Training a Multilayer Perceptron to predict a car speed in a simulator: Comparing RPROP, PSO, BFGS, and a memetic PSO-BFGS hybrid

Artur Henrique Gonçalves Coutinho Alves* Vinícius Veloso de Melo†

Federal University of São Paulo (UNIFESP), Department of Computer Science, Brazil

ABSTRACT

One of the biggest challenges in developing an autonomous racing pilot is planning ahead during a race. The pilot may have knowledge of the track beforehand, but understanding this information and choosing the best action for each situation in real-time is a very complex problem. To deal with such problem, machine learning methods such as artificial neural networks (ANNs) have been applied to learn from simulation data and generalize during a race making correct decisions to drive the vehicle. ANNs are usually trained with the classical backpropagation technique, but here we investigate other optimization methods. Our proposed hybrid of Particle Swarm Optimization (PSO) with the local optimizer called BFGS, resulting in what is usually called a Memetic algorithm, is compared in two versions to a well-known gradient descent based training technique (Resilient Propagation, RPROP), the regular PSO, and the BFGS. The objective here is not yet to drive the car, but to predict its future speed on the track, which can be useful for mapping and planning trajectories. Experiments were performed on data generated by a popular car racing simulator, and the results show that the hybrid is a promising approach for the investigated problem. The hybrid optimization method outperformed both PSO and BFGS and is able to compete with RPROP, yielding high-quality results.

Keywords: Car driving, neural network, global numerical optimization, evolutionary algorithm, meta-heuristic

1 INTRODUCTION

Complex problems, such as pattern recognition and non-linear regression, can be solved by various techniques; one such technique is neurocomputing. Specifically, since the introduction of the Perceptron model [41] and, later on, the adoption of the Multilayer Perceptron model (MLP) [2] thanks to the discovery of the backpropagation training method [42], Artificial Neural Networks (ANNs) have been in the spotlight both as a solution to various challenges and as a challenge themselves, given the complexity of parameter tuning of the training procedure (including the network structure), which is proportional to the complexity of the network. It is easy to find research papers proposing and testing diverse techniques for solving these problems, including hybrid methods. While classical training methods can be efficient local optimizers and reasonable explorers, hybrid methods are supposed to be better at finding a global optimum region and then applying the local optimizer for fine tuning the neurons' weights.

Usually, training methods comparisons are made against the classical backpropagation (BP) method which is employed for historical reasons but known to be inefficient [45, 13]. In this work, we make comparisons with the Resilient Propagation [40] (RPROP), a direct adaptive and fast learning method shown to outperform BP [13]. In fact, our work also evaluates other three methods: a successful general numerical local optimizer called BFGS [4, 14, 15, 44] (Broyden-Fletcher-Goldfarb-Shanno), a widely-employed metaheuristic called Particle Swarm Optimization [12] (PSO), and a PSO and BFGS hybrid implemented for this work. We created the hybrid as a proof-of-concept from the canonical PSO, even though there are many different variations [48, 19, 3, 36]; therefore, many future works can be done.

Contextualizing, the problem to be solved by the ANN in this paper is the approximation of a small part of the physics model used in a racing simulation game (SCRC, [25]) known for its use in academic competitions[1] already held at WCCI [26], CEC, CIG, and GECCO. This simulator has been used in works such as [39, 35, 6, 34, 10] and may lead to new discoveries in robotics, specially in mapping and navigation. The discovered model can be used to help the creation of automatic pilots capable of better adapting to the conditions of each track by predicting the car speed at the X and Y axes. The prediction is useful to plan the car's trajectory and calculate its absolute position as only the relative position is available in-game. The model creation should be performed without having to drive several laps collecting data and testing potential solutions. For that reason, a high-quality model must be obtained through fast training with a small sample size (in terms of laps). Thus, in this contribution we evaluate how the methods perform on such task. The main contributions of this paper are summarized below:

- For the best of our knowledge, this is the first attempt of using such specific hybrid combination for training ANNs;
- Correctly predicting a car speed in a simulator can help the pilot take better decisions, reducing risks and mistakes; as far as we know, there is no research on such topic for the simulator investigated in this work;
- A comparison of five training methods, including a memetic one that presents competitive results with RPROP.

The remaining of the paper is organized as follows. In the next section, we present a preliminary research on techniques and problems related to this context; in Section 4 we detail the problem, the methodology, present and discuss the experiment. Finally, the work is concluded with critical notes and future proposals.

2 BACKGROUND AND RELATED WORK

This section presents basic concepts needed to understand the problem and the proposed experiments.

2.1 Neural Networks

The term "neural networks" refers to a broad class of computational intelligence approaches used to solve complex problems. In general, all present artificial neuron models interconnected by

*E-mail: artur.alves@unifesp.br
†E-mail: vinicius.melo@unifesp.br

[1] Simulated Car Racing Championship: http://cs.adelaide.edu.au/~optlog/SCR2015/index.html

weighted synapses that react to input patterns and produce some output. One such approach, arguably the most common one, is the feedforward multilayer Perceptron (MLP) network, based on the Perceptron model [41]. Such networks are usually presented in a layered model, with distinct input units, hidden neurons and output neurons [20]. Throughout the years since the creation of error backpropagation (BP) training [42], many researchers have proposed and tested increasingly complex problems, discussing the capabilities of the general technique [2].

One such class of problems is regression, where the network must represent a model of an unknown function. Since a multilayer neural network is able to solve non-linearly separable problems, it should, in theory, be able to adequately model very complex functions. Harder problems unfortunately require not only more data but also better training approaches, since some simpler ones might take an unfeasible time to converge, converge into local minima, or even not converge at all. Therefore, researchers have also proposed new training techniques to improve or replace the traditional BP.

