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. We further 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), and we also provide a diverse set of simpler scenarios with the *Football Academy*.

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) 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 available learning environments exist, they often come with a few drawbacks for research:

Easy to solve. Reinforcement learning algorithms and their infrastructure scale have improved significantly in the last few years. As a result, most synthetic scenarios can be solved to a reasonable degree in just a few hours with well-established algorithms. For instance, the *Arcade Learning Environment* (Bellemare et al. 2013) provides access to a broad selection of 200 Atari games where about 50 of them are commonly used for research, and most are routinely solved to super-human level. The same applies to the *DeepMind Lab* (Beattie et al. 2016), a navigation-focused environment that provides a number of relatively simple tasks in mazes with a first person viewpoint. In addition, in many cases it is not possible to smoothly tune the difficulty level of the task.

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 advanced, physics-based 3D simulation.

Lack of stochasticity. The real-world is far from deterministic and, accordingly, we 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. Unfortunately, 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, using sticky actions or imposing soft-Q or ϵ-greedy policies), this randomness may still be too structured and

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

easy to predict and incorporate during training (Machado et al. 2018; Hausknecht and Stone 2015). Understanding the impact of different sources of stochasticity in reinforcement learning reproducibility is an active open question (Nagarajan, Warnell, and Stone 2018).

Computationally expensive. Some of the most recent video-game simulators combine very long episodes with 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 a specific task in the game. Unfortunately, training algorithms in such settings requires substantial computational resources that may not be available to a large fraction of researchers.

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 only provide access to the closed-source binary. On the other hand, in some cases, researchers may want to edit and modify the environment to test specific research ideas.

Known model of the environment. Learning algorithms have been fairly successful in playing board games such as Backgammon (Tesauro 1995), Chess (Hsu 2004), or Go (Silver et al. 2016). In those cases, the environment model is often assumed to be known. While an algorithm designer may decide to assume otherwise, strong known benchmarks in these games will discourage this approach. Thus, promising research directions like model-based reinforcement learning and world-models may be hard to investigate in these setups.

Single-player. In many available scenarios, we can only control one agent in the environment. The other agents, if any, have a fixed strategy and are thus an integral part of the environment. Examples of single-agent scenario include Atari and DeepMind Lab. 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 must then be robust against a variety of opponents. Self-play is a promising technique to bootstrap the learning of such an agent and ideally reach the Nash equilibrium of the zero sum game. Cooperative multi-agent learning also offers many opportunities and challenges, including but not limited to communication between agents, agent behavior specialization, or robustness to the failure of some of the agents. Multi-player environments with collaborative or competing agents (or a mixture of both) can help foster research around those challenges. In particular, football exhibits and exposes many of the aforementioned aspects.

Focused on single RL aspects. 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.

This highlights the need for a simulated environment that is challenging from a learning standpoint, customizable, while still accessible for research both in terms of licensing and in terms of required computational resources. Moreover, it should ideally provide the tools to tackle relevant modern research topics such as the impact of stochasticity, self-play, multi-agent setups and model-based reinforcement learning, while allowing for smart decisions, tactics, and strategies at multiple levels of abstraction.

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 reinforcement learning 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, and
- provide a simple API to completely customize and define new football reinforcement learning scenarios.

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 it accepts input actions from both teams. The game includes the most common football aspects, such as goals, fouls, corners, penalty kicks, or offsides (see Figure 2 for a few examples). There are also 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 ours, these environments are more tailored towards continuous control and based on low-level physical

(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.

actions. We now describe in detail some of the salient properties of the engine.

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, penalty kicks, and substitutions. The length of the game is measured in terms of the number of frames, and the default duration of a full game is 3000 such frames, while this can be modified in customized scenarios. The initial number and position of players can also be edited in customized scenarios (see the Academy section below). Players on both teams get tired over time, and each team can perform at most three subs.

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, we can replace the opponent bot with any of our trained models.

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. Note the identity could in principle be one such transformation. 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. More importantly, 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. Note 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, in principle, be inferred.

Super Mini Map. The SMM representation consists of four 96×72 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, ball, or active player in the corresponding coordinate, or not.

Floats. The floats representation provides a more compact encoding. It 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.

Actions. The actions available to an individual agent (player) are displayed in Table 1. They include standard move actions (up, down, left, right), 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).

Accessibility. Researchers can also directly experience how the game works 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.

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 25 million steps per day on a single hexacore machine (see Figure 3).

