Journal of Artificial Intelligence in Electrical Engineering, Vol. 1, No. 2, September 2012

Neuro-Optimizer: A New Artificial Intelligent Optimization Tool and Its Application for Robot Optimal Controller Design

Mohammad Pourmahmood Aghababa
Young Researchers Club, Ahar Branch, Islamic Azad University, Ahar, Iran,
Email: m.pour13@gmail.com

ABSTRACT

The main objective of this paper is to introduce a new intelligent optimization technique that uses a prediction-correction strategy supported by a recurrent neural network for finding a near optimal solution of a given objective function. Recently there have been attempts for using artificial neural networks (ANNs) in optimization problems and some types of ANNs such as Hopfield network and Boltzmann machine have been applied in combinatorial optimization problems. However, ANNs cannot optimize continuous functions and discrete problems should be mapped into the neural networks architecture. To overcome these shortages, we introduce a new procedure for stochastic optimization by a recurrent artificial neural network. The introduced neuro-optimizer (NO) starts with an initial solution and adjusts its weights by a new heuristic and unsupervised rule to compute the best solution. Therefore, in each iteration, NO generates a new solution to reach the optimal or near optimal solutions. For comparison and detailed description, the introduced NO is compared to genetic algorithm and particle swarm optimization methods. Then, the proposed method is used to design the optimal controller parameters for a five bar linkage manipulator robot. The important characteristics of NO are: convergence to optimal or near optimal solutions, escaping from local minima, less function evaluation, high convergence rate and easy to implement.

KEYWORDS: numerical optimization, neural networks, objective function, weight updating, five bar linkage manipulator robot.

1. INTRODUCTION

The objective of optimization is to seek values for a set of parameters that maximize or minimize objective functions subject to certain constraints. In recent years, many optimization algorithms have been introduced. Some of these algorithms are traditional optimization algorithms. Traditional optimization algorithms use exact methods to find the best solution. The idea is that if a problem can be solved, then the algorithm should find the global best solution. However, as the search space increases the objective value of these algorithms increases. Therefore, when the search space complexity increases the exact algorithms can be slow to find the global optimum. Linear and nonlinear programming, brute force or exhaustive search and divide and conquer methods are some of the most common exact optimization methods.

Calculus provides the tools and elegance for finding the optimum value of many objective functions. It quickly finds a single optimum but requires a search scheme to find the global optimum. Continuous functions with analytical derivatives are necessary (unless derivatives are taken numerically, which results in even more function evaluations plus a loss of accuracy). If there are too many variables, then it is difficult to find all the extrema. The gradient of the objective function serves as the compass heading solution to the steepest downhill path. It works well when the optimum is nearby, but cannot deal with cliffs or boundaries, where the gradient cannot be calculated.

Other optimization algorithms are stochastic algorithms, consisted of intelligent, heuristic and random methods. Stochastic algorithms have several advantages compared to other algorithms as follows:

1) Stochastic algorithms are generally easy to implement.
2) They can be used efficiently in a multiprocessor environment.
3) They do not require the problem definition function to be continuous.
4) They generally can find optimal or near-optimal solutions.

There are several stochastic algorithms such as: Genetic algorithms (GA) (Holland, 1975), Guided Local Search (GLS) (Voudouris, 1997), Tabu Search (TS) (Glover,1989), Variable Neighborhood Search (VNS) (Mladenovic and Hansen, 1997), Iterated Local Search (ILS) (Stützle, 1999), Simulated Annealing (SA) (Kirkpatrick et al. 1983), Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende, 1995), Memetic Algorithms (MA) (Moscato, 1989), Scatter Search (SS) (Cung et al. 1997), Ant Colony Optimization (ACO) (Marco Dorigo et al. 1999), Particle Swarm Optimization (PSO) (Kennedi and Eberhart 1995) and Shuffled Frog Leaping algorithm (SFL) (Eusuff, Lansey 2003), etc. These algorithms are implemented in many optimization problems and they have many applications in practical problems.

Artificial neural networks (ANNs) have been introduced as an effective tool in artificial intelligence field. Artificial neural networks have been used in many fields of science and engineering for many applications such as function approximation, prediction, pattern classification and control.

Among many types of the ANNs, the Hopfield network [1], the Boltzmann machine [2], the Mean Field network [3], the Gaussian machine [4], the Self Organizing Map network [5] and several others can be used as optimizers. Several authors have suggested the use of the neural networks as a tool to provide approximate solutions for combinatorial optimization problems such as the traveling salesman problem [6, 7], scheduling problems [8], [9], graph problems [10], [11], Knapsack Problems [12], [13], Placement Problems [14],vehicle routing problems[15], [16], Satisfaction Problems [17], [18], Large Scale Puzzles [19], channel assignment problems [20], [21], Circuit Partitioning [22], etc.

Hopfield optimizer solves combinatorial optimization problems by gradient descent, which has the disadvantage of being trapped in local minima [23]. Mean Field, Boltzmann and Gaussian machines are stochastic in nature and allow escaping from local optima. Moreover, In order to use a neural optimizer to solve combinatorial optimization problems, one must cast problems into the neural network model. In other words, the constraints and the objective function should be mapped in energy function of neural network. However, using this mapping procedure may require higher order neural networks for solving the problem. There is also no direct method for mapping constrained optimization problems on to a neural network except through addition of terms in the energy function which penalize violation of the constraints. In addition, SOM network is only applicable to Euclidean problems. Therefore, application of artificial neural networks for optimization problems is restricted. In fact, there are two major restrictions: 1) the problem should be discrete and 2) the problem should be mapped onto the neural network. These mappings are not possible for many problems and many of the real world optimization problems are continuous.

