Deep Reinforcement Learning in RoboCup 2D Soccer

Yona Cohen, Orr Krupnik, Daniel J. Mankowitz, Tom Zahavy, Shie Mannor
Electrical Engineering Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

RoboCup is a challenge designed to push the state-of-the-art in robotics and AI. In this paper we attempt to solve one sub-domain of simulated RoboCup soccer, namely RoboCup 2D, which is in itself a dynamic, high-dimensional domain. To approach this challenge we use Deep Reinforcement Learning. Numerous previous solutions exist; however, to the extent of our knowledge, we present the first such solution which uses the raw screen image of the game as input. This is made possible by the recent development of Deep-Q-Networks (Mnih et al. (2015)) which have shown competitive results using the same approach on dozens of Atari 2600 games. Applying this approach to the Half Field Offence RoboCup environment (Hausknecht et al.), using a simple modular interface also presented in this paper, we obtain a success ratio of near 100% on an empty goal, over 65% goals scored in a 1-on-1 offensive game, and promising results in a 1-on-2 scenario. The modularity of our solution will enable experimentation with additional learning challenges in this framework, such as additional agents or more complex skills and strategies. An case study for one of these extensions, of two offensive players cooperating to score, is presented as well.

Introduction

RoboCup is an ongoing global project, aiming to push forward the state-of-the-art in robotics, machine learning and artificial intelligence, by posing a formidable challenge: by the mid-21st century, a team of fully autonomous humanoid robots shall win a game of soccer against the winning team of the most recent World Cup [Kitano et al. (1997)].

Since its inception, the RoboCup challenge has been broken down into multiple domains which focus on various areas of research, ranging from robotics, through computer vision and control theory, to strategy algorithms and machine learning. One such domain is RoboCup 2D, which can simulate a full soccer game on a 2D screen, allowing participating teams to ignore the mechanics of autonomous robots in favor of focusing on strategy, game play and learning. This domain has been subject to much research over the years (Lattner et al. (2005), Wu and Chen (2007), Celiberto Jr et al. (2007), Bai, Wu, and Chen (2012)), and is still considered a challenging domain for learning systems.

The challenges of RoboCup 2D as a platform for machine learning are numerous, the major ones being (a) an extremely large and continuous state space, resulting from the variable nature of the game [Hausknecht and Stone (2015)]; (b) a very dynamic environment, with parameters (such as locations of the ball and players, player speeds and the game state) changing on-line and in quick succession and (c) a multiple agent environment, which requires the learning algorithm to react and adapt to actions of other agents on the field.

Figure 1: An example of the RoboCup 2D environment

In recent years, an exciting new method which could solve this type of problem has resurfaced, namely deep reinforcement learning. More specifically, Deep-Q-Networks [Mnih et al. (2015)] (DQN) equip reinforcement learning agents with a powerful non-linear function approximator, which enables them to tackle the problem of large state spaces and complex environments. DQN has been shown to reach high level control results in various game environments, in a mostly domain independent fashion, using only the pixels of the screen image as input [Mnih et al. (2015)]. This makes DQN a powerful tool to use when trying to solve RoboCup.

In this paper, we approach the RoboCup 2D challenge in this more general approach, without relying heavily on prior knowledge of the domain and the specific information available to the agent. We attempt to have an agent learn how to play RoboCup, using only the screen image of the game as input. This approach makes DQN the perfect

tool for the task, as it has been shown to be successful in similar setups [Mnih et al. (2015)]. We use a sub domain of RoboCup 2D, Half Field Offense (HFO; see Hausknecht et al.), which is simpler to control and understand than a full RoboCup 2D simulation.

Previous attempts at solving RoboCup 2D (and specifically, HFO) using various methods have been made [Bai, Wu, and Chen (2012)], including some using reinforcement learning [Barrett and Stone (2015)], and even the DQN algorithm [Hausknecht and Stone (2015)]. However, our work is the first time the screen image of the game itself is used, as opposed to hand crafted features relying on previous knowledge of the game. This receives support from multiple previous examples of usage of reinforcement learning to make sense of image data, such as Asada et al. (1996), Peng and Bhanu (1998), Michels, Saxena, and Ng (2005) and of course Mnih et al. (2015). In this paper, we present the first setup allowing an agent to learn in the RoboCup 2D environment using the image as input. This image-based approach shows good results, as will be shown later, compared to other proposed solutions. Additionally, this is one of the first solutions to use the relatively new DQN, and pushes it outside the relatively simple environment of Atari 2600 games into the much more dynamic and complicated RoboCup domain.