Football Benchmarks

The *Football Engine* is an efficient, flexible and highly customizable learning environment with many features that let researchers try new ideas. However, in order to facilitate a fair comparison of different algorithms and approaches, 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.

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

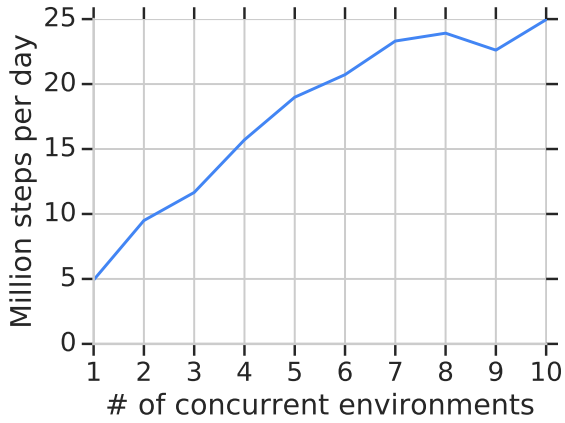

Figure 3: Number of steps per day versus number of concurrent environments for the *Football Engine* on a hexa-core Intel Xeon E5-1650 v2 CPU with 3.50GHz.

The goal of the *Football Benchmarks* is to win a full game[1] against the opponent bot provided by the engine. As discussed in the previous section, we set the difficulty level to provide three versions: the easy, medium, and hard benchmarks. These scenarios only differ in the strength of the opponent AI. On one hand, by having several difficulty levels we hope to allow the testing of a wide range of research ideas under different computational constraints such as single machine setups or powerful distributed settings. On the other, the *Football Benchmarks* have the advantage that all of them aim to solve the same learning task, which makes performance comparison meaningful. 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: proximal policy optimization (Schulman et al. 2017) which runs multi-processes on a single machine, IMPALA (Espeholt et al. 2018) which we run in a distributed cluster of 500 actors, and Ape-X DQN (Horgan et al. 2018), which is a modern DQN implementation. We report results for IMPALA and Ape-X DQN in this section, and for IMPALA and PPO in the next section, where we focus on the *Football Academy*.

We now provide more details about the three algorithms.

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.

[1]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.

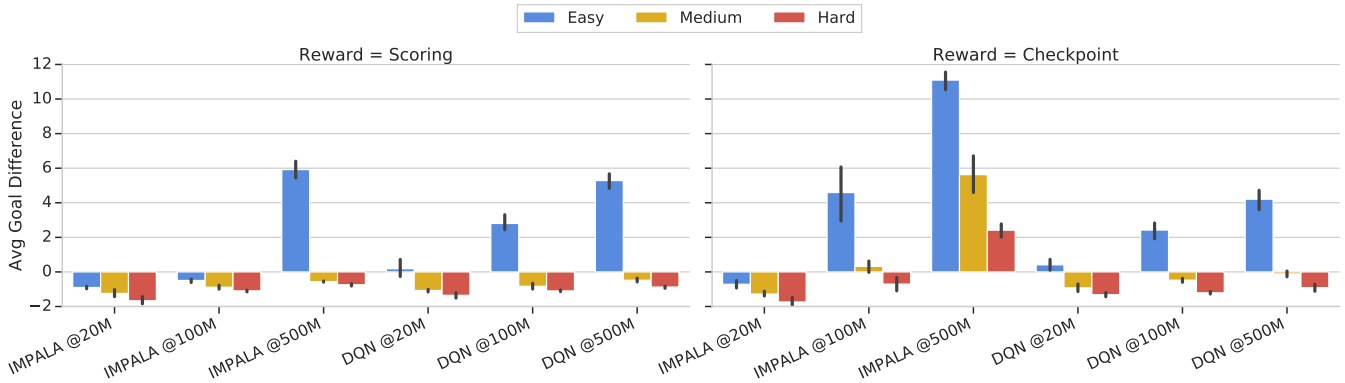

Figure 4: Average Goal Difference on *Football Benchmarks* for IMPALA and DQN with SCORING and CHECKPOINT rewards.

In order to deal with off-policy data, IMPALA introduces an actor-critic update for the learner called V-trace.

In our experiments we used 500 actors. Parameters were optimized using the Adam optimizer (Kingma and Ba 2015) over 500 million steps. For details on hyper-parameters and architectures, please refer to the appendix.