In this paper, a recurrent artificial neural network called neuro-optimizer (NO) is introduced to overcome the mentioned shortages of the optimizer neural networks. The neuro-optimizer is a recurrent neural network that can evaluate stochastic optimization and adjust its weights by a new unsupervised heuristic rule to achieve optimal or near optimal solutions. There is no mapping in NO procedure and any continuous problem can be easily optimized. NO has not any train or test phases, because it updates its weights during optimization process using the heuristic unsupervised rule. The proposed neural optimizer is very fast and easy to implement. These claims can be shown by using simulation results in finding the minimums of several benchmark functions. For comparison, the results of NO are compared to the results of two well known intelligent optimization methods, Genetic Algorithm (GA) [24] and Particle Swarm Optimization (PSO) [25].

PID (Proportional-Integral-Derivative) control is one of the earliest control strategies. It has been widely used in the industrial control field. Its widespread acceptability can be recognized by: the familiarity with which it is perceived amongst researchers and practitioners within the control community, simple structure and effectiveness of algorithm, relative ease and high speed of adjustment with minimal down-time and wide range of applications where its reliability and robustness produces excellent control performances. However, successful applications of PID controllers require the satisfactory tuning of three parameters (which are proportional gain (KP), integral time constant (KI) and derivative time constant (KD)) according to the dynamics of the process. Unfortunately, it has been quite difficult to tune properly the gains of PID controllers because many industrial plants are often burdened with problems such as high order, time delays and nonlinearities [26].

Traditionally, these parameters are determined by a trial and error approach. Manual tuning of PID controller is very tedious, time consuming and laborious to implement, especially where the performance of the controller mainly depends on the experiences of design engineers. In recent years, many tuning methods have been proposed to reduce the time consumption on determining the three controller parameters. The most well known tuning method is the Ziegler-Nichols tuning formula [27]; it determines suitable parameters by observing a gain and a frequency on which the plant becomes oscillatory.

Considering the limitations of the Ziegler-Nichols method and some empirical techniques in raising the performance of PID controller, recently artificial intelligence techniques such as fuzzy logic [28], fuzzy neural network [29] and some stochastic search and optimization algorithms such as simulated annealing [30], genetic algorithm [31], particle swarm optimization approach [26], immune algorithm [32] and ant colony optimization [33] have been applied to improve the performances of PID controllers. In these studies, it has been shown that these approaches provide good solutions in tuning the parameters of PID controllers. However, there are several causes for developing improved techniques to design PID controllers. One of them is the important impact it may give because of the general use of the controllers. The other one is the enhancing operation of PID controllers that can be resulted from improved design techniques. Finally, a better tuned optimal PID controller is more interested in real world applications.

This paper proposes the NO technique as a new optimization algorithm. The proposed method is applied for determining the optimal values for parameters of PID controllers. Here, we formulate the problem of designing PID controller as an optimization problem and our goal is to design a controller with high performance by adjusting four performance indexes, the maximum overshoot, the settling time, the rise time and the integral absolute error of step response. An optimal PID controller is designed for a five bar linkage manipulator robot using NO algorithm. The advantages of this methodology are that it is a simple method with less computation burden, high-quality solution and stable convergence specifications.

The rest of this paper is organized as follows. In the Section 2, the neuro-optimizer is explained in details and its optimization algorithm is described. Section 3 deals with the description of genetic algorithm and particle swarm optimization as two well known optimization algorithms. In section 4, the neuro-optimizer is compared to genetic algorithm and particle swarm optimization technique by means of simulations. Finally, the paper ends with some conclusions in Section 5.

2. BASICS OF NEURO-OPTIMIZER

Our lives confront us with many opportunities for optimization. What is the best route to work? Which project do we tackle first? When designing something, we shorten the length of this or reduce the weight of that, as we want to minimize the objective value or maximize the appeal of a product. In fact, we do optimization in our lives and this work is performed by our nervous system. If we want to find an optimum value, without any use of calculation devices, the following process may be occurred. We start with selecting a random solution in search space and then calculating its objective function value. Afterwards we select another solution and compute its objective value. Comparing these two results, we can find a better solution as third solution. Therefore, by this way, we use the results of previous solutions to determine a better solution in the next step. This process can be continued until a satisfactory solution is obtained.

Now, we try to model the mentioned human optimization process. We can use artificial neural networks. ANNs have been introduced as a model of human neural networks. As mentioned earlier, ANNs can do two important tasks: function approximation and pattern classification. Recently there are also attempts for using ANNs in optimization problems. But there are two major restrictions, first, the problem should be discrete and second, the problem should map onto neural network that these mappings are not possible for many problems. Therefore, introducing a new human inspired tool that can easily and efficiently optimize any continuous problems is required.

With considering the mentioned assumptions about the quality of human optimization, we introduce a neuro-optimizer (NO). A neuro-optimizer is a recurrent neural network that works as a stochastic intelligent optimizer, for continuous functions. NO uses a prediction-correction strategy supported by a recurrent neural network to find an optimum of a given function. In each iteration, using an unsupervised heuristic weight updating rule, NO produces new better solutions, stochastically. The stochastic nature of NO prevents of being in any local optimum trap. Heuristic weight updating rule updates the weights to next solutions and move toward global or near global solutions, rapidly. This causes a fast convergence rate with less function evaluations. Figure (1) shows the schematic diagram of a neuro-optimizer with

$x=(x_1,x_2,...,x_m)$: A solution in search space.

m: Dimension of the search space.

Q: Number of neurons in the hidden layer.

$y=(y_1,y_2,...,y_m)$: Next solution generated by neuro-optimizer.

w_{ij}: Connection weight between ith neuron in input layer and jth neuron in hidden layer.

u_{jk}: Connection weight between jth neuron in hidden layer and kth neuron in output layer.

The outputs of neurons in hidden layer (z) and output layer (y) are achieved by (1) and (2), respectively.

$$z_j = f(\sum_i x_i w_{ij}) \tag{1}$$

$$y_k = g(\sum_j z_j u_{jk}) \tag{2}$$

f and g are two linear or nonlinear functions.

