Evolutionary Optimization of a Neural Network Controller for Car Racing Simulation

Damianos Galanopoulos, Christos Athanasiadis, and Anastasios Tefas

Department of Informatics, Aristotle University of Thessaloniki
Box 451, 54124, Greece
tefas@aiia.csd.auth.gr

Abstract. In this paper a novel method for car racing controller learning is proposed. Car racing simulation is an active research field where new advances in aerodynamics, consumption and engine power are modelled and tested. The proposed approach is based on Neural Networks that learn the driving behaviour of other rule-based bots. Additionally, the resulted neural-networks controllers are evolved in order to adapt and increase their performance to a given racing track using genetic algorithms. The proposed bots are implemented and tested on several tracks of the open racing car simulator (TORCS) providing smoother driving behaviour than the corresponding rule-based bots and increased performance using the evolutionary adaptation.

Keywords: TORCS, Neural networks, Genetic algorithms, Evolutionary Optimization.

1 Introduction

Nowadays, video games are becoming more and more important, as a hot consumer product, as well as a great opportunity for research in artificial intelligence. The main goal is to offer fun to the player. In previous years this goal has been achieved partly through the visual realism and interesting game scenarios. But every video game player knew that the current AI in the games was way far from the actual human behaviour. When we are playing a game versus one or more NPC (non-character player) we can easily realize that we are not playing versus another human because either the other player is too simple to beat, figuring out a specific efficient strategy, or the AI is so complicated that the human loses every time. Artificial intelligence in computer games is infused into non-playable characters with a view to giving the human player the illusion of a clever human opponent. Initially we have to create a NPC that imitates the behaviour of a human player [3]. However, we have to bear in mind that the NPC must also have the ability to adapt depending on the current state and environment and the current opponents in the game. Computational intelligence methods can be implemented to deal with the adaptation task. Such methods can be retrieved from evolutionary algorithms and Neural Networks [2].

I. Maglogiannis, V. Plagianakos, and I. Vlahavas (Eds.): SETN 2012, LNAI 7297, pp. 149–156, 2012.

Previous approaches to car racing were already developed for the forerunner of TORCS, the robot auto racing simulator (RARS) [7]. For example, Stanley et al. [8] developed a car racing strategy that depended on range-finders and developed a sensory-motor mapping with the incremental neural evolution of augmenting topologies (NEAT) approach.

The Cognitive BOdySpaces for Torcs-based Adaptive Racing (COBOSTAR), which was developed by B. Martin V. Butz and Thies D.Lonneker [9], is divided in two parts: on-track optimization and off-track optimization. Actually, they implemented heuristic functions for mapping input data into decision. These functions were different when the controller was on-track or off-track.

Another approach has been proposed in [4]. The idea behind this bot was to have a driving architecture based on a set of simple controllers. Each controller is applied as a separate module in charge of a basic driving action. Two important modules are the learning module, which finds where the bot have to increase or reduce its speed, and the opponents management module, which adapts the agent behaviour when the opponents are close.

Luigi Cardamore's approach [10] consists of an evolved neural network, implementing a basic driver behaviour, compounded with code for basic tasks such as the start, the crash-recovery, the gear change, and the overtaking. They use neuroevolution of augmented topologies(NEAT)[8] for predicting target speed and target position for a given input configuration. The implemented fitness functions is the error between the actual values and the predicted ones.

The combination of a fuzzy logic module with a classifier module and a finite state machine is proposed [11] with a view to tackling the great variety of TORCS commands from Diego Perez and Yago Saez.

The basic idea in our controller is to use Neural Networks for the decisions of the driver bot. That is, the bot is trained using back-propagation in a single hidden layer NN. The training data are collected by using other bots that are using rule-based AI to control the car. That is, the proposed bot is trained by other bots. We expect to enhance the performance of the rule-based bots since the NN will smoothly imitate their behaviour. Moreover, in order to improve its performance, the proposed bot has the ability to evolve its NN using evolutionary algorithms. That is, the proposed bot, firstly, learns from other bots how to drive and adapts the weights of its NN using genetic algorithms in order to improve its lap time for given racing tracks. The novelty of our method is the use of the time performance as the fitness function, which changes the NN weights depending on the current lap time.

