

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

5,500

Open access books available

136,000

International authors and editors

170M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Tuning Artificial Neural Network Controller Using Particle Swarm Optimization Technique for Nonlinear System

Sabrina Slama, Ayachi Errachdi and Mohamed Benrejeb

Abstract

This chapter proposes an optimization technique of Artificial Neural Network (ANN) controller, of single-input single-output time-varying discrete nonlinear system. A bio-inspired optimization technique, Particle Swarm Optimization (PSO), is proposed to be applied in ANN to avoid any possibilities from local extreme condition. Further, a PSO based neural network controller is also developed to be integrated with the designed system to control a nonlinear systems. The simulation results of an example of nonlinear system demonstrate the effectiveness of the proposed approach using Particle Swarm Optimization approach in terms of reduced oscillations compared to classical neural network optimization method. MATLAB was used as simulation tool.

Keywords: neural networks, particle swarm optimization, indirect control, nonlinear system

1. Introduction

We are interested, in this chapter, in adaptive system control of a class of single-input single-output (SISO) non-linear systems using neural network. In fact, this system control is a very general approach to adaptive control since one can combine in principle any parameter estimation scheme with any control strategy. In addition, its architecture is based on two neural network blocks corresponding to the system controller and the model identification of the dynamic behavior of the system [1, 2].

The use of artificial neural network (ANN) for identification, diagnosis, modeling and control has generated a lot of interest for quite some time now, because they have proved to be excellent function approximators, mapping any function to an arbitrary degree of accuracy, coupled with their ability for generalization, self-organization and self-learning [3].

Many architectures of neural networks are used. Among them, the most common and the most popular architecture is the multilayered perceptron, implemented with the standardized backpropagation algorithm. If the initial set of weights is not selected properly, this algorithm, employing a gradient descent search technique is seriously prone to getting trapped in local optimum solutions.

However, the calculations could slowly occurred and may even overflow or fluctuate between the optima. These limitations encouraged researchers to look for more powerful optimizations techniques that can help reach the optimal solution in an improved fashion, guarantee the convergence of control system and increase the learning speed [3].

A lot of optimization techniques of neural network are widely used. Among them, particle swarm optimization (PSO), the subject of this chapter, studied in different papers like in [3], and originates from the behavioral simulation of fish schooling and bird flocking. The conceptual model, at some point in the evolution of the algorithm, was an optimizer following which a number of parameters extra-neous to optimization were eliminated leading to the basic PSO algorithm [3].

This technique is used in many applications of neural network optimization like identification, control and modeling. For instance, in [4], the authors used the PSO based neural network optimization for prediction of diameter errors in a boring machine. In their work, they established an improvement in the quality of optimization of the neural networks and error compensations with the use of the PSO algorithm, which achieves a better machining precision with fewer numbers of iterations.

The PSO algorithm is proposed to get optimal the parameters of ANN. This algorithm is well used because it has convergent result and not require many iterations, so in relative calculation relative quick. PSO is a population-based approach, which uses the swarm intelligence generated by the cooperation and competition between the particles in a swarm. It has been emerged successfully to a wide variety of search and optimization problems.

For example, in [5], the authors compared the performance of the PSO technique with other EAs for both continuous and discrete optimization problems in terms of processing time, convergence speed, and quality of the results. In addition, in [6], the authors proposed a PSO learning algorithm that self-generates radial basis function neural network (RBFN) to deal with three non-linear problems. This proposed PSO allows a high accuracy within a short training time when determining RBFN with small number of radial basis functions. Then, in [7], a PSO algorithm was developed by the authors to find the optimum process parameters which satisfy given limit, tool wear and surface roughness values and maximize the productivity at the same time. Also, in [8], the authors described an evolutionary algorithm for evolving the ANN which was based on the PSO technique. Both the architecture and the weights of the ANN were adaptively adjusted according to the quality of the neural network until the best architecture or a terminating criterion was reached. Moreover, the performance of the basic PSO algorithm with the constriction PSO on some test functions of different dimensions was compared by [9] and they found that the use of constriction PSO with mutation provided significant improvement in certain cases.

Further, in [10], it is presented an improved PSO algorithm for neural network training employing a population diversity method to avoid premature convergence. Furthermore, in [11], the authors used the PSO technique to optimize the grinding process parameters such as wheel and workpiece speed, depth and lead of dressing, etc. subjected to suitable constraints with the objective of minimizing the production cost and obtaining the finest possible surface finish. As well as, by comparing the PSO algorithm results with genetic algorithms and quadratic programming techniques, the PSO algorithm gives the global optimum solution with the objective to obtain minimum cost of manufacturing, [12]. Equally, in [13], the authors applied ANN - PSO approach for selection of optimum process parameters for minimizing burr size in drilling process. Besides that, the PSO algorithm was applied for optimization of multi objective problem in tile manufacturing process [14] and also

for machinery fault detection [15]. Finally, in [16] the authors used PSO to tune the radial basis function networks for modeling of MIG welding process.

The PSO is well used in control system, for instance, in [17], the parameters of PID are tuned by ANN where their weights are optimized using PSO method to avoid any local minima/maxima in its searching procedure. In [18], the authors proposed a design of decentralized load-frequency controller for interconnected power systems with ac-dc parallel using PSO algorithm. The experiment result illustrated that their method have rapid dynamic response ability. In [19], the PSO algorithm is implemented to optimize the own five parameters of $PI^\lambda D^\delta$ controller via El-Khazali's approach in order to minimize several error functions satisfying some step response specifications such as the set of time domain and frequency domain constraints; overshoot, rise time and settling time. In [20], a comparative analysis of PSO algorithms is carried out, where two PSO algorithms, namely PSO with linearly decreasing inertia weight (LDW-PSO), and PSO algorithm with constriction factor approach (CFA-PSO), are independently tested for different PID structures.

In this chapter, a comparison of the performance of the PSO optimized neural network with the standard back-propagation is presented for the adaptive indirect control of nonlinear time-varying discrete system.

The present chapter is organized as follows. After this introduction, section 2 reviews the problem statement. Furthermore, in section 3 the neural network optimization methods are shown. In Section 4, tuning neural network controller using classical approach is presented. However, the section 5 details tuning neural network controller using PSO approach. An example of nonlinear system is studied, in section 6, to illustrate the proposed efficiency of the method. Section 7 gives the conclusion of this chapter.

