Google Research Football: A Novel Reinforcement Learning Environment

Karol Kurach,* **Anton Raichuk,*** **Piotr Stańczyk,*** **Michał Zając†**
Olivier Bachem Lasse Espeholt Carlos Riquelme Damien Vincent
Marcin Michalski Olivier Bousquet Sylvain Gelly
Google Research, Brain Team

Abstract

Recent progress in the field of reinforcement learning has been accelerated by virtual learning environments such as video games, where novel algorithms and ideas can be quickly tested in a safe and reproducible manner. We introduce the *Google Research Football Environment*, a new reinforcement learning environment where agents are trained to play football in an advanced, physics-based 3D simulator. The resulting environment is challenging, easy to use and customize, and it is available under a permissive open-source license. In addition, it provides support for multiplayer and multi-agent experiments. We propose three full-game scenarios of varying difficulty with the *Football Benchmarks* and report baseline results for three commonly used reinforcement algorithms (IMPALA, PPO, and Ape-X DQN). We also provide a diverse set of simpler scenarios with the *Football Academy* and showcase several promising research directions.

Introduction

The goal of reinforcement learning (RL) is to train smart agents that can interact with their environment and solve complex tasks (Sutton and Barto 2018). Real-world applications include robotics (Haarnoja et al. 2018), self-driving cars (Bansal, Krizhevsky, and Ogale 2018), and control problems such as increasing the power efficiency of data centers (Lazic et al. 2018). Yet, the rapid progress in this field has been fueled by making agents play games such as the iconic Atari console games (Bellemare et al. 2013; Mnih et al. 2013), the ancient game of Go (Silver et al. 2016), or professionally played video games like Dota 2 (OpenAI 2019) or Starcraft II (Vinyals et al. 2017). The reason for this is simple: games provide challenging environments where new algorithms and ideas can be quickly tested in a safe and reproducible manner.

While a variety of reinforcement learning environments exist, they often come with a few drawbacks for research, which we discuss in detail in the next section. For example, they may either be too easy to solve for state-of-the-art algorithms or require access to large amounts of computational resources. At the same time, they may either be (near-)deterministic or there may even be a known model of the environment (such as in Go or Chess). Similarly, many learning environments are inherently single player by only modeling the interaction of an agent with a fixed environment or they focus on a single aspect of reinforcement learning such as continuous control or safety. Finally, learning environments may have restrictive licenses or depend on closed source binaries.

Figure 1: The *Google Research Football Environment* (`github.com/google-research/football`) provides a novel reinforcement learning environment where agents are trained to play football in an advance, physics based 3D simulation.

This highlights the need for a RL environment that is not only challenging from a learning standpoint and customizable in terms of difficulty but also accessible for research both in terms of licensing and in terms of required computational resources. Moreover, such an environment should ideally provide the tools to a variety of current reinforcement learning research topics such as the impact of stochasticity, self-play, multi-agent setups and model-based reinforcement learning, while also requiring smart decisions, tactics, and strategies at multiple levels of abstraction.

*Indicates equal authorship. Correspondence to Karol Kurach (kkurach@google.com).

†Student at Jagiellonian University, work done during internship at Google Brain.

(a) Kickoff (b) Yellow card (c) Corner kick

Figure 2: The *Football Engine* is an advanced football simulator that supports all the major football rules such as (a) kickoffs (b) goals, fouls, cards, (c) corner kicks, penalty kicks, and offside.

Contributions In this paper, we propose the *Google Research Football Environment*, a novel open-source reinforcement learning environment where agents learn to play one of the world's most popular sports: football (a.k.a. soccer). Modeled after popular football video games, the Football Environment provides a physics-based 3D football simulation where agents have to control their players, learn how to pass in between them and how to overcome their opponent's defense in order to score goals. This provides a challenging RL problem as football requires a natural balance between short-term control, learned concepts such as passing, and high level strategy. As our key contributions, we

- provide the *Football Engine*, a highly-optimized game engine that simulates the game of football,
- propose the *Football Benchmarks*, a versatile set of benchmark tasks of varying difficulties that can be used to compare different algorithms,
- propose the *Football Academy*, a set of progressively harder and diverse reinforcement learning scenarios,
- evaluate state-of-the-art algorithms on both the *Football Benchmarks* and the *Football Academy*, providing an extensive set of reference results for future comparison,
- provide a simple API to completely customize and define new football reinforcement learning scenarios, and
- showcase several promising research directions in this environment, *e.g.* the multi-player and multi-agent settings.

Motivation and Other Related Work

There are a variety of reinforcement learning environments that have accelerated research in recent years. However, existing environments exhibit a variety of drawbacks that we address with the *Google Research Football Environment*:

Easy to solve. With the recent progress in RL, many commonly used scenarios can now be solved to a reasonable degree in just a few hours with well-established algorithms. For instance, $\sim$50 commonly used Atari games in the *Arcade Learning Environment* (Bellemare et al. 2013) are routinely solved to super-human level (Hessel et al. 2018). The same applies to the *DeepMind Lab* (Beattie et al. 2016), a navigation-focused maze environment that provides a number of relatively simple tasks with a first person viewpoint.

