

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.

© 2009 Elsevier Ltd. All rights reserved.

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, Stinchcombe, & White, 1989; Widrow & Lehr, 1990). Feedforward neural networks have been used in applications such as adaptive control (Liu, Kadiramanathan, & 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 (Charytoniuk 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).

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 \mathbf{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).

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

where

$$s_{net} = \sum W_{\Sigma i} X_i + X_{o\Sigma} \quad (2)$$

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).

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

where

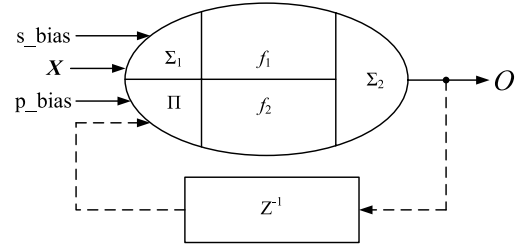


Fig. 1. Structure of a GN (solid lines) and RGN (with feedback shown in dotted lines).

$$p_{net} = \prod W_{\Pi i} X_i \cdot X_{o\Pi} \quad (4)$$

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).

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

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).

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

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

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

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).

PSO consists of a population (or a swarm) of s particles, each of which is a candidate solution. The particles explore an n -dimensional hyperspace in search of the global solution, where n represents the number of parameters to be optimized. The d th dimension of the particle i has the position x_{id} and the velocity v_{id} , $1 \leq i \leq s$ and $1 \leq d \leq n$. The particles are initially assigned random positions and velocities within fixed boundaries, i.e., $x_{\min} \leq x_{id} \leq x_{\max}$ and $-v_{\max} \leq v_{id} \leq v_{\max}$. Each particle in the swarm is evaluated by an objective function $f(x_1, x_2, \dots, x_n)$, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The cost (or the fitness) of a particle is determined from its position in the search space. The cost of a particle closer to the global solution is lower than that of a particle that is farther away. Alternatively, the fitness of a particle closer to the global solution is higher than that of a particle that is farther away. PSO strives to minimize a cost function, or maximize a fitness function. In each iteration, velocities and positions of all particles are updated to persuade them to achieve a lower cost (or, a higher fitness). The process of updating is repeated iteratively, either until a particle approaches the desired solution within a permissible tolerance limit f_T , or until a sufficiently large number of iterations k_{\max} is reached.

In the global-best version of PSO, each particle has a memory to store $pbest_{id}$, the position where it had the lowest cost, and $gbest_d$, the position of the best particle in the population. At each iteration k , the velocity v_{id} and position x_{id} of each particle are updated using (9) and (10).

$$v_{id}(k+1) = w \cdot v_{id}(k) + c_1 \cdot rand_1 \cdot (pbest_{id} - x_{id}) + c_2 \cdot rand_2 \cdot (gbest_d - x_{id}) \quad (9)$$

$$x_{id}(k+1) = x_{id}(k) + v_{id}(k+1). \quad (10)$$

Here, $rand_1$ and $rand_2$ are random numbers with a uniform distribution in the range of $[0, 1]$. The velocity update equation (9) shows that the magnitude and direction of the movement of a particle are influenced by its inertia, its experience and the knowledge it acquires from social interaction with the other particles in the swarm. The first term has the inertia coefficient w , $0.2 < w < 1.2$, which denotes the inertia of the particle (Shi & Eberhart, 1998). The second term has the cognitive acceleration constant c_1 . This component steers the particle towards the position where it had its lowest cost. The third term has the social acceleration constant c_2 . This component steers the particle towards the current best particle. The net change in a particle's velocity is equal to the vector sum of these individual components.

The velocity of a particle in each dimension is bounded between properly chosen limits v_{\max} and v_{\min} (generally, $v_{\min} = -v_{\max}$). This prevents particles from moving in uncontrolled trajectories. Similarly, the position of a particle in each dimension is restricted between properly chosen constants x_{\min} and x_{\max} . The pseudocode for the global best version of PSO for maximization of a fitness function is given in Algorithm 1.

In addition to the global best version, there exists a local best version of PSO, wherein each particle has access to information about positions of only its immediate neighbors (Engelbrecht, 2007). A study indicates that the global best version can converge to the final solution fast, but can get trapped into a local minimum. In contrast, the local best version can avoid local minima, though it converges slowly (Kennedy, 1999). Many other neighborhood topologies exist as well. In addition, there exist real, discrete and binary versions of PSO. Literature is rife with numerous applications of PSO, its variants and hybrids to many complex problems. A comprehensive discussion on the basic concepts of PSO and its variants is given in del Valle et al. (2008).