2. Problem statement

The indirect adaptive control that is used, in this chapter, is composed of two blocks: a block of neural network model and a block of control system. The proposed control system is a neural network controller. At the simulation, it is assume that the neural network controller parameter's depending of the model parameter's as given in **Figure 1**.

In this architecture of indirect control, $r(k)$ is the desired value, $u(k)$ is the control law from the controller, $y(k)$ is the output of the nonlinear system, $yr(k)$ is the output of the neural network model, $e(k)$ is the identification error, $\hat{e}_c(k)$ is the estimated tracking error, $e_c(k)$ is the tracking error and k is the discrete time.

The aim of this chapter is to find a control law $u(k)$ to the nonlinear system, given by the Eq. (1), based on the tuning neural network controller's parameters in order that the system output $y(k)$ tracks, where possible, the desired value $r(k)$.

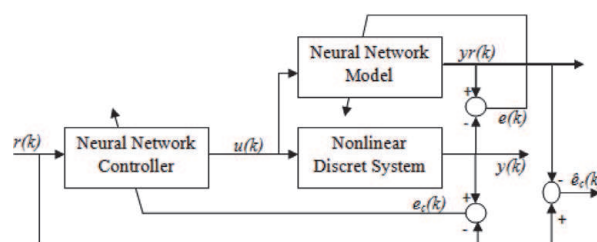


Figure 1.
 The architecture of indirect neural control.

$$y(k+1) = f(y(k), \dots, y(k-n_y), u(k), \dots, u(k-n_u)) \quad (1)$$

$f(\cdot)$ is the nonlinear function mapping specified by the model, n_y and n_u are the number of past output and input samples respectively required for prediction.

In this structure, the neural network controller and the neural network model must be updated at the same time. But, it is a difficult task to have a neural identifier learn the system fast enough to adapt to varying parameters. Therefore, the neural controller is ineffective on variations of the system parameters. In this chapter, to solve this problem we propose a fast learning algorithm based on a particle swarm optimization approach.

3. Neural network optimization methods

3.1 Gradient back-propagation algorithm

An Artificial Neural Networks (ANN) with randomly initialized weights usually shows poor results. That's why the most interesting characteristic of a neural network is its ability to learn, in other words, to adjust the weights of its connections according to the training data so that after training phase the faculty of generalization is obtained. This formulation turns the problem of learning into a problem of optimization.

In general, optimizing a system's parameters for a given task requires defining a metric that captures the inadequacy of the system for that task. This measure is called the cost function. Using an optimization algorithm, it is about finding the optimal parameters of the neural model that minimizes the cost.

For this kind of problem, there are two important classes of cost minimization search algorithms. Classical or gradient algorithms in which the central concept is that of direction of descent, are based on the derivatives of the cost function and any constraints, and advantageously made use of the specific information provided by the derivatives of different orders of these functions.

The alternative to these approaches is the use of meta-heuristics or heuristics such as genetic, stochastic, or evolutionary algorithms. Despite the notable advantage of not assuming regularity and their ability to locate the global minimum, these algorithms are strongly penalized by the relatively low convergence speeds and long computation times.

In the case of algorithms using the information provided by the derivatives of the functions defining the problem, each iteration comprises two main phases: the search for a direction of descent d_k and the determination of a step of descent η given by the following formula:

$$x_{k+1} = x_k + \eta d_k \quad (2)$$

The difference between these algorithms is manifested in the way these two steps are performed.

There are also three classes for such algorithms according to the strategy used to calculate the direction of descent:

1. gradient algorithms such as:

$$d_k = -\nabla f(x_k) \quad (3)$$

2. algorithms based on Newton's method in which the direction of descent is the solution of the system

$$\nabla^2 f(x_k) d_k = \nabla f(x_k) \quad (4)$$

3. the algorithms of the quasi-Newton type, in which an approximation H_k of the Hessian matrix evaluated in the iterates is built, the direction being then, as for the method of Newton, the solution of the linear system

$$H_k d_k = \nabla f(x_k) \quad (5)$$

The gradient back-propagation algorithm is the most widely used for weight adaptation, the goal of which is to find the appropriate combination of connection weights that minimizes the error function E defined by:

$$E = \frac{1}{2} \sum_k (y_k - yr_k)^2 \quad (6)$$

y_k and yr_k being, respectively, the desired output and the actual output of the k neuron for a given input vector.

This procedure is based on an extension of the Delta rule which involves a gradient descent and which consists in propagating an observation of the input of the neural network through the neural layer, to obtain the output values.

Compared to the desired outputs, the resulting errors allow the weights of the output neurons to be adjusted. Without the presence of the hidden layer, the knowledge of these errors allows a direct calculation of the gradient and makes the adjustment of the weights of these single neurons, easy as shown by the Delta rule. So for a network with hidden layers, ignoring the desired outputs of the hidden neurons, it thus remains impossible to know the errors of these neurons. So, as it is, this process cannot be used for weight adjustment of hidden neurons. The intuition which solves this difficulty and which gave rise to back-propagation was as follows: the activity of a neuron is linked to neurons of the preceding layer. Thus, the error of an output neuron is due to the hidden neurons of the previous layer in proportion to their influence; therefore according to their activation and the weights that connect the hidden neurons to the output neuron. Therefore, we seek to obtain the contributions of the L hidden neurons which gave the error of the output neuron k .

The back-propagation procedure consists in propagating the error gradient (error produced during the propagation of an input vector) in the network. In this phase, the propagation of an output neuron's error starts from the output layer to the hidden neurons.

It is therefore sufficient to retrace the original activation path backwards, starting from the errors of the output neurons, to obtain the error of all the neurons in the network. Once the corresponding error for each neuron is known, the weight adaptation relationships can be obtained.

3.2 Second order optimization method

Another class of methods, more sophisticated than the previous one, is based on second order algorithms, based on Newton's method which adapts the weights according to the following relation:

$$\Delta w = -H^{-1} \nabla E \quad (7)$$

where the element H_{ij} of the Hessian matrix H relates to the second partial derivatives of the cost function, compared to the weights. The elements of this matrix are defined by

$$H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \quad (8)$$

Like gradient-only methods, second-order methods determine the gradient by the back-propagation algorithm and generally approximate the Hessian matrix or its inverse, since the cost of its computation may quickly become prohibitive.