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. We use the convolutional neural network architecture proposed in (Mnih et al. 2015), as implemented in the OpenAI Baselines. Also, we rely on the Adam optimizer for training. Details on hyper-parameters are deferred to the appendix.

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. Details on hyper-parameters are deferred to the appendix.

Rewards Two types of rewards are considered: SCORING and CHECKPOINT.

SCORING corresponds to the most intuitive reward in the *Football Engine* 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 requires a long sequence of consecutive events: overcoming the defense of a potentially strong opponent, and scoring against a keeper.

Due to the sparsity of SCORING, we also propose the CHECKPOINT reward. In this case, we augment the SCORING one with an additional auxiliary reward contribution for moving the ball close to the goal in a controlled fashion. More specifically, the agent obtains an additional reward of $+0.1$ the first time an active player with the ball crosses each of the checkpoint regions. We divide the opponent's field in $n = 10$ regions according to the Euclidean distance to the opponent goal. First time our player steps into one region with the ball, the reward coming from that region and all previously unvisited further ones will be collected. In total, the extra reward can be up to $+1$, the same as scoring a goal. To avoid penalizing an agent that would not go through all the checkpoints before scoring, any non-collected checkpoint reward is added to the scoring reward. Checkpoint rewards are only given once per episode. Note that, under most representations, this reward is not Markovian. The idea behind this reward is to encode our domain knowledge that scoring requires the ball to be close to the opponent's goal.

In addition, researchers can easily add their new reward functions using their own wrappers.

Other details For both IMPALA and Ape-X DQN, we use Super Mini Map representation . For each of the six considered settings (three *Football Benchmarks* and two reward functions), we run five random seeds for 500 million steps each.

Experimental Results

Our experimental results for the *Football Benchmarks* are shown in Figure 4. Note these are preliminary experimental runs that we plan to update and complete soon, based on a wider sweep of hyper-parameters.

We compare the performance of Impala and Ape-X DQN in the three difficulty levels (easy, medium, and hard) under two different rewards (scoring and checkpoint).

Our main findings can be summarized as follows.

First, the impact of different difficulty levels is clear from the plots. Harder tasks take significantly more steps to make some progress. Second, with the raw scoring reward, Impala is only able to convincingly solve the easy task. Even in this

(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.

case, it finds a way to consistently score only after 100M training steps. However, performance slowly improves over training steps in all tasks. On the other hand, the checkpoint rewards speed up training for Impala, especially after 20M steps. In particular, after 500M steps, Impala is able to repeatedly score in games at all levels. Despite the fact that the agent ends up scoring, the amount of required experience is simply enormous, and improved sample efficiency remains a big open question. Third, Ape-X DQN does not seem to benefit from the checkpoint reward; its performance is very similar with both reward functions. Notably, DQN makes progress in the easy scenario quickly (actually, it wins games on average at all stages shown in the plot). Unfortunately, learning progress on the medium and hard cases seems minimal, and further training is probably required.

Football Academy

Training agents for the *Football Benchmarks* can be challenging. Accordingly, we also provide the *Football Academy*: a diverse set of scenarios of varying difficulty. Its main goal is to allow researchers to get started on new research ideas quickly, and iterate on them. Examples of the Academy scenarios include settings where agents have to learn how to score against an empty goal, how to run towards a keeper, how to quickly pass in between players to defeat the defense line, or how to execute a fast counter-attack. Most of them are standard exercises in team practice sessions; see Figure 5 for some concrete instances.

A *fundamental* advantage of the Football Environment is that, by using a simple API, researchers can also easily define their own scenarios and train agents to solve them. We hope this empowers algorithm designers with the freedom to tackle a wide range of research questions without having to worry about the underlying simulator infrastructure.

We first provide a high-level description of each scenario:

1. *Empty Goal Close.* Our player starts inside the box with the ball, and needs to score against an empty goal.
2. *Empty Goal.* Our player starts in the middle of the field with the ball, and needs to score against an empty goal.
3. *Run to Score.* Our player starts in the middle of the field with the ball, and needs to score against an empty goal. Five opponent players chase ours from behind.
4. *Run to Score with Keeper.* Our player starts in the middle of the field with the ball, and needs to score against a keeper. Five opponent players chase ours from behind.
5. *Pass and Shoot with Keeper.* Two of our players try to score from the edge of the box, one is on the side with the ball, and next to a defender. The other is at the center, unmarked, and facing the opponent keeper.
6. *Run, Pass and Shoot with Keeper.* Two of our players try to score from the edge of the box, one is on the side with the ball, and unmarked. The other is at the center, next to a defender, and facing the opponent keeper.
7. *3 versus 1 with Keeper.* Three of our players try to score from the edge of the box, one on each side, and the other at the center. Initially, the player at the center has the ball, and is facing the defender. There is an opponent keeper.
8. *Corner.* Standard corner-kick situation, except that the corner taker can run with the ball from the corner.
9. *Easy Counter-Attack.* 4 versus 1 counter-attack with keeper; all the remaining players of both teams run back towards the ball.
10. *Hard Counter-Attack.* 4 versus 2 counter-attack with keeper; all the remaining players of both teams run back towards the ball.

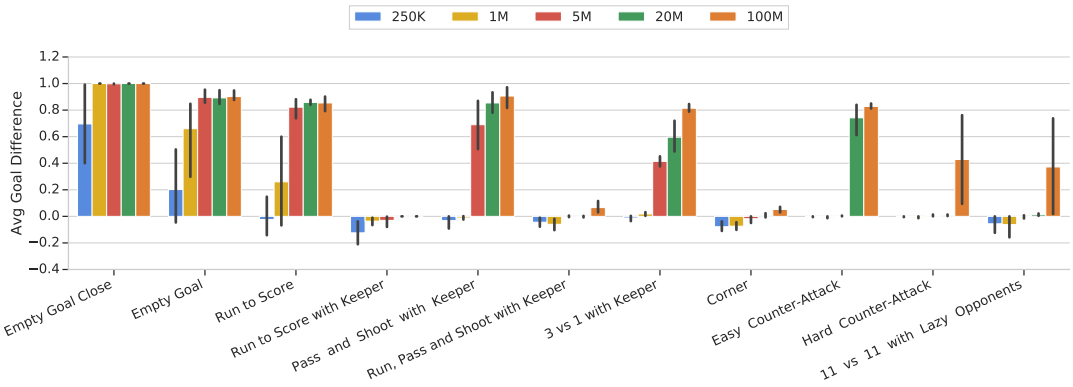

Figure 6: Average Goal Difference on *Football Academy* for IMPALA with SCORING reward.

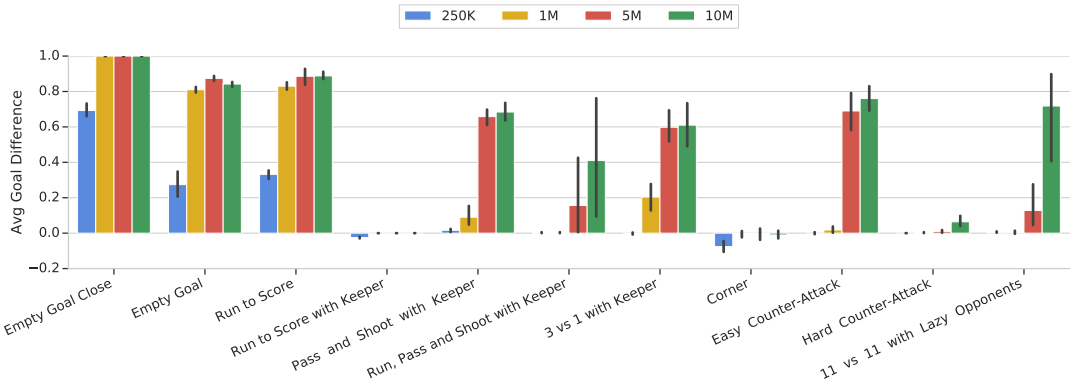

Figure 7: Average Goal Difference on *Football Academy* for PPO with SCORING reward.

11. *11 versus 11 with Lazy Opponents.* Full 11 versus 11 game, where the opponents cannot move but they can only intercept the ball if it is close enough to them. Our center-back defender has the ball at first.

Experimental Setup

The experimental setup for Academy scenarios is mostly similar to that of the *Football Benchmarks*. The main difference concerns how the episodes end. In order to speed up learning, and to make algorithms fail fast, we end the episode when one of the following events happen: we score, our team loses the ball, or the game stops (for example, when the ball goes out, or there is any kind of kick: side-kick, goal-kick, free-kick...). There is also a maximum length of 400 frames, after which the episode automatically ends. There are two exceptions; for the *Corner* scenario we do not end the episode if we lose the ball possession (but we do if we score or the game stops), and for the *11 versus 11 with Lazy Opponents* scenario, the maximum length is 3000 frames (like in the full games). All scenarios are stochastic. Note that in the Corner scenario the opponent team can (and will sometimes) score goals, and in other scenarios we may score own goals, leading sometimes to negative scores.