4. Numerical simulation and results

The flexibility and effectiveness of GN and RGN are demonstrated in the following case studies arranged in the order of increasing complexity.

Algorithm 1 The global-best version of PSO for maximization of a fitness function f

```

1: Initialize  $w$ ,  $c_1$  and  $c_2$ 
2: Initialize maximum allowable iterations  $k_{\max}$ 
3: Initialize the target fitness  $f_T$ 
4: Initialize  $x_{\min}$ ,  $x_{\max}$ ,  $v_{\min}$  and  $v_{\max}$ 
5: for each particle  $i$  do
6:   for each dimension  $d$  do
7:     Initialize  $x_{id}$  randomly:  $x_{\min} \leq x_{id} \leq x_{\max}$ 
8:     Initialize  $v_{id}$  randomly:  $v_{\min} \leq v_{id} \leq v_{\max}$ 
9:   end for
10: end for
11: Iteration  $k = 0$ 
12: while ( $k \leq k_{\max}$ ) AND ( $f(gbest) < f_T$ ) do
13:   for each particle  $i$  do
14:     Compute  $f(x_i)$ 
15:     if  $f(x_i) > f(pbest_i)$  then
16:       for each dimension  $d$  do
17:          $pbest_{id} = x_{id}$ 
18:       end for
19:     end if
20:     if  $f(x_i) > f(gbest)$  then
21:       for each dimension  $d$  do
22:          $gbest_d = x_{id}$ 
23:       end for
24:     end if
25:   end for
26:   for each particle  $i$  do
27:     for each dimension  $d$  do
28:       Compute velocity  $v_{id}(k+1)$  using (9)
29:       Restrict  $v_{id}$  to  $v_{\min} \leq v_{id} \leq v_{\max}$ 
30:       Compute position  $x_{id}(k+1)$  using (10)
31:       Restrict  $x_{id}$  to  $x_{\min} \leq x_{id} \leq x_{\max}$ 
32:     end for
33:   end for
34:    $k = k + 1$ 
35: end while

```

4.1. Case study 1: Realization of XOR function using a GN

Simulation of an XOR learning task is chosen to demonstrate the learning capability and robustness of the GN model. The XOR operation is linearly inseparable, due to the fact that input patterns cannot be separated into the output categories by a single threshold. Historically, this problem has been popular as a good test of a network model and learning algorithm. This popularity probably originates from the fact that it was shown that the single layer perceptron cannot learn this function (Minsky & Papert, 1969). The two-input XOR function takes binary inputs A and B from 0 or 1. Output Y is 0 if both inputs are the same, and 1, if they are not the same.

A two-input GN structure is trained to implement the XOR function. Here, the total number of trainable parameters is nine; Two Σ weights, two Π weights, two biases, W , λ_s and λ_p . The PSO used for training the task has 30 particles, and each particle has nine dimensions. The PSO training is carried out for 1000 iterations. Training inputs are superimposed with additive Gaussian noise with a varying degree of variance. Training patterns, namely, $\{0, 0\}$, $\{0, 1\}$, $\{1, 0\}$ and $\{1, 1\}$ are presented ten times in a random order in each training epoch. Post training validation is performed by presenting each of the test patterns 100 times. Therefore, the total number of validation patterns N is 400. The mean square error (MSE) between output and target output is computed using (11).

