agent reinforcement learning approaches to deal with issues as mentioned above. The simplest approach is to train each agent independently by maximizing their individual reward, while regarding other agents as part of the environment. This approach regard agent to be trained in a stationary environment, which is a fixed environment without changing over time passes. However, in most tasks, there will not be only one single agent to interact with a stationary environment, nonetheless, in the real world, most cases tend to interact with multiple agents which makes the environment non-stationary (Ryan Lowe, 2017), (Shariq Iqbal, 2019). Therefore, multi-agent methods are recently proposed to further improve the performance. They attempt to learn new behaviors online by considering the observations and actions of other agents simultaneously, which will theoretically be more stable than the single-agent methods.

In this project, we examined several single-agent and multi-agent methods with competing goals in a 3D world through a simulated soccer game built by Unity, which consists 4 agents which are equally separated into two teams (i.e. two players for each team) in the playground. The main idea is to demonstrate different behavior model among several existing single-agent and multi-agent methods. The intuition behind employing multi-agent methods comes from the fact that, in a real world soccer game, it is beneficial for teams that players understand how to play the game in order to achieve a higher win rate by cooperating with teammates. For example, while attacking the opponents, goalies can participate by passing the ball to strikers under some strategies learned by the multi-agent methods.

In more detail, the agents would need to learn highly developed skills in order to succeed in the competitive environment. We train the agents using single-agent methods of recent policy gradient algorithms, Proximal Policy Optimization (PPO) (J. Schulman & Klimov., 2017) and Advantage Actor-Critic (A2C) (Volodymyr Mnih, 2016). Moreover, to compare whether the single-agent methods is not powerful enough or not as several proposed multi-agent methods mentioned, we also implement two multi-agent methods such as Multi-Agent Deep Deterministic Policy Gradient (MADDPG) (Ryan Lowe, 2017) and Actor-Attention-Critic for Multi-Agent Reinforcement Learning (MAAC) (Shariq Iqbal, 2019) in order to compare the performance for single-agent and multi-agent methods by using

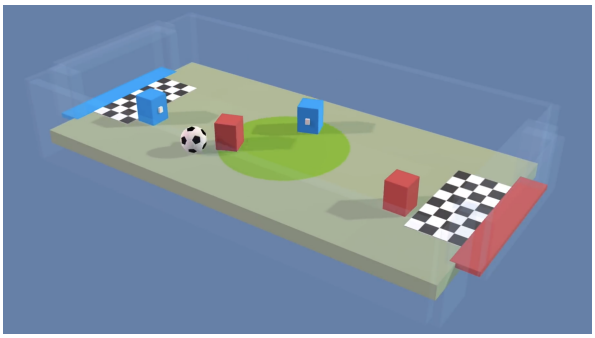

Figure 1. Unity SoccerTwo environment. The multi-agent environment we used in the experiment. This environment is both competitive and cooperative, which is suitable to evaluate whether multi-agent reinforcement learning method outperform single-agent reinforcement learning method.

our environment with multiple agents.

By employing simple exploration curriculum which aid the learning progress improve policies in the environment, we find out that agents learn a high level of intelligence in order to achieve their goals. Specifically, the agents learned a wide variety of skills and behaviors that include kicking, passing the ball and defending the goal. However, in the experimental studies, we discover an astonishing result that although recent results keep claiming their proposed multi-agent methods outperform single-agent methods, in our result, an absolute opposite result is shown. Therefore, we give a detailed discussion for explaining this phenomena. Moreover, since reward setting has always been an important issue in reinforcement learning, we also adjust the reward setting in this task in order to observe the variation difference for the performance.

The remainder of this report is organized as follows. In Section 2, we discuss the background of our project, followed by a detailed description of our used methods in Section 3. Experimental studies are provided in Section 4 and Section 5 concludes the project report.

2. Background

In this section, the necessary background on single-agent and multi-agent RL is introduced. First, several single-agent methods is defined in detail, which are basis for the implemented methods in this project in the next section. Then, the discussion for the necessary existence of multi-agent methods is defined.