2.1.1 Training

The most traditional MLP training algorithm, BP [42, 9], is a supervised learning approach, i.e., the network uses desired output (labeled) data to check if it is predicting them correctly and, if not, calculate how much it must change its weights. In each iteration, it passes the training data forward, activating the neurons and collecting the generated output; then it compares these values to the labeled data and calculates an output error. This error is then passed backwards through the network, when the algorithm calculates an error for each unit and changes their weights accordingly. This is done in a way that, with time, reduces the output errors, i.e., the network outputs values become sufficiently close to the expected results.

BP is a gradient-descent algorithm, which means it may yield low-quality results when applied to problems with error surfaces that present discontinuities or many local minima. Many improvements have been proposed, such as the use of momentum and adaptive learning rates for escaping local minima [32], but one learning scheme stands out: the resilient propagation (RPROP) [40]. In this technique, each synapse has an individual update-value, used to determine by how much that weight will be increased or decreased. Basically, if the partial derivative of the weight changes signs (meaning the last update was too big and the current solution has "jumped" over a minimum), the update-value is decreased; inversely, if the partial derivative keeps its current sign after an update (suggesting it is in the correct direction), the update-value is increased. This value is then used to increase or decrease the actual weight, according to the error derivative sign. Finally, if a weight update changes the error derivative sign, it means once again that solution has skipped a minimum; to correct this, the algorithm reverts the last weight update, in a process called weight backtracking.

As well as improving upon the gradient descent approach, new learning techniques have been proposed and tested throughout the years, such as Levenberg-Marquardt [18] and evolutionary algorithms [37, 22]. Swarm Intelligence, although a relatively new research area, presents some opportunities as well.

2.2 Particle Swarm Optimization

Bio-inspired computing is a research topic of Computational Intelligence that covers approaches inspired by biology capable of solving a variety of complex problems. Besides neural networks, which are inspired by the topology and functioning of the brain, one can find in the spotlight subareas such as evolutionary computation, based on Darwin's Theory of Evolution such as Genetic Algorithms, and swarm intelligence (SI) based on the social behavior of individuals.

One SI approach is Particle Swarm Optimization [12], an algorithm based on flock behavior, i.e., how birds fly, communicate and self-organize when searching for food. Each particle (an individual, based on a bird) in the population has a position vector (representing a potential solution to the optimization problem) and a velocity vector in the search space. The optimization occurs by changing the position of a particle; it "flies" through the search space according to its individual velocity vector, effectively exploring the problem and eventually converging into a minimum - hopefully, global. At each epoch, traditionally, the only operators applied are the velocity update function and the position update function. While the new position is simply an element-wise sum of the current position and the recently calculated velocity vectors, the velocity update is the important step in this approach.

Each particle also stores its personal best, i.e., the position where the objective function value using the particle's position as input is the best found so far. At each epoch, this information is updated as needed. The population also stores a historical best, the global best. These two vectors are key components in the velocity update function, defined as Equation 1, where w represents the inertia weight (how much the current velocity affects the new velocity), $rand()$ are random uniform numbers between 0 and 1, c_1 is the cognitive coefficient, c_2 is the social coefficient, *lBest* is the personal best and *gBest* is the global best. c_1 and c_2 are defined by the user but commonly set as 2.04.

$$v(t+1) = w * v(t) + rand() * c_1 * lBest + rand() * c_2 * gBest \quad (1)$$

The cognitive and social coefficients are responsible for making the algorithm focus more on exploration (by increasing the cognitive coefficient) or on exploitation (by increasing the social coefficient). Also, the inertia weight can be dynamically adjusted so the algorithm initially performs a global search (with a high inertia weight) and, with time, lowers such weight to perform a local search.

Even though PSO is able to perform some kind of local search, it is still considered a global search method; it slows down considerably when approaching a local minimum but may not be able to actually reach it. Therefore, a hybrid approach, mixing PSO with a different, gradient-based local-search technique might yield interesting results.

As PSO is a general optimization algorithm, it can be used to train and/or evolve a neural network including its structure, as shown in [31, 21, 30, 8, 43, 49, 16, 48]. Most work compared tuned PSO algorithms versus the canonical BP showing that PSO can achieve better prediction accuracy but usually taking a longer training time. Few used RPROP in the comparison [7], and in these cases RPROP was the best method, not significantly improving PSO in the hybrid versions. Therefore, similar results could be expected here.

A pseudocode of PSO for training an MLP neural network is shown in Algorithm 1. This pseudocode has a small improvement: after a pre-specified number of epochs without reaching a better validation error, the entire population is purged and randomly reinitialized (a restart procedure), while keeping the global best information. New populations, with random positions and velocities, can search for other promising regions that can be better than the gBest's region.

2.3 Hybridization

Hybridization is, basically, the use of two or more distinct methods that present orthogonal characteristics in tandem in order to solve some kind of problem. A successful hybrid approach should offer better results than its separate parts, in exchange for its higher complexity. Some works propose a mixture of techniques aiming to overcome their downsides, like a PSO-EA hybrid [5] and a PSO-SQP hybrid [46].

Algorithm 1 Particle Swarm Optimization applied to neural network training.