Computationally expensive. On the other hand, training agents in recent video-game simulators often requires substantial computational resources that may not be available to a large fraction of researchers due to combining hard games, long episodes, and high-dimensional inputs (either in the form of pixels, or hand-crafted representations). For example, the *StarCraft II Learning Environment* (Vinyals et al. 2017) provides an API to *Starcraft II*, a well-known real-time strategy video game, as well as to a few mini-games which are centered around specific tasks in the game.

Lack of stochasticity. The real-world is not deterministic which motivates the need to develop algorithms that can cope with and learn from stochastic environments. Robots, self-driving cars, or data-centers require robust policies that account for uncertain dynamics. Yet, some of the most popular simulated environments – like the Arcade Learning Environment – are deterministic. While techniques have been developed to add artificial randomness to the environment (like skipping a random number of initial frames or using sticky actions), this randomness may still be too structured and easy to predict and incorporate during training (Machado et al. 2018; Hausknecht and Stone 2015). It remains an open question whether modern reinforcement learning approaches such as self-imitation generalize from the deterministic setting to stochastic environments (Guo et al. 2018).

Lack of open-source license. Some advanced physics simulators offer licenses that may be subjected to restrictive use terms (Todorov, Erez, and Tassa 2012). Also, some environments such as StarCraft require access to a closed-source binary. In contrast, open-source licenses enable researchers to inspect the underlying game code and to modify environments if required to test new research ideas.

Known model of the environment. Reinforcement learning algorithms have been successfully applied to board games such as Backgammon (Tesauro 1995), Chess (Hsu 2004), or Go (Silver et al. 2016). Yet, current state-of-the-art algorithms often exploit the fact that the rules of these games (*i.e.*, the model of the environment) are specific, known and can be encoded into the approach. As such, this may make it hard to investigate learning algorithms that should work in environments that can only be explored through interactions.

Single-player. In many available environments such as Atari, one only controls a single agent. However, some modern real-world applications involve a number of agents under either centralized or distributed control. The different agents can either collaborate or compete, creating additional challenges. A well-studied special case is an agent competing against another agent in a zero sum game. In this setting, the opponent can adapt its own strategy, and the agent has to be robust against a variety of opponents. Cooperative multi-agent learning also offers many opportunities and challenges, such as communication between agents, agent behavior specialization, or robustness to the failure of some of the agents. Multiplayer environments with collaborative or competing agents can help foster research around those challenges.

Other football environments. There are other available football simulators, such as the *RoboCup Soccer Simulator* (Kitano et al. 1995; 1997), and the *DeepMind MuJoCo Multi-Agent Soccer Environment* (Liu et al. 2019). In contrast to these environments, the *Google Research Football Environment* focuses on high-level actions instead of low-level control of a physics simulation of robots (such as in the RoboCup Simulation 3D League). Furthermore, it provides many useful settings for reinforcement learning, e.g. the single-agent and multi-agent settings as well as single-player and multiplayer player modes. *Google Research Football* also provides ways to adjust difficulty, both via a strength-adjustable opponent and via diverse and customizable scenarios in Football Academy, and provides several specific features for reinforcement learning research, e.g., OpenAI gym compatibility, different rewards, different representations, and the option to turn on and off stochasticity.

Other related work. Designing rich learning scenarios is challenging, and resulting environments often provide a useful playground for research questions centered around a specific reinforcement learning set of topics. For instance, the *DeepMind Control Suite* (Tassa et al. 2018) focuses on continuous control, the *AI Safety Gridworlds* (Leike et al. 2017) on learning safely, whereas the *Hanabi Learning Environment* (Bard et al. 2019) proposes a multi-agent setup. As a consequence, each of these environments are better suited for testing algorithmic ideas involving a limited but well-defined set of research areas.

Football Engine

The Football Environment is based on the Football Engine, an advanced football simulator built around a heavily customized version of the publicly available *GameplayFootball* simulator (Schuiling 2017). The engine simulates a complete football game, and includes the most common football aspects, such as goals, fouls, corners, penalty kicks, or offsides (see Figure 2 for a few examples).

Supported Football Rules. The engine implements a full football game under standard rules, with 11 players on each team. These include goal kicks, side kicks, corner kicks, both yellow and red cards, offsides, handballs and penalty kicks. The length of the game is measured in terms of the number of frames, and the default duration of a full game is 3000 (10 frames per second for 5 minutes). The length of the game, initial number and position of players can also be edited in customized scenarios (see *Football Academy* below). Players on a team have different statistics[1], such as speed or accuracy and get tired over time.

Opponent AI Built-in Bots. The environment controls the opponent team by means of a rule-based bot, which was provided by the original *GameplayFootball* simulator (Schuiling 2017). The difficulty level θ can be smoothly parameterized between 0 and 1, by speeding up or slowing down the bot reaction time and decision making. Some suggested difficulty levels correspond to: easy ($\theta = 0.05$), medium ($\theta = 0.6$), and hard ($\theta = 0.95$). For self-play, one can replace the opponent bot with any trained model.

Moreover, by default, our non-active players are also controlled by another rule-based bot. In this case, the behavior is simple and corresponds to reasonable football actions and strategies, such as running towards the ball when we are not in possession, or move forward together with our active player. In particular, this type of behavior can be turned off for future research on cooperative multi-agents if desired.

