Training a RoboCup Striker Agent via Transferred Reinforcement Learning

Warren Blair Watkinson II(✉) and Tracy Camp

Department of Computer Science, Colorado School of Mines, Golden, CO 80401, USA
{wwatkinson,tcamp}@mines.edu

Abstract. Recent developments in reinforcement learning algorithms have made it possible to train agents in highly complex state and action spaces, including action spaces with continuous parameters. Advancements such as the Deep-Q Network and the Deep Deterministic Policy Gradient were a critical step in making reinforcement learning a feasible option for training agents in real world scenarios. The viability of these technologies has previously been demonstrated in training a RoboCup Soccer agent with no prior domain knowledge to successfully score goals; however, this work required an engineered intermediate reward system to direct the agent in its exploration of the environment. We introduce the use of transfer learning rather than engineered rewards. Our results are positive, showing that it is possible to train an agent through a series of increasingly difficult tasks with fewer training iterations than with an engineered reward. However, when the agent's likelihood of success in a task is low, it may be necessary to reintroduce an engineered reward or to provide extended training and exploration using simpler tasks.

Keywords: Reinforcement learning · Transfer learning · Multiagent systems

1 Introduction

An elusive goal in artificial intelligence is the training of a robotic agent to solve problems or to act within a domain without being specifically programmed, modeled, or provided with heuristics to direct its behavior. Reinforcement learning techniques, in which an agent can explore and develop an understanding of its environment on its own, have the potential to realize that goal. The most interesting applications of reinforcement learning are in domains having extremely large or continuous state or action spaces. Despite several recent advances that have yielded excellent results in these types of domains, reinforcement learning continues to have challenges in domains where an agent must follow a long sequence of actions before it achieves a goal. These challenges are especially pronounced when the actions available to an agent have continuous parameters. The compounding effect of the long action sequence along with the infinite number of actions available to the agent at each step in the sequence may make it impossible for the agent to successfully explore the environment to reach a goal state.

The most common way to deal with the divide between an agent's initial state and a goal state is with intermediate rewards. Some domains have naturally occurring intermediate rewards, such as video games, where points might be earned for destroying enemies. Using reinforcement learning, an agent will explore its options within the game and quickly learn the value of actions that destroy enemies. If destroying enemies eventually leads to winning the game, the intermediate rewards will help direct the agent toward a winning goal state. Where the domain does not have naturally occurring intermediate rewards, the reinforcement learning scenario designer might engineer intermediate rewards that he or she believes will lead an agent toward an ultimate goal state. For example, in a video game where an agent starts on the extreme "left side" of a world with the goal of reaching the extreme "right side," such as in Super Mario Brothers on the 8-bit Nintendo Entertainment System (NES), an intermediate reward might be given every time the agent successfully advances toward the right. While these intermediate rewards address the lack of feedback between an agent's initial state and goal state, this approach will be suboptimal since the agent would likely not take shortcuts or other optimizations along the way. As an alternative to this intermediate reward approach, we explore a method of *transfer learning* in the RoboCup Soccer domain.

Transfer learning is the idea that the experience an agent gains while learning one task can help it successfully learn a different task. In this paper, we explore the viability of using transfer learning to train a robotic soccer striker agent to successfully score a goal. A striker agent has the problem we described previously. Specifically, from a random start state, the number of actions the agent must correctly select in order to score a goal, including moving to the ball, dribbling the ball toward the goal, and eventually scoring, is extremely large, and at each new state, the agent has nearly an infinite number of actions from which to choose. In order to overcome this challenge of exploration, we first train an agent on a simple goal scoring task followed by a series of increasingly difficult tasks.

In the sections that follow, we summarize background on reinforcement learning using Markov Decision Processes (MDPs) and recent advances that have seen success in the application of reinforcement learning approaches in large and continuous domain and action spaces, such as in RoboCup Soccer. We also provide an overview of the approach we took to train a RoboCup Soccer striker agent in 2d simulation and the results of our simulations. Lastly, we discuss our conclusions and future research opportunities.

