Undergraduate Research Project

CE491A – Under-Graduate Research I

UNDER guidance of DR. ASHU JAIN

Submitted by – SHUBHAM MITTAL (13687)

DETAILED ANALYSIS OF RAINFALL RUNOFF NEURAL NETWORK MODELLING

Shubham Mittal

ABSTRACT

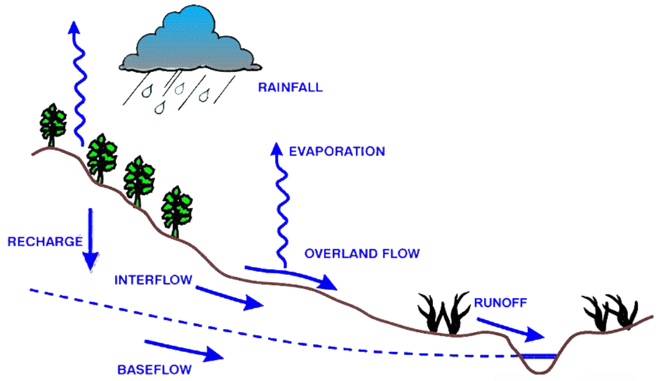
Artificial Neural Networks (ANN) is a tool of predictive analysis which can learn complex non-linear processes. One such complex process, rainfall-runoff (RR) modelling in Hydrology has an objective to predict runoff for a catchment and help to build safety and utility systems for water, a scarce resource. The purpose of this study is two-fold. Foremost one is to provide a detailed analysis of comparison between conventional conceptual models like AWBM & TANK and relatively new modelling methodology like Multi-Layer Perceptron Neural Networks (MLPNN). Second objective is to evaluate the performance of recently published deep learning optimization algorithm.

Keywords – *Rainfall- Runoff (RR) Modelling, Artificial Neural Network (ANN), AWBM Model, TANK Model, Conceptual Runoff Models.*

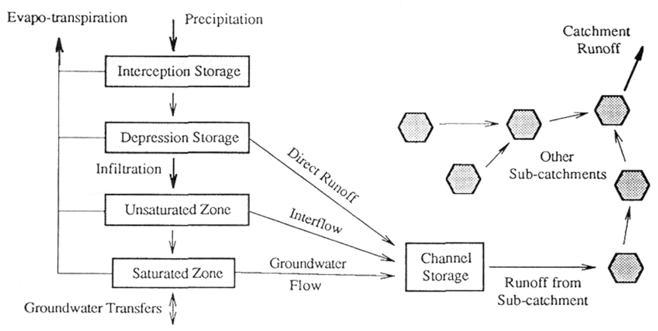
I. INTRODUCTION

Rainfall-Runoff is a hydrologic process which can be arguably considered as one of the most researched topic in the field of hydrology modelling. Different methodologies have been applied in an attempt to constantly improve the modelling of this highly complex and implicit but yet deterministic chaotic system [1]. Researchers and engineers have tried both physical & mathematical modelling with both having advantages and disadvantages of their own. Modelling of this non-linear, dynamic (time-variant) & continuous process continually becomes a topic of research as soon as new modelling methodologies in scientific community immersed. An objective of designing better hydraulic systems like culverts, dams, at al. depends on the estimation of runoff values for a catchment. Few of the most widely applied techniques to estimate runoff values are physical modelling and mathematical modelling. As described in fig.1, in physical modelling, an output is directly estimated from analogues or theoretical simulations of the real processes while mathematical modelling manipulates the hidden interrelation (linear or non-linear) of dependent and independent variables with the help of various mathematical concepts to predict the regression function. Each modelling process has been scrutinized and analyzed thoroughly to increase their reliability and interpretability for real world and real time risk-averse hydraulic systems.

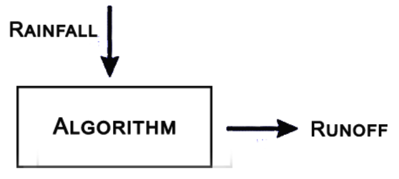
Rainfall-runoff modelling, a non-linear process estimates runoff or streamflow for a stream (leaf), river or catchment with the use of different modelling tools & techniques as can be seen in fig.1a. The relationship between rainfall and runoff is very important. This is due mainly to the fact that rainfall data are commonly used in flood forecasting and good forecasting methods may be obtained once the appropriate relationship (with relevant values of parameters estimated) is established. Moreover, since rainfall data are normally available for a longer period than runoff, this availability can be used for filling in missing values of runoff, or in extending (most frequently backward) runoff records. For these purposes, the above relationship is very useful [2]. However, the physical process involved in RR modelling comprises an arguable level of uncertainty (because of its chaotic nature i.e., large number of variable interdependency makes RR modelling more complex) due to which existing modelling techniques and results are often questioned and improved. This uncertainty can be attributed to human developments and the changes they bring to this hydrologic process. To mitigate this uncertainty as well as improve the accuracy of modelling results, researchers continuously work with new theories and modelling research. One such theory is Neural Networks, a concept designed to imitate the functioning of human brain. The cardinal concept was originally developed by Warren McCulloch and Walter Pitts during 1943 [3]. They created a simple computation model for processing data and extracting the unobservable (by human mind) insights. However, neural network research stagnated after machine learning research by Minsky and Papert (1969), [4] who discovered two major key issues with computational machines that processed neural networks. The foremost was that basic perceptrons (developed by McCulloch and Pitts) were incapable of processing the exclusive-or circuit. The second and the main reason was stagnation was that computers didn't have enough processing power to effectively handle large neural networks. Because of these reasons, neural network research slowed until computers achieved far greater processing power. When such power was achieved, neural networks research picked the pace and they also got recognized in different fields for computational and modelling purposes because of its unparalleled accuracy levels and quick deployment nature. A core contribution to this outburst can be acknowledged to non-SMEs (Subject Matter Experts) in both industries and academia as NNs were easy to model and didn't require detailed and in-depth process functioning. This soft computing methodology, Neural networks, combines different theories from computer science, mathematics, and cognitive science and is one of the most popular modelling technique of this decade with applications in many fields of social science & natural science. The framework used for Neural network RR modelling is illustrated in fig.1b. The input parameters (rainfall) are feed to a black-box model which estimate the regression function between response (runoff) and input. The main advantage of Neural networks over other conventional modelling methods is its ability to easily map non-linear functional processes. Hence, it is only reasonable to apply this revolutionary technique for the hydrologic process.