State & Observations. We define as *state* the complete set of data that is returned by the environment after actions are performed. On the other hand, we define as *observation* or *representation* any transformation of the state that is provided as input to the control algorithms. The definition of the state contains information such as the ball position and possession, coordinates of all players, the active player, the game state (tiredness levels of players, yellow cards, score, etc) and the current pixel frame.

We propose three different representations. Two of them (pixels and SMM) can be *stacked* across multiple consecutive time-steps (for instance, to determine the ball direction), or unstacked, that is, corresponding to the current time-step only. Researchers can easily define their own representations based on the environment state by creating wrappers similar to the ones used for the observations below.

Pixels. The representation consists of a 1280×720 RGB image corresponding to the rendered screen. This includes both the scoreboard and a small map in the bottom middle part of the frame from which the position of all players can be inferred in principle.

Super Mini Map. The SMM representation consists of four 72×96 matrices encoding information about the home team, the away team, the ball, and the active player respectively. The encoding is binary, representing whether there is a player or ball in the corresponding coordinate.

Floats. The floats representation provides a compact encoding and consists of a 115-dimensional vector summarizing many aspects of the game, such as players coordinates, ball possession and direction, active player, or game mode.

[1] Although players differ within a team, both teams have exactly the same set of players, to ensure a fair game.

Actions. The actions available to an individual agent (player) are displayed in Table 1. They include standard move actions (in 8 directions), and different ways to kick the ball (short and long passes, shooting, and high passes that can't be easily intercepted along the way). Also, players can sprint (which affects their level of tiredness), try to intercept the ball with a slide tackle or dribble if they posses the ball. We experimented with an action to switch the active player in defense (otherwise, the player with the ball must be active). However, we observed that policies tended to exploit this action to return control to built-in AI behaviors for non-active players, and we decided to remove it from the action set. We do *not* implement randomized sticky actions. Instead, once executed, moving and sprinting actions are sticky and continue until an explicit stop action is performed (Stop-Moving and Stop-Sprint respectively).

Rewards. The *Football Engine* includes two reward functions that can be used out-of-the-box: SCORING and CHECKPOINT. It also allows researchers to add custom reward functions using wrappers which can be used to investigate reward shaping approaches.

SCORING corresponds to the natural reward where each team obtains a $+1$ reward when scoring a goal, and a -1 reward when conceding one to the opposing team. The SCORING reward can be hard to observe during the initial stages of training, as it may require a long sequence of consecutive events: overcoming the defense of a potentially strong opponent, and scoring against a keeper.

CHECKPOINT is a (shaped) reward that specifically addresses the sparsity of SCORING by encoding the domain knowledge that scoring is aided by advancing across the pitch: It augments the SCORING reward with an additional auxiliary reward contribution for moving the ball close to the opponent's goal in a controlled fashion. More specifically, we divide the opponent's field in 10 checkpoint regions according to the Euclidean distance to the opponent goal. Then, the first time the agent's team possesses the ball in each of the checkpoint regions, the agent obtains an additional reward of $+0.1$. This extra reward can be up to $+1$, *i.e.*, the same as scoring a single goal. Any non-collected checkpoint reward is also added when scoring in order to avoid penalizing agents that do not go through all the checkpoints before scoring (*i.e.*, by shooting from outside a checkpoint region). Finally, checkpoint rewards are only given once per episode.

Accessibility. Researchers can directly inspect the game by playing against each other or by dueling their agents. The game can be controlled by means of both keyboards and gamepads. Moreover, replays of several rendering qualities can be automatically stored while training, so that it is easy to inspect the policies agents are learning.

Stochasticity. In order to investigate the impact of randomness, and to simplify the tasks when desired, the environment can run in either stochastic or deterministic mode. The former, which is enabled by default, introduces several types of randomness: for instance, the same shot from the top of the box may lead to a different number of outcomes. In the latter, playing a fixed policy against a fixed opponent always results in the same sequence of actions and states.

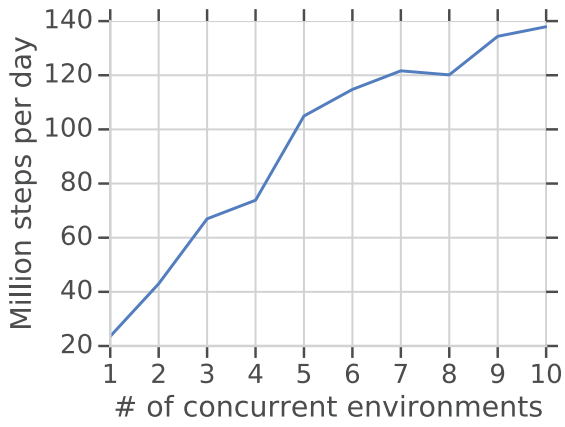

Figure 3: Number of steps per day versus number of concurrent environments for the *Football Engine* on a hexa-core Intel Xeon W-2135 CPU with 3.70GHz.

API & Sample Usage. The Football Engine is out of the box compatible with the widely used OpenAI Gym API (Brockman et al. 2016). Below we show example code that runs a random agent on our environment.

```
import gfootball.env as football_env

env = football_env.create_environment(
    env_name='11_vs_11_stochastic',
    render=True)
env.reset()
done = False
while not done:
    action = env.action_space.sample()
    observation, reward, done, info = \
        env.step(action)
```

Technical Implementation & Performance. The Football Engine is written in highly optimized C++ code, allowing it to be run on commodity machines both with GPU and without GPU-based rendering enabled. This allows it to obtain a performance of approximately 140 million steps per day on a single hexacore machine (see Figure 3).

