An Application of Neural Networks to an Autonomous Car Driver

K. Albelihi[1] **and D. Vrajitoru**[1]
[1]Computer and Information Sciences Department, Indiana University South Bend, South Bend, IN, USA

Abstract— *In this paper, we present a car driving system called "Gazelle" for a simulated racing competition. For this application, we used both procedural methods and a neural network capable of learning. We hoped that using neural networks could lead the controller to derive more accurate equations for driving the car based on data acquired in the training process. We also expected that the more the network is trained, the more precisely it will calculate the driving information.*

Keywords: neural networks, intelligent systems, learning

1. Introduction

In this paper we present a study on various methods that can be applied for successfully driving a car in a simulated environment in the presence of opponents. A longer and more detailed version can be found in [1].

The interest in developing autonomous vehicles increases day by day with the purpose of achieving high levels of safety, performance, sustainability, and enjoyment. Driverless cars are ideal to use in crowded areas, on highways, and because they ease the flow of the cars. Autonomous cars can also reduce the chances of occurring accidents caused by other cars or by pedestrians. There are many research centers established around the world for developing smart systems for driverless cars, such as the Center for Automotive Research at Stanford University (CARS) [2].

In this paper we present an adaptive racing car controller called Gazelle developed within TORCS (The Open Racing Car Simulator) [3]. The TORCS system visualizes racing cars with complex graphics based on physics principles [4]. TORCS attracts a wide community of developers and users, and it is the platform for popular competitions organized every year as a part of various international conferences [5]. The program offers a server which implements races combining multiple cars on a variety of tracks. A client module can be written by the user supplying the actions of an individual car. Our driver, called Gazelle, was submitted to the TORCS competition organized by the Genetic and Evolutionary Computation Conference in 2013 [6].

In this environment, each car is controlled by a process that can access the current state of the car in the race, consisting of information about the track, the car, and the opponents [7]. Based on this information the controller makes decisions to modify the following control units:

- the steering wheel with values in the range [-1, +1] for a change in direction: -1 corresponds to -45^o while +1 to 45^o;
- the gas pedal [0, +1] for accelerating; a value of 0 will result in losing speed;
- the brake pedal [0, +1] for decelerating;
- the gearbox with possible values in the set -1,0,1,2,3,4,5,6 for choosing the gear [5].

Figure 1 shows a snapshot of the TORCS application. The upper screen displays the client car and its information such as the car's rank, the total time that the car spent from the beginning of the race, the best time that has been taken to complete a lapse, and other measurements. The lower screen shows the race from another angle that can highlight opponent cars if any are present. We can also see some statistics of the car such as gear levels and the speed of the car.

Fig. 1: A Screenshot of TORCS during the race

The work in this paper is based on the EPIC controller as presented by Guse and Vrajitoru in [5]. Epic was submitted to the GECCO 2009 competition [8]. The Epic driver is based on two components: determining the target angle for turning in each frame, and determining the target speed in the next frame. The controller calculates the target angle based on the free available distance ahead. It also provides a sharp turn detecting system which adjusts the target speed for an approaching sharp turn to keep the car inside the track. It uses Hill-Climbing techniques to adapt the speed parameters to new tracks. However, this controller lacks a component to handle opponents, and the movement along the track requires

more fluency. The Gazelle driver improves the Epic code on these two aspects.

Many approaches can be found in the literature for track prediction with the purpose of optimizing the performance. Such an example is the track segmentation approach, in which the track is divided into pieces that are classified as pre-defined types of polygons. Then the controller reconstructs a full track model from these polygons, as presented in [9]. Another controller based on the track segmentation principle is proposed by Onieva et al. [4]. The architecture of the controller consists of simple modules that control gear shifting, steer movements, and pedals positions. In addition, the target speed is adjusted by the "TSK fuzzy system". The most important aspect of this work is the opponent modifier. It controls the driving behavior in situations where an opponent is nearby by adjusting the steering controller and the braking controller immediately.

A more recent work [7] introduces a driving controller called AUTOPIA for the simulated racing car competition. It provides a full driving architecture including six separate main tasks: gear control, pedal control, steering control, stuck situation manager, target speed determination, opponent modifier, and learning module. The paper provides a simple and a powerful architecture especially for the opponent modifier using heuristic rules.