This type of method localizes, in a single iteration, the minimum of a quadratic empirical error criterion and requires several iterations when this criterion is not ideally quadratic.

In practice, the convergence of the corresponding algorithm towards an optimal solution is rapid so that a good number of error hyper-surfaces present a quadratic curvature in the immediate vicinity of the minima. This method nevertheless remains subject, when the error hyper-surfaces are complex, to convergence towards non-minimal solution points for which the gradient of the empirical error criterion is canceled out at the inflection points or at the saddle points. In addition, there are the possibilities of divergence of the algorithm when the Hessian matrix is not positive definite.

The evaluation and memorization of the inverse Hessian matrix, on which the second-order methods are based, is, however, a major handicap in the context of learning large networks.

But, the main drawback lies in the calculation of the second derivatives of E which is most often expensive and very difficult to carry out. A certain number of algorithms propose to get around this difficulty by using approximations of the Hessian matrix.

This approximation, at the basis of Gauss-Newton or Levenberg–Marquardt algorithms, is widely used in the identification of rheological parameters. This method is adapted especially for the problems of small dimensions since the computation of the Hessian matrix is easy. Whereas if the problem presents a large number of variables, it is generally advised to couple it with the conjugate gradient method or a Quasi-Newton method. Or, when the relative improvement in the objective function becomes too low, we automatically switch to the conjugate gradient method.

The Levenberg–Marquardt method, Marquardt method, another second-order method, very close to the Newton method described previously, in fact offers an interesting alternative by adjusting the weights as following:

$$\Delta w_{ij} = -[H + \mu I]^{-1} \nabla E \quad (9)$$

μ is the Levenberg–Marquardt parameter and I is the identity matrix.

This method, making a compromise between the direction of the gradient and Newton's method, has the particularity of adapting to the shape of the error surface. Indeed, for low values of μ , the Levenberg–Marquardt method approaches Newton's method, and for large values of μ , the algorithm is quite simply a function of the gradient, knowing that the parameter μ is automatically updated based on the convergence of each iteration.

Stabilization is possible thanks to a reiterative process, i.e. if an iteration diverges, it can be started again by increasing the parameter μ until a convergent iteration is obtained. However, the phenomenon of strong divergence when

approaching the optimum, inherent in Newton's method, is in no way suppressed here. At most, the divergence can be reduced.

Despite the interesting properties of this method, calculating the inverse of $(H + \mu I)$, makes its use tricky for heavy neural networks. As a result, like Newton's method, it is advisable to automatically switch to the conjugate gradient method when this divergence phenomenon appears. Second-order methods greatly reduce the number of iterations, but increase the computation time.

3.3 Heuristic optimization methods

The advantage of heuristic optimization methods is the minimization of non-derivable cost functions, even for a large number of parameters ($1 < n < 10^5$).

Among the effective methods, we distinguish the optimization algorithm by Particle Swarms introduced by Kennedy and Eberhart and improved by Clerc is an optimization technique by agents which is essentially inspired by the behavior social in flocks of birds or schools of fish.

In addition, genetic algorithms, evolutionary algorithms, also constitute stochastic optimization techniques inspired by the theory of evolution according to Darwin, now widely used in numerical optimization, when the functions to be optimized are complex, irregular, poorly known or in question. Combinatorial optimization.

These heuristic methods of the methods presented previously (Levenberg-Marquardt, Newton, Conjugate Gradient, ...) by three main aspects:

1. they do not require the gradient calculation,
2. they study a population as a whole while deterministic methods treat an individual who will evolve towards the optimum,
3. they involve random operations.

Experience has also shown that if the components as well as the evolution parameters are carefully tuned, it is possible to obtain extremely efficient and fast algorithms. However, this adjustment step can be very delicate and constitutes a drawback of the implementation of these methods.

4. Tuning neural network controller using classical approach

The architecture shown in **Figure 1** assumes the role of two neural blocks. Indeed, the weights of the neural model are adjusted by the identification error $e(k)$, however the weights of the neural controller are trained by the tracking error $e_c(k)$.

The multi-layer perceptron is used in the neural model and in the neural controller. Each block consists of three layers. The sigmoid activation function s is used for all neurons.

Concerning the neural network model, the j^{th} output layer of the hidden layer is described as follows

$$h_j = \sum_{i=1}^{n_1} w_{ji} x_i \quad j = 1, 2, \dots, n_2 \quad (10)$$

where n_1 is the number of nodes of the input layer, w_{ji} is the hidden weight, x_i is the input vector of the neural model, $x = [u(k), u(k-1), u(k-2), \dots]^T$, $u(k)$ is the control input to the system and n_2 is the number of nodes of the hidden layer given in the expression (3).

The output of the neural network model is given by the following equation

$$yr(k+1) = \lambda s \left(\sum_{j=1}^{n_2} w_{1j} s(h_j) \right) \quad (11)$$

where w_{1j} is the weight from the hidden layer to the output layer and λ is a scaling coefficient. The compact form of the output is given by the following equation

$$yr(k+1) = \lambda s(h_1) = \lambda s[w_1^T S(Wx)] \quad (12)$$

with

$$\begin{aligned} x &= [x_i]^T, i = 1, \dots, n_1, \\ W &= [w_{ji}], i = 1, \dots, n_1, j = 1, \dots, n_2, \\ S(Wx) &= [s(h_j)]^T, j = 1, \dots, n_2, \\ w_1 &= [w_{1j}]^T, j = 1, \dots, n_2. \end{aligned}$$

The incremental change of the hidden weights Δw_{ij} , $i = 1, \dots, n_1$ and $j = 1, \dots, n_2$, is

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial e} \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial h_j} \frac{\partial h_j}{\partial w_{ji}} \quad (13)$$

$$\Delta w_{ji} = \eta \lambda s'(h_1) S'(Wx) w_{1j} x^T e(k) \quad (14)$$

with η is the learning rate, $0 \leq \eta \leq 1$, $S'(Wx) = \text{diag}[s'(h_j)]^T$, $j = 1, \dots, n_2$, $s'(h_1)$ is the derivative of $s(h_1)$ defined as follows

$$s'(h_1) = s(h_1)(1 - s(h_1)) \quad (15)$$

$e(k)$ is the identification error which is given by

$$e(k) = y(k) - yr(k) \quad (16)$$

and the function cost which is given by the following equation

$$E = \frac{1}{2} \sum_{k=1}^N (e(k))^2 = \frac{1}{2} \sum_{k=1}^N (y(k) - yr(k))^2 \quad (17)$$

where N is the number of observations.

