CS7642 Project 3 Report

Git hash - 3fa00be3a0c77fd397ab592346b762026e87447b

***Abstract*- The goal of this project is to develop a multi-agent reinforcement learning algorithm for Google Research's football environment. Three baseline teams have been provided, and a variety of algorithms are tested to gauge their ability to improve on certain aspects of the baseline teams. It was found that the Ape-X DQN and PPO algorithm, implemented using Ray's RLlib library, were most suitable for best returns. The Ape-X algorithm in particular is analysed and compared to the PPO baselines, with suggestions made on how its results could be improved upon.**

Introduction

The *Google Research Football Environment* is a reinforcement learning (RL) environment where agents are trained to play football (a.k.a. soccer) in a physics-based 3D simulator and is designed to provide support for multi-agent RL experiments [1]. Published research in this environment investigates the use of reinforcement learning algorithms such as IMPALA, PPO and APE-X DQN [1]. Additionally, similar environments exist that utilise a multi-agent approach, such as DeepMind's MuJoCo [2].

Problem Definition

For this assignment, three baselines teams have been provided, each trained using the Proximal Policy Optimisation (PPO) Algorithm using Ray's RLlib library [3]. However, each one has been trained using a different set of parameters. The goal of this assignment is to implement another algorithm/model that improves upon certain aspects of the baseline teams.

The environment created is a 3v3 scaled-down version of the original 11v11 implementation. The agent implemented will take control of two outfield players in the left team (attacking to the right), where the goalkeeper is automatically controlled. The opponent AI controls all three players in the right team (attacking to the left).

The observation space for this environment is high-dimensional and continuous and represented as a 43-dimensional vector. Additionally, the action space is a discrete set of 19 actions. The reward function is split into two categories; the score reward and the checkpoint reward. If a team scores a goal, it receives a +1 reward, while the opposition receives a -1 reward. Additionally, up to an additional 10 checkpoint rewards are provided if a player in possession crosses distance checkpoints, receiving a +0.1 reward for each checkpoint crossed. The score and checkpoint rewards combined are the "episode reward".

Chosen Solution Strategies

The first strategy is to trial a variety of algorithms to discover which one has the greatest potential to beat the baseline. Given the computational and time limitations, the algorithm had to meet the following two criteria to be used for evaluation/analysis later on. They are referred to as the "target training criteria", and use the following custom metrics:

- *win_percentage_episode_max*- All baselines were able to return a value of 1 for this metric after 5 hours of wall-clock time (excluding the rare outliers). Therefore, If the new model was unable to achieve this trend, it was not considered further.
- *"win_episode_percentage_mean"* & *"episode _reward_max"*- these two are used in tandem. For certain models, their average percentage of episodes won plateaus, which may give an impression that they've stopped learning, however, this is not the case (as shown by Baseline 3 in Figure 1). If this behaviour is noticed for longer than 5 hours of wall-clock time, then *"episode _reward_max"* is analysed. If the agent is not scoring close to the max of 4.0 (i.e. 3.8 or higher), that means it may be struggling to acquire its checkpoint rewards and score, hence is not considered for evaluation.

This approach may be considered empirical and therefore, not necessarily definitive. This is because it is likely for a model to take several hours to perform poorly at the metrics above but then learn rapidly to outperform expectations (similar to baseline 3). However, it is difficult to predict this in training, hence the aforementioned time limits are imposed in the "target training criteria". Furthermore, the wall-clock hours taken in training are used as a threshold instead of timesteps taken in the assessment because the algorithms have different training rates (e.g. Ape-X can complete over 2 million timesteps in an hour, whereas PPO would complete 650,000).

As previously mentioned, Google's Brain team analysed three algorithms, PPO (Proximal Policy Optimisation) [3], IMPALA (Importance-Weighted

Actor Learner Architecture) [4] and Ape-X DQN (Distributed Prioritized Experience Replay) [5]. These three algorithms were selected for this analysis, along with SAC (Soft Actor-Critic) [6], deployed using Ray's RLlib package, and were trained and analysed based on the two "target training criteria".