Many learning approaches are used to find the optimal path for the car to reduce the time required to complete the race. An evolutionary learning approach for this purpose is presented in [10]. Here, a self-adaptive evolutionary strategy (SAESs) is used to derive the parameters involved in determining the target speed in an efficient and easy to generalize way. This driver also lacks an opponent handling system. Another controller using an evolutionary learning system is presented in [9]. This controller uses a simple evolutionary learning approach to plan the path ahead for the car.

More recently, another learning approach has used hyper-heuristics in a real-valued mode in [11]. This system approaches the TORCS-based car system as a real valued optimization problem and studies the performance of different methodologies. These include a set of heuristics and their combination controlled by a selection hyper-heuristic framework. The study shows that hyper-heuristics perform well in the TORCS environment.

Artificial neural networks (NN) are also used as a learning system, well recognized by the computer science community and with many applications [12]. In [13], a human-like controller using NN was submitted to the 2010 Simulated Racing Car Championship. The controller builds a model of the tracks using the NNs to determine the trajectory of the car and the target speed. The NNs were trained with data retrieved from a human player. This work shows satisfying results of predicting the trajectory on new tracks; however, the target speed is slower than the human's on the same tracks.

The remainder of the paper is organized in the following way. Section 2 introduces the procedural algorithms in Gazelle. Section 3 presents the application of NNs to compute the target direction. Section 4 shows results from the experiments, and the paper ends with conclusions.

2. Procedural Drivers

We started this research from a procedural driver previously developed in [5] called Epic, submitted to the GECCO 2009 competition [8]. After several improvements that we will describe below, the new driver is called Gazelle. We compare its performance with Epic as well as with a pilot provided by the TORCS software called Simple Driver.

2.1 Epic Driver

The general algorithm of Epic consists in the following steps [5]:

- calculate the target direction and speed,
- determine the correct gear,
- calculate the target angle based on the target direction,
- calculate the acceleration and the brake based on the target angle and speed.

First, for the target direction, Epic starts by deciding if the car can continue to travel in the current direction. If the car is inside the track, close enough to the centerline, and there is enough free distance ahead, then the car can persist in the same direction. Otherwise Epic takes a new direction by modifying the steering angle to get closer to the road centerline.

Second, the target speed is computed. If the car is going almost straight, the free distance ahead is large enough, and no sharp turn is expected shortly, then the car speeds up towards the maximum value. In any other case, a large value for the target speed is set to start with, which is first scaled by the sine of the target angle for steering the angle and with the available free distance in the target direction.

Epic used a simple Hill Climbing technique to adjust several parameters that affect the control units of the car. Thus, the controller uses a dynamic adaptation mechanism to tune the racing car's behavior to a new track. If no damage has been recorded during this first lapse, the parameters used for calculating the maximal speed in each situation are incremented to make the car go faster. Otherwise every time the car gets out of the track or records damage without an opponent being close by, the pilot will keep the same values for these parameters or will decrease them to make its behavior safer.

Epic has several well-developed functions, however, it required some improvements such as handling opponents, enhancing the trajectory stability on the road, and anticipating sharp curves better.

2.2 The Gazelle Controller

The Gazelle controller consists of three components: the target direction unit, the target speed unit, and the opponent adjuster. The target direction unit controls the direction in which the car is moving. The target speed unit adjusts the speed based on the target direction, while the opponent adjuster adjusts the direction and speed based on the opponents' presence. Below we will describe each unit in more detail.

Target Direction Unit

The unit determines the target angle using the following guidelines:

If the current direction of the car is close enough to the road centerline, there is enough distance straight ahead, and the car is safely inside the track, then the car can continue in the same direction.

Otherwise, we start from the direction of the road centerline, and scan by 10 degrees in the direction in which the distance ahead increases, until we find an angle at which it decreases, or we reach the maximal turn angle of $\pm 45^o$.

If the car is too close to the border of the road or gets outside, we add a direction change to move it back inside. Currently, the borders threshold, denoted by $safelyInsideTrack$, is at 85% distance from the center of the road, to account for the width of the car. Let $trackPos$ be the current position of the car on the road, taking values between -1 and 1. If $|trackPos| > safeInsideTrack$, then the new target angle is computed as:

$$-25 * \text{sign}(trackPos)(|trackPos| - safeInsideTrack)$$

where the function sign returns -1 for a negative number, 0 for 0, and 1 for a positive number. This formula scales 25 degrees by how far the car is from the threshold. If the computed target angle already has a value of the same sign but of a larger absolute value, then this new target angle is not used because the normal method is performing the adjustment already.