1: **Input:** data set, number of units in each layer, stopping criteria (minimum error threshold and maximum number of iterations), population size, number of iterations to wait until population purging
2: **Output:** trained network (the network weights array is the Global Best position, the position with lowest historical validation error)
3:
4: - Initialize PSO parameters: search space boundaries, velocity boundaries, learning rates and inertial weight boundaries
5: - Create base network
6: - Generate initial population with random positions and velocities
7:
8: **while** stopping criteria are not met **do**
9: Update inertia weight
▷ Population update procedure
10: **for each** $particle \in [1, 2, \ldots, populationSize]$ **do**
11: Update velocity of *particle* according to Equation 1
12: Apply box constraints to velocity
13: $position_{particle} = position_{particle} + velocity_{particle}$
14: Apply box constraints to position
15: $fitness_{particle}$ = network training error with $position_{particle}$ as weights
16: **if** $fitness_{particle} < localBestFitness_{particle}$ **then**
17: Local Best position and fitness are replaced by current position and fitness
18: **end if**
19: **if** $fitness_{particle} < globalBestFitness$ **then**
20: Global Best position and fitness are replaced by current position and fitness
21: **end if**
22: **end for**
▷ Purging check
23: **if** the best validation error has not improved in the last *purgeIterations* without purging **then**
24: Generate new random population while keeping the global best
25: **end if**
26: **end while**

In the current context of adjusting the weights of an ANN, a hybrid may be used to reach better results and/or converge faster than a standalone method. Several works employed hybridization to improve the training process, achieving high-quality results [29, 1, 47, 50, 27]. Memetic Algorithms [33] are one kind of hybrid approaches, where a population-based method is combined with a local search method. Petalas et al. [38] conclude that a memetic approach, inspired by Dawkins' definition of memes [11], is viable when one such implementation is compared to a regular global optimizer.

PSO is a very popular metaheuristic and memetic versions have been used before for solving several combinatorial and numerical problems [23, 24, 46]. However, we are not aware of the memetic version proposed here for solving the same task investigated in this contribution.

3 The Proposed Method

The hybrid implementation proposed in this work is based on that presented in Algorithm 1. Here we propose two approaches:

- **MPSO**: a PSO followed by a local search method, configuring a simple version of a memetic PSO; after PSO finishes, the search is taken over by another method - in this case, BFGS. This behavior can be seen in Algorithm 2. Basically, the objective of the hybridization is to let PSO do multiple searches for promising regions of the search space while BFGS, which is a more efficient method, handles the local optimization process. Since it does so by searching specifically at the best promising region found by PSO, it is only executed after the global search is over.
- **MPSOR**: a PSO alternating with a local search method, to compare with the previously presented simple version; after a specific number of iterations, the algorithm pauses the PSO method and uses BFGS for a small number of iterations on the current *gBest*. PSO, then, returns to its normal behavior using the new *gBest*. This process is repeated as long as there are iterations available or the result is not satisfactory. This process can be reproduced by placing line 6 from Algorithm 2 inside the *if* block starting in line 23 in Algorithm 1.

BFGS is a quasi-Newton method, i.e., it is a mathematical optimizer that only needs the first derivative to minimize problems; such derivative can be estimated by finite-difference of sampled data. Because of this, however, its performance on non-convex surfaces may be lackluster. Therefore, in theory, the hybrid approach can apply both methods where they excel.

Algorithm 2 Hybrid approach.

1: **Input:** data set, number of units in each layer, stopping criteria (minimum error threshold, maximum number of PSO iterations and maximum number of BFGS iterations), population size, number of iterations to wait until population purging
2: **Output:** trained network
3:
4: - Initialization and PSO execution occur as in Algorithm 1
▷ Switch from PSO to BFGS
5: **if** the best validation error is higher than threshold **then**
6: Start BFGS optimization with the best solution found by PSO. Stop after *max_BFGS_iter* iterations or when the error is lower than the pre-specified threshold
7: **end if**
8: Network weights = position with lowest historical validation error

4 Experiments

This section presents the experiments, results, and discussion.

4.1 Simulation data

The data set was extracted from the Simulated Car Racing Championship (SCRC) game through a framework[2] developed in Python for the creation of new autonomous pilots. The code using the framework parses the raw data from the simulator (*sim*) into an easy-to-use Python object containing all sensorial information available. The sim offers dozens of sensors and information about the vehicle. However, as the goal of this work is to approximate the vehicle physics only, any information regarding external data such as track and obstacle positioning is discarded.

Two tracks were used here: a simple tarmac track, *cgspeedway1*, seen in Figure 1, and a complex dirt track, *Dirt 3*, seen in Figure 2, both available in SCRC. The autonomous pilot chosen for this work shows a good performance in the first, easy track; the second one is harder, having the pilot struggling to deal with the terrain and the various jump sections. Among the large selection of tracks available, the first one was chosen for yielding easier data, while the second is an intermediate dirt track that, with the struggling pilot, offers data that should be harder to approximate.

The goal of this work is to have a model that correctly predicts the X and Y speeds (both axes) for time $t+1$, only; extended predictions are yet to be investigated. As the data is polled from the simulator at 50Hz, the difference between speeds in subsequent samples is small. Thus, it has been decided to subsample the data, using one instance every 10 instances, effectively reducing the polling rate to 5Hz.

Five variables available from the simulator were selected as input for the model: current speed along the X and Y axes of the

[2] SnakeOil: http://xed.ch/project/snakeoil/index.html

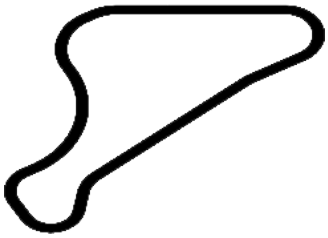

Figure 1: *cgspeedway1* track.