Main Contributions: (1) We show the plausibility of an image-based Deep Reinforcement Learning approach to learning in the RoboCup 2D environment, by presenting substantial learning results achieved by this approach. (2) A novel interface between DQN and the HFO environment, allowing additional future work on our solution. (3) Our platform is a modular solution to the problem, which will allow future experimentation with numerous learning challenges in the RoboCup 2D domain, such as multi-agent learning, larger action spaces or more complex skills.

Background

Reinforcement Learning: A reinforcement learning agent attempts to learn a policy $\pi : S \rightarrow A$ by maximizing the reward it receives in return for performing specific actions. A policy is a mapping from the state space of the problem, S, to a probability distribution over the actions, A, available at a certain state. At each time step t, the agent chooses an action $a_t \in A$ available at the current state, $s_t \in S$, and observes the reward received by taking this action, r_t, and the resulting state $s_{t+1} \in S$. To look at a suitable performance criterion, in infinite horizon problems, we consider the cumulative return at time t which is given by $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$, where $\gamma \in [0, 1]$ is the discount factor. We have a representation of the expected return after taking an action from a specific state, by defining the action-value function: $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$. The action-value function obeys the Bellman Equation, which is the basic recursive update equation for any reinforcement learning problem:

$$Q^\pi(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a'} Q^\pi(s_{t+1}, a')]$$

Deep-Q-Networks: The DQN algorithm (Mnih et al. (2015)) uses a Convolutional Neural Network (CNN, Krizhevsky, Sutskever, and Hinton (2012)) to approximate the optimal Q function, and from it learn the optimal policy. This is done by optimizing the weights of the network in a way which minimized the Temporal Difference (TD) error of the optimal Bellman Equation:

$$\mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \| Q_\theta(s_t, a_t) - y_t \|_2^2$$

where

$$y_t = \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_{a'} Q_{\theta_{target}}(s_{t+1}, a') & \text{otherwise} \end{cases}$$

DQN is an offline algorithm; therefore, it doesn't use the current state and action to update the network weights. Instead, it samples a minibatch of tuples $[s_t, a_t, r_t, s_{t+1}, \gamma]$ from an **Experience Replay** [Lin (1993)], which is a buffer storing the agent's experiences in each time step to enable later training updates of the DQN network. The DQN maintains two separate Q-networks: one current Q-network with parameters θ, and a target Q-network with parameters θ_{target}. Once every fixed number of training steps, DQN sets θ_{target} to θ, to avoid frequent updates of the target network and therefore noisy learning.
We have added three upgrades to the DQN architecture, which have shown improvement of results in other domains, and have somewhat boosted ours as well.

Success Experience Replay: similar to, but simpler than the Prioritized Experience Replay (Schaul et al. (2015)), it is sometimes preferential to select from the Experience Replay described above according to a distribution other than uniform. In the Success Replay, an additional buffer is maintained, holding transitions created in successful learning episodes. Then, with a certain (controllable) probability, minibatch samples are selected from this success memory instead of the general one. This gives the agent more successful episodes to learn from, which helps navigate a large, mostly unsuccessful state space. Our implementation is similar to the one used by Tessler et al. (2016).

Double DQN (Van Hasselt, Guez, and Silver (2016)): Double DQN (DDQN) prevents overly optimistic estimates of the value function. This is achieved by performing action selection with the current network θ and evaluating the action with the target network θ_{target} yielding the DDQN target update $y_t = r_t$ if s_{t+1} is terminal, otherwise $y_t = r_t + \gamma Q_{\theta_{target}}(s_{t+1}, max_a Q_\theta(s_{t+1}, a))$. DDQN has been shown to improve learning performance, and does so in our work as well.

Dueling Networks Architecture (Wang, de Freitas, and Lanctot (2015)): The Dueling architecture employs two streams at the final layer of the DQN network architecture: one to estimate the value function, and the other to for the Advantage function, which relates the value and Q-functions: $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. The two streams