2.1. Policy Gradient

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm.

$$\pi_\theta(\tau) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|a_t,s_t), \quad (1)$$

where π_θ denotes as policy network, a_t is an action while time step t and s_t denotes as state of time step. From equation(1), the objective function $J(\theta)$ is given as:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\sum_{t=1}^{T} r(s_t,a_t)] = E_{\tau\sim\pi_\theta(\tau)}[R(\tau)], \quad (2)$$

where $r(s_t,a_t)$ is a reward function. Since the reward function is a random variable, the objective function calculates the expected value. Also, the gradient estimator has the form:

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta log\pi_\theta(\tau)R(\tau)]. \quad (3)$$

2.2. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) (Schulman & Abbeel, 2015) introduced the concept of importance sampling, which allows TRPO to training off-policy, that is, the behavior policy can operate by sampling all actions, whereas the estimation of policy can be deterministic.

2.3. Q-learning

Q-learning (Mnih & Riedmiller, 2013) is an algorithm that seeks to find the best action to take given the current state. More specifically, Given a pair of actor and critic it evaluates how good the actor is by state-action value function:

$$\pi'(s) = \operatorname*{argmax}_a Q^\pi(s,a), \quad (4)$$

and the reward function:

$$\Delta\bar{R} \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(Q^\pi - V^\pi(s_t^n))\nabla log p_\theta(a_t^n|s_t^n), \quad (5)$$

where Q^π and V^π are both neural network. Note that Q^π is the expected value of V^π, hence V^π can be viewed as a baseline. Given actor $\pi(s)$, actions a and states s, the target is to find another $\pi'(s)$ which outperform current $\pi(s)$. In Atari, the performance of Q-Learning has overwhelming human![Video]

2.4. Multi-Agent Reinforcement Learning Approach

In Reinforcement Learning, an agent interacts with the environment by sequentially taking actions, observing consequences, and altering its behaviors to maximize the reward. However, in a cooperative and competitive environment, we recommend the multi-agent system. Multi-agent reinforcement learning hypothesis that the agent learns how to interact with the environment while considering the actions takes by others.

3. Methods

In this section, several single-agent and multi-agent methods are introduced. First, we represent PPO with clipped objective based on TRPO as illustrated in the previous section. Next, a well known method Advantage Actor-Critic (A2C) based on policy gradient and Q-learning in the previous section. Finally, followed by two multi-agent methods, MADDPG and MAAC, these methods represent the strategies used to deal with environments consist of multiple agents.

3.1. Proximal Policy Optimization (PPO) with Clipped Objective

On the basis of TRPO, Proximal Policy Optimization (PPO) (J. Schulman & Klimov., 2017) uses KL divergence as the constraint of two objective functions. It acts as a regularization term while updating the objective function. It performs comparably or better than state-of-art approaches while being much simpler to carry out and tune. Furthermore, it has achieved large success in learning to walk, run, turn, use its momentum to recover from minor hits, and how to stand up from the ground when it is knocked over.[Video]

PPO with clipped objective defines a clipped surrogate objective. The main objective function is as follows:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(A_t(\theta, 1-\epsilon, 1+\epsilon))], \tag{6}$$

where ϵ is a hyperparameter. $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$. This objective function acts as a constraint which limits the updated value of the neural network.

In this project, the actor is constructed by three fully connected layers with two $\tanh$ activation function after the first and second layer while makes the $softmax$ activation function as the output layer. The value state function is also constructed pretty similar to the actor but unplug the $softmax$ layer and change the output dimension into 1.

3.2. Advantage Actor-Critic (A2C)

Advantage Actor-Critic (Volodymyr Mnih, 2016) is a well known algorithm based on policy gradient and Q-learning. The reward function has the form:

$$X_t^n(s) = r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n), \tag{7}$$

$$\Delta\bar{R} \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(X_t^n(s))\nabla log p_\theta(a_t^n|s_t^n). \tag{8}$$

Advantage Actor-Critic substitutes Q^π as $r_t^n + V^\pi(s_{t+1}^n)$, hence there is no reason to tolerance the error caused by two neural network.