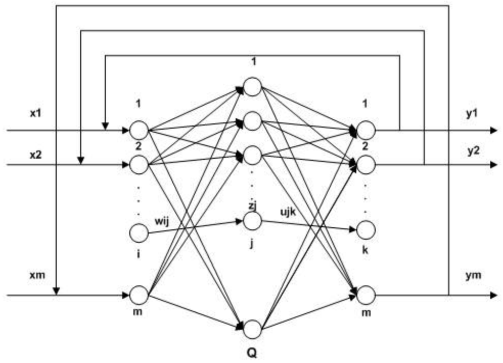

Fig. 1. The schematic of neuro-optimizer.

In this paper, we consider minimization problems and introduce the neuro-optimizer as a minimizer; one can change a maximization problem to minimization one: just slap a minus sign on the front of the objective function to change a maximization problem to a minimization one.

For a minimization problem, the proposed neuro-optimizer works as follows:

First, initial conditions are set and weight matrices are initialized with random values. Then a random solution is selected in search space. This solution is assigned to the best solution and its objective value is calculated and assigned to the best objective value. This solution is fed to NO, as input, and the output of NO is obtained by using (1) and (2). Then the objective value of the output solution is computed. If the new objective value is less than the best one, the best objective value and the best solution are updated (replaced by new ones), otherwise no updating (replacement) occurs. Thereafter, the weights are updated by

$$w_{ij}(n+1)=\eta_1\times w_{ij}(n)+\beta_1\times(OBJ_{GLOBAL}-OBJ(n))\times w_{ij}(n)+\gamma_1\times(OBJ(n-1)-OBJ(n))\times w_{ij}(n)+\alpha_1\times(w_{ij}(n)-w_{ij}(n-1)) \quad (3)$$

and

$$u_{ij}(n+1)=\eta_2\times u_{ij}(n)+\beta_2\times(OBJ_{GLOBAL}-OBJ(n))\times u_{ij}(n)+\gamma_2\times(OBJ(n-1)-OBJ(n))\times u_{ij}(n)+\alpha_2\times(u_{ij}(n)-u_{ij}(n-1)) \quad (4)$$

where n is the iteration number, $\eta_i\in[0,1]$, i=1,2 is the inertia coefficient, $\beta_i\in[0,\beta_{\max}]$ and $\gamma_i\in[0,\gamma_{\max}]$, i=1,2 are global and local learning factors, respectively ($\beta_{\max}$ and $\gamma_{\max}$ are problem dependent constants), $\alpha_i\in[0,1]$, i=1,2 is the momentum coefficient and OBJ(n) and $\text{OBJ}_{\text{GLOBAL}}$ are the current objective function value and the best objective value calculated so far, respectively.

Then the current solution, the output of the previous input, is fed back to NO, as the new input, to generate the next solution. In fact, in each iteration, current solution is fed to NO to produce an output as the next solution and after updating the weights, the best solution and corresponding objective value, this new solution is fed back to NO as the current input to generate the next output as the next solution. Therefore, in NO procedure just one function evaluation is assessed in each iteration and there is no mapping onto the network and any continuous problem can be optimized using NO. In other words, NO just acts as a solution generator. This process continues until one of the stop conditions is satisfied. When the process is stopped, the saved best solution is the optimum of the problem.

The pseudo-code of the neuro-optimizer procedure is as follows:

```
Initialization.
Generate a random solution in search space and save it as the best solution.
Calculate objective value of the solution and save it as the best objective value.
While one of the stop conditions is not true
Repeat
Insert the presented solution to NO as the input and obtain the output as the new solution.
Compute the new solution objective value:
if it is better than the best objective value then replace new solution and its objective value as the best solution and the best objective value, respectively, otherwise do no replacements.
Update the weights.
End of while (if at least one of the stop conditions is true).
Export the best solution as the optimum of the problem.
End.
```

Remark 1. The number of hidden layer's neurons (*Q*) depends on the complexity of the problem, objective value and search space (search space dimension). In general, large *Q* causes the algorithm to work slowly, and less *Q* causes to fail in local minima. A suitable value for *Q*, is (By trial and error) $2<Q<5m$, where m is the search space's dimension.

Remark 2. The results of several simulations have shown that when the number of hidden layers exceeds of two, the convergence rate goes slower and no significant improvement is occurred. So, most of time it is suitable to select the number of hidden layers one, or at most two. It causes to a simple, fast and easily implemented NO.

Remark 3. Bias terms cannot be added in input layer, because the number of neurons in input layer is fixed and is equal to search space dimension. However we can use the bias terms to hidden layer(s) that can help the diversification and escaping from local minima.

Remark 4. Common linear and nonlinear functions (such as pure linear, unipolar sigmoid and bipolar sigmoid functions) can be considered as the output function of neurons in hidden and output layers (f and g in (1) and (2)), with the following considerations:

a) If the variables are not constrained, then when the output function is pure linear we don't need to renormalize the outputs of neurons to the variables range, but if we use nonlinear functions then we have to renormalize the outputs of functions to the suitable range, which is a time wasting process.

b) If linear functions are employed for the outputs of layers, the outputs of the output layer are only a linear combination of the inputs. So, the hidden layers may not help to diversification. But, if nonlinear functions are employed, a nonlinear combination of input and hidden layers make the output that the hidden layers will help to diversification and escaping from local minima trap.

Although, nonlinear functions such as sigmoid functions have better diversification characteristics than linear functions but linear functions such as pure linear function don't need to renormalize the outputs and has less time consumption. Therefore, there is a tradeoff between diversification and time consumption. So, selecting the output functions is a problem dependant issue and can be selected by trial and error.

Remark 5. The major difference between artificial neural networks and neuro-optimizer is in the procedure of weight adjusting. Neuro-optimizer adjusts its weights using a new heuristic rule (Eqs. (3) and (4)). This means that neuro-optimizer has not train or test phase. There is also no mapping problem in NO.