Algorithms/ Models attempted

SAC (Soft Actor-Critic)

The algorithm consists of an actor-critic architecture (similar to DQN) with separate policy and value function networks. It also has an off-policy formulation that enables the reuse of previous data, and entropy maximisation to encourage stability and exploration [6].
SAC was unable to pass the two target training criteria since it failed to consistently win after 5 hours. Therefore, it was not evaluated any further.

IMPALA

This is a highly scalable algorithm that decouples the acting and learning elements. A central learner is provided with trajectories of experience from individual workers. The RLlib implementation uses DeepMind's reference V-trace code [7]. IMPALA, similar to SAC, was also unable to pass the target training criteria, and hence was not evaluated further.

PPO

Proximal Policy Optimization is an online policy gradient algorithm that alternates between sampling data through interaction with the environment and optimizing a "surrogate" objective function using stochastic gradient ascent [3]. This is the algorithm chosen for all baselines and lends itself well for use in multi-agent environments. As shown by the baseline values, PPO passed both the target training metrics and a custom model is considered for further assessment.

APEX-DQN

DQNs use a multi-layer perceptron, rather than a Q-table, to estimate the Q-values as state-action pairs for the given state [8]. However, DQN showed a poor training rate and was unable to pass the target training criteria. Given that the number of workers could not be increased to scale for the environment, Ape-X DQN was selected instead. Additionally, Ape-X also allows for faster training at a comparable timestep efficiency [9]. In particular, Ape-X extends the prioritized experience replay memory to the distributed setting to enable higher scalability, done via multiple actors. The learner then samples from this memory, updating the network and the priorities of the experience.

Experiments and Analysis

Initial Parameters

Certain parameters remained constant from their baseline implementations to all the custom models implement. These include:

- *"rollout_fragment_length"*- set to 100.
- *"batch_mode"*- set to "truncate_episodes", so each call to "sample()" will return a batch of length *"rollout_fragment_length"* at most.
- "model"- all are set to "fcnet", and consist of 2 hidden layers.
- "train_batch_size"- set to 2800

Custom models were created using PPO and Ape-X DQN to maintain simplicity. The main reason for doing so was to try and directly compare the effectiveness of PPO relative to Ape-X. This is because PPO takes some parameters not used by Ape-X, such as the "kl_coeff" parameter. Hence using similar parameters is done in an attempt to allow for a more direct comparison.
The key hyperparameters used are shown below:

	Custom PPO	**Custom APE-X**
Framework	PyTorch	TensorFlow
Learning rate	0.0003	0.0003
Gamma	0.99	0.99
Hidden layers	[512,512]	[512,256]
LSTM usage	No	Yes
SGD minibatch size	256	Not specified

Table 1- Key hyperparameters used for the custom models

Training analysis

It should be noted that the goal of this assignment was to implement new algorithms other than PPO, therefore, the focus is on training, improving and analysing the custom Ape-X DQN model rather than the custom PPO model. Due to time and computational constraints, the custom PPO model was only trained for approximately 13 million timesteps (relative to the 50 million spent on the baseline models). Alternatively, the Custom Ape-X model was trained for approximately 70 million timesteps (data shown in Figures 1 and 2).

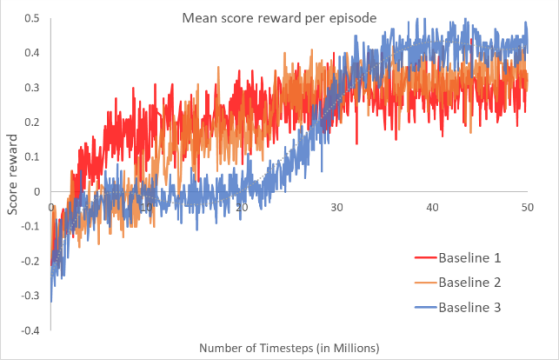

Figure 1- Training graph for the baseline models, showing the average score reward per episode

