Neurocomputing 272 (2018) 10–16

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Artificial neural networks used in optimization problems

Gabriel Villarrubia, Juan F. De Paz*, Pablo Chamoso, Fernando De la Prieta

University of Salamanca, BISITE Research Group, Edificio I+D+I, 37007, Salamanca, Spain

ARTICLE INFO

Article history:
Received 28 October 2016
Revised 11 March 2017
Accepted 3 April 2017
Available online 23 June 2017

Keywords:
Neural networks
Optimization problems
Non-linear optimization

ABSTRACT

Optimization problems often require the use of optimization methods that permit the minimization or maximization of certain objective functions. Occasionally, the problems that must be optimized are not linear or polynomial; they cannot be precisely resolved, and they must be approximated. In these cases, it is necessary to apply heuristics, which are able to resolve these kinds of problems. Some algorithms linearize the restrictions and objective functions at a specific point of the space by applying derivatives and partial derivatives for some cases, while in other cases evolutionary algorithms are used to approximate the solution. This work proposes the use of artificial neural networks to approximate the objective function in optimization problems to make it possible to apply other techniques to resolve the problem. The objective function is approximated by a non-linear regression that can be used to resolve an optimization problem. The derivate of the new objective function should be polynomial so that the solution of the optimization problem can be calculated.

1. Introduction

Optimization problems are an important part of soft computing, and have been applied to different fields such as smart grids [1], logistics [2,3], resources [4] or sensor networks [5]. Such problems are characterized by the presence of one or more objective maximizing or minimizing functions [5] and various restrictions that must be met so that the solution is valid. The problems are easy to resolve when we are working with linear restrictions and objective functions because there are methods to obtain the optimal solution. However, in the case of non-linear restrictions or objective functions, it may be necessary to use heuristics [2,5] to obtain a pseudo-optimal solution. The management of heuristic solutions is continually evolving, which is precisely why we are looking for alternatives to problems in which it is not feasible to find an optimal solution. When working with linear restrictions and objective functions, optimization problems can be resolved with algorithms such as the Simplex [6], which limits the study of this type of problem. Certain non-linear problems can be optimally resolved by using algorithms such as Lagrange multipliers or Kuhn–Tucker conditions [7]. In many cases, it is not possible to resolve a problem with Lagrange multipliers because the generated system of equations cannot be resolved without resorting to numerical methods, which would prevent a direct approach to resolving the problem. In other cases, the Kuhn–Tucker conditions are not met. There is a broad range of opportunities to study optimization problems that cannot be solved with an exact algorithm. These problems are usually solved by applying a heuristics and metaheuristics solution such as genetic algorithms [8], particle swarm optimization [9], Simulated annealing [10], ant colony optimization [11] etc.

This work proposes the use of neural networks such as heuristics to resolve optimization problems in those cases where the use of linear programming or Lagrange multipliers is not feasible. To resolve these problems a multilayer perceptron is applied to approximate the objective functions; the same process could be followed in the restrictions. The proposal establishes the activation function to be used and the criteria to conduct the training using a dataset according to the defined domain of the variables. This process makes it possible to transform objective functions into other functions, which can then be applied to resolve optimization problems that can be resolved without metaheuristics. The objective function is approximated with a non-linear regression with the objective to obtain a new function that facilitates the solution of the optimization problem. The activation function of the neural network must be selected so that the derivate of the transformed objective functions should be polynomial. Once the new objective functions has been calculated the problem can be resolved with other techniques. The same process can be applied to non-equality restrictions, but it is necessary to introduce gaps to satisfy the restrictions.

This paper is organized as follows: Section 2 revises related works, Section 3 describes the proposal, and finally Section 4 shows the results and conclusions obtained.

* Corresponding author.
E-mail addresses: gvg@usal.es (G. Villarrubia), fcofds@usal.es, fcofds@gmail.com (J.F. De Paz), chamoso@usal.es (P. Chamoso), fer@usal.es (F.D. la Prieta).

http://dx.doi.org/10.1016/j.neucom.2017.04.075
0925-2312/

2. Heuristics applied to optimization

On certain occasions, optimization problems cannot be solved by applying methods such as Simplex or Lagrange. Methods such as Simplex are applicable only when problems are linear, so the algorithm cannot be properly applied when the objective function or constraints are nonlinear. Lagrange makes it possible to resolve optimization problems even when problems are not linear, but it is not always possible to resolve the equations after applying Lagrange. When exact algorithms do not allow obtaining an optimal solution, it is necessary to apply heuristics and metaheuristics algorithms. Some heuristics, such as ant colony optimization, are oriented to resolve optimization problems in graphs [12], although they can be applied in other optimization fields such as control processes [13]. The authors in this study [13] applied fuzzy logic in a nonlinear process to improve the efficiency in the learning process with regard to execution time. Other alternatives such as simulated annealing or PSO (Particle Swarm Optimization) are commonly applied in optimization functions. In general, several evolutional algorithms can be applied to resolve optimization problems, as seen in various studies [9,13].