2 Background

2.1 Reinforcement Learning and Markov Decision Processes

The most common approach in reinforcement learning problems is to model the domain as an MDP [14]. In an MDP, an agent perceives that it is in some state $s \in S$ within the environment where s is a feature vector: $s = \langle x_1, x_2, \ldots, x_n \rangle$. The agent chooses an action $a \in A$, which causes the agent to enter a new state according to a transition function $T\colon S \times A \rightarrow S$, a probability distribution mapping each state and action pair to the resulting state of the environment after the action is executed. The agent receives a numerical reward according to a reward function $R\colon S \rightarrow \mathbb{R}$ which is a mapping of each state to the instantaneous numerical award for arriving in that state. Usually, the

MDP is designed such that most states yield a reward of 0 and the goal state yields a positive reward. An agent's policy $\pi\colon S \rightarrow A$ is a probability distribution that governs which action the agent will choose from a particular state.

In a typical reinforcement learning problem, the agent can observe (or partially observe) its state s and is aware of the set of actions A available, but it does not know R or T. Some reinforcement learning approaches attempt to directly learn the model R and T. Other approaches seek to estimate the action-value or the state-value, which is the discounted value of a present action or state in earning a future reward according to a discount factor γ. The effectiveness of an agent is determined by how well its policy obtains rewards over the long run. The goal of reinforcement learning is to find an optimal policy π^* that maximizes future rewards. When the state or action space is relatively small, developing such a policy is trivial (as the entire environment could be explored by the agent). On the other hand, many domains, such as RoboCup Soccer, have such vast state and action spaces that it would be impossible to directly explore every possible state and action sequence. For example, see Table 1 for the state and action space complexity of various domains. In such domains, it is helpful to use reinforcement learning techniques that approximate an action-value or state-value by inferring similarities between various states. These action-value or state-value functions can then be used to discover an optimal policy.

Table 1. Approximate state and action space for various games

Game	State space	Action space
Tic-tac-toe	10^3	4.5
Checkers [2]	10^{18}	2.8
Backgammon [16]	10^{20}	20
Chess [4]	10^{46}	35
Go [18]	10^{127}	181
RoboCup Soccer[a]	10^{408}	10^{320}
Starcraft 2 [10]	10^{1685}	$[10^{50}, 10^{200}]$

[a]The RoboCup Soccer state space is a lower-bound estimate using 22 players, their positions and velocities, and individual stamina. To calculate the RoboCup state space, we assume the ball and each of the 22 players can be in one of 680×1050 positions on the field with a velocity in one of 360° and a magnitude as one of 10 values. Furthermore, each player has a stamina [0, 8000]. When we consider that these features are actually continuous and that each player may have many different player characteristics, the state space is far greater than this estimate. The actions used to calculate the action space is the subset of RoboCup actions available to the agents in our simulations.

2.2 Large and Continuous State and Action Spaces

Some of the earliest reinforcement learning work in 2D RoboCup Soccer [11] developed aggressive defensive behaviors in RoboCup defenders. Using a neural network to optimize the action-value function, Riedmiller et al. saw a significant increase in the success of an agent using learned behavior over hand-coded behavior. This success relied on significant reductions in the state and action space. For example, the learning agents used only the dash and turn commands and the parameters were discretized such that only 76 actions were available to each agent. The state space was also significantly reduced, as the set of starting states was limited to 5,000, and an episode would proceed for 35 time steps. While this seminal work was significant, to fully realize reinforcement learning in the large state and action space of RoboCup soccer, algorithmic advances were necessary.

In 2015, Mnih et al. developed the Deep-Q Network (DQN), an Artificial Neural Network (ANN) to estimate the action-value function [9], and demonstrated that an agent could learn to play Atari games, which involve highly complex state spaces. The DQN developed by Mnih et al. received game state information in the form of screen pixels. The DQN would output one of 18 discrete joystick commands that directed the agent's behavior, and the agent received the change in game score as reward. The DQN uses an *experience replay*, a stored vector of experiences $e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle$, at each time step t. All previous experiences are pooled together and sampled randomly during the gradient-descent update of the ANN. This experience replay has several advantages when approximating the action-value function. First, each sample can be reused many times to update the ANN, increasing the efficiency of data collected as the agent explores the environment. Second, because sequential states tend to be very similar, sampling from the experience replay avoids using highly correlated experiences to update the network. Finally, the experience replay approach provides an off-policy mechanism for developing an action policy. This is advantageous because samples in an on-policy mechanism can become saturated with states dictated by the current action policy. This saturation could lead to limited exploration of the state space and a policy that stagnates at a local optimum. The DQN also utilized a target network that would follow the actual ANN at a rate of $tau \ll 1$. The target network generated the target action-values used to update the network, thereby stabilizing learning. These benefits combine to make DQNs effective in high-dimensional state spaces. Mnih et al. demonstrated the success of the DQN in reinforcement learning on 49 Atari games; their trained agent learned to play Atari games at a human-expert level of performance with only the raw frames from the Atari game as input.