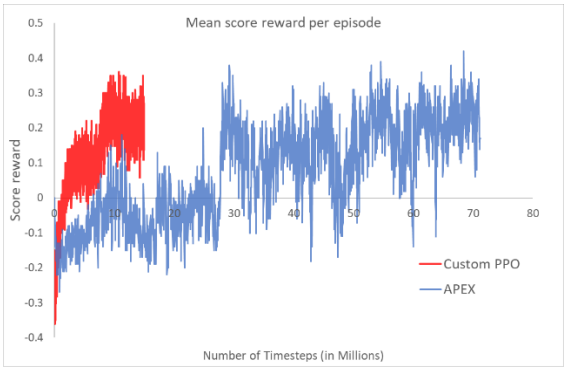

Figure 2- Training graph for the custom PPO and APE-X models, showing the average score reward per episode

As shown in the figures above, Baseline 3 achieves the highest average score reward of all models, with a value of approximately 0.45, whereas baselines 1 and 2 both achieve between 0.3 and 0.35. As for the custom models, both custom PPO and Ape-X are close to baseline 1, with an average score reward of approximately 0.3. It should be noted that the average score reward per episode is highly correlated with the average wins per episode. When viewing the Ape-X training graph, it can be seen that Ape-X has an appreciable increase in its score reward after approximately 30 million timesteps. After this, the reward seems to stabilise, with no such appreciable increase.

Testing analysis

After training the models, they are then evaluated in the 3v3 game environment. This is done for a total of 100 episodes, which alternate between the controlled agent kicking-off and the opposition kicking-off (i.e. 50 episodes for each team). Along with the win rate, metrics such as the win, loss and tie rate for each agent if they had their kick-off is analysed. This is coupled with the number of slide tackles the controlled agents make. Finally, each model is analysed by the average number of timesteps taken to win.

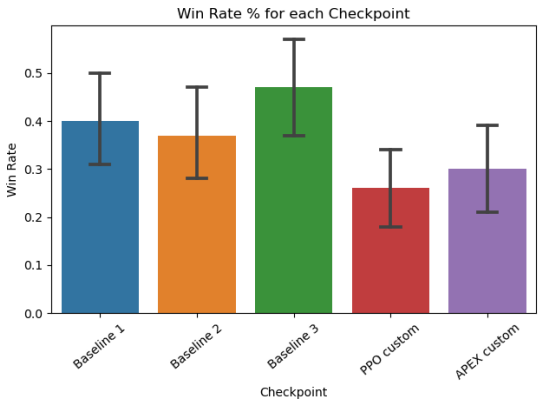

Figure 3- Win rate for each model (i.e. Average number of wins per episode), along with the standard deviation

As shown in the figure above, the average values and standard deviations show that the custom APE-X algorithm implemented is competitive with the baseline algorithms.
However, it is useful to analyse this winning trend further by analysing another metric; game outcome based on which team kicked off.

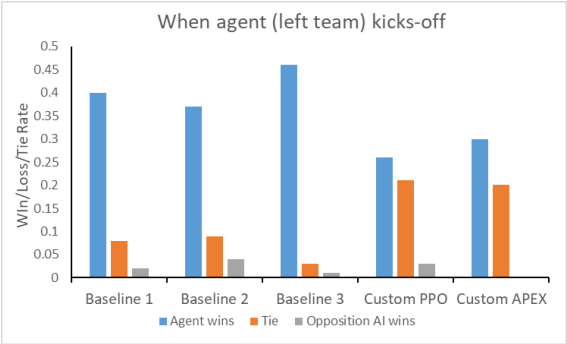

Figure 4- Rate of game outcome for model (Win/loss/opposition win) when the controlled agent (left team) kicks-off

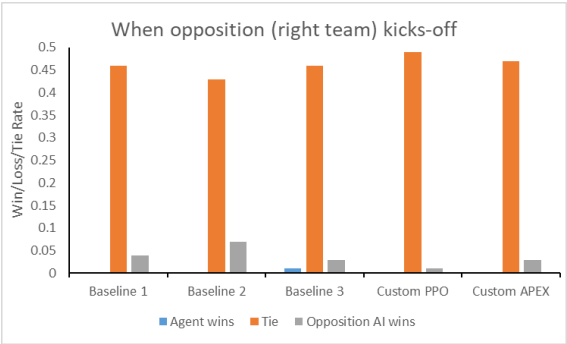

Figure 5- Rate of game outcome for model (Win/loss/opposition win) when the opposition (right team) kicks-off

