Developing a Competitive TORCS Controller using Evolutionary Strategies

Group 38: Arran Lyon* and Fiona Lippert**

1 Introduction

The Open Racing Car Simulator `TORCS` [1] offers a platform for artificial intelligence research where complex agents can be developed within a fast-paced and competitive environment. The challenge in this project is to use nature-inspired computing methods to develop a driver with an ability to process limited localised input data of it's immediate surroundings to make quick decisions in a race setting. Our approach starts with a basic controller with limited driving capabilities, then through the use of evolutionary methods we refine its skill and with the introduction of a second network we expand its repertoire of techniques to become a competitive and cooperative driver[1].

2 Methodology

2.1 The Basic Controller

The first fundamental step towards an autonomously driving car is to create a simple controller that enables the car to stay on an arbitrary track. For this task we use an Echo State Network (ESN) that transforms the continuous sensor data stream into appropriate driving commands.

We opted for an ESN for three main reasons. Firstly, as a recurrent neural network that relies on a complex reservoir, it has a non-linear short-term memory [2] which is expected to be beneficial for a car driving in a dynamic environment; previous carstates and actions should have an influence on what decision to make next to facilitate smooth actions such as steering. Secondly, for the network's ease and speed of training; as the reservoir connections remain unchanged, only the readout weights need to be trained. Finally, for applicability of evolutionary algorithms (EA) [2]; the separation between reservoir and readout together with the inexpensive training facilitate a supplementary learning process where both reservoir weights and internal structure can be evolved easily.

To obtain useful training data for the initial ESNs, we extracted the sensor data and corresponding command instructions from a pre-bundled `TORCS` driver. After trailing various bots we chose the driver `damned` for it's simple driving strategy that our network found easiest to mimic with it's given inputs. Data is collected from multiple laps on five different tracks that appear to cover an appropriate variety of track characteristics.

* MSc. Computational Science, University of Amsterdam, ID:11800003
** MSc. Computational Science, University of Amsterdam, ID:11799846
[1] Source code is available at `https://bitbucket.org/FionaLippert/ci_project/src`

Following the procedure discussed in [3], the reservoir weights are initialized randomly, using a reservoir of *size* = 50, a high sparsity of 80% to obtain slower dynamics, and *spectral radius* = 0.9 to assure the echo state property. We include teacher forcing into the ESN architecture, to enable the network not only recollect the history of sensor inputs but also the previous driving actions.

For learning the output weights, we feed the training sequences to the ESN. The inputs comprise information about forward speed, distance from track center, angle between car and track axis, and 19 range finders. To prevent the ESN from inefficient concurrent use of accelerator and brake, these commands are compressed into one output in range $[-1, 1]$. The second output value dictates the steering. For each of the tracks, reservoir states are initially reset to zero and the first 5 time steps are treated as washout time, being ignored in the subsequent learning process. The optimal readout weights are then computed according to [3]. For simplicity, we implemented an automatic transmission system to handle gear changes.

Collecting training data from a competent driver results in lacking experience with crashing and recovery. Therefore, we implement a basic hard coded recovery driver that is capable of returning the car back on track, headed in the correct direction, if the car spends too much time off-road or not moving. As the ESN is reliant on previous state information, data is continuously fed before it regains control after recovery to ensure a smooth handover. For stability reasons, a speed limit of 110kph is imposed.

2.2 Evolving the Basic Controller

Facing the unsatisfactory skills of the simple ESN controller, we apply evolutionary computing techniques to improve and refine its performance. EAs are stochastic search methods that do not rely on domain knowledge [4] and hence, they are well-suited for the task of ESN reservoir initialization for which only rough guidelines exist [3]. We evolve both reservoir weights and structure, but maintaining the size from 2.1. The optimization comprises two phases: First, an off-line EA generates a set of high potential solutions. These results are used as heuristic initialisation of an on-line evolution stage for refinement. The inexpensive ESN training allows learning the readout weights before every evaluation of a newly emerging network.

Off-Line Evolution The off-line EA design rests on Evolutionary Strategies (ES) which has been successfully applied to learn neural network weights [5]. The detailed algorithm setup is presented in Table. 1. Initially, individuals contain uniformly random weights and random numbers of zero-weights to assure a variety of reservoir structures. During reproduction, mutation adjusts existing connections with probability p_{m}, whereas cross-over introduces new structural combinations at rate p_{cx}. For each evaluation, we train the respective ESN by presenting the same training sets as in Sect. 2.1. The resulting controller is simulated on three unknown tracks for 60 seconds each. Then, we determine the fitness to be maximised as a combination of rewards depending on the distance

raced, and penalties for severe deviations from the track center, large angles between car and track orientation, causeless stops, and for leaving the track.

