Neural Networks 22 (2009) 1011–1017

Contents lists available at ScienceDirect

Neural Networks

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

Generalized neuron: Feedforward and recurrent architectures

Raghavendra V. Kulkarni, Ganesh K. Venayagamoorthy *

Real-Time Power and Intelligent Systems Laboratory, Department of Electrical and Computer Engineering, Missouri University of Science and Technology, Rolla, MO, USA

ARTICLE INFO

Article history:
Received 21 January 2008
Received in revised form 10 July 2009
Accepted 17 July 2009

Keywords:
Density estimation
Generalized neuron
Nonlinear function approximation
Particle swarm optimization (PSO)
Classification
Recurrent generalized neuron

ABSTRACT

Feedforward neural networks such as multilayer perceptrons (MLP) and recurrent neural networks are widely used for pattern classification, nonlinear function approximation, density estimation and time series prediction. A large number of neurons are usually required to perform these tasks accurately, which makes the MLPs less attractive for computational implementations on resource constrained hardware platforms. This paper highlights the benefits of feedforward and recurrent forms of a compact neural architecture called generalized neuron (GN). This paper demonstrates that GN and recurrent GN (RGN) can perform good classification, nonlinear function approximation, density estimation and chaotic time series prediction. Due to two aggregation functions and two activation functions, GN exhibits resilience to the nonlinearities of complex problems. Particle swarm optimization (PSO) is proposed as the training algorithm for GN and RGN. Due to a small number of trainable parameters, GN and RGN require less memory and computational resources. Thus, these structures are attractive choices for fast implementations on resource constrained hardware platforms.

1. Introduction

Multilayer perceptrons (MLPs) have received significant research attention in recent years due to their ability to approximate any continuous function accurately by means of spatially-local basis functions (Hornik, Stinchombe, & White, 1989; Widrow & Lehr, 1990). Feedforward neural networks have been used in applications such as adaptive control (Liu, Kadirkamanthan, & Billings, 1990), pattern classification (Ou & Murphey, 2007), and forecasting (Atiya, El-shoura, Shaheen, & El-sherif, 1999). Due to internal recurrence, recurrent neural networks (RNNs) possess the capability to dynamically incorporate their past experience. The feedback connection in the RNN topology is used to memorize past information allowing the capability to process the temporal information (Cichocki & Unbehauen, 1993; Wang, 1996).

Function approximation is a critical task in a broad spectrum of applications ranging from system identification to pattern classification. Typical nonlinear function approximation is carried out using polynomials, spline functions, fuzzy logic networks, wavelet networks and neural networks.

Many statistical process control methods rely on the estimation of the probability density function (PDF) of their variables. A large amount of data, which is encountered in most scientific applications, needs to be modeled in a probabilistic manner. Power demand forecasting (Charytoniukand et al., 1999), speech recognition (Pazhayaveetil & Franzon, 2007), medical image registration (Niu, 2005), independent component analysis (Xu, Chen, Cong, Yang, & Shi, 2005) *etc.* are some of the most common applications of PDF estimation.

There has been an immense research interest in the forecasting of real-world time series. These are dynamic in nature, and therefore, time series prediction amounts to dynamic modeling. The structures that have been applied for chaotic time series prediction include MLPs (Lapades & Farber, 1987), radial basis function neural networks (Haykin & Principe, 1998), support vector regression (Mukherjee, Osuna, & Girosi, 1997), support vector machines (Shi & Han, 2007), recurrent neural networks (Cai, Zhang, & Venayagamoorthy, 2007) and nonlinear autoregressive networks (Principe & Kuo, 1995). The Mackey glass chaotic time series is a popular time series used in literature as a benchmark (Mackey & Glass, 1997).

Backpropagation (BP) is the most popular algorithm used for training multilayer feedforward neural networks (Widrow & Lehr, 1990). Several methods have been proposed for enhancing the learning speed of BP and strengthening the generalization capability of layered networks. Apart from multilayer network architectures various simplified architectures and nonlinear activation functions have been used. MLP is not an attractive choice for complex applications due to the storage and computational expense involved in adapting weights for a large number of neurons. A reduced number of trainable weights reduces the memory and computational expense, accelerates the convergence, and reduces the number of training patterns necessary for robust generalization.

* Corresponding author. Tel.: +1 573 341 6641; fax: +1 573 341 4532.
E-mail addresses: arvie@ieee.org (R.V. Kulkarni), gkumar@ieee.org (G.K. Venayagamoorthy).

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.07.027

This paper presents a generalized neuron (GN) structure for classification, nonlinear functional approximation and probability density estimation applications. Further, a recurrent version of GN, called recurrent GN (RGN) is presented for chaotic time series prediction. This study uses a particle swarm optimization (PSO) algorithm for fast and optimal training of GN and RGN (Kennedy & Eberhart, 1995). The compactness of GN and RGN that have smaller numbers of trainable weights than in MLP and RNN, and the convergence speed of PSO over BP as a training algorithm are exploited in this paper. As an MLP training algorithm, PSO has been shown to outperform BP in terms of the speed and the quality of training (Gudise & Venayagamoorthy, 2003). A small number of trainable weights makes GN a compact structure suitable for real time hardware implementation on simple hardware platforms such as microcontrollers. The rest of this paper is organized as follows: Section 2 covers details on the GN and RGN compact structures. Section 3 outlines the PSO algorithm for training the GN and RGN structures. Section 4 discusses the simulation aspects and the results obtained in the four case studies conducted. And finally, concluding remarks are given in Section 5.

