CS 7642 Project 3 Football

Ying Huang yhuang836@gatech.edu

Git hash: e15e5c217921345ba2523d067b0e2623a301b8ee

***Abstract*— Reinforcement learning with a multiagent system is a more complex problem than single-agent reinforcement learning. In multi-agent RL we need to achieve not only maximize the reward but studying how multiple agents interact in the same environment. The interaction between agents can be cooperation, competition, or mixed, depending on the environment and training goal. In this report, I experimented with several multi-agent RL algorithms, including Proximal Policy Optimization (PPO) and Importance-weighted Actor-Learner Architecture (IMPALA), in the google research Football environment. My focus is on maximizing the reward while improving the cooperation behavior of players. To compare these algorithms, I analyzed the training results of the trained agents, and the learning metrics and agents' behavior statics during training.**

Keywords—Reinforcement learning, multiagent, cooperation, PPO, Centralized Critic, IMPALA.

I. INTRODUCTION OF THE FOOTBALL ENVIRONMENT

"Football" is a multi-agent environment developed by Google Research, where agents are trained to play football in a simulator. As a multiagent reinforcement learning environment, the goal is to train smart agents that not only can interact with the environment but cooperate with other agents and obtain high rewards.

In this project, I work with a modified football environment which is a 3v3 game where each team has two field players and one goalkeeper. I need to train the two field players to maximize their chances of scoring and strengthen their defense against the opposing team.

In addition, I'm provided with three baseline teams, which are trained by the Proximal Policy Optimization(PPO) algorithm. The three teams are trained differently, and I will compare my team's win rate and behavior metrics with those baseline teams.

A. State, Action and Reward

The observation space I will be using is a modified vector of 43-dimension, which are:
6: (x,y) coordinates of the three left team players
6: (x,y) direction vector of the three left team players
6: (x,y) coordinates of the three right team players
6: (x,y) direction vector of the three right team players
3: (x,y,z) coordinates of the ball
3: (x,y,z) direction vector of the ball
3: one-hot encoding of ball possession (left, right, none)
3: one-hot encoding of active player
7: one-hot encoding of the game mode (which is not applicable for this project).

The action space is discrete 19-action, including a player's action, different ways to kick the ball, and whether to sprint, slide tackle, or dibble.

When a team scores a goal, it gets a +1 reward and the opposing team gets -1. Besides, when a player moves across the opponent's field with the ball, the team can earn a +0.1 reward at each checkpoint. (10 checkpoints in total).

II. ALGORITHM AND APPROACH

Since the football problem is a multi-agent system where the agents have the same goal--win the game, my focus is to improve the collaboration between the players while maximizing the reward.

A. Proximal Policy Optimization (PPO) + Centralized Critic

PPO is an important algorithm in the field of actor-critic and policy gradient, which is proven to have reliable performance and high data efficiency. I want to test this algorithm in football environment and see if it can improve the performance while keeping the training time reasonable. Besides, the centralized critic method can produce a combined observation space, I want to see if it can help with improving the collaboration between the agents.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization is the algorithm that the baseline teams trained with, also is the default training algorithm I'm provided within docker. The PPO algorithm is based on Policy Gradient, which is a method that computes an estimator of the policy gradient and plugs it into a stochastic gradient ascent algorithm.[1] There is a policy loss function that optimizes the objective of policy gradient methods:

$$L^{PG}(\theta) = \widehat{E}_t\left[\log \pi_\theta(a_t|s_t)\, \widehat{A}_t\right].$$

The policy π is the neural network that is trained by a lot of state observation, it will produce the suggested actions. The $\widehat{A}_t$ represents the estimate of the relative value of the selected action, which is calculated with a discount factor γ just like the value function. The $\log \pi_\theta(a_t|s_t)$ represents the logged possibility that policy will output the action a_t. The neural network will be trained by the collected data in the environment which can be very noisy due to the complexity of the system. So the training of the policy neural network can be inaccurate sometimes, it can update the network weights that lead to the wrong policy.