(a) Rainfall-Runoff Process



(b) Physical Model



(c) Neural Network Model (Black-Box model)

Fig.1. Rainfall-Runoff Process and Models

Currently, variants of neural networks have been and are being developed improving on various limitations of earlier ones like Multi-Layer Perceptrons Neural Networks (MLPNN). Different variants of neural networks suitable for modelling temporal data like rainfall-runoff are Recurrent Neural Networks (RNNs), Long Short Term Memory (LSTMs), et al. These methods are relatively new in rainfall-runoff modelling and are currently a part of ongoing research. This study, however, concerns with MLPNN with an overall objective of determining best possible models for three different catchments. Different parameters and functionalities (architecture) of MLPNN are exploited here to achieve the overall objective of this study. The results also present a comparative analytical study of neural networks for different spatial locations. The earliest study for developing a neural network model is dated back to early 1990s. The application of ANNs in RR modelling started with preliminary study by Halff et al. (1993) who used a three layer feed-forward ANN for the prediction of hydrographs. Since then, Karunanithi et al. (1994), Hsu et al. (1995), Smith & Eli (1995), Minns & Hall (1996), Sajikumar et al. (1999), Ehrman et al. (2000), Birikundavyi et al. (2002), Jain & Indurthy (2003), Jain & Kumar (2007), Narain & Jain (2010) among many others developed ANN RR models using collected data from real catchments [5]. All these studies clearly demonstrate that ANN RR models are powerful tools to forecast runoff in large catchments which is quite difficult for conceptual models because of chaotic nature of rainfall runoff process.

In addition, this study also serves as a comparison between conventional physical models and latest mathematical models like MLPNN. Physical or conceptual models replicate the real process mathematically. An illustration of a physical (conceptual or direct) model can be seen in fig. 1.c. These models are based on conceptual theories of rainfall-runoff cyclic process where different new factors like evaporation, surface flow, interflow, baseflow, infiltration are also considered for evaluating final runoff in sub-catchments of a catchments and finally combining them for catchment runoff. For this study, physical models TANK and AWBM (to be described later) are used for modelling rainfall runoff process and its results are compared with ANN models developed for same catchments. Fig.2 shows different modelling methods in both category from which models (with bold text) are chosen for this study. However, there are other methods available outside author's limited knowledge and should be explored as per requirement.

