Transfer of Driving Behaviors Across Different Racing Games

Luigi Cardamone, Antonio Caiazzo, Daniele Loiacono and Pier Luca Lanzi

Abstract— **Transfer learning might be a promising approach to boost the learning of non-player characters' behaviors by exploiting some existing knowledge available from a different game. In this paper, we investigate how to transfer driving behaviors from The Open Racing Car Simulator (TORCS) to VDrift, which are two well known open-source racing games featuring rather different physics engines and game dynamics. We focus on a neuroevolution learning framework based on NEAT and compare three different methods of transfer learning: (i) transfer of the learned behaviors; (ii) transfer of the learning process; (iii) transfer of both the behaviors and the process. Our experimental analysis suggests that all the proposed methods of transfer learning might be effectively applied to boost the learning of driving behaviors in VDrift by exploiting the knowledge previously learned in TORCS. In particular, transferring both learned behaviors and learning process appears to be the best trade-off between the final performance and the computational cost.**

I. INTRODUCTION

In the recent years, the computer games market is growing fast both in the number of published titles and in the complexity of the games. To boost the game development it is possible to use third-party packages like graphics engines and physics engines. In this context, it became important to have some techniques to speed-up also the development of the Artificial Intelligence (AI) of the game, i.e., the behaviors of the non-player characters (NPCs). There is the need of some sort of *pluggable AI*, i.e., a set of algorithms independent from the specific game and from its representation, that can work on different games to achieve some common tasks (e.g. navigating a 3D environment, driving, aiming, etc.).

A first step towards a pluggable AI has been done with the General Game Playing (GGP) competition [10]. In this competition an agent has to play an unknown game described in first order logic. Although good results have been achieved, the agents can only play games that are deterministic and with perfect information (like board games), where it is easy to simulate the future states of the game given the description of the rules. These conditions usually do not hold in many computer games where the agent has to control continuous outputs and the future game states cannot be predicted.

A different approach for boosting the development of the AI of NPCs is to re-use an existing AI already developed in a similar game. This type of problem is part of the area known as *Transfer Learning* (TL) [26], [23]. In this paper, we show how transfer learning can be applied to exploit the knowledge gathered in one source domain for accelerating the learning in a target domain. We take into account two racing games with many differences related to the physics engine and the game dynamics: TORCS and VDrift. In particular we focus on a driving behavior represented by a neural network and evolved using neuroevolution. We study how the knowledge already available for TORCS can be used to accelerate the learning of a driving behavior for VDrift. In general, transfer learning, involves the transfer of policies or functions that represent a particular behavior. In this work, we use the term transfer learning in a broader sense since we study also how it is possible transfer the fitness function and the evaluation mechanism across different domains. In our experimental analysis we compare three different approaches for transfer learning: (i) transfer of the learned behaviors, by copying the existing behaviors from the source domain to the target domain; (ii) transfer of the learning process, by replicating in the target domain the evaluation mechanism and the fitness designed for the source domain; (iii) transfer of both the behaviors and the process, by replicating the evolutionary process of the source domain and by performing an incremental learning using the source behavior as a seed. The results shows that all the three approaches work and the adaptation technique has the best trade-off in term of driving performance and evolution time. This paper is organized as follows: in Section II, we review the related works about racing games and transfer learning; in Section III, we present the neuroevolutionary approach used; in Section IV, we describe the racing games used for the experiments; in Section V, we describe the sensors and the actuators of the driver; in Section VI, we present three approaches for applying transfer learning and the results achieved; finally in Section VII, we present the conclusions of the work.

II. RELATED WORK

Some of the early works on racing games were performed by Togelius et al. using a simple java-based 2D car racing game. In [27], [28], Togelius et al. investigated several schema of sensory information (e.g., first person based, third person based, etc.) and studied the generalization capabilities of the evolved controllers.

More recently, Cardamone et al. [6], [5] applied neuroevolution to evolve a driving and an overtaking behavior, in a sophisticated 3D car racing simulator (TORCS). The overtaking issue was consider also in [18] using a modular fuzzy architecture and in [15] using Reinforcement Learning.