Another significant advancement in reinforcement learning [12] concerns the Deep Deterministic Policy Gradient (DDPG). In DQN, the policy gradient is estimated stochastically over an integration of both the action and state spaces, and it was believed that a deterministic policy gradient could not be found without a model-based learning approach. Silver et al. proved that a deterministic descent policy gradient can be integrated over state spaces alone in model-free learning. Lillicrap et al. incorporated the deterministic descent policy gradient into DQN in 2016 [8] creating the DDPG. This key advancement extended DQNs to allow agents to select actions with continuous parameters.

The DDPG is an actor-critic network where both the actor and critic is an ANN. The input to the actor network is the state feature vector. The actor network has two linear output layers. The first output layer selects discrete actions, and the second output layer provides continuous parameter values that correspond with the actions. The actor's outputs are then provided as inputs to the critic network with the state feature vector. The critic network generates an estimate of the action-value. Back-propagation of the critic network calculates the gradients of the action-value function with respect to the action. These gradients are provided as input to the actor for back-propagation, and the actor updates the agent's policy.

Hausknecht and Stone extended the DDPG actor-critic network by implementing an *inverting gradients* technique that reduces the magnitude of the gradient as it approaches its bounds and then inverts the gradient when the parameter exceeds its bounds. The inverting gradients method reduces the tendency of the critic to demand that the actor push the continuous parameters outside the parameter bounds [5]. They also introduced other enhancements to the DDPG algorithm including a hyperparameter β as a ratio of on-policy and off-policy learning updates. These enhancements stabilized the learning of the agent and yielded more consistent results from the DDPG algorithm.

Their results are impressive; after approximately 3 million iterations, they trained a RoboCup Soccer striker agent to score a goal against an expert goalie. As it would be impossible for an agent to discover the correct sequence of actions leading to a goal state in such a large and continuous state and action space, Hausknecht and Stone used an engineered intermediate reward. To direct the exploration of the agent within the soccer domain, the intermediate rewards provided up to one point for approaching the ball, up to three points for moving the ball toward the goal, and finally, five points for scoring a goal.

2.3 Transfer Learning

Despite extensive use of transfer learning techniques in reinforcement learning [15], we are aware of only one instance where transfer learning has been used in RoboCup Soccer. Torrey et al. [17] trained agents in RoboCup Keepaway [13] and transferred the knowledge gained to agents in Half-Field Offense [6]. Keepaway is a RoboCup part-task simulator in which *M keepers* keep the ball away from *N takers*. Half-Field Offense is a game in which *M* attackers attempt to score against *N* defenders. In their experiments, the authors of [17] initialized the reinforcement learning keepers with *advice* such as "when a taker is close, pass the ball to a teammate." Over a series of Keepaway games, the learning keeper agents refined the advice. A human user then mapped the refined advice learned in the Keepaway task to appropriate scenarios in the Half-Field Offense task. Thus, the learning attacker agents in Half-Field Offense benefited from the refined advice learned in Keepaway. Torrey et al. discovered that, initially, Half-Field Offense attackers without advice outperformed attackers receiving advice; however as learners continued learning and refining the advice over many games, the attackers which received advice outperformed attackers with no advice.

In [17], agents started with domain awareness. In particular, agents understood the meaning of state features, such as the distance to the ball, and they had the ability to perform fairly high level tasks, such as kick the ball to a distant teammate. We are not aware of any other use of transfer learning to support reinforcement learning in RoboCup Soccer, and certainly none where the agents are starting *tabula rasa*. In the following section, we describe an approach using transfer learning to train a RoboCup Soccer striker agent to score on a goal with no prior domain knowledge.