Serving as an intelligent seeding for the on-line EA, this first stage is designed to push the entire population towards higher performance instead of seeking for only one optimum.

On-Line Evolution To overcome the potential of over-fitting to particular track sections and to encourage fine-tuning of the network we use an externally centralised on-line approach [6] to introduce and evaluate networks in ever changing situations. In the course of an ongoing `TORCS` race with four drivers, a concurrently running 'evolver' script conducts the evolutionary process. It repeatedly selects a network at random from a gene pool, applies mutation, and injects the result into one of the racing cars to evaluate. Evaluated controllers are reintroduced to the gene pool via tournament selection [7]. The evolver determines the lowest rated network within a random tournament and replaces it by the evaluated one. The entire evaluation procedure occurs over seven different tracks of various difficulty ratings to ensure a diverse range of conditions.

Table 1 details the parameters used during the on-line evolution stage, where there are smaller perturbations on non-zero weights to make finer adjustments. p_{wm} is the likelihood of each non-zero weight being perturbed, and p_{sm} determines the rate at which the structure is mutated where connections are either added or removed from the reservoir. The fitness is calculated by the car during the race, being rewarded for staying close to the center line, for keeping the car angled within 15 degrees of the track's axis, and for it's distance raced since takeover. If the car either leaves the track or stays still for 3 seconds then a large penalty is applied and the evaluation terminates. The recovery system returns the car to the track to ensure a fair evaluation of the next network.

Table 1: Evolutionary Algorithm Design

	Method	Parameters		
		Off-Line	On-Line	MLP
Representation	weight matrix			
Cross-over	uniform	$p_{\text{cx}} = 0.2$ $p_{\text{ind}} = 0.1$	N/A	N/A
Mutation	Gaussian	$p_{\text{m}} = 0.75$ $p_{\text{ind}} = 0.05$ $\sigma = 0.3$	$p_{\text{wm}} = 0.1$ $p_{\text{sm}} = 0.3$ $\sigma = 0.1$	$p_{\text{m}} = 0.95$ $p_{\text{ind}} = 0.25$ $\sigma = 0.4$
Parent selection	random			
Survivor selection	tournament	size = 10	size = 10	size = 5
Generational strategy	(μ, λ)	$(20, 80)$	steady state	$(20, 60)$

2.3 Racing Against Opponents

Following the subsumption architecture approach presented in [8], we design a multi-layer perceptron (MLP) as an additional controller component that aims

to handle opponents in the car's immediate vicinity without abandoning the previously acquired driving skills.

This component is activated if opponents are within a 50m radius and adjusts the output of the first layer controller based on supplementary sensor information on opponents positions. Feeding the ESN outputs together with 36 opponent sensor values to the MLP, it predicts deviations to be added to the ESN outputs, which in turn is biased by the modified previous output to produce smooth transitions. The speed limit is temporarily raised to encourage overtaking.

To overcome the drawbacks of popular supervised training methods [5], such as being prone to local optima and relying on training data, which is unavailable for our particular problem, we apply a variant of the off-line EA from Sect. 2.2 to find optimal MLP weights. The EA settings are adjusted in a way that enhances exploration, because in contrast to the ESN evolution, no additional learning is involved that pushes the EA individuals towards feasible solutions. Further, the evaluation is modified to a race against 6 bots starting in front and 6 bots behind to assure diverse confrontations. The associated fitness function is calculated based on rewards for successful overtaking and high speed during MLP control, together with penalties for lost positions and high levels of damage. Further, we evaluate the distance raced at three different time points. Here, the goal is to encourage the evolution of appropriate strategies without incorporating too specific behavior control.

2.4 Team Cooperation Racing

Aiming at collaborative team racing with two identical controllers, we incorporate two complementary cooperation strategies.

First, inspired by an ant's indirect communication method in route-finding [9], the leading car leaves 'pheromones' for other team mates to communicate basic information of the course ahead. For the ant, this stigmergy indicate a shorter route back to the colony, and for our cars they indicate points on the track that need special care. In this way any team member behind the leader (and indeed the leader himself on successive laps) knows to slow down on the approach to one of these pheromones.

Second, based on communication about positions within the team, the driver behind experiences an attraction towards approaching opponents. We motivate the car behind to be oriented towards the track position of approaching opponents by shifting the ESN input variable representing the distance to the track center dependent on the opponent's distance. Additionally, the level of acceleration is reduced if the opponent drives directly behind. This results in impeding competing cars to overtake, which in turn promotes the leading team mate.