The incremental change of the hidden weights Δw_{ij} is used in the following equation

$$w_{ji}(k+1) = w_{ji}(k) + \Delta w_{ji}(k) \quad (18)$$

However, the output weights are updated by the following equation

$$w_{1j}(k+1) = w_{1j}(k) + \Delta w_{1j}(k) \quad (19)$$

where Δw_{1j} is

$$\Delta w_{1j} = -\eta \frac{\partial E}{\partial w_{1j}} = -\eta \frac{\partial E}{\partial e} \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_{1j}} \quad (20)$$

$$\Delta w_{1j}(k) = \eta \lambda e(k) s'(h_1) S(Wx) \quad (21)$$

Concerning the neural network controller, the j^{th} output layer of the hidden layer is

$$h_{cj} = \sum_{i=1}^{n_3} v_{ji} x_{1i} \quad j = 1, \dots, n_4 \quad (22)$$

where n_3 is the number of nodes of the input layer, v_{ji} is the hidden weight and x_{1i} is the input vector of the neural network controller $x_1 = [r(k), r(k-1), r(k-2), \dots]^T$, $r(k)$ is the desired value.

The output of the neural network controller is given by the following equation

$$u(k) = \lambda_c s \left(\sum_{j=1}^{n_4} v_{1j} s(h_{cj}) \right) = \lambda_c s \left(\sum_{j=1}^{n_4} v_{1j} s \left(\sum_{i=1}^{n_3} v_{ji} x_{1i} \right) \right) \quad (23)$$

where n_4 is the number of nodes of the hidden layer, λ_c is a scaling coefficient and v_{1j} is the output weight.

The compact form of the output of the neural network controller is given by the following equation

$$u(k) = \lambda_c s(h_{c1}) = \lambda_c s[v_1^T S(Vx_1)] \quad (24)$$

with

$$\begin{aligned} x_1 &= [x_{1i}]^T, i = 1, \dots, n_3, \\ V &= [v_{ji}], i = 1, \dots, n_3, j = 1, \dots, n_4, \\ S(Vx_1) &= [s(h_j)]^T, j = 1, \dots, n_4, \\ v_1 &= [v_{1j}]^T, j = 1, \dots, n_4. \end{aligned}$$

Concerning the hidden synaptic weights, they are updated by

$$v_{ji}(k+1) = v_{ji}(k) + \Delta v_{ji}(k) \quad (25)$$

where Δv_{ji} is given by

$$\Delta v_{ji} = -\eta_c \frac{\partial E_c}{\partial e_c} \frac{\partial e_c}{\partial y} \frac{\partial y}{\partial h_1} \frac{\partial h_1}{\partial s(h_j)} \frac{\partial s(h_j)}{\partial h_j} \frac{\partial h_j}{\partial u} \frac{\partial u}{\partial h_{c1}} \frac{\partial h_{c1}}{\partial h_{cj}} \frac{\partial h_{cj}}{\partial v_{ji}} \quad (26)$$

with η_c is the learning rate, $0 \leq \eta_c \leq 1$ and the function cost defined as follows

$$E_c = \frac{1}{2} \sum_{k=1}^N (e_c(k))^2 = \frac{1}{2} \sum_{k=1}^N (y(k) - r(k))^2 \quad (27)$$

where N is the number of observations and $e_c(k)$ is the tracking error which is given by the following equation

$$e_c(k) = y(k) - r(k) \quad (28)$$

where $r(k)$ is the desired output. So Δv_{ji} comes

$$\Delta v_{ji} = \eta_c \lambda_c e_c(k) s'(h_1) w_{1j} S'(Wx) w_{ji} s'(h_{c1}) v_{1j} S'(Vx_1) x_1^T \quad (29)$$

with $S'(Vx_1) = \text{diag}[s'(h_j)]^T, j = 1, \dots, n_4$.

The output synaptic weights of the neural network controller are updated as

$$v_{1j}(k+1) = v_{1j}(k) + \Delta v_{1j}(k) \quad (30)$$

where Δv_{1j} is given by

$$\Delta v_{1j} = -\eta_c \frac{\partial E_c}{\partial v_{1j}} = -\eta_c \frac{\partial E_c}{\partial e_c} \frac{\partial e_c}{\partial h_{c1}} \frac{\partial h_{c1}}{\partial v_{1j}} \quad (31)$$

So

$$\frac{\partial e_c(k)}{\partial h_{c1}} = \frac{\partial (y(k) - r(k))}{\partial h_{c1}} = \frac{\partial y(k)}{\partial h_{c1}} = \frac{\partial y(k)}{\partial u(k)} \frac{\partial u(k)}{\partial h_{c1}} \quad (32)$$

and the Eq. (31) becomes

$$\Delta v_{1j} = -\eta_c \frac{\partial E_c}{\partial e_c} \frac{\partial y(k)}{\partial u} \frac{\partial u(k)}{\partial h_{c1}} \frac{\partial h_{c1}}{\partial v_{1j}} \quad (33)$$

or in Eq. (33), $y(k)$ does not depend on h_1 , for this reason we use $yr(k)$ instead of $y(k)$ under the condition that the neural model is equal to the system behavior which gives

$$\frac{\partial y(k)}{\partial u} = \frac{\partial yr(k)}{\partial u} = \frac{\partial yr(k)}{\partial h_1} \frac{\partial h_1}{\partial s(h_j)} \frac{\partial s(h_j)}{\partial h_j} \frac{\partial h_j}{\partial u} \quad (34)$$

from where approximately

$$\Delta v_{1j} = -\eta_c \frac{\partial E_c}{\partial e_c(k)} \frac{\partial e_c(k)}{\partial y(k)} \frac{\partial yr(k)}{\partial h_1} \frac{\partial h_1}{\partial s(h_j)} \frac{\partial s(h_j)}{\partial h_j} \frac{\partial h_j}{\partial u(k)} \frac{\partial u(k)}{\partial h_{c1}} \frac{\partial h_{c1}}{\partial v_{1j}} \quad (35)$$

the obtained incremental change Δv_{1j} is rewritten as

$$\Delta v_{1j} = \eta_c \lambda_c e_c(k) s'(h_1) w_{1j} S'(Wx) w_{ji} s'(h_{c1}) S(Vx_1) \quad (36)$$

In this section, we used a fixed learning rate, $\eta(k)$ (respectively $\eta_c(k)$), and a derivative of sigmoid function $s'(h_1) = s(h_1)(1 - s(h_1))$.