2 TORCS Environment

The open race car simulator (TORCS) provides an open source car racing environment with a very realistic simulator that has a sophisticated physic engine which takes into consideration real car racing issues such as fuel consumption, collisions or traction. Besides that, TORCS offers a very realistic game-play and graphics. It is a well designed simulator which can be compared with the finest

race game titles. Additionally, TORCS provides a very flexible and simple client-server architecture. Server stands for the game's functions, and client stands for the car agent handling. The above characteristics justify why it has been used for research purposes in the scientific community, especially for solving the simulated racing car challenge task. In 2011, three different challenges were held; the EVO-2011 in Torino, ACM GECOO-2011 in Dublin and the IEEE CIG-2011 in Seoul.

The first approaches appeared, were hand-crafted rule-based and only slightly optimized on several aspects. Our approach in this challenging task is based on Neural networks combined with evolutionary algorithms. We tried to tackle the challenge of imitating other players or car agent behaviour.

We have used two different controllers, as trainers of the proposed bot. The first controller is a simple rule-based controller (named "Simple Driver"), which is offered with the client server architecture, and the second one is called "Simplicity". The proposed controller is based on a feed forward ANN (Artificial Neural Networks) that was trained with data generated by other controllers using back-propagation. The final step in our method was to adopt modified genetic algorithms in order to achieve better results.

TORCS consists of a server component that supports the general TORCS setup [1] and returns information sensors about the controller and the track. The client component uses these information to apply its strategy. The controllers (clients) run as external programs and communicate with the server with UDP connections. At each game tic the controller receives sensor data that corresponds to the car's current state and its surrounding environment (the tracks and the components). The controller has to calculate four output parameters (the wheel steering, the gas pedal, the fuel level and the break pedal). Its strategy depends on the current input (information from sensors). In the proposed method we use learning methods in order to build output commands.

The sensors novelty is in that they do not contain the whole track information, but they carry only simulated local information instead. In particular, the available sensors are an angle sensor, which specifies the current angle between the car direction and the track axis, the current speed in longitudinal and transverse axes of the car, 19 range sensors, which sample the free track space in front of the car and they are only valid while on track, 36 opponent sensors, which notice opponents around the car, the current engine speed in rounds per minute, the current gear, the track position with respect to the track edges, and the current rotation speed of the four wheels. Moreover, there is further racing information available including the current lap time, the damage of the car, the distance from the start line along the track line, the total distance raced, the amount of remaining fuel, the last lap time and the standing in the race. For the car control, there is a gas pedal and a brake pedal, gear shifting, and steering values available.

Thus, the agent strategy cannot receive information about tracks morphology (such as which curve comes next) and it depends only on the local information (the sensors they were described before).

3 Proposed Approach

Our agent was trained by applying two rule based controllers in order to collect data fast and for the different sensors cited in the previous section. Data collection was made for several game tracks. More specifically, we use two different tracks, named F-Speedway and CG-Speedway No1 for data capturing. For each track we used approximately 10.000 input states for each controller. Each input state is composed of 9 input data that represent the controller's environment and two output values that correspond to controller acceleration and wheel steering. These two outputs are in fact the desired outputs of the designed Neural Networks. For each output "label", we created a separate Neural Network. That is, one neural network decides about the acceleration and the other decides about the steering. Values are normalized in $[-1, 1]$.