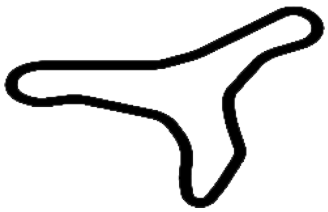

Figure 2: *Dirt 3* track.

vehicle, throttle value, brake value, and steering amount. Two input variables were created to reduce the correlation between current and future speeds, storing the difference between the speed in times t and $t-1$ for both axes.

For the preliminary results presented in this work, the stock SnakeOil pilot developed for that Python framework [3] was set to run three laps in the two selected tracks. A restriction set by the competition model is that the pilot may only access the current track in a race in order to collect data; as such, it must use the same track for training and testing the model, since a new model is created for each race.

4.2 Development

The tests were executed on a stock quad-core Core i5-2400 at 3.1GHz with 8GB DDR3 RAM running Python version 3.5.0 and *R* version 3.2.2 on Windows 10 64-bit. Except for the pilot and the simulator, the code was implemented in *R* programming language, using the *neuralnet* package [17] as the basis for creating the structure of the neural network, training it with RPROP, and testing the other training algorithms. The package already has an easy-to-use training function which supports both BP and RPROP; for these experiments, RPROP with weight backtracking was used. PSO, PSOr, MPSO and MPSOR were coded from scratch by the researchers, except for the BFGS function - its implementation is in the *R* function *optim*.

While the RPROP approach is handled entirely by the *neuralnet* package and the output is simply the trained network, the other optimization methods tested here must iterate directly over the weights matrix; this matrix is extracted from a network generated by the package and replicated with random weights for each particle. Therefore, each PSO particle is an array of weights, which must be properly reshaped to be evaluated by the neural network.

As for the data, the race log is loaded, randomly split into 10-fold, and then separated into training and validation (each training fold), and test sets. The columns of each subset are scaled independently by their standard deviation. Training, validation, and test results are evaluated by sum of squared errors (SSE) and mean squared errors (MSE), where the final value is the sum of the X and Y errors.

[3] SCRC'15: http://cs.adelaide.edu.au/~optlog/SCR2015/awards.html

4.3 Preliminary investigation

The algorithms are compared directly both on computational cost (time to finish training) and on result quality (difference between the predicted and the expected outputs). A challenge presented by this proposal is how to directly compare algorithms, so a *grid-search* was performed to tune RPROP.

The feedforward network has two output neurons (for X and Y speeds), logistic activation functions for the hidden nodes, and linear outputs. It was initially configured with various numbers of hidden neurons and trained by RPROP with different error derivative thresholds. For each result, we calculated the validation error - this offers both a way to determine a stopping criterion for the other approaches and a way to compare all algorithms. Each combination was run 10 independent times on data obtained from the *cgspeedway1* track, and the mean results are shown in Tables 1 and 2 (errors, in this case, are calculated by SSE). This helps comparing their effectiveness, but their efficiency is harder to measure, since each technique needs a different number of objective function calculations (in this case, applying a set of weights to the network and running all data sets). Also, for the current application, the real elapsed training time is more important than the actual number of objective function calculations; thus, real time in seconds is used as the cost measure.

The best results were obtained with two hidden neurons with a threshold of 0.005, but the average training time is too high when compared to the second best average result (threshold of 0.01). Therefore, all subsequent tests were executed with two hidden neurons and a threshold of 0.01 (interestingly, the default threshold in *neuralnet* is 0.01), meaning a sacrifice of precision over performance. For each following test case, RPROP was run in these conditions and its minimum validation error was set as stopping criterion for PSO, BFGS, and the hybrid. Again, using such criterion one wants to evaluate processing time.

4.4 Configuration of the multiple comparison

Two tracks were used in this experiment: *cgspeedway1* and *Dirt 3*. While the pilot's performance was good in the first track, in the second one it crashed the car a few times; therefore, it is expected that the data may be harder to process because velocity suddenly changed from many km/h to almost zero.

For each data set (comprising sensor and actuator information at 200ms - 5Hz - steps for three laps), RPROP was executed 30 independent times with different seeds to train the network. The best validation error was, then, used as a stopping criterion for BFGS, PSO, PSOr (PSO with restart), and the two memetic versions - MPSO and MPSOR, each also run 30 independent times. As convergence is not guaranteed, these techniques also had iteration limits: BFGS was limited to 1000, pure PSO to 5000, MPSO to 500 in the first phase and 1000 in the second phase and MPSOR to 500 PSO iterations and 100 BFGS iterations every 50 consecutive PSO iterations without improvement. Each run uses a new random split of the data set.

PSO has other parameters, set as: $c_1 = c_2 = 2.04$, $maxW = 0.9$, $minW = 0.05$ and $w_{t+1} = w_t - ((maxW - minW)/maxIter)$, where w is the inertia weight, linearly updated at each iteration. PSO, PSOr, and the two memetic variations use 20 particles. PSOr and the memetic versions purge their populations after 50 iterations without improvement, while PSO has no purging procedure. A grid-search could also be done to identify better configurations as the current one was chosen arbitrarily after running the methods a few times. All techniques were used to train a network with one hidden layer composed of two neurons, using a logistic activation function for hidden units, and linear outputs for the output layer.

Table 1: RPROP SSE validation errors to determine the stopping criterion and best network topology.