As seen, all of the agents' wins come when it kicks off the game, and none of them occurs when the opposition kicks off (apart from a single with for Baseline 2). Upon visualising the match videos, it is evident this is due to the inability to win the ball successfully when the opposition has possession. The quickest method for an agent to regain

possession of the ball from the opposition is to successfully execute a sliding tackle. Therefore, the sliding tackles each agent makes are shown in Figure 6. The PPO models are found to execute several tackles when not in possession, especially Baseline 1. However, an inspection of the videos shows that an overwhelming majority simply miss the ball, resulting in the agent not acquiring possession successfully. If these slide tackles are done on an agent, an overwhelming majority leads to fouls, which ends the episode prematurely. Almost none lead to possession regained fairly and successfully. As for the Ape-X model, its drawback is its reluctance to slide. The agent never engaged in a slide tackle when not in possession, leaving it up to only the goalkeeper to win back possession. This severely affects its ability to overcome its opponent when it does not have its kick-off.

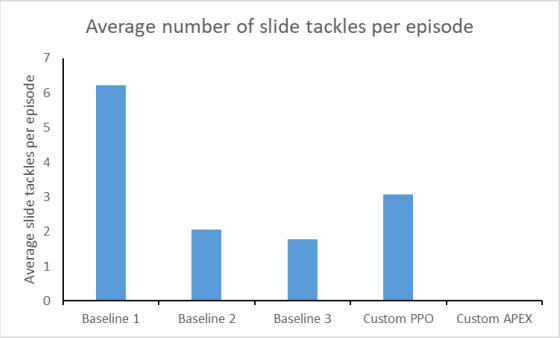

Figure 6- Depiction of the average number of slide tackles the agent executes per episode

Additionally, given that the agents only win when they kick-off, a useful metric to analyse would be the number of timesteps the agent takes to score.

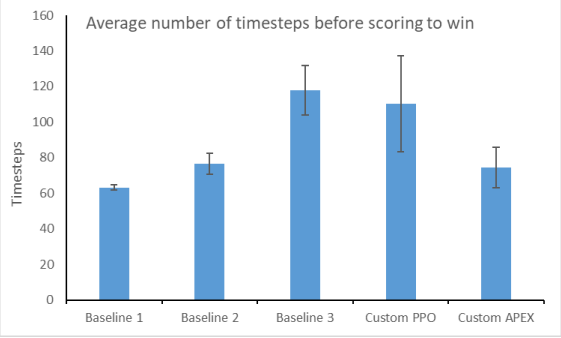

Figure 7- Depiction of the average number of timesteps taken to score when the agent wins after kicking-off (NOTE: outliers more than three times greater than the average were not considered)

As seen from Figure 7, Baseline 1 takes the least number of timesteps (on average) to score, closely followed by Baseline 2 and the Ape-X model. Additionally, Custom PPO and Baseline 3 take up to an additional 50 timesteps to score when compared to Baseline 1. Upon inspecting the videos, it is found that the model for Baseline 1 tends to sprint directly at the goal and take a shot early, whereas the agent using Baseline 3's model prefers to dribble patiently and wait to take a shot. Upon closer inspection of the parameters, this is thought to be a product result of the inclusion of the LSTM (Long short-term memory) parameter as part of the neural network. Baseline 1, baseline 2 and Ape-X contain LSTM in their neural networks model, whereas Baseline 3 and Custom PPO do not have them. LSTM is known to help speed up the agent's learning by exploiting its network structure [7], which could be the reason why Baselines 1 and 2 learned faster than Baseline 3 in the first 10 million timesteps (refer to Figure 1), and also try to maximise reward capturing efficiency by racing through the checkpoints and scoring as early as possible. Generally, it seems that Baseline 3's approach is more suitable due to its ability to dribble around an opponent, whereas baseline 1 and Ape-X can end up sprinting directly into opponents, losing possession, which explains their inferior win rate relative to baseline 3. However, Custom PPO was not as effective as baseline 3 in terms of win rate because it tried to run around the keeper rather than shoot in time, making it lose the ball easily. Additionally, an attempt was made to train Ape-X without LSTM, but this did not prove to be successful as the agent failed to surpass a mean win rate of 0 in training after 20 million timesteps.