Table 1: Action Set

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

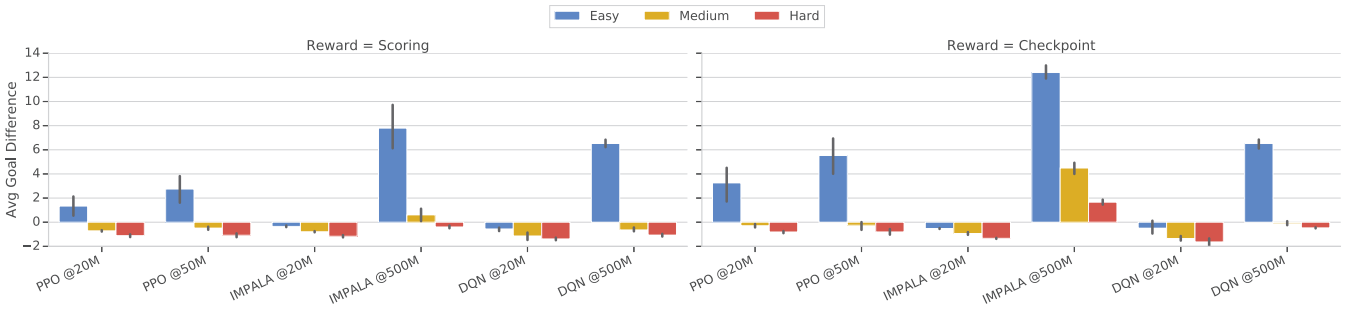

Figure 4: Average Goal Difference on the *Football Benchmarks* for IMPALA, PPO and Ape-X DQN with both the SCORING and CHECKPOINT rewards. Error bars represent 95% bootstrapped confidence intervals. Results for GRF v1.x

Football Benchmarks

The *Football Engine* is an efficient, flexible and highly customizable learning environment with many features that lets researchers try a broad range of new ideas. To facilitate fair comparisons of different algorithms and approaches in this environment, we also provide a set of pre-defined benchmark tasks that we call the *Football Benchmarks*. Similar to the Atari games in the *Arcade Learning Environment*, in these tasks, the agent has to interact with a fixed environment and maximize its episodic reward by sequentially choosing suitable actions based on observations of the environment.

The goal in the *Football Benchmarks* is to win a full game[2] against the opponent bot provided by the engine. We provide three versions of the *Football Benchmarks* that only differ in the strength of the opponent AI as described in the last section: the easy, medium, and hard benchmarks. This allows researcher to test a wide range of research ideas under different computational constraints such as single machine setups or powerful distributed settings. We expect that these benchmark tasks will be useful for investigating current scientific challenges in reinforcement learning such as sample-efficiency, sparse rewards, or model-based approaches.

Experimental Setup

As a reference, we provide benchmark results for three state-of-the-art reinforcement learning algorithms: PPO (Schulman et al. 2017) and IMPALA (Espeholt et al. 2018) which are popular policy gradient methods, and Ape-X DQN (Horgan et al. 2018), which is a modern DQN implementation. We run PPO in multiple processes on a single machine, while IMPALA and DQN are run on a distributed cluster with 500 and 150 actors respectively.

In all benchmark experiments, we use the stacked Super Mini Map representation and the same network architecture. We consider both the SCORING and CHECKPOINT rewards. The tuning of hyper-parameters is done using easy scenario, and we follow the same protocol for all algorithms to ensure fairness of comparison. After tuning, for each of the six considered settings (three *Football Benchmarks* and two reward functions), we run five random seeds and average the results.

[2]We define an 11 versus 11 full game to correspond to 3000 steps in the environment, which amounts to 300 seconds if rendered at a speed of 10 frames per second.

For the technical details of the training setup and the used architecture and hyperparameters, we refer to the Appendix.

Results

The experimental results[3] for the *Football Benchmarks* are shown in Figure 4. It can be seen that the environment difficulty significantly affects the training complexity and the average goal difference. While the easy benchmark can be solved by all methods relatively quickly, the medium and especially hard benchmarks are significantly more difficult. The medium benchmark can be beaten by IMPALA with 500M training steps (albeit only barely with the SCORING reward) while PPO and Ape-X DQN in the considered setting do not appear to achieve a positive reward. The hard benchmark is even harder with only IMPALA with the CHECKPOINT reward and 500M training steps achieving a positive score. We observe that the CHECKPOINT reward function appears to be helpful for speeding up the training for policy gradient methods but does not seem to benefit Ape-X DQN as the performance is similar with both the CHECKPOINT and SCORING reward functions. We conclude that the *Football Benchmarks* provide interesting reference problems for research and that there remains a large headroom for progress, in particular in terms of performance and sample efficiency on the harder benchmarks.