Remark 6. The terms $\beta_i \times (OBJ_{GLOBAL} - OBJ(n))$ and $\gamma_i \times (OBJ_{GLOBAL} - OBJ(n))$, i=1, 2, are named global and local adaptive coefficients, respectively. In each iteration, the former term defines the weights changing, proportional to movement towards the global solution found so far, and the later term defines weights altering, proportional to relative improvement of presented solution with respect to the previous solution, adaptively. In other words, the adaptive coefficients decrease or increase the weights size relative to being close or far from the optimum point, respectively. The terms $\alpha_1 \times (w_{ij}(n) - w_{ij}(n-1))$ and $\alpha_2 \times (u_{ij}(n) - u_{ij}(n-1))$ are momentum statement and are known as a tool for escaping from local minima. Upper and lower limits are also considered for weight matrices to prevent the weights saturation as follows: $w_{min}<w<w_{max}$ and $u_{min}<u<u_{max}$.

Remark 7. The stop condition can be as follows:

a) When no improvement has been made for a certain number of iterations.

b) The maximum number of iteration has been reached.

Of course, other stop conditions can be considered depending on the problem.

Remark 8. For diversification in search and escaping from local minima as well as to speed up the convergence, the following contraptions can be applied.

a) Modifying the number of hidden layers and the number of their neurons.

b) Using bias terms in hidden layer(s).

c) Using momentum in weight updating.

d) Selecting different functions in the outputs of neurons.

e) Restarting the algorithm from other (better) starter solutions.

f) Adapting other values for parameters: $\alpha, \beta, \gamma, \eta$, Q, w_{max}, and u_{max}.

3. GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION

3.1. Genetic algorithm

Genetic Algorithm [24] is the most famous population based method and has been applied to a large number of different types of problems. The idea stems from attempting to copy the way in which nature has evolved and selected the fittest individuals for reproduction, whilst occasionally mutating the chromosomes or genes of these individuals.

The algorithm starts with creating an initial population of solutions, and then creating a new generation, by means of probabilistically selecting parents and individuals (this may be by means of a kind of roulette wheel mechanism which biases the selection towards fitter individuals) to perform crossover, mutation and reproduction until the new population has reached the predefined population size. This process then continues until some termination condition is reached.

3.2. Particle swarm optimization

A particle swarm optimizer [25] is a population based stochastic optimization algorithm modeled after the simulation of the social behavior of bird flocks. In a PSO system a swarm of individuals (called particles) fly through the search space. Each particle represents a candidate solution to the optimization problem. The position of a particle is influenced by the best position visited by itself and the position of the best particle in its neighborhood. When the neighborhood of a particle is the entire swarm, the best position in the neighborhood is referred to as the global best particle.

The global optimizing model proposed by Shi and Eberhart [25] is as follows:

$$v_{i+1} = w \times v_i + c_1 \times RAND \times (P_{best} - x_i) + c_2 \times rand \times (G_{best} - x_i) \quad (5)$$

$$x_{i+1} = x_i + v_{i+1} \quad (6)$$

where v_i is the velocity of particle i, x_i is the particle position, c_1 and c_2 are the positive constant parameters, RAND and rand are random functions in the range [0,1], P_{best} is the best position of the ith particle, G_{best} is the best position among all particles in the swarm and w is the inertial weight [25].

4. TEST RESULTS

The efficiency of NO was tested using a set of benchmark functions. To avoid any misinterpretation of the optimization results, related to the choice of any particular initial parameters, we performed each test 100 times, starting from various randomly selected solutions, inside the hyper rectangular search domain specified in the usual litterateur.

The results of NO tests performed on 11 functions listed in Appendix 1 are shown in Table 1. To evaluate the efficiency of the proposed NO algorithm, we retained the following criteria summarizing results from 100 minimizations per function: the rate of successful minimizations (RATESM), the average of the objective function evaluation numbers (AVERAGEOBJEN) and the average error (AVERAGEERROR). These criteria are defined precisely below.

When at least one of the termination tests is verified, NO stops and provides the coordinates of a located solution, and the objective function value $OBJ_{N.O}$ at this solution. We compared this result with the known analytical minimum OBJ_{ANAL}; we considered this result to be successful if the following inequality held:

$$|OBJ_{N.O} - OBJ_{ANAL}| < \varepsilon_{rel}|OBJ_{INIT}| + \varepsilon_{abs} \qquad (7)$$

where $\varepsilon_{rel} = 0.01$, $\varepsilon_{abs} = 0.0001$ and OBJ_{INIT} is an empirical average of the objective function value, calculated over typically 100 solutions, randomly selected inside the search domain, before running the algorithm. The average of the objective function evaluation numbers is evaluated in relation to only the successful minimizations and it shows the convergence rate of the algorithm. In fact, this criterion measures the speed of the algorithm and shows whether it is fast or slow. The average error is defined as the average of OBJ gaps between the best successful solution found and the known global optimum. This criterion shows the accuracy of the algorithm in finding the global optimum. As Table 1 shows, when the search space is more complicated the rate of successful minimization is decreased. Hence, NO can escape from local minima trap because of its stochastic and intelligent nature. For all functions, the average of the objective function evaluation numbers does not exceed 1000 with a suitable accuracy. This shows that the convergence of the NO is fast. For all functions, average of OBJ gaps between the best successful solution found and the known global optimum is less than 0.1. This accuracy is acceptable for many real world optimization problems.

Table 1. Results of NO for 15 benchmark functions.

Benchmark function	$RATE_{SM}$ (%)	$AVERAGE_{OBJEN}$	$AVERAGE_{ERROR}$
RC	96	285	0.01
ES	96	446	0.04
GP	96	290	0.015
B2	96	185	0.015
SH	94	318	0.01
R_2	95	347	0.03
Z_2	95	205	0.035
DJ	95	272	0.02
$H_{3,4}$	83	361	0.05
$S_{4,5}$	81	467	0.03
$S_{4,7}$	80	445	0.01
$S_{4,10}$	78	431	0.04
R_5	80	688	0.05
Z_5	82	651	0.055
$H_{6,4}$	84	664	0.065

The performance of NO is then compared to continuous GA and PSO algorithms. The experimental results obtained for the test functions, using the 3 different methods, are given in Table 2. In our simulations, each population in GA has 20 chromosomes and a swarm in PSO has 20 particles. Other parameters of 3 algorithms are selected optimally, by trial and error. For each function, we give the average number of function evaluations for 100 runs. The best solution found by 3 methods was similar, so there were not given in Table 2. We notice that results from NO are better than results from GA and PSO methods (NO is faster than GA and PSO). This is because that NO is not a population based algorithm and it evaluates just one objective function in each iteration, while GA and PSO (and any other population based algorithms) evaluate a population of objective functions.