Luigi Cardamone (cardamone@elet.polimi.it), Antonio Caiazzo (antonio.caiazzo.work@gmail.com), Daniele Loiacono (loiacono@elet.polimi.it) and Pier Luca Lanzi (lanzi@elet.polimi.it) are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. Pier Luca Lanzi (lanzi@illigal.ge.uiuc.edu) is also member of the Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA.

In [9] evolutionary computation is applied to find the optimal racing line between the shortest path and the maximum curvature path of the track. Several works, like [16] [7] [29], applied imitation learning both to learn from human data and to generate believable drivers, i.e., drivers with a human driving-style. In [31] and in [8], genetic algorithms were applied to optimize the car setup using respectively the games Formula One Challenge and TORCS. Some of the most recent works [19], [3], [17] on racing games originate from the submission to the Simulated Car Racing Championship [14], an international competition organized by Loiacono et al. where the goal is to develop the best driver for TORCS able to race alone and against the other competitors.

Finally, the computational intelligence techniques have been also applied to some commercial racing games. In *Colin McRae Rally 2.0 (Codemasters)* a neural network is used to drive a rally car, thus avoiding the need to handcraft a large and complex set of rules [11]: a feedforward multilayer neural network has been trained to follow the ideal trajectory, while the other behaviors ranging from the opponent overtaking to the crash recovery are programmed. In *Forza Motorsport (Microsoft)* the player, can train his own *drivatars*, i.e., a controller that learns the player's driving style and that can take his place in the races.

In this work, we exploit the knowledge available in one game (TORCS) to accelerate the learning in an another game (VDrift). This type of problem belongs to the area known as *Transfer Learning* (TL) [26], [23]. This area studies how the knowledge learned on a *source* task can be exploited to speed up the learning on a new *target* task. Several methods of transfer learning have been specifically devised to be applied in Reinforcement Learning (RL) or Temporal Difference (TD) learning [21]. Such methods either focus on the transfer of value functions (e.g., [22], [24]) or experience instances (e.g., [13]). Only few works in the literature (see [23] for an overview) focused on transferring the learned policies and, thus, can be applied to the methods that perform a search in the policy space (like NEAT [20]). One of the earliest examples of transfer learning in on-line evolution is due to Lanzi [12] who used populations of condition-action-prediction rules (called classifiers) evolved for simple tasks to seed the evolution of solutions in similar problems involving more complex state-action spaces.

The works that are more related to our study are [25] and [4]. In [25], Taylor et al. show that is possible to transfer the knowledge evolved using NEAT, when the target task has either a different state space or a different action space. In particular, they used the population evolved by NEAT in the source task to seed the initial population for the target task. In [4], transfer learning is applied to transfer driving behaviors within the racing game TORCS. In particular, a driving policy evolved on one source track was than used to accelerate the learning on a different target track. At the best of our knowledge there is no work in the literature which studies transfer learning of policies across similar computer games.

III. NEAT

In this study, we focused on Neuroevolution with Augmenting Topology or NEAT [20], one of the most successful and widely applied neuroevolution approach. NEAT is specifically designed to evolve neural networks without assuming any a priori knowledge on the optimal topology nor on the type of connections (e.g., simple or recurrent connections). NEAT is based on three main ideas. First, in NEAT the evolutionary search starts from a network topology as simple as possible, i.e. a fully connected network with only the input and the output layers. Complex structures emerge during the evolutionary process and survive only when useful. Second, NEAT deals with the problem of recombining networks with different structures through an *historical marking* mechanism. Whenever a structural mutation occurs, a unique *innovation number* is assigned to the gene representing this innovation. This mechanism is then used to perform the recombination and to identify similarities between the networks without the need of a complex and expensive topological analysis. Third, NEAT protects the structural innovations through the mechanism of *speciation*. The competition to survive does not happen in the whole population but it is restricted to niches where all the networks have a similar topology. Therefore, when a new topology emerges, NEAT has enough time to optimize it. Again, the historical marking is used to identify the niches by using the innovation numbers of the genes for measuring the similarities between different topologies. To prevent a single species from taking over the population, the networks in the same niche share their fitness. Then, in the next generation more resources will be allocated to the species with greater fitness.

IV. Open Source Racing Games

In this section we review all the open-source car racing games currently available. We focus only on open-source racing games because the possibility of modifying the source code is essential for any research work. Finally, we motivate why in this work we focus on the game VDrift.

