# Reinforcement Learning: Football with Manchester

Team 2: Florina Asani, Pascal Bakker, Amisha Jindal, Oscar Garcia Fernandez

*Abstract*—**Complex games such as football are difficult for A.I. to solve. However, one particular type of A.I., reinforcement learning, offers the capability of developing a high-performance algorithm for such games. This project seeks to investigate different types of reinforcement learning algorithms, particularly, policy-based algorithms, to evaluate which ones perform optimally in multi-agent environments.**

***Keywords— dqn, trpo, ppo, a3c, reinforcement learning, football***

## I. Introduction

The objective of Reinforcement Learning (RL) is to prepare brilliant agents that can cooperate with their current circumstance and unravel complex errands, with true applications towards advanced mechanics, self-driving vehicles, and the sky's the limit from there. The fast advancement in this field has been energized by making agents mess around, for example, the notable Atari console games, the antiquated round of Go, or expertly played computer games like Dota 2 or Starcraft 2, all of which give testing conditions where new calculations and thoughts can be immediately tried in a protected and reproducible way.

In many of these games, played by reinforcement learning agents there is a clear sequence of events that leads to achieving a positive reward or avoiding a negative one. In "Breakout" one could train the agent to break the bricks to get points or to prevent the ball from reaching the bottom of the screen to avoid a game over.

Football is not as simple. The team must simultaneously strategize how to score goals and how to defend from the opponent. Precise gameplay is necessary to win. In the simulation, the agent will only be able to control one player at a time and must develop strategies that utilize the entire team such as making a pass to circumvent a goalie. Success in this project requires a single agent to solve multiple situations and leverage teamwork to win.

## II. Related work

### A. Google Research Football Environment

Google Research Football Environment [2] is a novel RL climate where agents intend to dominate the world's most mainstream sport - football. Demonstrated after famous football computer games, the Football Environment gives a material science based 3D football reproduction where agents control possibly one or all football players in their group, figure out how to pass among them, and work out how to defeat their rival's guard to score objectives. The environment gives a few urgent segments: an exceptionally improved game motor, a requesting set of exploration issues called Football Benchmarks, as well as the Football Academy, a bunch of continuously hard RL situations [3].

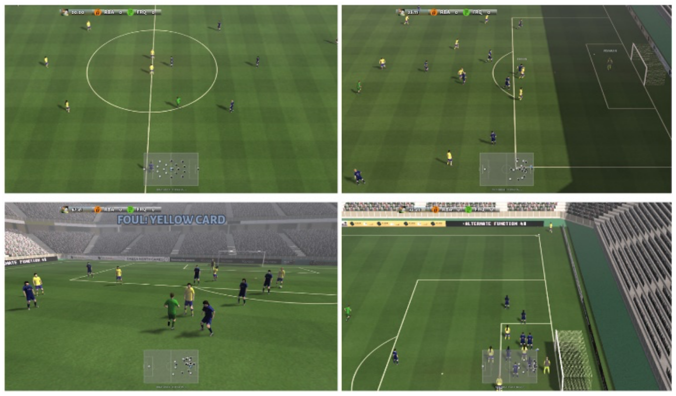

*Figure 1. Football Engine supports kickoffs (top left), goals (top right), fouls, cards (bottom left), corner and penalty kicks (bottom right), and offside*

### B. Football Engine

The center of the Football environment is a serious football reenactment, called Football Engine, which depends on a vigorously changed form of Gameplay Football. In view of info activities for the two rival groups, it mimics a match of football including objectives, fouls, corner and extra shots, and offsides. The Football Engine is written in profoundly advanced C++ code, allowing it to be run on off-the-rack machines, both with GPU and without GPU-based delivering empowered. This permits it to arrive at an exhibition of around 25 million stages for each day on a solitary hexa-center machine [3].

The Football Engine has extra highlights prepared explicitly for RL. To begin with, it permits gaining from both distinctive state portrayals, which contain semantic data such as the player's locations, as well as grasping from crude pixels. Second, to explore the effect of arbitrariness, it very well may be executed in both a stochastic mode (empowered naturally), in which there is irregularity in both the climate and adversary AI activities, and in a deterministic mode, where there is no irregularity. Third, the engine is out of the crate viable with the generally utilized OpenAI Gym API. Lastly, scientists can get an inclination for the game by playing against one another or their agents, utilizing either consoles or gamepads [3].

### C. *Football Benchmarks*

With the Football Benchmarks, a bunch of benchmark issues are proposed for RL research dependent on the Football Engine. The objective in these benchmarks is to play a "standard" round of football against a fixed guideline-based rival that was hand-designed for this reason. Three versions are given: the Football Easy Benchmark, the Football Medium Benchmark, and the Football Hard Benchmark, which just vary in the strength of the adversary.