This approach has two drawbacks. First, to find a suitable fixed learning rate $\eta(k)$ (respectively $\eta_c(k)$), several tests are called which decreases the on-line operation. Second, when we use this type of derivative of sigmoid function, a large amount of error should not be spread to the weights of the output layer and the learning speed becomes very slow. In order to increase the learning speed, some of the new proposed approaches are proposed in the next section.

5. Tuning neural network controller using particle swarm optimization

An alternative technique is proposed, in this section, to optimize the neural network controller by implementing Particle Swarm Optimization algorithm. This algorithm works like animal behavior on finding foods and avoiding danger, where they will coordinate with each other to find the best position to settle. Likewise, PSO is directed by the movement of the best individual from the population, known as the social compound, and their own experience, known as the cognitive compound. The algorithm moves the set of solutions to find the best solution among them.

5.1 Mathematical formulation

In this study, the Particle Swarm Optimization Feedforward Neural Network (PSO NN) is applied to a multi-layered perceptron where the position of each particle, in a swarm, represents the set of synaptic weights of the neural network for the current iteration. The dimensionality of each particle is the number of synaptic weights.

Let us consider a search space of dimension D . A particle i of the swarm is modeled by a position vector

$$x_{ij} = [x_{i1}, x_{i2}, \dots, x_{iD}]^T \quad (37)$$

and a velocity vector denoted

$$v_{ij} = [v_{i1}, v_{i2}, \dots, v_{iD}]^T \quad (38)$$

There is no concept of backpropagation in PSO NN where the direct neural network produces the learning error, objective function of each particle, based on the set of synaptic weights and biases, the positions of the particles. Each particle moves in the weighting space trying to minimize the learning error and keeps in memory the best position through which it passed, denoted

$$Pbest_{ij} = [pbest_{i1}, pbest_{i2}, \dots, pbest_{iD}]^T \quad (39)$$

whereas the best position reached by the swarm is denoted

$$Gbest_{ij} = [gbest_{i1}, gbest_{i2}, \dots, gbest_{iD}]^T \quad (40)$$

Changing the position means updating the synaptic weights of the neural network controller to generate the proper control law by reducing tracking error.

In each iteration k , the particles update their position by calculating the new velocity and move to the new position. At the iteration $(k + 1)$, the velocity vector is calculated as follows:

$$v_{ij}(k+1) = wv_{ij}(k) + c_1r_1[pbest_{ij}(k) - x_{ij}(k)] + c_2r_2[gbest_{ij}(k) - x_{ij}(k)] \quad (41)$$

where:

w being a variable parameter making it possible to control the changing of the particle at the next iteration,

wv_{ij} is a physical changing component,

$c_1r_1[pbest_{ij}(k) - x_{ij}(k)]$ is a cognitive changing component,

$c_2r_2[gbest_{ij}(k) - x_{ij}(k)]$ a social component of changing,

c_1 and social c_2 are respectively, two cognitive confidence coefficients and social which are present the degree of attraction towards the best position of a particle and that of these informants,

r_1 and r_2 are two random numbers drawn uniformly in the interval $[0, 1]$ represent the proper exploration of particles in the search space.

The smallest learning error of each particle $Pbest_i$ and the smallest learning error found in the whole learning process $Gbest_i$ are applied to produce a fit of the positions towards the best solution or the targeted tracking error.

The position, at the iteration $(k+1)$, of particle i is then defined as follows:

$$x_{ij}(k+1) = x_{ij}(k) + v_{ij}(k) \quad (42)$$

Once the change in positions has taken place, an update affects both $Pbest_i$ and $Gbest_i$ vectors. At the iteration $(k+1)$, these two vectors will be updated according to the following two formulations:

$$Pbest_i(k+1) = \begin{cases} Pbest_i(k) & \text{if } f(x_i(k)) \geq f(Pbest_i(k)) \\ x_i(k) & \text{else} \end{cases} \quad (43)$$

and

$$Gbest_i(k+1) = Arg \min_i [Pbest_i(k+1)] \quad (44)$$

The algorithm is executed as long as one of the three, or all at the same time, of the following convergence criteria is verified:

- the maximum number of iterations defined has not been reached,
- the variation in particle speed is close to zero,
- the value of the objective is satisfactory, with respect to the following relation:

$$|f(gbest_{jk}(k)) - f(gbest_{jk}(k-\alpha))| \leq \epsilon, \quad \text{with } \alpha \in [1, N] \quad (45)$$

The parameter ϵ represents a tolerance chosen, most often, of the order of 10^{-5} and N is a number of iterations chosen of the order of 10.

5.2 The proposed algorithm of particle swarm optimization

In this section, a summary of the proposed algorithm of the PSO neural network controller is presented.

- Random initialization of the positions and velocity of the N particles in the swarm,
- **For** $k: 1..N$ **Do**,
- Repeat,
- **For** all particles i **Do**,
- Calculation of the control law $u_i(k)$ from the controller input vector $x_c(k)$,
- Calculation of the outputs of the system $y_i(k + 1)$,
- Evaluation of the positions of particles in the research space,
- **If** the current positions of particle i produce the best objective function in its history **Then**,
- $Pbest_i \leftarrow e_{c_i}$,
- **If** the objective function of particle i is the best overall objective function **Then**,
- $Gbest_i \leftarrow e_{c_i}$,
- End If,
- End If,
- End For,
- Moving of particles according to Eqs. (41) and (42),
- Evaluation of particle positions,
- Update $Pbest_i$ and $Gbest_i$ according to Eqs. (43) and (44),
- **Until** reaching the stop criterion,
- End of PSO.

