

Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game

Urszula Markowska-Kaczmar · Mateusz Koldowski

Published online: 3 December 2014

© The Author(s) 2014. This article is published with open access at Springerlink.com

Abstract The paper presents two neural based controllers for the computer car racing game. The controllers represent two generations of neural networks—a multilayer perceptron and a spiking neural network. They are trained by an evolutionary algorithm. Efficiency of both approaches is experimentally tested and statistically analyzed.

Keywords Computer game controller · Spiking neural network · Multilayer perceptron · Evolutionary algorithm

1 Introduction

Neural networks (NN) are widely applied in many areas because of their ability to learn.

Controllers for various devices are examples of their successful application. In the context of this paper the project ALVINN was one of the first fruitful controller based on a neural network approach. It learns to control a vehicle by observing live sensor data as a human drives the vehicle. The classic multilayer perceptron (MLP) neural network trained using backpropagation method was applied. The project, started in early 90-ties. The input to ALVINN were 30×32 images taken by a camera connected to a digitizer. This technique was further developed by other researchers, eg. the paper (Oh et al. 1998) presents application of reinforcement learning. The work (Cao et al. 2007) describes the controller which comprises a MLP and a radial basis function (RBF). The MLP network is used to adjust the parameters of the

controller on-line. The RBF network is used to establish a nonlinear prediction model. In the recent paper (Zheng et al. 2013) the impact of heavy vehicles on lane-changing decisions of following car drivers has been investigated, and the lane-changing model based on two neural network models is proposed.

Computer games can be perceived as a virtual representation of real world problems, and they provide rich test beds for many artificial intelligence methods. An example which can be mentioned here is successful application of multilayer perceptron in racing car computer games, described for instance in Ebner and Tiede (2009) and Togelius and Lucas (2005). Inspiration for our research were the results shown in Yee and Teo (2013) presenting spiking neural network based controller for racing car in computer game. This study has shown that cars controlled by evolved spiking neuron models could perform well and they demonstrated sophisticated driving behaviors.

The aim of our research was to check whether spiking neural network based car controller outperforms those based on the classic MLP in comparable conditions. The paper differs from the approach described in Yee and Teo (2013) in that, apart from NN weights, parameters of spiking neural model are evolved also. Our racing game is built on purpose of the research referring to the neural based controllers. We performed also a statistical comparison of the results of both types of controllers.

The paper consists of six sections. Section 2 presents related works. In Sect. 3 a short description of MLP and spiking networks are presented. In Sect. 4 the basis of evolutionary algorithms used to train both types of neural networks is briefly described. Section 5 presents the details of the developed *Neural Racing* game and methods applied in the system. The experimental results are shown in Sect. 6. Summary concludes the paper.

Communicated by E. Lughofer.

U. Markowska-Kaczmar (✉) · M. Koldowski
Wrocław University of Technology, Wyb. Wyspińskiego 27,
Wrocław, Poland
e-mail: urszula.markowska-kaczmar@pwr.edu.pl

2 Related works

In the literature we can find many papers devoted to the controllers in games which are based on neural networks. The paper (Charles and McGlinchey 2004) presents a general survey of neural networks applied to computer games. A short survey of neural-based agent approaches in computer games is included in Qualls and Russomanno (2009). These agents have the ability to overcome some of the shortcomings associated with implementing classical AI techniques in computer game design. Neural networks can be used in many diverse ways in computer games ranging from agent control, environmental evolution, to content generation. The paper (Togelius and Lucas 2005) describes the evolution of controllers based on neural networks for racing a simulated radio-controlled car around a track, modelled on a real physical track. Cardamone et al. (2009) claim that online evolutionary learning of a fast controller from scratch can effectively improve the performance achieved during the learning process.

Most projects mentioned so far used typical multilayer perceptron networks that are example of second generation of neural networks. Spiking neural networks fall into the third generation of neural network models, which increase the level of realism in a neural simulation. They are more and more popular in developing controllers for robots (Lee and Hallam 1999; Floreano and Mattiussi 2001; Wang et al. 2008; Huemer et al. 2009; Mitic and Miljkovic 2014). The paper (Batllori et al. 2011) describes a sequence of experiments in which a neural network based controller was evolved. The task was light-seeking while avoiding obstacles. The paper (Hagras et al. 2004) proposes adaptive genetic algorithm to train spiking neural network. Bouganis and Shanhahan (2010) present a spiking neural network architecture that autonomously learns to control a 4 degree-of-freedom robotic arm. As it was mentioned, in our research we were inspired by the paper of Yee and Teo (2013), but we were focused on the comparison of both types of controllers in the same environment.

3 Neural networks

Information processing performed in artificial neural network imitates processes in human brain. When we speak about neural network the following elements are essential: its architecture (the way the neurons are connected), the model of a single neuron and a learning rule. The first model of a nerve cell was proposed by Pitts and McCulloch. It was very simple. A step function was implemented as its activation function, so on its outputs only 0 or 1 was produced. It is the first generation of neural networks. The main problem with these networks was a lack of training rule that would

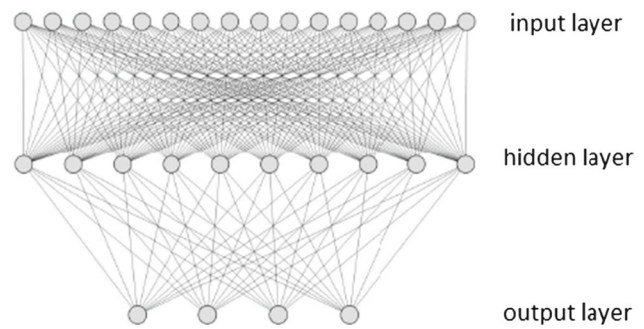


Fig. 1 Feedforward, two layered neural network architecture

allow to use more complex networks. The second generation of neural networks was started with backpropagation learning rule elaboration. It needs a differentiable activation function therefore a continuous activation function is necessary in such a model. In order to simulate processes existing in a human brain in more adequate way spiking neural networks were proposed. They can be also trained using the backpropagation method. The evolutionary approach to train NN is very useful when there is no possibility to acquire desired output for a given input pattern, as it is needed in backpropagation training method.

3.1 MLP neural networks

MLP neural network with the sigmoidal or hiperbolic tangent activation function is the most popular network in various applications. It consists of layers of neurons. There are no connections between neurons in the same layer. Neurons between neighboring layers are fully interconnected. Information is processed from the input layer to the output layer. The architecture of such a network with one hidden layer is shown in Fig. 1.

Neurons in the input layer distribute information only. Sometimes they do a specific function, like normalization or scaling. A typical neuron model in a hidden or an output layer is presented in Fig. 2. It performs two operations.

First, it calculates the net input by summing up all weighted input signals x_i (Eq. 1).

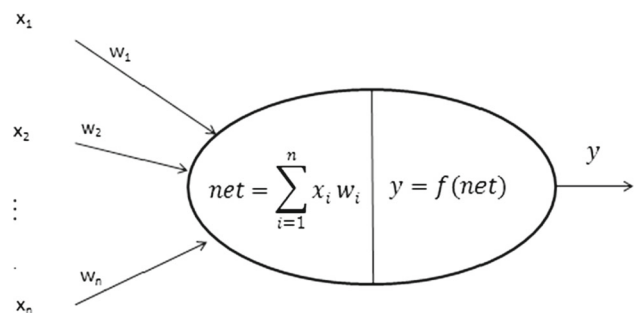


Fig. 2 The model of a neuron used in the first and the second generation of neural network

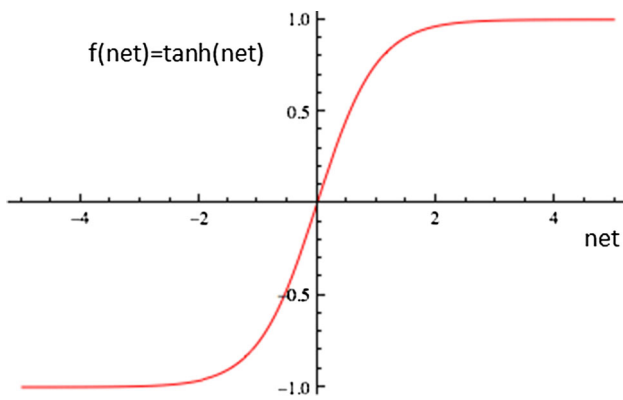


Fig. 3 Hyperbolic tangent as an example of the activation function in the MLP

$$\text{net} = \sum x_i w_i. \tag{1}$$

Next, the activation function f produces the output y as follows:

$$y = f(\text{net}). \tag{2}$$

In our system hyperbolic tangent was implemented as the activation function. The function is presented in Fig. 3.