Challenges, pitfalls and potential improvements

Reward shaping

Reward shaping was also tried with the Ape-X model to try to improve the win rate when the agent kicks off. After watching videos, it was apparent that most successful goals were scored by staying in a fairly central region and the final third of the opponent's half. Therefore, the agent was rewarded for bringing and keeping the ball in this region (ranging from +0.001 and +0.002 per timestep). However, when implemented on Ape-X DQN, this reduced the mean episodes won during training, hence was not considered any further.

Upon realising that the Ape-X model was unable to win the ball when the opponent kicked off, the intention was to try and implement a reward-shaping that would incentivise the agent to win the ball back. Initially, positive rewards (of +0.05) were provided for every slide tackle the agent executed when not in possession of the ball. The intention was to reward the agent only if the ball was won, but implementing this code was challenging as it

was found that the agent may not own the ball immediately after the tackle, hence tracking whether the tackle was successful in winning possession was difficult. However, in hindsight, I would have trained Ape-X with rewards assigned for only executing a tackle regardless of success, just to get the APE-X agent to tackle as frequently as the PPO agents, and test its success.

Feature Engineering (i.e. action masking)

As a potential improvement going forward, it would have been worthwhile to implement "action masking". This would be useful in trying to improve the APE-X agent's ability to win games where the opponent kicks off. Upon analysing the videos, it was noticed that the agent was attempting actions that were unnecessary when not in possession, such as making high passes, long passes, short passes and shots. Excluding these actions from the agent's availability in training could help in speeding training by not allocating resources to these irrelevant actions. Similarly, actions that are irrelevant when the agent owns the ball can also be removed, such as executing sliding tackles. This has been done when implementing RL algorithms in multi-agent environments with success [10].

Hyperparameters Tuning

The parameters used for the custom Ape-X model are not tuned as thoroughly as the baseline PPO models seem to be. Given that the goal of this project was to implement another algorithm and study its effectiveness, rather than tuning hyperparameters, the latter was not done in detail. However, if conducted, it may enhance the Ape-X model such that it outperforms the PPO models.

Conclusion

PPO has been found to perform strongly (in terms of the number of games won) when used in multi-agent environments without a lot of hyperparameter tuning and domain-specific modifications. This has also been observed in similar research in multi-agent environments [11].

Ape-X DQN was deployed to try and assess its applicability in performing well in such a multi-agent environment relative to PPO. Ape-X was able to be competitive with the PPO (with a win rate of 30%) equivalent despite not being able to surpass them (where the best PPO baseline models had win rates between 35% to 45%). However, it performed significantly better than other models designed for multi-agent environments, e.g. SAC & IMPALA. Ape-X's success here may be due to the use of prioritised experience replay and reward clipping [5].

The key drawback for all models evaluated was their inability to win the ball from the opponent to regain possession. However, potential improvements in the form of reward shaping and action masking were recommended.

Finally, the inclusion of LSTM in the neural network structure of all models was found to make the agent more eager to score quickly (taking fewer timesteps) but did not improve its win rate.

References

[1] Google Research, Brain Team, "Google Research Football: A Novel Reinforcement Learning Environment," Association for the Advancement of Artificial, 2020.

[2] DeepMind, "EMERGENT COORDINATION THROUGH COMPETITION," in *ICLR 2019*, London, 2019.

[3] OpenAI, "Proximal Policy Optimization Algorithms".

[4] "DeepMind Technologies," London, 2018.

[5] DeepMind, "DISTRIBUTED PRIORITIZED EXPERIENCE REPLAY," in *ICLR*, 2018.

[6] T. H. e. al., "Soft Actor-Critic Algorithms and Applications," 2019.

[7] DeepMind Technologies, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," London, 2018.

[8] K. K. D. S. e. a. Voldymr Mnih, "Playing Atari with Deep Reinforcement Learning," DeepMind, 2013.

[9] The Ray Team, "RLLIB Algorithms," [Online]. Available: https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#dqn. [Accessed 23 07 2022].

[10] Tencent AI Lab, Tencent Timi Studio, "Mastering Complex Control in MOBA Games with Deep Reinforcement Learning," Association for the Advancement of Artificial Intelligence, 2020.

[11] A. V. e. a. Chao Yu, "The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games," 2021.