In this section, we have proposed the PSO optimization steps of an indirect neural network adaptive controller. The corresponding algorithm will be applied for discrete SISO nonlinear systems.

6. Results and discussion

In this section, a time-varying nonlinear discrete systems is used which is described by the input–output model in the following Equation [21].

$$y(k + 1) = \frac{y(k)y(k - 1)y(k - 2)u(k - 1)(y(k - 2) - 1) + u(k)}{a_0(k) + a_1(k)y^2(k - 1) + a_2(k)y^2(k - 2)} \quad (46)$$

where $y(k)$ and $u(k)$ are respectively the output and the input of the time-varying nonlinear system at instant k ; $a_0(k)$, $a_1(k)$ and $a_2(k)$ are given by

$$\begin{cases} a_0(k) = 1 \\ a_1(k) = 1 + 0.2 \cos(k) \\ a_2(k) = 1 + 0.2 \sin(k) \end{cases} \quad (47)$$

The trajectory of $a_1(k)$ and $a_2(k)$ are given in the following **Figure 2**.

In this section, in order to examine the effectiveness of the proposed algorithm of neural network controller and the PSO neural network controller different performance criteria are used. Indeed, the mean squared tracking error (MSE_{e_c}) and the mean absolute tracking error (MAE_{e_c}) are, respectively, given by respectively, given by

$$MSE_{e_c} = \frac{1}{N} \sum_{k=1}^N (y(k) - r(k))^2 \quad (48)$$

and

$$MAE_{e_c} = \frac{1}{N} \sum_{k=1}^N (y(k) - r(k)) \quad (49)$$

where $y(k)$ is the time-varying system output, $r(k)$ is the desired value and the used number of observations N is 100.

In this simulation, the desired value, $r(k)$, is given in the following

$$r(k) = \sin(2\pi k/25); \quad (50)$$

6.1 Simulation system using classical NN controller

In this section, we examine the effectiveness of the used classical neural network controller in the adaptive indirect control system. Indeed, in offline phase, using a reduced number of observations ($M = 3$) to find, either, the parameter initialization of the neural network parameters (w_{1j} , w_{ji} , v_{1j} , v_{ji}).

In online phase, at instant $(k + 1)$, we use the input vector of the neural network controller $x_1 = [x_r(k), x_r(k - 1), x_r(k - 2), x_r(k - 3), x_r(k - 4)]^T$. The results of simulation are given by **Figures 3–5**.

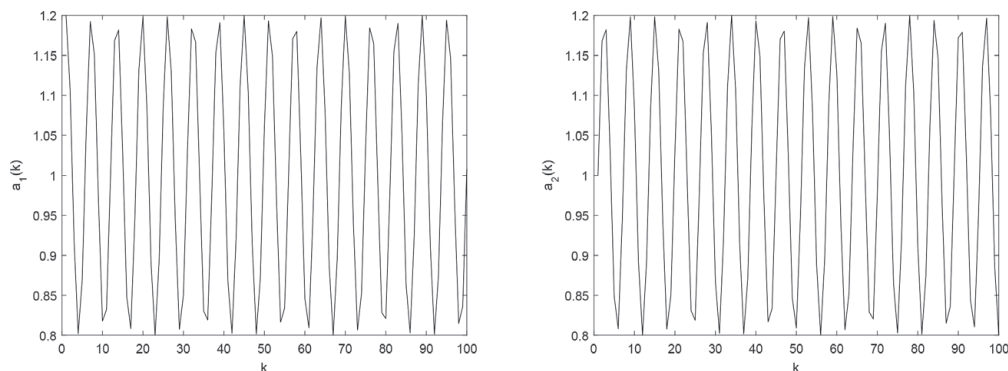


Figure 2.
 $a_1(k)$ and $a_2(k)$ trajectories.

In this case, both neural network model and neural network controller consist of single input, 1 hidden layer with 8 nodes, and a single output node, identically. The used scaling coefficient is $\lambda = \lambda_c = 1$ and $\varepsilon_1 = \varepsilon_2 = 10^{-2}$.

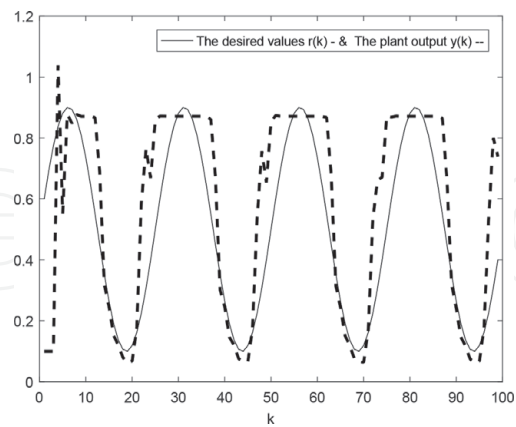


Figure 3.
The NN control system output and the desired values.

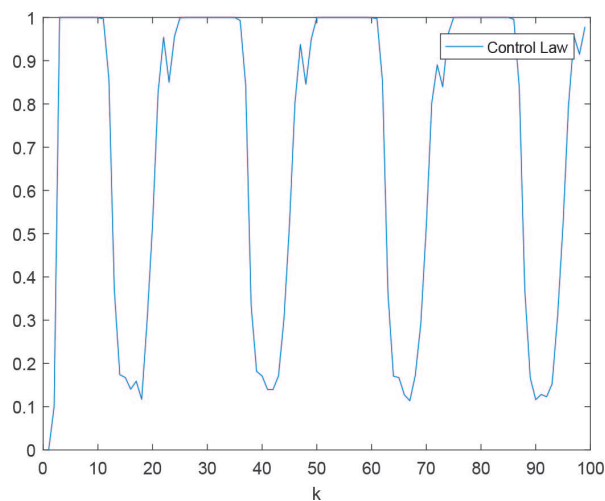


Figure 4.
The control law.

