Prediction Of Water Consumed Pattern Using Time Series Data (Recurrent Nueral Network) In Water Test Bed Grid (Network).

P. Vasanth Sena, Sammulal Porika, M.Venu Gopalachari
Mr. P.Vasantha Sena, IT department, CBIT, Hyderabad, INDIA. Email: vasanthmtechsit521@gmail.com
Professor Sammulal Porika, CSE department, JNTU CEJ, Hyderabad, India. Email: sammulalporika@gmail.com
Dr.M.Venu Gopalachari, Assoc Professor, IT department, CBIT, Hyderabad, INDIA. Email:mvenugopalachari_it@cbit.ac.in

Abstract. There are several factors that make forecasting a hydrologic time series a difficult endeavour, including a wide range of data, the lack of accurate data, and a lack of enough data. It has recently become common practise to use artificial neural networks (ANNs) for time series forecasting in numerous industries. Forecasting river flows using artificial neural networks is demonstrated in this research. A feed forward network and a recurrent neural network have been selected for the experiment. The recurrent neural network is trained utilising the method of ordered partial derivatives, while the feed forward neural network is taught with the usual back propagation approach. Both networks' architectures and training methods are described in detail. ANN models were used to train and estimate monthly flows of an Indian river with a catchment area of 5189 km2 up to the gauging station using the models that were chosen for this task Both single-step and multiple-step forecasts may be made using the trained networks. A comparison of the two networks reveals that feed forward networks were outperformed by recurrent neural networks. In addition, recurrent neural networks had smaller architectures and took less time to train. For both single-step and multiple-step forecasting, the recurrent neural network performed better. The use of recurrent neural networks in river flow forecasting is therefore strongly advocated.
Key words: forecasting, hydrologic time series, recurrent neural networks, river flows

Introduction

In order to make a good decision, you must first consider the context in which the decision was made. It is possible to make a better decision if you are able to forecast the uncontrolled components of these occurrences before making the decision. When making a choice, forecasting helps to minimise uncertainty. Several water resources initiatives, such as dam building, reservoir operation, flood control, and wastewater disposal, need data on stream flow at various points of interest. ARIMA (Auto Regressive Integrated Moving Average) models developed by Box and Jenkins (1976) are the most often utilised stochastic models for river flow forecasting (e.g. Mujumdar and Nagesh Kumar, 1990). ASCE (2000a) and Govindaraju and Rao provide excellent introductions to the basic ideas of artificial neural networks (ANNs) (2000). ASCE provides a comprehensive collection of references to ANN applications in hydrology (2000b). Kang et al. (1993) predicted daily and hourly streamflows in a river basin in Korea using ANNs and autoregressive moving average models On the basis of data from stream gauge stations 30 kilometres upstream and 20 kilometres downstream of the sample site, Karunanithi et al. (1994) calculated streamflows at an ungauged location. To estimate monthly streamflows at a gauging station in Southern Colorado, Markus et al. (1995) employed neural networks using the back-propagation method and inputs such as snow water equivalent and temperature. According to Raman and Sunilkumar (1995), they used artificial neural networks (ANNs) to simulate multivariate monthly hydrologic time series. ANNs were utilised by Thirumalaiah and Deo (1998) to predict river level. Hydrology and allied areas have made use of recurrent neural networks (RNNs). An RNN model was utilised by Gong et al. (1996) for storm event solid transport in sewage systems In order to estimate rainfall in Hong Kong, Chow et al. (1997) built a recurrent sigma-P neural network model. Predicting watershed runoff is easier using RNNs than other ANN designs, according to Anamala and Co. (2000).

Even though we don't know exactly how an ANN works, we've seen promising preliminary outcomes. In addition, they provide the following benefits:

Neural networks do not need prior understanding of how a system works.

When conducting an examination of a process, it is not necessary to know everything about it.

- ANNs are data-driven contrast to traditional techniques, which are model-driven.

1.1. WORKING OF AN ARTIFICIAL NEURAL NETWORK

In most ANNs, there are three distinct levels of complexity (Vemuri, 1988). Input neurons are found in the top layer of the brain, and they are responsible for receiving information from the outside world. There may be one or more hidden layers above the input layer, with the higher levels getting information from the lower layers. This is the layer directly adjacent to the input layer. The output layer is the final layer of the network. The network's output is provided by the neurons at this layer. From other artificial neurons or from the outside environment, artificial neurons (ANs) obtain their information. An activation function's argument is the weighted sum of these inputs.. It is presumed that this function is nonlinear. The most commonly employed non linearities are hard limiting, such as the step or signum function threshold, and soft limiting, such as the sigmoidal. The activation function's output is the AN's output. Weighted connections are used to distribute this output to other ANs. A computer's memory is akin to a person's ability to establish a weight limit. ANNs are sometimes known as connectionist models of computing because of the importance of these weighted connections.