If the current turning angle is good enough, we maintain it for movement continuity. This is determined by comparing the free distance ahead with the free distance 10 degrees left and right; if the distance ahead is the largest of the three values, then we can maintain the current angle. This is an addition to the Gazelle controller to improve the fluency of the car's movement.

Target Speed Unit

The target speed is computed once we know the target angle. The unit determines the speed using the following guidelines:

If we are going almost straight or on a fast curve, if the distance ahead is large enough, and if no sharp turn is coming ahead, we aim for a configurable high speed parameter called $sundayDriver$.

Otherwise the target speed starting from the $sundayDriver$ value is first scaled directly proportional with the cosine of the target angle for the change in direction and with the available distance in the aimed direction. This way, the smaller the turning angle is, the larger the speed will be. Similarly, the more distance is available ahead, the faster the car will go.

Let $safeSpeed$ be a value for the speed that we think will be safe for any curve, such as 30 km/h. Let $spaceFactor$ be the available free distance in the aimed direction normalized by the maximal sensor range (100m). The speed is computed as:

$$targetSpeed = safeSpeed + (sundayDriver - safeSpeed) * \cos(targetAngle) * spaceFactor^2$$

The resulting target speed is scaled afterwards by a factor depending on the sharpest turn in the road detected ahead, 20 degrees left and right of the aimed direction. The purpose of this is to anticipate situations where the speed needs to be reduced. Thus, a small difference in the free distance ahead represents a possible sharp turn approaching that requires a slower speed to be taken safely.

The sharp turn detection algorithm and the basic ideas in computing the speed are the similar in Gazelle to Epic, but the practical equations they comprise have been refined.

Opponent Adjuster Unit

We put more efforts into building a component for dealing with opponents because the car's performance can be improved by handling the opponents properly. As we mentioned previously, most of the controllers we discussed in the introduction don't handle the opponents well or at all. Neither the Simple Driver, the controller provided as an example by the TORCS competition, nor the Epic controller can deal with the opponents. In our opponent adjuster, if an opponent violates the chosen tolerance values of closeness as determined by the opponent sensors in each direction, then the gas/brake control and steering control will be modified to avoid the collision the following way:

If there is an opponent at a distance of 200m or less, then a test will determine if it violates the safe distance (the tolerance values) in each of the available sensor directions.

If there is an opponent in the front of the car, on the sides, or in the rear of the car within an unallowable space, the following flags are turned on, causing a reaction of the respective modules:

A *Brake flag* for an opponent in the front. This flag takes care of the sensors in the range of -40^o to 40^o [4]. If an opponent is found within an unallowable space and its speed is close to ours, the car should brake immediately by modifying the brake/accelerate value to half of the current speed. The tolerance values are shown in Table 1 and were adopted from [4].

Orientation of the Opponent Sensor	Tolerance Value
$\pm 40^o$	6m
$\pm 30^o$	6.5m
$\pm 20^o$	7m
$\pm 10^o$	7.5m
0^o	8m

Table 1: Opponents adjuster over the gas and brake action

A *Steering flag* for an opponent in the front or on the side takes care of the opponent sensors in the range of -100^o to 100^o, also adopted from [4]. An overtaking maneuver requires to modify the steering angle if the opponent violates the tolerance values. The tolerance values are shown in Table 2.

Orient. of the Opponent Sensor	Tolerance Value	Increment Value
0^o, $\pm 10^o$	20 m	$\pm 0.20^o$
$\pm 20^o$	18 m	$\pm 0.18^o$
$\pm 30^o$	16 m	$\pm 0.16^o$
$\pm 40^o$	14 m	$\pm 0.14^o$
$\pm 50^o$	12 m	$\pm 0.12^o$
$> \pm 50^o$	10 m	$\pm 0.10^o$

Table 2: Opponent sensors values for overtaking

An *Accelerating flag* for an opponent at the rear of the car driving at an equal or higher speed than ours. Increments values are summarized in the third column of Table 2.

Trouble Spots Register

This component was added in order to avoid accidents caused by mistakes in predicting the right steering angle, leading the car out of the track. In TORCS competitions, the race starts with a training lapse which allows drivers to learn the track. After that, the actual race takes place in the second lapse. Thus, we introduced the "Trouble Spots Register" detecting and storing places in the track where the car gets out of the road starting from the training lapse. In the subsequent lapses of the circuit, to avoid repeating these mistakes, we use a method decreasing the speed whenever the car is close to a trouble spot, by an amount inversely proportional to the distance to the trouble spot.

