# Evolutionary product unit based neural networks for hydrological time series analysis

Dulakshi S. K. Karunasingha, A. W. Jayawardena and W. K. Li

## ABSTRACT

Artificial Neural Networks (ANNs) are now widely used in many areas of science, medicine, finance and engineering. Analysis and prediction of time series of hydrological/and meteorological data is one such application. Problems that still exist in the application of ANN's are the lack of transparency and the expertise needed for training. An evolutionary algorithm-based method to train a type of neural networks called Product Units Based Neural Networks (PUNN) has been proposed in a 2006 study. This study investigates the applicability of this type of neural networks to hydrological time series prediction. The technique, with a few small changes to improve the performance, is applied to some benchmark time series as well as to a real hydrological time series for prediction. The results show that evolutionary PUNN produce more transparent models compared to widely used multilayer perceptron (MLP) neural network models. It is also seen that training of PUNN models requires less expertise compared to MLPs.

**Key words** | evolutionary algorithms, neural networks, prediction, time series

**Dulakshi S. K. Karunasingha** (corresponding author)
Department of Engineering Mathematics,
Faculty of Engineering,
University of Peradeniya,
Sri Lanka
E-mail: *dulakshi@yahoo.com*

**A. W. Jayawardena**
International Centre for Water Hazard and Risk Management (ICHARM) under the auspices of UNESCO,
Public Works Research Institute,
Tsukuba,
Japan

**W. K. Li**
Department of Statistics and Actuarial Science,
The University of Hong Kong,
Hong Kong SAR,
China

## ABBREVIATION AND NOTATION

| | |
|---|---|
| $\alpha_i(t)$ | Adaptive parameter |
| $\beta_i$ | Coefficient corresponding to the basis function $B_i(x)$ |
| $\Delta_{\max}$ | Maximum number of mutations |
| $\Delta_{\min}$ | Minimum number of mutations |
| $\Delta A$ | Difference in fitness function before and after a random step |
| $\lambda$ | A user-defined parameter to control adaptive parameters |
| $\rho$ | A user-defined parameter to control adaptive parameters |
| $v$ | A random error |
| $\xi_i(t)$ | Normally distributed random variable |
| $A(g)$ | Fitness function |
| ANN | Artificial neural network |
| ARMA | Autoregressive moving average |
| $B_i(x)$ | Basis function |
| $-C, C$ | Lower and upper bounds for the values for coefficients |
| $-E, E$ | Lower and upper bounds for the values for exponents |
| $H_j(x)$ | Output of a hidden neuron |
| $M_j(x)$ | Output of a multiplicative node |
| MLP | Multilayer perceptrons |
| $MSE(g)$ | Mean square error of the individual $g$ |
| $N_R$ | Number of individuals in the evolving population |
| $NRMSE$ | Normalized root mean square error |
| NSE | Nash–Sutcliffe coefficient of efficiency |
| PU | Product units |
| PUNN | Product units based neural network |
| SPACF | Sample partial autocorrelation function |
| $ST_i(t)$ | Flow measurement at station $i$ at time $t$. |
| $T(g)$ | Temperature in simulated annealing |
| $W_{jt}$ | Weights of neural network |

doi: 10.2166/hydro.2010.099

| | |
|---|---|
| $w_{ji}$ | Exponents of PUNN |
| $\boldsymbol{x}$ | $n$-dimensional input vector |
| $x_i$ | Observed value |
| $\hat{x}_i$ | Predicted value |
| $\bar{x}$ | Average value of the observed time series |
| $y$ | One-dimensional response vector |

## INTRODUCTION

The prediction of impending floods is an essential component of any early warning system for water-related disaster mitigation. One approach of addressing this problem is to consider past information of hydrological time series and attempt to forecast the future. The same principle can be applied to weather forecasting also using past meteorological information. In the recent past, linear stochastic approaches such as autoregressive moving average (ARMA) models were widely used for this purpose. More recently, with the advent of machine learning techniques which have a remarkable ability to deal with nonlinear phenomena, artificial neural networks (ANNs) have gained popularity in the prediction of hydrological/meteorological time series. Multilayer perceptrons (MLP) with sigmoid basis functions are currently the most widely used network type. Many successful applications of MLP, also called Sigmoidal Neural Networks, have appeared in different problems in the last two decades. The ASCE Task Committee on Application of Artificial Neural Networks in Hydrology (2000) summarizes many such applications (see Karunasingha & Liong (2006*a*), Jayawardena (2009), among others, for recent updates). However, one frequent criticism towards ANN models is their lack of transparency. With the sigmoid type basis function the approximated relationship between the inputs and outputs is complex and difficult to interpret. Several alternative based functions have been proposed to alleviate this problem. Among them, multiplicative neural networks could be the most promising. Product unit based neural networks (PUNN) are a type of such multiplicative neural networks, which have the advantages of increased information capacity and the ability to form higher order combinations of the inputs. Despite these advantages one major drawback of this type of networks is the difficulty in training with the back-propagation algorithm compared to standard sigmoidal networks.

Recently, Martinez-Estudillo *et al.* (2006) have proposed a model of evolutionary programming for automatically obtaining the structure and the weights of product units based neural networks. Their method has the added advantage over standard ANN, that it does not require expertise for architectural design (i.e., determination of the number of hidden neurons, connections etc.) and training. Their evolutionary programming model is shown to work well on several benchmark problems of synthetic data and a real dataset of microbial growth, and is considered to be robust. This approach alleviates the problem of tuning many parameters in normal sigmoidal networks, which require the user to be experienced. Also, Martinez-Estudillo *et al.* (2006) have shown (with proof) that similar to sigmoidal neural networks, the product units they have used can approximate any function to a given accuracy.

According to Martinez-Estudillo *et al.* (2006) the advantages of the evolutionary model to train neural networks based on product units compared to those using other techniques are as follows.

- It can obtain both the structure and the weights of the neural network. For a given problem, it is very difficult to know the optimal architecture *a priori* and requires a long process of trial and error. The evolutionary model in which the architecture evolves partially alleviates this problem.
- In theory, it can be viewed as a global optimization algorithm. Although, the convergence of evolutionary algorithms to a global optimum is only guaranteed in a weak probabilistic sense, one of the strengths of evolutionary algorithms is that they perform well on 'noisy' functions where there may be multiple local optima. Evolutionary algorithms tend not to get 'stuck' on local minima and can often find globally optimal solutions. This feature is important as networks based on product units are prone to be trapped in local minima.
- The algorithm does not require an explicit error function. Therefore, it can be applied to problems where such error functions do not exist.

Evolutionary algorithm-based techniques for optimal construction of ANNs is a topic of current interest for helping inexperienced users to use neural networks. Examples of their application to hydrological problems include

groundwater level prediction (Giustolisi & Simeone 2006), rainfall – runoff modeling (Dawson *et al.* 2006), river level prediction (Leahy *et al.* 2008) and reservoir operation (Chaves & Chang 2008), among others. In many such studies, the application of evolutionary techniques is limited to either structural optimization (e.g. number of neurons and connections) or parameter (e.g. weights and biases) optimization. Almost all of these studies have used additive nodes (neurons) with sigmoid-type transfer functions thereby ending up with black-box type ANN models. The evolutionary PUNN proposed by Martinez-Estudillo *et al.* (2006) simultaneously optimizes both the structure and the parameters and produces more transparent ANN models.

This paper explores the applicability of the said evolutionary model to hydrological/meteorological time series prediction. It also compares the evolutionary method with the widely used sigmoidal-type neural networks. The proposed method is first verified using a series of benchmark problems, and then applied to a problem of predicting river flow time series data.

## NEURAL NETWORKS BASED ON PRODUCT UNITS

This section gives an introduction to PUNN applied to regression problems, looks at some of the literature on PUNN and then the PUNN model used in this study and the evolutionary algorithm model proposed by Martinez-Estudillo *et al.* (2006) for training PUNN are explained in detail.