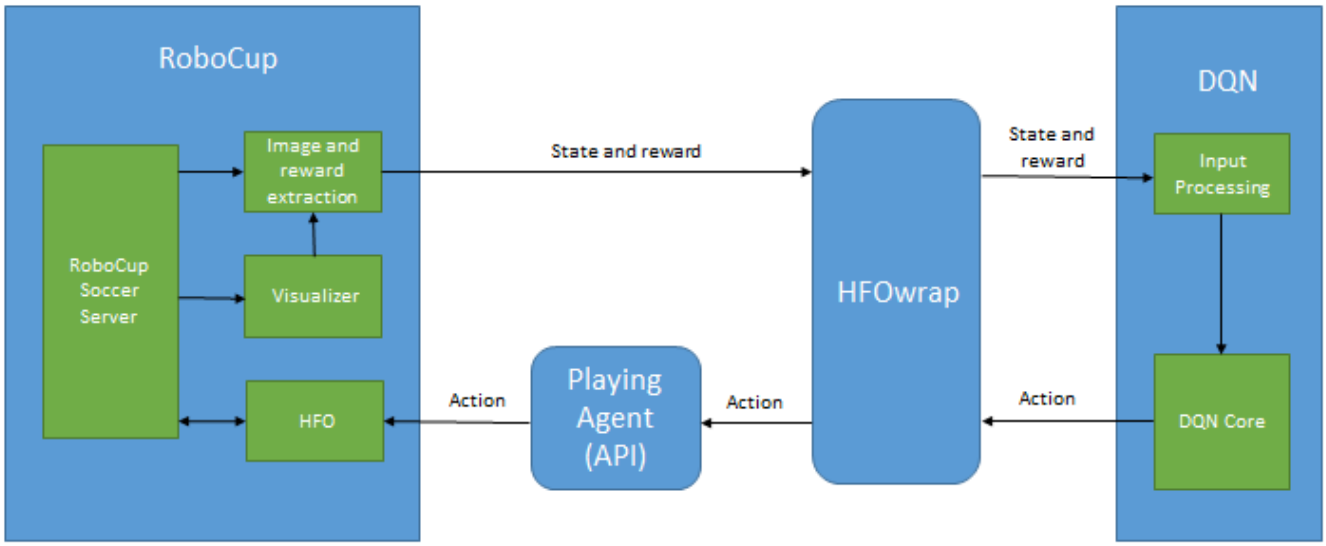

Figure 2: Block description of the relations and data flow between the various blocks comprising our system

are combined for an estimate of the Q-function, using $Q(s,a;\theta) = V(s;\theta) + (A(s,a;\theta) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta))$. This method has been shown to produce more accurate estimates of the Q-values.

Method

Our system delivers a modular interface between DeepMind's Deep-Q-Networks (DQN) framework [Mnih et al. (2015)] and the RoboCup 2D Half Field Offense (HFO) environment [Hausknecht et al.].

HFO is an easy-to-use, albeit challenging learning environment, which supplies a platform for the sub-task of a full soccer game by simulating the offensive side of a soccer field. In essence, HFO is an API which enables agents to interact with the RoboCup Soccer Server [Itsuki (1995)], the underlying engine of any RoboCup2D simulation. The RoboCup Soccer Server simulates a 2-dimensional soccer game by allowing multiple agents (or clients) to connect to it simultaneously, receive game information and convey actions. Specifically, the HFO environment enables a software agent to use simple commands for various tasks in relation to the RoboCup Server, including requesting information about the state of the game (game status: in play, out of bounds etc., ball location, player locations), and informing the server of the player's intended actions.

An additional module on the RoboCup side of our system is the soccerwindow2 visualizer, which allows us to view the RoboCup2D game as it progresses. The visualizer is also the module which creates the actual image from the parameters supplied by the server (the server can run the game "headless", meaning no image is shown). For our purposes, as the screen image of the game is used as the raw state for the learning process, we insert a TCP port into the visualizer code, to stream the image as input to the DQN module. This is done any time there is a change in the image, and it is repainted by the visualizer. The TCP port can be configured to send the image to one or more DQN modules, in order to enable multi-agent games.

On the other end of our framework, we use the original implementation of DQN supplied by DeepMind. To enable it to communicate with the RoboCup Server, we replace the original ALE Wrap framework [Bellemare et al. (2012)], with our own HFO Wrap. HFO Wrap is a Lua module, which serves a dual purpose:

- Receive the screen image from the soccerwindow2 visualizer via a TCP port, prepare it and supply it to the DQN core as input.
- Receive a selected action from DQN core, and pass it on as a command to the RoboCup Server (via HFO). In return, receive an update on the reward gained and the status of the game.

HFO Wrap is written in Lua; however, the HFO API does not supply a Lua interface. Instead, we use Lunatic Python [Niemeyer (2006)] to enable usage of Python objects inside Lua code. The object we use is a Python interface capable of interacting with the HFO Python API, named APIAgent. APIAgent merely simplifies control of the HFO API from inside Lua code, and lets HFO Wrap ignore the specifics of controlling the RoboCup server. One major advantage of this setup, as we will discuss below, is its modular structure, which may enable us to work with multiple agents in future development.

See figure 2 for a schematic of our whole system, as described in the previous paragraphs. Each box depicts one of the modules or sub-tasks mentioned above. Arrows (with

their respective labels) explain the data flow between the various modules.