Error threshold \ Hidden Neurons	1	2	3	4	5	10	20	50
1	79.3982	2.3649	2.1524	2.2295	2.8064	3.1403	4.8589	9.4516
0.5	78.8612	1.5109	1.3140	1.3202	1.3662	1.6922	2.6316	6.4228
0.1	78.4171	0.6606	0.6144	0.5587	0.5838	0.7542	1.2853	3.4431
0.05	78.3627	0.5389	0.5041	0.4913	0.4868	0.6777	1.0473	3.1155
0.01	78.2965	**0.3994**	0.4367	0.4704	0.5070	0.7843	1.1128	2.2247
0.005	78.2882	**0.3773**	0.4812	0.4959	0.4931	0.8753	1.0733	3.2844
0.002	89.5168	0.4294	0.6196	0.6029	0.5334	0.7891	1.7260	4.2075

Table 2: RPROP training times (in seconds)

Error threshold \ Hidden Neurons	1	2	3	4	5	10	20	50
1	0.101	0.178	0.127	0.116	0.085	0.092	0.194	0.374
0.5	0.198	0.348	0.302	0.211	0.183	0.195	0.232	0.683
0.1	1.111	2.022	2.004	1.132	0.825	0.641	0.991	2.961
0.05	2.358	4.214	4.106	2.331	2.15	1.828	2.386	4.9170
0.01	16.029	**38.285**	45.449	60.87	44.565	15.801	16.455	28.564
0.005	32.255	**90.71**	176.197	148.277	80.637	43.004	52.315	85.508
0.002	80.589	250.944	420.631	485.008	349.947	153.393	224.402	261.509

4.5 Results

A summary of the MSE results can be found in Tables 3, 4, 5, and 6. Pairwise comparisons using Wilcoxon's rank sum test (single-sided) on the test set errors are shown in Tables 7 and 8, while comparisons of the training times are in Tables 9 and 10; all p-values< 0.05 mean that the row method presented lower times than the column method - otherwise, no conclusion can be made. Figures 3 and 4 have plots comparing the observed values with the predicted values; perfect fits are perfect diagonal lines.

In both tests, MPSO achieved the lowest average error, followed by RPROP; regarding median error, the two memetic approaches won by a small margin (the MPSOR has its average worsened by runs stuck in local optima in *Dirt 3*). However, the hypothesis tests found no statistical evidence of difference between them (see Tables 7 and 8).

The best (lowest) times were achieved by RPROP, with the two memetic versions coming in second. The hypothesis tests confirm this, as no technique is statistically faster than RPROP and the two memetic algorithms are statistically faster than the remaining techniques.

Table 3: Test errors in *cgspeedway1*. PSO with restart (PSOr), Memetic PSO (MPSO), and Memetic PSO with Return (MPSOR).

	RPROP	BFGS	PSO	PSOr	MPSO	MPSOR
Average	0.003996	0.263675	0.110572	0.023097	**0.003845**	0.003951
Std. Dev.	0.000764	0.319624	0.054253	0.021826	0.000823	**0.000662**
Median	0.004071	0.255030	0.094894	0.012584	**0.003704**	0.003856

Table 4: Training times (in seconds) in *cgspeedway1*.

	RPROP	BFGS	PSO	PSOr	MPSO	MPSOR
Average	**32.69**	110.15	310.46	306.75	92.31	90.41
Std. Dev.	7.98	19.69	**3.53**	4.95	16.97	25.66
Median	**33.57**	119.93	311.20	308.1	90.60	94.03

Table 5: Test errors in *Dirt 3*.

	RPROP	BFGS	PSO	PSOr	MPSO	MPSOR
Average	0.004057	0.307323	0.123548	0.027718	**0.003827**	0.100128
Std. Dev.	0.003257	0.329420	0.032478	0.029732	**0.002423**	0.2112
Median	0.002779	0.288398	0.120994	0.017652	0.003061	**0.002431**

Table 6: Training times (in seconds) in *Dirt 3*.

	RPROP	BFGS	PSO	PSOr	MPSO	MPSOR
Average	**119.68**	180.62	458.54	467.56	152.48	130.55
Std. Dev.	60.72	16.70	**3.07**	4.35	25.42	47.78
Median	**103.08**	186.43	459.12	468.70	150.54	142.72

Table 7: Table of p-values for a pairwise comparison using Wilcoxon's rank sum test for *cgspeedway1* test errors. All p-values< 0.05 mean that the row method presented lower errors than the column method.

	RPROP	BFGS	PSO	PSOr	MPSO
BFGS	1	-	-	-	-
PSO	1	5.57e-01	-	-	-
PSOr	1	1.33e-01	2.98e-13	-	-
MPSO	8.78e-01	1.53e-07	1.27e-16	1.27e-16	-
MPSOR	1	8.90e-06	1.77e-10	1.77e-10	1

Table 8: Table of p-values for a pairwise comparison using Wilcoxon's rank sum test for *Dirt 3*.

	RPROP	BFGS	PSO	PSOr	MPSO
BFGS	1	-	-	-	-
PSO	1	4.01e-02	-	-	-
PSOr	1	1.17e-03	3.15e-12	-	-
MPSO	1	6.55e-09	1.27e-16	5.33e-15	-
MPSOR	1	1.97e-04	9.02e-05	1.89e-04	1

BFGS, the second slowest algorithm, was also the less accurate and presented by far the most dispersed plot. It may be related

Table 9: Table of *p*-values for a pairwise comparison using Wilcoxon's rank sum test for *cgspeedway1* training times. All *p*-values< 0.05 mean that the row method presented lower times than the column method.

	RPROP	BFGS	PSO	PSOr	MPSO
BFGS	1	-	-	-	-
PSO	1	1	-	-	-
PSOr	1	1	1.33e-02	-	-
MPSO	1	2.32e-04	1.27e-16	2.06e-10	-
MPSOR	1	1.87e-03	2.06e-10	2.06e-10	1

Table 10: Table of *p*-values for a pairwise comparison using Wilcoxon's rank sum test for *Dirt 3* training times. All *p*-values< 0.05 mean that the row method presented lower times than the column method.