$$MSE = \sum_{i=1}^N \frac{(O_i - (A_i \oplus B_i))^2}{N} \quad (11)$$

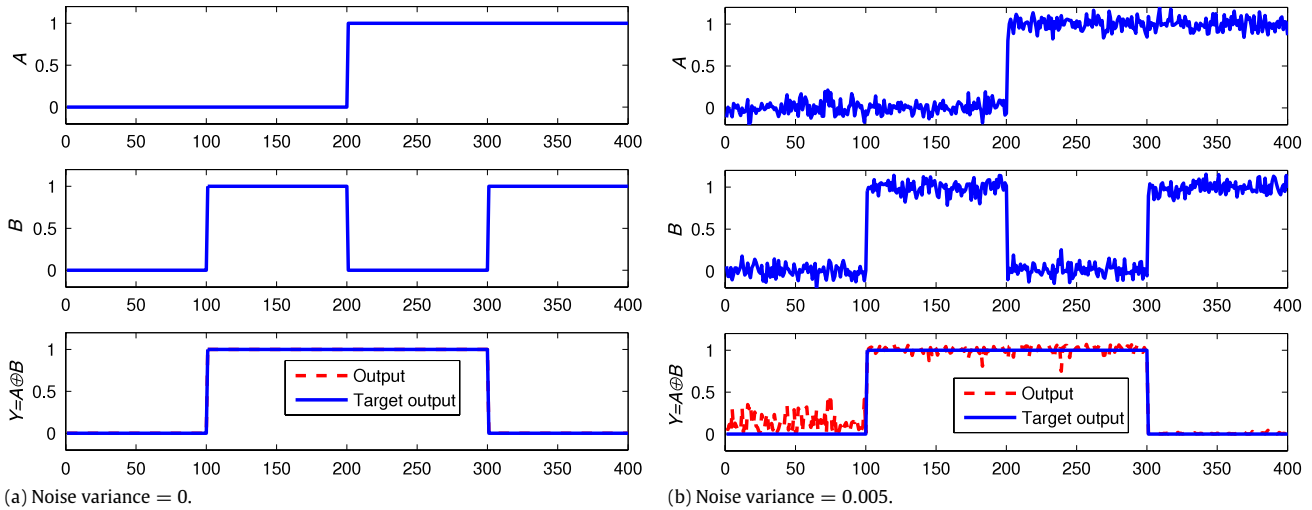


Fig. 2. Outputs of GN XOR gates.

Table 1

Average MSE and its standard deviation over 20 trial runs of XOR simulation.

Variance	MSE	Standard deviation
0	2.3299×10^{-32}	7.3679×10^{-32}
0.0001	6.1354×10^{-4}	4.1759×10^{-4}
0.0005	3.3733×10^{-3}	2.4788×10^{-3}
0.001	4.5204×10^{-3}	3.8445×10^{-3}
0.005	1.0939×10^{-2}	4.8165×10^{-2}

where A_i and B_i represent inputs, and O_i represents output of the GN for the validation pattern i .

The average of MSE and the standard deviation in MSE observed over 30 trial runs for different values of variance is shown in Table 1. Outputs of the GN for a noise variance of 0 and 0.005 are shown in Fig. 2(a) and (b) respectively.

4.2. Case study 2: Approximation of nonlinear function using a GN

In this study, the GN is used to approximate the nonlinear function $f(x) = 2x^2 + 1$, where $x = \sin(2\pi ft)$, $f = 2$ Hz, and t is taken incrementally from $[0, 1]$ in steps of 0.01. The input weights W_Σ , W_Π , W and the bias weights $X_{0\Sigma}$, $X_{0\Pi}$ are taken as trainable parameters. Therefore, the number of training patterns N is 101 and the number of trainable parameters P is five. The gain factors λ_s and λ_p are set to 10 and 1 respectively. The result of function approximation, which is almost indistinguishable from the target output $f(x)$, is shown in Fig. 3. In Table 2, results of GN are compared with that of a $1 \times 5 \times 1$ MLP. This MLP has five input weights, five bias weights and five weights for synaptic connections between hidden and output layer, adding to a total of 15 trainable parameters. This size of MLP is chosen because its performance comes closest to that of the GN. Hidden layer neurons have summation aggregation function and sigmoidal activation function. Output layer neuron has linear activation function. Both GN and MLP are trained using PSO with 30 particles and 1000 iterations. The mean square error between output and the target output, as defined in (12), is used as a measure of fitness in both MLP and GN. P_M , the metric defined in (13), is used for comparing performances of the MLP and GN, where T is training time. The higher the value of P_M , the superior the performance is

$$MSE = \sum_{i=1}^N \frac{(O_i - f(x_i))^2}{N} \quad (12)$$

$$P_M = \frac{1}{P \cdot MSE \cdot T} \quad (13)$$

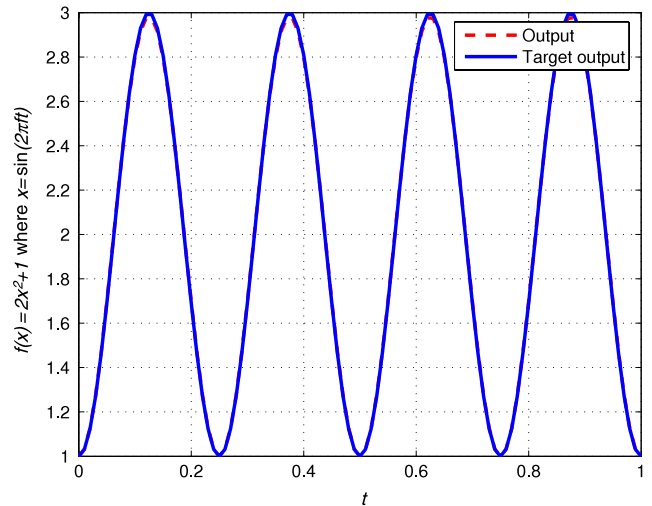


Fig. 3. Results of nonlinear function approximation using GN.