1.2. ARTIFICIAL NEURAL NETWORKS MODELS

A neural network's architecture, nodes, and training and learning rules all go into its construction. In order to optimise performance, these rules provide an initial set of weights and describe how those weights should be adjusted. Currently, there is a great deal of research being conducted on both the design and training algorithms. For pattern recognition, the Hopfield model was the first to be suggested in the field of artificial neural networks. The input and output layers of the single layer perceptron (Rosenblatt, 1959) are directly coupled. Between the input and output nodes, multi-layered perceptrons (Rosenblatt, 1959) are feed forward networks. There are hidden units or nodes in the extra levels that aren't connected to the input or output layers directly. Although multi-layered perceptrons have overcome many of the drawbacks of single-layered perceptrons, the lack of training algorithms prevented them from being widely employed in the past. The nonlinearities inside the nodes are what give multi-layered perceptrons their powers. More difficult issues may be handled using the network as the number of layers (or hidden layer nodes) between input and output nodes rises. It was invented by Fahlman and Lebiere, and it is a highly effective training method (1990). The training technique of this algorithm incorporates the notion of incremental architecture and learning. An input and output layer are all that's needed to begin training. The network keeps growing until the problem is solved. As a result, the cascade correlation method automatically builds an appropriate architecture for a particular issue (Karunanithi et al., 1994). ANN models just recently began to incorporate recurrent neural networks.

2. Backpropagation Algorithm

The most widely used ANN training algorithm is backpropagation (Rumelhart et al., 1986). Fundamentally, the BPA is a process for developing feed-forward models (FFMs). These models are known as FFMs, and they only allow outputs to be passed to the next layers. Figure 1 depicts a typical feed forward network. Carling (1995) divided architecture selection into three steps: (1) fixing the design, (2) preparing the network for use, and (3) testing the network. Documentation of these stages and the process of training the network is readily available (ASCE, 2000a). The standard Backpropagation method has been improved several times in the literature, and some of these changes are detailed in the next section and applied in this work.

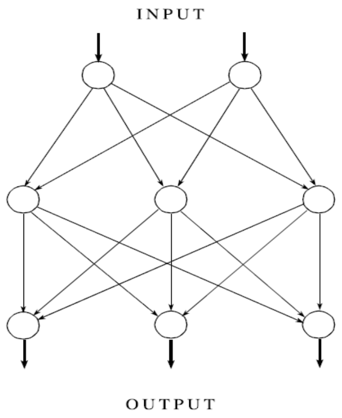


Figure 1. Typical Feed Forward Neural Network.

2.1. IMPROVING BPA TRAINING USING OPTIMUM LEARNING RATE AND MOMENTUM

In order to enhance the BPA algorithm, this strategy involves altering the learning rate and momentum terms (Yu and Chen, 1997). After each iteration, new values for the learning rate and momentum rate are determined based on the error for each pattern. Differentiating the output values for each pattern with regard to the learning rate and the momentum yields the best possible values.

$$\mu_{opt} = \frac{\sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \mu}\right)^t * \Delta_p * \sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \beta}\right)^t * \Delta_p * \sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)}{\sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \mu}\right|^2 * \sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p} \left[\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)\right]^2} \tag{1}$$

$$\beta_{opt} = \frac{\sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \beta}\right)^t * \Delta_p * \sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \mu}\right|^2 - \sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \mu}\right)^t * \Delta_p * \sum_{p=1}^{N_p} \left(\frac{\partial fout_p}{\partial \beta}\right)^t * \left(\frac{\partial fout_p}{\partial \mu}\right)}{\sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \mu}\right|^2 * \sum_{p=1}^{N_p} \left|\frac{\partial fout_p}{\partial \beta}\right|^2 - \sum_{p=1}^{N_p} \left[\left(\frac{\partial fout_p}{\partial \mu}\right)^t * \left(\frac{\partial fout_p}{\partial \beta}\right)\right]^2} \tag{2}$$

where
_*p* = (*aoutp* – *foutp*) taking $\mu = 0$ and $\beta = 0$;
aoutp = the actual output expected from the network for the pattern *p*;
foutp = the computed final output from the network for the pattern *p*;
Np = the number of patterns;
μ = the learning rate generally in the range of 0 to 1;
β = the momentum.