2. Feedforward and recurrent generalized neuron architectures

The general structure of a typical neuron contains an aggregation function and an activation function. It is shown in the literature that MLPs are universal approximators of continuous functions for a given set of input–output patterns (Hornik et al., 1989). A typical neuron uses summation or multiplication aggregation functions, and hard-limiter, log sigmoidal, radial basis, or linear activation functions.

The crisp aggregation operators used in the neurons generally overlook the fact that most of the processing in neural networks is done with incomplete information. The GN presented in this paper uses both summation and multiplication aggregation functions, and both sigmoid and Gaussian activation functions. Therefore, the GN has flexibility and resilience to the nonlinearities of real world problems. The compact structure of a GN and RGN (GN with feedback connection) is shown in Fig. 1.

A GN uses both Σ (sum) and Π (product) aggregation functions. The weighted vector of inputs $\boldsymbol{X}$ is summed by an aggregation function Σ_1. Output of this unit is processed by an activation function f_1. Similarly, weighted inputs are multiplied by an aggregation function Σ_2. Output of this unit is processed by a different activation function f_2. Weighted outputs of these two units are summed up. Two different activation and aggregation functions endow the GN with flexibility that is not possible in regular MLPs having single activation and aggregation functions (Chaturvedi, Malik, & Kalra, 2004).

The Σ part of the GN is associated with the summation of weighted inputs, and it uses a sigmoidal activation function. The output is obtained as (1).

$$O_\Sigma = f_1(s_{net}) = \frac{1}{1 + \exp(-\lambda_s \cdot s_{net})} \tag{1}$$

where

$$s_{net} = \sum W_{\Sigma i} X_i + X_{o\Sigma}. \tag{2}$$

Here, W_Σ are input weights, $X_{o\Sigma}$ is the bias weight of the Σ section and λ_s is the gain factor of the Σ section. The Π part of the GN is associated with multiplication of weighted inputs. It uses a Gaussian activation function given by (3).

$$O_\Pi = f_2(p_{net}) = \exp(-\lambda_p \cdot p_{net}^2) \tag{3}$$

where

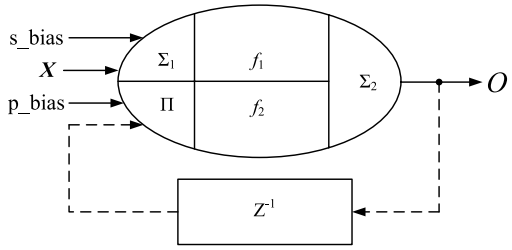

Fig. 1. Structure of a GN (solid lines) and RGN (with feedback shown in dotted lines).

$$p_{net} = \prod W_{\Pi i} X_i \cdot X_{o\Pi}. \tag{4}$$

Here, W_Π are input weights, $X_{o\Pi}$ is the bias weight of the Π section and λ_p is the gain factor of the Π section. The output is obtained as expressed in (5).

$$O_{GN} = W \cdot O_\Sigma + (1 - W) \cdot O_\Pi. \tag{5}$$

A GN has multiple inputs, but only one output. Therefore, the number of GNs required equals the desired number of outputs. A GN having n inputs has $(2n + 1)$ weights and two biases, a total of $(2n + 3)$ trainable parameters. Either or both gain factors λ_s and λ_p can be taken as trainable parameters as well, in which case, the total number of trainable parameters increases accordingly. Other activation functions, such as sine, cosine, or hyperbolic tangent, can also be used. The GN in which the weighted outputs of Σ and Π parts of the proposed GN are added together is called a summation type GN. Weighted outputs of Σ and Π parts can be multiplied in order to construct a multiplication type GN. Both Σ and Π parts partially implement the nonlinear mapping between inputs and the output and constitute the final output. This adds to the flexibility of the GN.

RGN is obtained with a unit delayed output feedback connection in the structure of the GN. Modification to a GN that converts it into an RGN structure is shown in dotted lines in Fig. 1. This modification is represented by (6), (7) and (8).

$$O_{RGN} = W \cdot O_\Sigma + (1 - W) \cdot O_\Pi \tag{6}$$

$$s_{net} = \sum W_{\Sigma i} X_i(t) + X_{o\Sigma} + W_{f\Sigma} \cdot O_{RGN}(t - 1) \tag{7}$$

$$p_{net} = \prod W_{\Pi i} X_i(t) \cdot X_{o\Pi} \cdot W_{f\Pi} \cdot O_{RGN}(t - 1). \tag{8}$$

Here, $W_{f\Sigma}$ and $W_{f\Pi}$ are the feedback weights of the Σ and Π sections respectively. A detailed discussion of the contributions of Σ and Π parts of a GN is presented in Section 4.6.

3. PSO based training algorithm

PSO is an evolutionary-like computational technique that belongs to swarm intelligence, a paradigm of computational intelligence (Venayagamoorthy, 2009). It is a population based parallel search algorithm that models the social behavior of birds within a flock (Kennedy & Eberhart, 1995). It uses a simple concept, and can be implemented with a few lines of computer code. It requires only primitive mathematical operators, and therefore, it is inexpensive in terms of memory and computational requirements. PSO is shown to be more computationally efficient than BP in a neural network training task (Gudise & Venayagamoorthy, 2003). PSO has been applied in optimization problems in such diverse fields as reactive power systems (del Valle, Venayagamoorthy, Mohagheghi, Hernandez, & Harley, 2008), sensor networks (Wimalajeewa & Jayaweera, 2008), and adaptive phased array control (Donelli, Azaro, Natale, & Massa, 2006).