A. TORCS

The Open Racing Car Simulator (TORCS) [1] is a state-of-the-art open source car racing simulator which provides a sophisticated physics engine, full 3D visualization (Figure 1), several tracks and models of cars, and different game modes (e.g., practice, quick race, championship, etc.). The car dynamics is accurately simulated and the physics engine takes into account many aspects of racing cars such as traction, aerodynamics, fuel consumption, etc. The game distribution includes several programmed bots which can be easily customized or extended to build new bots. TORCS has a big community of users which contribute to the game providing additional resources like tracks, textures, and car models. TORCS has been used for several competitions both in the academic [14] and non-academic [8] field.

Fig. 1. A screenshot from TORCS.

B. VDrift

VDrift [30] is one of the best open-source car racing game. The developments was started in 2005 by *Joe Venzon*. Its main main features are represented by the high quality 3D visualization and by a good playability. For the physics simulation it uses Bullet[1], an open source physics engine featuring 3D collision detection, soft body dynamics, and rigid body dynamics (also used in commercial games like for example *Grand Theft Auto IV*[2]). Thus, the physics simulation is very accurate. Particular attention is devoted to the simulation of the loss of traction that result in the *drift*. The only aspect missing is a model for the damage suffered by the cars. The community around VDrift is smaller that the TORCS one and it is hard to find additional resources.

Fig. 2. A screenshot from VDrift.

C. Others

Among the other open-source car racing games we cite *Speed Dreams*, *RARS*, and *Vamos*.

Speed Dreams [2] is a fork of TORCS, which aims to implement exciting new features to make a more enjoyable game for the player, as well as constantly improving visual and physics realism. The main claim of the Speed Dreams project is a faster release process (1 or 2 releases per year) thanks to a more active community of users.

RARS (Robot Auto Racing Simulator) is an old 3D racing simulator designed to enabled pre-programmed AI drivers to race against one another. RARS was used as the base for TORCS.

Vamos is an automotive simulation framework with an emphasis on thorough physical modeling. It was used as a base for developing VDrift.

D. Discussion

The goal of this work is to compare the results already available for TORCS [6] with a similar game type but with enough differences in the game dynamics and in the physics simulation. We discarded RARS since its code was used as a base for TORCS. We discarded Speed Dreams as it is a fork of TORCS and share the same physics engine (even if there can be some upgrades). The only remaining choices are Vamos and VDrift. At this point, VDrift was a quite obvious choice since it is a more sophisticated version of Vamos.

TORCS and VDrift have completely different physics engines: the first one has a physics engine specifically designed for that racing game; the latter use a general purpose 3D physics engine (Bullet). Also the simulation granularity is different: TORCS updates the game state 50 times every game second while VDrift updates the game state 100 times per seconds. These differences lead also to different game dynamics in terms of drivability, steering sensitivity and achievable speeds. Other differences are introduced by the fact that the car models available in VDrift are different from the one used in [6].

To have an idea of the differences between TORCS and VDrift we compared the performance of the bots available in each game driving in the same tracks. Some tracks are available for both games since they use the same file format (a common standard for 3D objects called AC3D [3]). Table I reports the lap times achieved by the bot Inferno in TORCS and the bot VDrift-AI in VDrift. It is possible to see that the lap times achieved in VDrift are higher than in TORCS with an average overhead of 32%. Part of this overhead is due to the different programmed policy used by the bots. The different driving policy is not enough to justify such a big difference. From a deeper analysis it seems that the overhead is mainly due to the different accelerations achievable and so to the maximum speed and the braking capabilities.

V. Driver Model

The goal of this work is to transfer a driving behavior across two different racing games. The transfer learning techniques applied heavily rely on the model used for representing the target behavior. In this work, we want to exploit the results already available in TORCS from one of our previous work [6]. Thus, in this work we will use the same

[1]http://bulletphysics.org/
[2]http://www.rockstargames.com/IV/

[3]http://www.inivis.com/ac3d/man/ac3dfileformat.html

TABLE I
COMPARISON BETWEEN THE VDRIFT-AI AND ONE OF THE BEST BOT OF TORCS.