3.1 Off-line Neural Network Learning by Imitation

The first part of our work was to imitate controller's behaviour by implementing Artificial Neural Networks. Initially we obtained the data from the controller we wanted to imitate (trainer) and a neural network was trained with the back-propagation algorithm. We used neural networks with a specific structure [5]. Hecht-Nielsen [6] proved that one hidden layer is sufficient to approximate a bound continuous function with a specific mean square error. Thus, we chose to generate our neural network with one hidden layer with 50 neurons. The number of neurons was selected using cross-validation in the training set in order to avoid over-training. Using the previously described structure we reduced the time and code complexity. As an activation function for the NN we have used the logistic function $f(t) = \frac{1}{1+e^{-t}}$ and the learning rate h was set to 0.5 and the momentum m to 0.1. The Neural Network weights determine the controller decisions, which will define the behaviour of our bot in real-time gaming.

Using the back-propagation algorithm we were able to train neural networks to return the desired output data. Thus, we can use the outputs of the trained neural networks in order to control the acceleration and steering of the car. In fact we are using the following expression:

$$s = \frac{2}{1 + e^{(speed_{currnet} - speed_{target})}} - 1$$

where $speed_{current}$ is the speed in the previous game tic and the $speed_{target}$ is the output of the Neural Network every game tic. This expression takes values in $[-1, 1]$. When this expression returns 1, it means that our controller has to fully accelerate and when it returns -1 controller has to fully break. Similarly, we utilize the same expression for the wheel steering. The objective was to succeed convergence in bots behaviour (between the controller that was used as trainer and our agent). The next step is to modify the NN weights in order to improve controller's behaviour in unknown tracks and simultaneously to increase agent steering smoothness and reduce lap time. To do so, we use adaptation of the NN weights using genetic algorithms.

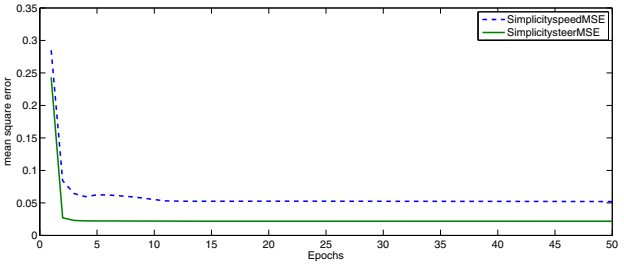


Fig. 1. NN convergence

3.2 On-line Learning Using Genetic Algorithms

Genetic algorithms(GA) are heuristic methods that try to mimic the natural process of evolution. Genetic algorithms became popular through the work of John Holland in the early 1970's. Actually, GA try to generate solutions to optimization and search problems. These solutions are evaluated by a fitness function. To do so, the genetic operations of mutation and cross-over attempt to find an optimal solution. The operation of mutation and cross-over of GA are described in [12].

In our work we employed GA in order to improve the performance of the neural networks trained in the first stage. Thus, the GA try to optimize the fitness function by searching for better weights for the trained NNs. We use as a fitness function the $currentLapTime$, which can be returned from the game with the goal to evaluate the performance of current weights. That is, the trained NNs are evolved trying to achieve the fastest lap.

When the evolution begins we store three different NN weight sets for acceleration control and three for steering control and the lap time that the controller achieved with those weights. These initial weights that will be used for producing the offsprings are produced by training the NN with different learning parameters (e.g., learning rate, initialization, etc). Then, we modify these weights by using mutation and cross-over. Mutation is simulated by adding uniform random noise (transported to $[-1, 1]$) to certain probabilistically selected weights. The weights are normalized in $[-1, 1]$. We add multiplicative noise on the selected weights as follows:

$$weights_{new} = weights + weights * \frac{noise}{k}$$

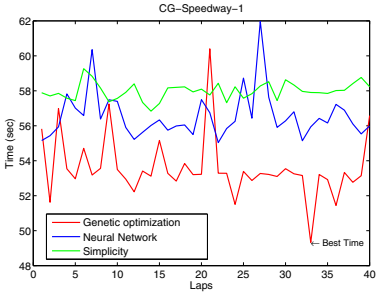
, where k is a normalization constant which affects the impact of noise. That is, the selected weights are slightly increased or decreased according to the noise value.