One strategy to prevent that is the Trust Region Policy Optimization(TRPO). In TRPO [Sch+15b], an objective function (the "surrogate" objective) is maximized subject to a constraint on the size of the policy update[2]:

$$maximize_\theta \quad \widehat{E}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\widehat{A}_t]$$

$$subject\ to \quad \widehat{E}_t\big[KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\big] \le \delta$$

The π_θ is the policy after the update and $\pi_{\theta old}$ is the previous policy. The $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the ratio between the action under the current policy and the action under the previous policy. In this way, the policy gradient step can be limited so that it won't update too far away from the previous policy. Combine with TRPO, the PPO purposed the clipped surrogate objective:

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

The expectation is the minimum of normal PG objective and clipped PG objective, which is truncated with clipping operation. The min operation lets the PG objective behave differently, the image below shows the L^{CLIP} under positive and negative advantage estimates.

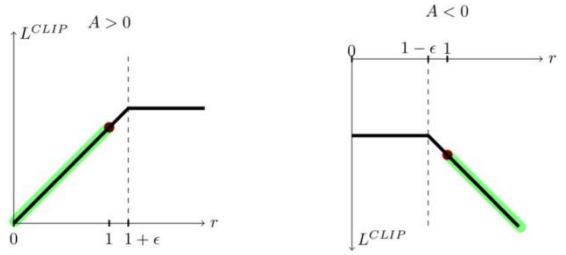

Figure 1. Left image shows the positive advantages and right image shows the negative advantages. The red circle is the optimization starting point.

In Figure 1, the left graph selects the action that has a better effect, and when r goes too high or the action is too different from previous action, the clip objective becomes a flat line to prevent the current policy from changing too much. The right graph shows the disadvantage scenario, when r goes too small near zero, the L^{CLIP} flattens out to prevent stepping too far away from the previous policy.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \le NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 2: PPO algorithm of parallel actors

Now let's look at the policy updating algorithm shown in figure 2. PPO works in two phases, the first phase collects data and calculates advantage estimates. It will sample a minibatch from the data for later use. In the second phase, a large number of actors are performed in parallel. Then we can construct the surrogate loss and optimize it. The algorithm uses SGD optimization to find the policy that maximizes the objective.

Centralized Critic

Based on the PPO algorithm, I implemented the centralized critic method in the training model. The centralized critic networks use centralized observations O_t^j as the input, where j represents each agent, and t is the current time step. Each agent j contributes weights θ_v^j to the networks.

The architecture of the centralized critic is shown in figure 3. The central observation combines all agents' partial local observation and creates a "fully-observable" observation. The observation O_t^j is depending on how much information the environment can provide to the agents. After combining the observations, the workers will optimize the critic loss function $L_{V_t}^j$ by minimize the squared difference between the value estimation $V(S_t^j, \theta_v^j)$.

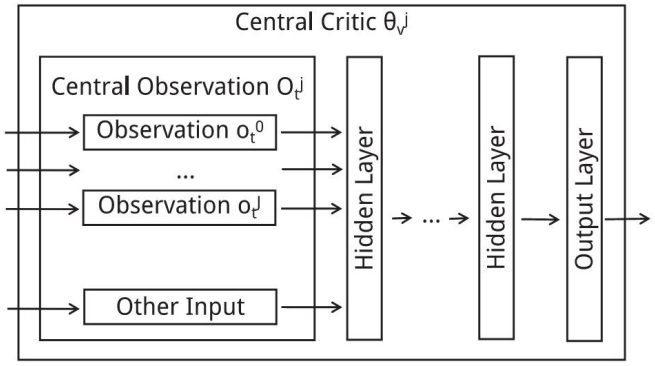

Figure 3. A centralized critic example

B. *Importance Weighted Actor-Learner Architecture (IMPALA)*

IMPALA is a high throughput algorithm that has a stable learning performance. I want to test if IMPALA can produce high performance in a relatively shorter time. Besides, IMPALA uses V-trace to correct the policy lag, I want to test if it can also improve the collaboration behavior.

One of the biggest difficulties I faced in this project is the massive data and overwhelming training time. I have a machine with 8 GPUs and 16GB of memory, so it's critical to find a way to deal with the increasing data and long training time.

IMPALA is similar to the A3C algorithm, which consists of multiple weighted independent agents/workers, each of the workers has a parallel copy of the environment, so that they can explore more possible spaces in a limited time. The workers in A3C are trained independently but will update to a shared global network periodically. The updates between the worker aren't happening at the same time, which is an asynchronous process, and each worker will keep training independently after it.

The IMPALA is proven to be more efficient in data processing ad resource utilization than A3C, it uses decoupled acting and V-trace off-policy correction method to achieve

high throughput. Instead of periodically updating the workers' network to the global network, IMPALA actors communicate the trajectories of experience by a centralized learner. The learner is also training constantly in parallel. This mechanism can provide high throughput, but it also has the disadvantage that the centralized learner's training is behind the actual gradient updates. The learning becomes off-policy, for that IMPALA provides a new actor-critic algorithm V-trace.