During the course of our work, we have encountered several situations in which it would have been beneficial to run HFO and DQN in two separate processes (or threads), as opposed to the current mode of work where DQN calls the HFO functions and they run in the same process. One such example was our attempt to run two separate HFO agents in parallel with the same network. Our suggestion for enabling this process is to reconstruct the architecture connecting the modules, and use TCP ports (or some other thread communication mechanism) for communication between the HFO Wrap module and the APIAgent module. This would enable independent running of the processes, and replace the sometimes cumbersome Lunatic Python syntax.

Experiments

An HFO game is a subset of a full RoboCup2D soccer game, and consists of one offensive soccer episode. This begins with the appearance of the ball and offensive players in random positions on the offensive half of the field, and ends with either a goal scored, the ball captured by defense, the ball rolling out of bounds or the game timing out (this is a controlled parameter). To score a goal, the offensive agent has to learn how to find the ball, move it to an advantageous position and kick it towards the goal, avoiding the defensive players and goalkeeper.
DQN is a deep reinforcement learning framework; to enable it to learn how to play RoboCup, we need to pose the HFO domain as a reinforcement learning problem. Therefore, we should define the states, actions and rewards correctly to facilitate the learning process of the DQN agent.

Figure 3: Raw state as extracted from the soccerwindow2 visualizer

State: The raw state extracted from the soccerwindow2 visualizer, in our setup, is a square 252x252 matrix describing the pixel data of the screen image. We chose to crop the full screen image of the HFO game, as one side of the field never changes and does not supply any relevant information for learning. In addition, we chose to make some changes to the image creation settings, in order to make some of the image features more distinguishable after the pre-processing step, which is the first layer of the DQN. These changes include:

- Doubling the size of the ball
- Enlarging the players
- Remove the stamina bar attached to players

See figure 3 for an example of an image extracted from the visualizer.
The image is then fed into the DQN pre-processing step. This further reduces the state size by down-sampling the image to 84x84, and also converts the color of the image from full RGB to grey-scale (as described in Mnih et al. (2013). See figure 4 for an example of the resulting image). The final, full state is obtained by concatenating four consecutive images into one column image. This is then fed as input into the DQN core, which is described below.

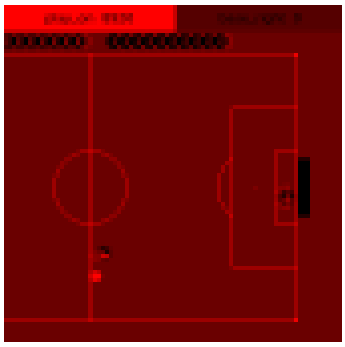

Figure 4: Down-scaled state after DQN pre-processing step

Actions: One of the main challenges in playing RoboCup is that the full action space is virtually continuous - agent's can chose to look or move in any direction, and when in posession of the ball, to kick it in any direction and at many levels of force. HFO allows three abstraction levels of actions, which are translated into lower-level operations sent to the RoboCup server. In our framework, we use the highest level actions only, which simplifies the action space into three main actions:

- Move: locate the ball and move towards it.
- Shoot: Kick the ball towards the goal, if it is in possession.
- Dribble: Move with the ball towards the goal, if it is in possession.

Additional high level actions exist, such as MoveTo or Pass; these are irrelevant for our setup, as we only have one agent mounting the offense. We may use additional actions in future work.

Game Event	Reward
Goal scored by agent	1000.0
Ball captured by defense	-4.0
Ball out of bounds	-1.0
Game out of time	-1.0
In game (any other state)	-0.001

Table 1: Reward function for the offense agent

Reward: Previous work attempting reinforcement learning in RoboCup [Celiberto Jr et al. (2007), Hausknecht and Stone (2015)] has included extensive reward engineering to facilitate any learning by a RoboCup agent. This has been done due to the sparse nature of the "natural" rewards in the game, which typically occur at the end of a game trajectory. In our setup, we found that a simple reward scheme suffices to reach satisfactory learning and results. Table 1 shows our reward function as received at various game events.

The DQN agent is the core of our learning system, and uses the input of states and rewards to select an action to play (by sending it back to the RoboCup server). In the core of the DQN algorithm, a Deep Neural Network receives the pre-processed state as input, and passes it through multiple hidden layers (two convolutional, one fully connected rectifier unit layer) to the output layer. This is a fully connected linear layer, with one output node for each available action (see Mnih et al. (2013) for details).

We conducted two main experiments to observe the learning ability of DQN in the RoboCup environment: one of a game between one offensive player and a goalkeeper, and a second with an additional defensive player. A third experiment we conducted, as a baseline, involved one offensive player scoring against an empty goal.