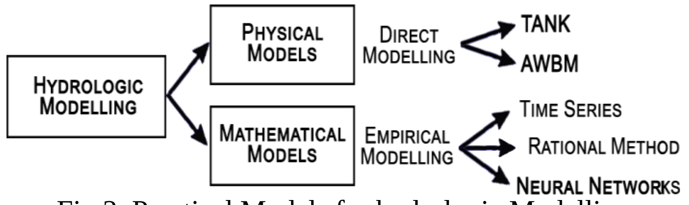


Fig.2. Practical Models for hydrologic Modelling

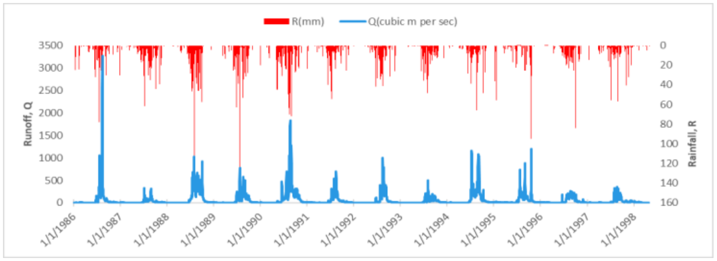
II. MODEL DEVELOPMENT

A. STUDY AREA AND DATA:

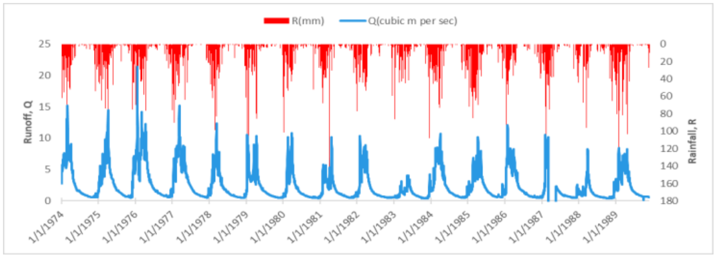
Data from catchment of Godaveri, and Jardine rivers were employed for this study. A brief overview of three catchments and data used is provided here. Godaveri River is India's second largest with an average discharge of 3,505 cubic meter per sec and its river delta supports 729 persons/km^2 twice the density of India [6]. The daily streamflow (m^3/sec) data chosen for this study is taken from parts of Godaveri's main tributaries Purna, Pravara, Manjira & Manair. Daily rainfall (mm), maximum (°C), and minimum temperature (°C) data from 4 gauge stations at Medak, Nizambad, Hanmakonda & Ramagundam is collected. The daily total catchment rainfall in mm is aggregated using simple mean. Sequences of rainfall and runoff comprises of total 4503 data points which encompasses 1st January 1986 to 30th April 1998 time period. After calibrating missing rainfall values using spatial relationship between different stations, rainfall series was missing two months of time steps which were handled during training and testing division. Daily maximum and minimum temperature of 4 stations were used to evaluate daily potential evapotranspiration (mm) which was an input parameter for physical models. The method used was Hargreaves ET_o equation [7].

The daily streamflow (or runoff) (m^3/s) data at Telegraph line (gauging station number 927001), the average of daily rainfall (mm) data from five rain gauges located at Bamaga, Cape York Post Office, Eliot Falls, Jardine Monument, and Peak point stations distributed all through the catchment, daily evapotranspiration (mm) data from Jardine River near Far North Queensland, Australia for a 16-year period (1/1/1974–27/11/1989), 5810 days, was considered for this study. The catchment occupies an area of 2,500 square km. [8]. Runoff time series was missing 78 continuous values. The data was collected from Rainfall Runoff Library (RRL) [9] default datasets. The Rainfall Runoff Library (RRL) is designed to simulate catchment runoff using daily rainfall and evapotranspiration data. RRL is used to implement conceptual physical models for RR modelling.

The statistics of study data is provided in Table 1 and graphical representation of rainfall and runoff series for catchments can be observed in fig.3. The study data is divided into training and testing data (as mentioned above) with lengths such as statistical characteristics of training data were shadowing or atleast similar to that of testing data to avoid erroneous result as it is intuitively evident that a statistically different or overpowered test data will result in over-fitting (or generalization) issues. Data pre-processing and preparation of lagged series brought the final data count of Godaveri and Jardine data sets to 4442 and 5731 respectively.



a. Godaveri



b. Jardine

Fig. 3. Rainfall and Runoff Series of Catchments

Table 1: Statistical Analysis of Catchment Data

Variables	Rainfall (mm)		Runoff (cubic m per sec)	
	Training	Testing	Training	Testing
	Godaveri Data			
Count	3000	1442	3000	1442
Min.	0.0	0.0	0.362	0.272
Max.	139.8	95.1	3270.6	1195.1
Mean	2.87	2.08	61.93	55.34
SD	8.97	7.17	169.96	133.17
	Jardine Data			
Count	3500	2231	3500	2231
Min.	0.0	0.0	0.45	0.18
Max.	162.6	135.8	21.41	12.09
Mean	4.63	4.32	2.59	2.11
SD	12.3	11.76	2.64	2.16

B. MODEL PERFORMANCE:

The performance of the models developed in this study was evaluated using different standard statistical measures. The employed error statistics and their detailed description is given below:

1. **Mean Absolute Error, MAE**:

MAE has a clear interpretation as the average absolute difference between two variables predicted runoff (Q_p) and observed runoff (Q_o). It is calculated as simple mean of total absolute difference over time steps N.

$$MAE = (1/N)\sum |Q_o(t)-Q_p(t)|$$

Where, $Q_o(t)$ is observed output at time t,

$Q_p(t)$ is predicted output at time t,
N is total number of time steps for which MAE is calculated.

As obvious, lower values of MAE depict a better model.

2. **Average Absolute Relative Error, %AARE:**

AARE metric is useful in measuring the relative error of a variable's single observation. AARE prevents large observations to misrepresent the performance of a model. It is calculated by taking mean of relative error for a data point.

$$AARE = (1/N)\sum |Q_o(t)-Q_p(t)|*100/Q_o(t)$$

Where, terms are similar as mentioned above.

Similarly to MAE, lower values represent better model compare to large values.

3. **Root Mean Square Error, RMSE**:

It is a standard measure of error in many function approximation problems like RR modelling. However, RMSE is prone to give bias results towards high magnitude differences between observed and predicted variable due to difference's square in the numerator as can be seen below.

$$RMSE = \{(1/N)\sum [Q_o(t)-Q_p(t)]^2\}^{1/2}$$

AARE is a better indicator than RMSE as RMSE is biased towards large values of a variable to be evaluated.