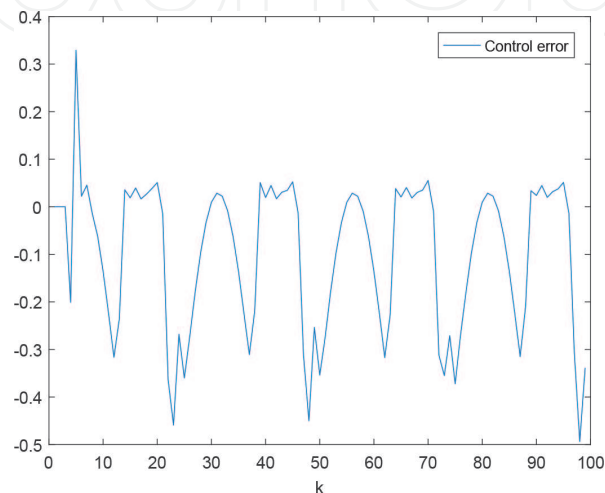


Figure 5.
The control error.

Using a multilayered perceptron architecture, three layers: one input layer, one hidden layer and one output layer. The result of simulation are given by the following figures.

6.2 Simulation system using PSO NN controller

The PSO parameters values are respectively the number of variables ($m = 50$), the population size ($pop = 10$), the maximum of inertia weight 0.9, the minimum of the inertia weight 0.4, the first acceleration factor ($c1 = 2$) and the second acceleration factor ($c2 = 2$).

Using a multilayered perceptron architecture, three layers: one input layer, one hidden layer and one output layer. The result of simulation are given by the following figures.

Figure 6 presents the control system output and the desired values. In this case, the neural network parameters of controller are optimized by PSO technique. A concordance between the desired values and the control system output is noticed, although the time-varying parameters.

However, **Figures 7 and 8** present respectively the control law and the control error. These figures reveal that the PSO NN controller has smaller errors than the other controller.

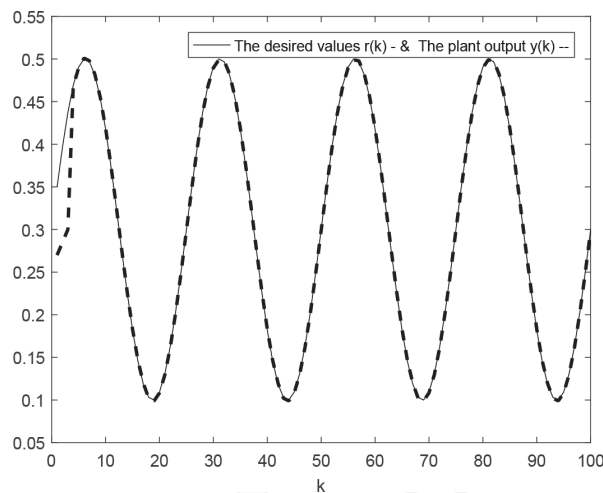


Figure 6.
The PSO NN control system output and the desired values.

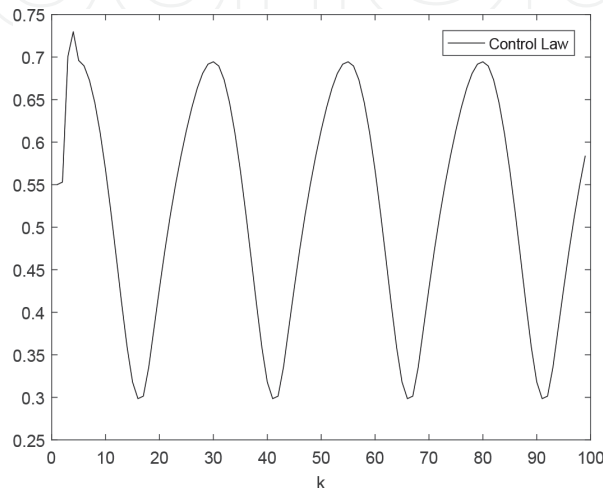


Figure 7.
The control law.

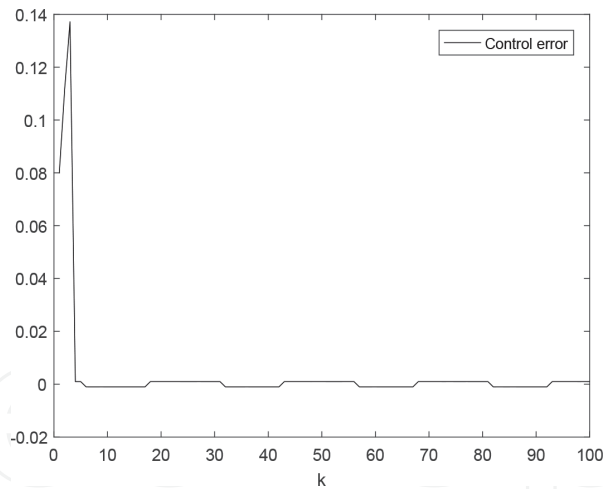


Figure 8.
 The control error.

	NN controller	PSO NN controller
η	variable	variable
MSE_{e_c}	0.0349	$3.7730e - 04$
$\max(e_c)$	0.3291	0.1372
$\min(e_c)$	-0.4907	$-9.9998e - 04$
MAE_{e_c}	0.1305	0.0043
time (s)	323.829	100.926

Table 1.
 The influence of the PSO technique in the control error.

Table 1 presents the influence of the PSO technique in the control error.

From **Table 1** we observe that, using the neural network controller, the PSO neural network controller has the smallest performance criteria in the control error $e_c(k)$. These results are shown in **Figures 6–8**.

6.3 Effect of disturbances

An added noise $v(k)$ is injected to the output of the time-varying nonlinear system in order to test the effectiveness of the proposed optimization technique of the neural network controller. To measure the correspondence between the system output and the desired value, a Signal Noise Ratio (SNR) is taken from the following equation:

$$SNR = \frac{\sum_{k=0}^N (y(k) - \bar{y})}{\sum_{k=0}^N (v(k) - \bar{v})} \quad (51)$$

with $v(k)$ is a noise of the measurement of symmetric terminal δ , $v(k) \in [-\delta, \delta]$, \bar{y} and \bar{v} are an output average value and a noise average value respectively. In this paper, the taken SNR is 5%.

Using the desired value $r(k)$, the sensitivity of the proposed neural network controller is examined in **Table 2**.