A list of "trouble spots" on the road will be stored by the Gazelle driver in a persistent memory space in order for it to be accessible at later points during the race. To achieve this, the latest position of the car on the road is stored in each frame. Then when the code detects that the car got out of the road, this position is added to the list.

In each frame, the current position of the car is compared to the trouble spots. If we are close enough to one of them, the speed will be adjusted as mentioned above. The closer we are to the trouble spot, the faster the car will decelerate. The issue arises from the fact that the visibility of the driver is limited to 100m ahead and that it is difficult to break down the speed fast enough if the situation requires it. For this reason we adopted the approach of detecting a sharp turn on a road combined with the trouble spots detector.

3. Neural Network Application

We employed a NN in the Gazelle system to determine the direction of movement of the car. We used a classic perceptron NN with one input layer, two hidden layers, and one output layer. We trained it with the back-propagation algorithm and *tanh* as the neuron activation sigmoid function. We used recommendations from literature [14], [15] and experiments to determine the number of hidden layers and the number of neurons on each of them.

In our model, the input layer consists of 5 neurons for the following input values:

- the last target angle in the previous frame,
- the current angle with the road direction,
- the lateral offset of the car with the center of the road,
- $distance1$ and $distance2$, computed as the difference between the free distance ahead in the direction of movement and the free distance ahead at an angle of $\pm 10^o$. This should indicate in which direction the road is turning.

The output of the NN consists of the target angle for steering the car. We tested two separate models for it: one with a single output neuron, and one with 5 output neurons. For the second model, we divided the output interval $[-34^0, 34^o]$ for the output into 5 intervals: $[-34^0, -17^o)$, $[-17^0, -0^o)$, $[0^0, 17^o)$, and $[17^0, 34^o]$. This is based on the observation that $\pm 34^o$ was the highest value for steering the vehicle in the data we collected for training. We assigned a neuron to each of the values -34^o, -17^o, 0^o, 17^o, 34^o, then we aggregated the output as a sum of the assigned values weighted by the neurons' output.

When training the NN, we first mapped the value of an angle to the closest interval, which identified two neurons with non-zero output. Then we assigned target values between 0 and 1 to these two neurons based on the placement of the angle in the interval.

Data Collection and Training

For this experiment, we trained the NN with data collected from the procedural Gazelle. Thus, while the car is driving with the pilot described in the previous section, we collect the input as described in each frame, and then the output angle determined by this method as intended output for the NN. The model with 5 output neurons does not need separate data collection, as the angles can be mapped to the output for each of the 5 neurons on the spot as needed.

We used three tracks offered by TORCS for our experiments: Alpine2, E-Road, and Forza, of total length 3773.47m, 3260.43m, and 5784.10m respectively. Figure 2 shows the shape of these tracks.

The raw data collected this way is not ready for training. The number of individual points we collected varies between 3765 and 13140, which is too much. Another problem we had to deal with is the uneven distribution of the data. Thus,

Fig. 2: Test tracks: Alpine 2 (left), E-Road (center), and Forza (right)

as the car goes straight for a large portion of each track, data points where the output angle is close to 0 are more numerous than any of the others. This can result in an over-specialization of the NN to these angle values.

The collected data are also non-uniformly distributed from a sequencing point of view. For the same reason as above, data points where the output angle is almost 0 also tend to appear *together* in clusters representing straight stretches of the road. This can also contribute to over-specialization.

The first operation that we performed was filtering the data to have a more even distribution. We started by dividing the interval of observations for the output angle into interval of 0.1 radians length. We isolated one special interval between [-0.01, 0.01] radians. This represents parts of the track where the car goes almost straight. The rest of the interval between -.5 to .6 radians was divided into 11 intervals of a length of 0.1 radians. We added one interval for any angle <-.5 radians and one for any angle >.6 radians. This gave us 14 intervals total.

For each of these intervals, we selected about 50 random data points from each of them. For intervals where the number of observations was less than 50, we kept all of the observations. Figure 3 shows an example of the number of data points in each interval before and after this filtering process. We can see that the middle intervals initially contained a lot more data than the extreme ones. After filtering, the data is better distributed between the intervals.