Track	VDrift-AI(s)	Torcs-AI (s)	Overhead
Dirt-3	100.87	65.5	35%
Ruudskogen	92.67	65.9	29%
Suzuka-2005	170.15	114.6	33%

behavior model and the same sensors/actuators model used [6]. In this way we can focus on the mechanism for adapting the behavior between two games with different dynamics rather than on the mapping between different representations of the behavior or of the sensors model. In particular, in [6], the driving behavior is represented through a neural network evolved using NEAT while the sensors and the actuators model is based on a representation already used in many works (e.g., [28]) and in the Simulated Car Racing Competition [14].

The sensor model is essentially based on an egocentric representation of the track. It is implemented through a rangefinder sensor. The rangefinder casts an array of beams, with a range of 100m, around the car and returns the distance from the car to the track edge (Figure 3). Each beam is identified by the angle relative to the car axis. The available beams range from -90^o (left side of the car) to $+90^o$ (right side of the car).

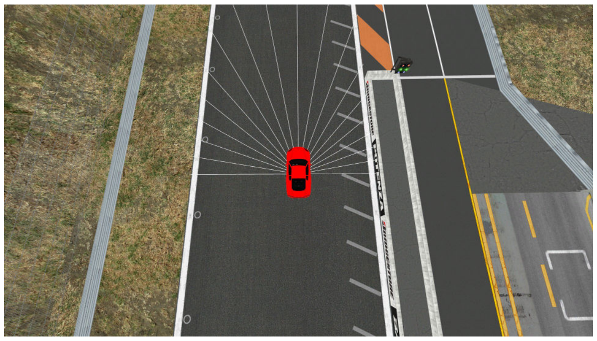
Fig. 3. A screenshot of the rangefinder sensor.

In particular the input sensors used by the neural network are 8: (i) six rangefinder sensors to perceives the track edges along the directions { - 90^o , - 60^o , - 30^o , + 30^o , + 60^o , + 90^o }; (ii) an aggregated frontal sensor computed as the biggest value among the ones returned from the three rangefinders along the directions {- 10^o , 0^o , + 10^o }; (iii) the current speed of the car.

The actuators used by the car are the usual ones: the steering wheel, the acceleration, the braking pedal and the gear stick. Similarly to what done in [6] the neural controller, has two continuous outputs in the range [-1,1]. The first one is used to control the steering wheel. The second one is used to control the gas and brake pedals as follows. If the output is less than zero it is assigned, as a positive value, to the brake effector, resulting in a braking command. Otherwise it is assigned to the throttle effector, resulting in an acceleration command. In addition, when the car is in a straight segment of the track, i.e., when the frontal sensor does not perceive the track edge within 100 meters, the gas pedal is set to 1 and the brake pedal is set to 0.

Finally, to deal with the gear shifting, we used a programmed policy: while it is quite complex to develop a good policy that controls the speed and the trajectory of the car, an effective gear shifting policy can be quite easily programmed. The controller is also provided with a scripted recovery policy to be used when the car goes outside of the track.

VI. EXPERIMENTAL RESULTS

In this section, we describe three methodologies for transferring the knowledge acquired in one domain (TORCS) for learning a driving policy in another domain (VDrift). For each methodology, we present the experimental results and finally we discuss which is the approach with the best trade-off.

A. Using an existing Driver

The simplest way for transferring the knowledge from TORCS to VDrift is to copy the given behavior as it is, from the source domain to the target domain. In [6] the driving behavior is represented through a neural network that can be easily loaded and simulated in another game. From [6] we selected the neural network with the best driving performance (i.e., with the highest fitness). This neural network is the core of the driver that resulted to be one of the best entries of the Simulated Car Racing Championship 2009 [14] and the winner of the 2008 IEEE CIG Simulated Car Racing competition. Thus we will refer to this particular neural network with the name *Champion2009*. This driving behavior was evolved in the track *Wheel-1*, depicted in Figure 4, which is available only for TORCS. To simulate the neural network exactly in the same way we used the same version of NEAT employed in [6].

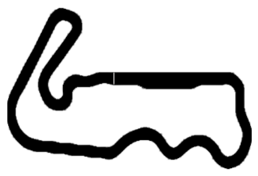
Fig. 4. Track Wheel-1 used in [6] to evolve the driver Champion2009 for TORCS.