4. **Nash-Sutcliffe Efficiency, E**:

Nash-Sutcliffe Coefficient of Efficiency compares predicted and observed values and evaluates the efficiency of network to explain the variance of data. Its value varies from $-\infty$ to 1 with 1 being the best and value closer to $-\infty$ as worst and value just less than 0 means that mean of a variable (naïve model) is a better predictor than the model. It is calculated as,

$$E = (E_1-E_2)/E_1$$
$$E_1 = \sum [Q_o(t)- (1/N)\sum Q_o(t)]^2$$
$$E_2 = \sum [Q_o(t)-Q_p(t)]^2$$

As compared to above error metrics, higher values of E are preferable for better model performance.

5. **Pearson Coefficient of Correlation, R**:

Correlation coefficient, R, measures the strength of linear correlation between observed and predicted output. R is calculated as covariance (Cov) between observed and predicted outputs divided by standard deviations (SD) of both predicted and observed outputs. Its values ranges from -1 to +1 with value close to 1 as best and close to 0 as worst model performance. -1 represent a perfect negative relationship between two variables but in RR modelling, values always lie between 0 and 1. Mathematically, R is calculated as:

$$Cov = \sum \{[Q_o(t)- (1/N)\sum Q_o(t)]*[Q_p(t)- (1/N)\sum Q_p(t)]\}$$
$$SD_o = \{\sum [Q_o(t)- (1/N)\sum Q_o(t)]^2\}^{1/2}$$
$$SD_p = \{\sum [Q_p(t)- (1/N)\sum Q_p(t)]^2\}^{1/2}$$

$$R = Cov / (SD_o * SD_p)$$

Where, SD_o is standard deviation for observed output,
SD_p is standard deviation for predicted output.

6. **Normalized Mean Bias Error, NMBE %**:

NMBE indicates the overall bias of the model i.e., whether the model is overestimating or underestimating the output variable. It is measured by calculating mean error of output and dividing it by mean of observed output. The mathematical expression for MBE is:

$$NMBE = [(1/N)\sum (Q_o(t)-Q_p(t)]*100/[(1/N)\sum Q_o(t)]$$

Positive value of NMBE indicates overall overestimation while negative value mean overall underestimation by the model in question.

7. **Threshold Statistics, TS_x**:

All the above mentioned error statistics give us an idea of overall performance of a given model. But, usually, for a better interpretation of a model at different level of Absolute relative error (ARE) is needed. Threshold Statistic (TS) evaluates the percentage of output values forecasted below a certain level of ARE (say, x %). Here, ARE is calculated as

$$ARE (\%) = |Q_o(t)-Q_p(t)|*100/Q_o(t)$$

Hence,

$$TS_x = (n_x/N)*100$$

Where, n_x is number of data points forecasted below x% ARE,
ARE dependent terms and remaining ones can be referred in above error measures

Intuitively, large % of TS_x at small x represent a good model performance.

8. **Relative Error in Maximum Flow, %MF**:

The relative error in maximum value of output variable provides information about the over or under-estimation in predicting the maximum value of output variable. It is computed by following expression.

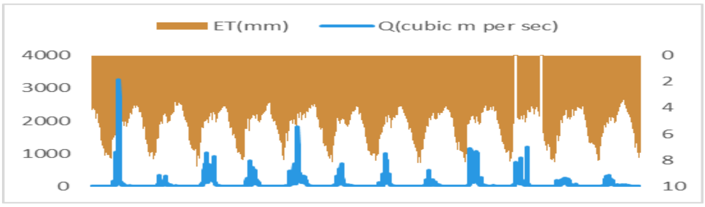
$$\%MF = Q_o(max)-Q_p(max)*100/Q_o(max)$$

Where, $Q_o(max)$ is maximum value of output variable,
$Q_p(max)$ is the predicted value of maximum value of output variable.

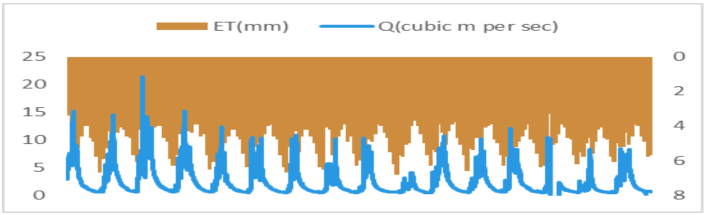
Minimum value closer to 0 in any case (positive or negative) means a very good model performance.