Football Academy

Training agents for the *Football Benchmarks* can be challenging. To allow researchers to quickly iterate on new research ideas, we also provide the *Football Academy*: a diverse set of scenarios of varying difficulty. These 11 scenarios (see Figure 5 for a selection) include several variations where a single player has to score against an empty goal (*Empty Goal Close*, *Empty Goal*, *Run to Score*), a number of setups where the controlled team has to break a specific defensive line formation (*Run to Score with Keeper*, *Pass and Shoot with Keeper*, *3 vs 1 with Keeper*, *Run, Pass and Shoot with Keeper*) as well as situations commonly found in football games (*Corner*, *Easy Counter-Attack*, *Hard Counter-Attack*). Using a simple API, researchers can also easily define their own scenarios and train agents to solve them.

[3]All results in this paper are for the versions v1.x of the GRF.

(a) Empty Goal Close (b) Run to Score (c) 11 vs 11 with Lazy Opponents

(d) 3 vs 1 with Keeper (e) Pass and Shoot (f) Easy Counter-attack

Figure 5: Example of *Football Academy* scenarios.

Experimental Results

Based on the same experimental setup as for the *Football Benchmarks*, we provide experimental results for SCORING reward[4] for both PPO and IMPALA for the *Football Academy* scenarios in Figures 6 and 7. We note that the maximum average scoring performance is 1 (as episodes end in the *Football Academy* scenarios after scoring) and that scores may be negative as agents may score own goals and as the opposing team can score in the *Corner* scenario.

The experimental results indicate that the *Football Academy* provides a set of diverse scenarios of different difficulties suitable for different computational constraints. The scenarios where agents have to score against the empty goal (*Empty Goal Close*, *Empty Goal*, *Run to Score*) appear to be very easy and can be solved both PPO and IMPALA with both reward functions using only 1M steps. As such, these scenarios can be considered "unit tests" for reinforcement learning algorithms where one can obtain reasonable results within minutes or hours instead of days or even weeks. The remainder of the tasks includes scenarios for which both PPO and IMPALA appear to require between 5M to 50M steps for progress to occur (with minor differences between the SCORING and CHECKPOINT) rewards). These harder tasks may be used to quickly iterate on new research ideas on single machines before applying them to the *Football Benchmarks* (as experiments should finish within hours or days). Finally, the CORNER appears to be the hardest scenario (presumably as one has to face a full squad and the opponent is also allowed to score).

Promising Research Directions

In this section we briefly discuss a few initial experiments related to three research topics which have recently become quite active in the reinforcement learning community: self-play training, multi-agent learning, and representation learning for downstream tasks. This highlights the research potential and flexibility of the Football Environment.

[4]For CHECKPOINT reward results, please see the Appendix in https://arxiv.org/abs/1907.11180

Multiplayer Experiments

The Football Environment provides a way to train against different opponents, such as built-in AI or other trained agents. Note this allows, for instance, for self-play schemes. When a policy is trained against a fixed opponent, it may exploit its particular weaknesses and, thus, it may not generalize well to other adversaries. We conducted an experiment to showcase this in which a first model A was trained against a built-in AI agent on the standard 11 vs 11 medium scenario. Then, another agent B was trained against a frozen version of agent A on the same scenario. While B managed to beat A consistently, its performance against built-in AI was poor. The numerical results showing this lack of transitivity across the agents are presented in Table 2.

Table 2: Average goal difference $\pm$ one standard deviation across 5 repetitions of the experiment.

A vs built-in AI	4.25 ± 1.72
B vs A	11.93 ± 2.19
B vs built-in AI	-0.27 ± 0.33

Multi-Agent Experiments

The environment also allows for controlling several players from one team simultaneously, as in multi-agent reinforcement learning. We conducted experiments in this setup with the *3 versus 1 with Keeper* scenario from Football Academy. We varied the number of players that the policy controls from 1 to 3, and trained with Impala.

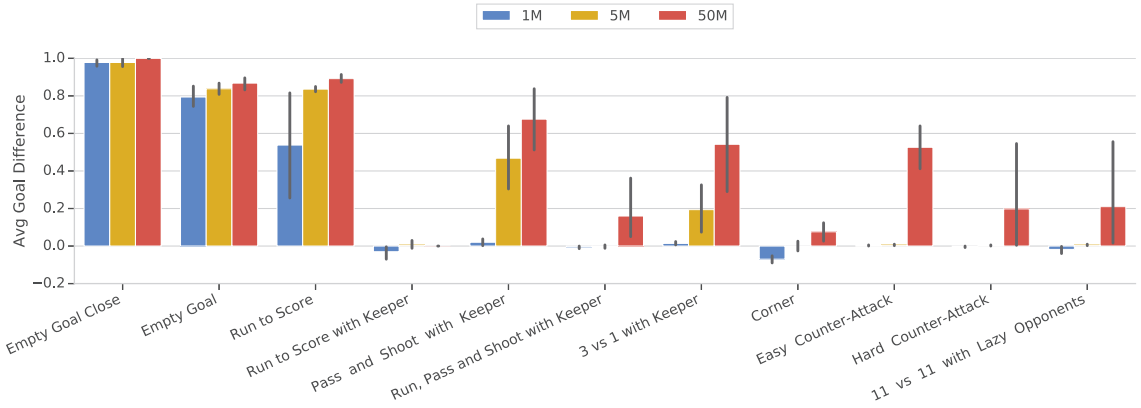

Figure 6: Average Goal Difference on *Football Academy* for IMPALA with SCORING reward.