	NN controller	NN PSO controller
η_c	variable	variable
MSE_{e_c}	0.0351	$3.7728e - 04$
MAE_{e_c}	00.1247	0.0042
$max(e_c)$	0.3123	0.1372
$min(e_c)$	0.5000	$-9.9999e - 04$
time (s)	44.456972	337.728385

Table 2.

The influence of the PSO optimization in the control error.

From this table, we observe that, using the PSO as a method to optimize the parameters of neural network controller, we have got the smallest performance criteria in the control error.

According to the obtained simulation results, the lowest MSE_{e_c} , MAE_{e_c} and $max(e_c)$ are obtained using a combination between the neural network controller and the PSO technique, although the added disturbance in the system output and the time-varying parameters thanks to the PSO technique.

7. Conclusion


In this chapter, a comparative study between the neural network controller and neural network PSO controller is proposed and is applied with success in indirect adaptive control. For instance, the lowest MSE_{e_c} , MAE_{e_c} , $min(e_c)$ and $max(e_c)$ are obtained and it is proved that the PSO method is the best. The effectiveness of the proposed algorithm is successfully applied to single-input single-output system, with and without disturbances, and it proved its robustness to reject disturbances and to accelerate the speed of the learning phase of the neural model and neural controller.

Author details

Sabrina Slama, Ayachi Errachdi* and Mohamed Benrejeb
Tunis El Manar University, National Engineering School of Tunis, Department of
Electrical Engineering, Le Belvédère B.P. 37, Tunis, Tunisia

*Address all correspondence to: errachdi_ayachi@yahoo.fr

IntechOpen

© 2021 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Slama S., Errachdi A. and Benrejeb M., Adaptive PID controller based on neural networks for MIMO nonlinear systems, *Journal of Theoretical and Applied Information Technology*, **97**, no. 2, pp. 361–371, 2019.
- [2] Errachdi A. and Benrejeb M., Performance comparison of neural network training approaches in indirect adaptive control, *International Journal of Control, Automation and Systems*, **16**, no. 3, pp. 1448–1458, 2018.
- [3] Saurabh G., Karali P. and Surjya K.P., Particle swarm optimization of a neural network model in a machining process, *Sadhana* **39**, Part 3, June 2014, pp. 533–548. *Indian Academy of Sciences*
- [4] Zhou J., Duan Z., Li Y., Deng J. and Yu D., PSO-based neural network optimization and its utilization in a boring machine. *J. Material Process Technol.* **178**, pp. 19–23, 2006.
- [5] Elbeltagi E., Hegazy T. and Grierson D., Comparison among five evolutionary-based optimization algorithms. *Advanced Eng. Informatics* **19**, pp. 43–53, 2005.
- [6] Feng H.M., Self-generation RBFNs using evolutionary PSO learning. *Neurocomputing* **70**, pp. 241–251, 2006.
- [7] Karpat Y. and Ozel T., Hard turning optimization using neural network modelling and swarm intelligence. *Transactions of NAMRI/SME* **33**, pp. 179–186, 2005.
- [8] Zhang, R., Tao, J., Lu, R. and Jin, Q. Decoupled ARX and RBF neural network modeling using PCA and GA optimization for nonlinear distributed parameter systems. *IEEE Transactions on Neural Networks and Learning Systems*, **29**, no. 2, pp. 457–469, 2018.
- [9] Stacey A., Jancic M. and Grundy I., Particle swarm optimization with mutation. *Proceedings of IEEE*, pp. 1425–1430, 2003.
- [10] Zhao F., Ren Z., Yu D. and Yang Y., Application of an improved particle swarm optimization algorithm for neural network training. *Proceedings of IEEE International Conference on Neural Networks and Brain, Beijing, China*, pp. 1693–1698, 2005.
- [11] Asokan P., Baskar N., Babu K., Prabhakaran G. and Saravanan R., Optimization of surface grinding operations using particle swarm optimization technique. *J. Manufacturing Sci. Eng.* **127**, pp. 885–892, 2005.
- [12] Haq A.N., Sivakumar K., Saravanan R. and Karthikeyan K., Particle swarm optimization (PSO) algorithm for optimal machining allocation of clutch assembly. *Int. J. Advance Manufacturing Technol.* **27**, pp. 865–869, 2006.
- [13] Gaitonde V.N. and Karnik S.R., Minimizing burr size in drilling using artificial neural network (ANN)-particle swarm optimization (PSO) approach. *J. Intelligent Manufacturing* **23**, pp. 1783–1793, 2012.
- [14] Navalertporn T. and Afzulpurkar N.V., Optimization of tile manufacturing process using particle swarm optimization. *Swarm and Evolutionary Computation* **1**, pp. 97–109, 2011.
- [15] Samanta B. and Nataraj C., Use of particle swarm optimization for machinery fault detection. *Eng. Appl. Artificial Intelligence*, **22**, pp. 308–316, 2009.
- [16] Malviya R. and Pratihari D.K., Tuning of neural networks using particle swarm optimization to model

MIG welding process. Swarm and Evolutionary Computation **1**, pp. 223-235, 2011.

[17] Garro B.A. and Vazquez R.A., Designing Neural Networks Using Particle Swarm Optimization, Research Article, Computational Intelligence in Neuroscience, Vol. 2015.

[18] Selvakumaran S., Parthasarathy S., Karthigaivel R. and Rajasekaran V., Optimal decentralized load frequency control in a parallel ac-dc interconnected power system through hvdc link using pso algorithm, Energy Procedia **14**, pp. 1849–1854, 2012.

[19] Shaher M., Reyad E. and Iqbal M.B. Tuning PID and $PI^{\lambda}D^{\delta}$ controllers using particle swarm optimization algorithm via El-Khazali's Approach, Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics (AMEE'19) AIP Conf. Proc. 2172, 050003–1–050003-8; <https://doi.org/10.1063/1.5133522> Published by AIP Publishing. 978-0-7354-1919-3/30.00

[20] Stimac G., Braut S. and Ziguli R, Comparative analysis of PSO algorithms for PID controller tuning, Chinese Journal of Mechanical Engineering, **27**, No. 5, 2014.

[21] Narendra K.S. and Parthasarthy K., Identification and control of dynamical systems using neural networks, IEEE Trans. on Neural Networks, **1**, 1, 4–27, 1990.