3. Recurrent Neural Networks

Depending on the amount of persistence components in a hydrologic time series, forecasting is dependent on the prior values (memory). This capability is provided by recurrent neural networks (RNNs), which include several feed-back loops. Any of the layers can supply input to or receive output from a generalised RNN in either direction. Consequently, not only do the network's outputs depend on its own internal state but also on that of its preceding time step, as well. Static neural networks may incorporate'memory' in a variety of ways (ASCE, 2000a). These are (in descending order of increasing difficulty and capability):
There is explicit access to previous inputs in tap delay line models, allowing the network to decide its response in a certain moment in time (Mozer, 1989).
Partial recursive models, which are also known as context models or context models, maintain the previous output of nodes rather than the past raw inputs. In these models, all nodes have full feedback and connections with each other (Almeida, 1988).

The current study employs context models. Recurrent backpropagation is briefly described by Islam and Kothari (2000), who cover both the mathematical principles and the practical specifics.

3.1. ADVANTAGES OF RNNs OVER FFNs

The RNNs have crucial qualities, such as the ability to store and utilise data from previous calculations. Complex outputs can be generated from basic inputs that change over time. Despite the lack of any safeguards, it has been discovered that RNNs seldom settle in local minima (Carling, 1995).

3.2. TRAINING METHODOLOGY IN RNNs

RNNs can be trained in the same way as FFN models are. Using a basic example, we'll go over the training algorithm. Figure 2 depicts a simple network with just two input neurons, a hidden layer with three neurons, and a single output neuron. An additional neuron collecting input from the output layer and linked to the hidden layer has been included. Additional neurons are added to RNNs by this neuron. Initial weights are tiny and seemingly random. Just like with the FFN, the inputs are supplied to the input units, and the network's output is calculated. All of the designs are subjected to the same procedure. A approach known as ordered partial derivatives is used to discover the source of the system fault and then train the network using the steepest gradient method.

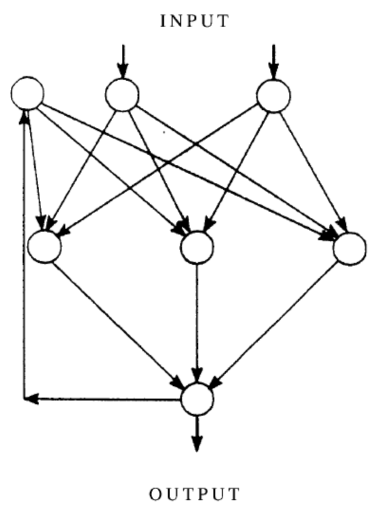
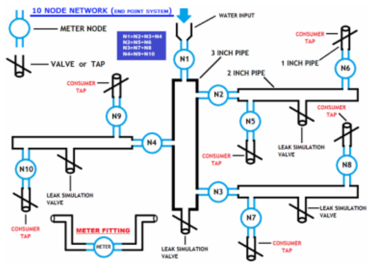
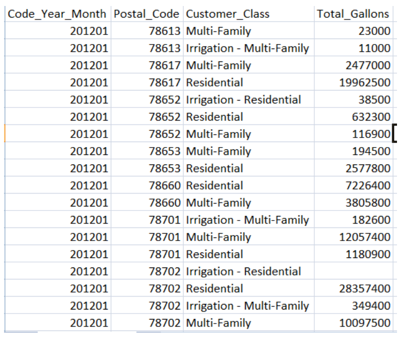


Figure 2. Typical Recurrent Neural Network.

Prototype smart meter network
constructed on Test bed in the lab:





Code_Year_Month	Postal_Code	Customer_Class	Total_Gallons
201201	78613	Multi-Family	23000
201201	78613	Irrigation - Multi-Family	11000
201201	78617	Multi-Family	2477000
201201	78617	Residential	19962500
201201	78652	Irrigation - Residential	38500
201201	78652	Residential	632300
201201	78652	Multi-Family	116900
201201	78653	Multi-Family	194500
201201	78653	Residential	2577800
201201	78660	Residential	7226400
201201	78660	Multi-Family	3805800
201201	78701	Irrigation - Multi-Family	182600
201201	78701	Multi-Family	12057400
201201	78701	Residential	1180900
201201	78702	Irrigation - Residential	
201201	78702	Residential	28357400
201201	78702	Irrigation - Multi-Family	349400
201201	78702	Multi-Family	10097500

Smart water meter generated generates sine wave, collecting data and processing.

$\sin(x) = \sum_{k=0}^{\infty} (-1)^k x^{2k+1} / (2k+1)!$------------(1)

The equation (1) can be expanded like the below.

= x - (1/3!)x3 + (1/5!)x5 - (1/7!)x7 and so on

The input for the equation is expecting radians only. The given degrees are converted to radian by multiplying the given input x with pi value 3.14 divided by 180, and then these input passed to the above equation and the expecting results lies between [0,1] both are inclusive.