HFO is an entirely different environment from the Atari games on which the original DQN framework was tested on by Mnih et al. (2015). Therefore, to facilitate learning, we changed multiple parameters controlling the learning process.

- Experience Replay [Lin (1993)] size: we dramatically lowered the size of the experience replay memory, to 120,000 states. This was necessary due to the large part of the state space in RoboCup which does not supply any useful information on how to score goals. We found that a replay memory which is too small also prevents any significant learning from occurring, as not enough states are remembered to enable re-enactment of successful trajectories.
- Larger learning rate: the original small learning rate meant learning would take to long to show any meaningful results. The larger learning rate of 0.002 enabled our agent to learn in a reasonable amount of time, however, was not large enough to create a noisy and erratic learning process.
- Learning steps and learn start: with the larger learning rate, we found 10 million learning steps were enough to reach good results and show successful learning of goal scoring methods. Additionally, we were able to start the learning process earlier, as our experience replay was smaller and took less time to fill up.
- Reward clipping: in order to enable clear distinction between various reward situations, we canceled the reward clipping set in the original DQN framework. This allows our rewards (see table 1) to be meaningful and push learning forward.

The results of our two experiments, depicting our agent playing against a goalkeeper only, and playing against a goalie and an additional defensive player, can be seen in figure 5. Scoring percentages can also be observed compared to some benchmarks in the next section. An additional plot of results is given for the scenario of scoring against an empty goal, as a baseline and for the sake of completeness. This can be seen in figure 6.

In the one-on-one scenario, we achieve very nice results, with the agent scoring goals about 50% of the time at the end of the training process. In the 2 on 1 scenario, the scoring percentage is somewhat lessened, due to the higher difficulty level of the task. Both results were somewhat augmented by implementing a Successe Experience Replay [Tessler et al. (2016)] based on the Prioritized Experience Replay [Schaul et al. (2015)]. This additional part of the replay memory allows the DQN core, while sampling trajectories to update the target network, to replay successful trajectories more often and therefore learn more efficiently. Both of these scenarios achieve results lower than the empty goal baseline, which is expected due to the addition of the goalkeeper, which raises the difficulty of the task. We discuss why the task becomes so much harder under these conditions later in this report.

Multi-Agent Scenarios: A natural extension to our work was attempting to play with more than one offensive player. This is a requirement for scaling up to full RoboCup games, as well as an interesting problem in itself. With two players on the field, both the state space and the action space grow larger, and gameplay and strategy for each of the players are different from the single player scenario. Therefore, multi-agent HFO requires some changes to the problem setup and framework, which we describe below.
First, to enable meaningful multiplayer offense, we added two available high level actions: Pass, which passes the ball to a specified player; and MoveTo, which moves the player to a specified spot on the field to wait for the ball. After establishing the additional actions, we decided to tackle the problem of cooperation between two agents using two different methods:

- Two players with two networks: Each player plays similarly to a single player (with the added actions), using the same input. This brings to mind scenarios of ad-hoc teamwork, as described by Hausknecht et al. and Stone et al. (2010). In this kind of team game, both players need to learn independently how to work with each other and score goals. It is easy to imaging how this method can