Training of neural network relies on iteratively adjusting weights. The most popular neural network training—backpropagation method, assumes that for each input the desired output is defined and the error between current output and desired one is the basis for weight change. In the case when the desired output is not known evolutionary approach can be applied.

3.2 Spiking neural networks

Spiking neural networks are characterized by high realism (Gerstner and Kistler 2002; Maass and Bishop 1998). Various models of spiking neuron offer different level of closeness to nervous cell. In our system we applied the Izhikevich model as a compromise between relatively small computation requirements and high biological accuracy. In spiking

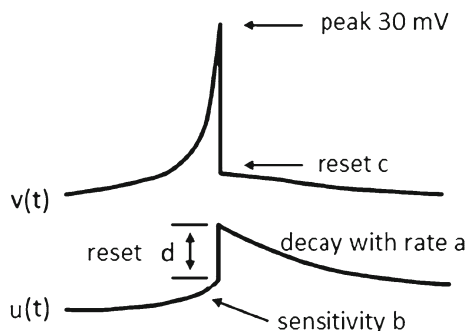


Fig. 4 The parameters of Izhikevich neuron model

networks the current activation level (modeled as some differential equation) is normally considered to be the neuron’s state, with incoming spikes pushing this value higher, and then either firing or decaying over time. The Izhikevich model is described by differential equations (Eqs. 3, 4 and expression 5).

$$\frac{dv}{dt} = 0.004v^2 + 5v + 140 - u + I \tag{3}$$

$$\frac{du}{dt} = a(bv - u) \tag{4}$$

$$\text{if } v \geq 30 \text{ mV then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{5}$$

where a, b, c, d are parameters of the model. They are explained in the Fig. 4 (Izhikevich 2003). I is the neuron current, v is the potential of neural membrane and u is the variable which affects membrane reset. When the potential reaches 30 mV, v and u variables are going to change according to the expression 5. A proper choice of the parameters a, b, c, d is crucial for modeling and for successful application of the model. Various coding methods exist for interpreting the outgoing spike train as a real-value number, either relying on the frequency of spikes, or timing between spikes, to encode information.

Architecture of a spiking neural network can be diverse. In the *Neural Racing* system the feedforward network architecture with one hidden layer is assumed (Fig. 1). Generally, as we mentioned before, the backpropagation method can be also applied to train spiking neural networks, but the desired outputs and special encoding information is needed in this case. Therefore, we used evolutionary algorithm to train the network.

4 Evolutionary algorithm

In this section the evolutionary algorithm applied to train both types of networks will be presented. The general idea behind this technique is as follows (Michalewicz 1992). An initial population of individuals (encoded solutions) is created randomly. Then, each individual is evaluated by computing the fitness function value. Based on the fitness some of the candidates are chosen to seed the next generation by applying mutation and crossover operations. The crossover operation relies on genes exchange between parents with the assumed probability. Mutation acts on one individual and it relies on a random change of a gene value with the assumed probability. Operation of mutation and crossover create individuals of a new population. This process is iteratively executed until a candidate with an assumed fitness function value is found or other stopping criteria are satisfied.

5 Neural Racing game

It is difficult to compare the spiking and the MLP neural network based controllers analytically, because they have embedded randomness in their performance. To achieve the goal of our research (comparison of both types neural controllers), it was necessary to build a system where their efficiency could be experimentally tested. As a test bed environment the computer game *Neural Racing* with racing cars was implemented. For simplicity, we assumed that interactions between competitors will not be included in the system. It means that ability to overtake or to defend current position of a car is not considered. Cars do not clash. The only embedded controller skill is faultless and fast driving on a race track circuit. A player races against other virtual opposing cars.

5.1 Game characteristics

The idea of the game relies on Rallycross competitions. A racetrack characterizes by various paving. Some parts are covered with asphalt, and partially with a layer of gravel. Players compete by driving race cars. The game was written in C++. It was developed for research reasons that is why the tasks connected with visual aspects refer only to:

- racetrack,
- race cars control by players,
- interface elements.

Figure 5 shows an exemplary screenshot of the game interface. In Fig. 6 three developed racetracks are shown. They are presented as 2D bitmaps but they are entrenched in 3D.

5.1.1 Game physics

The physical model of the game was designed and implemented by the authors. Making turns depends on the car speed and the road base (asphalt or gravel). The simula-

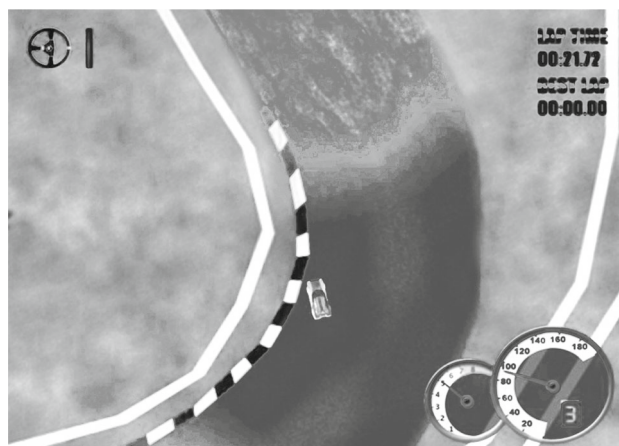


Fig. 5 Screenshot from the developed *Neural Racing* game. Elements of the interface are shown: tachometer and speedometer (right bottom corner), crossing times (top right corner) and visualisation of player action (top left corner)

tor reflects a basic reaction of the extremal driving. Abrupt braking can cause blocking wheels and the car can fall out the road. Overuse of an accelerator during a turn manoeuvre frequently ends with a skid and vehicle diversion. As we mentioned before, the game is no-frills. The interface enables to make a decision referring to a turn action:

- maximally turn right,
- maximally turn left,
- drive straight.

While controlling a driving direction, the player can also:

- accelerate maximally,
- brake maximally,
- not use brake nor accelerator.

The interface does not allow for precise speed regulation and setting turn level. The gearbox is automatic. Pressing

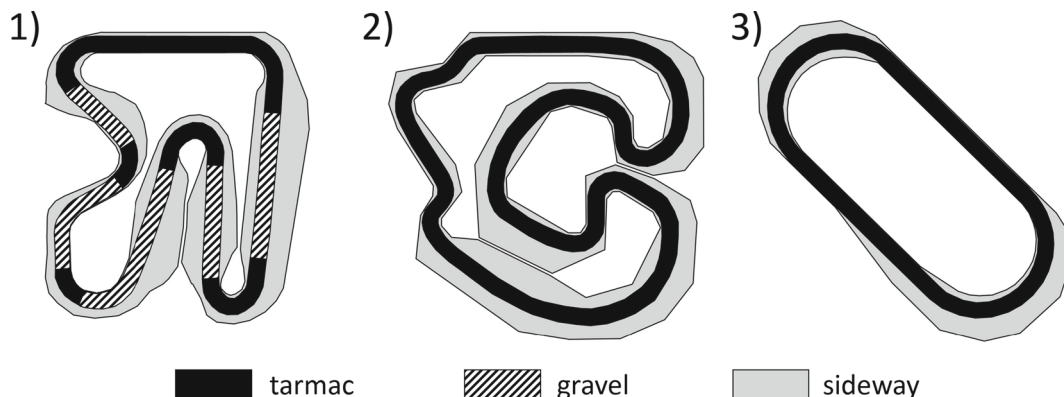


Fig. 6 Three racetracks implemented in the *Neural Racing* game

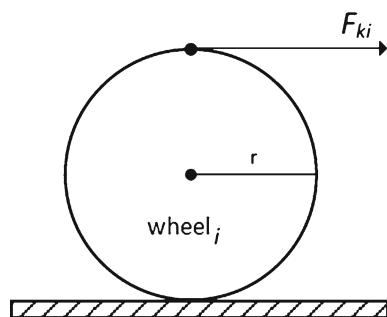


Fig. 7 Force F_{ki} of wheel i . The starting point of the force vector is always on the top of wheel. Its value is equal to the product of wheel radius r and torque of wheel i

gas pedal while stopping causes reverse driving. Detection of collision with bands is implemented by checking whether a point located on the vehicle buffer crosses one of the outer soft shoulder. Each type of track paving characterises by a specific grip parameter and rolling resistance. A sideway has a small road adhesion. It limits the speed to 48 h/km on such surface. In this way it bridges the advantages resulting from shortening the road by using the sideway.