### D. *Football Academy & Future Direction*

As preparing agents for the full Football Benchmarks can be testing, we likewise give Football Academy a varied arrangement of situations of changing trouble. This permits researchers to get the show on the road with new exploration thoughts, permits testing of high-level ideas (for example, passing), and gives an establishment to examine curriculum learning research thoughts, where agents gain from continuously harder situations. Instances of these situations incorporate settings where agents need to figure out how to score against an empty goal, where they need to figure out how to rapidly pass among players, and where they learn how to execute a counterattack. Utilizing a simple API, researchers can additionally characterize their own situations and train agents to work through them.

## III. Environment specifications

### A. *State and Observation Space*

The raw observation gives us the state definitions which are as follows: ball position and possession, player positions, the current scoreline, the game state (yellow cards, red cards, score etc.), current active player.
In addition to raw representation, the environment also provides us with two other representation wrappers such as SMM (Simplified Spatial Representation) and Simple115.

### B. *Action Space*

In this project, we were only able to control a single player and the number of actions that we are provided with is 19.

| | | | |
|---|---|---|---|
| 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 |
| Spring | Stop Moving | Stop Spring | |

*Table 1. Available action set in the Football Environment*

### C. *Rewards*

The engine includes two reward functions, Scoring and Checkpoint.

1. Scoring: +1 for scoring a goal and 01 for conceding a goal. This reward can be hard to observe during the initial stages of training
2. Checkpoint: is a reward by encoding the domain knowledge that scoring is aided by advancing across the pitch. The opponent's field is divided into 10 checkpoint regions according to the Euclidean distance to the opponent goal. This reward is given once per episode.

## IV. Background

### A. *Trust Region Policy Optimization (TRPO)*

TRPO is a scalable algorithm that optimizes policies by gradient descent. Being a policy gradient method, it is model-free and does not require a model of the environment and is more stable practically. While they have a straightforward application to new problems, it's difficult to scale them to large, nonlinear policies [4].
TRPO updates policies using the largest step possible to improve performance, while considering a specific constraint that discusses how close the new and old policies can be, unlike the normal policy gradient which keeps close proximity between the new and old policies. The constraint is expressed in terms of KL-Divergence, which is said to be a measure of distance between probability distributions.
But even seemingly small differences in parameter space can create a large difference in performance, resulting in a single misstep causing the policy to collapse. Hence, it's very risky to use large step sizes along with vanilla policy gradients, affecting its sample efficiency. TRPO prevents this kind of mishap and works towards improving performance quickly and monotonically [5].

### *B. A3C*

A3C is a deviation of Q-Learning, utilizes a policy gradient with a functional approximating in the form of a neural network. This network uses an actor-critic model, with the benefit of asynchronous, allowing for multiple agents to train in parallel [6].

### *C. Double Deep Q Netork (Double DQN)*

Double DQN is a model free, off policy, algorithm that implements Double Q-Learning with Deep Neural Networks. This algorithm is proposed in [2] H. van Hasselt, 2015 and it uses two different neural networks, a DQN and target network to learn and predict what action to take at every step. The network referred to as the Q network, is used to predict what to do when the agent encounters a new state.

All DQN agents learn and improve themselves through experience replay, which is a method where the agent has to remember each step of the game after it happens, each memory consisting of the action taken, the state that was in place when the action was taken, the reward given from taking the action, and the state that resulted from taking the action. For this purpose, the second network, called the target network is used to run the experience replay procedure after each taken step in an episode.

The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation [7]

### *D. Proximal Policy Optimization (PPO)*

PPO is another model free but an on-policy algorithm which can be used in either discrete or continuous environments. This algorithm was introduced in 2017 by OpenAI and is widely being used across a wide range of challenging tasks. PPO is a type of policy gradient training that alternates between sampling data through environmental interaction and optimizing a clipped surrogate objective function using stochastic gradient descent. The clipped surrogate objective function improves training stability by limiting the size of the policy change at each step [8].

## V. Methodology

In our project we trained on the 11 vs 11 easy level scenario. We only control one player at a time, and we have trained four models to complete this task.

The models we have used are: Double DQN, PPO, TRPO and A3C.

### *A. Double DQN*

A baseline model was trained using Double DQN using the PFRL library. This model trained using an epsilon greedy strategy in the default 11 vs 11 games versus the environment default PPO agent for 800000 steps running an evaluation game (episodes) every 3000 training steps, this takes around 9 hours to run using Kaggle GPUs. The rewards used are scoring and checkpoints as described in the environment section. The mean rewards from the validation episodes are presented, the model seems to start doing better around the 150-episode mark, but its scores are very unstable and cannot beat the environment base AI consistently.