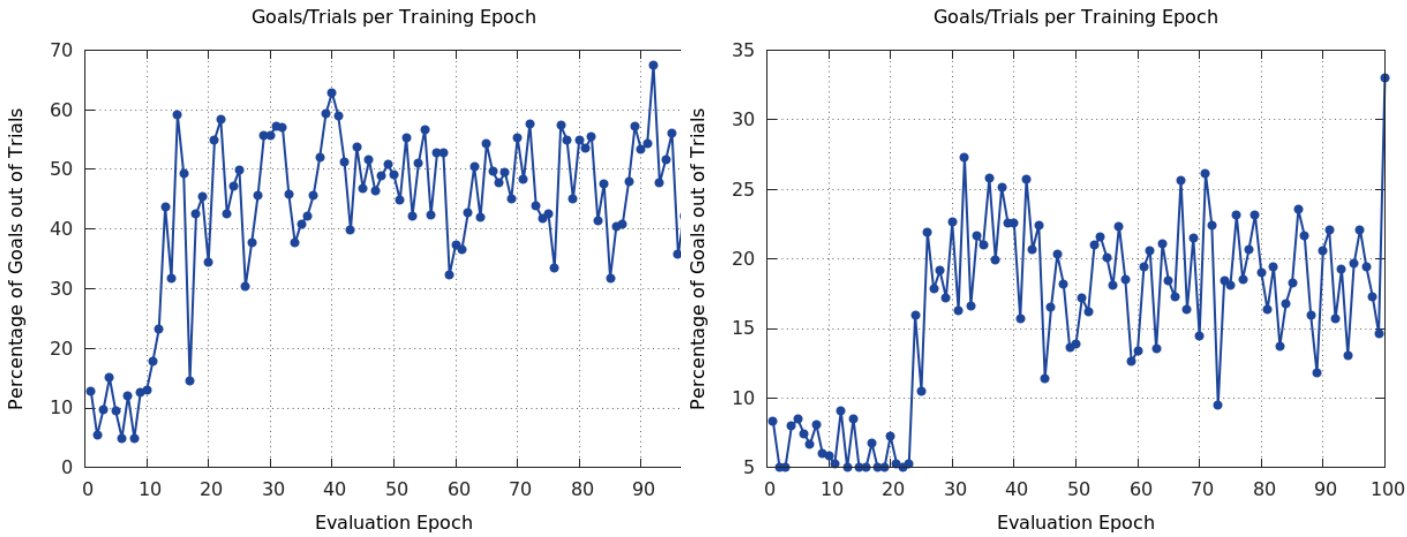

Figure 5: Results of our experiments, shown by percentage of goals scored out of trials at the testing stage, at the end of each training epoch. **Left:** One offensive agent playing against the goalkeeper. **Right:** One offensive agent playing against a goalkeeper and an additional defender.

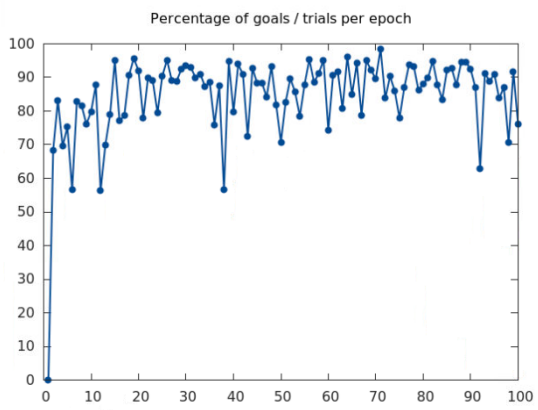

Figure 6: Percentage of goals per trials scored by a player against an empty goal.

easily scale up to a full RoboCup soccer team, if successful. The specifics of this kind of learning will be discussed below.

- Two players learning using the same network: Both players are controlled by the same network receiving the image of the field as input. The output is a pair of actions, one for each player. Since the amount of possible actions grows exponentially with the number of players, this method could be harder to scale up.

In practice, we have only been able to experiment with the first of these two methods, as the second one has faced multiple technical challenges (see our suggestion at the end of the Method section, above, as one example).

After several attempts and iterations, we found that to gain satisfactory results when playing with two players on two networks, some additional changes had to be made to the setup. We found that letting both players play similarly to a single player caused both players to learn to score at the same level as a single player, without using teamwork capabilities. Moreover, in some situations the players interfered with each other, resulting in scoring percentages slightly lower than the one player scenario. We achieved better results by assigning different roles to both players: one, dubbed the Shooter, was given actions suitable for an offensive player: kicking, dribbling and moving to a spot to wait for the ball, while the other player filled the role of a Midfielder, moving to the ball and passing to the Shooter. In this setup, we had to configure the rewards according to each player's role on the field. The Shooter gained the reward for scoring, and the Midfielder received reward for getting to the ball and for passing it to the Shooter. These changes resulted in better cooperation between both players, as seen visually while watching gameplay, although scoring percentage has not risen significantly. The experiments we made in this setup were two players scoring on an empty goal, two players playing against a goalkeeper and two players taking on a goalie and a defensive player. The results of these scenarios can be seen in Figure 7 and also compared to benchmarks in Table 2 and Table 3.

Discussion

As far as we know, we have shown the first results of a learning RoboCup 2D agent using an image-based approach and the DQN algorithm [Mnih et al. (2015)]. The learning process has been relatively successful, and compares to other results obtained in previous work in the HFO do-