Table 2. Average number of objective function evaluations used by three methods.

Function/Method	NO	GA	PSO
RC	285	486	426
ES	446	920	903
GP	290	410	395
B2	185	325	334
SH	318	576	615
R_2	347	657	633
Z_2	205	620	622
DJ	272	601	556
$H_{3,4}$	361	712	681
$S_{4,5}$	467	915	822
$S_{4,7}$	445	766	741
$S_{4,10}$	431	792	816
R_5	688	2516	2441
Z_5	651	1712	1945
$H_{6,4}$	664	1956	2122

Larger population size causes to more function evaluation numbers. So, convergence rate of GA and PSO is population size dependant while NO is not related to population.

5. APPLICATION OF NO FOR ROBOT OPTIMAL CONTROLLER DESIGN

In this section, an optimal PID controller is designed for a five-bar-linkage manipulator robot.

5.1. Problem Formulation

The PID controller is used to improve the dynamic response and to reduce the steady-state error. The transfer function of a PID controller is described as:

$$G(s)= Kp + K_I / s + K_D s \tag{8}$$

where KP, KI and KD are the proportional gain, integral and derivative time constants, respectively. For designing an optimal PID controller, a suitable objective function that represents system requirements, must be defined in the first step. A set of good control parameters KP, KI and KD can produce a good step response that will resultant in minimization of performance criteria. The optimal PID controller parameters that minimize the performance indexes are designed using the proposed NO algorithm. In the design of a PID controller, the performance criterion or objective function is first defined based on some desired specifications and constraints under input testing signal. Some typical output specifications in the time domain are overshoot, rise time, settling time, and steady-state error. In general, three kinds of performance criteria, the integrated absolute error (IAE), the integral of squared-error (ISE), and the integrated of time-weighted-squared-error (ITSE) are usually considered in the control design under step testing input, because they can be evaluated analytically in the frequency domain. It is worthy to notice that using different performance indices probably makes different solutions for PID controllers. The three integral performance criteria in the frequency domain have their own advantages and disadvantages. For example, a disadvantage of the IAE and ISE criteria is that their minimization can result in a response with relatively small overshoot but a long settling time. Although the ITSE performance criterion can overcome the disadvantage of the ISE criterion, the derivation processes of the analytical formula are complex and time-consuming [26]. The IAE, ISE, and ITSE performance criteria formulas are as follows:

$$IAE=\int_0^\infty |r(t)-y(t)|\ dt=\int_0^\infty |e(t)|\ dt \tag{9}$$

$$ISE=\int_0^\infty e^2(t)\ dt \tag{10}$$

$$ISTE=\int_0^\infty te^2(t)\ dt \tag{11}$$

In this paper, another time domain performance criterion defined by

$$\min_K W(K) = (1/(1+e^{-\alpha})) \times (T_s + T_r) + (e^{-\alpha}/(1+e^{-\alpha})).(M_p + E_{ss}) \quad (12)$$

Where, K is [KP, KI, KD], and $\alpha \in$ [-5, 5] is the weighting factor. The optimum selection of α depends on designer's requirements and the characteristics of the plant under control. We can set α to be smaller than 0 to reduce the overshoot and steady-state error. On the other hand, we can set α to be larger than 0 to reduce the rise time and settling time. Note that, if α is set to 0, then all performance criteria (i.e. overshoot, rise time, settling time, and steady-state error) will have the same worth.

For designing an optimal PID controller, determination of vector K with regards to the minimization of performance index is the main issue. Here, the minimization process is performed using the proposed NO algorithm. For this purpose, step response of the plant is used to compute four performance criteria overshoot (Mp), steady-state error (Ess), rise time (Tr) and setting time (Ts) in the time domain. At first, the lower and upper bounds of the controller parameters should be specified. Then the NO method is applied to find the optimal solutions.

Here, to show the efficiency and desirable performance of the proposed algorithm in designing optimal PID controllers, a well known Mechatronics application, i.e., a robot is considered. The examined robot configuration is a five-bar-linkage. Dynamic equations of the robot are described in the following subsection. Afterwards, NO algorithm for an optimum PID controller is utilized.

5.2. Dynamic equations of five-bar-linkage manipulator robot

In recent years, there has been a growing interest in the design and control of lightweight robots. Several researchers have studied the modeling and control of a single link flexible beam [34]. Fig. 2 shows the 5 bar linkage manipulator built in our robotics research lab. Also, Fig. 3 depicts the five-bar linkage manipulator schematic where the links form a parallelogram. Let qi, Ti and Ihi be the joint variable, torque and hub inertia of the ith motor, respectively. Also, let Ii, li, dCi and mi be the inertia matrix, length, distance to the centre of gravity and mass of the ith link, correspondingly.

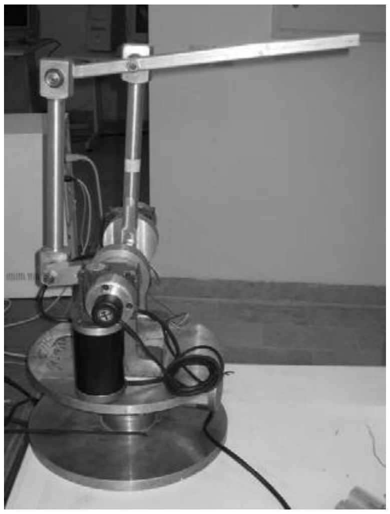

Fig. 2. Planar presentation of robot.