### Basic concepts

In regression applications, the objective is to derive a function that can estimate the desired output variables in terms of observed input variables where such a (unknown) function is believed to exist. Consider a data set $(x_i, y_i)$, $i = 1, 2, \ldots N$ where $\boldsymbol{x}$ is the $n$-dimensional input vector and $y$ is the corresponding one-dimensional response vector. The relationship between the input and output vectors are assumed to take the form

$$y = f(\boldsymbol{x}) + v \tag{1}$$

where $f(\boldsymbol{x})$ is the conditional expectation $E[y|\boldsymbol{x}]$ and $v$ is a random expectation error that represents the 'ignorance' about the dependence of $y$ and $\boldsymbol{x}$. In regression, the objective is to estimate $f(\boldsymbol{x})$ in Equation (1).

Neural networks can be thought of as basis function models (Denison *et al.* 2002) and the function $f(\boldsymbol{x})$ can be written as a linear combination of basis functions and corresponding coefficients as

$$f(\boldsymbol{x}) = \sum_{j=1}^{m} \beta_j B_j(\boldsymbol{x}) \quad \boldsymbol{x} \in \Re^k \tag{2}$$

where $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ is the set of coefficients corresponding to the set of basis functions $B = (B_1(x), B_2(x), \ldots, B_m(x))$. These basis functions are, normally, nonlinear transformations. For example, in radial basis function networks these are called radial basis functions. For a multilayer perceptron network with $k$ inputs and one hidden layer with $m$ nonlinear neurons and a linear output neuron, Equation (2) may be written as

$$f(\boldsymbol{x}) = \sum_{j=1}^{m} \beta_j H_j(\boldsymbol{x}) \tag{3}$$

where the output of a hidden neuron is given as

$$H_j(\boldsymbol{x}) = F\left(\sum_{i=1}^{k} W_{ji} x_i\right). \tag{4}$$

Here, $W_{ji}$ are the weights of the network. Function $F$ in Equation (4) is generally the logistic or the hyperbolic tangent function.

As given in Equations (3) and (4), MLP contains nodes (or units) that add their inputs. The success of the MLP depends on the belief that many real-world problems can be adequately modelled with functions based on linear combinations of the input variables. However, for some problems, higher order combinations of inputs, or ratios of inputs may be more appropriate. Multiplicative neural networks contain units that multiply their inputs instead of adding them. The output of a multiplicative node may be expressed in the form

$$M_j(\boldsymbol{x}) = \prod_{i=1}^{k} x_i^{w_{ji}}, \quad x_i \neq 0 \tag{5}$$

If the exponents, $w_{ji}$, are {0, 1} then it is called a sigma-pi unit. Durbin & Rumelhart (1989) introduced a variation

called product units. In contrast to the sigma-pi units, the exponents in the product units (PU) are not fixed and even may take real values. Such units provide more generality than just allowing polynomial terms, since $w_{ji}$ can take fractional and negative values. Negative exponents make it possible to consider even ratios of inputs.

Increased information capacity and the ability to form higher order combinations of the inputs are the main advantages of the product unit based neural networks. By using the capacity of learning random Boolean patterns, Durbin & Rumelhart (1989) showed empirically that the information capacity of a product unit is approximately $3N$, where $N$ is the number of input variables, compared to $2N$ for a summing unit. Another interesting feature is that similar to sigmoidal neural networks it is possible to obtain upper bounds of the Vapnik – Chervonenkis (VC) dimension (Schmitt 2001) for PUNN too.

The major drawback of the product units is that their training is more difficult than in standard sigmoidal neural networks (Durbin & Rumelhart 1989). The networks based on product units have more local minima and are more likely to get trapped in them. This is the main reason for the difficulty in their training (Ismail & Engelbrecht 2000). Janson & Frenzel (1993) while pointing out that back-propagation is inefficient in training product units used a genetic algorithm for evolving weights in a network with a pre-defined architecture. It has also been shown that global optimization methods such as simulated annealing and random search are impractical for larger networks (Leerink *et al.* 1995). Ismail & Engelbrecht (1999, 2000) showed acceptable performance using particle swarm optimization, genetic algorithm and leapfrog optimization, on three functions in function approximation with low dimensionality. Later, Ismail & Engelbrecht (2002) used a pruning algorithm to improve the structure as well as the training of weights of product unit neural networks. However, all such previous attempts have dealt with only parametric learning. Martinez-Estudillo *et al.* (2006) stand out as the first successful attempt to design both the structure and the weights of the product unit based neural networks.

## Product unit based neural networks

Martinez-Estudillo *et al.* (2006) considered three-layer networks where there is an input layer, a single hidden layer of product units and an output layer of additive units (Figure 1). It is similar to the architecture of a multilayer perceptron with a single hidden layer except that the output function of the two are quite different. The output function of the product unit based neural network can be given as

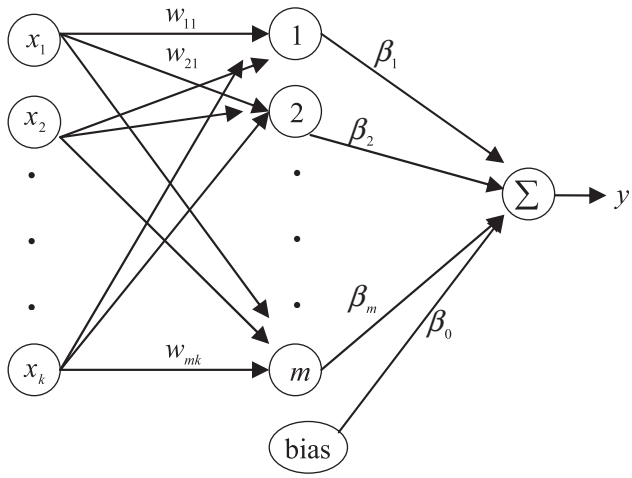


**Figure 1** | Architecture of product units based neural network.

$$f(x_1, x_2, ..., x_k) = \beta_0 + \sum_{j=1}^{m} \beta_j \left( \prod_{i=1}^{k} x_i^{w_{ji}} \right) \tag{6}$$

where $x_i$: $i = 1, 2, \ldots k$ are the input variables, $\beta_j$: $0, 1, \ldots m$ are scalar coefficients and $w_{ji}$: $i = 1, 2, \ldots k$; $j = 1, 2, \ldots m$ are the exponents. It should be noted that when a negative number is raised to a noninteger power, it produces a complex number. This can happen when negative inputs are used in Equation (6). Since neural networks with complex outputs are rarely used in applications, Durbin & Rumelhart (1989) suggested using only the real part for further processing. In such cases, no finite VC dimension bounds can be derived (Schmitt 2001). Therefore, Martinez-Estudillo *et al.* (2006) restricted the input domain to a nonnegative set $A = \{(x_1, x_2, \ldots, x_k) \in \Re^k : x_i > 0,\ i = 1, 2, \ldots, k\}$. This study also uses nonnegative inputs.

## EVOLUTIONARY ALGORITHM FOR TRAINING PRODUCT UNIT NEURAL NETWORKS (PUNNs)

Martinez-Estudillo *et al.* (2006) used a population-based evolutionary algorithm for automatically obtaining the architectural

design and the weights of a neural network based on product units.

In evolutionary algorithms a feasible solution for the problem of interest is coded in an entity called an individual. A group of such individuals is called a population. The population is progressed through generations by applying certain rules called operators. Operators are defined such that the population approaches better solutions over the generations. In this study, an individual consists of the parameters of PUNN, that is, $w_{ji}$: $i = 1, 2, \ldots, k$; $j = 1, 2, \ldots, m$ and $\beta_j$: $j = 0, 1, \ldots, m$, and the architecture of the PUNN, that is, the connections from inputs to hidden layer and the number of hidden nodes. In Figure 1, connections from inputs to hidden layer and hidden to output layer are denoted by arrows. In an individual in the evolutionary algorithm, a connection is denoted by 1, if the connection exists, or zero if the connection does not exist. Unlike in MLPs where all the nodes/inputs are fully connected, the PUNN may have nodes that are not connected to other nodes/inputs.

Similar to any other evolutionary algorithm, Martinez-Estudillo *et al.* (2006) starts with an initial population and the population is updated with some operators in a series of subsequent generations. The population consists of individuals representing the parameters and the architecture of the product unit based networks, that is, one individual represents one neural network based on product units. The algorithm uses the replication operation and two types of mutation operations: parametric and structural mutation. Parametric mutation modifies the coefficients and exponents of the network using a simulated annealing algorithm. In structural mutation, the architecture of the network is modified through the addition or deletion of nodes and connections. Therefore the structural mutation modifies the function represented by the network and allows exploration of different regions of the search space. The crossover operator is not used in the algorithm due to its potential disadvantages in evolving neural networks (Angeline *et al.* 1994). The population is evolved through generations until predefined stopping criteria are met. The main steps of the algorithm of Martinez-Estudillo *et al.* (2006) can be summarized as follows.