The second operation performed on the data was to randomize it. We performed a simple fair shuffle on the data before feeding it to the NN for training.

In a first phase, we tested the NN trained with the data collected from each track on the track itself. This testing is called *retrospective*. In the second phase, we selected the set of data that performed the best retrospectively, and tested the NN trained with it on all the tracks. The first evaluation shows if the NN is capable of learning. The second evaluation shows if the NN can learn from one track and then perform well on another one.

During the training process, we stored the weights of the NN in the configuration where the average error is the lowest. We called these the *best weights*. We compared the performance of these settings with the use of the weights in the NN at the end of the training process. The latter are called the *last weights*.

Finally, we tested two modes for training the NN: one

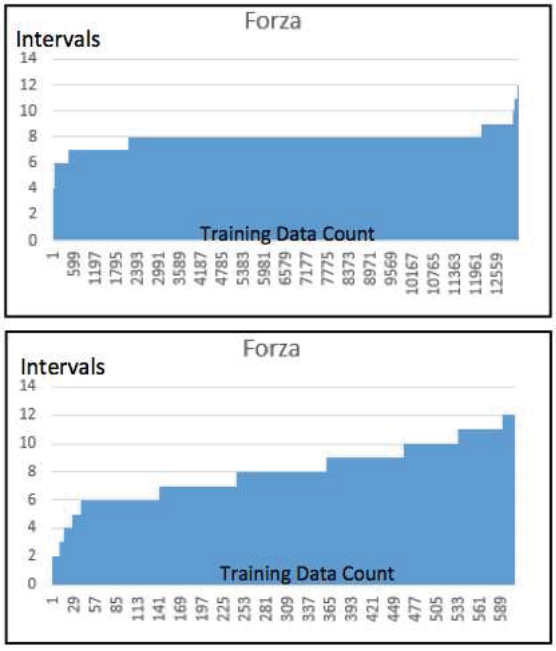

Fig. 3: Training data collected from Forza before filtering (top) and after (bottom)

where we fed the data file to the NN in 100 iterations, and one where we fed it in 500 iterations. The experiments show that the system doesn't learn significantly more in 500 iterations, though the best weights are often recorded between 100 and 200 iterations.

4. Experiments

In this section we present the results of the various models on the 5 tracks with the parameters we described above. For each track, we set the race at two lapses in all the experiments. This allows us to see the effect of any learning process taking place in the first lapse over the performance of the car in the second lapse. An example of such a process is the recording of the trouble spots. We used the value of 100 km/h as the safe speed and 150km/h as the maximum speed for all the experiments, unless specified otherwise.

At the end of the two lapses, the program itself outputs some information, such as the total time and the damage. We will also report some other measures of performance: the total damage to the car and the total distance covered by the end of the race. The more distance is covered in one lapse, the less efficient the driver is.

Table 3 shows a comparison of the procedural Gazelle with the Epic and the Simple drivers. The table contains the total time to finish two lapses (in seconds), the total distance covered by the car (in meters), and the total damage to the car. A time marked as N/A means that the car did not finish the two lapses. The races are configured to terminate early

if the car takes longer than 12 minutes to finish, or if the damage is too high.

Track	Driver	Time (s)	Distance (m)	Damage
Alpine2	Simple	7:08	7573.55	0
	Epic	6:44	7574.1	0
	Gazelle	5:33	7571.96	3314
E-Road	Simple	5:38	6547.46	0
	Epic	5:09	6546.23	0
	Gazelle	3:36	6546.05	0
Forza	Simple	6:14	5783.91	0
	Epic	N/A	5783.73	2186
	Gazelle	2:42	5784.09	1051

Table 3: Procedural Gazelle results on 3 tracks, single car

From this table we can tell that the procedural Gazelle is an improvement over Epic and the Simple Driver. It completed all 3 tracks faster than the two other drivers and with a slightly more efficient trajectory, based on the total distance. Although the damage was rather high for the Alpine2 track, it is still within the allowed amount.

The next experiment shown in Table 4 tested the three drivers in the presence of opponents on the road. For this purpose we added 3 drivers provided by the TORCS environment: berniw1, lnfHist1, and inferno10. These opponents have various levels of performance: high, medium, and low respectively, as specified in the TORCS manual [16].