To allow this neural network to work properly, we implemented in VDrift the same sensor model used in [6], as described into Section V. To test the performance of Champion2009 when driving in VDrift we selected 4 tracks with increasing difficulty and the car *Ferrari 360 Modena*. The 4 tracks used for the experiments of this work are reported in Figure 5. The tracks Dirt3, Ruudskogen, Suzuka

TABLE II
LAP TIMES OF CHAMPION2009 VERSUS VDRIFT-AI IN VDRIFT.

Track	*Champion2009* (s)	*VDrift-AI* (s)
Dirt-3	106.55	100.87
Ruudskogen	117.50	92.67
Suzuka-2005	227.47	170.15
Sepang	271.47	159.37

are available both for TORCS and VDrift while the track Sepang is available only for VDrift.

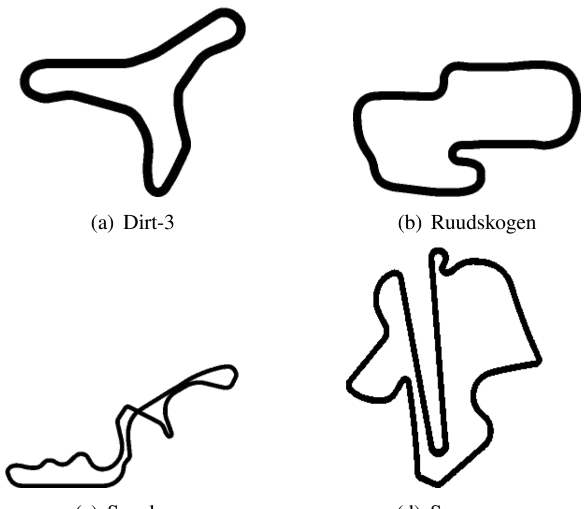
(a) Dirt-3 (b) Ruudskogen (c) Suzuka (d) Sepang

Fig. 5. VDrift tracks used for the experiments reported in this paper.

Table II reports the performance of the Champion2009 when drives in VDrift. The performance are measured as the lap time in seconds achieved during the second lap. To evaluate the driver on the entire track, we added a simple recovery policy just in case the car goes off-road. The neural network cannot recover the car by its self since the rangefinder sensor is not defined outside the track. As reference, Table II reports also the lap time achieved by the best AI available for VDrift. In the track Dirt-1, the Champion2009 drives well and its lap time is similar to the one achieved by VDrift-AI. In Ruudskogen, the performance is worse but the Champion2009 is able to drive well for the entire track. In the most difficult tracks, Suzuka and Sepang, the car goes outside the track in some critical points. The recovery policy is used to take the car on track. The lap times are much higher since each time the recovery policy is activated, it takes around 20 seconds to bring the car on track. If we consider that there are many differences between the dynamics of the two games (as pointed out in Section IV), it is surprising how well the driver behaves in VDrift: it drives well on two medium difficulty tracks and for a large part of the two most difficult tracks.

B. Evolution From Scratch

In general, applying neuroevolution to learn a target behavior in a given game is not an easy task. It requires a good knowledge of the problem in order to design an effective evaluation process of the individuals. This usually involves: (i) the definition of a simulation in the game to extract some measures of the target task; (ii) the definition of the fitness which is a combination of the extracted features. Finally, it may be necessary to tune the parameters of evolutionary algorithm, that in some cases can affect drastically the final performances. This activity, is usually a trial and error process that is quite time consuming. In this second approach we want to exploit the knowledge acquired in [6] for TORCS to speed-up the design of the evolutionary process for learning a driver in VDrift. Thus, the goal is to understand if the neuroevolutionary approach proposed for TORCS is general enough and can be applied to another game without any tuning. In this context, the term Transfer Learning is overloaded since it is usually associated to transfer a policy rather than to the design of the learning experiment and the fitness function.