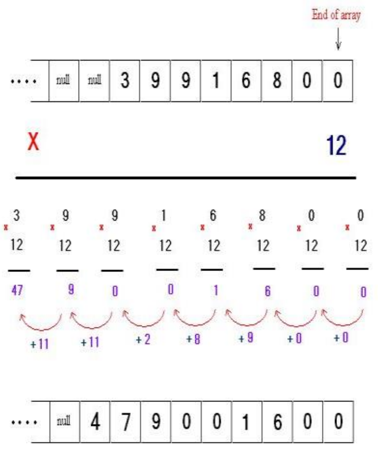
1.1 Optimal method

We know to calculate factorial of a number; you should multiply all numbers from 1 to itself, for example:

1000!=1*2*3* -------999*1000

1000! = 1*2*3*……*998*999*1000

There is no variable in any programming language that can store the result of this multiplication exactly; except storing in scientific notation. Because of sequential multiplication's huge result, finding a new strategy or mechanism is necessary.
In my recommended algorithm, the number is not looked at as a number, instead it is treated as sequential digits where each one has its own numerical value. In fact, an array of digits is available. Each time, the second number is multiplied by each digit of the first number (index of array) and then added with carry number up from the previous multiplication.
This process is shown, for example 12!:12!=12*11! But we know11!=39916800.



1.2 Algorithm for Optimal method

```
Procedure Optimal_Facrorial()
{
MAX  :=100;
Integer  I,j,c,ia[MAX],tot,coeft,;
        Ia[MAX-1] :=1;
WRITE  “Enter number to find the factorial”;
READ n
FOR  j:=0 to MAX-1 do step by 1
        Ia[j]=0;
END FOR
        Coeft:=0;
FOR  c:=2 to n do step   by 1
        I:=MAX-1;
        WHILE(i>0)
        BEGIN
                Tot:=ia[i]*c+coeft;
                IF(tot>9)
                        Ia[i]:=tot%10;
                        Coeft:=tot%10;
                ELSE
                Ia[i]:=tot;
                Coeft:=0;
                END IF
        I:=i-1;
        ENDWHILE
ENDFOR
FOR c:=2 to MAX-1
WRITE ia[c];
ENDFOR
END Procedure
```

1. Data Engineering

In order to apply feature engineering in data Science, the performance of the model is depending on data preprocessing and data handling. Suppose if we build a model without Handling data, we got an accuracy of around 70%. By applying the Feature engineering on the same model there is a chance to increase the performance from 70% to more. Simply, by using Feature Engineering we improve the performance of the model.

The main aim of feature selection is nothing but a selection of required independent features. Selecting the important independent features which have more relation with the dependent feature will help to build a good model. There are some methods for feature selection:
Correlation Matrix with Heatmap is a graphical representation of 2D (two-dimensional) data. Each data value represents in a matrix. Firstly, plot the pair plot between all independent features and dependent features. It will give the relation between dependent and independent features. The relation between the independent feature and the dependent feature is less than 0.2 then choose that independent feature for building a model.

In this Univariate Selection model, Statistical tests can be used to select the independent features which have the strongest relationship with the dependent feature. SelectKBest() method can be used with a suite of different statistical tests to select a specific number of features. Which feature has the highest score will be more related to the dependent feature and choose those features for the model.
the ExtraTreesClassifier method will help to give the importance of each independent feature with a dependent feature. Feature importance will give you a score for each feature of your data, the higher the score more important or relevant to the feature towards your output variable.

Handling Missing Values:

In some datasets, we got the NA values in features. It is nothing but missing data. By handling this type of data there are many ways:In the missing value places, to replace the missing values with mean or median to numerical data and for categorical data with mode. Drop NA values entire rows through df.dropna(how='all) method. Drop NA values entire features. (it helps if NA values are more than 50% in a feature) Replace NA values with 0 through the method df.fillna(0) utility. If you choose to drop options, there is a chance to lose important information from them. So better to choose to replace options.

Handling imbalanced data:

We need to handle imbalanced data, because of to reduce over fitting and under fitting problem. suppose a feature has a factor level2(0 and 1). The imbalanced data consists of true's is ten percentage and false values is ninety percentage. It is called imbalanced data.
Under-sampling the majority class will resample the majority class points in the data to make them equal to the minority class. Oversampling minority class will resample the minority class points in the data to make them equal to the majority class.Over Sampling minority class using Synthetic Minority Oversampling Technique (SMOTE)
In this method, synthetic samples are generated for the minority class and equal to the majority class.