(1) Generation of an initial population B of size $10N_R$, where $N_R$ is the number of individuals in the evolving population.

(2) Repetition of the following steps until the stopping criteria are fulfilled.

   (a) Calculation of the fitness of every individual in the population.
   (b) Ranking of the individuals according to their fitness.
   (c) Copying of the best individual into the new population. (The copy of the elite individual will not go through any mutation in the steps (e) and (f)).
   (d) Replacement of the 10 per cent worst individuals with the replications of the 10 per cent of best individuals.
   (e) Application of parametric mutation to the 10 per cent best individuals.
   (f) Application of structural mutation to the remaining 90 per cent of individuals.

The present study generally follows this algorithm with only a few changes that are explained next.

## Generation of the initial population

The initial population is chosen from a larger set of individuals of ten times the population size in the evolutionary process (i.e., 10 $N_R$ where $N_R$ is the population size in the subsequent generations). Martinez-Estudillo *et al.* (2006) generated the number of neurons in the hidden layer within $[0, m/2]$ where $m$ is the maximum number of hidden nodes allowed in subsequent populations. This study selects the hidden neurons from a discrete uniform distribution in the interval $[1, m]$ for each network in the initial population. If the algorithm is as robust as the original study claims, it is believed that such small changes should not affect the performance of the algorithm. The number of connections between a hidden node and the inputs is chosen from a discrete uniform distribution in the interval $[1, k]$, where $k$ is the number of independent variables. Unlike in Martinez-Estudillo *et al.* (2006), all the hidden neurons in this study have connections to the output node. Once the architecture is determined, values are assigned to the exponents and the coefficients, whose connections are defined from continuous uniform distributions in the predefined intervals $[-E, E]$ and $[-C, C]$ respectively. Here, $-E$, $E$, $-C$, and $C$ are the lower and upper bounds for the values for exponents and

coefficients in the first generation. By this procedure, the exponents are determined, the individuals are evaluated and the best $N_R$ solutions among the 10 $N_R$ individuals are chosen as the initial population.

## Fitness function

For an independent and identically distributed training data set $\{(\boldsymbol{x}_i, y_i): i = 1, 2, \ldots, n\}$ of $n$ samples, Martinez-Estudillo *et al.* (2006) used the strictly decreasing function

$$A(g) = 1/(1 + MSE(g)) \tag{7}$$

as the fitness function where $MSE(g)$ is the mean square error of the individual $g$. In this study, the normalized root mean square error ($NRMSE$) is used in place of $MSE(g)$ in Equation (7) to define the fitness of an individual. The $NRMSE$ is expressed as

$$NRMSE = \sqrt{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \Big/ \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{8}$$

where $x_i$ is the observed value, $\hat{x}_i$ is the predicted value, $n$ is the number of points predicted and $\bar{x}$ is the average value of the observed time series. A value of zero for the $NRMSE$ denotes a perfect prediction and a value greater than 1 indicates that the predictions are no better than the long-term average. The $MSE$ depends on the range of output (observed) values. The advantage of using the $NRMSE$ as the fitness function is that it does not depend on the range of observed values. The influence of the fitness function on the efficiency of the algorithm will be discussed in the results and discussion section.

## Parametric mutation

In parametric mutation the exponents ($w_{ji}$) and the coefficients ($\beta_j$) of the product unit networks are updated. This stage can also be seen as an exploitation stage of the evolutionary algorithm compared to structural mutation stage where the main thrust is on exploration. Martinez-Estudillo *et al.* (2006) used a simulated annealing algorithm (Kirkpatrik *et al.* 1983; Otten & Ginneken 1989; Geyer & Thompson 1995) in parametric mutation. Simulated annealing is a strategy that advances through jumps from a current state to another state according to some user-defined mechanism. The closeness of the actual function to any solution of the problem is represented by the 'temperature'. Severity of mutation of an individual $g$ depends on this 'temperature', $T(g)$ which is given by

$$T(g) = 1 - A(g), \quad 0 \le T(g) \le 1. \tag{9}$$

For parametric mutation, the nodes and connections are selected sequentially in the given order, with probability $T(g)$ of the network in a certain generation. Thus, once an individual approaches a solution, the chances of its exponents/coefficients being selected for parametric mutation are less. From one generation to another, each parameter $w_{ji}$, $\beta_j$ of a product unit based network selected for parametric mutation as explained above is changed by an amount of Gaussian noise where the variance depends on the 'temperature' of the network. This allows an initial coarse-grained search, and as the model approaches a solution, a finer grained search. Exponents ($w_{ji}$) update of a single hidden unit is performed in one batch. The exponents, $w_{ji}$, of a network are updated as follows

$$w_{ji}(t+1) = w_{ji}(t) + \xi_1(t), \quad \begin{matrix} i = 1, 2, \cdots, k, \\ j = 1, 2, \cdots, m \end{matrix} \tag{10}$$

where $\xi_1(t) \in N(0, \alpha_1(t)T(g))$ represents a normally distributed random variable with mean 0 and variance $\alpha_1(t)T(g)$, and $\alpha_1(t)$ is an adaptive parameter (which will be explained later) that determines the severity of a mutation. Similarly the coefficients $\beta_j$ of a network are updated as follows

$$\beta_j(t+1) = \beta_j(t) + \xi_2(t), \quad j = 1, 2, \cdots, m \tag{11}$$

where $\xi_2(t) \in N(0, \alpha_2(t)T(g))$ represents a normally distributed random variable with mean 0 and variance $\alpha_2(t)T(g)$. Modification of each $\beta_j$ is made one at a time. This study recognizes $\beta_0$ to be different from the other $\beta_j$s. Therefore, $\beta_0$ is modified in a way different from that of Martinez-Estudillo *et al.* (2006). Once all the other parameters are known $\beta_0$ is the only unknown and hence it is taken to be the average of the training error as

$$\beta_0 = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \hat{y}_i \right) \tag{12}$$

where $y_i$ is the actual value; $\hat{y}_i$ is the simulated value from Equation (6) without taking $\beta_0$ and $n$ is the number of training samples.

After the mutation is performed, the fitness of each individual $g$ is calculated and a simulated annealing criterion is applied. If $\Delta A$ is the difference in the fitness function before and after a random step (i.e., either updating $w_{ji}$ as in Equation (10) or updating $\beta_j$ as in Equation (11)), then, following the simulated annealing strategy, the new step is accepted with probability 1 if $\Delta A > 0$ and with a probability $\exp(\Delta A/T(g))$ if $\Delta A \leq 0$.

The amount of change in the modification of exponents, $w_{ji}$, is different from the modification of coefficients, $\beta_j$. The adaptive parameter of $w_{ji}$ is much smaller than that of $\beta_j$ (i.e., $\alpha_1(t) << \alpha_2(t)$). The adaptation of parameters attempt to avoid the evolutionary process getting trapped in local minima. These are changed in every generation using the following rule.

$$\alpha_k(t) = \left\{ \begin{array}{ll} (1+\lambda)\alpha_k(t), & \text{if } A(g_s) > A(g_{s-1}), \forall s \in \{t, t-1, \cdots, t-\rho\} \\ (1-\lambda)\alpha_k(t), & \text{if } A(g_s) = A(g_{s-1}), \forall s \in \{t, t-1, \cdots, t-\rho\} \\ \alpha_k(t), & \text{otherwise} \end{array} \right\} \quad (13)$$

In Equation (13), $k \in \{1, 2\}$ and $A(g_s)$ is the fitness of the best individual $g_s$ in the generation $s$. The values of $\lambda$ and $\rho$ have to be defined by the user. This study used the same values that Martinez-Estudillo *et al.* (2006) have used (see Table 1). A generation is considered successful if the best individual is better than the best individual of the previous generation.

The parameters $\alpha_1(t)$, $\alpha_2(t)$ and the 'temperature' change throughout the evolutionary process. They determine the severity of mutation and drive the individuals towards better solutions by changing the solution severely when circumstances are good for exploration (i.e., when there are many consecutive successes) or by changing them by small amounts when larger steps produce poor results (i.e., when fitness of the best individual does not change over several generations).