C. CONCEPTUAL MODEL DEVELOPMENT:

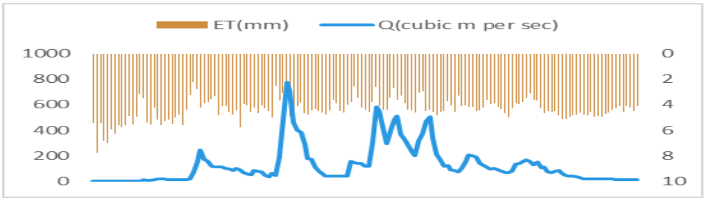
A **conceptual model** is representation of a system (physical or man-made), made of composition of various concepts (theories about the system) which are used to simulate the system the model represents (Conceptual Model, Wikipedia). Rainfall-Runoff hydrologic process, a system of various observationally developed concepts, had been simulated by different conceptual models in literature. In a conceptual model developed here, input parameters (rainfall & potential evapotranspiration (PET)) considered to be interrelated to output variable are converted to output (runoff) with the help of concepts engaging these variables with each other. Rainfall dependency can be seen from fig.3. Fig.4 shows how evapotranspiration (ET) has a seasonal as well as positive relationship with runoff (Q). During each year, just before runoff peaks (for both catchments), evaporation peaks can be observed which can infer to a lagged relationship of Q(t) with ET(t-k) where t-k is k days before tth day. Fig. 4 also have a small timeframe's (1/6/1989 to 31/10/1989) graph, where the inference can be held valid.



a. Godaveri



b. Jardine



c. Subset of Jardine

Fig.4. Evapotranspiration Series of Catchments

Among few conceptual models, AWBM and TANK are used in this study for benchmarking the performance to be compared by ANN models. These concept of these conceptual models is briefly described here.

AWBM:

The Australian Water Balance Model (AWBM) is a catchment water balance model that relates daily rainfall and evapotranspiration to runoff, and calculates losses from rainfall for flood hydrograph modelling. It was originally developed by W.J. Boughton [10]. The input variables for AWBM are daily rainfall and potential evapotranspiration (PET). Compared to rainfall, evapotranspiration has little influence on the water balance at a daily time scale and thus areal potential evapotranspiration (calculated using Hargreaves ET_o equation) is used (Boughton & Chiew 2003). Structure of AWBM rainfall-runoff model is illustrated in fig.5. The AWBM model uses three surface stores to simulate partial areas of runoff, each representing user defined land use or soil classifications as proportions of the area of the catchment. The water balance of each surface store is calculated independently of the others. At each time step, in the model, rainfall is added to each of the three surface moisture stores and evapotranspiration is subtracted from each store.

$$\mathbf{Store_n = Store_n + rain\text{-}evaporation\ (n=1,\ 2,\ 3)}$$

If the value of moisture in the store becomes negative, the moisture content of the store is set to zero, as the Evapotranspiration demand is higher than the available moisture. If the value of moisture in the store exceeds the capacity of the store, the excess moisture is counted as runoff and the moisture content of the store is set to capacity.

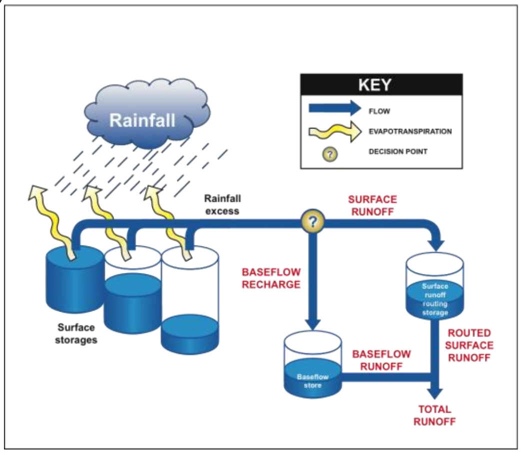


Fig.5. Structure of AWBM rainfall-runoff model [https://wiki.ewater.org.au/display/SD41/]

When runoff occurs from any store, part of the runoff becomes recharge of the Baseflow store. Its value being BFI*runoff where BFI is called Base Flow Index. This is defined to be the ratio of Base Flow to Total Flow in the stream flow. The remainder (i.e. (1-BFI)*runoff) is surface runoff.

The baseflow store is depleted at the rate of (1-K)*BS where BS is the current moisture in the base flow store and K is the base flow recession constant of the time step being used. The surface runoff can be routed through a store if required to simulate the delay of surface runoff reaching the outlet of a medium to large catchment. The surface store acts in the same manner as the base flow store and is depleted at the rate of (1-KS)*SS, where SS is the current moisture in the surface runoff store and KS is the surface runoff recession constant of the time step being used. The total runoff is calculated as the sum of routed surface runoff and the base flow. The output may be saved in ML/day, m³/s or mm/day.

TANK Model:

The TANK model (Sugawara, 1961) is also applied to analyze daily discharge from daily precipitation and daily evaporation inputs. It is a simple model consisting of four tanks placed vertically in a series (Fig.6).

In TANK model, precipitation is poured into topmost tank and evaporation is subtracted. This process is carried out subsequently for tanks 2 and 3 moving downwards. As each tank is emptied the evaporation shortfall is taken from the next tank down until all tanks are empty. The outputs from the side outlets (q_{11}, q_{12}, q_2 & q_3) are taken as runoff. Runoff outputs from the top 3 tanks are calculated using a formulation which can be studied in [5]. If the water level is below the outlet no discharge occurs. Runoff from the different tanks are as follows:

1. Top tank: Surface runoff
2. Second tank: Intermediate runoff
3. Third tank: Sub-surface runoff
4. Fourth tank: Base-flow