Algorithm 1 PPO with Clipped Objective

Inputs: initial policy parameters θ_0, clipped threshold ϵ
for k = 0,1,2,... **do**
Sample set of partial trajectories D_k on policy $\pi_k = \pi(\theta_k)$
Estimate $\hat{A}_t^{\pi_k}$ using any advantage function
Compute policy update

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}} L_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$clip = clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k}$$

$$L_{\theta_k}^{CLIP} = E_{\tau\sim\pi_k}[\sum_{t=0}^{T}[min(r_t(\theta)\hat{A}_t^{\pi_k}, clip)]]$$

end for

In this project, the actor and critic share three fully connected layers and two $\tanh$ activation function. Then, the actor contains an additional fully connected layer and a $softmax$ activation function to output the action probability distribution. The critic (value state function) contains an additional fully connected layer to output a state value.

3.3. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG)

Multi-agent actor-critic for mixed cooperative-competitive environments (MADDPG) (Ryan Lowe, 2017) is an actor-critic algorithm to multi-agent problems with "centralized training and distributed testing" constraint. The trick is that the critic train in a centralized way, that is, it has access to all observations and policies because it is not used during testing. The equation of policy gradient, which is based on (David Silver, 2014) following as:

$$\nabla_{\theta i}J(\theta_i) = E_{s\sim\rho^\pi, a_i\sim\pi_i}[\nabla_{\theta_i} log\pi_i(a_i|o_i)Q_i^\pi(o,a)], \tag{9}$$

where o is the set of observations from agents , a is the set of actions taken by agents and i denotes as each agent. Replay buffer D contains the tuple $(x, x', a_1, ..., a_n, r_1, ...r_n)$, recording the experience of each agent. The centralized action-value function Q_i^μ is updated as:

$$y = r_i + \gamma Q_i^{\mu'}(x', a_1', a_2', ...a_n')|_{a_j'=\mu_{\theta_i'(o_i)}'}, \tag{10}$$

$$L(\theta_i) = E_{x,a,r,x'}[(y^j - Q_i^\mu(x^j, a_1^j, a_2^j, ...a_n^j))^2], \tag{11}$$

where $\mu' = \{\mu_{\theta_1'}, ..., \mu_{\theta_N'}\}$ is the set of target policies with delay parameters θ_i'.

Algorithm 2 Advantage Actor-Critic

Inputs: Initial parameters s, θ, w, and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$
for i = 1,2,...,T **do**
Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
Sample the next action $a' \sim \pi_\theta(a', s')$
Update the policy parameters:

$$\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta log \pi_\theta(a|s)$$

compute the (TD or MC) function for action at time t as: δ_t
Update Q function

$$w \leftarrow w + \alpha_w \delta_w \nabla Q_w(s,a)$$

$a \leftarrow a'$
$s \leftarrow s'$
end for

Algorithm 3 Multi-Agent Deep Deterministic Policy Gradient for N agents

Inputs: initial policy parameters θ_0
for episode = 1,2,...,M **do**
Initialize a random process N for action exploration
Receive initial state x
for t = 1 to max-episode-length **do**
For each agent i, select $a_i = \mu_{\theta_i(s_i)} + N_t$ with respect to the current policy and exploration
Execute action $a = (a_1, a_2, ...a_n)$ and observe reward r and new state x'
Store (x, a, r, x') in replay buffer D
$x \leftarrow x'$
for agent i = 1,2,...,N **do**
Sample a random minibatch of S samples (x^j, a^j, r^j, x'^j) from D
Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x^j, a_1^j, a_2^j, ...a_n^j)|_{a_i = \mu_{\theta_i(o_i^j)}}$
Update critic by minimizing the loss

$$L(\theta_i) = \frac{1}{s} \sum_j (y^j - Q_i^\mu(x^j, a_1^j, a_2^j, ...a_n^j))^2$$