Table 2

Summary of results of nonlinear function approximation by PSO trained MLP and GN.

Model	Trainable parameters P	MSE	Training time T	Performance metric P_M
MLP	15	6.24×10^{-5} (2.50×10^{-5})	60.28 (1.7271)	75.74 (143.41)
GN	5	2.46×10^{-5} (4.15×10^{-5})	1.2312 (0.0271)	3325.12 (1030.45)

Standard deviations are shown in parentheses.

4.3. Case study 3: Estimation of Gaussian probability density function using a GN

Nonparametric approaches, such as histograms, kernel estimators, wavelet based methods and neural network based methods, are frequently used for density estimation (Cichocki & Unbehauen, 1993). This case study introduces a GN structure as an attractive choice for nonparametric estimation of the density function. The Gaussian distribution is considered in this case study. For Gaussian distribution, the cumulative distribution function (CDF) and probability density function (PDF) are given by (14) and (15) respectively.

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt \quad (14)$$

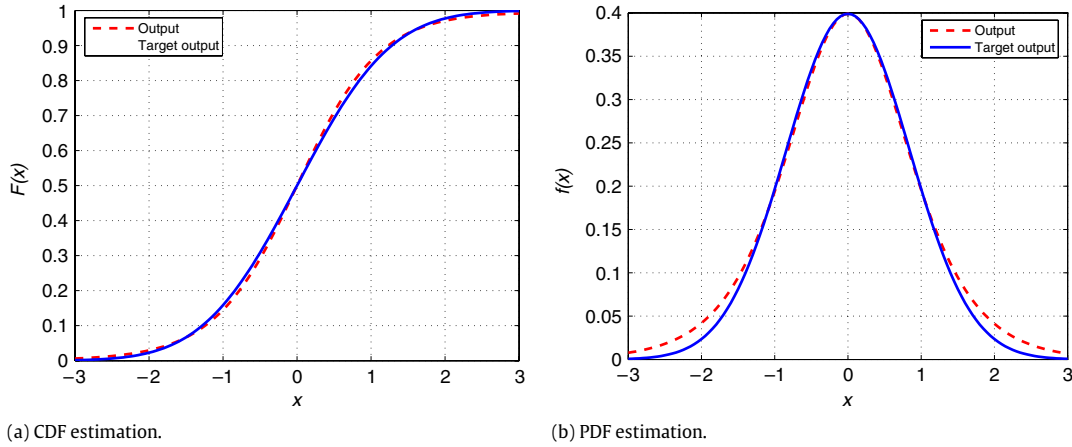


Fig. 4. Results of CDF and PDF estimation for $\mu = 0$ and $\sigma^2 = 2$.

Table 3
Results of CDF and PDF estimation by PSO-trained GN.

Mean (μ)	Variance (σ^2)	MSE in CDF estimation	MSE in PDF estimation
0	1	4.6779×10^{-6} (3.4232×10^{-21})	1.0441×10^{-4} (3.1176×10^{-6})
0.5	1	2.1762×10^{-5} (1.3090×10^{-4})	1.3060×10^{-4} (2.4176×10^{-6})
0.8	4	1.4599×10^{-5} (2.0105×10^{-6})	1.6385×10^{-4} (1.4587×10^{-5})
-0.5	1	3.7979×10^{-6} (2.5056×10^{-21})	7.3918×10^{-5} (1.0774×10^{-12})

Standard deviations are shown in parentheses.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right). \quad (15)$$

Here, μ and σ^2 are the mean and the variance of the distribution respectively. If there is an estimate of CDF $\hat{F}(x)$, then the PDF can be obtained as the derivative of $\hat{F}(x)$, i.e., $\hat{f}(x) = \hat{F}'(x)$. This is obtained as in (16), (17) and (18).