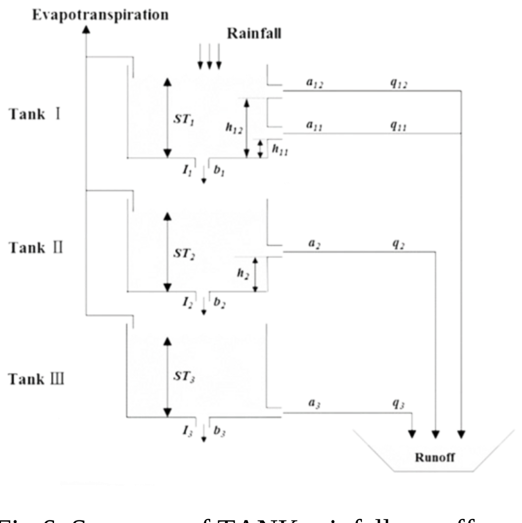


Fig.6. Structure of TANK rainfall-runoff model

The model seems to be simple but its output depends upon various parameters like the content of each store. The volume of each storage tank considerably effects the runoff. The tank model is a non-linear model. Here, instead of daily evaporation as input, daily evapotranspiration was used.

Both AWBM and TANK are easy to apply with the use of a tookit named Rainfall Runoff Library (RRL) (CRC, Corporate Research Centre for Catchment Hydrology). The toolkit can be downloaded from [9].

For both AWBM and TANK, Calibration Optimiser and Objective Function used were Genetic Algorithm and Nash-Sutcliffe criterion (Coefficient of efficiency, E). For conceptual models, Godaveri data set consisting a total of 3467 was used. The calibration was based on: Runoff difference in %. The Jardine data (5810 data points) for conceptual models was divided as 3486 for calibration and 2324 for testing (or verification). Similarly, Godaveri data (3467 data points) was divided with training and verification sets of 2139 and 1328 respectively.

Daily RR modelling results from AWBM and TANK for two catchments Godaveri and Jardine were found out using error statistics as described in section II. The results of AWBM and TANK are analyzed and compared with other models (ANN model) in section III.

D. NEURAL NETOWRK MODEL DEVELOPMENT:

1. ANN MODEL:

An artificial neural network (ANN) is a modelling technique developed to mimic a "brainlike" system of interconnected processing units (called neurons) proposed by McCullock and Pitts in 1943 that learn patterns from past data and predicts for new events. They have vast applications for different purposes in different fields, but are hugely appraised & popular for forecasting, and classification problems. In fig.7, an illustration of a simple three layer feedforward ANN is given. As can be seen, an input layer with multiple neurons feeds information (input variables data) to middle layer and middle layer processes the information to further move it to an output layer which finally produces the output. Now, input layer does not directly feed raw input to the middle layer neurons. Each input layer neuron is connected to every middle layer neuron and these connections provide different weights (w_{11}, et al.) to each input variables before entering into any middle layer neurons. The weights are initialized with random numbers and learned over epochs during training. Most of the research on ANNs justifies that the initialization of weights should be done in such a way that the summation of all weights should be exactly 1. This heuristic make ANN unbiased during training. Middle & output layers, similar to input layer, can also have one or more number of neurons depending upon the process. For RR modelling, output variable is runoff. Similar to input layer, each middle layer neuron is also characterized by a weight (w_{h1O}, w_{h2O}, et al.) mapped to output layer neuron.

These weights are also initialized similar to input layer neurons and are finally learned as training epochs end. Another parameter bias*weight (weights are b_{i1}, b_{i2},….,b_{hO} which are also learned during training) is added into weighted inputs' summation and feed to their respective hidden and output layer neurons before activation function is triggered. Bias neuron of both input and middle layers are taken as 1 which is later multiplied by its learned connection weight to next layer neurons. Unlike input layer neurons, middle layer neurons consist of activation functions (to be discussed later) which process the summation of all weighted inputs from input neurons and bound them. Usually the activation function bound the value (coming) between 0 and 1 to make the processing of ANN fast and efficient. The outgoing activated values from middle layer neurons are again weighted and summed in output layer neuron which may or may not apply activation function to finally produce the output.

A very popular ANN framework, Multi-Layer Perceptron Neural Network (MLPNN) with one or more layers between input and output layer is very widely used for supervised learning of non-linear processes. For RR modelling, it has been found out that one hidden layer ANNs are sufficient for good results as well as quick deployment & interpretability. It can be easily seen that more hidden layers would result in loss of model interpretability and more processing time. To understand the drawbacks of more hidden layers or complexity, one can refer to theory of trade-off between variance and bias of a statistical or mathematical model.