## Structural mutation

Structural mutation modifies the structure of the networks. This encourages exploration of the search space and maintains diversity of the population. Five different structural mutations are conducted. They are (1) node addition; (2) node deletion; (3) connection addition; (4) connection deletion; and (5) node fusion. These five mutations are applied sequentially to each network. For each mutation, the nodes and connections are selected randomly and the number of nodes or connections selected for mutation (except for node fusion) are chosen within the range of minimum ($\Delta_{\min}$) and maximum ($\Delta_{\max}$) number of mutations defined by the user. Thus, the number of elements chosen for mutation may be calculated as

$$\Delta_{\min} + \lfloor uT(g)(\Delta_{\max} - \Delta_{\min}) \rfloor \quad (14)$$

where $u$ is a random value in the interval [0, 1].

**Table 1** | Parameters used in the applications

| Parameter | Value |
|---|---|
| Population parameters: | |
| Maximum number of hidden nodes in a population | 8 |
| Exponent interval $[-E, E]$ | $[-5, 5]$ |
| Coefficient interval $[-C, C]$ | $[-5, 5]$ |
| Parametric mutation parameters: | |
| $\alpha_1(0)$ | 1 |
| $\alpha_2(0)$ | 5 |
| $\lambda$ | 0.1 |
| $\rho$ | 10 |
| Structural mutation parameters: | |
| Interval $[\Delta_{\min}, \Delta_{\max}]$ | |
| Node addition | [1, 2] |
| Node deletion | [1, 5] |
| Connection addition | $[1, 2k]$ |
| Connection deletion | $[1, 2k]$ |

$k$ is the number of input variables

### Node addition

One or more nodes are added to the hidden layer. Nodes may or may not have connections from all input variables. Whether a connection exists between a newly added node and a particular input variable is randomly decided. The $w_{ji}$ values associated with these connections are also generated randomly. The $\beta_j$ values associated with the connections to the output node are similarly assigned.

### Node deletion

One or more nodes are selected randomly and deleted together with their connections.

### Connection addition

One or more connections from input nodes to hidden nodes are added randomly. Their values are also random.

### Connection deletion

One or more randomly chosen connections between input nodes and hidden nodes are deleted.

### Node fusion

Two nodes, $a$ and $b$ are randomly selected from a network and they are replaced by a node $c$, which is a combination of the original two nodes. When there are connections from a certain input variable to the two nodes, such connections are replaced by one connection to node $c$ with weight given by

$$w_{ci} = \frac{w_{ai} + w_{bi}}{2} \tag{15}$$

Other connections from inputs to node $a$ or $b$ are included in $c$ with a probability of 0.5 and their weights are unchanged.

The coefficient of the connection from node $c$ to the output node is modified as

$$\beta_c = \beta_a + \beta_b \tag{16}$$

Martinez-Estudillo *et al.* (2006) claimed that the parameters they used (see Table 1) are quite robust for small variations. This study also used the same parameters.

### Stopping criterion

The evolutionary process is stopped when the maximum number of generations (user defined) is reached, or, when the values of $\alpha_1(t)$, $\alpha_2(t)$ are less than $10^{-4}$.

## METHODOLOGY

This study implemented the above algorithm in Matlab. The method is first applied to the first three benchmark problems used by Martinez-Estudillo *et al.* (2006) to check whether the changes adopted in this study have affected the performance of the algorithm, and to make sure that the current version of the algorithm is no worse than that originally proposed. Then, it is applied to a real hydrological time series (Mekong River flow series) prediction problem. The method is compared with a widely used MLP-type neural network (with a single hidden layer) for prediction accuracy, interpretability, computational time and ease of implementation.

MLP is implemented using the Matlab Neural Network Toolbox. Logistic sigmoid-type activation functions are used in the hidden layer whereas a linear activation function is used in the output layer. The Lavenberg–Marquardt optimization algorithm is used in training with back-propagation. For each dataset, the number of hidden neurons is selected from several trial and error evaluations. For all datasets used, a small number of hidden neurons showed good predictions and therefore the range 3–8 is explored. The prediction error on a testing set is used to ensure generalization (or to avoid over-fitting) of the MLP. For a certain number of neurons five different networks are trained with five different sets of initial weights. The MLP with the least prediction error on the testing set of all such networks corresponding to different number of neurons considered is selected as the best one and is used for the prediction of the unseen data.

Martinez-Estudillo *et al.* (2006) have used the fitness on the training set as the criterion for selecting the best models. However, this can cause overtraining if the number of hidden units is not sufficiently small. This study adopts a testing set and uses the fitness on the testing set to rank the individuals in PUNN. Such an approach is expected to give a balance between training and generalization.

In applications to real data, assessing the strengths and weaknesses of PUNN is given more importance than finding the best model for the problem. For example, for the Mekong River flow data, some previous studies (Jayawardena & Mak 2000; Tian 2007) have identified different combinations of input variables as optimal with different prediction techniques. This study does not attempt to optimize such issues, but rather compares the algorithm

**Table 2** | Benchmark functions

| Function | Domain |
|---|---|
| Friedman's function 1 | |
| $f_1(\boldsymbol{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon_1$ | $x_i \in (0,1), \quad i = 1,2,\ldots,5$ |
| Friedman's function 2 | |
| $f_2(\boldsymbol{x}) = \left(x_1^2 + \left(x_2 x_3 - \frac{1}{x_2 x_4}\right)^2\right)^{1/2} + \epsilon_2$ | $x_i \in [0,100],\ x_2 \in [40\pi, 560\pi],\ x_3 \in [0,1],\ x_4 \in [1,11]$ |
| Friedman's function 3 | |
| $f_3(\boldsymbol{x}) = \tan^{-1}\left(\frac{x_2 x_3 - (1/x_2 x_4)}{x_1}\right) + \epsilon_3$ | $x_i \in [0,100],\ x_2 \in [40\pi, 560\pi],\ x_3 \in [0,1],\ x_4 \in [1,11]$ |

with MLP on similar basis with a few input variable combinations.

The three Friedman's functions used in Martinez-Estudillo *et al.* (2006) are also used in this study. The description of the function and the domain of values used for each variable are summarized in Table 2. The application of the algorithm on these functions is performed in the same way as Martinez-Estudillo *et al.* (2006). The data patterns are randomly generated within the domains defined (Table 2) where $\epsilon_i \in N(0,\ \sigma)$. The standard deviation, $\sigma = 1$ for Friedman function 1 and for the other two functions are chosen so that the signal-to-noise ratio (SNR) is 3:1. Each data set is divided into three parts: training, testing and validation. For the Friedman functions, 200 patterns are taken for training, 200 for testing and 1000 without noise are taken for validation. For the river flow data, about 730 records are taken for training, 365 for testing and 365 for validation. In PUNN, individuals are ranked according to their fitness calculated with *NRMSE* on the testing set. For comparisons within the algorithm (e.g. when updating coefficients and exponents) fitness calculated on the training set is used. For each data set, the evolutionary algorithm is run five times starting with different initial populations corresponding to five different seeds. The maximum number of generations is 2000 for Friedman function 1 and 800 for all other series. The population size was 1000 for all series. In all the series except Friedman function 1, the input variables are normalized by scaling them to the interval [0.1, 0.9] and the outputs are scaled to the interval [1, 2].