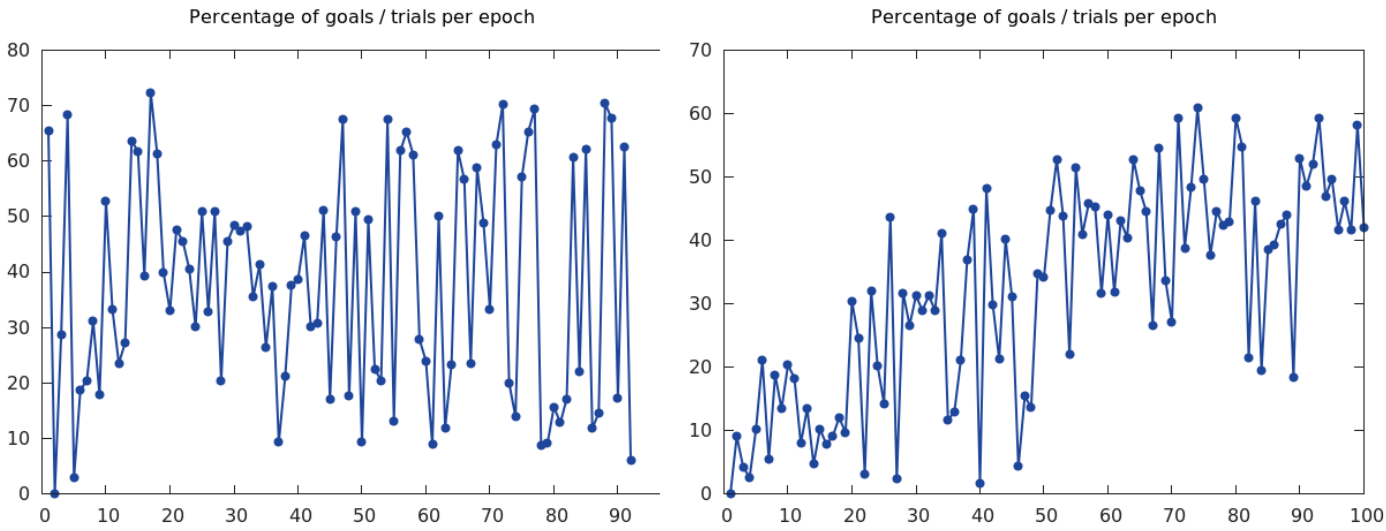

Figure 7: Results of our experiments, shown by percentage of goals scored out of trials at the testing stage, at the end of each training epoch. **Right:** Two offensive agents playing against the goalkeeper. **Left:** Two offensive agents playing against an empty goal, using a more complicated reward scheme.

Scenario	**1v0**	**1v1**	**1v2**	**2v1**	**2v2**
HFO DQN (our results)	98.45	66.7	20.6	61	12
Random (High Level Actions)	41	1.8	0.1	0.7	0
Helios (expert hand coded)	96.2	73.8	34.1	81.4	60
SARSA (High level features and actions)	91.1	88.9	40.4	92.3	62.8
Hand Coded	95.6	64.7	39.7	65.7	46.7

Table 2: Our results against benchmarks. Numbers indicate scoring percentage

main. Additionally, we presented a simple and easy-to-use interface linking DQN with the HFO environment. The modular approach in the construction of this interface will easily allow users to tackle further learning challenges in the RoboCup environment.

The numbers in Table 2 show our results compared to previous work done on the Half Field Offense domain, both by learning algorithms and by hand-coded methods. Most of the benchmarks supplied were collected by Hausknecht et al.. The Helios results were obtained using the algorithm played by the winning team of the 2012 2D RoboCup championship, the code for which can be found at Hidehisa (2010). Our results are created by "best seen in training" (although most have been observed consistently throughout experiments), while all other results in the table are averaged over 1000 epochs of testing (after training is done) and over 10 independent learning processes.

Additional results which may be relevant for comparison come from Hausknecht (2017), who employs various upgrades to the DQN algorithm and uses a parameterized (almost continuous) action space. This gives much better control over the agents and the variables of the game, while also making the problem much harder to solve algorithmically. Another difference is the way the rewards were awarded to the two players, also originating from the different action space. For these reasons, comparison of our work to these results may be unhelpful and may not reflect actual success rates. However, they are shown here for completeness. The results can be seen in Table 3; all results for these learning agents were chosen as the best results seen over 100 testing epochs, after the end of training.

For the most part, our results still do not surpass the achievements of solutions offered by previous work (Hausknecht et al.). Further refinement work on the DQN parameters and learning environment will be required to obtain more satisfactory goal scoring percentage. Additionally, in our work we used a very simple action space, consisting of three high level actions. Adding the ability to break down these actions into lower level, "micro-managing" commands, may enable the agent to modify its tactics and score in more scenarios. For instance, our agent can only shoot toward the goal; a lower level action may let the agent pick the right angle to shoot towards, and the strength with which to shoot. This kind of expansion to the action space could, however, extend the training time required

Scenario	2v0	2v1	Remarks
HFO DQN (our results)	72.3	61	Obtained using the experiments described above.
Independent Learning	98	N/A	Both agents learning independently on the same task
Centralized Control	96	1	One network controlling two agents with a concentrated state and action space
Shared network parameters	100	80	One network controlling two players, where the parameters of the first layers are shared
Shared replay memory	100	75 (maximum. average much lower)	Two networks for two players, with a shared replay memory to choose from

Table 3: Our results against results from Hausknecht (2017).

to reach any substantial results. The general non-heuristic approach has yet another drawback, which further extends the time required to obtain results. Some training sessions present weak results, while others display a "jump" in the learning curve, which then leads to much better performance. This makes the experimentation process even longer.