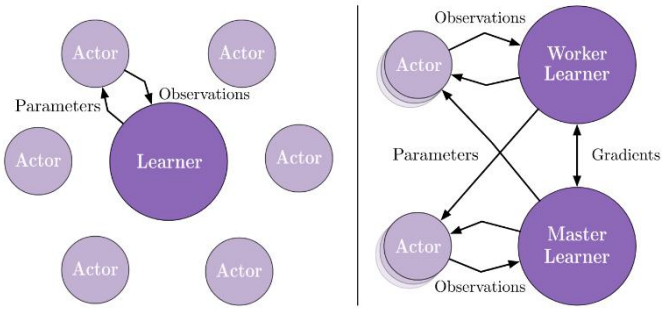

Figure 4. Left: Single Learner. Each actor generates trajectories and sends them via a queue to the learner. Before starting the next trajectory, actor retrieves the latest policy parameters from learner. Right: Multiple Synchronous Learners. Policy parameters are distributed across multiple learners that work synchronously.[3]

As shown in figure 1, IMPALA uses actor-critic to learn policy $\boldsymbol{\pi}$ and a baseline function V^{π}. The trajectory experience is generated first by actors, then the learners will use the experience to learn policy. Before generating the trajectory experience, the actor will update its local policy $\boldsymbol{\mu}$ to the learner's policy $\boldsymbol{\pi}$. Then the actor runs the local policy for n steps and sends the trajectory of states, action, reward $(x_1, a_1, r_1, \ldots, x_n, a_n, r_n)$,and the policy distribution $\boldsymbol{\mu}(a_t|x_t)$ to the learner. The learner will update the policy $\boldsymbol{\pi}$ by the trajectories.

Because of the decoupling architecture, the actor's policy $\boldsymbol{\mu}$ is several updates behind the learner's policy $\boldsymbol{\pi}$. The V-trace can correct the policy-lag while achieving a high throughput. Like the Markov Decision Process with discounted reward and infinite horizon, the goal is to find a policy that can maximize the expected sum of future rewards, and the goal for off-policy RL is to use trajectories generated by behavior policy $\boldsymbol{\mu}$ to learn the value function V^{π} of the target policy $\boldsymbol{\pi}$. Let's consider the trajectory $(x_n, a_n, r_n)_{t=s}^{t=s+n}$, the V-trace target for value approximation at state x_s can be defined as:

$$v_s = V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \Big(\prod_{i=s}^{t-1} c_i \Big) \delta_t V$$

The $\delta_t V = \rho_t(r_t + \gamma V(x_{t+1}) - V(x_t))$ is the temporal difference for V.

The $\rho_t = \min(\bar{\rho}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)})$ and $c_i = \min(\bar{c}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)})$ are truncated importance sampling weights.

Let's assume $\bar{c} \geq 1 \Rightarrow c_i, \rho_t = 1$, then we can get the equation below, which is transformed to n-steps Bellman equation:

$$v_s = V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s}(r_t + \gamma V(x_{t+1}) - V(x_t))$$

$$= \sum_{t=s}^{s+n-1} \gamma^{t-s} r_t + \gamma^n V(x_{s+n})$$

Thus in the on-policy case, V-trace reduces to the on-policy n-steps Bellman update. This property (which Retrace (Munos et al., 2016) does not have) allows one to use the same algorithm for off- and on-policy.[4]

III. Algorithm Implementation & Training Behavior

In this part, I implemented the algorithms and trained the multi-agents. I'm interested in the cooperative behavior between the players and their roles in the game. So I will use several behavior metrics to analyze their cooperation during the training.

To implement the PPO algorithm with a centralized critic model, I adapted the PPO algorithm config provided in the "train_agents.py" file and plugged in a centralized critic model architecture by defining the central critic loss function, and centralized critic policy. For IMPALA, I implemented the actor-critic method and V-trace and used multi-agent policies.