Finally, the method is tested on real world data using daily river flow time series of the Mekong River. Mekong River is perhaps the most important and most controversial river in Asia. It originates in China and runs through Myanmar, Lao, Thailand, Cambodia and finally Vietnam before it drains into the South China Sea. The river has a length of about 4620 km and drains a combined area of 795,500 km$^2$. The middle reach of the river has 10 gauging stations. Figure 2 shows the relative locations of these 10 gauging stations

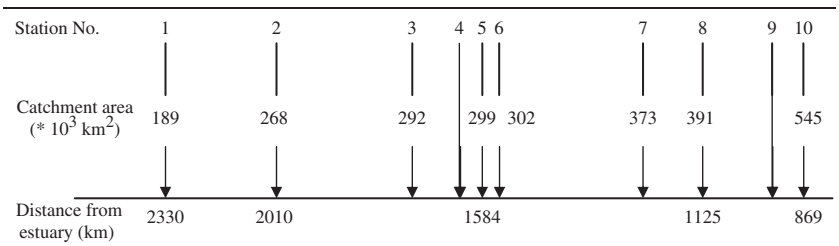


**Figure 2** | Schematic diagram of the locations of the gauging stations.

named Chiang Saen (most upstream), Luang Prabang, Chiang Khan, Pa Mong Dam Site, Vientiane, Nong Khai, Nakhon Phanom, Mukdahan, Khong Chiam and Pakse (most downstream), respectively. The River, while being the lifeline for the people living in the basin can also be destructive at times. Flooding is a recurrent phenomenon and flood damages are also increasing with time partly due to increased population growth in and around the banks of the river. Structural means of mitigating flood damages is quite costly and the riparian countries as well as the Mekong River Commission have adopted alternative nonstructural means of addressing the problem. Early warning is an essential component of nonstructural means of mitigating flood damages where prediction of impending floods plays a crucial role. Previous work carried out on the Mekong River flow and stage prediction include the application of the variable infiltration capacity (VIC) model (Jayawardena & Mahanama 2002), and several other studies including the stream flow synthesis and reservoir regulation (SSARR) model for the upper part of the basin, multiple regression models for the lower reach of the delta with overbank flow, and the MIKE-11 for flood mapping in Mekong Delta, carried out by the Mekong River Commission as reported by Manusthiparom & Apirumanekul (2005) and Apirumanekul (2006). However these works do not present any results.

Several previous studies have used ANN prediction of Mekong River flow data (Jirayoot & Al-Soufi 2000; Jayawardena *et al.* 2004; Jayawardena 2009; among others). In this study, daily river flows measured at the above referenced 10 stations of the Mekong River from January 1991–December 1994 are used (IDI, Water Series No. 10). These are averaged values compiled from two-stage measurements made every day at 7:00 am and 7:00 pm (pers. comm., Mekong River Commission 1998). The stage measurements are converted into discharge measurements using a rating curve which makes use of more frequent stage and corresponding flow measurements. Two different models with different input variable combinations are considered: (1) predicting one day-ahead flow at Pakse with current day's flow and the previous two days' flows (i.e., two time-lagged values) of Pakse station and (2) predicting one day-ahead flow at Pakse station with the current records of Pakse and the other nine stations. The two models may be expressed respectively by the following Equations.

$$ST_{Pakse}(t+1) = F_1(ST_{Pakse}(t), ST_{Pakse}(t-1), ST_{Pakse}(t-2)) \quad (17)$$

$$ST_{Pakse}(t+1) = F_2(ST_1(t), ST_2(t), \ldots, ST_{10}(t)) \quad (18)$$

The two cases are expected to represent two common situations in predicting real world data: (1) predicting a variable of a system with its own time-lagged records (time series data), and (2) predicting a variable with the help of other variables of the system.

## RESULTS AND DISCUSSION

The results obtained in this study for benchmark problems are shown in Tables 3 (with MLP) and 4 (with PUNN). The results obtained by Martinez-Estudillo *et al.* (2006) are also summarized in Table 5 for comparison purposes. The results obtained for the Mekong River flow prediction are shown in Tables 6 (with MLP) and 7 (with PUNN). For the latter two cases, the Nash–Sutcliffe coefficient of efficiency (NSE) is also provided. It should be noted that the best solutions of Martinez-Estudillo *et al.* (2006) are chosen from 30 trials and in this study from only five trials. For MLP models, the mean MSE is calculated considering only the best six models

**Table 3** | Regression errors on benchmark functions with MLP

| | Regression errors | | | | | | |
|---|---|---|---|---|---|---|---|
| | Training | | | Validation | | | |
| Function | NRMSE | MAE | MSE | NRMSE | MAE | MSE | Mean MSE |
| Friedman 1 | 0.1845 | 0.7071 | 0.8176 | 0.1318 | 0.4992 | 0.4252 | 1.0001 |
| Friedman 2 | 0.4793 | 160 | 42690 | 0.1444 | 43 | 2824 | 4579 |
| Friedman 3 | 0.5361 | 0.1562 | 0.0393 | 0.2433 | 0.0389 | 0.0061 | 0.0105 |

**Table 4** | Regression errors on benchmark function with evolutionary PUNN

| | Regression errors | | | | | | |
|---|---|---|---|---|---|---|---|
| | Training | | | Validation | | | |
| Function | NRMSE | MAE | MSE | NRMSE | MAE | MSE | Mean MSE |
| Friedman 1 | 0.2302 | 0.8593 | 1.1518 | 0.1013 | 0.3858 | 0.2448 | 0.8517 |
| Friedman 2 | 0.5305 | 179 | 48,193 | 0.1330 | 39 | 2,404 | 2,555 |
| Friedman 3 | 0.5547 | 0.1580 | 0.0411 | 0.3092 | 0.0608 | 0.0108 | 0.0099 |

**Table 5** | Regression errors obtained by Martinez-Estudillo *et al.* (2006) on benchmark function with evolutionary PUNN

| | Regression errors | | |
|---|---|---|---|
| | Training | Validation | |
| Function | MSE | MSE | Mean MSE |
| Friedman 1 | 0.9941 | 0.1697 | 0.9360 |
| Friedman 2 | 39452 | 1851 | 5538 |
| Friedman 3 | 0.2657 | 0.0098 | 0.0500 |

obtained from 30 such models corresponding to different numbers of neurons and different initial weight settings.

## Application to benchmark problems

The results of different studies may not be directly comparable. However, comparison of the results in Tables 3 and 4 with those in Table 5 indicate that this study has also achieved the level of accuracy of Martinez-Estudillo *et al.* (2006) and the performance of the algorithm is not affected by the small changes incorporated in this study. Also, consistently smaller mean MSE values obtained in this study compared to those of Martinez-Estudillo *et al.* (2006) on the validation set suggest that the use of a testing set may have been effective in better generalization. Tables 3 and 4 show that except in Friedman function 3, the PUNN performance is markedly better than that of MLP. The best evolved network for Friedman function 1 is given below.

$$\begin{aligned} f_1(\boldsymbol{x}) = {} & 4.552 + 18.703x_1^{0.65625}x_2^{0.71813}x_3^{-0.04125} \\ & + 9.6839x_4^{1.0912} - 23.78x_1^{3.4989}x_2^{3.6323} \\ & + 7.0429x_3^{4.4319}x_4^{0.01628} + 5.097x_5^{1.1022} \\ & - 6.4168x_3^{0.42733} + 1.0873x_1^{0.82454}x_2^{0.6984}x_3^{0.45759} \\ & + 0.54907x_1^{0.67348}x_3^{0.9882}x_4^{0.49859} \end{aligned} \tag{19}$$

Although, the exponents in Equation (19) have been reported to five decimal places, rounding them off to two decimal places does not significantly affect the performance of the models.

An interesting result noted in this study is, that in Friedman functions 2 and 3 where the variable $x_4$ plays a minor role, the best solution did not contain $x_4$. It is to be noted that for the considered domains, $x_4$ has, on average, no effect on the function values, as shown below, and ignoring the term $\frac{1}{x_2 x_4}$ completely in functions 2 and 3 makes no difference. For example, the ratio calculated according to

$$ratio = \sum_i (abs(Fx_i - Fxr_i)) / \sum_i (abs(Fx_i)) \tag{20}$$

**Table 6** | Prediction errors on Mekong River flow with MLP

| | Prediction errors | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Training | | | | Validation | | | | |
| Model | NRMSE | NSE | MAE ($m^3/s$) | MSE $(m^3/s)^2$ | NRMSE | NSE | MAE ($m^3/s$) | MSE $(m^3/s)^2$ | Mean MSE |
| Equation (17) | 0.0670 | 0.9955 | 250 | 332,068 | 0.0693 | 0.9952 | 349 | 546,433 | 563,070 |
| Equation (18) | 0.0604 | 0.9964 | 266 | 262,785 | 0.0904 | 0.9918 | 556 | 930,063 | 1,489,007 |