Regarding our multi-agent attempts, there may be other factors limiting cooperation (and hence, the level of results). One issue we have encountered was the shaping of the reward function. Different reward setups led to wide variation in simulation results, as agents could be rewarded together (as a team) or separately for their actions. Team reward can lead to players receiving high rewards for actions they should not have learned to repeat, since a team member may have scored on their own. On the other hand, personal reward may leave some team members out and prevent them from learning to assist in scoring goals. Another problem is differentiation between the agents, as all players receive the same state, which is the screen image of the game. Finally, the limitations of the high level actions manifest in the multi-agent scenario as well, and may be even worse than in the single player scenario, since more specific actions are required to enable players to pass the ball, plan their location and score.

Several approaches for future work using our framework make themselves clear. These can be divided into a few distinct areas. First, some technical improvements to our framework may eventually push results further. One of those improvements, discussed above, is to restructure the connectivity between DQN and HFO using threads or multiple processes. Enabling better connectivity for multiple networks opens up the domain of multi-agent learning, whether on the same network (controlling both players), on one network with separate final layers, using two networks with a shared experience replay or with two completely independet networks. In the action space, it may be beneficial to break down actions into lower level, more precisely controlled actions and observe the changes to the learning process. Additionally, more high-level actions can be constructed heuristically, to better capture the requirements of playing as a team. Another approcah for better actions could be using longer sequences of actions, or Skills [Sutton, Precup, and Singh (1999)], which could enable learning of deeper tactics and more complicated game play. As mentioned previously in this section, construction of a better reward function along with the augmented actions may be crucial for achieving significantly better results. Finally, recent research has been conducted on the game theoretic aspects of multi-agent learning [eg. Leibo et al. (2017)], which may be beneficial for learning in the HFO domain.

A combination of some of the above could ultimately lead to our approach enabling learning for a full RoboCup team.

References

Asada, M.; Noda, S.; Tawaratsumida, S.; and Hosoda, K. 1996. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. In *Recent Advances in Robot Learning*. Springer. 163–187.

Bai, A.; Wu, F.; and Chen, X. 2012. Online planning for large mdps with maxq decomposition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, 1215–1216. International Foundation for Autonomous Agents and Multiagent Systems.

Barrett, S., and Stone, P. 2015. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *AAAI*, 2010–2016.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2012. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*.

Celiberto Jr, L. A.; Ribeiro, C. H.; Costa, A. H.; and Bianchi, R. A. 2007. Heuristic reinforcement learning applied to robocup simulation agents. In *Robot Soccer World Cup*, 220–227. Springer.

Hausknecht, M., and Stone, P. 2015. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*.

Hausknecht, M.; Mupparaju, P.; Subramanian, S.; and Stone, P. Half field offense: An environment for multiagent learning and ad hoc teamwork.

Hausknecht, M. J. 2017. *Cooperation and communication in multiagent deep reinforcement learning*. Ph.D. Dissertation.

Hidehisa, A. 2010. Agent2d base code. https://osdn.jp/projects/rctools/.

Itsuki, N. 1995. Soccer server: a simulator for robocup. In *JSAI AI-Symposium 95: Special Session on RoboCup*. Citeseer.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; Osawa, E.; and Matsubara, H. 1997. Robocup: A challenge problem for ai. *AI magazine* 18(1):73.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lattner, A. D.; Miene, A.; Visser, U.; and Herzog, O. 2005. Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In *Robot Soccer World Cup*, 118–129. Springer.

Leibo, J. Z.; Zambaldi, V.; Lanctot, M.; Marecki, J.; and Graepel, T. 2017. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 464–473. International Foundation for Autonomous Agents and Multiagent Systems.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. Technical report, DTIC Document.

Michels, J.; Saxena, A.; and Ng, A. Y. 2005. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, 593–600. ACM.

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.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Niemeyer, G. 2006. Lunatic python.

Peng, J., and Bhanu, B. 1998. Closed-loop object recognition using reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(2):139–154.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Stone, P.; Kaminka, G. A.; Kraus, S.; Rosenschein, J. S.; et al. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2016. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *AAAI*, 2094–2100.

Wang, Z.; de Freitas, N.; and Lanctot, M. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Wu, F., and Chen, X. 2007. Solving large-scale and sparse-reward dec-pomdps with correlation-mdps. In *Robot Soccer World Cup*, 208–219. Springer.