$$\hat{f}(x) = W \cdot dO_{\Sigma} + (1 - W) \cdot dO_{\Pi} \quad (16)$$

where

$$dO_{\Sigma} = O_{\Sigma} \cdot (1 - O_{\Sigma}) \cdot \lambda_s \cdot W_{\Sigma} \quad (17)$$

and

$$dO_{\Pi} = -2 \cdot O_{\Pi} \cdot \lambda_p \cdot x(i) \cdot W_{\Pi}^2. \quad (18)$$

The results produced by the GN in estimation of CDF and PDF for $\mu = 0$ and $\sigma^2 = 1$ are shown in Fig. 4(a) and (b) respectively. MSE for different combinations of μ and σ^2 are given in Table 3. The PSO training algorithm used has 30 particles and 1000 iterations. Input weights, bias weights and the common weight W in (2) and (4) have been taken as trainable parameters; therefore, each PSO particle has five dimensions.

Table 4 gives the comparison of results of a GN in CDF and PDF estimation with that of a $1 \times 5 \times 1$ MLP having a total of 15 trainable weights including biases. This MLP is also trained using PSO with 30 particles and 1000 iterations. Both GN and MLP structures compared are trained for CDF estimation and their derivatives are taken as PDF.

4.4. Case study 4: Prediction of Mackey Glass chaotic time series using a GN and an RGN

Time series prediction through a hybrid of PSO and evolutionary algorithm trained RNN is discussed in Cai et al. (2007). In the

Table 4
Comparison of results of PDF estimation by MLP and GN for $\mu = 0$ and $\sigma^2 = 1$.

Model	Trainable parameters	MSE	Training time	P_M
MLP	15	1.9813×10^{-4} (3.1101×10^{-5})	360.07 (9.2059)	0.1901 (0.0275)
GN	5	1.0441×10^{-4} (3.117×10^{-6})	129.82 (0.2855)	14.7625 (0.1062)

Standard deviations are shown in parentheses.

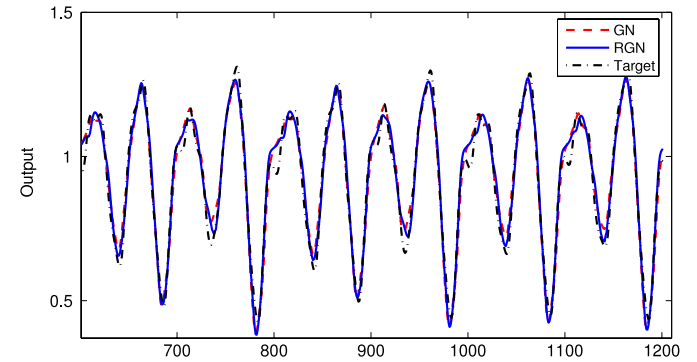


Fig. 5. Predicted and target Mackey Glass chaotic time series.

present study, a recurrent GN is used for predicting the Mackey Glass chaotic time series expressed by (19).

$$\frac{dx(t)}{dt} = \frac{a \cdot x(t - \tau)}{1 + x^c(t - \tau)} - b \cdot x(t). \quad (19)$$

Here, the constants are chosen as $a = 0.2$, $b = 0.1$, $c = 10$ and $\tau = 17$, as done in most of published work. The prediction problem, mathematically amounts to finding a mapping function f_{pr} such that $x(t + Q) = f_{pr}\{x(t), x(t - \Delta), x(t - 2\Delta), x(t - 3\Delta)\}$. Constants chosen are: $\Delta = Q = 6$, and $x(0) = 1.2$. The first 600 input-output data pairs used for training, and the next 601 data pairs used for validation, are extracted from the 1201 length benchmark series. Thus, in validation, 577 data points from 625 through 1201 are predicted. The inputs are taken in the format: $\{x(t - 18), x(t - 12), x(t - 6), x(t); x(t + 6)\}$. The GN used for this problem has 8 input weights, 2 bias weights and one shared weight W . In addition, the gain factors λ_s and λ_p are taken as adaptable weights. Therefore, each PSO particle has 13 dimensions. For RGN, due to two additional feedback input weights, the dimensionality increases to 15. Fig. 5 depicts the target outputs and the outputs of GN and RGN.