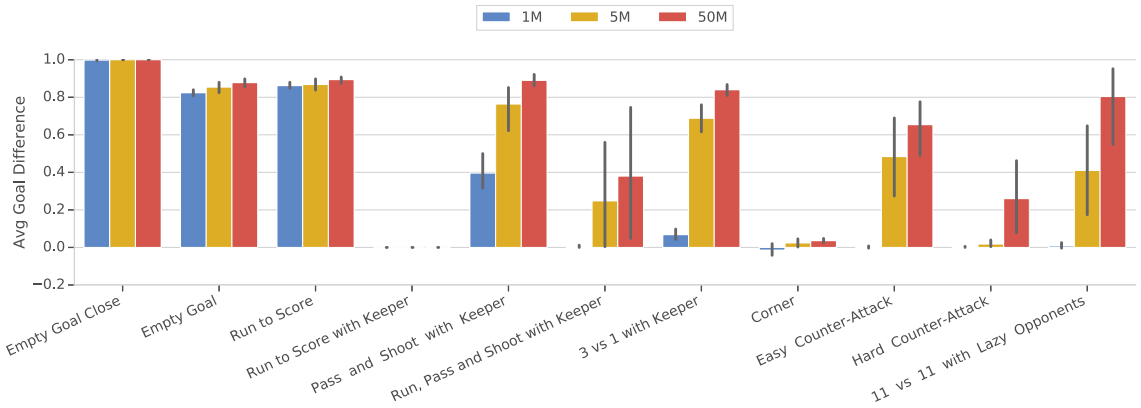

Figure 7: Average Goal Difference on *Football Academy* for PPO with SCORING reward.

Table 3: Scores achieved by the policy controlling 1, 2 or 3 players respectively, after 5M and 50M steps of training.

Players controlled	**5M steps**	**50M steps**
1	0.73 ± 0.06	0.79 ± 0.05
2	0.04 ± 0.04	0.90 ± 0.04
3	0.02 ± 0.02	0.93 ± 0.05

Table 4: Average goal advantages per representation.

Representation	**100M steps**	**500M steps**
Floats	1.56 ± 0.24	5.31 ± 2.59
Pixels gray	-0.43 ± 0.09	8.64 ± 0.62
SMM	7.57 ± 1.06	10.04 ± 1.43
SMM stacked	6.84 ± 1.02	12.19 ± 1.12

As expected, training is initially slower when we control more players, but the policies seem to eventually learn more complex behaviors and achieve higher scores. Numerical results are presented in Table 3.

Representation Experiments

Training the agent directly from raw observations, such as pixels, is an exciting research direction. While it was successfully done for Atari, it is still an open challenge for most of the more complex and realistic environments. In this experiment, we compare several representations available in the *Football Engine*. *Pixels gray* denotes the raw pixels from the game, which are resized to 72×96 resolution and converted to grayscale. While pixel representation takes significantly longer time to train, as shown in Table 4, learning eventually takes place (and it actually outperforms hand-picked extensive representations like 'Floats'). The results were obtained using Impala with Checkpoint reward on the easy 11 vs. 11 benchmark.

Conclusions

In this paper, we presented the *Google Research Football Environment*, a novel open-source reinforcement learning environment for the game of football. It is challenging and accessible, easy to customize, and it has specific functionality geared towards research in reinforcement learning. We provided the *Football Engine*, a highly optimized C++ football simulator, the *Football Benchmarks*, a set of reference tasks to compare different reinforcement learning algorithms, and the *Football Academy*, a set of progressively

harder scenarios. We expect that these components will be useful for investigating current scientific challenges like self-play, sample-efficient RL, sparse rewards, and model-based RL.

Acknowledgement

We wish to thank Lucas Beyer, Nal Kalchbrenner, Tim Salimans and the rest of the Google Brain team for helpful discussions, comments, technical help and code contributions. We would also like to thank Bastiaan Konings Schuiling, who authored and open-sourced the original version of this game.

Hyperparameters & Architectures

For our experiments, we used three algorithms (IMPALA, PPO, Ape-X DQN) that are described below. The model architecture we use is inspired by Large architecture from (Espeholt et al. 2018) and is depicted in Figure 8. Based on the "Representation Experiments", we selected the stacked Super Mini Map as the default representation used in all *Football Benchmarks* and *Football Academy* experiments. In addition we have three other representations.

For each of the six considered settings (three *Football Benchmarks* and two reward functions), we run five random seeds for 500 million steps each. For *Football Academy*, we run five random seeds in all 11 scenarios for 50 million steps.

Hyperparameter search For each of IMPALA, PPO and Ape-X DQN, we performed two hyperparameter searches: one for SCORING reward and one for CHECKPOINT reward. For the search, we trained on easy difficulty. Each of 100 parameter sets was repeated with 3 random seeds. For each algorithm and reward type, the best parameter set was decided based on average performance – for IMPALA and Ape-X DQN after 500M, for PPO after 50M. After the search, each of the best parameter sets was used to run experiments with 5 different random seeds on all scenarios. Ranges that we used for the procedure can be found in Table 7 for IMPALA, Table 8 for PPO and Table 9 for DQN.