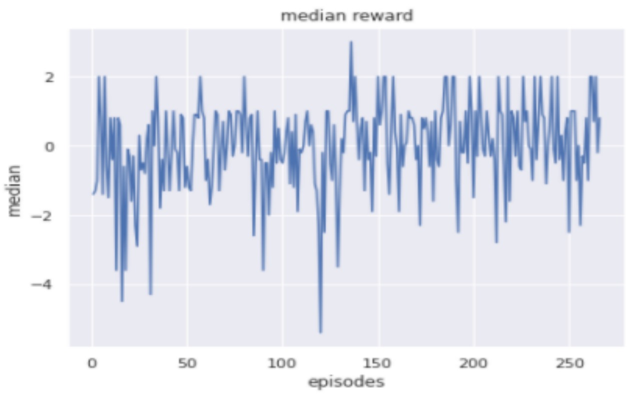

*Figure 2. Median rewards for the Double DQN trained model*

A video of this agent playing against itself is attached to this report where it can be seen that the agent constantly makes passes to its own team and does not try to run towards the opposing goal sometimes causing an own goal with a faulty pass. A reason why the agent might not be learning very well is the very sparse rewards despite the presence of checkpoints rewards. Possible solutions to this problem will be discussed in the planning section.

| **Parameter** | **Value** |
|---|---|
| Gamma | 0.99 |
| Batch size | 32 |
| Replay start size | 20000 |
| Target update interval | 32000 |
| Optimizer | Adam |
| Learning rate | 6.25e-5 |
| eps | 0.00015 |
| Replay buffer size | 200000 |

*Table 2. Hyperparameters used for the Double DQN model*

## *B. PPO*

Another baseline model was trained using a PPO agent on different scenarios provided by the football environment. The architecture that was used, is the same as the one provided in the original "Google research football: A novel reinforcement learning environment" [3] paper.
The same reward system as in the Double DQN model was applied, with negative rewards indicating own goals. This model was trained for 450 episodes which amounts to 1350000 steps with a validation game running every 10 episodes. As seen in the validation scores the agent is not able to beat the default AI of the environment consistently during the training.
A video is attached to this project where it can be seen that the PPO agent performs better than the DQN agent, although it does not appear to have any sense of strategy and just blindly pushes to the opponent zone and does not attempt to score a goal running out of the bounds of the game.

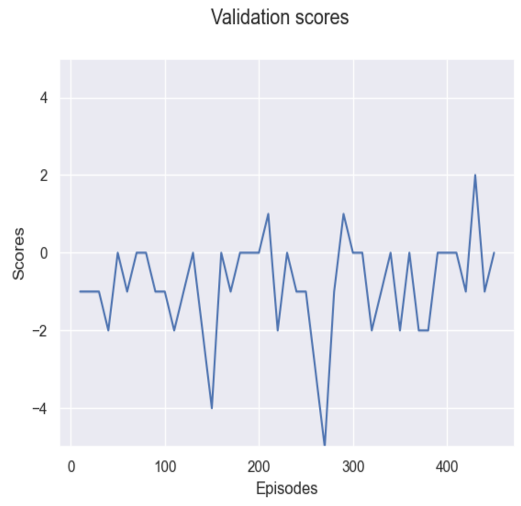

*Figure 3. Validation Scores for the PPO trained model*

## *C. A3C*

To run A3C, a docker container was created based on the Nvidia GPU and Google Football base image. This process was tedious and full of errors. However, after completion the network was able to be trained. There were 64000 time steps.

The designed network for both the actor and the critic had the following layout:

**2D Convolution**
**Batch Norm**
**Relu**
**2D Convolution**
**Batch Norm**
**Relu**
**2D Convolution**
**Batch Norm**
**Relu**
**Linear**
**Linear**

The first three layers theoretically will parse the input image of the environment, then process that information in such a way that it acquires game state information, and then, compute an action to take in the environment. Since this is A3C, it requires both an actor and a critic model for the algorithm, and thus weights must be computed for two models. Unfortunately, the model created had errors, and was unable to learn from the football environment.