A value of the car speed is calculated on the basis of the physical model and is normalized. To define a car speed simplified assumption was introduced. The motion is reciprocating. The instantaneous car speed dv is then defined by the differential equation:

$$dv = a dt \tag{6}$$

where: a is acceleration; dt is the time interval between calculation steps. Acceleration is assessed on the basis of the following equation:

$$a = \frac{F_k - F_o}{m} \tag{7}$$

where m is car mass; F_o is a force resulting from rolling resistance, while F_k is a sum of forces produced by each wheel during car accelerating or braking, i.e. $F_k = F_{k1} + F_{k2} + F_{k3} + F_{k4}$ (Fig. 7). The car ability to corner depends on its speed and pavement. To increase realism some principles were introduced that define how the car should behave in a given circumstance. The designed simulator enables basic reaction of the racing car during extreme car driving. Too precipitate braking can cause blocking wheels and prolapsing a racetrack. Overusing a gas pedal while turn results in slipping car and its rotation.

5.2 Neural car controllers

In the *Neural Racing* system there are two car controllers: the first one is based on the classic MLP neural network (with hyperbolic tangent activation), the second one is based on the spiking neural network, which has the architecture similar to

the MLP NN (Fig. 1). For both types of controllers the input and output information is the same. The input data deliver information necessary to take a decision how to drive, which is made upon:

- car speed,
- location of track edges,
- the optimal racing line.

A racing human driver makes his driving decisions on the basis of what he sees mostly. In the case of *Neural Racing* game, being inspired by [Togelius and Lucas \(2005, 2006\)](#), cars are equipped with sensors detecting track edges. Visually, they are presented in Fig. 8. As it can be noticed, the action of the sensor depends on the place where the car is (ie. road or roadside).

The sensor measures the distance to the cross point with a detected object. The radius of sensing is equal to 80 m. If the distance to the object is longer than the radius, it is not detected. If the car is on the roadside a measurement is made in reference to the opposite road edge. On this basis the controller receives information how to return on the road.

As show the results from the work [Yee and Teo \(2011\)](#), neural network is not able to learn how to optimally surmount turns for more complicated tracks. It has a tendency to move across the track center. The optimal racing line is created by a trajectory resembling a circle with radius as large as possible (Fig. 9a). It enables a high speed on the way out of the turn. Similarly, it is necessary to shorten the road by driving closely to the turn apex in Fig. 9. The prediction of the optimal point on the tracing line is defined in reference to the current speed of the car. The rule is that the higher the car speed is, the further point on the line is chosen.

Adjustment to the mentioned above rule is a key element in obtaining good results on a racetrack. Therefore the decision about developing racing line was made. It was created on the basis of one lap performance by an advanced human player. During the drive, a car position was recorded every 70 ms. These points assign in advance the desired direction of a drive for the controller. The point on the desired drive direction is chosen in relation to the current speed of the car.

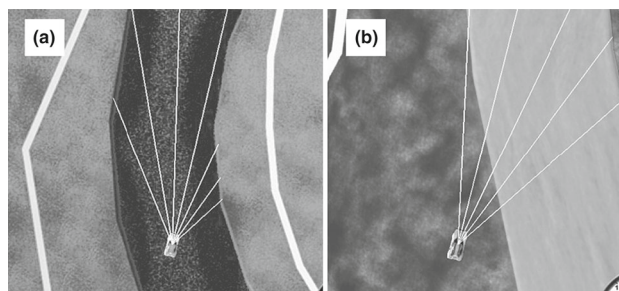


Fig. 8 The sensor action detecting track edges depends on the place where the car is **a** on the track, **b** on a roadside

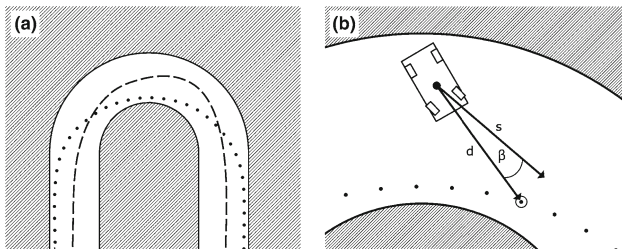


Fig. 9 **a** The optimal trajectory for the corner (*dotted*) and way that usually controllers go when they use only distance and speed sensors (*dashed*). **b** Describes how to get a deviation angle. β is an angle between vectors: d directed to the optimal trajectory and s the current velocity vector. The point on the optimal trajectory is chosen in relation to the current car speed

The angle β is given to the controller (Fig. 9b) as an input. It defines how far is the current velocity vector s from the optimal trajectory d .

Localization on the border is measured by sensors, so the number of inputs referring to this information depends on the sensor number and will be set experimentally. Maximally, 14 sensors is considered. Their localization is shown in Fig. 10.

5.3 The spiking car controller

The controller is implemented on the basis of NeMo open source code. The neural network architecture is two layered feedforward, as it is shown in Fig. 1. Each neuron in the hidden and output layers behaves according to the Izhikevich model. As a NN input the following information is given:

- speed,
- localization of the road border,
- deviation from the racing line.

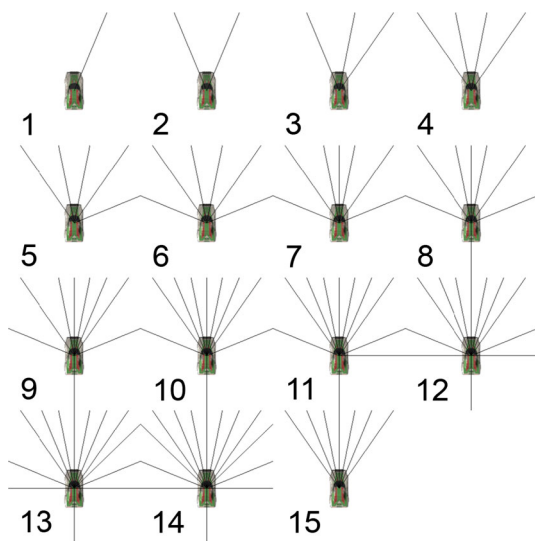


Fig. 10 The tested sensor configurations

Encoding information for a spiking neural network is crucial for its performance. In order to choose the best suited encoding method, in the *Neural Rally* system four encoding methods are implemented.

The first one is rate coding which assumes that the input value (stimulus) is proportional to the firing rate of a neuron (Fig. 11, part 1). We call it further single encoding.

The second one is based on the idea from Florian (2006). It uses two neurons, (Fig. 11, part 2), one of them encodes information by rate coding in the way similar to the single encoding method. Its complementary neuron encodes information in a reverse way (as the encoded value rises, the frequency of the firing decreases).

The third method is called double encoding (Wiklendt et al. 2008). It enables encoding negative values of angles. It needs two neurons. The neurons will be called a and b . If the input value $x \in [-1, 1]$ is positive, then neuron a produces spike as in the single encoding method and neuron b does not produce any stimulation. If the value x is negative there is an opposite situation, neuron b creates a spike and neuron a is inactive, (Fig. 11, part 3).