Handling outliers

firstly, calculate the skewness of the features and check whether they are positively skewed, negatively skewed, or normally skewed. Another method is to plot the boxplot to features and check if any values are out of bounds or not. if there, they are called outliers.

how to handle these outliers: –
first, calculate quantile values at 25% and 75%.

- next, calculate the Interquartile range

IQR = Q3 – Q1

- next, calculate the upper extreme and lower extreme values

lower extreme=Q1 – 1.5 * IQR
upper extreme=Q3– 1.5 * IQRe

- lastly, check the values will lie above the upper extreme or below the lower extreme. if it presents then remove them or replace them with mean, median, or any quantile values.

Binning

Binning is nothing but any data value within the range is made to fit into the bin. It is important in your data exploration activity. We typically use it to transform continuous variables into discrete ones. Suppose if we have AGE feature in continuous and we need to divide the age into groups as a feature then it will be useful.

Encoding:

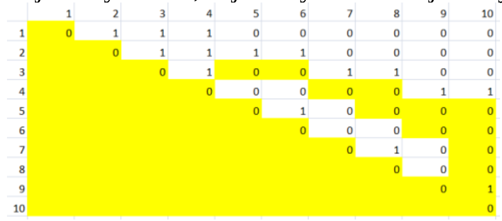
Why this will apply? because in datasets we may contain object datatypes. for building a model we need to have all features are in integer datatypes. hence and One Hot Encoder and Label Encoder are used to convert object datatype to integer datatype.

- Label Encoding

Before applying Label Encoding the height values are tall, medium and short.
After applying label encoding then apply the column transformer method to convert labels to 0,1and 2.one Hot **Encoding:** By applying get_dummies we convert directly categorical to numerical
Feature scaling: we need to apply this approach, because to reduce the variance effect and to overcome the fitting problem. there are two types of scaling methods:Standardization: When this method is used?.when all features are having high values, not 0 and 1. It is a technique to standardize the independent features that present in a fixed range to bring all values to the same magnitudes. Z=X-µ/sigma, in standardization, the mean of the independent features is 0 and the standard deviation is 1.After encoding feature labels are in 0 and 1. This may affect standardization. To overcome this, we use Normalization.
Normalization also makes the training process less sensitive by the scale of the features. This results in getting better coefficients after training. In Min Max Scaler the norm is calculated using thisformula: Norm(x) is equal to x-min(x)/ max(x)-min(n). It is a method to rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.

2. Matrix representation

Adjacency matrix, adjacency list and adjacency set



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2		0	1	1	1	1	0	0	0	0
3			0	1	0	0	1	1	0	0
4				0	0	0	0	0	1	1
5					0	1	0	0	0	0
6						0	0	0	0	0
7							0	1	0	0
8								0	0	0
9									0	1
10										0

In graph theory, an adjacency matrix is a dense way of describing the finite graph structure. It is the 2D matrix that is used to map the association between the graph nodes. If a graph has **n** number of vertices, then the adjacency matrix of that graph is **n x n**, and each entry of the matrix represents the number of edges from one vertex to another. An adjacency matrix is also called as **connection matrix**. Sometimes it is also called a **Vertex matrix**.

Adjacency Matrix Representation

If an Undirected Graph G consists of n vertices then the adjacency matrix of a graph is n x n matrix A = [aij] and defined by -

$a_{ij} = 1$ {if there is a path exists from V_i to V_j}
$a_{ij} = 0$ {Otherwise}

Let's see some of the important points with respect to the adjacency matrix.

- If there exists an edge between vertex V_i and V_j, where i is a row, and j is a column, then the value of $a_{ij} = 1$.
- If there is no edge between vertex V_i and V_j, then the value of $a_{ij} = 0$.
- If there are no self loops in the simple graph, then the vertex matrix (or adjacency matrix) should have 0s in the diagonal.
- An adjacency matrix is symmetric for an undirected graph. It specifies that the value in the i^{th} row and j^{th} column is equal to the value in j^{th} row i^{th}
- If the adjacency matrix is multiplied by itself, and if there is a non-zero value is present at the i^{th} row and j^{th} column, then there is the route from V_i to V_j with a length equivalent to 2. The non-zero value in the adjacency matrix represents that the number of distinct paths is present.

How to create an adjacency matrix?