Update actor using the sampled policy gradient:

$$X_i^\mu = Q_i^\mu(x^j, a_1^j, a_2^j, ...a_n^j)|_{a_i = \mu_{\theta_i(o_i^j)}}$$

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j) \nabla_{a_i} X_i^\mu$$

end for
Update target network parameters for each agent i:

$$\theta_i' \leftarrow \tau \theta_i + (1-\tau)\theta_i'$$

end for
end for

3.3.1. Gumbel Softmax

In the MADDPG paper (Ryan Lowe, 2017), action space for agents is a continuous two-dimensional vector space. However, in our project, the action space is a discrete vector space, which means that the way updating the policy model should be changed. If we use the same way to update the policy model, it will not have any gradient back to itself. Therefore, we leverage Gumbel Softmax method (Eq. 12), which makes the discrete action space differentiable, thus, the policy model can use the same (Eq. 9) to normally update the parameter without leveraging extra entropy method.

$$y_i = \frac{\exp(\frac{\log(\pi_i)+g_i}{\tau})}{\sum_j \exp(\frac{\log(\pi_i)+g_i}{\tau})} \tag{12}$$

In this project, the actor is constructed by three fully connected layers with two activation function after the first and second layer while makes the Gumbel softmax activation function as the output layer. The critic is also constructed pretty similar to the actor but unplug the Gumbel softmax layer and change the output dimension into 1.

3.4. Actor-Attention-Critic for Multi-Agent Reinforcement Learning (MAAC)

Actor-attention-critic for multi-agent reinforcement learning (MAAC) (Shariq Iqbal, 2019) ought to find an approach among numerous multi-agents. It hypothesis that not all the actions were taken by each agent is equally important at the same state. Hence, the main idea behind MAAC approach is to learn the critic for each agent by selectively paying attention to information from other agent. MAAC has used multiple attention head (Ashish Vaswani, 2017) to calculate the attention weight α_j. α_j compares the embedding e_j with $e_i = g_i(o_i, a_i)$, where g_i is a one-layer multi-layer perceptron (MLP) embedding function, o_i is the observation and a_i is the action for all agent index from $i \in \{1, ..., N\}$, using a bilinear mapping and passing the similarity value between these two embeddings into a softmax:

$$\alpha_j \propto exp(e_j^T W_k^T W_q e_i), \tag{13}$$

where W_q transforms e_i into a query and W_k transforms e_j into a key. The matching is then scaled by the dimensionality of these two matrices to prevent vanishing gradients (Ashish Vaswani, 2017). Then the contribution from other agents, x_i, is a weighted sum of each agents value:

$$x_i = \sum_{i \neq j} \alpha_j v_j = \sum_{i \neq j} \alpha_j h(V_{g_j}(o_j, a_j)), \tag{14}$$

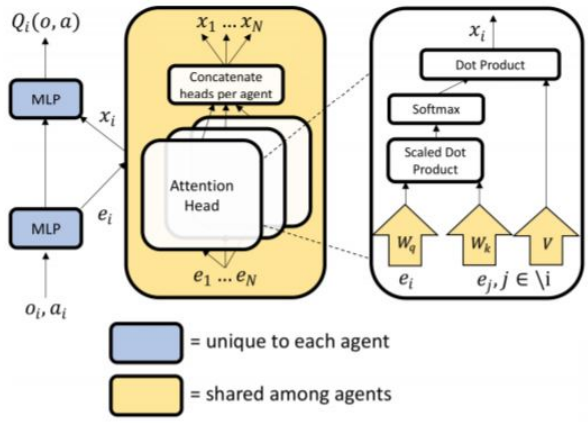

Figure 2. Calculating $Q_i^{\psi}(o,a)$ with attention for agent i. Each agent encodes its observations and actions, sends it to the central attention mechanism, and receives a weighted sum of other agents encodings (each tranformed by the matrix V)

where the set j represent the set of all agent except i as $\backslash i$. Then, the Q-value function, $Q_i^{\psi}(o,a)$, is a function of agent i's observation and action, as well as other agents' contributions:

$$Q_i^{\psi}(o,a) = f_i(g_i(o_i,a_i),x_i), \tag{15}$$

where f_i is a two-layer MLP. Also, in order to encourage exploration and avoid converging to non-optimal deterministic policies, Soft Actor-Critic (Tuomas Haarnoja, 2018) is introduced by modifying the policy gradient to incorporate an entropy term:

$$X_\theta(a,s) = \nabla_\theta log(\pi_\theta(a|s))(-\alpha log(\pi_\theta(a|s))), \tag{16}$$

$$\nabla_\theta J(\pi_\theta) = E_{s\sim D, a\sim\pi}[X_\theta(a,s) + Q_\psi(s,a) - b(s)]. \tag{17}$$

Combining the attention and soft actor-critic, all the critics are updated together to minimize a joint regression loss function, due to the parameter sharing:

$$y_i = r_i + \gamma E_{a'\sim\pi_{\bar{\theta}}(o')}[Q_i^{\bar{\psi}}(o',a') - \alpha log(\pi_{\bar{\theta}_i})(a_i'|o_i')], \tag{18}$$

$$L_Q(\psi) = \sum_{i=1}^{N} E_{(o,a,r,o')\sim D}[(Q_i^{\psi}(o,a) - y_i)^2], \tag{19}$$

where $\bar{\psi}$ and $\bar{\theta}$ are the parameters of the target critics and target policies respectively. Note that $Q_i^{\bar{\psi}}$, the action-value estimate for agent i, receives observations and actions for all agents. α is the temperature parameter determining the balance between maximizing entropy and rewards. The individual policies are updated by ascent with the following gradient:

$$X_{\theta_i}(a_i,s_i) = \nabla_{\theta_i} log(\pi_{\theta_i}(a_i|o_i))(-\alpha log(\pi_\theta(a_i|o_i))), \tag{20}$$

$$\nabla_{\theta_i} J(\pi_\theta) E_{o\sim D, a\sim\pi}[X_{\theta_i}(a_i,s_i) + Q_i^{\psi}(o,a) - b(o,a_{\backslash i})], \tag{21}$$

where $b(o,a_{\backslash i})$ is the multi-agent baseline.

Algorithm 4 Training Procedure for Attention-Actor-Critic

Inputs: Initialize E parallel environments with N agents
Inputs: Initialize replay buffer, D
$T_{update} \leftarrow 0$
for $i_{ep} = 1$... num epsodes **do**
  Reset environments, and get initial ${o_i}^e$ for each agent, i
  for $t = i$ ... steps per episode **do**
    Select actions $a_i^e \sim \pi_i(\cdot|o_i^e)$for each agent, i, in each environment, e
    Send actions to all parallel environments and get ${o'}_i^e, r_i^e$ for all agents
    Store transitions for all environments in D
    $T_{update} = T_{update} + E$
    if T_{update} >= min steps per update **then**
      for $j = 1$...num critic updates **do**
        Sample minibatch, B
        UPDATECRITIC(B)
      end for
      for $j = 1$ ... num policy updates **do**
        Sample
        $$m \times (o_{1\dots N}) \sim D$$
        UPDATECRITIC($o_B^1 \dots N$)
      end for
      Update target parameters:
      $$Q = \bar{\psi} = \tau\bar{\psi} + (1-\tau)\,\psi$$
      $$\bar{\theta} = \tau\bar{\theta} + (1-\tau)\,\theta$$
      $T_{target} \leftarrow 0$
    end if
  end for
end for

In this project, the actor contains two policies, original policy and target policy. Each policy contains three fully connected layers with leaky relu activation and a $softmax$ activation function to output the action probability distribution. The attention critic (value state function) also contains two critics, original critic and target critic. Each critic is formed by a fully connected followed by an attention head and output fully connected layer. The attention head contains three independent fully connected layers, and the output of the fully connected layers are dot product with each other.