Experimental Results

Our experimental results for the *Football Academy* are shown in Figures 6, 7, 9, and 10 (last two are provided in the appendix). Figures 6 and 7 correspond to the scoring reward, for Impala and PPO respectively, while Figures 9 and 10 display the checkpoint reward. As in the previous section, note these are preliminary experimental runs, and that we plan to update and complete them soon, based on a wider sweep of hyper-parameters.

Our main findings follow. First, we observe task difficulties are certainly heterogeneous. Some tasks like empty goal and run to score can be solved in no more than 5M steps, while corner or the counter-attack scenarios are actually quite challenging. Second, IMPALA convincingly solves (say, average goal difference ≥ 0.8 somewhere in the confidence interval) the following scenarios with both reward functions: Empty Goal Close, Empty Goal, Run to Score, Pass and Shoot with Keeper, 3 vs 1 with Keeper, and Easy Counter-Attack. However, in some of these (3 vs 1 with Keeper, and Easy Counter-Attack) it takes up to 100 million steps to learn such a policy. In some of the scenarios, we do not expect the checkpoint reward to help speed up learning. However, in other scenarios it does clearly help. In particular, IMPALA with checkpoint reward is able to perform well in Run to Score with Keeper, Run, Pass and Shoot with Keeper, and 11 vs 11 with Lazy Opponents, whereas

under scoring reward it did rarely succeed. Unfortunately, the Corner and Hard Counter-Attack are not solved under 100M steps while there is evidence of some progress. Third, for PPO we only include results from training up to 10M steps due to limited parallelizability compared to other algorithms. PPO's performance is reasonably similar to IMPALA's on the following scenarios: Empty Goal, Empty Goal Close, Run to Score (whereas PPO learns faster here), and Pass and Shoot with Keeper. On the other hand, PPO seems to learn faster in both 3 vs 1 with Keeper, and Easy Counter-Attack. After 10M training steps, PPO with scoring reward has not been able to score goals in the hardest scenarios: Run to Score with Keeper, Corner, and it has had scored less than 0.1 on average on Hard Counter-Attack. In particular, the checkpoint reward does not seem to be helpful for PPO so far, with the exception of Run to Score with Keeper scenario.

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 some 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 hope 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.

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.; Kttler, H.; Lefrancq, A.; Green, S.; Valds, 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.

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*.

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.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

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*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland,

A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Nagarajan, P.; Warnell, G.; and Stone, P. 2018. Deterministic implementations for reproducibility in deep reinforcement learning. *arXiv preprint arXiv:1809.05676*.

OpenAI. 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.; et al. 2017. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.

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.

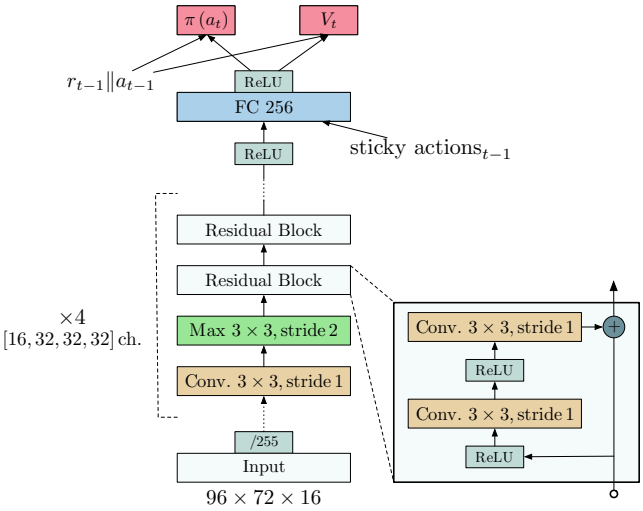

Figure 8: Architecture used for IMPALA experiments.

Hyper-parameters & architectures

IMPALA

In Table 2 we list hyper-parameter setting that was used to run all our reported IMPALA results. The model we use is inspired by Large architecture from (Espeholt et al. 2018) and is depicted in Figure 8.