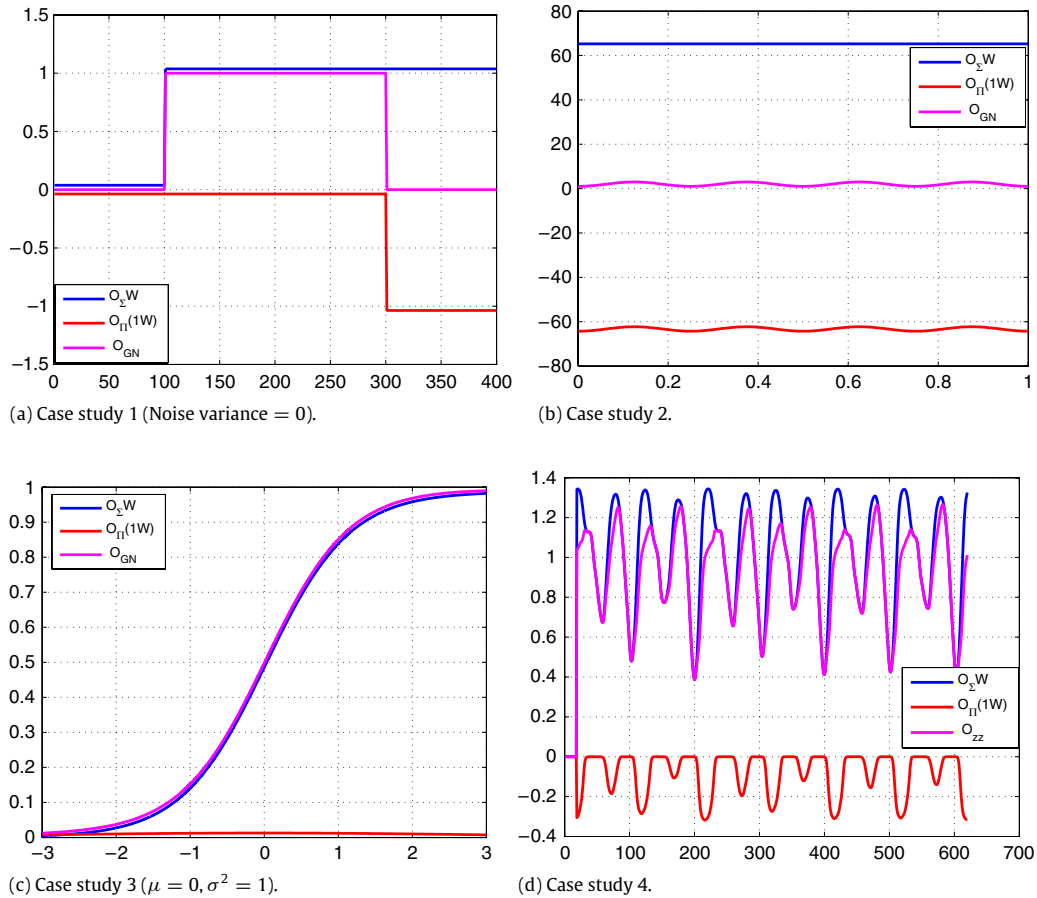


Fig. 6. Contributions from Σ and Π sections of the GN to its output.

Table 5 gives the comparison of performance of GN and RGN with that of MLP and Jordan type RNN. MLP is of size $4 \times 10 \times 1$, and has 40 input weights, ten bias weights, ten weights at the output of hidden layer, and one weight at the bias input of the output layer neuron. Therefore, the number of trainable parameters P is 61. In RNN, due to ten additional feedback input weights, P increases to 71. All models compared here are trained using PSO with 30 particles and 10,000 iterations. MSE in validation data is used in computing P_M .

4.5. Discussion on results

The results of case study 1 show that the GN can perform robust classification. Results of case study 2 indicate that the GN can accurately approximate the nonlinear function. Table 2 shows that GN's performance metric is over 180 times higher than that of MLP. This indicates superior performance. Similarly, in case study 3, GN performs Gaussian PDF estimation accurately for all tested combinations of μ and σ^2 . Table 4 shows that GN has a better performance metric, which is over 75 times higher than that of MLP. Results of case study 4 shown in Table 5 indicate that RGN performs better than MLP, RNN and GN do in chaotic time series modeling. In case study 2, only input weights are trained and in case study 3, input and bias weights are trained. In case study 4, in addition to input and bias weights, gains λ_s and λ_p are adapted. The number of parameters chosen for training depends on the complexity of the problem. A smaller number of trainable parameters and accurate function approximation are the hallmarks of GN and RGN architecture. This makes GN and RGN attractive choices for hardware implementation in stringent memory restrictions.

4.6. Analysis of contributions from Σ and Π parts to the output of the GN

The outputs of the Σ and Π parts, O_Σ and O_Π respectively, together constitute the final output of the GN through a shared weight W as expressed in (5). The presence of two aggregation functions (Σ and Π), and two activation functions (Sigmoid and Gaussian) endows the GN with a flexibility that is difficult to achieve with an MLP having an equal number of weights. Analysis of the contributions of Σ and Π parts shown in Fig. 6 illustrates this fact.