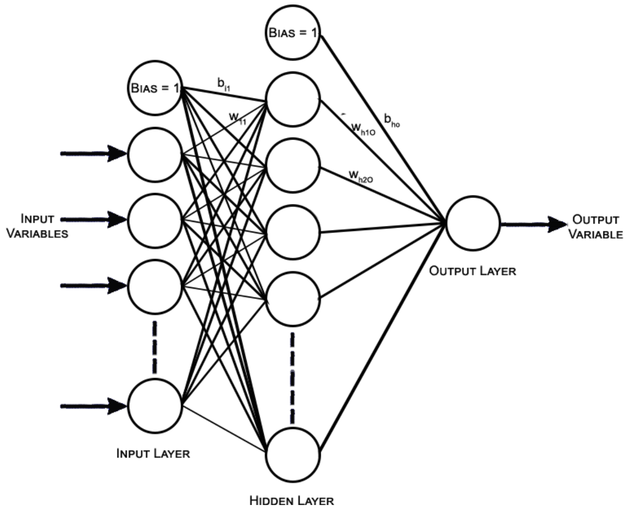


Fig.7. Structure of a feedforward ANN model

For this study, ANN models with one hidden layer of multiple neurons and an output layer of one neuron were developed and compared for performance evaluation. Activation function, ReLU (Rectified Linear Unit, to be discussed later) was equipped in middle layer neurons with a linear activation function in output layer neuron. Linear activation function is a linear regression line (y = x) which gives output same as input. From this, we can infer that output (rainfall) is a linear combination of inputs from hidden layer neurons with weights as parameters.

To decide on number of neurons in hidden layers, various studies on different datasets have been analyzed to observe a pattern and a generalized rule of thumb is now used to limit the number of possibilities to save time and resources. There are many rule-of-thumb methods for determining a good number of neurons to be used in hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, if not sufficient then plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Activation Function:

For deploying activation function at ANN model neurons, there are various functions available. The most widely and popularly used in hydrological modelling (Dawson & Wilby, 2001), Sigmoid function is a non-linear continuous, bounded, non-decreasing, and differentiable. It maintains the output value from a neuron within 0 and 1. The use of such logistic function induces non-linearity in ANN models which makes them a perfect candidate for modelling non-linear chaotic yet deterministic processes like RR hydrological process. Fig.8 shows the sigmoid function. Mathematically, sigmoid function is described as,

$$f(x) = 1/(1+e^{-x})$$

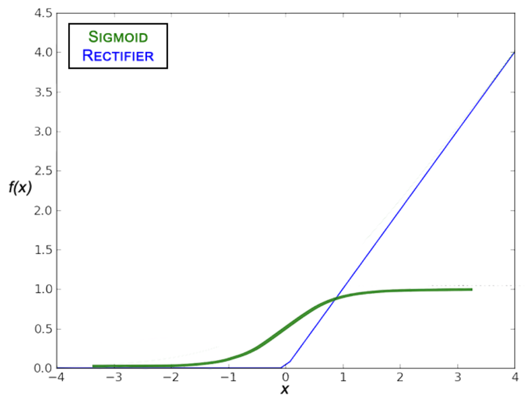


Fig.8. Activation Functions

However, for this study, a relatively new (Hahnloser et al. 2000, [11]) activation function, Rectifier (fig.8) which is widely used in deep neural networks [12] was used**.** It is defined as,

$$f(x) = max(0,x)$$

Rectifier is non-differentiable at 0 (differentiable at all other values) which is a disadvantage in learning of the ANN model but can be managed by manipulating input variables to never be 0 (data scaling was done between 0.1 and 0.9 which is good for a rectifier function). A unit employing a rectifier is called Rectified Linear Unit (ReLU). Rectified linear units, compared to sigmoid function or similar activation functions, allow for faster and effective training of deep neural architectures on large and complex datasets. The advantage of rectifier function is that it functions as a regularizer (like ridge or lasso regularizers) to balance between bias and variance as complexity of a model increases.

Representation of ANN model's output:

Now, the final output from a three-layer ANN model with ReLU activation function can be represented as:

$$HI_j = \sum I_i * w_{ij} + 1 * b^I_j$$
$$HO_j = f_H(HI_j) = \max(0, HI_j)$$
$$OI_k = \sum HO_j * w_{jk} + 1 * b^H_k$$
$$O_k = f_O(OI_k) = OI_k$$

Where, $\mathbf{I_i}$ = input to and from ith input layer,
$\mathbf{b^I_j}$ = connection weight of input layer bias and jth hidden layer neuron,
$\mathbf{w_{ij}}$ = connection weight of ith input layer neuron and jth hidden layer neuron,
$\mathbf{HI_j}$ = input to jth hidden layer neuron,
$f_H(\mathbf{HI_j})$ = ReLU activation function deployed on jth hidden layer neuron,
$\mathbf{w_{jk}}$ = connection weight of jth hidden layer neuron and kth output layer neuron,
$\mathbf{b^H_k}$ = connection weight of hidden layer bias and kth output layer neuron,
$\mathbf{OI_k}$ = input to kth output layer neuron,
$f_O(\mathbf{OH_k})$ = linear activation function at kth output layer neuron, and
$\mathbf{O_k}$ = output from kth output layer neuron

Optimization Learning Algorithm:

Until now, we have discussed about the architecture of ANN model. But, ANN models are mostly appraised for its ability to learn highly complex functions which many mathematical or conceptual models fail to map. ANN model maps the input variables with their respective output variable with the help of a learning algorithm which optimizes the error between predicted and observed output to lowest possible value. ANN models starts with random weights and iteratively updates the connection weights through a backpropagation method of iteratively transmitting output error with the help of gradient (total or batch) calculated w.r.t. to individual neuron or bias weight until optimal weights for every connection are found. Various algorithms had been used for finding optimal ANN models in RR modelling. One of the most widely used and superiorly proven optimization method (for RR modelling) is Levenberg-Marquardt (LM)**.** LM algorithm [13] has been constantly used in RR modelling in literature and found out to outperform other backpropagation algorithms like gradient descent and resilient backpropagation.

