# Train agents to master the world's most popular sport

# Title in hungarian language:
# A világ legnépszerűbb sportját játszó tanuló algoritmus tanítása

Dominguez Zoltán

2020

# Contents

# Abstract

In this semester I am creating a machine learning model with reinforcement learning techniques to play in Google Football environment. Reinforcement Learning models can solve extremely hard problems or win in hard games [1] [1], so this task seems achiavable, but it is still a fair challange.

This paper [2] by Google Research Team describes the environment and provides benchmark results for three stateof-the-art reinforcement learning algorithms. Two of them are popular policy gradient methods (PPO [3], IMPALA [4]) and one is a modern DQN implementation (Ape-X DQN [5]). In this paper I am describing the basic concepts of Reinforcement Learning, mainly focusing on DQN methods. I am using a Rainbow DQN Agent and compare the results to other models.

# Abstract in hungarian

A szemeszterbern egy megerősítéses tanulást használó modellt készítettem, ami a Google Football környezetében képes játszani. Az ilyen modellek extrém nehéz problémákat is képesek megoldani, vagy bonyolult játékokban győzedelmeskedni. Ebből kifolyólag a feladat elérhetőnek tűnik, de kellő kihívással rendelkezik.

A Google Research Team bemutatja a környezetet [2] és teszt eredményeket biztosít három korszerű megerősítéses tanítási algoritmushoz. Két algoritmus ezek közül népszerű Policy optimalizáló algoritmus (PPO, IMPALA), a harmadik pedig egy modern, konvolúciós hálót használó, úgynevezett DQN algoritmus (Ape-X DQN). Én egy Rainbow DQN modellt használok és annak a jelenlegi eredményeit prezentálom.

# Introduction

Reinforcement learning methods have a long past [6], but because they need a lot of computational power, they only made meaningful progress in the recent years.

As football is arguably the world's most famous sport, Google Research Team made a football environment for RL agents to learn in. The Football Engine is out of the box compatible with OpenAI Gym API, so any approaches used on OpenAI Gym projects work in this special problem as well. The environment can render the game, and the game can be played on Ubuntu computers. When training an agent, it is easier and more optimal to use more compressed data format than the visual representation. For example I used the Super Mini Map (SMM) representation which I visualized to get a deeper understanding of the representation.

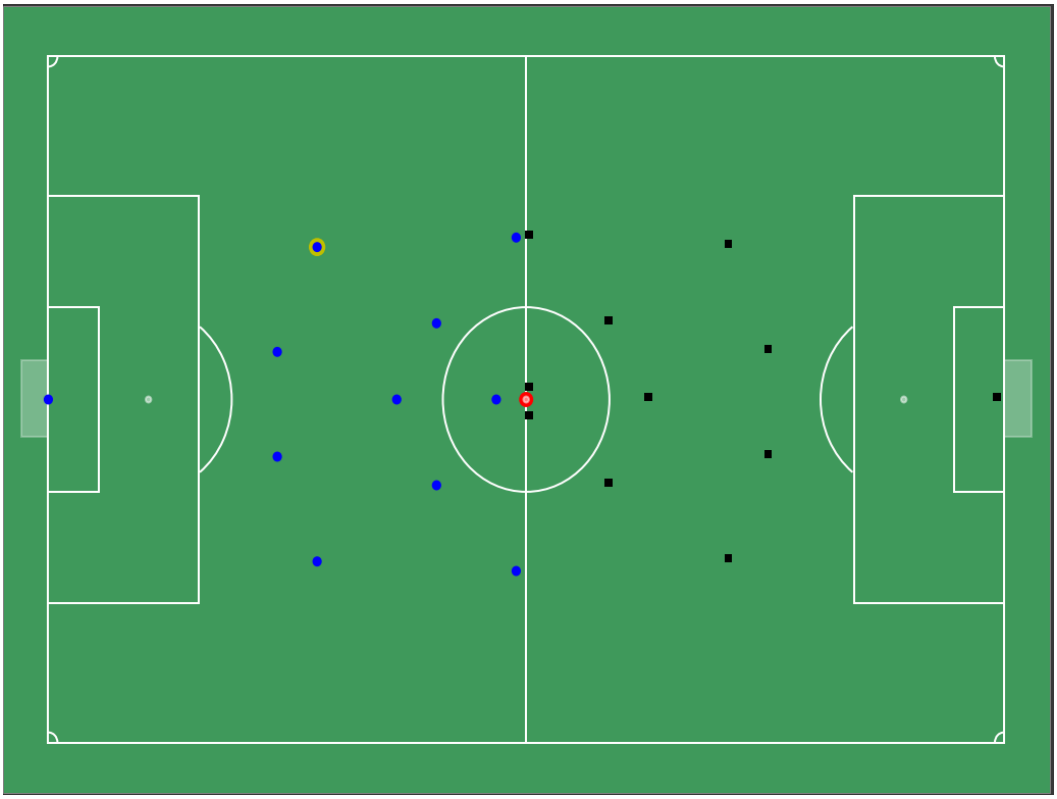


**Figure 1. SMM representation visualized after env.reset()**

The possible 19 actions that an agent can take:

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

**Figure 2. Possible actions**

## 1.1 Reinforcement Learning

A specific type of RL among others is Q learning [7]. Q-learning learns the action-value function Q(s, a) which means how good to take an action at a particular state. For example, for a chess game, how good to move a given pawn two steps forward from the starting position. Literally, we assign a scalar value over the benefit of making such a move.