In mathematics, there are heuristics methods that work with approximation functions. Approximation functions are usually defined around a point, which would make it possible to use polynomials to approximate functions by applying the Taylor theorem. Based on this idea, it would be possible to solve non-linear optimization problems by applying Taylor nonlinear functions. This idea has been applied in algorithms such as Frank-Wolfe [14], which allows linearizing objective functions by applying derivatives in a point to calculate the straight line, plane or hyperplane crosses through that point. The solutions are calculated iteratively with a new hyperplane for each iteration. MAP (Method of Approximation Programming) is a generalization of the Frank-Wolfe algorithm, which permits linearizing the restrictions.

This work proposes carrying out this approximation in a more generic manner, making it possible to solve the problem without needing to calculate a new approximation for each tentative solution. We propose to do so by applying neural networks.

3. Proposal

Komogorov's theorem says that a multilayer perceptron with 3 layers makes it possible to precisely define any continuous function. However, for the approximation to be exact, it is necessary to define an activation function and parameters for which there are no calculation procedures. It is not possible to apply just any activation function, because we must take into account the objective of the functions to simplify the problem.

The proposal to solve an optimization problem is explained in Fig. 1. The system generates a dataset in the domain of the variables to train a neural network. The objective function of the optimization problem is redefined with the multilayer perceptron that transforms the function, making it possible to generate a polynomial equation to resolve the optimization problem. Finally, when the new objective function is calculated another solution can be applied to resolve the problem.

To define a neural network, it is necessary to establish parameters, such as the connections, number of layers, activation functions, propagation rules etc. In the case of the multilayer perceptron, we need to consider its two different stages: the learning stage, and the prediction process. In both stages, the number of layers and activation functions have to be the same. In the prediction stage, other parameters such as the learning rate or the momentum are not relevant. In the case of the multilayer perceptron, the propagation rule is the weighted sum, and it is defined according to (1).

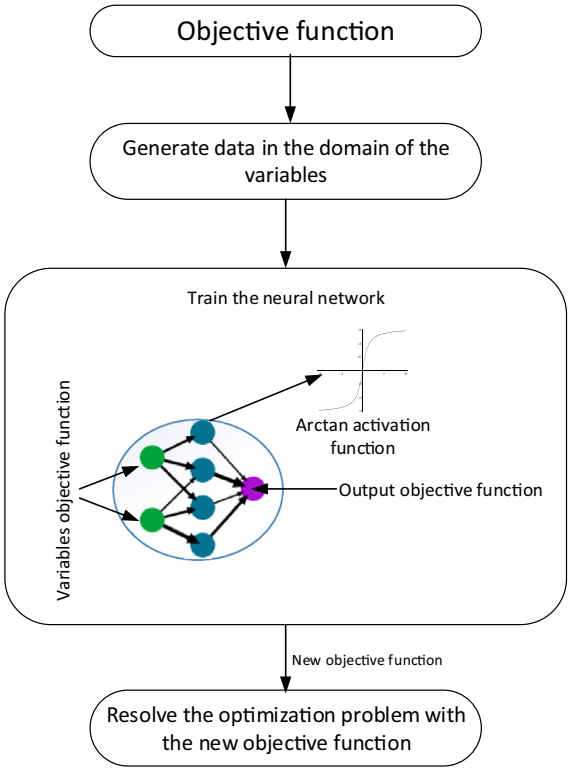

Fig. 1. Workflow optimization problem.

$$\sum_{i=1}^{n} w_{ij} x_i(t) \tag{1}$$

Where w_{ij} is the weight that connects neuron i in the input layer with neuron j in the hidden layer, x_i is the output from neuron i in the input layer, n is the number of neurons in the input layers, and t is the pattern.

In case of having bias in the neuron, the result would be what is shown in (2).

$$\sum_{i=1}^{n} w_{ij} x_i(t) + \theta_j \tag{2}$$

After calculating the propagation rules, we should apply the activation function. If the activation function is linear, we would have an output of neuron j that would be a linear combination of the neurons in the input layer and, consequently, y_j would be a linear function. Therefore, if the activation function is the identity, the net output would correspond to the output of the neuron.

So, if neuron k in the output layer also has the activation function f, the output would be defined as (3).

$$y_j(t) = f\left(\sum_{i=1}^{n} w_{ij} x_i(t) + \theta_j\right) \tag{3}$$

Bearing in mind that the multilayer perceptron has three layers, it is necessary to apply the propagation rule on two occasions in order to transmit the value in the input layer to the neuros in the output layer (4).

$$y_k(t) = \sum_{j=1}^{m} w_{jk} y_i(t) + \theta_k \tag{4}$$

Where k represents neuron k in the output layer, and m is the number of neurons in the hidden layer.