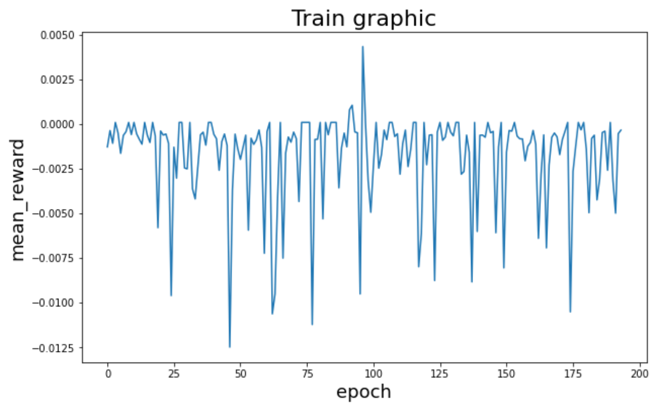

*Figure 4. Mean Rewards for the A3C trained model*

## *D. Trust Region Policy Optimiztation*

In the final baseline model, a TRPO agent was trained with the Reinforcement Learning Replications (rl-replicas) in Pytorch. The model was set up with Policy and ValueFunction. These two core components have their own neural network and optimizer. To run inference, Policy was then initialized with the trained network.
The model trained in the default 11 vs 11 games with the number of epochs and steps varied. The running time ranged from 3 hours to 9 hours in Kaggle GPUs. The same reward system as in the Double DQN model was applied, with negative rewards indicating own goals. All the experiments were done without any hyperparameter tuning, as specified below:

| **Parameter** | **Value** |
|---|---|
| Policy Architecture | [64, 64] |
| Value Function Architecture | [64, 64] |
| Value Function Learning | 1e-3 |
| Optimizer | Conjugate Gradient |

*Table 3. Hyperparameters for the TRPO model*

The average episode rewards are shown as below:

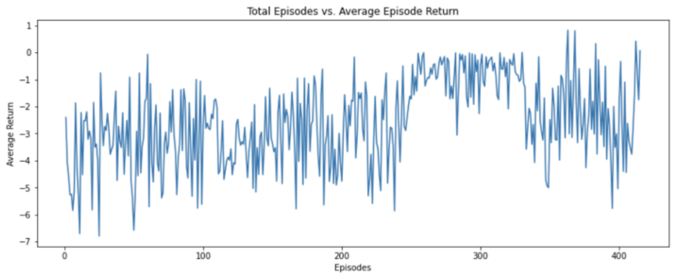

*Figure 5. Average episode rewards for the TRPO model*

The graph is not entirely conclusive but it does show that the average rewards will probably increase with the number of episodes at a later stage. The algorithm also takes more time to converge as compared to others. Also attached is a video of the best case scenario where the agent has managed to score a goal.

| Number of Epochs | Steps per Epoch | Final Score (Goals) |
|---|---|---|
| 5 | 200000 | 1 - 0 |
| 10 | 50000 | 0 - 0 |
| 20 | 50000 | 0 - 0 |
| 10 | 250000 | Timeout |
| 20 | 10000 | 0 - 0 |
| 5 | 250000 | 0 - 0 |

*Table 4. Experiment specifications and rewards for the TRPO model*

## VI. Discussion and Future Approach

The sparse reward caused by using goals as reward can be problematic when training an agent. A possible solution to this is using the environment "academy scenarios" these include scenarios with empty goals, 3 players vs 1 and one keeper, run to score, run to score with keeper and so on. The idea is that by running these environments the agent will be able learn "piecemeal" and later on training on the standard 11 vs 11 scenario. New rewards could be defined to promote the agent to do certain things like giving positive rewards every time it steals the ball from the enemy team and negative rewards when it loses the ball.

Another possible improvement is using a more skilled agent compared to the default PPO AI provided by the environment to train, this would provide more high-quality training to our agent and lead to higher scores. We also believe that the algorithms would've been able to learn more efficiently had they been trained for more episodes which was difficult due to computational constraints.

## VII. Code Documentation

Code Documentation:
To Train A3C:

```
sudo docker build -t ri_final_project .
sudo docker run -it ri_final_project A3C.py
```

All other agents were trained inside Kaggle, their codes can be found on the github.

Code for Project:
**https://github.com/pascalbakker/ReinforcementLearning-Football**

## References

1. Google Research. Google Research Football with Manchester City F.C., 2020, https://www.kaggle.com/c/google-football. Accessed 11 Dec. 2020.
2. Google Research. Google Research Football, 2020, https://github.com/google-research/football. Accessed 11 Dec. 2020.
3. Kurach, Karol, et al. "Google research football: A novel reinforcement learning environment." arXiv preprint arXiv:1907.11180 (2019).
4. Depth First Learning, https://www.depthfirstlearning.com/2018/TRPO.
5. Trust Region Policy Optimization, https://spinningup.openai.com/en/latest/algorithms/trpo.html.
6. Asynchronous Methods for Deep Learning
7. Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." arXiv preprint arXiv:1509.06461 (2015).
8. J Schulman, F Wolski, P Dhariwal, A Radford, O Klimov. "Proximal policy optimization algorithms" arXiv preprint arXiv:1707.06347(2017)