1. Learning Curve

I ran the training for 5M times for each algorithms and get the learning curves as below (figure 4), which are represented by the average reward of each training episode:

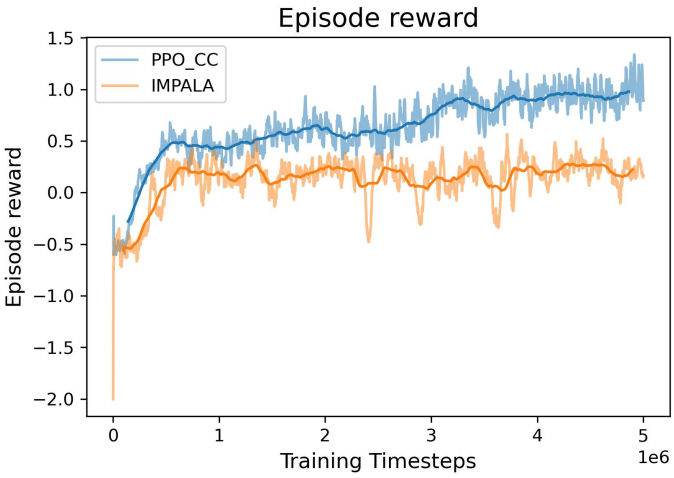

Figure 5: Reward per episode while training

We can see that the episode average reward increases for both algorithms, but PPO with centralized critic has better growth. The IMPALA grows fast in the first million training, but it stops growing after that. PPO_CC grows constantly till 5M and reaches 1.0 reward.

2. Teammate-Spread-Out

Teammate-spread-out is the average distance between the two players on the pitch during the game. It shows the evolution of the spread of teammates and how they learned to cooperate not just chase the ball.

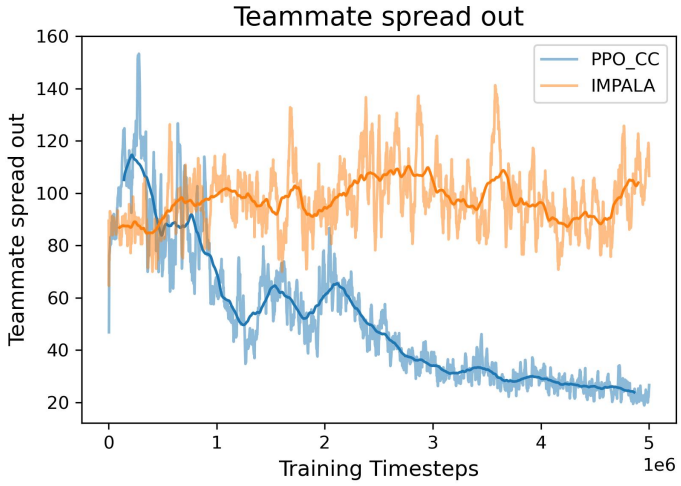

Figure 6: Teammate spread out while training

We can observe that the IMPALA team always has a stable and high spread-out, which may imply that they have a stable cooperation strategy, or maybe they don't care what the other player is doing. Based on the learning curve, the latter is more possible. PPO_CC team first spreads out largely but then becomes close to each other. This may show that PPO_CC teammates stay close and chase the ball together as the training goes, they might not develop a very good cooperation strategy.

3. Tired Factor Rate

During the game, each player has a tired factor which shows how much distance the player runs and some difficult skills the player uses. The tired factor rate is calculated by player_1_tired_factor/player_2_tired_factor. The ideal rate should be near 1.0, that way the efforts of each player on the pitch are balanced.

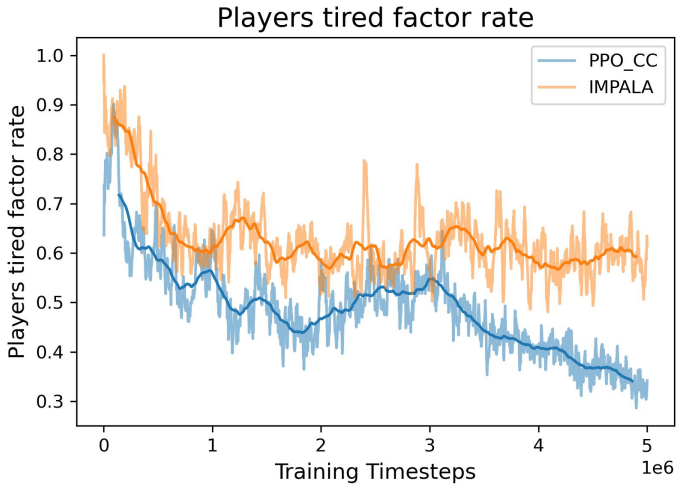

Figure 7: Players tired factor rate= player_1_tired_factor/player_2_tired_factor

We can see that both teams have a rate below 1.0, which means player_2s are doing more work than player_1s. As the training step goes up, this imbalance is getting worse, which could be because the training gets more effective to player_2s, and player_2s become dominant in the game. But the IMPALA team is more balanced than PPO_CC

4. Pass/Interception Rate

Pass is when players from the same team kick the ball consecutively, and interception is when players from the different team kick the ball consecutively. The pass/interception rate is the average ratio of the team passing the ball and getting balls from the opposing team. Pass is a more difficult skill because it requires cooperation between the players.