IMPALA Importance Weighted Actor-Learner Architecture (Espeholt et al. 2018) is a highly scalable algorithm that decouples acting from learning. Individual workers communicate trajectories of experience to the central learner, instead of sending gradients with respect to the current policy. In order to deal with off-policy data, IMPALA introduces an actor-critic update for the learner called V-trace. Hyperparameters for IMPALA are presented in Table 7.

PPO Proximal Policy Optimization (Schulman et al. 2017) is an online policy gradient algorithm which optimizes the clipped surrogate objective. In our experiments we use the implementation from the OpenAI Baselines (Dhariwal et al. 2017), and run it over 16 parallel workers. Hyperparameters for PPO are presented in Table 8.

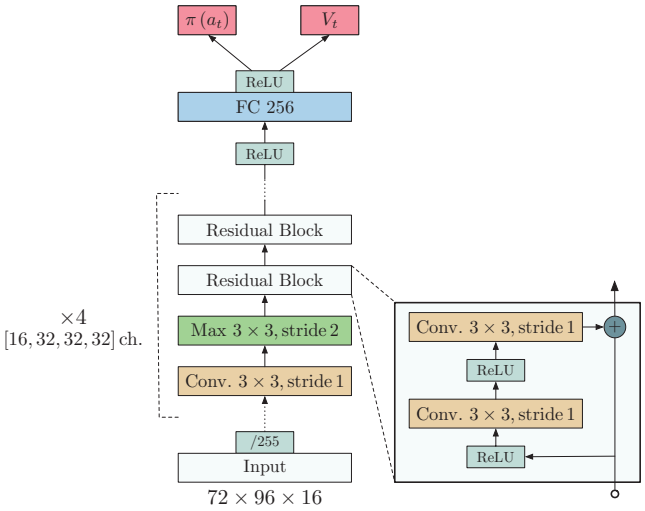

Figure 8: Architecture used for IMPALA and PPO experiments. For Ape-X DQN, a similar network is used but the outputs are Q-values.

Ape-X DQN Q-learning algorithms are popular among reinforcement learning researchers. Accordingly, we include a member of the DQN family in our comparison. In particular, we chose Ape-X DQN (Horgan et al. 2018), a highly scalable version of DQN. Like IMPALA, Ape-X DQN decouples acting from learning but, contrary to IMPALA, it uses a distributed replay buffer and a variant of Q-learning consisting of dueling network architectures (Wang et al. 2016) and double Q-learning (Van Hasselt, Guez, and Silver 2016).

Several hyper-parameters were aligned with IMPALA. These includes unroll length and n-step return, the number of actors and the discount factor γ. For details, please refer to the Table 9.

Table 5: Benchmark results for SCORING reward.

MODEL	EASY	MEDIUM	HARD
PPO @20M	1.34 ± 0.94	-0.71 ± 0.07	-1.11 ± 0.13
PPO @50M	2.75 ± 1.31	-0.49 ± 0.14	-1.08 ± 0.19
IMPALA @20M	-0.35 ± 0.08	-0.79 ± 0.06	-1.16 ± 0.11
IMPALA @500M	7.79 ± 2.05	0.61 ± 0.58	-0.39 ± 0.12
DQN @20M	-0.56 ± 0.17	-1.14 ± 0.34	-1.39 ± 0.13
DQN @500M	6.52 ± 0.37	-0.64 ± 0.21	-1.07 ± 0.14

Table 6: Benchmark results for CHECKPOINT reward.

MODEL	EASY	MEDIUM	HARD
PPO @20M	3.26 ± 1.57	-0.29 ± 0.17	-0.81 ± 0.11
PPO @50M	5.53 ± 1.82	-0.25 ± 0.37	-0.81 ± 0.28
IMPALA @20M	-0.52 ± 0.05	-0.94 ± 0.13	-1.34 ± 0.05
IMPALA @500M	12.40 ± 0.63	4.49 ± 0.51	1.66 ± 0.24
DQN @20M	-0.49 ± 0.66	-1.35 ± 0.22	-1.63 ± 0.34
DQN @500M	6.52 ± 0.45	-0.09 ± 0.19	-0.47 ± 0.05

Table 7: IMPALA: ranges used during the hyper-parameter search and the final values used for experiments with scoring and checkpoint rewards.

Parameter	Range	Best - Scoring	Best - Checkpoint
Action Repetitions	1	1	1
Batch size	128	128	128
Discount Factor (γ)	$\{.99, .993, .997, .999\}$	.993	.993
Entropy Coefficient	Log-uniform $(1e{-6}, 1e{-3})$	0.00000521	0.00087453
Learning Rate	Log-uniform $(1e{-5}, 1e{-3})$	0.00013730	0.00019896
Number of Actors	500	500	500
Optimizer	Adam	Adam	Adam
Unroll Length/n-step	$\{16, 32, 64\}$	32	32
Value Function Coefficient	.5	.5	.5

Table 8: PPO: ranges used during the hyper-parameter search and the final values used for experiments with scoring and checkpoint rewards.