**Table 7** | Prediction errors on Mekong River flow with evolutionary PUNN

| | Prediction errors | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Training | | | | Validation | | | | |
| Model | NRMSE | NSE | MAE ($m^3/s$) | MSE $(m^3/s)^2$ | NRMSE | NSE | MAE ($m^3/s$) | MSE $(m^3/s)^2$ | Mean MSE |
| Equation (17) | 0.0746 | 0.9944 | 277 | 410,622 | 0.0603 | 0.9964 | 314 | 413,460 | 417,064 |
| Equation (18) | 0.0773 | 0.9940 | 312 | 430,602 | 0.0716 | 0.9949 | 455 | 583,258 | 654,182 |

where $Fx_i$ is the functional value using the given formulae, $Fxr_i$ is the functional value excluding $\frac{1}{x_2x_4}$ from the given formulae, abs denotes absolute value, gives a value close to $7^*10^{-7}$ for Friedman function 2 and a value close to $10^{-6}$ for Friedman function 3. Thus, it amounts to model a function of the other 3 variables: $x_1$, $x_2$ and $x_3$. For Friedman function 3, four out of the five best PUNN solutions (corresponding to five different seeds) did not contain $x_4$. For Friedman function 2 it was one out of five. This observation suggests that PUNN has some ability to identify the most important variables. However, it is to be noted that the evolutionary algorithm does not guarantee the exclusion of unnecessary variables from their evolved models over a limited number of generations. Rather checking the number of occurrences of a certain variable in the best solutions of the final generation may provide an idea about the relative importance of the variables.

## Application to Mekong River

The best model obtained by PUNN when only data records of Pakse are used is given as

$$y = 0.83472 + 1.2765x_1^{1.4857}x_2^{-0.5156} - 0.25013x_1^{4.4023}x_2^{4.2687} \tag{21}$$

where $x_1$ = normalized $ST_{Pakse}(t)$, $x_2$ = normalized $ST_{Pakse}(t-1)$, and $y$ = scaled flow value at Pakse at time $t+1$ and, the best ANN model is given as

$$y = f(\boldsymbol{X}) = (\boldsymbol{W}2 * \text{logsig}(\text{W}1 * X + \text{B}1) + B2) * (n2 - n1) + n1 \tag{22}$$

where $X$ are the variables,

$W1$ are weights from inputs to hidden layer,

$B1$ are the biases in the hidden layer,

$W2$ are the weights from hidden to output layer,

$B2$ is the bias in the output layer and

$n1$ and $n2$ are values used to normalize data and $\text{logsig}(n) = 1/(1+\exp(-n))$

$$\boldsymbol{X} = [x_1, x_2, x_3]$$

where $x_1$ = normalized $ST_{Pakse}(t)$, $x_2$ = normalized $ST_{Pakse}(t-1)$ and so on

$$\text{W}1 = \begin{pmatrix} 7.1194 & -8.5445 & -4.7602 \\ 1.1398 & 13.3273 & -1.8527 \\ 2.6703 & -8.9358 & 8.0118 \\ -6.9870 & 12.2898 & 7.6340 \\ -6.8036 & -4.5599 & 5.3689 \end{pmatrix}$$

$$\boldsymbol{W}2 = (-0.8106 \quad -0.3265 \quad 0.8682 \quad 0.2742 \quad -1.3554)$$

$$\boldsymbol{B}1 = \begin{pmatrix} 0.7354 \\ -1.5815 \\ -0.9867 \\ -8.2243 \\ -1.2881 \end{pmatrix}$$

$$\text{B}2 = (1.4457) \tag{23}$$

Note that Equation (22) gives the general formulae corresponding to the MLP model used in this study, and when the variables, weights and biases given in Equation (23) are substituted in Equation (22)), it gives the specific MLP model.

Prediction errors in the Mekong River flow (Tables 6 and 7) show that PUNN has outperformed MLP in prediction accuracy in validation. The results are consistent with all error indicators. The scatter and series plots (Figures 3 and 4) of predicted and observed data in validation also show that the predictions are very good. Another interesting aspect of PUNN results compared to that of ANN is the transparency in the derived models. It can be seen that Equation (21) obtained by PUNN is much simpler than those for ANN (Equations (22) and (23)). Also, a simple examination of the equation gives some idea as to how the variables and the individual terms affect the output. The first term of Equation (21) indicates that when $x_1$, with its positive coefficient and

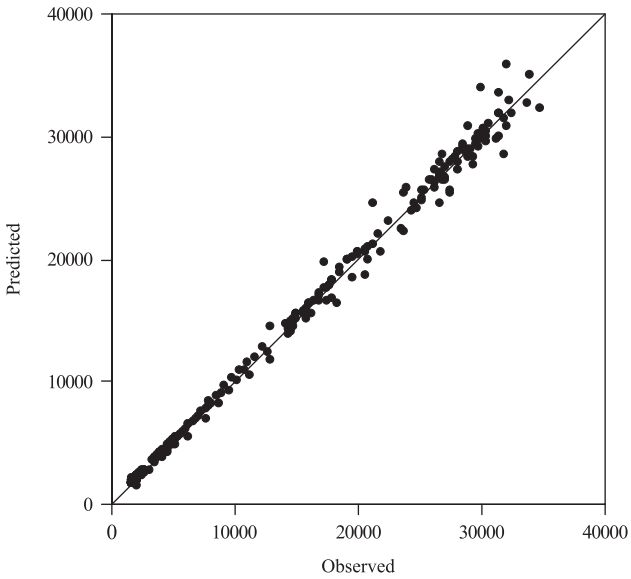


**Figure 3** | Predicted versus observed values (in $m^3/s$) for validation data using model Equation (17).

exponent, increases $y$ also increases, and when $x_2$, with its positive coefficient and negative exponent, increases then $y$ decreases. The magnitude of the second term, in the range of the normalized input variables, is considerably small compared to that of the first term and, the overall effect of $x_1$ and $x_2$ on $y$ remains unchanged as explained by the first term. This fact can be more mathematically verified by noting that, the first partial derivatives of $y$ with respect to $x_1$ and $x_2$ remain positive and negative respectively throughout the domain considered showing that $y$ is monotonically increasing and decreasing with respect to $x_1$ and $x_2$. Although the contribution of the second term in Equation (21) is negligible at low flow rates (i.e., normalized values close to 0.1) it has not been removed from the model since it could be a useful contributor at high flow rates even if its magnitudes are relatively small.

Partial Autocorrelations

| Lag | Correlation | -1 9 8 7 6 5 4 3 2 1 0 1 2 3 4 5 6 7 8 9 1 |
|---|---|---|
| 1 | 0.99615 | .\|********************\| |
| 2 | -0.52323 | **********\|. \| |
| 3 | 0.06152 | .\|* \| |
| 4 | 0.08678 | .\|** \| |
| 5 | 0.08996 | .\|** \| |
| 6 | -0.05256 | *\|. \| |
| 7 | -0.03812 | *\|. \| |
| 8 | -0.00731 | .\|. \| |
| 9 | 0.05432 | .\|* \| |
| 10 | -0.00697 | .\|. \| |
| 11 | 0.06483 | .\|* \| |
| 12 | -0.01896 | .\|. \| |
| 13 | -0.04831 | *\|. \| |
| 14 | 0.00534 | .\|. \| |
| 15 | -0.03180 | *\|. \| |

**Figure 5** | SAS output for sample partial autocorrelation function of flow series at Pakse.

Such an interpretation of the model is not straightforward with the ANN model.

Interesting observations are also made with the sample partial autocorrelation function (SPACF) of the flow series. Figure 5 shows the SPACF of Mekong River flow measured at Pakse. There is a significant positive spike at lag 1 and a negative spike at lag 2, indicating that $x_1$ and $y$ have a positive correlation, and $x_2$ and $y$ have a negative correlation when the effect of $x_1$ is accounted for. This observation is consistent with what was observed in Equation (21). Figure 5 also shows that after lag 2, the SPACF cuts off, indicating that further lags do not have much explanatory power when the effects of $x_1$ and $x_2$ are accounted for. Although SPACF is a measure of linear correlation, practically, the implications do not differ very much for nonlinear dependencies. An unimportant lag 3 supports the derived PUNN model (Equation (21)), where $x_3$ is absent.