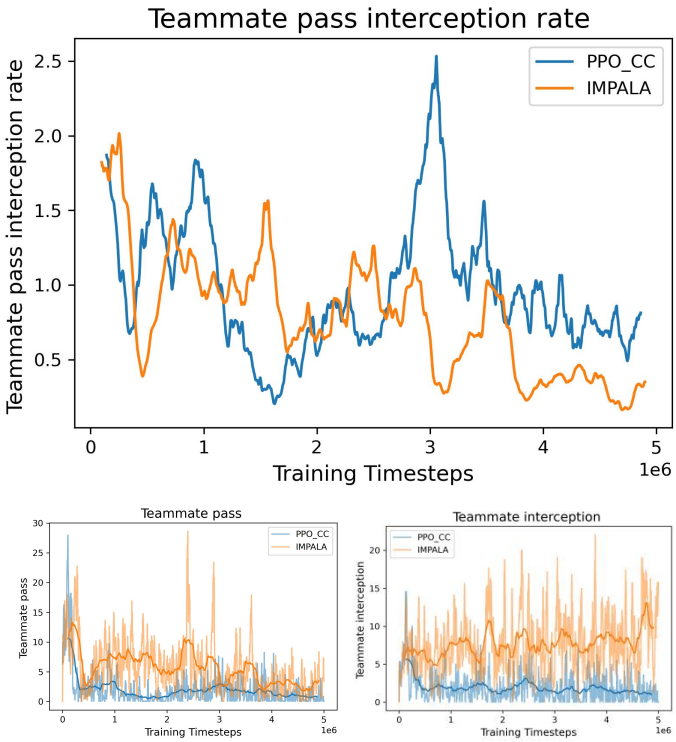

Figure 8: Pass/interception rate

We can see that IMPALA has more pass count and interception count than the PPO_CC, combined with the learning curve, it can be explained as PPO_CC is more efficient in terms of game strategy, it can score higher with fewer passes and interceptions. According to the pass/interception rate, we can observe that overall the ratios are near 1.0, which is normally a good sign meaning the agents learned the difficult skill of passing. But based on the fact that the agents are not trained very well, this could be a coincidence that the agents are still exploring the policy space.

IV. COMPARISON WITH THE BASELINE TEAMS

After training the model, I selected the best checkpoint of each algorithm, then let my trained teams and the baseline teams play against the hard-coded game AI. Finally, I compared the performance and behavior metrics of my team with the three baseline teams.

1. Win Rate

The win rate shows how many games the trained teams and baseline team can win in 100 games against the game AI.

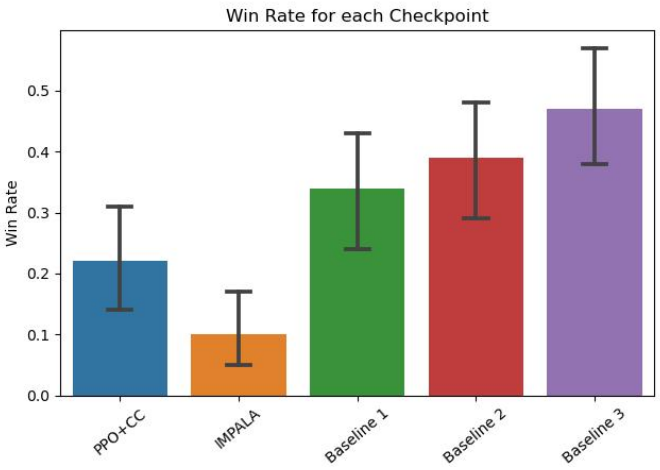

Figure 9: Win rate and undefeated rate comparison

The figure shows that the win rate is Baseline teams > PPO_CC > IMPALA. The 2 trained teams didn't beat the baseline teams in win rate, I think this is due to I didn't train the teams with more time steps, and the hyper-parameters I set are not optimal for the football environment. If I have more time, I would train the agents longer and get a better result.