Parameter	Value
Action Repetitions	1
Batch Size	64/128
Discount Factor (γ)	.997
Entropy Coefficient	.00025
Learning Rate	Linear decay from .00014 to .000075
Number of Actors	500/200
Optimizer	Adam
Reward Clipping	[-1, 1]
Unroll Length/n-step	30
Value Function Coefficient	.5

Table 2: Hyper-parameters used for training with IMPALA.

PPO

We present hyper-parameters used with PPO in Table 3.

Parameter	Value
Action Repetitions	1
Clipping Range	.04344
Discount Factor (γ)	.999
Entropy Coefficient	.001
GAE (λ)	.95
Gradient Norm Clipping	.5
Learning Rate	.0000494
Number of Actors	16
Optimizer	Adam
Training Epochs per Update	4
Training Mini-batches per Update	4
Unroll Length/n-step	256
Value Function Coefficient	.5

Table 3: Hyper-parameters used for training with PPO.

Ape-X DQN

In Table 4 we list hyper-parameter setting that was used to run all our reported Ape-X results.

Parameter	Value
Action Repetitions	1
Batch Size	512
Discount Factor (γ)	.997
Evaluator ϵ	.01
Importance Sampling Exponent	.2
Learning Rate	.00001
Number of Actors	500
Optimizer	Adam
Replay Priority Exponent	.6
Reward Clipping	[-1, 1]
Target Network Update Period	2500
Unroll Length/n-step	30

Table 4: Hyper-parameters used for training with Ape-X DQN.

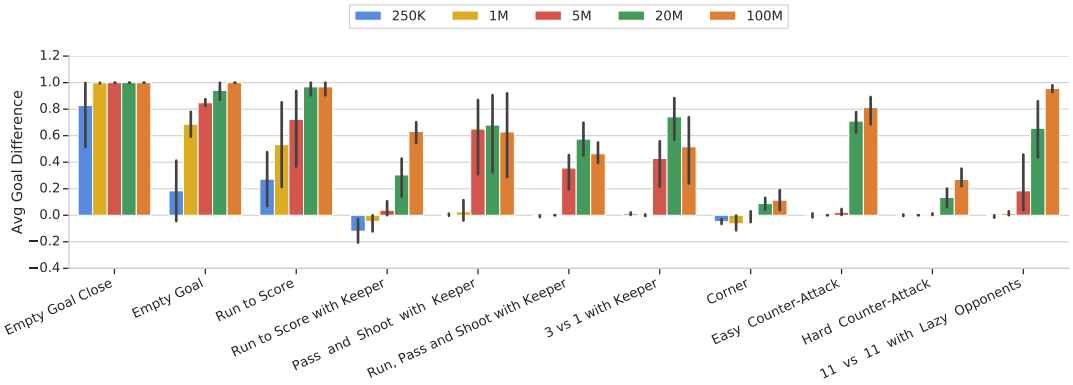

Figure 9: Average Goal Difference on *Football Academy* for IMPALA with CHECKPOINT reward.

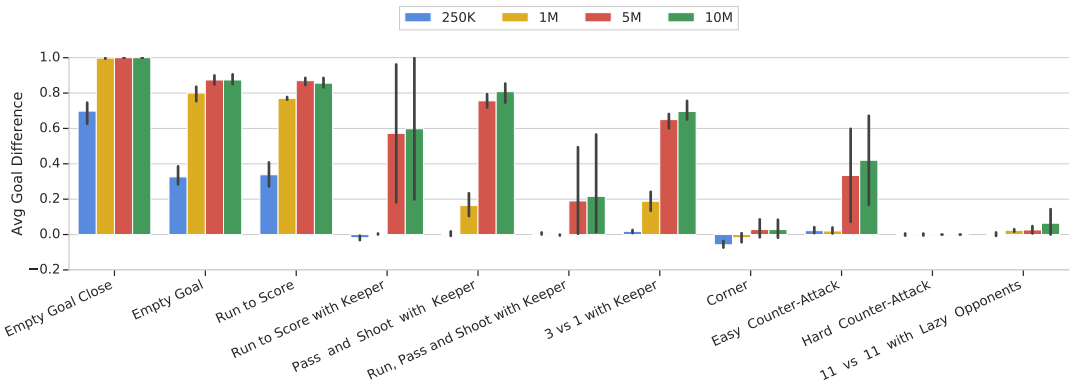

Figure 10: Average Goal Difference on *Football Academy* for PPO with CHECKPOINT reward.