	RPROP	BFGS	PSO	PSOr	MPSO
BFGS	1	-	-	-	-
PSO	1	1	-	-	-
PSOr	1	1	1	-	-
MPSO	1	2.29e-05	1.27e-16	1.96e-10	-
MPSOR	1	3.32e-06	1.27e-16	1.96e-10	6.22e-01

to the default configuration used in the experiment; thus, a grid-search could also be helpful. This may also be caused by its higher difficulty of correctly converging in non-convex surfaces [28].

As for PSO and PSOr, their results were worse than RPROP's and the memetic techniques', but consistent across the board; on the other hand, the purging/restart process appears to largely improve' performance; this can be observed not only in the direct comparison of Tables 3 and 5, but confirmed in the hypothesis tests. Such behavior may be due to the technique's difficulty in actually converging after finding a promising region; even more likely, PSOr is exploring more than PSO. Another interesting fact is that the training times did not present significant changes throughout many tests, because they never actually reached their main stopping criterion and were ended instead by the maximum number of iterations.

Considering these issues, MPSO appears to be able to adequately find a high-quality solution and improve it with the local search method, that is, from a promising region found by PSO it can better use BFGS to actually converge to the local optimum, expecting to quickly reach the global optimum.

One may see in Figures 3 and 4 that the plots show very small dispersion along the reference line, meaning that predictions were accurate for X-speed but less accurate for Y-speed. We are still investigating these results to understand such difference and try to improve the prediction. A possible reason for such difference between axes is that, while the X-speed represents the actual speed of the vehicle, the Y-speed represents its lateral movement; therefore, it changes more drastically and changes signal very often, leading to more difficult predictions.

Some prediction issues can be observed in the plots, but we still have no final explanation for them. For instance, BFGS finds clusters of predictions close to values -1 and 1 for X-speed, and 0.5 for Y speed. It also appears for MPSOR for Dirt 3. We suppose that this is because of a large amount of observations (expected values) with such values.

Another interesting characteristic is the curvilinear plots shown for PSO and PSOr, where it seems that for some runs these methods could not find weights able to reach the expected values, resulting in a limited maximum output.

The last characteristic is what looks like vertical lines. For them, we suppose that the input weights are close to 0.0 and only the bias weights are substantially different. Such behavior was mostly observed for BFGS and MPSOR.

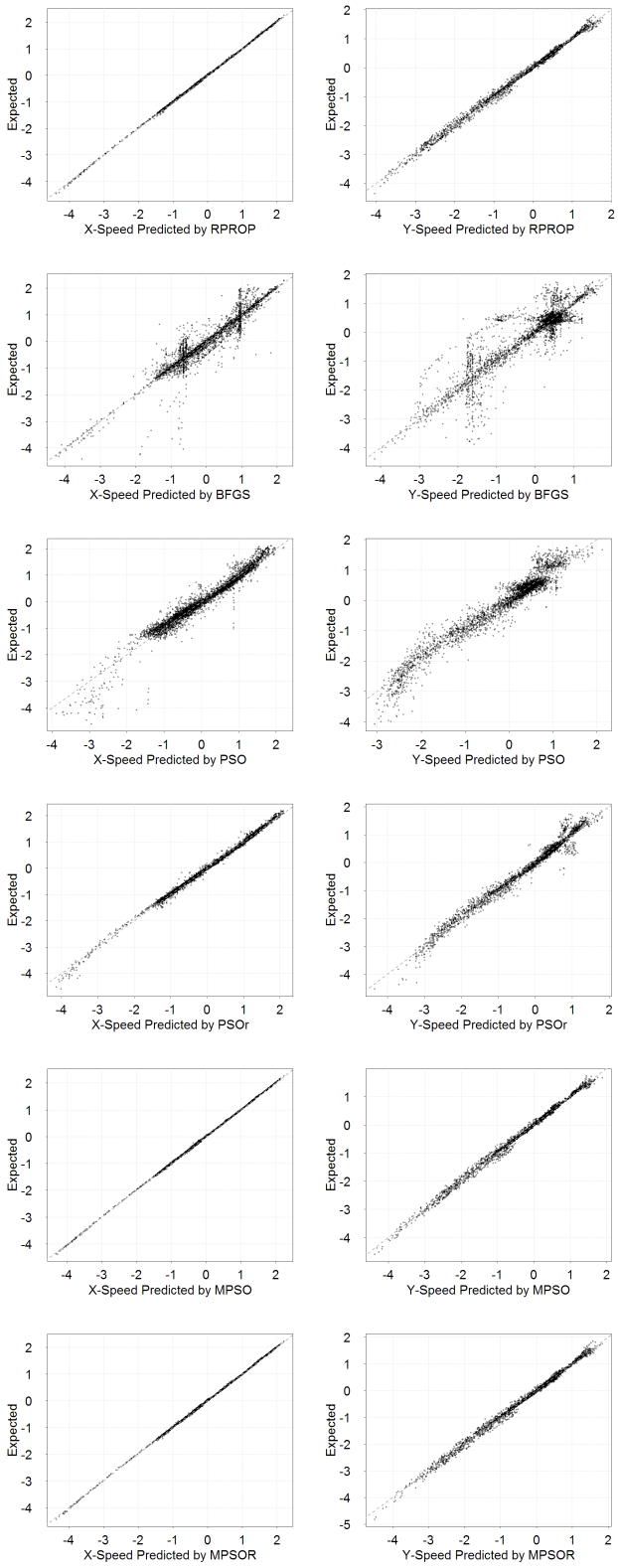

Figure 3: Plots of expected values (vertical axis) and predicted values (horizontal axis) for both speed variables in *cgspeedway1*. The closer the points are to the line $expected = predicted$, the better the prediction is.