3 Results and Discussion

The evolutionary ESN optimisation successfully refined the rather weak basic network into a more stable controller that is capable on numerous previously unseen tracks (see Fig. 1, Tab. 3 (a)). However, as Fig. 2 illustrates, this refinement stage did not find an optimum solution and advances could still be made.

During the MLP evolution interesting competitive behaviors emerged. Without having explicitly taught desired strategies, the controller learned to attack its opponents, and to find its way through a crowded field. Nevertheless, this behavior results in increasing instability which is why performance estimation of this component is restricted to short-term results.

Team strategy has been observed in a race with many opponents, and from simple tactics and basic communication the lead driver is able to improve it's position compared to with no team-mate, as shown in Tab. 3 (b). Disappointingly however, the final network did not learn an effective braking strategy, making the imposed speed limit of 110kph necessary to ensure lap completion.

Possible improvements would be the delegation of steering and speed commands into two independent networks. A more sophisticated multi-objective evolution strategy could single out specific behaviors and areas of improvement for this multi-network approach. Performance may also be improved with the addition of principle component analysis of the many range sensors. We speculate a network could better respond and learn with the dimensional reduction of this information.

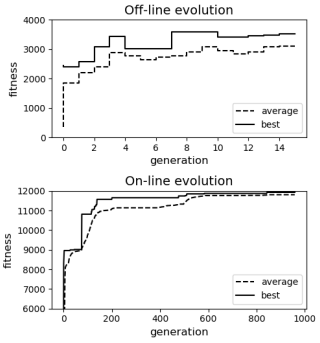

Fig. 1: Both evolution stages exhibit typical EA characteristics

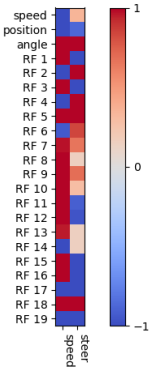

Fig. 2: The correlation between the inputs and their contribution to the outputs in the driving ESN. While this diagram does not describe the temporal nature of the network, it does reveal the 'logic' that developed after the off-line evolution stage. With careful study the self-correction mechanism that steers the car back into the center of the track can be observed, and how the controller will accelerate when there is clear track ahead. However, parts of this logic are not ideal; position and angle should not have direct bearing on the acceleration or brake in a symmetric system. These biases are down to the stochastic nature of evolution, and the asymmetry of the tracks in the training and evaluation.

Table 2: Improvements in the network through evolution. Lap times in races without opponents.

	cg-track-3	e-track-4	dirt-5	alpine-2	wheel-1
simple ESN	5:55:36	d.n.f.	3:00:25	d.n.f.	d.n.f.
evolved ESN (early stage)	1:54:74	4:37:89	1:27:58	3:14:53	2:52:08
evolved ESN (final stage)	1:42:66	3:52:72	1:11:67	2:25:40	2:41:46

Table 3: Overtaking (a) and team strategy (b). Opponents are 10 bots with max speed capped at 110kph for fair comparison. Starting in 6th and team mate (when applicable) in 7th position.

(a) position after 30 seconds

	e-track-4	e-road
ESN only	9th	9th
ESN+MLP	5th	6th

(b) 10 opponents (best final position)

	cg-track-3	e-track-4	e-road
solo	11th	11th	9th
team	8th	6th	3rd

References

1. B. Wymann, E. Espi and C. Guionneau. *The Open Racing Car Simulator TORCS*. v1.3.7, http://torcs.sourceforge.net/index.php, 2016
2. M. Lukosevicius and Herbert Jaeger. *Reservoir Computing Approaches to Recurrent Neural Network Training*. Preprint submitted to Computer Science Review, School of Engineering and Science, Jacobs University Bremen, 2010
3. H. Jaeger. *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. GMD Report 159, German National Research Center for Information Technology, 2002.
4. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, pages 20–24, 2003.
5. X. Yao. Evolving Artificial Neural Networks. In *Proceedings of the IEEE*, volume 87, number 9, pages 1423–1447, September 1999.
6. A.E. Eiben, E. Haasdijk and N. Bredeche. Embodied, On-line, On-board Evolution for Autonomous Robotics. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, Cognitive Systems Monographs 7, Springer, pages 361–382, 2010.
7. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, pages 84–85, 2003.
8. R.A. Brooks. *A robust layered control system for a mobile robot*. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1985.
9. M. Dorigo, M. Birattari and T. Stutzle. Ant colony optimization. In *IEEE Computational Intelligence Magazine*, volume 1, number 4, pages 28–39, November 2006.