3 Methodology

3.1 Overview

Our experiments leveraged the idea of *learning from easy missions* [3,15]. To train a striker agent, we used a DDPG actor-critic network similar to the one used in [5]. Previous work used *reward shaping*, or an engineered intermediate reward, which is an artificial reward signal rather than a reward based directly on the actual goal. In our work, we trained an agent over a series of successively difficult learning phases. The action-value function approximation learned in previous phases, along with the replay memory, was preserved in the transfer, and allowed the agent to "jump start" its learning

Table 2. Agent learning phases

	$Ball_x$	$Ball_y$	Dist
Phase1	1.6	[-13.6,13.6]	0.0
Phase2	[10.5,26.3]	[-20.4,20.4]	0.0
Phase3	[10.5,26.3]	[-20.4,20.4]	0.1
Phase4	[10.5,26.3]	[-20.4,20.4]	5.0
Phase5	[42,52.5]	[-27.2,27.2]	~
Phase6	[42,52.5]	[-27.2,27.2]	~

(a) Experiment 1: Empty Goal Task

	$Ball_x$	$Ball_y$	Dist
Phase1	1.6	[-13.6,13.6]	0.0
Phase2	[10.5,26.3]	[-20.4,20.4]	0.0
Phase3	[10.5,26.3]	[-20.4,20.4]	0.1
Phase4	[10.5,26.3]	[-20.4,20.4]	5.0
Phase5	[42,52.5]	[-27.2,27.2]	~
Phase[6-10]	[42,52.5]	[-27.2,27.2]	~

(b) Experiment 2: Defended Goal Task

Starting ball positions represent the location of the ball relative to the center of the goal opening and to the goalie in meters. $Ball_x$ and $Ball_y$ define the ball's position relative to the goal where $Ball_x$ is the ball's distance away from the goal line, and $Ball_y$ is the ball's distance away from a line bisecting the goal opening. Where scalar values are depicted, such as 1.6, every trial started with the ball in the same position. Range values, such as [10.5,26.3], depict trials where the ball was placed stochastically between those values. Dist is the distance the agent is placed away from the ball beyond the agent's kickable range. A dist of ~ is designated for where the agent was placed stochastically on the field without regard to where the ball was placed. The first five learning phases between the two experiments were identical, with the exception that the defended goal task had a stationary goalie in the center of the goal opening.

in the new phase. Through reinforcement and transfer learning, we trained agents to perform in two different experiments in Half-Field Offense: an empty goal experiment (*Experiment 1*) and a defended goal experiment (*Experiment 2*).

In the empty goal experiment, we trained an agent to start from somewhere on the right half of the soccer field, run to the ball positioned randomly on the same side of the field, and then kick the ball into an undefended goal. In the defended goal experiment, we also trained an agent to run to the ball and kick it into a goal, but this time the goal was defended by an Agent2d [1] hand-coded expert goalie. Agents in *Experiment 1* used 59 egocentric features such as distance and angle to the goal, ball, and other landmarks on the field. Each feature was a scalar value ranging from $[-1, 1]$. The game state was fully observable to the agent. As a low-level feature set, these features represented a fundamentally basic viewpoint in the agent's frame of reference and did not include synthesized features, such as the direction of the largest angle between the goalie and a goal post, which a programmer might synthesize for the agent based on an understanding of how to successfully score. Agents in *Experiment 2* had an additional nine features representing information about the goalie. The agents had four discrete actions and a total of six continuous parameters: dash, with continuous parameters of power and degrees; turn, with a continuous parameter of degrees; tackle, with a continuous parameter of degrees; and kick, with continuous parameters of power and degrees.

We trained the agents using a simple reward, i.e., five points for a goal. Since the agent received no other feedback for positive behavior, such as advancing the ball to the goal or attempting a shot on the goal, we used a series of increasingly difficult learning phases, transferring the agent's learning from one phase to the next.

3.2 Training Experiences

The series of learning phases listed in Table 2 represent our attempt to train the agent to understand how its behavior affects the environment and how it can score goals. *Experiment 1*, *Phase1* is a simple task in which the agent and ball are placed at a random