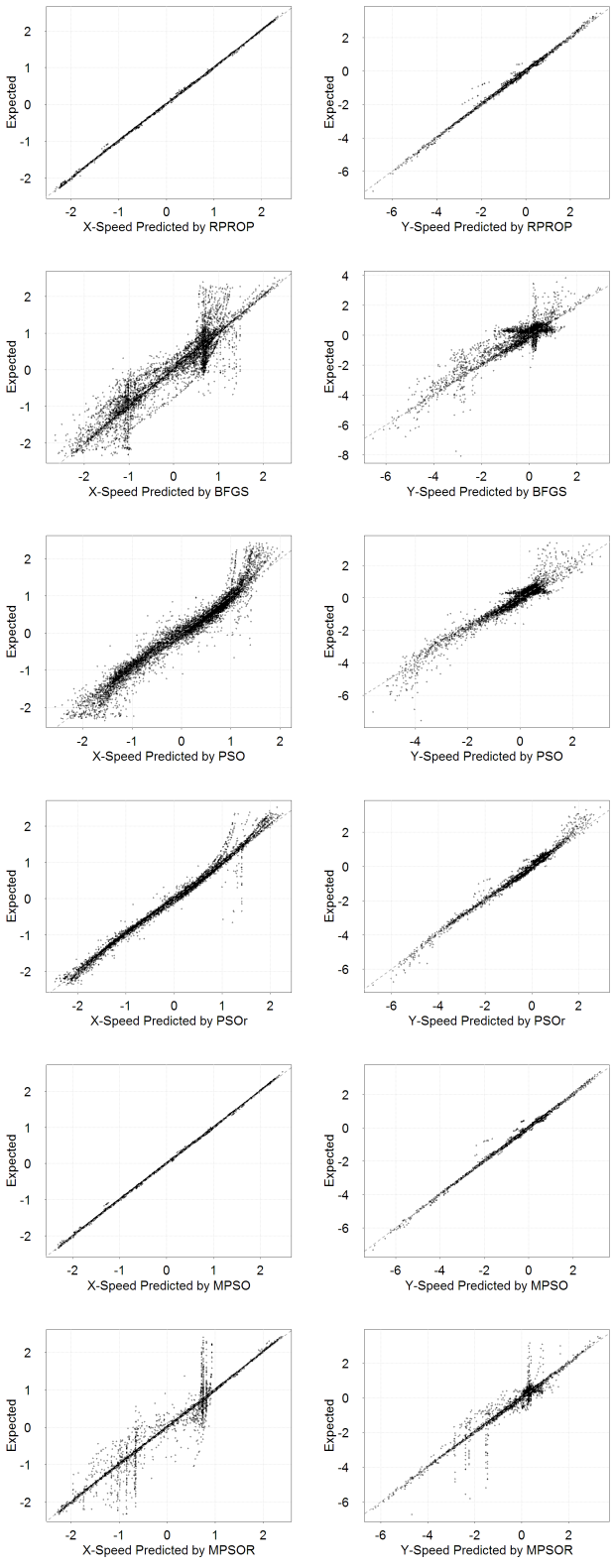

Figure 4: Plots of expected values (vertical axis) and predicted values (horizontal axis) for both speed variables in *Dirt 3*. The closer the points are to the line $expected = predicted$, the better the prediction is.

5 Conclusions

This paper investigated five methods for training a Multilayer Perceptron artificial neural network for predicting the speed of a car in a race using a racing simulator. This paper had two initial goals: to demonstrate the applicability of artificial neural networks for physics modeling based on recovered simulated data, and the capability of memetic/hybrid approaches in training such networks. As far as we know, this is the first time such application was investigated for such racing simulator.

As expected from the extensive works available in literature, a traditional feedforward artificial neural network was able to approximate a complex function representing the physics behavior of a virtual vehicle. It was able to predict, with an acceptable precision, the axes speeds according to different actuator values, such as acceleration, braking, and steering, among other information.

For training the network, the proposed memetic PSO-BFGS hybrid algorithms managed to exploit the advantages of each technique and were capable of competing with the well-established RPROP algorithm, achieving statistically comparable prediction errors even though being usually slower. The hybrid versions were much more efficient than its parts, and the purge/restart mechanism improved the performance of PSO; the regular PSO got stuck in local optima and presented much higher errors.

Future work will focus on improving both global and local approaches by using more efficient versions of PSO and other local search methods. Also, it is important to code in a faster programming language, such as C, and comparing with other methods. More racing tracks will be used, adding diverse contexts with varying complexity and data set sizes for testing.

Finally, we intend to use such models in the development of an autonomous pilot for the racing simulator.

Acknowledgements

This work has been supported by "Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil" (CNPq) grant #486950/2013-1 and "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil" (CAPES) scholarship.

References

[1] E. Alba and J. F. Chicano. Training neural networks with ga hybrid algorithms. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 852–863. Springer, 2004.

[2] E. B. Baum. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4(3):193–215, 1988.

[3] T. Blackwell, J. Branke, et al. Multi-swarm optimization in dynamic environments. In *EvoWorkshops*, volume 3005, pages 489–500. Springer, 2004.

[4] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[5] X. Cai, N. Zhang, G. K. Venayagamoorthy, D. C. Wunsch, et al. Time series prediction with recurrent neural networks using a hybrid pso-ea algorithm. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 1647–1652. IEEE, 2004.