The crossover operation is implemented by combining probabilistically selected solutions (NN weight sets) according to their fitness. The selected NNs exchange randomly selected weights in order to create their offsprings. The agent uses the new weights in three consecutive laps and the time achieved is used as fitness function. The selection procedure keeps only the three best offsprings

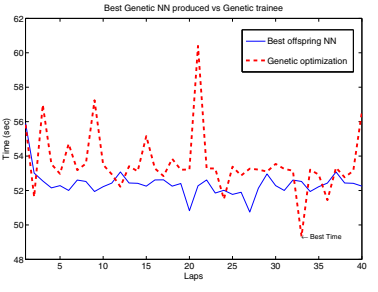
weights and replace the previous weights accordingly. Our anticipation here is to optimize the current lap time every three-laps. This is expected since GA at least keeps the initial weights in case of non improvement. The procedure is repeated every three laps. We have noticed that the lap time is not continuously decreasing. When we test the same weights multiple times the returned lap time is not exactly the same. Thus, we expect some small fluctuations in the convergence. We let our agent to repeat the above procedure many times. For every track, we have tested the controller for 40 laps.

4 Experiments

In our experiments we used several tracks that TORCS offers. For the sake of convenience we are going to present results for the following tracks Oval B-Speedway, F-Speedway, CG-Speedway No1. Two benchmark observation comparisons to test and reveal differences between trainer and trainee are the current lap time, that indicates the speed performance, and the distance from the track axis, which indicates the wheel steering performance. Our expectations in those two observations are, firstly, our bot to achieve better lap times and, secondly, to accomplish a more smooth driving trajectory. In Figures Fig. 2(a), Fig. 2(b), Fig. 3(a), Fig. 3(b) we can see the performance of the trainer and the trainee in lap time and smooth driving behavior in two TORCS tracks, namely, CG-Speedway-1 and F-speedway.



(a) CG-Speedway1



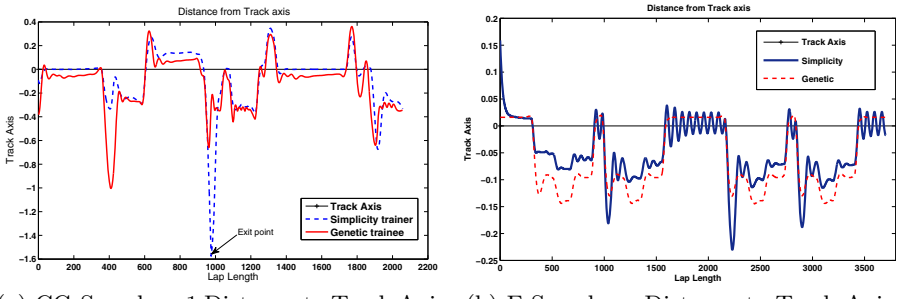
(b) Best offspring NN vs Genetic optimization

Fig. 2.

In Figure 2(a) the lap-time for 40 laps on the CG-Speedway-1 track is shown. The three curves correspond to the performance of the Simplicity driver (trainer), the trained Neural Network that learns from the behavior of the Simplicity driver and the best adapted NN for each generation during the genetic optimization for 40 laps. In Figure 2(a) it is clear that the the trained NN achieves better lap times than the rule-based Simplicity driver. The performance is improved on average by two seconds. Moreover, it is obvious that genetic optimization further improves the performance of the NN trainee and, on average, the performance is

improved by 5 seconds compared to the Simplicity driver. The best performance in a single lap is observed in the 34th lap during the genetic optimization. That is, the trained NN has learned the driving behavior from its rule-based trainer and enhanced his performance due to smoother driving trajectories as we will see in the following. Moreover, by genetically evolving its weights during the actual driving procedure the proposed NN based driver further improved the lap-time and during the genetic optimization the global minimum lap time has been reached in the 34th lap.