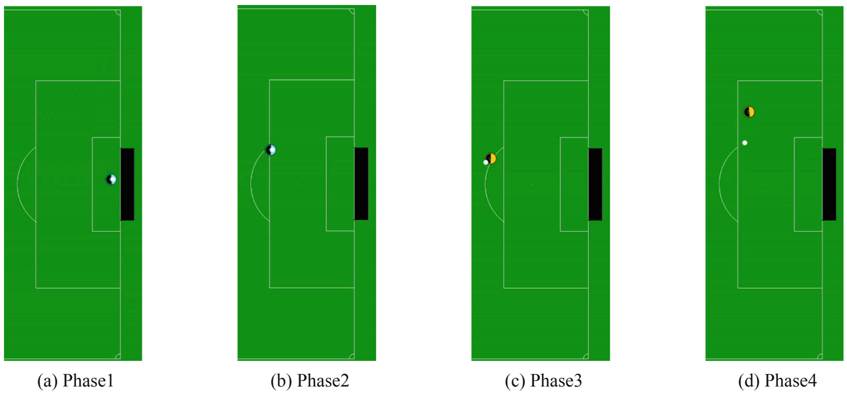


Fig. 1. Sample starting positions of *Learning Phases 1* through *4*

position directly in front of the goal. To be successful, the agent simply needs to kick the ball with any amount of power in the direction of the goal. By the end of this task, the agent should understand the effect of kicking the ball, with a marginal understanding of kick direction and power. Building on its former experience, *Phase2* places the agent with the ball at a random position near the penalty line in order to have the agent develop a greater understanding of kick direction and power. In *Phase3*, the ball is placed at a random position near the penalty line, but the agent is placed just outside its kickable range from the ball. In this phase, the agent learns to move toward the ball in order to kick it, thus it learns about movement and direction of movement. *Phase4* keeps the ball at a random position near the penalty line, but the agent is moved further away from the ball. This phase requires the agent to sustain a movement direction over time to approach and kick the ball. *Phase5* places both the ball and the agent at a random position on the field. This phase brings all of the previous learning of the agent together as the agent is expected to run across the field and, through a series of kicks, score on an empty goal. In *Phases 1* through *5* the agent is able to fully observe its state without noise (full-state information is used), regardless of what its sensors actually perceive. *Phase6* has the agent observe its environment through noisy and limited sensors (standard view used in RoboCup competitions). This phase demonstrates how effectively an agent, having learned in a fully observable simulation, can transfer its knowledge to a partially observable and noisy version of the same task. To assist with limited and noisy sensors, the agent has an underlying layer which updates an approximate world model by integrating information over multiple observations. This layer also governs the focus and direction of the visual sensor so that the agent can update and maintain its world model. This layer was not controlled by our DDPG (Fig. 1).

The first five phases are common to both experiments, with the exception of the addition of a goalie in the defended goal experiment. In *Phases 1* through *5* of the defended goal experiment, the goalie is stationary in the center of the goal. In these phases, the goalie makes no attempt to intercept the ball, but if the ball hits the goalie, the ball is "captured" and no goal is scored. In *Phase6*, the goalie is permitted to move, but because the hand-coded goalie possesses significant skill, we restrict the goalie's ability so it can move in only 10% of the server iterations. In *Phases 7*, *8*, and *9*, the goalie is allowed to move more frequently in 25%, 50%, and 75% of server iterations, respectively, and in *Phase10*, the goalie defends the goal at full capacity.

3.3 Training Implementation

In a given learning phase, an agent would explore its environment and learn to perform a task by executing multiple simulation episodes. An episode begins with starting positions as described in the previous section and ends under one of the following conditions: (1) the agent scores a goal, (2) the goalie captures the ball, (3) the ball goes out of bounds, or (4) more than 50 real world seconds have elapsed. The agent evaluates its environment and executes an action every $^1/_{10}$ of a real world second, which we refer to as an iteration. Thus, a single episode can have up to 500 iterations, though many episodes are much shorter.

We trained four agents in each of our two experiments, for a total of eight agents. In *Experiment 1*, $Agent_1$ and $Agent_2$ progressed from one learning phase to the next immediately after demonstrating a 96% success rate against the goal. $Agent_3$ and $Agent_4$ progressed to the next learning phase after completing 500,000 iterations of the current learning phase. Similarly, in *Experiment 2*, $Agent_5$ and $Agent_6$ progressed to the next learning phase after demonstrating a 96% success rate, and $Agent_7$ and $Agent_8$ progressed after completing 500,000 iterations.

3.4 DDPG Architecture Details

We employed a similar actor-critic network as in [5]. Both the actor and critic had the same network architecture consisting of four fully connected layers with 1024, 512, 256, and 128 units. Inputs to the actor network were the 59 or 68 (if a goalie is present) state features. Inputs to each neuron in the hidden layers were first processed through a leaky rectified linear unit (ReLU) with a negative slope of 0.01. The actor network had two linear output layers. The first was for the four discrete actions, and the second was for the six continuous parameters. These outputs were provided as inputs to the critic network, in addition to the 59 or 68 state features. The critic had the same hidden layer architecture as the actor network, and provided a scalar representing the approximated action-value. The actor used the feedback from the critic to update the agent's policy. The critic's update to the actor regarding the parameter gradient was bound according to the inverting gradients algorithm.