2. Players Tired Factor Rate

I also compared the players' tired factor rate during the 100 consecutive games, the players' tired factor rate is calculated as player_1_tired_factor/player_2_tired_factor. We can see in figure 10 below that the baseline teams have tired factor rates close to 1.0, which means the efforts players take in the same team are more balanced. In reverse, the PPO_CC and IMPALA teams have less balanced tired factor rates, player 2s are doing more work during the game. Again, the baseline teams are better in collaboration because they well utilized both players and get a good training result.

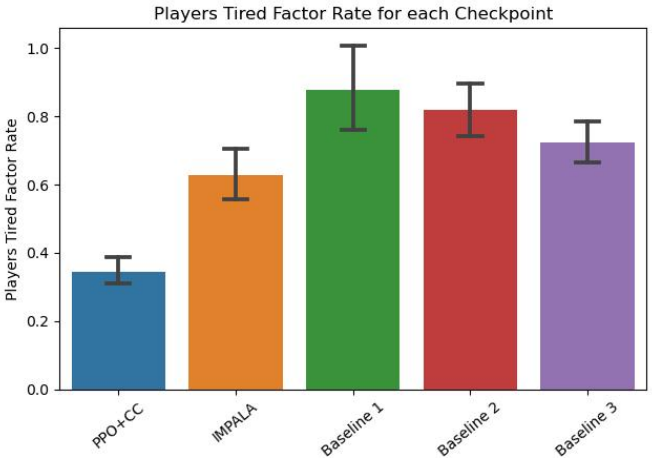

Figure 10: Player tired factor rate comparison

3. Teammate Pass Number

Pass is when the two players in the same team pass the ball to each other, the metric shows the average times the team passes a ball in the game. From figure 11 we can observe that the IMPALA team passes a lot of balls in the game, and PPO_CC almost never passes a ball in the game. For baseline teams 1 and 3, they rarely passed the ball. However, the baseline team 2 passes two balls on average for each game, which is an ideal cooperation behavior. We can assume that baseline team 2 is trained to have more cooperation skills. IMPALA passes too many balls in a single game which is not necessary, it's possible that the IMPALA agents pass the ball back and forth which is not a very good collaboration skill.

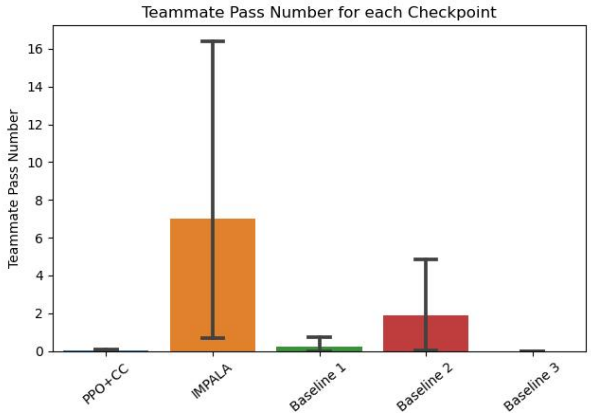

Figure 11: Teammate pass numbers comparison

CONCLUSION AND PITFALLS

After comparing the PPO_CC and IMPALA with the baseline teams, I have to sadly admit that baseline teams won on both win rate and collaboration skills. The main reason is that I didn't train the agents with enough time steps and I'm running out of time. Besides, I took too much time trying to understand the hyper-parameter in the Ray library but still didn't find a good combination. I also spent a lot of time trying to implement other algorithms like MADDPG, COMA, and QMIX, but they either didn't implement successfully or failed to get a good win rate.

After all, within the two algorithms I implemented, PPO with centralized critics performs better than IMPALA. I believe that PPO with centralized critics has a good potential to get good performance if I train it with more time steps. IMPALA is a high throughput algorithm, the training time is at least two times shorter than the PPO_CC, but the training result is not as good. Based on the pass/interception rate and players' tired factor rate, I assume that the IMPALA has more collaboration behaviors, but passing the ball so many times in a game is a little odd, if I have more time I would study what's the cause of it.

REFERENCES

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

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

[3] Espeholt, Lasse, et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures." International conference on machine learning. PMLR, 2018.

[4] Espeholt, Lasse, et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures." International conference on machine learning. PMLR, 2018.