Algorithm 5 Update Calls for Critic and Policies

function: UPDATECRITIC(B)
Unpack minibatch

$$(o^B_{1,..,N}, a^B_{1,..,N}, r1^B_{1,..,N}, o'^B) \leftarrow B$$

Calculate $Q^{\psi}_i(o^B_{1,..,N}, a^B_{1,..,N})$ for all i parallel
Calculate $a'^B_i \sim \pi^{\theta}_i(o'^B_i)$ using target policies
Calculate $Q^{\bar{\psi}}_i(o^B_{1,..,N}, a^B_{1,..,N})$ for all i in parallel, using target critic
Update critic using $\nabla L_Q(\psi)$ and Adam (Diederik P. Kingma, 2014)
end function
function: UPDATECRITIC($o^B_{1,..,N}$)
Calculate $a^B_{1,..,N} \sim \pi^{\theta}_i(o'^B_i), i \in 1, .., N$
Calculate $Q^{\psi}_i(o^B_{1,..,N}, a^B_{1,..,N})$ for all i in parallel
Update policies using $\nabla_{\theta_i} J(\pi_\theta)$ and Adam
end function

4. Evaluation

In this section, first, we introduce details in our environment for not only how agents obtain their own observations but also the details in reward functions. Then, the result provided as win rate in order to compare the performance of four different methods in our environment is shown, followed by a discussion to find out single-agent and multi-agent method, which one is the better in a non-stationary environment. Lastly, since the reward setting plays an important role in the training process, we give examples and description for training under adjusting different reward setting.

4.1. Environment

Our environment is based on Unity, and we use gym (Brockman et al., 2016) and Unity ml-agents API (Juliani et al., 2018) to build the and interact with the environment.

We test various capabilities of single-agent and multi-agent methods through Unity soccer environment. Each agent has their own individual observation instead of global observation.

For each agent, observation is consist of a 112 dimensions vector, which can be divided into $2 \times 7 \times 8$ vector. $2 \times 7 \times 8$ implies 2 z offset, one hot vector [ball, its goal, , enemy's goal, wall, teammate, enemy, none, distance], and 8 directions. Note that one hot vector's element is true when the agent see that object.

The goalie gains -1 when soccer enters ally's goal, +0.1 when soccer enters opponent's goal and an existential bonus +0.001. The striker gains +1 when soccer enters opponent's goal, -0.1 when ball enters ally's goal and an existential penalty -0.001.

4.2. Result

To evaluate different algorithms such as PPO, A2C, MADDPG and MAAC, we train each model for around 10000 episodes. Initially, they play against themselves while in the training process [3]. This training method called self-play. At the beginning of 1500 episodes, these algorithms may only learn the simplest actions. For instance, striker tends running into the goal without kicking the ball, while goalie tends to stuck in the middle of goal and does not know how to defend the coming attack from the opponent. After training the models, we let each model play against another for 100 episodes respectively. Thus, since each model should play against three different models, 300 episodes are played in total for each of them. Hence, we can calculate their win rate. The win rate of different methods are shown in $Table1$. For the video of evaluation, please see the appendix A.

Out of expectation, we find out that A2C, which is single-agent method, has the best performance, while MADDPG, which is a multi-agent algorithm, has the worst performance. We propose some possible reason for this phenomenon.

- MADDPG and MAAC are evaluated on OpenAI particle environment (Mordatch & Abbeel, 2017) in their paper. Though they evaluate their method on competitive environment, the competitive plays in different roles. For example, in predator-prey task of OpenAI particle environment, one team tries to chase other team. However, though our environment is competitive, the two team play the same roles.
- The number of agents may not be high enough in the environment for MADDPG and MAAC, therefore, some issues will be caused. Since each team only consists of 2 members, a striker and a goalie, the importance of cooperation has become much lower. Moreover, if strikers can only focus on kicking the ball instead of taking care of goalies, a team can still get a good win rate.

4.3. Reward Setting

In RL, agents need to interact with the environment to obtain training data (experience). We can adjust the reward setting, which probably makes our model learn more different strategies in the end. Below will show reward setting which we have used during the experiment.