We used the Caffe Deep Learning Framework by Berkeley Artificial Intelligence Research with the Adam [7] solver using the hyperparameters identified in [5] that yielded the best results through experimentation: a 10^{-5} learning rate for the actor and 10^{-3} for the critic; 20% on-policy updates, and 80% off-policy updates; and τ of 0.001 to temper the rate of change in learning. At initialization, the agent selected all actions at random, and over the first 10,000 actions we anneal this stochastic selection linearly to 10%, favoring the best action according to our trained policy 90% of the time. We use a reward discount rate γ of 0.99.

Table 3. Performance of each agent in the empty goal experiment

	Full observability		Partial observability	
	Iter (10^3)	Goal rate	Iter (10^3)	Goal rate
[5]	~1500	1.00	N/A	N/A
$Agent_1$	710	1.00	980	1.00
$Agent_2$	820	1.00	890	1.00
$Agent_3$	2290	1.00	2620	1.00
$Agent_4$	2300	1.00	2580	1.00

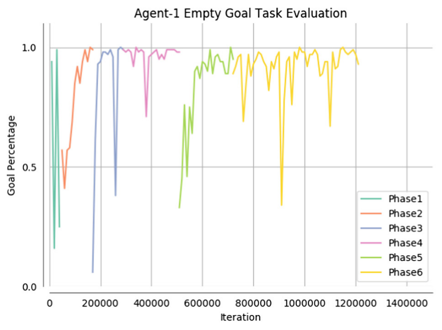
We estimate that it was approximately 1,500,000 iterations to train the agent in [5] after which it was able to consistently score against an empty goal. We note the authors of [5] did not present results for a scenario with partial observability.

4 Results

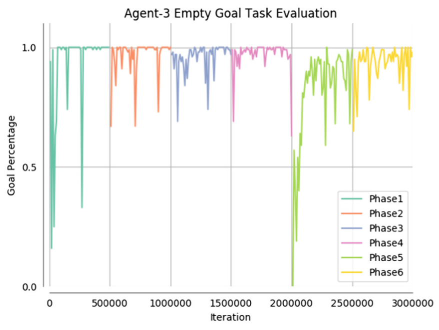
Overall, the agents trained in the empty goal experiment performed exceedingly well. Alas, those trained against the defended goal ultimately stopped learning and failed when the goalie was allowed to move more quickly (i.e., in *Phases 6–10*). We found it necessary to reintroduce a shaped reward to assist the agent's learning. Figure 2 shows the change in the agents' performance (with respect to goal percentage) as the agents experienced additional iterations. For the sake of space, the figures depict only the odd-numbered agents; the even-numbered agents had a similar performance compared to their odd-numbered counterpart. The following sections discuss our results in more depth.

4.1 Empty Goal Experiment

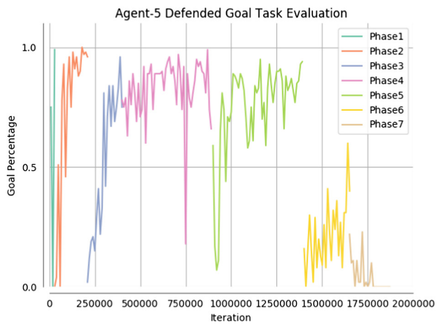
In the empty goal experiment, we found that our transfer learning phases provided a responsive learning opportunity from one phase to the next. In general, $Agent_1$ and $Agent_2$, which transferred to the next phase immediately after demonstrating success in the current phase, had an initial significant drop in performance when assigned to the new phase. These agents, however, returned to an acceptable degree of performance in the course of training (see Fig. 2a, transitions from *Phase1* to *Phase2* and from *Phase2* to *Phase3*). In *Phase5* and *Phase6*, we continued training the agents until they reached flawless performance. The iterations required to reach this level of performance are reported in Table 3. Despite the early struggles with a new phase, $Agent_1$ and $Agent_2$ learned the empty goal task more quickly than any previous benchmark. Moreover, all four agents learned the empty goal task in the fully observable scenario (*Phase5*) and were able to transfer that learning successfully to the partially observable scenario (*Phase6*).