The last tested method assumes that the range $[A, B]$ of input data values is divided into n equal subsets. In this method n neurons are used to encode one value (Dzienkowski and Markowska-Kaczmar 2010). Depending on which subset the value belongs to, the appropriate neuron generates spikes. Other neurons are inactive. We will refer to this method as one of n encoding.

The controller produces on the outputs information referring to two actions:

- turn direction,
- speed regulation.

After preliminary tests that are not described here, each of these actions is encoding on two output neurons (a and b), so together we have four neurons (double encoding). Intuitively, if neurons a and b have the same output values with a given precision, then the car should move straight. The precision is defined by a threshold value l , which is adjusted experimentally. More precisely, spikes are counted on each neuron in the time window $T = 100$ ms. As a result the spike numbers s_a and s_b are defined. If $\frac{s_a}{s_b} > l$, then the car turns left. If $\frac{s_b}{s_a} > l$ then the car turns right. In other case the car moves straight. Similar solution is applied to speed regulation. The threshold is adjusted experimentally. According to our experimental study, the best value of l is equal to 1.5.

In the system for the research purpose, there is a possibility to choose which values of parameters will be searched by the evolutionary algorithm during training. It is possible to train only weights, weights and neuron delays, weights and neuron parameters and all weights plus parameter values. For all mentioned cases the appropriate chromosomes referring

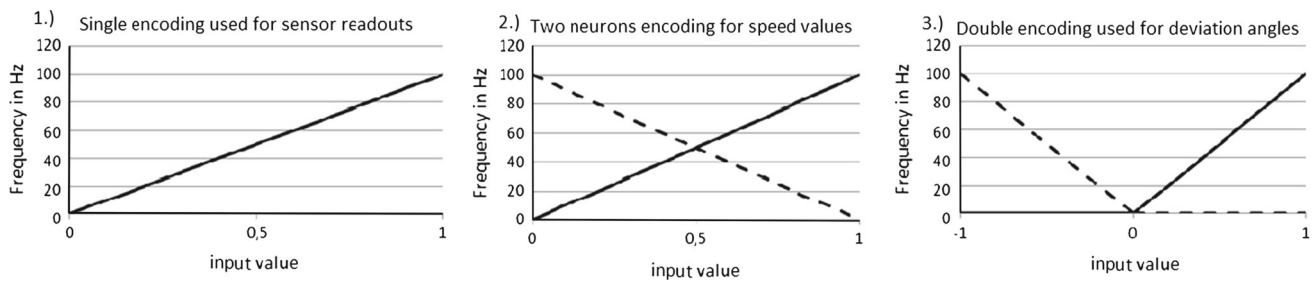


Fig. 11 The encoding methods

to one neuron are presented in Fig. 12. A whole genotype consists of as many chromosomes as many neurons exists in the network. The initial values and ranges for each parameter are given in Table 1. The values for parameters a, b, c, d of the Izhikevich model come from Izhikevich (2003).

In order to evaluate individuals in the current population in each generation the parameters from individuals were decoded and recorded in the neural network, which was then tested as a controller on the racetrack 1. The fitness value is assumed as a sum of points collected by the car during the points. The car driving on the racetrack raises the points increasing its fitness function value. Additionally, there are a couple of rules that decreases the fitness function value. One of them is the collision of a car with the band. In this case the race is broken and there is no possibility for further increase of the fitness value. The penalty points are given, if:

- the car center of gravity goes beyond the track boundaries,
- the car moves with a speed less than 1 m/s.

In consequence, if the sum of penalty points is higher than 50, then the test is also broken. This value was adjusted exper-

imentally. Its role is on the one hand to force controllers to move a car dynamically, committing a small offence during a race, but on the other hand to make certain that the race will be finished.

As soon as all individuals have a fitness value assigned, a new generation of individuals is created. In the system, for the experimental purpose, two selection methods are implemented—the rank method and the roulette wheel method. The system offers two operators—mutation and crossover. A mutation operator can be used as one from two implemented options:

- *Linear mutation* changes a current value g of the gene with the assumed probability p_{mut} to the new value $g \rightarrow g \cdot (1 + Rand \cdot s_{mut})$, where Rand is pseudo random number from the range $[-1, 1]$; s_{mut} is a mutation coefficient.
- *Nonlinear mutation* is a change of the current value g to the new value $g \rightarrow g + (RandP \sim N(0, 1) \cdot s_{mut})$; where: s_{mut} is a mutation coefficient, $RandP \sim N(0, 1)$ is an argument of Gaussian function randomly chosen from $[-r_{mut}, r_{mut}]$.

Crossover is implemented as uniform one. This operation allows to exchange corresponding gene values between two chromosomes of individuals. It is performed with the assumed probability p_{cross} . Offspring individuals inherit characteristics of both parents. Elitism is used. It protects the best individuals against destruction.

The evolution is stopped when the results are satisfying or the generation limit is achieved.

5.4 The MLP based car controller

In order to fair compare both neural network solutions, the MLP architecture was similar to that chosen for the spiking neural network and its parameters were carefully adjusted. A kind of input information delivered as inputs is the same as for spiking neural network. More precisely, it is: data referring to the speed, deviation from the optimal racing line and information from sensors. They are presented as numbers for the MLP network. The NN outputs give information about

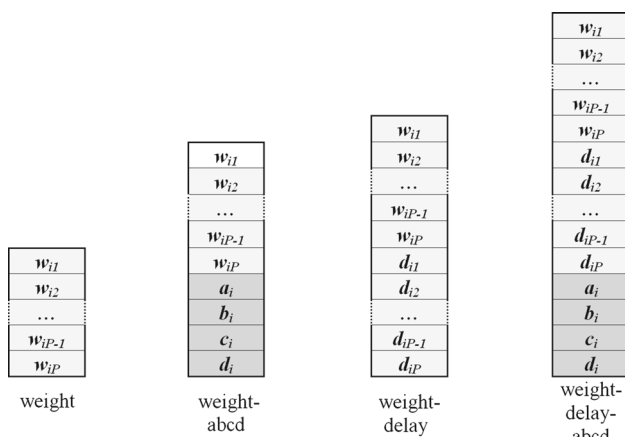


Fig. 12 Chromosome schemes representing various parameter sets searched by evolutionary algorithm for the i th neuron with P inputs; In the scheme w represents weights, $a, b, c,$ and d refer to the parameters in the Izhikevich model of a neuron, d_{iP} means delay on the incoming connection

Table 1 The assumed initial, min and max parameter values for chromosomes

| Parameter | Weights | Delay | a | b | c | d |
|---------------|---------|-------|------|-------|-------|-------|
| Initial value | -15; 15 | 1 | 0.02 | 0.2 | -65 | 2 |
| Min value | -2000 | 1 | 0.01 | 0.121 | -70 | 0.003 |
| Max value | 2000 | 2000 | 0.12 | 0.3 | -45.1 | 9 |

a turn and the proposed speed of a car. They were encoded on two or four output neurons. The number of hidden neurons is set experimentally depending on the number of inputs and output neurons. The network was trained using the same evolutionary algorithm as for spiking NN. The structure of a chromosome is shown in Fig. 12. In this case only weights are encoded.

6 Experimental study

In this section the experiments evaluating the proposed two controllers will be presented. First, we will shortly describe adjusting the training parameters of neural networks, their architecture and the number of sensors. The number of sensors assumed in the experiments, as well as the parameter values of evolutionary algorithm, were chosen on the basis of spiking neural network controller, first. Then, after some experiments evaluating these values for MLP, which are skipped in this paper, these parameter values were applied in further experiments for controllers based on both networks. In the experiments, the controllers were trained on the race-track no 1 presented in Fig. 6. The first racetrack characterizes by relatively high number of turns and varied pavement. The second and third racetracks were used to evaluate the controllers. They differ much from the first one. The second racetrack composes of specific turns that do not exist in the first one. The third one is relatively easy. There exists only two left turns that do not require a reduction of a high speed. The final result of NN training depends on the initial weights that are set randomly. Therefore experiments were performed several times with the same values of the training parameters (starting from various initial weights). From each run of evolution the best individual with the highest fitness value m_{best} was chosen. In order to generally assess the population, the average value s_{av} of the fitness function of the population was calculated.