The average final values of W determined by PSO in 30 trial runs of each case study are shown in Table 6. The outputs of Σ and Π parts, namely, $W \cdot O_\Sigma$ and $(1 - W) \cdot O_\Pi$ for case studies 1, 2, 3 and 4 are shown in Fig. 6(a), (b), (c) and (d) respectively.

In case study 1, it is clear from Fig. 6(a) that the outputs of Σ and Π parts compensate for each other. For example, for the patterns between 301 and 400, the output of the Σ part equals the negative of the output of the Π part to produce the desired output of zero. For case study 2, it can be seen that the Π part implements the sinusoidal output with a strong negative offset, which the Σ section nullifies.

Similarly, in case study 3, it can be seen that the Σ part implements the sigmoidal output with a small positive offset, which the Π part nullifies. The target output $O_{GN} = F(x)$ for this experiment is sigmoidal in shape, and the sigmoidal activation function of the Σ section implements it by itself. The Σ part contributes 99.07% of the output, while the Π part contributes only 0.93%.

Case study 4 is more complex problem. Fig. 6(d) shows that for every peak in $O_\Sigma \cdot W$ that exceeds the target, there exists a crest in $O_\Pi \cdot (1 - W)$ to nullify it. The Σ part contributes positive

Table 5

Comparison of results of MLP, RNN, GN and RGN for Mackey Glass time series prediction.

Model	P	Training MSE	Validation MSE	Training time	P_M
MLP	61	0.0166 (1.3×10^{-6})	0.22 (1.5×10^{-5})	1415 (3.67)	4.23×10^{-5} (3.4×10^{-9})
RNN	71	0.028 (1.3×10^{-6})	0.0179 (1.5×10^{-6})	1682 (1.16)	4.72×10^{-5} (8.7×10^{-8})
GN	13	0.0038 (2.7×10^{-8})	0.0017 (1.3×10^{-7})	103 (0.22)	0.19 (0.002)
RGN	15	0.00301 (3.2×10^{-8})	0.00125 (1.1×10^{-7})	123 (0.21)	0.22 (0.002)

Standard deviations are shown in parentheses.

Table 6Average final values of W determined by PSO in 30 trial runs in each case study.

Case study	W	$1 - W$
1	2 (0)	-1
2	72.7817 (7.8564)	-71.7817
3	0.9907 (0)	0.0093
4	1.3634 (0.0023)	-0.3634

Standard deviations are shown in parentheses.

78.95% of the output, while the Π part contributes negative 21.05%. These examples show that the GN uses the outputs of its Σ and Π sections depending on the complexity of the problem, which reflects its superior flexibility compared to that of an MLP.

5. Conclusion

The flexibility of GN for classification, nonlinear function approximation, Gaussian probability density estimation and time series prediction has been investigated in this paper. GN is shown to perform nonlinear function approximation with a very low MSE. The performance of GN in probability density estimation is observed to be accurate for different combinations of mean and variance. RGN has been shown to have accurate predictions of chaotic time series prediction with a smaller number of weights than in MLP and RNN. The GN and RGN can be used to approximate any function, provided the nonlinear activation functions f_1 and f_2 are selected according to the function representing the data. The advantage of GN and RGN structures lies in the smaller number of trainable weights in comparison with that in MLPs and RNNs. Therefore, GN and RGN are suitable for fast and real time implementations on hardware platforms that have memory constraints. Future scope for research lies in the investigation of the issues involved in real-time practical implementation and online training of GN and RGN for nonlinear function approximation and probability density estimation for the chosen real-world problems.

Acknowledgments

Authors acknowledge the support received from the National Science Foundation, USA, under the grants CAREER, ECCS #0348221 and SENSORS, ECCS #0625737.

References

- Atiya, A., El-shoura, S. M., Shaheen, S. I., & El-sherif, M. S. (1999). A comparison between neural-network forecasting techniques – Case study: River flow forecasting. *IEEE Transactions on Neural Networks*, 10, 402–409.
- Cai, X., Zhang, N., Venayagamoorthy, G. K., & Wunsch II, D. C. (2007). Time series prediction with recurrent neural networks trained by a hybrid PSO-EA algorithm. *Neurocomputing*, 70(13–15), 2342–2353.
- Charytoniukand, W., Boxand, E. D., Lee, W. J., Chen, M. S., Kotas, P., & Van Olinda, P. (1999). Neural network based demand forecasting in a deregulated environment, In E. D. Box (Ed.), *Proc. IEEE industrial and commercial power systems technical conference*, p. 7.