The NN that achieved the best performance during the genetic optimization is selected as the best offspring and evaluated again for 40 laps. In Fig. 2(b) we compare the performance of the best NN's for each generation during the evolutionary optimization with the overall best NN that was produced on the 34th lap and was the outcome of the evolutionary optimization. It is clear that the genetically optimized NN achieves better results. However, for the tested 40 laps it cannot reach the best time achieved during the optimization. This is easily explained since the performance of the trained drivers is not always the same and we search for a bot that is on average better than the trainer.



(a) CG-Speedway1 Distance to Track Axis (b) F-Speedway Distance to Track Axis

Fig. 3.

In Figures Fig. 3(a), Fig. 3(b) we test the smoothness of genetic trainee and simplicity driver trainer in two different tracks. What we can notice, especially in Fig. 3(b), is that the trainee overcame the rule-based decision behaviour and simulated a smoother driving trajectory. Moreover, an interesting noticeable observation is highlighted in Fig. 3(a). At the 980 meter of track there is a sharp turn where the rule-based trainer came out of the track in every lap. The neurogenetic bot manage to stay in the track bounds and save time achieving a smoother driving trajectory at that point.

5 Conclusion

In this work we proposed a computational intelligence method using the combination of neural networks with genetic algorithms so as to develop an agent which can be trained while playing The Open Race Car Simulator (TORCS),

attempting, this way, to achieve higher performance compared with the performance of its trainer bot. Our results show that we managed to create a controller that enhances the behaviour of its trainer in regards to acceleration, achieving better lap -times from its trainer and steering by implementing a smoother driving trajectory. Moreover, by utilizing evolutionary adaptation we managed to improve even more the performance of our bot and to gradually achieve the best performance.

References

1. Torcs the open race car simulator, `http://torcs.sourceforge.net/`
2. Petrakis, S., Tefas, A.: Neural Networks Training for Weapon Selection in First-Person Shooter Games. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) ICANN 2010, Part III. LNCS, vol. 6354, pp. 417–422. Springer, Heidelberg (2010)
3. Laird, J.E., Van Lent, M.: Human-level AI's Killer Application: Interactive Computer Games. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 1171–1178 (2000)
4. Onieva, E., Pelta, D.A., Alonso, J., Milanés, V., Pérez, J.: A modular parametric architecture for the TORCS racing engine. In: Proceedings of the 5th International Conference on Computational Intelligence and Games (CIG 2009), pp. 256–262. IEEE Press, Piscataway (2009)
5. Haykin, S.S.: Neural networks and learning machines, 3rd edn. Multi Layer Perceptron, pp. 152–258 (2008)
6. Hecht-Nielsen, R.: Theory of back-propagation neural network. In: Proc. of the Int. Conf. of Neural Networks I, pp. 593–611 (1989)
7. Robot auto racing simulator (2006), `http://torcs.sourceforge.net`
8. Stanley, K., Kohl, N., Sherony, R., Miikkulainen, R.: Neuroevolution of an automobile crash warning system. In: Genetic and Evolutionary Computation Conference, CECCO 2006, pp. 1977–1984 (2006)
9. Martin, B., Butz, V., Lonneker, T.D.: Optimized Sensory-motor Couplings plus Strategy Extensions for the TORCS car Racing Challenge. In: IEEE Symposium on Computational Intelligence and Games, CIG 2009, pp. 317–324 (2009)
10. Cardamone, L., Loiacono, D., Lanzi, P.L.: Applying cooperative coevolution to compete in the 2009 TORCS Endurance World Championship. In: Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
11. Perez, D., Saez, Y., Recio, G., Isasi, P.: Evolving a rule system controller for automatic driving in a car racing competition. In: Symposium on Computational Intelligence and Games (CIG), pp. 336–342. IEEE (2009)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems, 2nd edn. MIT Press, Cambridge (1992)