The dynamic equations of the manipulator are [35]:

$$T_1 = (M_{11} + I_h^1)\ddot{q}_1 + M_{12}\ddot{q}_2 + \frac{\partial M_{12}}{\partial q_2}\dot{q}_2^2 + g(m_1 d_{c1} + m_3 d_{c3} + m_4 l_1)\cos q_1$$

$$T_2 = (M_{22} + I_h^2)\ddot{q}_2 + M_{21}\ddot{q}_1 + \frac{\partial M_{21}}{\partial q_1}\dot{q}_2^2 + g(m_1 d_{c2} + m_3 l_2 + m_4 d_{c4})\cos q_2 \quad (13)$$

Where, g is the gravitational constant and

$M_{11} = I_{11}^1 + I_{11}^3 + m_1 d_{c1}^2 + m_3 d_{c3}^2 + m_4 I_1^2$

$M_{22} = I_{11}^2 + I_{11}^4 + m_2 d_{c2}^2 + m_3 l_2^2 + m_4 d_{c4}^2$

$M_{12}=M_{21} = (m_3 dc_3 l_2 - m_4 dc_4 l_1)\cos(q_1 - q_2)$ (14)

It's noticed from (13)-(14) that for

$$m^3 lc^3 l_2 = m^4 lc^4 l_1 \qquad (15)$$

We have M_{12} and M_{21} equal to zero, that is, the matrix of inertia is diagonal and constant. Hence the dynamic equations of this manipulator will be

$$T_1 = (M_{11} + I_h^1)\ddot{q}_1 + g\,(m_1 lc_1 + m_3 lc_3 + m_4 l_1)\cos q1,$$

$$T_2 = (M_{22} + I_h^2)\ddot{q}_2 + g(m_1 lc_2 + m_3 l_2 + m_4 lc_4)\cos q2 \qquad (16)$$

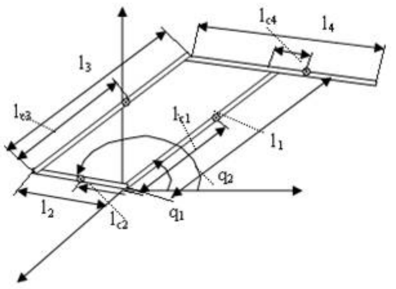

Fig. 3. Planar presentation of the robot.

Notice that T1 depends only on q1 but not on q2. On the other hand T2 depends only on q2 but not on q1. This discussion helps to explain the popularity of the parallelogram configuration in industrial robots. If the condition (15) is satisfied, then we can adjust the two rotations independently, without worrying about interactions between them.

5.3. Simulation Results

Having 2 motors, the manipulator specification consisting of mass, length and centre of gravity of links are given in Table 3. The main purpose is designing an optimal PID controller for each of motors to control their rotations, with good performance. Using equation (16), five-bar-linkage manipulator robot is easily simulated using Matlab and Simulink. The block diagram of the five-bar-linkage manipulator robot with PID controller for motor 1 is shown in Fig. 4. The block diagram for motor 2 is similar to this figure. The maximum iteration of all experiments is considered equal to 200. Also, α is set to 0 for all performance criteria to have the same merit in the objective function.

The following process is done to determine the optimal values of the PID controller parameters (i.e., vector K). First, the lower and upper bounds of the three controller parameters are selected as 0 and 30, respectively. Then, the network is initialized, randomly. Each solution K (the controller parameters) is sent to Matlab® Simulink® block and the values of four performance criteria in the time domain, i.e., Mp, Ess, Tr and Ts are calculated iteratively. Afterwards, the objective function is evaluated for each solution according to these performance criteria.

Table 3. Five-bar-linkage manipulator data.

Link	Mass (Kg)	Length (m)	C of G (m)
1	0.288	0.33	0.166
2	0.0324	0.12	0.06
3	0.3702	0.33	0.166
4	0.2981	0.45	0.075

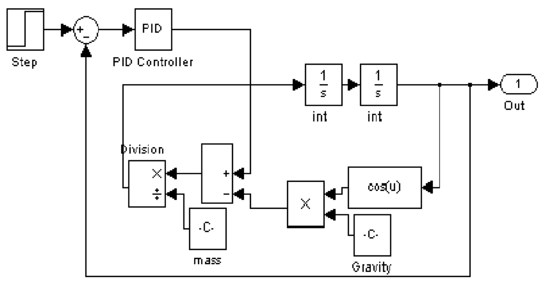

Fig. 4. Block diagram of the motor with PID controller.

Then, the procedure of NO algorithm is performed, as illustrated the pseudo-code in Section 2. At the end of any iteration, the program checks the stop criterion. When one termination condition is satisfied, theprogram stops and the latest global best solution is the best solution of K.

Fig. 5 illustrates the step response without PID controllers for two motors. Figures 6 and 7 show the step response of rotation for motors 1 and 2, respectively.

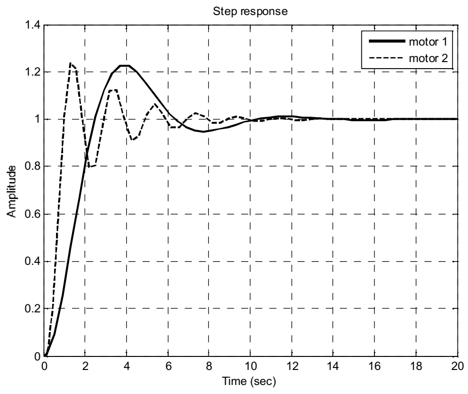

Fig. 5. Step response of the robot motors without PID controller.

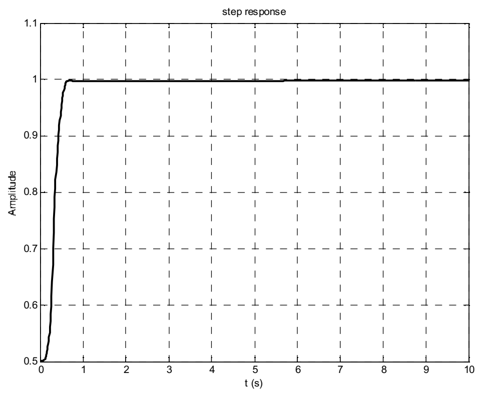

Fig. 6. Step response of the motor #1 rotation using NO method.