In this experiment we replicated the same experimental setup and the same fitness used in [6] without any tuning to the coefficient of the formula or to the parameters of the evolutionary algorithm. To train the driving behavior we evolved a population of 100 networks for 150 generations with the standard C++ implementation of NEAT [20]. The performance of each neural network is evaluated using a simulation in VDrift. During the simulation, the neural network is used to control the car to drive on a given track. The evaluation ends when the a lap is completed or when the simulation time is greater than a certain threshold. When the simulation is over, the fitness of the neural network is computed as follows:

$$F = C_1 - T_{out} + C_2 \cdot \bar{s} + d,$$

where T_{out} is the number of game tics the car is outside the track; $\bar{s}$ is the average speed (meters for game tic) during the evaluation; d is the distance (meters) raced by the car during the evaluation; C_1 and C_2 are two constants introduced respectively to make sure that the fitness is positive and to scale the average speed term. C_1 was set 1000 like in [6]. Instead C_2 was set to 6000, the double of the value used in [6], just because TORCS engine simulates 50 tics per second while VDrift simulate 100 tics per second.

Table III reports the results of the evolution performed in four different tracks. For each experiment we report the performance of the best driver evolved. In particular we report the fitness, the lap time, the seconds spent outside the track and percentage of track covered by the driver. From the results, it is possible to see that in every track was possible to evolve a driver that can cover the entire track without going outside. The lap times are generally better than the previous case. The only exception is track *Sepang* where to avoid going off-road, the evolution generated a very safe and slow driver. Thus, the experimental design presented [6] worked without any modification for VDrift and allowed to avoid a trial and error activity to setup an effective learning experiment.

TABLE III

EVOLUTION FROM SCRATCH IN VDRIFT.

Track	*Laptime (s)*	*Fitness*	*Outside (s)*	*Raced*
Dirt-3	111.96	4412.10	0.09	100%
Ruudskogen	110.60	5835.08	0.00	100%
Suzuka-2005	256.92	8167.85	1.04	100%
Sepang	403.68	7342.87	0.22	100%

TABLE IV

SEEDED EVOLUTION IN VDRIFT.

Track	*Champion2009*	*Champion Adaptation*	*Improvement*
Dirt-3	3711.16	4387.49	18%
Ruudskogen	5269.72	6005.55	14%
Suzuka-2005	4172.36	8657.21	107%
Sepang	2379.44	7742.95	225%

C. Adapting an Existing Driver

The third approach that we investigated is somehow a combination of the previous approaches. We take the neural network evolved in TORCS and we apply another run of neuroevolution to adapt that behavior to the new dynamics of VDrift. To do so, we follow the same approach outlined in the evolution from scratch with the same experimental setup, the same evaluation mechanism and the same fitness. While in the evolution from scratch the initial population is random, in this case the initial population is seeded with slightly muted copies of the neural network of the Champion2009.

The results are reported in Table IV where we report the fitness of the Champion2009 (the seed), the fitness of best driver evolved and the percentage of the improvement. The evolution is always able to improve the performance of the Champion2009. While in the medium difficulty track the improvement is moderate in the most difficult track the improvement is very high.

D. Discussion

In this section we want to compare the performance of the three approaches presented so far. Table V reports the lap time, the average speed and the time spent outside the track for all the drivers developed with the considered methods. It is possible to see that the behavior of the Champion2009 is very aggressive: it drives very fast but often can go off track. This is particularly true in the last track where its aggressive policy drives very fast but it is not able to brake effectively before the two narrow turns after the two main straights (see Figure 5(c)). This can happens because of the difference in the simulation engine between TORCS and VDrift. On the other side the best drivers evolved from scratch present a very cautious driving policy that never goes off road. While in the first three tracks the drivers achieve competitive lap times, in the last track a too cautious driving policy results in a very high lap time: to avoid going off-road in the two strongest turns of Sepang the driver learns a sub-optimal policy which drives very slow during all the straight. The drivers evolved using adaptation show the best trade-off between the aggressive driving policy of the Champion2009 and the cautious one of the drivers evolved from scratch. So, the aggressive and fast policy of the Champion2009 is somehow adapted to the new dynamics generated from the VDrift engine. The best drivers evolved with adaptation achieve the best lap time in 3 tracks over 4 and only in one case a driver goes off-road for more than one second.