Suppose there is a Graph **g** with **n** number of vertices, then the vertex matrix (or adjacency matrix) is given by -

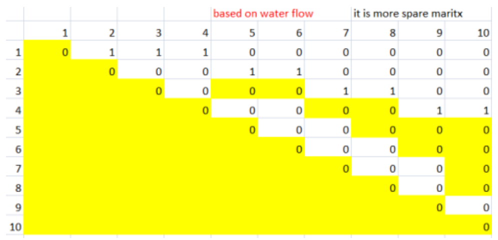
$$A = \begin{array}{lll} a_{11} & a_{12} \; . \; . \; . \; . \; . & a_{1n} \\ a_{21} & a_{22} \; . \; . \; . \; . \; . & a_{2n} \\ . & . & . \\ . & . & . \\ . & . & . \\ a_{n1} & a_{n2} \; . \; . \; . \; . \; . & a_{nn} \end{array}$$

Where the a_{ij} equals the number of edges from the vertex i to j. As mentioned above, the Adjacency matrix is symmetric for an undirected graph, so for an undirected graph, $a_{ij} = a_{ji}$.
When the graphs are simple and there are no weights on the edges or multiple edges, then the entries of the adjacency matrix will be 0 and 1. If there are no self-loops, then the diagonal entries of the adjacency matrix will be 0.

Adjacency matrix for an undirected graph

In an undirected graph, edges are not associated with the directions with them. In an undirected graph, if there is an edge exists between Vertex A and Vertex B, then the vertices can be transferred from A to B as well as B to A. Let us consider the below-undirected graph and try to construct the adjacency matrix of it.
In the graph, we can see there is no self-loop, so the diagonal entries of the adjacent matrix will be 0. In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called the initial node, while node B is called the terminal node.

- An adjacency matrix is a matrix that contains rows and columns used to represent a simple labeled graph with the numbers 0 and 1 in the position of (V_I, V_j), according to the condition of whether or not the two V_i and V_j are adjacent.
- For a directed graph, if there is an edge exists between vertex i or V_i to Vertex j or V_j, then the value of $A[V_i][V_j] = 1$, otherwise the value will be 0.
- For an undirected graph, if there is an edge that exists between vertex i or V_i to Vertex j or V_j, then the value of $A[V_i][V_j] = 1$ and $A[V_j][V_i] = 1$, otherwise the value will be 0.



					based on water flow			it is more spare maritx		
	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2		0	0	0	1	1	0	0	0	0
3			0	0	0	0	1	1	0	0
4				0	0	0	0	0	1	1
5					0	0	0	0	0	0
6						0	0	0	0	0
7							0	0	0	0
8								0	0	0
9									0	0
10										0

3. **Distance vector routing algorithm and application of test bed network.**

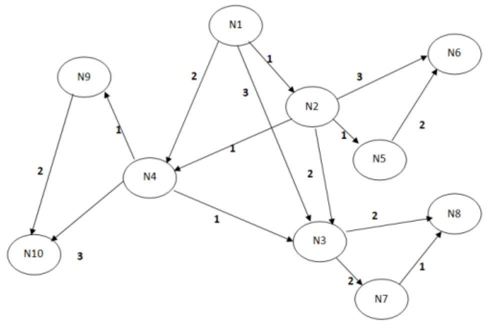
Algorithm
At each node x,
Initialization

```
for all destinations y in N:
$D_x(y) = c(x,y)$     // If y is not a neighbor then $c(x,y) = \infty$
for each neighbor w
$D_w(y) = ?$     for all destination y in N.
for each neighbor w
send distance vector $D_x = [\ D_x(y)\ : y \text{ in } N\ ]$ to w
loop
  wait(until I receive any distance vector from some neighbor w)
  for each y in N:
  $D_x(y) = \min_v\{c(x,v)+D_v(y)\}$
If $D_x(y)$ is changed for any destination y
Send distance vector $D_x = [\ D_x(y) : y \text{ in } N\ ]$ to all neighbors
forever
```

4. **Eccentricity of the Graph and Prime nodes and centric points.**

Highest eccentricity nodes becomes to prime nodes, it need to be take more care about that smart water meter nodes form external factors.

Graph representation of Smart water meter Grid:



Eccentricity of N10: 3
Cost of min path from n4 to n10 is: 3
Cost of min path from n9 to n10 is: 2

Eccentricity of N9: 1
Cost of min path from n4 to n9 is: 1

Eccentricity of N8: 3
Cost of min path from n3 to n8 is: 3
Cost of min path from n7 to n8 is: 2

Eccentricity of N7: 1
Cost of min path from n3 to n7 is: 1

Eccentricity of N6: 3
Cost of min path from n2 to n6 is: 3
Cost of min path from n5 to n6 is: 2
Eccentricity of N5: 1
Cost of min path from n2 to n5 is: 1

Eccentricity of N4: 2
Cost of min path from n1 to n4 is: 2
Cost of min path from n2 to n4 is: 1

Eccentricity of N3: 3
Cost of min path from n1 to n3 is: 3
Cost of min path from n2 to n3 is: 2
Cost of min path from n4 to n3 is: 1