Bellow are described the most interesting experiments and those making significant development of the *Neural Racing* game.

6.1 Experiment 1. The influence of a sensor configuration on the fitness values

In this experiment only information from the sensors and about the car speed was given as an input to the neural network. The input data were encoded using the single encoding method. The configurations of sensors, presented in Fig. 10, differ in the number of sensors and their angle settings. The experiment was repeated five times only, because of the relatively high number of tested sensors and the time required for one test. To give the reader a rough idea about the time required for one test we can say that in average it lasts approximately 1.4 h for 300 generations of genetic algorithm on PC computer with: CPU-Intel Celeron G550, RAM-4 GB 1,333 MHz and GPU: Intel HD Graphics. Only the nonlinear mutation was used as a genetic operator. A crossover operator was not applied. The details referring to the parameter values of the evolutionary algorithm are shown in Table 2.

Figure 13 presents the influence of the sensor configuration on the achieved results, which were measured by fitness values. It is worth mentioning that with an increase of the number of sensors the number of weights in the network, which have to be adjusted, rises. Therefore configuration with a small number of sensors achieved better results in the initial phase of evolution.

Contrary to the initial feeling, an increase of the number of sensors not necessarily gave improvement of a fitness function value. The results strongly depended on where the sensors were directed. The best was fourth configuration where four sensors were in the front of a car (Fig. 10). In average (Fig. 13, part 1), the configurations: 4, 5, 6, and 15 give similar results, although 5 and 15 (Fig. 13, part 3) have similar and small values of standard deviation. So, they could be the winners in the range of tested configurations. We can explain this fact in the following way. Neural network inputs should be as much as possible informative. When the delivered input is not relevant to the problem solved by a neural network, it is treated as a noise that disturbs the network in a proper performance. In the tested racetracks there are much more right turns than the left ones, so probably

Table 2 The parameters of the evolutionary algorithm in the Experiment 1

| Generation no | Individual no | Selection method | Unchanged no of indiv. | p_{cross} | Type | p_{mut} | r_{mut} | s_{mut} | Chromosome version |
|---------------|---------------|------------------|------------------------|--------------------|---------|------------------|------------------|------------------|--------------------|
| 300 | 100 | Ranking | 50 | 0.0 | Nonlin. | 1.0 | 14 | 30 | Weights |

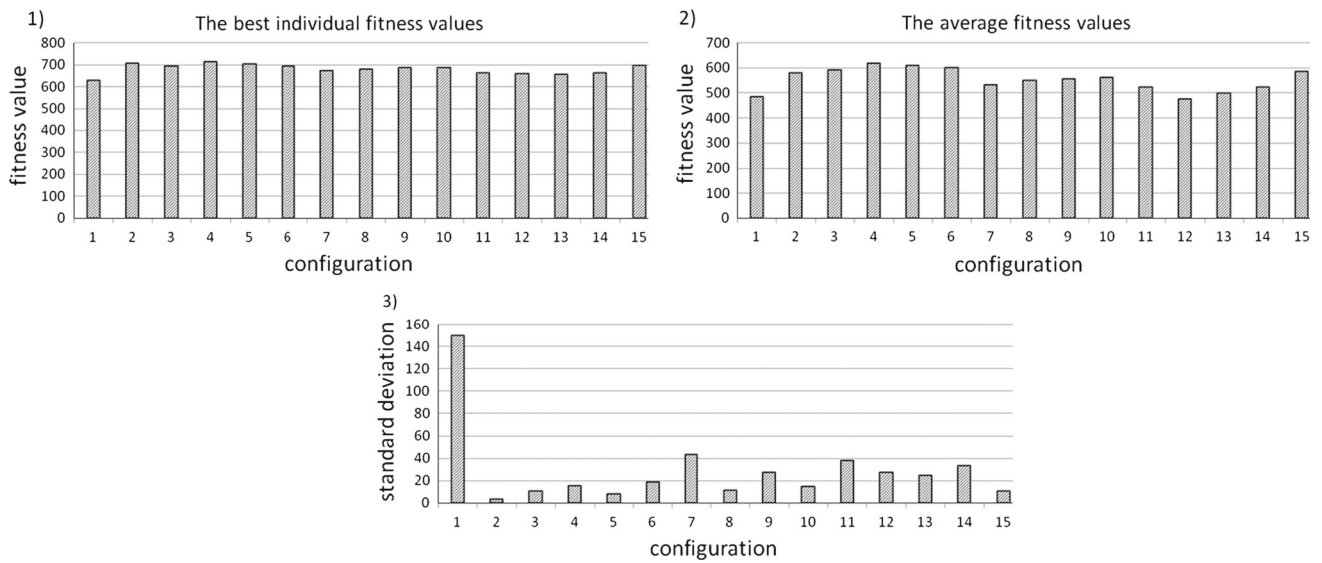


Fig. 13 The influence of sensor configuration on the fitness value

the right-side sensor in configuration 5 helps to overcome right turns. On the other hand the configuration 15 has the straight-ahead sensor that can help to predict road in a better way.

6.2 Experiment 2. The influence of the input information encoding

During this experiment the neural network receives full input information, ie. apart from the speed of the car and its location on the road as in the Experiment 1, the deviation from the racing line was delivered also. The aim of the experiment was searching for the best method of input information encoding for this attribute. The experiment was repeated ten times for each setting. The double encoding and one of n methods were considered for deviation encoding, because they offer easy mapping negative values. The first method needs two neurons and the second one eight input neurons to encode the deviation from the racing line. The sixth configuration of sensors was applied. Other values of parameters are the same as in Experiment 1.

It is worth noticing that introducing deviation from the racing line deserved consideration. Essential improvement of the results was observed. A car moved closer to the optimal racing line what caused the speed growth while turning but on the straight segments the car did not use the maximal speedup. The speed constantly oscillated around 80 km/h. It allows to make a turn without pressing a brake. Figure 14 shows the fitness values and standard deviations for both considered encoding methods for deviation from the racing line. It is difficult to notice a dominant method. The double encoding seems to be more favourable. It needs less input neurons to encode information. Therefore it will be used in the further tests.

6.3 Experiment 3. The influence of evolutionary algorithm parameters

In this experiment we tested various evolutionary algorithm settings. Here we will describe four representative configurations, presented in detail in Table 3. Two types of selection methods were considered. Nonlinear mutation was used

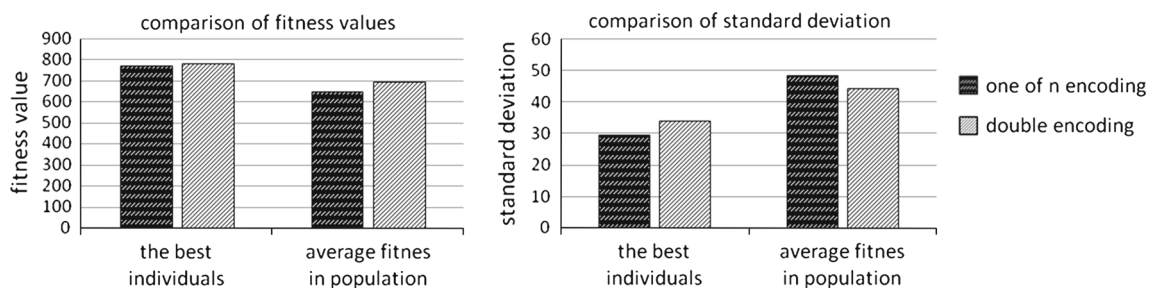


Fig. 14 The influence of the deviation angle encoding method on a fitness value

Table 3 Four representative configurations of evolutionary algorithm tested in Experiment 3

| No. | Selection method | No. of elite indiv. | p_{cross} | Mut. type. | p_{mut} | r_{mut} | s_{mut} |
|-----|------------------|---------------------|-------------|------------|-----------|-----------|-----------|
| 1 | Ranking | 50 | 0.0 | Nonlinear | 1.0 | 14 | 30 |
| 2 | Roulette | 3 | 0.0 | Nonlinear | 1.0 | 14 | 30 |
| 3 | Ranking | 50 | 1.0 | Linear | 0.05 | – | 2 |
| 4 | Roulette | 3 | 1.0 | Linear | 0.05 | – | 2 |

without crossover. The parameter values referring to the genetic operations were chosen on the basis of preliminary experiments. The experiment was repeated ten times. The results of this experiment are presented in Fig. 15. There is noticeable fitness value improvement for the best individuals when using crossover and the linear mutation. There is also a visible change in car driving in this case. The controller has a higher speed on the straight segments. Releasing the gas pedal in turns was sufficient for adequate speed decreasing. Unfortunately, these observations were noticed only in some runs. The results were not stable and high standard deviation can be observed in Fig. 15.