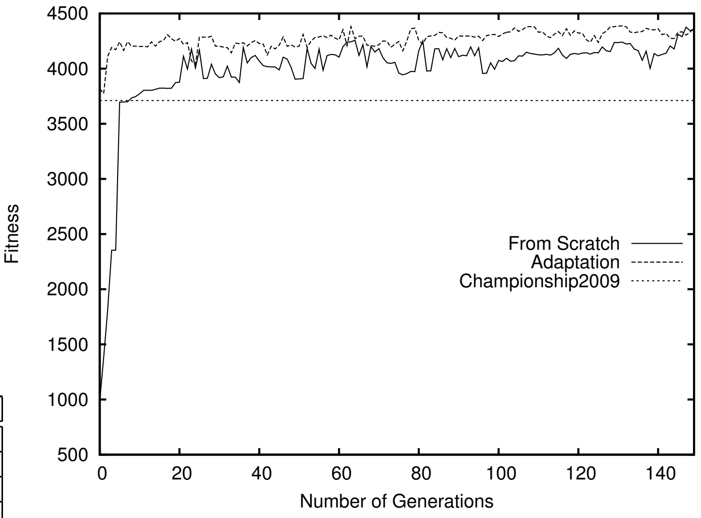

Fig. 6. Fitness trend of the evolutionary processes From Scratch and with Adaptation in track Dirt-3. For comparison it is also reported the fitness of the Champion2009.

It is also interesting to consider the evolution time for finding a good driver. The evolution time is very important since, each run can last from 10 to even more than 100 hours [4] depending on the length of the track and the performance of the individuals. This happens because each fitness evaluation requires to simulate a race of one lap which takes few seconds, but in some cases can be longer than 10-20 seconds [5]. As an example we report the performance on the track Dirt-3. Figure 6 shows the fitness trend of the evolutionary process From Scratch and with Adaptation in the track Dirt-3. For comparison it is also reported the fitness of the Champion2009. The process with Adaptation takes only 4 generations to find a competitive driver with a fitness of 4192.47. The process from scratch takes 24 generations to find a driver with fitness 4176.88, and 37 generations for a champion of 4191.4.

From our analysis it is possible to see that the drivers evolved with adaptation represent the best trade-off in terms of driving performance and learning speed (it can save several hours of computation). Thus, adapting for few generations a pre-existing model seems to be a promising way for transferring the knowledge across similar domains.

[4]The computer used for the experiments has a Core i5 and 4 GB of RAM

[5]The simulation is performed using a customized version of VDrift where we disabled the graphical visualization. In this way, it was possible to get a simulation around 10 times faster.

TABLE V

COMPARISON BETWEEN THE CHAMPION2009 AND THE BEST DRIVERS EVOLVED FROM SCRATCH AND WITH ADAPTATION.

Track	Statistics	Champion2009	From Scratch	Adaptation
Dirt-3	outside (s)	7.39	0.09	**0.00**
	lap time (s)	**106.55**	111.96	112.64
	avg speed (Km/h)	**74.7**	71.1	70.7
Ruudskogen	outside (s)	4.91	**0.00**	0.17
	lap time (s)	117.50	110.60	**100.91**
	avg speed (Km/h)	97.8	104.0	**114.0**
Suzuka-2005	outside (s)	28.38	1.04	**0.00**
	lap time (s)	250.48	256.92	**189.00**
	avg speed (Km/h)	84.1	82.0	**111.5**
Sepang	outside (s)	76.38	**0.22**	2.27
	lap time (s)	271.47	403.68	**268.86**
	avg speed (Km/h)	73.6	49.5	**74.3**

VII. CONCLUSIONS

In this paper we investigated how to transfer the knowledge between two complex domains. In particular we took into account two types of knowledge: (i) the behavior of an agent represented by a neural network; (ii) the fitness and the experimental design used to evolve the agent for a particular task. We considered two racing games with enough differences regarding the simulation engine: TORCS and VDrift.

We investigated three different methodologies for transferring knowledge. The first one consists in copying directly the behavior evolved in TORCS to VDrift. The second one consists in replicating in VDrift the same fitness and the same experimental setup used to evolve a driver for TORCS. In this case we do not transfer a behavior, but the process for evolving the desired behavior. The third one is a combination of the previous methods: the behavior evolved in TORCS, is adapted in VDrift applying another run of neuroevolution. In this way the original behavior is adapted to the different dynamics of the target game.