Eccentricity of N2: 1
Cost of min path from n1 to n2 is: 1

Model Application

4.1. TESTING WITH SYNTHETIC DATA

First, an ARMA model is used to evaluate FFNs and RNNs using synthetically created data. Mujumdar and Nagesh Kumar determined that an ARMA (1,2) was the best model for monthly flow data from the Hemavathi River using the maximum likelihood criteria (1990). Synthetic generation made use of this model and its associated parameters. Synthetic data will allow the ANN models to be tested without the noise or observational mistakes that may be present in the actual data. Seven years of testing are involved in this process (84 months). Comparison between FFN predictions with synthetic data is depicted in Figure 3. Figure 4 shows RNN predictions compared to synthetic data. When comparing ANN predictions to real data, the root mean square error (RMSE) and R2 values are utilised as statistics. Figures 3 and 4 illustrate the statistics for FFN and RNN predictions, respectively. These two graphs show that RNNs are substantially better at preserving monthly data variance than FFNs.

4.2. CASE STUDY

In Karnataka state, India, the monthly Hemavathi flow has been projected using the FFN and RNN models. An Indian tributary that flows into the Cauvery River is called Hemavathi. Akkihebbal in Karnataka state is the location of the gauging site. This gauging point is located in a 5189-square-mile catchment area. From 1916–17 through 1972–1973, monthly flow data from the location, spanning 57 years, is available. Monthly flow fluctuates from 1.84 M.cu.m. in the summer months to 2894.81 M.cu.m. in the monsoon months, depending on the season's rainfall. Table I shows the monthly average and standard deviation for the 57-year data set. Data from the first 50 years is utilised for training, while data from the latter 7 years is used for testing.

.

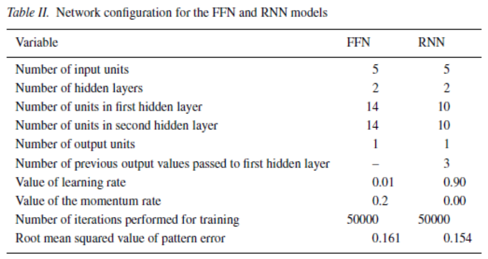


Table II. Network configuration for the FFN and RNN models

Variable	FFN	RNN
Number of input units	5	5
Number of hidden layers	2	2
Number of units in first hidden layer	14	10
Number of units in second hidden layer	14	10
Number of output units	1	1
Number of previous output values passed to first hidden layer	–	3
Value of learning rate	0.01	0.90
Value of the momentum rate	0.2	0.00
Number of iterations performed for training	50000	50000
Root mean squared value of pattern error	0.161	0.154

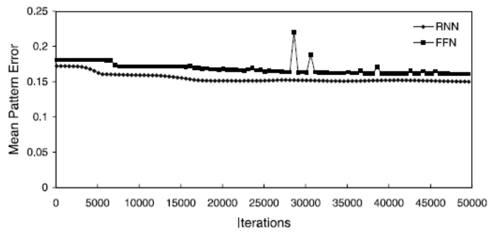


Figure 5. Variation of mean pattern error for RNN and FFN with number of iterations.

Conclusions

Compared to more traditional methods, neural networks have significant advantages. What matters most is their capacity for developing a generalised solution to an issue from a given collection of instances and for incorporating additional variations in the problem into that answer. It is because of this quality of generalisation that they may be used to solve a wide range of issues and still come up with viable answers despite data or description mistakes. Feed Forward Neural Network (FFNN) and Recurrent Neural Network (RNN) were used to predict monthly streamflows of the Hemavathi River in India. When it came to predicting monthly river flows, RNNs outperformed other methods. In order to lessen the effects of extreme weather events like flooding and droughts on human activities, it is necessary to monitor the environmental impact of hydrology and build real-time forecasting models. This project is expected to help achieve this goal.

References

Almeida, L. B.: 1988, 'Backpropagation in Perceptrons with Feedback', R. Eckmiller and Ch. Von der Malsburg (eds), *Neural Computers,* Springer Verlag, Berlin, pp. 199–208.
Anmala, J., Zhang, B. and Govindaraju, R. S.: 2000, 'Comparison of ANNs and empirical approaches for predicting watershed runoff', *J. Water Resour. Plann. Manage., ASCE* **126**(3), 156–166.
ASCE Task Committee on Application of Artificial Neural Networks in Hydrology: 2000a, 'Artificial neural networks in hydrology. I: Preliminary concepts', *J. Hydrol. Engineer. ASCE* **5**(2), 115– 123.
ASCE Task Committee on Application of Artificial Neural Networks in Hydrology: 2000b, 'Artificial neural networks in hydrology. II: Hydrologic applications', *J. Hydrol. Engineer., ASCE* **5**(2), 124–137.