Track	Driver	Time (s)	Dist. (m)	Damage	Rank
Alp2	Simple	3:06 + 1 lap	4379.32	3169	4/4
	Epic	3:06 + 1 lap	4394.89	1634	4/4
	Gazelle	3:03 + 1 lap	5919.3	3250	4/4
E-Rd	Simple	3:28	6548.62	403	4/4
	Epic	3:29	6648.71	139	3/4
	Gazelle	2:17 + 1 lap	4347.97	0	4/4
Forza	Simple	2:40 + 2 laps	2745.69	0	4/4
	Epic	2:40 + 2 laps	2745.78	0	4/4
	Gazelle	2:46 + 1 lap	5809.10	0	4/4

Table 4: Procedural Gazelle, race with 3 opponents

In this table, the total time shown in each case is the time when the race was finished. Races with multiple cars are terminated when all but one car have finished the two lapses, so that the complete ranking of the cars can be determined. The time shows how many lapses the car had not completed by the time the race was finished.

On Alpine all 3 drivers finished last and did not complete the last lapse. However, Gazelle managed to cover a much longer distance than the Simple Driver and Epic before the race was finished. The higher amount of damage could indicate that this car kept closer to the opponents and was damaged more by interacting with them. On E-road Gazelle did worse than the two other drivers, but with no damage. On Forza, Gazelle outperformed the two other drivers: it completed one lapse before the race finished, while the others didn't, and it covered about twice as much distance. Overall, we can conclude that Gazelle performs better than Epic and Simple Driver in the presence of opponents.

The next set of experiments shows the results obtained by the Gazelle using the NN for the target direction. Table 5 starts with the retrospective evaluation where the NN is trained for each track with the data collected from that same track. In this table, the time shown as "+1 lap" means that the race was terminated before the last lapse was completed, because the car ran out of time.

Track	Output	Weights	Time (s)	Dist. (m)	Damage
Alpine2	1	last	8:13	7572.02	0
	1	best	10:08	7572.78	510
	5	last	9:02	7572.1	24
	5	best	8:13	7573.58	0
E-Road	1	last	+1 lap	439	2133
	1	best	6:24	6546.11	0
	5	last	+1 lap	375.207	1454
	5	best	6:22	6545.95	40
Forza	1	last	+1 lap	2771.34	0
	1	best	+1 lap	9682.41	148
	5	last	6:03	9751.94	85
	5	best	11:44	11595.5	0

Table 5: Retrospective NN testing, 100 training iterations

Based on this table, we selected the data collected from the Alpine2 track, as the NN trained with it performed well more consistently than the others. Table 6 shows the results of this data set in 500 training iterations. As we can see, training the NN with additional iterations didn't seem to improve its performance.

Track	Output	Weights	Time (s)	Distance (m)	Damage
Alpine2	1	last	9:12	7572.8	1065
	1	best	10:00	7573.33	840
	5	last	9:26	7572.76	1604
	5	best	10:41	7572.28	0

Table 6: Retrospective NN testing, 500 training iterations

The data collected from the Alpine2 track is probably better for training because this track is particularly challenging by nature, presenting many curves of various difficulty in both directions. Figure 4 shows a close-up of this track, together with a snapshot of the car on this track.

Fig. 4: A screenshot of the Alpine2 track (left) and the car driving on it (right)

In the next evaluation stage, we have chosen the data file that showed the most promise: Alpine2, and tested the NN trained with it on all the tracks. Table 7 shows the results

of the NN trained with the data from the Alpine2 file in 100 iterations, with 5 output neurons, and using the best weights model. These settings showed the best performance in the retrospective evaluation. For easier comparison, we also added the results on the Alpine2 track itself, and the results without the NN.

Track	NN	Time (s)	Distance (m)	Damage
Alpine2	yes	8:13	7573.58	0
	no	5:33	7571.96	3314
E-Road	yes	8:31	6547.28	0
	no	3:36	6546.05	0
Forza	yes	+1lap	5150.39	1
	no	2:42	5784.09	1051

Table 7: Results of the NN trained on Alpine 2, 5 output neurons, 100 iterations, best weights

These results show that our NN can learn indeed to drive the car from the data we collected, but the performance is not as good as the procedural algorithm. However, the observed driving style of the car with a NN is smoother and less prone to sudden changes in direction than the procedural methods. With a mechanism that allows it to continue training while being in use, we hope to eventually achieve better performance than the procedural method on all aspects.

The procedural Gazelle has been submitted to the TORCS competition organized as part of GECCO 2013 [6]. We passed the qualification round and ended up in 7th place in the final round.