6.4 Experiment 4. Testing ability of the evolutionary algorithm to adjust weights and parameters of neurons

Heretofore the evolutionary algorithm was applied to search neural network weights. It means that the first scheme of the chromosome (Fig. 12) represented an individual. In the current experiment other developed chromosomes, with delay parameter and neuron parameters encoded, were tested. It was precluded by several experiments which aim was adjusting parameters: s_{mut} for various synaptic delays and neuron parameters in the Izhikevich model— a, b, c, d . The specific parameter values in this experiment were as follows:

selection method—roulette, $p_{cross} = 1.0$; mutation linear, $p_{mut} = 0.05$ (Table 4).

As it can be noticed in Fig. 16, adjusting parameters by the evolutionary algorithm has a very positive impact on the stability of achieved results. The standard deviation is smaller than for adjusting weights only. The fitness value of best individuals increases also.

Progress in driving is noticeable while using chromosomes *weights-abcd* and *weights-delays-abcd*. The lesser number of errors while driving characterizes such controllers. Delayed braking and improper driver reaction on skidding appear occasionally. Having this in mind, in the next experiments chromosome encoding *weights, delays* and a, b, c, d parameters was chosen.

6.5 Experiment 5. Comparison of spiking and MLP based controllers

Taking into account the subject of this paper, the most interesting experiment refers to the comparison of controllers based on the spiking and the MLP neural networks. To satisfy the fair comparison we tried to build similar configuration of both neural networks, but MLP does not need spike encoding. Data are directly introduced to the input as a number.

There are two output neurons, one referring to the speed and the second one relating to turning. After some prelimi-

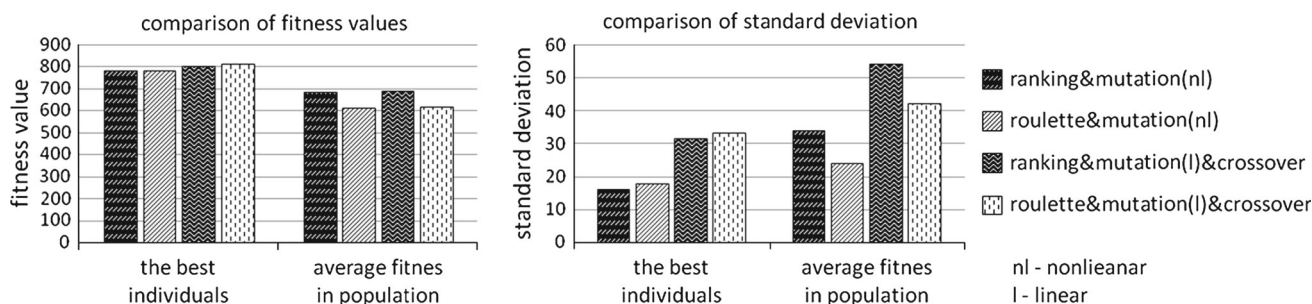


Fig. 15 The comparison of obtained results for various evolutionary algorithm settings

Table 4 The assumed values of parameters for various types of chromosomes in Experiment 4

| Chromosome | 1 | 2 | | 3 | | 4 | | |
|--------------------|---------|---------|--------|---------|-------------|---------|--------|-------------|
| s_{mut} | 2 | 2 | 40 | 2 | 0.25 | 2 | 40 | 0.25 |
| Encoded parameters | Weights | Weights | Delays | Weights | <i>abcd</i> | Weights | Delays | <i>abcd</i> |

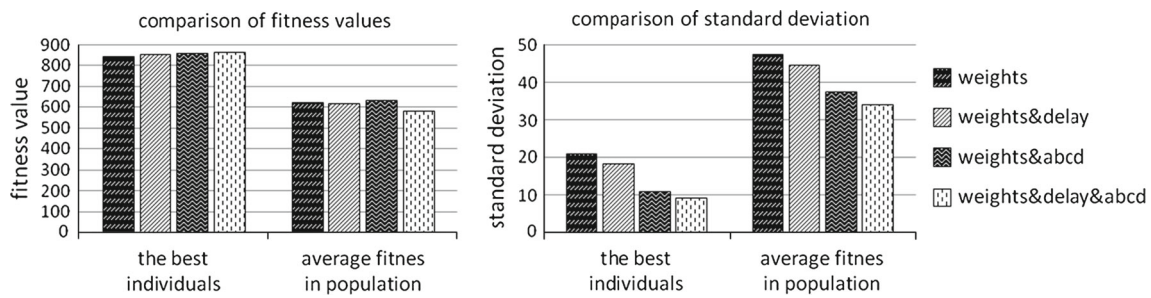


Fig. 16 The comparison of fitness values for various chromosome types

Table 5 Interpretation of outputs in MLP based controller with two output neurons

| Neuron representing turn | | Neuron representing speed | |
|--------------------------|------------|---------------------------|----------------|
| Range | Action | Range | Action |
| [0.2, 1] | Turn right | [0.05, 1] | Acceleration |
| (-0.2, 0.2) | Straight | (-0.05, 0.05) | Without action |
| [-1, -0.2] | Turn left | [-1, -0.05] | Braking |

Table 6 The assumed values of training parameters for both controller types

| Controller type | Generation no. | No. of individ. | Selection method | No. of elite individ. | Crossover parameter | Mutation parameters | | | | Chromosome data |
|-----------------|----------------|-----------------|------------------|-----------------------|---------------------|---------------------|-----------|-----------|-----------|-----------------|
| | | | | | | Type | p_{mut} | r_{mut} | s_{mut} | |
| Spiking | 500 | 100 | Roulette | 3 | 1.0 | Linear | 0.05 | - | 2 | Weights |
| | | | | | | | | | 40 | Delays |
| | | | | | | | | | 0.25 | <i>abcd</i> |
| MLP | 500 | 100 | Roulette | 3 | 1.0 | Linear | 0.05 | - | 2 | Weights |

nary tests with four output neurons in the MLP network, due to the lack of difference in the results, in the final experiments two output neurons were used. The outputs are decoded as shown in Table 5. The sensor configuration number 15 (Fig. 10) was used.

In the MLP network there was nine neurons: seven neurons receives information from sensors, one neuron contains information about deviation angle and one about speed value. There were 16 neurons in the hidden layer and 2 in the output layer. The architecture of the spiking neural network consisted of 11 input neurons: 7 neurons receives information from sensors, 2 neurons includes information about a deviation angle and 2 about a speed value. The network consists of 28 hidden neurons and 4 output neurons. The sensor configuration number 15 was applied. Input information encoding was as follows: information from sensors—the single encoding method, information about a speed—two neurons encoding method, an angle of deviation—double encoding method.

The evolutionary algorithm was used to train both controllers. The values of training parameters are described in Table 6.