- Chaturvedi, D. K., Malik, O. P., & Kalra, P. K. (2004). Experimental studies with a generalized neuron based power system stabilizer. *IEEE Transactions on Power Systems*, 19, 1445–1453.
- Cichocki, & Unbehauen, R. (1993). *Neural networks for optimization and signal processing*. Wiley.
- del Valle, Y., Venayagamoorthy, G. K., Mohagheghi, S., Hernandez, J. C., & Harley, R. G. (2008). Particle swarm optimization: Basic concepts, variants and applications in power systems. *IEEE Transactions on Evolutionary Computation*, 12(2), 171–195.
- Donelli, M., Azaro, R., De Natale, F. G. B., & Massa, A. (2006). An innovative computational approach based on a particle swarm strategy for adaptive phased-arrays control. *IEEE Transactions on Antennas and Propagation*, 54(3), 888–898.
- Engelbrecht, A. P. (2007). *Computational intelligence: An introduction* (2 ed.). New York, USA: John Wiley & Sons.
- Gudise, V. G., & Venayagamoorthy, G. K. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, In *Proc. IEEE swarm intelligence symposium SIS* (pp. 110–117).
- Haykin, S., & Principe, J. (1998). Making sense of a complex world [chaotic events modeling]. *IEEE Signal Processing Magazine*, 15(3), 66–81.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359–366.
- Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Proc. 1999 congress on evolutionary computation*, 1999. CEC 99, Vol. 3.
- Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization, In *Proc. IEEE international conference on neural networks*, (pp. 1942–1948) Vol. IV.
- Lapades, A., & Farber, R. (1987). How neural nets work. In *Proc. advances in neural information processing systems* (pp. 442–456).
- Liu, G. P., Kadiramanthan, V., & Billings, S. A. (1990). Vairable neural networks for adaptive control of nonlinear systems. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 29, 34–43.
- Mackey, M. C., & Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science*, 197, 287–289.
- Minsky, M. L., & Papert, S. A. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT Press.
- Mukherjee, S., Osuna, E., & Girosi, F. (1997). Nonlinear prediction of chaotic time series using support vector machines. In *Proc. IEEE workshop neural networks for signal processing [1997] VII* (pp. 511–520).
- Niu, C. (2005). Medical image registration based on mutual information using kriging probability density estimation. In *Proc. 27th annual international conference of the engineering in medicine and biology society IEEE-EMBS 2005* (pp. 3097–3099).
- Ou, G., & Murphey, Y. L. (2007). Multi-class pattern classification using neural networks. *Pattern Recognition*, 40(1), 4–18.
- Pazhayaveetil, D. C., & Franzon, P. (2007). Flexible low power probability density estimation unit for speech recognition, In *Proc. IEEE international symposium on circuits and systems ISCAS* (pp. 1117–1120).
- Principe, J. C., & Kuo, J. M. (1995). *Advances in Neural Information Processing Systems: Vol. 7. Dynamic modeling of chaotic time series with neural networks* (pp. 311–318). Cambridge, MA: MIT Press.
- Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In *EP'98: Proc. 7th int. conf. on evolutionary programming VII* (pp. 591–600). London, UK: Springer-Verlag.
- Shi, Z., & Han, M. (2007). Support vector echo-state machine for chaotic time-series prediction. *IEEE Transactions on Neural Networks*, 18(2), 359–372.
- Venayagamoorthy, G. K. (2009). A successful interdisciplinary course on computational intelligence. *IEEE Computational Intelligence Magazine*, 4(1), 14–23.
- Wang, J. (1996). *Recurrent neural networks for optimization*. McGraw Hill Inc.
- Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptrons, madaline, and backpropagation. *Proceedings of IEEE*, 78, 1415–1442.
- Wimalajeewa, T., & Jayaweera, S. K. (2008). Optimal power scheduling for correlated data fusion in wireless sensor networks via constrained PSO. *IEEE Transactions on Wireless Communications*, 7(9), 3608–3618.
- Xu, H., Chen, C. H., Cong, F., Yang, L., & Shi, X. (2005). Independent component analysis based on nonparametric density estimation on time–frequency domain. In *Proc. IEEE workshop on machine learning for signal processing* (pp. 171–176).