Bishop, C.M.: 1995, *Neural Networks for Pattern Recognition*, Oxford University Press, New York. Box, G. E. P. and Jenkins, G. M.: 1976, *Time Series Analysis: Forecasting and Control*, Holden-Day Publications, San Francisco, U.S.A.

Carling, Alison: 1995, *Introduction to Neural Networks*, Galgotia Publications, New Delhi. Chow, T. W. S. and Cho, S. Y.: 1997, 'Development of a recurrent sigma-P neural network rainfall forecasting system in Hong Kong', *Neural Comput. Appl.* **5**(2), 66–75.

Fahlman, S. E. and Lebiere, C.: 1990, 'The Cascaded-Correlation Learning Architecture', *Rep. CMU-CS-90-100*, Carnegie Melon University, Pittsburgh.

Gong, N., Denoeux, T. and Bertrand Krajewski, J. L.: 1996, 'Neural networks for solid transport modeling in sewer systems during storm events', *Water Sci. Technol.* **33**(9), 85–92.

Govindaraju, R. S. and Rao, A. R. (eds): 2000, *Artificial Neural Networks in Hydrology*, Kluwer Academic Publishers, Amsterdam.

Islam, S. and Kothari, R.: 2000, 'Artificial neural networks in remote sensing of hydrologic processes', *J. Hydrol. Engineer., ASCE* **5**(2), 138–144.

Kang, K. W., Kim, J. H., Park, C. Y. and Ham, K. J.: 1993, 'Evaluation of Hydrological Forecasting System Based on Neural Network Model', *Proc., 25th Congress of Int. Assoc. for Hydr. Res*., International Association for Hydraulic Research, Delft, The Netherlands, pp. 257–264.

Karunanithi, N., Grenney, W. J., Whitley, D. and Bovee K.: 1994, 'Neural networks for river flow forecasting', *J. Comput. Civil Engineer., ASCE* **8**(2), 201–218.

Kothari, R. and Agyepong, K.: 1997, 'Induced Specialization of Context Units for Temporal Pattern Recognition and Reproduction', J. Principe, L. Gile, N.Morgan and E.Wilson (eds), *Proc., IEEE Neural Networks for signal Processing VII*, Institute of Electrical and Electronics Engineers, New York, pp. 131–140.

Magoulas, G. D.: 1997, 'Effective back propagation training with variable step size', *Neural Networks* **10**(1), 69–82.

Markus, M., Salas, J. D. and Shin, H.-K.: 1995, 'Predicting Streamflows Based on Neural Networks', *Proc., 1st Int. Conf. on Water Resour. Engrg*., ASCE, New York, pp. 1641–1646.

Mozer, M. C. and Smolensky, P.: 1989, 'Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment', in D. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann, San Monteo, California, pp. 107–115.

Mujumdar, P. P. and Nagesh Kumar, D.: 1990, 'Stochastic models of streamflow – Some case studies', *Hydrol. Sci. J.* **35**, 395–410.

Piche, S. W.: 1994, 'Steepest descent algorithms for neural net controllers and filters', *IEEE Transactions on Neural Networks* **5**(2), 198–212.

Raman, H., and Sunilkumar, N.: 1995, 'Multi-variate modeling of water resources time series using artificial neural networks', *Hydrol. Sci. J.* **40**, 145–163.

Rumelhart, D. E, Hinton, G. E. and Williams, R. J.: 1986, 'Learning internal representation by backpropagating errors', *Nature* **323**, 533–536.

Rosenblatt, R.: 1959, *Principles of Neuro Dynamics*, Spartan Books, New York.

Salomon, R. and Hemmen, J. L. V.: 1996, 'Accelerating back propagation through dynamic selfadaptation', *Neural Networks* **9**(4), 589–601.

Sperduti, A. and Starita, A.: 1993, 'Speed up learning and network optimization with extended back propagation', *Neural Networks* **6**, 365–383.

Thirumalaiah, K. and Deo, M. C.: 1998, 'River stage forecasting using artificial neural networks', *J. Hydrol. Engineer., ASCE* **3**(1), 26–32.

Vemuri, V.: 1988, 'Artificial Neural Networks: An Introduction', in *Artificial Neural Networks: Theoretical Concepts*, The Computer Society of the IEEE, pp. 1–12.

Yu, X.-H. and Chen, G.-A.: 1997, 'Efficient back propagation learning using optimal learning rate and momentum', *Neural Networks* **10**(3), 517–527.