4.3.1. Looking for ball

If we use the original reward setting, reward sparse is serious at the beginning of training. Since the striker can't find the

[3] The self-play training video: [link]

Table 1. Win rate of different algorithms. From the table, we can find that A2C outperform other methods.

Method	Win Rate
PPO (J. Schulman & Klimov., 2017)	0.12
A2C (Volodymyr Mnih, 2016)	0.54
MAAC (Shariq Iqbal, 2019)	0.28
MADDPG (Ryan Lowe, 2017)	0.03

ball, no striker kicks the ball into the goal. Therefore, we try to add new additional reward. When striker sees the ball on any angle, the reward will add 0.0002 on that time step, otherwise, the reward will minus 0.0002 on that time step. Pros are that the striker model will learn to search the ball first when the game start. Cons are that striker model will attack its goal if the striker faces its goal and sees the ball at the same time. However, if we train the policy for a long time, the agent may learn to stick to the ball. Since the training process is self-play, all the strikers will stick the ball together and come to a deadlock[4]. Therefore, the reward will become sparse since nobody kicks the ball to the goal.

4.3.2. Face for ball

Intuitively, we expect strikers to learn kicking the ball straight forward into the opponent's goal. Hence, when a striker faces the ball in straight direction, reward will add 0.0002 on that time step, otherwise, reward will minus 0.0002 on that time step. The striker model will learn to face the ball on straight direction and circle around the ball during training and inference time.

5. Conclusion

The ultimate goal of Reinforcement Learning is to design methods that can learn online in the environments in order to achieve the best performance afforded by the difficulty of the underlying domain. In this project, in order to compare the performance difference between single-agent and multi-agents, we implement four methods for training the environment with multiple agents such as PPO, A2C as single-agent methods and MADDPG, MAAC as multi-agent methods.

Recent results claim that under an environment with multiple agents, single-agent methods no longer have the ability to handle the varying non-stationary environments, hence, the needs to design methods especially for such tasks has grown rapidly. However, our result a different point of view which can be advocated. Under our simulated win rate for the four implemented methods, A2C tends to be the best method among them. This result directly contradicted to the claim from recent results. In our result, we provide a reasonable discussion to conclude the phenomenon between single-agent and multi-agent methods.

[4]Training deadlock video: [link]

References

Ashish Vaswani, Noam Shazeer, N. P. J. U. L. J. A. N. G. L. K. I. P. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

David Silver, Guy Lever, N. H. T. D. D. W. M. R. Deterministic policy gradient algorithms. In *ICML*, 2014.

Diederik P. Kingma, J. B. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

J. Schulman, F. Wolski, P. D. A. R. and Klimov., O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347., 2017.

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. Unity: A general platform for intelligent agents. *CoRR*, abs/1809.02627, 2018. URL http://arxiv.org/abs/1809.02627.

Mnih, Volodymyr, K. K. S. D. G. A. A. I. W. D. and Riedmiller, M. Playing atari with deep reinforcement learning. In *n NIPS Deep Learning Workshop.*, 2013.

Mordatch, I. and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

Ryan Lowe, Yi Wu, A. T. J. H. P. A. I. M. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 2017.

Schulman, John, L. S. M. P. J. M. I. and Abbeel, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

Shariq Iqbal, F. S. Actor-attention-critic for multi-agent reinforcement learning. In *ICML*, 2019.

Tuomas Haarnoja, Aurick Zhou, P. A. S. L. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research*, pp. 1861–1870, Stockholmsmssan, Stockholm Sweden, 2018.

Volodymyr Mnih, Adria Puigdomenech Badia, M. M. A. G. T. P. L. T. H. D. S. K. K. Asynchronous methods for deep reinforcement learning. In *ICLR*, 2016.

A. Match of Different Algorithm

We show the video of our evaluation process.

- A2C (blue) versus PPO (red): [link]
- MAAC (blue) versus A2C (red): [link]
- PPO (blue) versus MADDPG (red): [link]
- MADDPG (blue) versus A2C (red): [link]
- MAAC (blue) versus PPO (red): [link]