Most of parameter values were the same in both cases. For the MLP NN a chromosome encoding weights was used

only. The number of generation in the evolutionary algorithm was set to 500. Learning was performed on racetrack number 1 and was repeated ten times, i.e. ten times evolution was performed to find neural network parameters. From each evolution (with elitism) the best ten individuals were then applied as controllers in the tests on the racetracks 1, 2, 3. Each race consisted of 25 laps, which should be run as quick as possible. Vehicles do not interfere with each other.

Two groups consisting of ten cars were equipped with the evolved neural controllers from one learning. The first group used spiking neural networks, the second one MLP NN. The cars started from the same point to have the same chance to win.

Table 7 presents the results of MLP and spiking NN controllers for all three racetracks available in the system. It is important to underline that one row in this table corresponds to controllers obtained in one learning process, but they are placed in the descending order in terms of average fitness function values. In other words, controllers described in 10th rows are the worst from all tested controllers.

In the Table 7 *record lap time* shows the best time attained on one lap. The number of cars finishing race (column—*no of cars fin.*), refers to the number of controllers that had

Table 7 The comparison of MLP and spiking NN results

| Race number | MLP | | | | | Spiking NN | | | | |
|-------------|-----------------|------------------|------------|--------------|--------|-----------------|------------------|------------|--------------|--------|
| | Record lap time | No. of cars fin. | Points no. | Av. position | Winner | Record lap time | No. of cars fin. | Points no. | Av. position | Winner |
| Racetrack 1 | | | | | | | | | | |
| 1 | 37.66797 | 10 | 155 | 5.5 | 1 | 38.0332 | 3 | 55 | 15.5 | 0 |
| 2 | 37.84863 | 10 | 155 | 5.5 | 1 | 38.17871 | 5 | 55 | 15.5 | 0 |
| 3 | 38.13379 | 3 | 86 | 12.4 | 1 | 38.30762 | 8 | 124 | 8.6 | 0 |
| 4 | 37.36914 | 6 | 118 | 9.2 | 1 | 38.5 | 0 | 92 | 11.8 | 0 |
| 5 | 37.96875 | 7 | 131 | 7.9 | 1 | 38.3418 | 4 | 79 | 13.1 | 0 |
| 6 | 37.8125 | 5 | 117 | 9.3 | 1 | 38.11914 | 2 | 93 | 11.7 | 0 |
| 7 | 37.55664 | 9 | 147 | 6.3 | 1 | 38.51563 | 1 | 63 | 14.7 | 0 |
| 8 | 37.31641 | 10 | 155 | 5.5 | 1 | 38.69336 | 0 | 55 | 15.5 | 0 |
| 9 | 38.26758 | 8 | 140 | 7 | 1 | 39.5 | 0 | 70 | 14 | 0 |
| 10 | 39.69922 | 5 | 125 | 8.5 | 1 | 41.66797 | 0 | 85 | 12.5 | 0 |
| Av. | 37.964063 | 7.3 | 132.9 | 7.71 | 10 | 38.785743 | 2.3 | 77.1 | 13.29 | 0 |
| Racetrack 2 | | | | | | | | | | |
| 1 | 42.26172 | 10 | 65 | 14.5 | 0 | 40.74023 | 9 | 145 | 6.5 | 1 |
| 2 | 42.44043 | 10 | 127 | 8.3 | 0 | 41.3457 | 5 | 83 | 12.7 | 1 |
| 3 | 42.56738 | 6 | 55 | 15.5 | 0 | 41.15137 | 10 | 155 | 5.5 | 1 |
| 4 | 42.58008 | 1 | 55 | 15.5 | 0 | 41.40625 | 10 | 155 | 5.5 | 1 |
| 5 | 42.45313 | 9 | 80 | 13 | 0 | 41.13086 | 9 | 130 | 8 | 1 |
| 6 | 42.02148 | 6 | 126 | 8.4 | 0 | 41.25781 | 4 | 84 | 12.6 | 1 |
| 7 | 42.59961 | 0 | 55 | 15.5 | 0 | 41.38867 | 10 | 155 | 5.5 | 1 |
| 8 | 42.82227 | 1 | 55 | 15.5 | 0 | 41.50195 | 10 | 155 | 5.5 | 1 |
| 9 | 42.79102 | 3 | 55 | 15.5 | 0 | 41.73438 | 10 | 155 | 5.5 | 1 |
| 10 | 44.55078 | 3 | 117 | 9.3 | 1 | 46.37891 | 0 | 93 | 11.7 | 0 |
| Av. | 42.70879 | 4.9 | 79 | 13.1 | 1 | 41.803613 | 7.7 | 131 | 7.9 | 9 |
| Racetrack 3 | | | | | | | | | | |
| 1 | 21.73633 | 10 | 55 | 15.5 | 0 | 21.19531 | 10 | 155 | 5.5 | 1 |
| 2 | 21.55273 | 10 | 98 | 11.2 | 0 | 20.94727 | 10 | 112 | 9.8 | 1 |
| 3 | 21.75879 | 10 | 135 | 7.5 | 1 | 21.87207 | 10 | 75 | 13.5 | 0 |
| 4 | 21.20313 | 8 | 118 | 9.2 | 1 | 21.32617 | 6 | 92 | 11.8 | 0 |
| 5 | 22.01172 | 5 | 78 | 13.2 | 0 | 21.31055 | 8 | 132 | 7.8 | 1 |
| 6 | 22.11328 | 1 | 78 | 13.2 | 0 | 20.98242 | 6 | 132 | 7.8 | 1 |
| 7 | 21.65234 | 9 | 141 | 6.9 | 1 | 22.16406 | 7 | 69 | 14.1 | 0 |
| 8 | 22.33203 | 10 | 102 | 10.8 | 0 | 22.20898 | 10 | 108 | 10.2 | 1 |
| 9 | 22.56641 | 10 | 94 | 11.6 | 1 | 22.61133 | 10 | 116 | 9.4 | 0 |
| 10 | 22.17188 | 5 | 152 | 5.8 | 1 | 23.90625 | 0 | 58 | 15.2 | 0 |
| Av. | 21.909864 | 7.8 | 105.1 | 10.49 | 5 | 21.852441 | 7.7 | 104.9 | 10.51 | 5 |

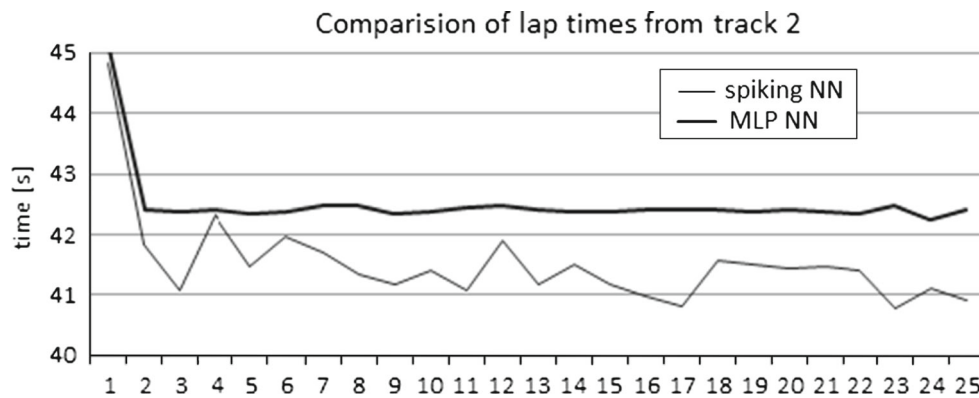


Fig. 17 The comparison of lap times for the MLP and spiking NN based controllers on the second racetrack

driving tempo comparable with the leader. More precisely, the individual (controller) that achieved the number of laps in the race less than leader about 2 laps, lost possibility to drive further. It means that the number of cars finishing the race is the number of cars that have the number of driven races no less than 2 in relation to the leader. The sum of points collected by the group of controllers (*points no*) is calculated assuming that the winner obtains 20 points and 1 point is for the last position in the competition.