Future Work

We intend to extend this research in two directions. First, we are currently working on developing some methods to allow the NN to continue training during the race, so that it can better adapt to new tracks. This involves some measures to let the driver know when the trajectory is not ideal, as for example, when exiting the road, or when colliding with the road shoulders. It also means that we must be able to estimate a better direction at that moment and feed it to the NN so it can train with it. A second direction involves collecting better training data, so that the NN has a chance to outperform the procedural algorithm. Some possible better sources of data would be either data collected from humans driving the car, or data collected by observing other pilots. It is also possible to use a trajectory optimization algorithm such as the one suggested in [17].

5. Conclusion

After the experiments we presented, we can conclude that Gazelle is an improvement over Epic because the Gazelle was able to handle the opponents efficiently and it succeeded in avoiding the damage caused either by colliding with other opponents or by hitting the hard shoulders on the side of the road. Also, we can conclude that the NN can learn to drive the car efficiently and is adaptable to new tracks of any type or any shape. Even though the Gazelle took a longer time to complete the race with the NN than with the procedural methods, we can adopt a higher value for the maximum speed for the next competition to improve the total time.

Our experiments showed that it is not always useful to train the NN a lot more than 100 iteration with one set of data. In addition, using the weights in the network sampled when the lowest value of the error is achieved is better. Dividing the output value into output for several neurons proved efficient, as each output neuron has less to learn. Finally, the quality of the data for training and the proper processing of this data is also important for a good performance of the NN.

References

[1] K. Albelihi, "The gazelle adaptive racing car pilot," Master's Thesis, Indiana University South Bend, 2014.

[2] "The center for automotive research at stanford," http://me.stanford.edu/groups/design/automotive/.

[3] D. Locaiono and L. Cardamone, "Simulated racing car championship competition software manual," April 2013.

[4] E. Onieva, D. A. Pelta, J. Alonso, V. MilanÃl's, and J. PÃl'rez, "A modular parametric architecture for the TORCS racing engine," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009, pp. 256–262.

[5] C. Guse and D. Vrajitoru, "The epic adaptive car pilot," in *Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference*, South Bend, IN, April 17-18 2010, pp. 30–35.

[6] C. Blum, Ed., *Proceedings of the Genetic and Evolutionary Computation Conference (SIGEVO)*, Amsterdam, The Netherlands, July 6-10 2013.

[7] E. Onieva and D. A. Pelta, "An evolutionary tuned driving system for virtual racing car games: The autopia driver," *International Journal of In telligent Systems*, vol. 27, no. 3, pp. 217–241, Oct 2012.

[8] G. Raidl, Ed., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Montreal, Canada, 2009.

[9] J. Quadflieg and M. Preuss, "Learning the track and planning ahead in a racing car controller," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIGâĂŹ10)*, Copenhagen, Denmark, August 18-21 2010, pp. 395–402.

[10] T. S. Kim and J. C. Na, "Optimization of an autonomous car controller using a self- adaptive evolutionary strategy," *International Journal of Advanced Robotic Systems*, vol. 9, no. 73, pp. 1–15, 2012.

[11] M. Kole and A. Etaner-Uyar, "Heuristics for car setup optimisation in TORCS," in *Proceeding of IEEE Symposium on Computational Intelligence and Games,*, Edinburgh, UK, 2012, pp. 1–8.

[12] P. N. S. Russell, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009.

[13] J. Muñoz and G. Gutierrez, "A human-like torcs controller for the simulated racing car championship," in *Proceedings of the IEEE Symposium on Computational Intelligence Games (CIG)*, Copenhagen, Denmark, August 18-21 2010, pp. 473–480.

[14] A. Blum and R. Rivest, *Advances in Neural Information Processing Systems*. San Mateo, CA: Morgan Kaufmann, 1989, ch. Training a 3-node neural network is NP-complete, pp. 494–501.

[15] *Applying Neural Networks: A Practical Guide*. Academic Press, 1996.

[16] D. Loiacono and L. Cardamone, "Simulated racing car championship competition software manual," April 2013.

[17] E. Velenis and P. Tsiotras, "Minimum time vs maximum exit velocity path optimization during cornering," in *Proceedings of the 2005 IEEE International Symposium on Industrial Electronics*, Dubrovnic, Croatia, June 2005, pp. 355–360.