Managing the Q table for every state, action pair takes a lot of memory and computational power. This is what DQN tries to solve with compressing this q function with a convolutional neural network.

## 1.2 System architecture

The system uses a Rainbow DQN [8] model which consists a Distributional Dueling DQN with some 2d convolutional and linear layers. The current model architecture:

```
DistributionalDuelingDQN(
  (conv_layers): ModuleList(
    (0): Conv2d(4, 32, kernel_size=(8, 8), stride=(4, 4))
    (1): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2))
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  )
  (main_stream): Linear(in_features=3136, out_features=1024, bias=True)
  (a_stream): Linear(in_features=512, out_features=969, bias=True)
  (v_stream): Linear(in_features=512, out_features=51, bias=True)
)
```

I use a Prioritized Replay Buffer [9] with Adam [10] optimizer and a Constant Epsilon Greedy explorer with 0.1 epsilon value. The Epsilon-Greedy explorer makes sure that

during training we explore the states, and do not get stuck in a local optimum. During evaluation I choose actions with Greedy algorithm.

# Implementation

## Data visualization, exploring the environment

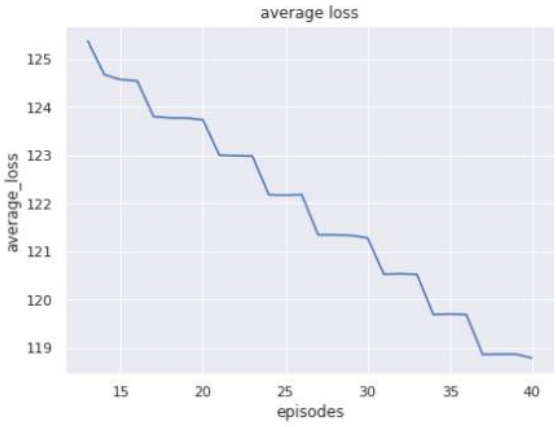
As I already described, I made a gfootball_visu notebook for visualisation and animation purposes. This way a model result and an action can be seen in a human-friendly way. Understanding the representation is also mandatory for a RL task.

## Training

I am training the DQN model on Google Colab with a GPU instance. I trained for multiple hours, which is enough to see that the average and median rewards are going up, although they are making . I am currently far from finding the very best model and weights, but with the saved weights, the training procedure can continue.

## Evaluation

The median rewards and the average loss during training are varied, so I need to make sure the hyperparameters are good before posting final evaluation result. However you can evalute model with the notebook provided. The current evaluation is:


## Testing the model

There was a kaggle competition during 2020 autumn, in which I would like to test my agent against other competitor's agent but that event has already ended. Howewer with the Ubuntu environment installed it is possible to play against the trained agent. Or with the visualization script an animation can be created to see how the agent performs.

# Plans for the future

Training with bigger Replay Buffers (which will need more memory) should increase the accuracy of the model. Training with more computational power and for longer periods of time is a must if we want to tell the limitations of this model.

Automated Hyperparameter optimization is also a plan but it will need a lot of computational power. Manual tweaks of the parameters, convulutional network sizes might also increase the model performance, so it must be inspected later.

Trying different models and inspecting their learning curves is a must, because other type of models(like PPO or IMPALA) are also proved to be performing well in this particular task.

I would like to compare results with Football Benchmarks benchmarking tool which is described in the original Google paper.

Making some tweaks into the reward system could also work well. For example we can reward the agent, if it gets close to the goal line, but not too close. If we use this less sparse reward than the goal score, the agent will learn much faster, that it is good to be close to the enemy goal line. From there it is possible to train it further with only the score reward to actually shot the ball more often.

Trying different RL frameworks and libraries are also a plan for me, but it is for self-educational purposes, doing this will probably not be a research result.

# References


[1] V. M. K. K. D. S. A. G. I. Antonoglou, „Playing Atari with Deep Reinforcement Learning," 2013.

[2] Google Research, Brain Team, „Google Research Football: A Novel Reinforcement Learning Environment".

[3] F. W. P. D. A. R. O. K. John Schulman, „Proximal Policy Optimization Algorithms," 2017.

[4] H. S. e. a. Lasse Espeholt, „IMPALA: Scalable Distributed Deep-RL with Importance Weighted," 2018.

[5] D. H. e. al., „Distributed prioritized experience replay," 2018.

[6] R. S. S. a. A. G. Barto, „Reinforcement learning: An introduction," 1998.

[7] V. M. e. al., „Human-level control through deep reinforcement," 2015.

[8] J. M. H. v. H. T. S. G. O. W. D. D. H. B. P. M. A. D. S. Matteo Hessel, „Rainbow: Combining Improvements in Deep Reinforcement Learning:," 2017.

[9] J. Q. I. A. a. D. S. Tom Schaul, „Prioritized Experience Replay," 2016.

[10] J. B. Diederik P. Kingma, „Adam: A Method for Stochastic Optimization," 2017.