On the first racetrack which served to train controllers, the MLP based controllers always won. Times achieved on the second and third racetracks of both controllers group are statistically analyzed. Because we have only ten runs, therefore the assumption referring to checking normality of distribution can not be satisfy (Derraca et al. 2011; Garcfa et al. 2009; Picek et al. 2012), that is why we used nonparametric Wilcoxon's signed rank test to compare the results. We create two hypothesis H_0 there is no difference between the results of both methods, and H_1 the difference is not zero. To perform the test we calculate differences d_i between the best times of two compared controllers on the i th run in all ten runs. The differences have to be normalized and then ranked. Next, two sums were created— R^+ and R^- . R^+ is a sum of ranks for runs in which the spiking NN controller outperforms MLP based controller, R^- is the sum of ranks for the opposite situation. If the null hypothesis is true, the sum of the positive ranks and the sum of the negative ranks are expected to be roughly equal. The statistics T is chosen as the smaller one from these two sums. For the second racetrack, therefore, $T = 10$. For $N = 10$, and significance level = 0.05 (one-tailed), the critical value of $T_{\text{critical}} = 10$. The rule is that if T is equal to or less than T_{critical} , we can reject the null hypothesis. Therefore, in this case, we reject the null hypothesis. That is, we can conclude that spiking NN based controllers are better than MLP based controllers. For the results achieved on the third racetrack, $R^+ = 34$ and $R^- = 21$. Therefore, in this case, we can not reject the null hypothesis.

On the basis of statistical analysis, it is difficult to decide which neural network is better to build controllers for this game. One can say that the MLP NN is much better when in the game we consider racetracks, which were used to train the MLP NN. The spiking neural network is less predictable. It is shown in Fig. 17. It presents two exemplary individuals—spiking and MLP based controllers on the second track during 25 laps.

The MLP NN has almost identical results on each lap. The spiking NN achieved variable times that on the one side is its drawback but on the other side it can be perceived as its advantage in this particular application, because it is less foreseen for a human player.

Observing the way of car driving, we noticed that the spiking NN based controllers frequently makes very sophisticated maneuvers. Its ability to generalize knowledge is higher but it coped poorly on one of turning of the first racetrack where reflex was necessary to control the car.

7 Summary

The experiments performed in our research do not allow for the unambiguous indication of the winner. The results of spiking controllers on the first (training) racetrack in term of achieved times and points were much worse in comparison with the classic MLP NN, but considering the second racetrack, which characterizes by the high difficulty level, they moved in more natural driving style and statistically achieved better results. It is worth underlying that the second track consists of turns that were not present in the training track. It contains a turn with a decreasing radius, a turn with an increasing radius, fast turns that can be done with a full gas and neighbouring turns located close to each other. The spiking NN was able to drive in other conditions than those existing during training. Frequently, on this racetrack the MLP based controller moved with the same race track. The third racetrack was relatively easy. There exist two turns

only. To move on it a small speed reduction was needed. The results of both types of controllers were comparable. On this basis we can formulate the conclusion that the MLP based controller should be preferable if we consider in the game racetracks given in advance, where the network can be trained. If we assume that apart from training racetrack exist unknown racetracks, then the spiking NN based controller will be better. It is able to generalize skills acquired during training and manifest very natural way of driving.

In the future development of the system we plan to experiment with a various fitness function that could better reflect the way of the car driving. Also our idea is to apply car collisions in the game, which would make driving more difficult and the game more interesting. Such approach creates a big challenge for controllers and can be the acid test for their intelligent behaviour. Avoiding collision will be an important element of their survival.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

- Batllori R, Laramee C, Land W, Schaffer J (2011) Evolving spiking neural networks for robot control. *Procedia Comput Sci* 6:329–334
- Bouganis A, Shanahan M (2010) Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity. In: *Proceedings of the 2010 international joint conference on neural networks (IJCNN)*, pp 1–8
- Cao J, Cao B, Chen W, Xu P, Wu X (2007) Neural network control of electric vehicle based on position-sensorless brushless dc motor. In: *Proceedings of IEEE international conference on robotics and biomimetics (ROBIO'07)*, pp 1900–1905
- Cardamone L, Loiacono D, Lanzi P (2009) On-line neuroevolution applied to the open racing car simulator. In: *IEEE congress on evolutionary computation (CEC'09)*, pp 2622–2629
- Charles D, McGlinchey S (2004) On the analysis of experimental results in evolutionary computation. In: *Proceedings of 5th conference on computer games: artificial intelligence, design and education*, pp 163–169
- Derraca J, Garcfa S, Molina DFH (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput* 1:318
- Dzienkowski BJ, Markowska-Kaczmar U (2010) Biologically inspired agent system based. In: *Agent and multi-agent systems: technologies and applications*. LNCS, vol 6071, pp 110–119
- Ebner M, Tiede T (2009) Evolving driving controllers using genetic programming. In: *IEEE symposium on computational intelligence and games (CIG'09)*, pp 279–286
- Floreano D, Mattiussi C (2001) Evolution of spiking neural controllers for autonomous vision-based robots. In: LNCS, vol 2217
- Florian RV (2006) Spiking neural controllers for pushing objects around. LNCS 4095:570–581
- Garcfa S, Molina D, Lozano MFH (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms behaviour: a case study on the cec2005 special session on real parameter optimization. *J Heuristics* 15:617644
- Gerstner W, Kistler W (2002) *Spiking neuron models*. Cambridge University Press, Cambridge
- Hagras H, Pounds-Cornish A, Colley M, Callaghan V, Clarke G (2004) Evolving spiking neural network controllers for autonomous robots. In: *IEEE international conference on robotics and automation*, pp 4620–4626
- Huemer A, Elizondo D, Gongora M (2009) Constructive neural networks. In: *A constructive neural network for evolving a machine controller in real-time*, vol SCI 258. Springer, Berlin, pp 225–242
- Izhikevich EM (2003) Simple model of spiking neurons. *IEEE Trans Neural Netw* 14(6):1569–1572
- Lee W-P, Hallam J (1999) Evolving reliable and robust controllers for real robots by genetic programming. *Soft Comput* 3(2):63–752
- Maass W, Bishop C (1998) *Pulsed neural network*. MIT Press, USA
- Michalewicz Z (1992) *Genetic algorithms + data structures = evolution programs*. Springer, New York
- Mitic M, Miljkovic Z (2014) Neural network learning from demonstration and epipolar geometry for visual control of a nonholonomic mobile robot. *Soft Comput* 18(5):1011–1025
- Oh SY, Lee JH, Choi DH (1998) Vision-based lateral control of vehicles for road following using reinforcement learning and neural networks, towards the new horizon together. In: *Proceedings of the 5th worlds congress on intelligent transport systems*. Paper no. 4002
- Picek S, Golub M, Jakobovic D (2012) On the analysis of experimental results in evolutionary computation. In: *Proceedings of the 35th international convention MIPRO*, pp. 1064–1069
- Qualls J, Russomanno D (2009) *Evolutionary computation. Ch. Applications of neural-based agents in computer game design*. <http://www.intechopen.com/books/evolutionary-computation/applications-of-neural-based-agents-in-computer-game-design>
- Togelius J, Lucas SM (2005) Evolving controllers for simulated car racing. In: *The 2005 IEEE congress on evolutionary computation*, pp 1906–1913
- Togelius J, Lucas SM (2006) Evolving robust and specialized car racing skills. In: *IEEE congress on evolutionary computation*, pp 1187–1194
- Wang X, Hou ZG, Zou A, Tan M, Cheng L (2008) A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing* 71(4–6):655–666
- Wiklendt L, Chalup S, Middleton R (2008) A small spiking neural network with lqr control applied to the acrobot. *Neural Comput Appl* 18:369–375
- Yee E, Teo J (2011) Evolutionary spiking neural networks as racing car controllers. In: *11th international conference on hybrid intelligent systems (HIS)*, pp 411–416
- Yee E, Teo J (2013) Evolutionary spiking neural networks as racing car controllers. *Int J Comp Inf Syst Ind Manage Appl* 5:365–372
- Zheng J, Suzuki K, Fujita M (2013) Evaluating the impact of heavy vehicles on lane-changing decisions of car drivers: a neural-network-based methodology, vol 9