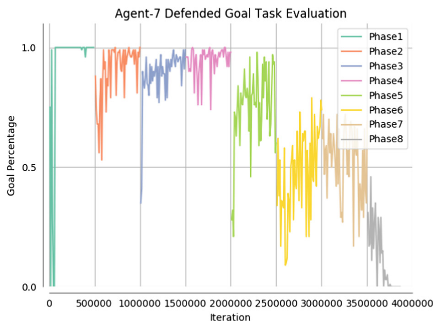
(a) *Agent*$_1$



(b) *Agent*$_3$



(c) *Agent*$_5$



(d) *Agent*$_7$

Fig. 2. Performance of agents in empty and defended goal experiments

4.2 Defended Goal Experiment

In contrast, the agents learning in the defended goal experiment had a much more difficult time. *Agent*$_5$ and *Agent*$_6$, which transitioned immediately to the next phase after becoming an expert in the current phase, initially struggled with the new phase (similar to *Agent*$_1$ and *Agent*$_2$) but were able to eventually learn the skills necessary to advance in the first five phases (see Fig. 2c). *Agent*$_7$ and *Agent*$_8$, which continued to practice the phase for a full 500,000 iterations before advancing, adapted more quickly to the new phase. After completing the first five phases in which the goalie was immobile, all four agents had significant difficulties as the goalie's capabilities increased. *Agent*$_5$ and *Agent*$_6$ struggled with the goalie at 25% capacity and were not able to continue to the next phase. *Agent*$_7$ and *Agent*$_8$ performed slightly better, but stopped learning with a goalie at 50% capacity. All four agents, when faced with a goalie that would block their goal attempts, would eventually stop attempting to score and simply run off the field. We suspect this result occurred because the agents prefer exploration behaviors when prior experience indicates the likelihood of success using previous strategies is marginal.

In summary, as the goalie's ability increased, the agents stopped attempting a shot in favor of exploring other avenues for earning a reward. We, therefore, experimented with reward shaping [15]. After training agents in *Phase5*, we provided agents with a proportional reward for moving the ball toward the goal, up to a max of three points. This artificial reward encouraged the agents to move the ball toward the goal. With this modification, all agents eventually learned to score against a skilled goalie at full ability. Table 4 summarizes the number of iterations required for each of these modified agents to reach a performance threshold, using transfer learning for *Phases 5–10* and shaped rewards for *Phases 6–10*. Remarkably, the agents quickly learned how to defeat the hand-coded goalie. The performance of most of the agents equaled or exceeded that of both the Agent2D base and the 2016 championship team *HELIOS* attacker (Table 4). With this alternative, we duplicated some of the efforts of [5], but demonstrate that shaped rewards can work alongside transfer learning.

Table 4. Performance of each agent with shaped reward

	Iter (10^3)	Goal rate
$Agent_5$	2950	0.97
$Agent_6$	2730	1.00
$Agent_7$	5210	0.96
$Agent_8$	4370	0.92
Agent2D	N/A	0.96
HELIOS	N/A	0.98

4.3 Other Observations

Early performance in a learning experiment was not a consistent indicator for how well the agents would ultimately succeed in that learning phase. For instance, the initial performance of $Agent_7$ and $Agent_8$ in *Phase5* was around 1% in terms of successful goals (Fig. 2d), but over time, their performance improved such that they were eventually scoring in over 90% of the scenarios. On the other hand, when $Agent_7$ and $Agent_8$ started *Phase8*, they were scoring 30% and 40% of the time; within 500,000 iterations, however, the performance dropped to zero, and the agents stopped attempting shots on the goal.

We also observed that more practice with a phase did not create a disadvantage when transitioning to the next phase. Indeed, those agents having more experience adapted to the new task more quickly.

We now consider our ad-hoc experiment using the shaped reward signal. After the agents learned how to successfully score consistently against the goalie, we removed the reward for advancing the ball towards the goal. Our preliminary results indicate that the agents continued to attempt goals against the goalie, and, despite no incentive for unsuccessful shots on the goal, they maintained a high level of performance.

Overall, there appears to be promise in utilizing the transferred learning approach in complex domains with large and continuous action and state spaces. Our approach, however, does introduce additional variables. For example, we believe the success and rate of learning is dependent upon the specific learning phase tasks chosen. If the gap between phases is too great, or if the learning phase tasks were performed in a different order, we expect the results would be different, e.g., agents either failing to learn the task or learning the task more quickly. With respect to the defended goal task, we suspect one issue is the learning phases have too large a gap between them. That is, in our experiment the goalie develops skill more quickly than the striker can learn. If we could improve the skill of the goalie more gradually, perhaps the striker would learn to score through successive learning phases.