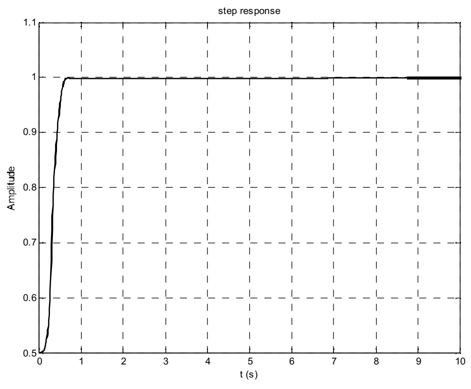

Fig. 7. Step response of the motor #2 angel using NO method.

The simulation results of the best solution are summarized in Table 4. These results demonstrate that cost function is converged rapidly. In conclusion, NO algorithm has rapid convergence characteristic and is highly effective in solving the optimal tuning problem of PID controller parameters.

Table 4. Summary of simulation results of five-bar-robot motors.

motor	algorithm	P	I	D	M_p	T_s	T_r	E_{ss}	cost	iteration
1	NO	29	1.6	2.4	0	0.141	0.12	0	0.1455	34
2	NO	31	1.5	2.5	0	0.193	0.115	0	0.1611	30

5. CONCLUSIONS

In this paper, a new optimization technique based on neural networks has been introduced and it called neuro-optimizer (NO). Implemented by a recurrent neural network, NO uses a heuristic new intelligent rule to update its weights, with no supervision. The new optimization algorithm (NO) was proposed to solve the optimization problems numerically. Different benchmarks were used to illustrate the mentioned advantages. Dealing with this problem, a new time domain performance criterion for PID controller design was proposed. In all case studies, NO performed better than GA and PSO approaches which exposed NO as a promising optimization method. The optimal controller design of the five-bar-linkage manipulator robot has been considered, as a practical application. The proposed method was implemented for tuning the controller for the robot. High promising results demonstrate that the proposed algorithm is robust, efficient and can obtain higher quality solution with better computational efficiency.

APPENDIX A

Some well-known benchmark functions of optimization problems:

Branin RCOS (RC) (2 variables):

$$RC(x,y)=(y-(5/13)x^2+(5/3.14)x-6)+10(1-(1/28))\cos(x)+10$$

B2 (2 variables):

$$B2(x,y)=x+2y^2-0.3\cos(10x)-0.4\cos(13y)+0.7$$

Easom (ES) (2 variables):

$$ES(x,y)=-\cos(x)\cos(y)\exp(-((x-3.14)^2+(y-3.14)^2)$$

Goldstein and Price (GP) (2 variables):

$$GP(x,y)=[1+(x+y+1)^2(19-14x+3x^2+6xy+3y^2)]\times[30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)]$$

Shubert (SH) (2 variables):

$$SH(x,y)=\left\{\sum_{j=1}^{5} j\cos[(j+1)x+j]\right\}\times\left\{\sum_{j=1}^{5} j\cos[(j+1)y+j]\right\}$$

De Joung (DJ) (3 variables):

$$DJ(x,y,z)=x^2+y^2+z^2$$

Hartmann (H3,4) (3 variables):

$$H_{3,4}(x_j)=-\sum_{j=1}^{4} c_i \exp[-\sum_{j=1}^{4} a_{ij}(x_j-p_{ij})^2]$$

a_{ij}			c_i	p_{ij}		
3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
0.1	10.0	35.0	3.2	0.0381	0.5743	0.8828

Shekel (S4,n) (3 variables):

$$H_{4,n}(X)=-\sum_{j=1}^{n}[(X-a_i)^T(X-a_i)+c_i]^{-1}$$

3 functions were considered: S4,5, S4,7 and S4,10;

i	a_i				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.0	7.0	3.6	0.5

Hartmann (H6,4) (6 variables):

$$H_{6,4}(x_j)=-\sum_{j=1}^{4} c_i \exp[-\sum_{j=1}^{6} a_{ij}(x_j-p_{ij})^2]$$

i	c_i	a_{ij}					
1	1.0	10.0	3.00	17.0	3.5	1.70	8.00
2	1.2	0.05	10.0	17.0	0.1	8.00	14.0
3	3.0	3.00	3.50	1.70	10	17.0	8.00
4	3.2	17.0	8.00	0.05	10	0.10	14.0

p_{ij}					
0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
0.2348	0.1451	0.35522	0.2883	0.3047	0.6650
0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Rosenbrock (Rn) (n variables):

$$R_n(X)=-\sum_{j=1}^{n}[100\sum_{j=1}^{n}(x_j^2-x_{j+1})^2+(x_j^2-1)^2]$$

Zakharov (Zn) (n variables):

$$Z_n(X) = \sum_{j=1}^{n} x_j^2 + \left(\sum_{j=1}^{n} 0.5 j x_j^2\right)^2 + \left(\sum_{j=1}^{n} 0.5 j x_j^2\right)^4$$

ACKNOWLEDGMENT

This work is financially supported by Young Researchers Club of Islamic Azad University of Ahar Branch.

REFERENCES

[1] J. Hopfield, and D. Tank, Neural computation of decisions in optimization problems, Biological Cybernetics, Vol. 52, 1985, pp. 141-152.

[2] G.E. Hinton, and T.J. Sejnowsky, Optimal perceptual inference, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washigton, 1983, pp. 448-453.

[3] D. Amit, H. Gutfreund, and H. Sompolinsky, Spin-Glass models of neural networks, Physical Review Letters A 32, 1985, pp. 1007-1018.

[4] Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso, Combinatorial optimization with gaussian machines, Proceedings IEEE International Joint Conference on Neural Networks 1, 1989, pp. 533–540.

[5] T. Kohonen, Self-Organized formation of topologically correct feature maps, Biological Cybernetics 43, 1982, pp. 59–69.