Parameter	Range	Best - Scoring	Best - Checkpoint
Action Repetitions	1	1	1
Clipping Range	Log-uniform $(.01, 1)$	.115	.08
Discount Factor (γ)	$\{.99, .993, .997, .999\}$	.997	.993
Entropy Coefficient	Log-uniform $(.001, .1)$	.00155	.003
GAE (λ)	.95	.95	.95
Gradient Norm Clipping	Log-uniform $(.2, 2)$	.76	.64
Learning Rate	Log-uniform $(.000025, .0025)$	.00011879	.000343
Number of Actors	16	16	16
Optimizer	Adam	Adam	Adam
Training Epochs per Update	$\{2, 4, 8\}$	2	2
Training Mini-batches per Update	$\{2, 4, 8\}$	4	8
Unroll Length/n-step	$\{16, 32, 64, 128, 256, 512\}$	512	512
Value Function Coefficient	.5	.5	.5

Table 9: DQN: ranges used during the hyper-parameter search and the final values used for experiments with scoring and checkpoint rewards.

Parameter	Range	Best - Scoring	Best - Checkpoint
Action Repetitions	1	1	1
Batch Size	512	512	512
Discount Factor (γ)	$\{.99, .993, .997, .999\}$	.999	.999
Evaluation ϵ	.01	.01	.01
Importance Sampling Exponent	$\{0., .4, .5, .6, .8, 1.\}$	1.	1.
Learning Rate	Log-uniform $(1e{-7}, 1e{-3})$	.00001475	.0000115
Number of Actors	150	150	150
Optimizer	Adam	Adam	Adam
Replay Priority Exponent	$\{0., .4, .5, .6, .7, .8\}$	.0	.8
Target Network Update Period	2500	2500	2500
Unroll Length/n-step	$\{16, 32, 64, 128, 256, 512\}$	16	16

Numerical Results for the *Football Benchmarks*

In this section we provide for comparison the means and std values of 5 runs for all algorithms in *Football Benchmarks*. Table 5 contains the results for the runs with SCORING reward while Table 6 contains the results for the runs with CHECKPOINT reward. Those numbers were presented in the main paper in Figure 4.

References

Bansal, M.; Krizhevsky, A.; and Ogale, A. 2018. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst.

Bard, N.; Foerster, J. N.; Chandar, S.; Burch, N.; Lanctot,

M.; Song, H. F.; Parisotto, E.; Dumoulin, V.; Moitra, S.; Hughes, E.; Dunning, I.; Mourad, S.; Larochelle, H.; Bellemare, M. G.; and Bowling, M. 2019. The hanabi challenge: A new frontier for ai research.

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; Schrittwieser, J.; Anderson, K.; York, S.; Cant, M.; Cain, A.; Bolton, A.; Gaffney, S.; King, H.; Hassabis, D.; Legg, S.; and Petersen, S. 2016. Deepmind lab.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

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

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. Openai baselines. https://github.com/openai/baselines.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 1406–1415.

Guo, Y.; Oh, J.; Singh, S.; and Lee, H. 2018. Generative adversarial self-imitation learning. *arXiv preprint arXiv:1812.00950*.

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic algorithms and applications.

Hausknecht, M., and Stone, P. 2015. The impact of determinism on learning atari 2600 games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; van Hasselt, H. P.; and Silver, D. 2018. Distributed prioritized experience replay. *CoRR* abs/1803.00933.

Hsu, F.-H. 2004. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1995. Robocup: The robot world cup initiative.

Kitano, H.; Tambe, M.; Stone, P.; Veloso, M.; Coradeschi, S.; Osawa, E.; Matsubara, H.; Noda, I.; and Asada, M. 1997. The robocup synthetic agent challenge 97. In *Robot Soccer World Cup*, 62–73. Springer.

Lazic, N.; Boutilier, C.; Lu, T.; Wong, E.; Roy, B.; Ryu, M.; and Imwalle, G. 2018. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, 3814–3823.

Leike, J.; Martic, M.; Krakovna, V.; Ortega, P. A.; Everitt, T.; Lefrancq, A.; Orseau, L.; and Legg, S. 2017. Ai safety gridworlds.

Liu, S.; Lever, G.; Merel, J.; Tunyasuvunakool, S.; Heess, N.; and Graepel, T. 2019. Emergent coordination through competition.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* 61:523–562.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

OpenAI. 2019. OpenAI Five.

Schuiling, B. K. 2017. GameplayFootball. `github.com/BazkieBumpercar/GameplayFootball`.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tassa, Y.; Doron, Y.; Muldal, A.; Erez, T.; Li, Y.; de Las Casas, D.; Budden, D.; Abdolmaleki, A.; Merel, J.; Lefrancq, A.; Lillicrap, T.; and Riedmiller, M. 2018. Deepmind control suite.

Tesauro, G. 1995. Temporal difference learning and td-gammon.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A. S.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J.; Quan, J.; Gaffney, S.; Petersen, S.; Simonyan, K.; Schaul, T.; van Hasselt, H.; Silver, D.; Lillicrap, T.; Calderone, K.; Keet, P.; Brunasso, A.; Lawrence, D.; Ekermo, A.; Repp, J.; and Tsing, R. 2017. Starcraft ii: A new challenge for reinforcement learning.

Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; and Freitas, N. 2016. Dueling network architectures for deep reinforcement learning. In Balcan, M. F., and Weinberger, K. Q., eds., *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, 1995–2003. New York, New York, USA: PMLR.