5 Conclusions and Future Work

We have applied a novel approach of transfer learning in the RoboCup Soccer domain by training agents via a series of learning phases of increasing difficulty. Using a DDPG with replay memory, we are able to select optimal actions in continuous action parameter space to meet a simple, single objective: score on a goal.

We found the agents exhibit no negative transfer in the range of iterations we're using for training. In fact, agents with more experience in the environment adapted more quickly to novel tasks than agents with less experience. On the empty goal task, our agents exceeded previous benchmarks in reinforcement learning, suggesting that transfer learning of increasingly complex tasks can reduce learning time when compared to all other known techniques. In addition, skills learned in a fully observable state are transferable to a partially observable state when the agent maintains an approximate world model integrated over several observations.

Our results with the defended goal task are less positive. Specifically, when the rewards received were too scarce, the agents stopped attempting to earn the reward and instead turned to exploring the model. We found that in those situations, it may be necessary to introduce reward shaping to help direct the agent's behavior toward successful strategies. After the agent has learned a policy which yields a higher probability of success, it appears as though we can remove the "training wheels" without ill-effect. That is, after the agent develops a policy yielding a higher probability of success, the agent continues to learn, successfully scoring while it improves its performance, even without the shaped reward.

As for future research, we would like to explore whether longer training times beyond 500,000 iterations would prove effective in the defended goal task. We would also like to adapt our transfer and reinforcement learning approach to train a striker agent and a goalie in tandem. After some initial orientation to the state and action space, we would like to see if it is possible to bring both a novice attacker and a novice goalie online together to learn as they compete against one another.

Further development toward principled methods for determining whether an agent can successfully learn via learning phases would be valuable to the reinforcement learning community. As it is, early performance in a target learning task provides little to no indication as to whether or not the agent will demonstrate continued learning over the long term. Finally, we'd like to explore other domains for reinforcement learning in continuous parameter action space. Pushing the DDPG model into more complex domains will help to further our understanding of the opportunities and limitations of the DDPG reinforcement learning model.

References

1. Akiyama, H.: Agent 2D Base Code 3.1.1 (2012). https://osdn.net/projects/rctools/releases/p4887
2. Allis, L.V.: Searching for Solutions in Games and Artificial Intelligence. Ponsen and Looijen, Wageningen (1994)
3. Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In: IAPR/IEEE Workshop on Visual Behaviors, Seattle, Washington, June 1994
4. Chinchalkar, S.: An upper bound for the number of reachable positions. Int. Comput. Chess Assoc. J. **19**, 181–183 (1996)
5. Hausknecht, M., Stone, P.: Deep reinforcement learning in parameterized action space, pp. 1–12. arXiv preprint arXiv:1312.5602, February 2016. arXiV:1511.04143v4
6. Kalyanakrishnan, S., Liu, Y., Stone, P.: Half field offense in RoboCup soccer: a multiagent reinforcement learning case study. In: Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (eds.) RoboCup 2006. LNCS, vol. 4434, pp. 72–85. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74024-7_7
7. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412:6908, December 2014
8. Lillicrap, T., et al.: Continuous control with deep reinforcement learning. In: Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, May 2016
9. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
10. Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game AI research and competition in StarCraft. IEEE Trans. Comput. Intell. AI Games **5**(4), 293–311 (2013)
11. Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement learning for robot soccer. Auton. Robots **27**(1), 55–73 (2009)
12. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, June 2014
13. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup soccer keep-away. Adapt. Behav. **13**(3), 165–188 (2005). 0301
14. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
15. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: a survey. J. Mach. Learn. Res. **10**, 1633–1685 (2009)
16. Tesauro, G.: Temporal difference learning and TD-Gammon. Commun. ACM **38**(3), 58–68 (1995)
17. Torrey, L., Walker, T., Shavlik, J., Maclin, R.: Using advice to transfer knowledge acquired in one reinforcement learning task to another. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS, vol. 3720, pp. 412–424. Springer, Heidelberg (2005). https://doi.org/10.1007/11564096_40
18. Tromp, J., Farnebäck, G.: Combinatorics of go. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 84–99. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75538-8_8