It should be noted that in this case, the positive and negative correlations of $x_1$ and $x_2$ with the output is apparent in the model for the whole domain considered. However, in

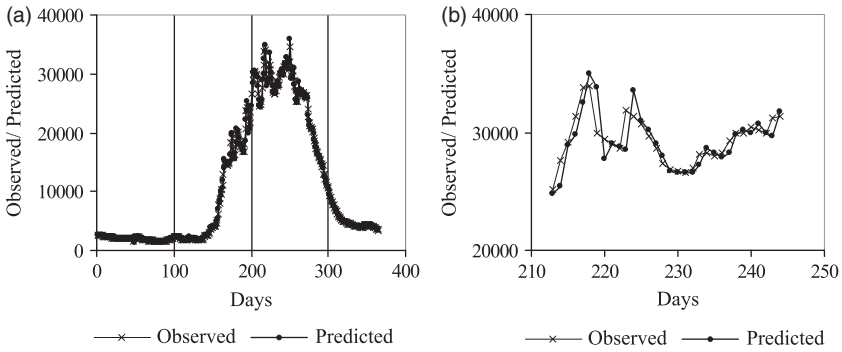


**Figure 4** | Observed and predicted series values (in $m^3/s$) for validation data using model Equation (17). (a) for the whole year 1994; (b) only August is focused on for better clarity.

general, for the models that describe nonlinear relationships, it is not fair to expect positive/negative correlations among dependent/independent variables to remain the same throughout the domain. Since the effect of individual terms on the output depends on the values of the coefficients, the exponents as well as the range of the input variables, one may have to search different regions of the input domain in order to find such different correlations.

The best models obtained by PUNN and ANN for Mekong River flow prediction with 10 upstream stations are given respectively as

$$\begin{aligned} y = {} & 0.8076 + 1.3727x_3^{-0.00751}x_8^{0.12244}x_9^{0.2643}x_{10}^{0.56921} \\ & + 0.47142x_2^{1.9317}x_7^{0.49439}x_8^{1.1259}x_9^{1.4487}x_{10}^{0.28721} \\ & - 0.3286x_4^{1.0772}x_5^{0.97303}x_7^{0.72872}x_9^{0.92274} \\ & + 0.17594x_1^{0.68219}x_2^{0.93295}x_3^{0.94393}x_4^{0.69752}x_8^{0.30639}x_9^{0.17184} \end{aligned} \tag{24}$$

where $x_1$ = normalized $ST_1(t)$, $x_2$ = normalized $ST_2(t)$ and so on and,

$$\mathrm{W1} = \begin{pmatrix} -1.963 & -2.113 & -3.356 & -1.788 & 4.228 & -0.601 & -1.386 & -6.415 & -0.692 & -0.947 \\ 0.477 & -1.216 & -1.560 & -5.028 & -3.185 & -1.340 & 2.925 & 1.513 & -2.233 & 2.863 \\ 0.044 & -4.546 & -0.439 & 4.029 & 4.257 & 3.723 & -1.784 & -0.609 & -3.580 & -0.614 \\ 0.037 & 1.118 & 0.235 & -1.473 & -0.658 & -0.561 & 1.146 & 1.251 & 1.671 & 3.334 \\ 0.988 & 2.812 & -1.253 & -1.534 & -0.004 & -3.121 & -1.068 & 1.802 & 7.732 & -2.659 \end{pmatrix}$$

$$\boldsymbol{W}2 = (\,0.175 \quad -0.098 \quad 0.233 \quad 0.821 \quad 1.374\,)$$

$$\boldsymbol{B}1 = \begin{pmatrix} 10.957 \\ 5.228 \\ 0.208 \\ -1.462 \\ -3.505 \end{pmatrix}$$

$$\mathrm{B2} = (0.815)$$

$$\boldsymbol{X} = [x_1, x_2, \ldots, x_{10}]. \tag{25}$$

This also shows the simplicity and transparency in equations derived by PUNN compared to those of ANN. In practical situations, these kinds of simpler expressions are always preferred compared to complicated expressions. It is seen that $x_6$ (i.e., the discharge at station 6) has no contribution to Equation (24). This may mean that the contribution of discharge from station 6 is already contained in other variables. Also, the second term in Equation (24) has the variable $x_3$ raised to the power of $-0.00751$. In this case, removing variable $x_3$ from that particular term does not change the performance of the model. In the case of the Mekong River, the interpretation of the derived model (i.e., Equation (24)) is not as straightforward as the model in Equation (21). It is a too complex system even to understand physically, and that is beyond the scope of this paper. However, the results are sufficient to show that the evolutionary PUNN can perform well with relatively large numbers of variables as well.

## Other observations

PUNN has been able to derive more transparent and simpler expressions compared to standard ANN models while still keeping the prediction accuracy (in fact, with better prediction accuracy for the time series considered in this study). The slight changes adopted in this study in implementing the algorithm of Martinez-Estudillo *et al.* (2006) have not adversely affected its effectiveness. This supports Martinez-Estudillo *et al.*'s claim that the algorithm is robust for small changes in parameters. One may wonder how the performance of PUNN compares with other popular regression techniques such as support vector machines (SVM). Karunasingha & Liong (2006b) compare ANN and SVM for prediction accuracy, computational time and ease of implementation. They report that for fairly large data records both ANN and SVM are equally good in terms of prediction accuracy. Combined with the results obtained in this study, it can be deduced that PUNN to be as equally good as SVM in terms of the prediction accuracy.

Our experience with real data (the method has also been tested on an algal bloom problem where results are not

shown) suggests that PUNN is more effective in terms of prediction accuracy than standard ANN on datasets with relatively large numbers (our data contained up to about 10) of variables. However, the computational time taken by PUNN (implemented in Matlab) is considerably higher than with ANN especially when the number of data records is large.

The results of this study confirm that the algorithm of Martinez-Estudillo *et al.* (2006), as it is, works well. Also, the study noted that small modifications can lead to even better performance in terms of both accuracy and computational effort. It was noted (results not shown), that updating $\beta_0$ in the suggested way leads to a fast convergence of evolved networks to better solutions. Such modifications for improvements are beyond the scope of this paper, however, our experience indicates investigations on the following areas may be useful.

In genetic algorithms small populations converge fast. It was noted that a population of size 200 and with 2000 generations and application of parametric mutation to 50 per cent of the population instead of 10 per cent in the original algorithm, produces models with equally good accuracies as those reported in Table 4 for Friedman function 1. This is about an 80 per cent reduction of computational burden. Therefore, we believe incorporation of the ideas of micro-genetic algorithms (Krishnakumar 1989), which uses the fast convergence of small populations, may improve the efficiency of the algorithm.

This study also noted that fitness function plays a considerable role in the convergence of the population and hence the efficiency of the algorithm. This is because $A(g)$ is related to the probabilities of parametric mutation. When $A(g)$ is 'large' the parametric mutation slows down since the number of connections and hence the number of networks that gets a mutation is very small. Therefore, it is important that $A(g)$ stays 'small' enough until the networks converge to acceptable solutions. This is especially important when small population sizes are used. Hence choosing appropriate error measures to define the fitness function should be given attention.

The Mekong River results showed that evolutionary PUNN is capable of producing simpler models, in which the dependencies among variables are apparent, compared to standard ANN. However, PUNN is still a data-driven model and, when the models are developed and chosen with prediction accuracy as the only criterion one might not always be guaranteed of a model that indicates the true physical nature of the relationship between the inputs and outputs. A user familiar with the physics of the system may prefer a model that explains such physical characteristics. Incorporating multi-objective optimization techniques with the evolutionary PUNN may provide the user with a set of feasible models to choose from. This study believes evolutionary algorithms for multi-objective optimization such as NSGA–II (see Srinivas & Deb 1994; Deb *et al.* 2002) may be used for this purpose.

Although, in this study, PUNN is illustrated in river flow prediction, this is not the only application for PUNN. There are many other problems in hydrology, water resources, fluid mechanics, meteorology etc. where models of this nature are preferred/useful. Application of the algorithm in different areas can be useful to identify its strengths and weaknesses compared to existing techniques.

## CONCLUSION

The product unit based neural networks trained with evolutionary programming technique applied in regression is an alternative technique to generally used sigmoidal-type neural network models. The technique provides the same or an even better level of accuracy compared to standard sigmoidal neural network models. Another advantage of evolutionary PUNN is that much less expertise is needed to determine the parameters. Widely used sigmoidal neural network models have more parameters to fine tune and requires an experienced user to build a good model. Evolutionary PUNN has the advantage of having a robust set of parameters that works well on many different problems. However, one drawback of evolutionary PUNN is that the computational time required is higher compared to traditional ANN.