[6] A.H. Gee, and R. W. Prager, Limitations of neural networks for solving traveling salesman problems, IEEE Trans. Neural Networks, vol. 6, 1995, pp. 280–282.

[7] M. Goldstein, Self-Organizing feature maps for the multiple traveling salesman problem (MTSP), Proceedings IEEE International Conference on Neural Networks, Paris, 1990, pp. 258–261.

[8] Y. P. S. Foo, and Y. Takefuji, Stochastic neural networks for job-shop scheduling: parts 1 and 2, Proceedings of the IEEE International Conference on Neural Networks 2, 1988, pp. 275–290.

[9] Y.P. S. Foo, and Y. Takefuji, Integer Linear programming neural networks for job shop scheduling, Proceedings of the IEEE International Conference on Neural Networks 2, 1988, pp. 341–348.

[10] J.S. Lai, S.Y. Kuo, and I.Y. Chen, Neural networks for optimization problems in graph theory, Proceedings IEEE International Symposium on Circuits and Systems 6, 1994, pp. 269–272.

[11] D.E. Van Den Bout, and T.K. Miller, Graph partitioning using annealed neural networks, IEEE Transactions on Neural Networks 1, 1990, pp. 192–203.

[12] S. Vaithyanathan, H. Ogmen, and J. IGNIZIO, Generalized boltzmann machines for multidimensional knapsack problems, Intelligent Engineering Systems Through Artificial Neural Networks 4, ASME Press, New York, 1994, pp. 1079–1084.

[13] A. Yamamoto, M. Ohta, H. Ueda, A. Ogihara, and K. Fukunaga, Asymmetric neural network and its application to knapsack problem, IEICE Transactions Fundamentals E78-A, 1995, pp. 300–305.

[14] K. Urahama, and H. Nishiyuki, Neural algorithms for placement problems, Proceedings International Joint Conference on Neural Networks 3, Nagoya, 1993, pp. 2421–2424.

[15] K.E. Nygard, P. Jueli, and N. Kadaba, Neural networks for selecting vehicle routing heuristics, ORSA Journal of Computing 2, 1990, pp. 353–364.

[16] A.I. Vakhutinsky, and B. L. Golden, Solving vehicle routing problems using elastic nets, Proceedings IEEE International Conference on Neural Networks 7, 1994, pp. 4535–4540.

[17] L. Fang, W. H. Wilson, and T. Li, Mean-Field annealing neural net for quadratic assignment, Proceedings International Conference on Neural Networks, Paris, 1990, pp. 282–286.

[18] G.A. Tagliarini, and E. W. Page, Solving constraint satisfaction problems with neural networks, Proceedings IEEE International Conference on Neural Networks 3, 1987, pp. 741–747.

[19] M. Kajiura, Y. Akiyama, and Y. Anzai, Solving large scale puzzles with neural networks, Proceedings Tools for AI Conference, Fairfax, 1990, pp. 562–569.

[20] N. Funabiki and Y. Takefuji, A neural network parallel algorithm for channel assignment problems in cellular radio networks, IEEE Trans. Veh. Technol., vol. 41, Nov. 1992, pp. 430–437.

[21] K. Smith, and M. Palaniswami, Static and

dynamic channel assignment using neural networks, IEEE Journal on Selected Areas in Communications 15, 1997, pp. 238–249.

[22] T. Bultan and C. Aykanat, Circuit partitioning using parallel mean field annealing algorithms, Proceedings 3rd IEEE Symposium on Parallel and Distributed Processing, 1991, pp. 534–541.

[23] U. Halici, Artificial neural networks, EE 543 Lecture Notes, Middle East Technical University, Ankara, Turkey, 2004.

[24] J.H. Holland, Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, MI, Internal Report, 1975.

[25] Y. Shi, and R. Eberhart, A modified particle swarm optimizer, Proceedings of the IEEE international conference on evolutionary computation, Piscataway, NJ: IEEE Press; 1998, pp. 69–73.

[26] J.G. Ziegler and N.B. Nichols, “Optimum settlings for automatic controllers,” Trans. On ASME., vol. 64, pp. 759-768, 1942.

[27] Z.L. Gaing, “A Particle Swarm Optimization Approach for Optimum Design of PID controller in AVR system,” IEEE Transactions on Energy Conversion, vol. 9, no. 2, pp. 384-391, 2003.

[28] Z.Y. Zhao, M. Tomizuka, and S. Isaka, “Fuzzy gain scheduling of PID controllers,” IEEE Trans. System, Man, and Cybernetics, vol. 23, no. 5, pp. 1392-1398, 1993.

[29] S.Y. Chu, C.C. Teng, “Tuning of PID controllers based on gain and phase margin specifications using fuzzy neural network,” Fuzzy Sets and Systems, vol. 101, no. 1, pp. 21-30, 1999.

[30] G. Zhou and J.D. Birdwell, “Fuzzy logic-based PID autotuner design using simulated annealing,” Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, pp. 67 – 72, 1994.

[31] R.A. Krohling and J.P. Rey, “Design of optimal disturbance rejection PID controllers using genetic algorithm,” IEEE Trans. Evol. Comput., vol. 5, pp. 78–82, 2001.

[32] D.H. Kim, “Tuning of a PID controller using a artificial immune network model and local fuzzy set,” Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference, vol. 5, pp. 2698 – 2703, 2001.

[33] Y.T. Hsiao, C.L. Chuang, and C.C. Chien, “Ant colony optimization for designing of PID controllers,” Proceedings of the 2004 IEEE Conference on Control Applications/ International Symposium on Intelligent Control/International Symposium on Computer Aided Control Systems Design, Taipei, Taiwan, 2004.

[34] D. Wang and M. Vidyasagar, “Modeling of a five-bar-Linkage Manipulator with One Flexible Link,” in Proc. IEEE Int. Symp, subject, Turkey, pp. 21–26, 1988.

[35] D. Wang, J.P. Huissoon and K. Luscott, “A teaching robot for demonstrating robot control strategies,” manufacturing research corporation of Ontario, 1993.