For the experiments we used 4 tracks: two with medium difficulty and two with very high difficulty. Three of these tracks are available for both games while one is available only for VDrift. The results show that all the three methods work. The behaviors obtained using the three techniques are quite different: (i) the first method yields a very fast and aggressive driver that in the most difficult tracks can go off-road; (ii) the second one evolves very safe drivers that in some tracks can be much slower; (iii) the third one generates drivers that are both fast and safe. The third method shows a good trade-off in terms of performance and evolution time, since only few generations are necessary to obtain a good driver. In conclusion, adapting an existing solution is a promising method for transferring knowledge between two similar domains.

This work can be extended in several ways. A first extension can be to consider other types of game like for example First Person Shooters (FPS). A second extension can involve pairs of different game types: for example transferring a driving behavior to a FPS for navigating a 3D environment. In this case, different sensors and actuators can be involved and it can be necessary to learn also the mapping between the source and the target input/output model. Finally, a very interesting extension is to study the transfer knowledge problem with different representations of the behavior (fuzzy, RL, etc.).

REFERENCES

[1] The open racing car simulator. http://torcs.sourceforge.net/.

[2] Wolf-Dieter Beelitz, Xavier Bertaux, Brian Gavin, Eckhard M. Jaeger, Kristf Kly-Kullai, Gbor Kmetyk, Enrico Mattea, Jean-Philippe Meuret, Haruna Say, and Andrew Sumner. Speed dreams - a free open motorsport sim and open source racing game. http://www.speed-dreams.org/.

[3] Martin V. Butz and Thies D. Lonneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 317–324, Sept. 2009.

[4] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(3):176 –190, sep. 2010.

[5] Luigi Cardamone. On-line and off-line learning of driving tasks for the open racing car simulator (torcs) using neuroevolution. Master's thesis, Politecnico di Milano, 2008.

[6] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.

[7] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning drivers for torcs through imitation using supervised methods. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 148–155, Sept. 2009.

[8] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Applying cooperative coevolution to compete in the 2009 torcs endurance world championship. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, jul. 2010.

[9] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 388 –394, aug. 2010.

[10] Michael Genesereth and Nathaniel Love. General game playing: Overview of the aaai competition. *AI Magazine*, 26:62–72, 2005.

[11] Jeff Hannan. Interview to jeff hannan, 2001. http://www.generation5.org/content/2001/hannan.asp.

[12] Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 337–344, Orlando (FL), July 1999. Morgan Kaufmann.

[13] Alessandro Lazaric. *Knowledge transfer in reinforcement learning*. PhD thesis, Politecnico di Milano, 2008.

[14] D. Loiacono, P.L. Lanzi, J. Togelius, E. Onieva, D.A. Pelta, M.V. Butz, T.D. Lonneker, L. Cardamone, D. Perez, Y. Saez, M. Preuss, and J. Quadflieg. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131 –147, jun. 2010.

[15] Daniele Loiacono, Alessandro Prete, Pier Luca Lanzi, and Luigi Cardamone. Learning to overtake in torcs using simple reinforcement learning. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1 –8, jul. 2010.

[16] Jorge Munoz, German Gutierrez, and Araceli Sanchis. Controller for torcs created by imitation. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 271–278, Sept. 2009.

[17] E. Onieva, D. A. Pelta, J. Alonso, V. Milanes, and J. Perez. A modular parametric architecture for the torcs racing engine. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 256–262, Sept. 2009.

[18] Enrique Onieva, Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Overtaking opponents with blocking strategies using fuzzy logic. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 123 –130, aug. 2010.

[19] J. Quadflieg, M. Preuss, O. Kramer, and G. Rudolph. Learning the track and planning ahead in a car racing controller. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 395 –402, aug. 2010.

[20] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[22] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[23] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[24] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[25] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 156–163, May 2007.

[26] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press, 1996.

[27] J. Togelius and S.M. Lucas. Evolving robust and specialized car racing skills. pages 1187–1194, 0-0 2006.

[28] Julian Togelius and Simon M. Lucas. Evolving controllers for simulated car racing. In *Proceedings of the Congress on Evolutionary Computation*, 2005.

[29] N. van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber. Robust player imitation using multiobjective evolution. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 652–659, May 2009.

[30] Joe Venzon. Vdrift: a cross-platform, open source driving simulation. `http://vdrift.net/`.

[31] Krzysztof Wloch and Peter J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. In *Proceedings of Eighth International Conference on Parallel Problem Solving From Nature*, pages 702–711, 2004.