Lack of transparency in the approximated function is the most frequent criticism towards sigmoidal-type neural networks. Unlike the functions derived by sigmoidal networks, the functions produced by evolutionary PUNNs have some simplicity and transparency in terms of input variables. This is a desirable feature when analysing real data. More applications of this technique in different real world problems are required to further explore and consolidate this feature.

## ACKNOWLEDGMENTS

W. K. Li's research was partially supported by the Area of Excellence Scheme, under the University Grants Committee of the Hong Kong Special Administration Region China. (Project No. AoE/P-04/2004). D.S.K. Karunasingha acknowledges the Department of Statistics and Actuarial Science of the University of Hong Kong for supporting this research through a visiting research associate position. Much of the work presented in this paper was carried out while A.W. Jayawardena was attached to the Department of Civil Engineering of the University of Hong Kong.

## REFERENCES

Angeline, P. J., Saunders, G. M. & Pollack, J. B. 1994 An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Netw.* **5**(1), 54–65.

Apirumanekul, C. 2006 Flood forecasting and early warning systems in Mekong River Commission. In: *Proc. of 4th Annual Mekong Flood Forum, Siem Reap, Cambodia*, 18–19 May 2006, Chapter 2 – *Flood Forecasting and Early Warning Systems in Mekong River Commission*. pp. 145–151.

ASCE Task Committee on Application of Artificial Neural Networks in Hydrology 2000 Artificial neural networks in hydrology II: Hydrologic applications. *J. Hydrol. Engng.* **5**(2), 124–137.

Chaves, P. & Chang, F. J. 2008 Intelligent reservoir operation system based on evolving artificial neural networks. *Adv. Water Resour.* **31**, 926–936.

Dawson, C. W., See, L. M., Abrahart, R. & Heppenstall, A. J. 2006 Symbiotic adaptive neuro-evolution applied to rainfall–runoff modeling in northern England. *Neural Networks* **19**, 236–247.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. 2002 A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 181–197.

Denison, D. G. T., Holmes, C. C., Mallick, B. K. & Smith, A. F. M. 2002 *Bayesian Methods for Nonlinear Classification and Regression*. Wiley, West Sussex, UK.

Durbin, R. & Rumelhart, D. 1989 Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Comput.* **1**, 133–142.

Geyer, C. J. & Thompson, E. A. 1995 Annealing Markov chain Monte Carlo with applications to ancestral inference. *J. Am. Stat. Assoc.* **90**(431), 909–920.

Giustolisi, O. & Simeone, V. 2006 Optimal design of artificial neural networks by a multi-objective strategy: ground water level predictions. *Hydrol. Sci. J.* **51**(3), 502–523.

Infrastructure Development Institute (IDI), Japan: (1960–1994): The Mekong River Hydrological Database (1960–1994) in CD-ROM.

Ismail, A. & Engelbrecht, A. P. 1999 Training product units in feedforward neural networks using particle swarm optimization. In: *Proc. of the International Conference on Artificial Intelligence – Development and Practice of Artificial Intelligence Techniques* (ed. in V. B. Bajic & D. Sha). Durban, South Africa. pp. 36–40: http://sci2s.ugr.es/keel/pdf/specific/congreso/Ismail_Engelbrecht_1999.pdf

Ismail, A. & Engelbrecht, A. P. 2000 Global Optimization Algorithms for Training Product Unit Neural Networks. In: *Proc. Institute of Electrical and Electronic Engineers (IEEE)- International Neural Networks Society(INNS) - European Neural Networks Society (ENNS) International Joint Conference on Neural Networks (IJCNN'00)* Como, Italy, July 24-27, 2000, vol. 1, pp. 132–137.

Ismail, A. & Engelbrecht, A. P. 2002 Pruning product unit neural networks, In: *Neural Networks, 2002, IJCNN'02, Proc. 2002 International Joint Conference on Neural Networks*, May 12–17, 2002, Honolulu, Hawaii, USA. Vol. 1 pp. 257–262.

Janson, D. J. & Frenzel, J. F. 1993 Training product unit neural networks with genetic algorithms. *IEEE Expert* **8**(5), 26–33.

Jayawardena, A. W. 2009 Riverflow prediction with artificial neural networks. In: *Engineering Applications of Neural Networks (EANN)* (ed. in: D. Palmer-Brown, C. Draganova, E. Pimenidis & H. Mouratidis), pp. 463–471. *EANN 2009*, Communications in Computer and Information Science Series (CCIS) 43, Springer-Verlag Berlin Heidelberg, London, UK.

Jayawardena, A. W. & Mahanama, S. P. P. 2002 Meso-scale hydrological modelling: Application to Mekong and Chao Phraya Basins, *J. Hydrol. Engng.* **7**(1), 12–26.

Jayawardena, A. W. & Mak T. W. 2000 Neural network approach to hydrological forecasting: Comparison of 'ROLS' algorithm with other algorithms. In: *Proc. of the International Symposium on Fresh Perspectives on Hydrology and Water Resources in Southeast Asia and the Pacific*, November 21–24, 2000, Christchurch, New Zealand, IHP-V Technical Document in Hydrology No. 7, Unesco Jakarta Office. pp. 58–69.

Jayawardena, A. W., Tian, Y. & Herath, S. 2004 Prediction of daily discharges of lower Mekong River using artificial neural networks. In: *Proc. of the International Conference on Water Sensitive Urban Design: "Cities and Catchments"*, November 22–23, Jakarta, Indonesia, pp. 83–90.

Jirayoot, K. & Al-Soufi, R. 2000 Prediction of daily river discharge by an artificial neural network model. In: *Mekong River Commission. Workshop on Hydrologic and Environmental Modelling in Mekong basin*, September 11–12, Phnom Penh, Cambodia, pp. 149–157.

Karunasingha, D. S. K. & Liong, S.- Y. 2006a Chaotic time series prediction with a global model: Artificial Neural Network. *J. Hydrol.* **323**, 92–105.

Karunasingha, D. S. K. & Liong, S.- Y. 2006b A comparison of support vector machines and artificial neural networks in hydrological/meteorological time series prediction. *Adv. Geosci* **6**, 91–96.

Kirkpatrik, S., Jr. Gelatt, C. D. & Vecchi, M. P. 1983 Optimization by simulated annealing. *Science* **220**(4598), 671–680.

Krishnakumar, K. 1989 Micro-genetic algorithms for stationary and non-stationary function optimization. In: *SPIE Proc. On Intelligent Control and Adaptive Systems*, SPIE, Philadelphia, PA 1196. pp. 289–296.

Leahy, P., Kiely, G. & Corcoran, G. 2008 Structural optimization and input selection of an artificial neural network for river level prediction. *J. Hydrol.* **355**(1), 192–201.

Leerink L. R., Horne, B. G., Giles, C. L. & Jabri, M. A. 1995 Learning with product units. In: *Advances in Neural Information Processing Systems 7*. MIT Press, *Cambridge, MA*. pp. 537–544.

Manusthiparom, C. & Apirumanekul, C. 2005 Flood forecasting and river monitoring system in the Mekong River basin. In: *Proc. of the 2nd Southeast Asia Water Forum*, August 29th–September 3rd, 2005, Bali, Indonesia: www.mrcmekong.org/download/papers/floodForecast_RiverMonitorSys.pdf.

Martinez-Estudillo, A., Martinez-Estudillo, F., Hervas-Martinez, C. & Garcia-Pedrajas, N. 2006 Evolutionary product unit based neural networks for regression. *Neural Networks* **19**, 477–486.

Mekong River Commission 1998 *Lower Mekong Hydrological Yearbook* (Also, personal communication in 2010).

Otten, R. H. J. M. & Ginneken, L. P. P. P. 1989 *The Annealing Algorithm*. Kluwer, Boston, MA.

Schmitt, M. 2001 On the complexity of computing and learning with multiplicative neural networks. *Neural Comput.* **14**, 241–301.

Srinivas, N. & Deb, K. 1994 Multiple objective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.* **2**(2), 221–248.

Tian, Y. 2007 Macro-scale flow modeling of the Mekong River with spatial variance. PhD Thesis, University of Hong Kong.

First received 15 November 2009; accepted in revised form 11 August 2010. Available online 22 December 2010