[6] C. Caldeira, C. Aranha, and G. Ramos. Torcs training interface: An auxiliary api for developing torcs drivers. *XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, 13, 2013.

[7] M. Carvalho and T. B. Ludermir. Hybrid training of feed-forward neural networks with particle swarm optimization. In *Neural information processing*, pages 1061–1070. Springer, 2006.

[8] K. Chau. Particle swarm optimization training algorithm for anns in stage prediction of shing mun river. *Journal of hydrology*, 329(3):363–367, 2006.

[9] Y. Chauvin and D. E. Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.

[10] V. K. Daros and F. S. da Silva. Identificando e classificando trechos de pistas no simulador de corridas torcs. In *Computer Games and Digital Entertainment (SBGAMES), 2014 Brazilian Symposium on*, pages 934–937. SBC, 2014.

[11] R. Dawkins. *The selfish gene*. Number 199. Oxford university press, 2006.

[12] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[13] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. 1988.

[14] R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[15] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[16] V. G. Gudise and G. K. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 110–117. IEEE, 2003.

[17] F. Günther and S. Fritsch. neuralnet: Training of neural networks. *The R Journal*, 2(1):30–38, 2010.

[18] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *Neural Networks, IEEE Transactions on*, 5(6):989–993, 1994.

[19] M. Hajian, A. M. Ranjbar, T. Amraee, and B. Mozafari. Optimal placement of pmus to maintain network observability using a modified bpso algorithm. *International Journal of Electrical Power & Energy Systems*, 33(1):28–34, 2011.

[20] S. Haykin and N. Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[21] K.-Y. Huang, L.-C. Shen, K.-J. Chen, and M.-C. Huang. Multilayer perceptron learning with particle swarm optimization for well log data inversion. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–6. IEEE, 2012.

[22] J. Ilonen, J.-K. Kamarainen, and J. Lampinen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1):93–105, 2003.

[23] S. Li, M. Tan, I. W. Tsang, and J. T.-Y. Kwok. A hybrid pso-bfgs strategy for global optimization of multimodal functions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(4):1003–1014, 2011.

[24] B. Liu, L. Wang, and Y.-H. Jin. An effective pso-based memetic algorithm for flow shop scheduling. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(1):18–27, 2007.

[25] D. Loiacono, L. Cardamone, and P. L. Lanzi. Simulated car racing championship: Competition software manual. *arXiv preprint arXiv:1304.1672*, 2013.

[26] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez. The wcci 2008 simulated car racing competition. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 119–126. IEEE, 2008.

[27] G. Magoulas, V. Plagianakos, and M. Vrahatis. Hybrid methods using evolutionary algorithms for on-line training. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 2218–2223. IEEE, 2001.

[28] W. F. Mascarenhas. The bfgs method with exact line searches fails for non-convex objective functions. *Mathematical Programming*, 99(1):49–61, 2004.

[29] S. McLoone, M. D. Brown, G. Irwin, and G. Lightbody. A hybrid linear/nonlinear training algorithm for feedforward neural networks. *Neural Networks, IEEE Transactions on*, 9(4):669–684, 1998.

[30] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC bioinformatics*, 7(1):125, 2006.

[31] R. Mendes, P. Cortez, M. Rocha, and J. Neves. Particle swarms for feedforward neural network training. *learning*, 6(1), 2002.

[32] A. Miniani and R. D. Williams. Acceleration of back-propagation through learning rate and momentum adaptation. In *Proceedings of International Joint Conference on Neural Networks*, volume 1, pages 676–679, 1990.

[33] P. Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[34] S. Nallaperuma, F. Neumann, M. R. Bonyadi, and Z. Michalewicz. Evor: an online evolutionary algorithm for car racing games. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 317–324. ACM, 2014.

[35] E. Onieva, D. A. Pelta, J. Godoy, V. Milanés, and J. Pérez. An evolutionary tuned driving system for virtual car racing games: The autopia driver. *International Journal of Intelligent Systems*, 27(3):217–241, 2012.

[36] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607. ACM, 2002.

[37] N. Pavlidis, D. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. Vrahatis. Spiking neural network training using evolutionary algorithms. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 4, pages 2190–2194. IEEE, 2005.

[38] Y. Petalas, K. E. Parsopoulos, and M. N. Vrahatis. Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127, 2007.

[39] J. Quadflieg, M. Preuss, and G. Rudolph. Driving faster than a human player. In *Applications of Evolutionary Computation*, pages 143–152. Springer, 2011.

[40] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[41] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[43] P. P. San, S. H. Ling, and H. T. Nguyen. Hybrid particle swarm optimization based normalized radial basis function neural network for hypoglycemia detection. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–6. IEEE, 2012.

[44] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[45] A. Van Ooyen and B. Nienhuis. Improving the convergence of the back-propagation algorithm. *Neural Networks*, 5(3):465–471, 1992.

[46] T. A. A. Victoire and A. E. Jeyakumar. Hybrid pso–sqp for economic dispatch with valve-point effect. *Electric Power Systems Research*, 71(1):51–59, 2004.

[47] D. Wang and W.-Z. Lu. Forecasting of ozone level in time series using mlp model with a novel hybrid training algorithm. *Atmospheric Environment*, 40(5):913–924, 2006.

[48] J. Yu, S. Wang, and L. Xi. Evolving artificial neural networks using an improved pso and dpso. *Neurocomputing*, 71(4):1054–1060, 2008.

[49] C. Zhang, H. Shao, and Y. Li. Particle swarm optimisation for evolving artificial neural network. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2487–2490. IEEE, 2000.

[50] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu. A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation*, 185(2):1026–1037, 2007.