For this study, a recently published variant of gradient-descent, *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, was used. Adam is based on adaptive estimates of lower-order moments. It is a family member of gradient descent algorithms with few changes like Adam's weight update expression does not contain a learning rate as it is adapted as per the importance of individual parameter during updates [14]. Informally, this unique adaption increases the learning rate for more sparse parameters (dry days) and decreases the learning rate for less sparse ones (wet days). This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters like rainfall are more informative as is the case in RR modelling.

Given weights $w^{(t)}$ and a loss function $L^{(t)}$, where t indexes the current training iteration (indexed at 1), Adam's parameter update is given by:

$$m_w^{(t+1)} = \alpha_1\, m_w^{(t)} + (1-\alpha_1)\, g(L^{(t)})$$
$$\upsilon_w^{(t+1)} = \alpha_2\, \upsilon_w^{(t)} + (1-\alpha_2)\, [g(L^{(t)})]^2$$
$$\dot{m}_w = m_w^{(t+1)}/(1-\alpha_1^t)$$
$$\bar{\upsilon}_w = \upsilon_w^{(t+1)}/\,(1-\alpha_2^t)$$
$$w^{(t+1)} = w^{(t)} - \eta(\dot{m}_w/(\sqrt{\bar{\upsilon}_w}+\varepsilon)$$

Where, ε is a small number used to prevent division by 0, and α_1 and α_2 are the decay rates for first moment ($m_w^{(t)}$) and second moment ($\upsilon_w^{(t)}$) of loss function's gradient ($g(L^{(t)})$), respectively. $\dot{m}_w$ and $\bar{\upsilon}_w$ are bias corrected 1st and 2nd moments respectively as it has been observed that when moments are initialized as vectors of 0s, future iterative values tend to bias towards 0.

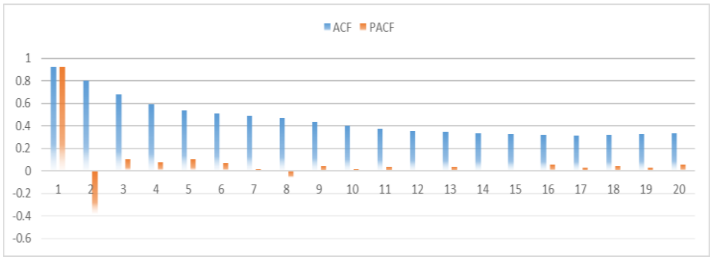
The default parameters (momentum factor η=0.001, α_1=0.9, α_2=0.999, and $\varepsilon=10^{-8}$) are proposed by authors and are considered to work on a broad spectrum of problems. This modified version of gradient-descent algorithm was published at International Conference on Learning Representations (ICLR) [Diederik & Jimmy, 2015] [15]. Although the purpose of this algorithm is to ease the process of solving large datasets and/or high-dimensional parameter spaces machine learning problems [15]. The main advantage of Adam is its robust and well-suitability to a wide range of non-convex optimization problems. More illustrious details on Adam can be found in its original paper. Despite author's recommendation of Adam's hyper parameters, a separate study to optimize the parameters can be performed if required. However, this study was an attempt to assess the Adam's performance on ANN RR modelling compared to conceptual physical models. As a preference, default parameters were taken as suggested by algorithm's authors. The results are presented in section III.

Input Variable Selection:

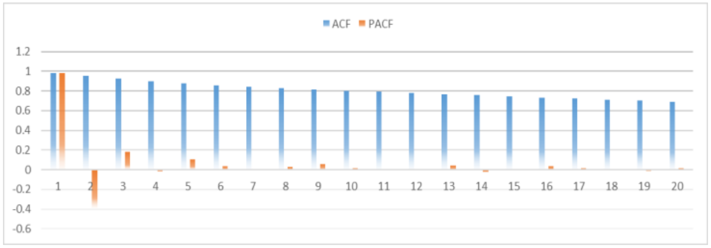
Processes with multiple dependent variables are highly complex which leads to certain disadvantages like loss of model interpretability or overfitting. To counter these disadvantages, feature selection plays a very important role in a modelling process. For feature selection, there are many methods available in literature. For this study, cross-correlation, auto-correlation, and partial auto-correlation analyses were carried out to choose the most influential and relatable input variables (number of input layer neurons). Cross-correlation is calculated similar to coefficient of correlation, R for two different variable's (runoff, Q(t) and rainfall, R(t)) time series. In time series forecasting, autocorrelation function (ACF) and partial autocorrelation function (PACF) quantify the dependency or correlation of a time series observation, Q(t) with its own lagged series, Q(t-1), Q(t-2),…..,Q(t_o). ACF finds the linear dependency (Coefficient of Correlation, R or Pearson Correlation) of Q(t) series on any of lagged Q(t-k) series while PACF calculates the autocorrelation between two series, Q(t) and Q(t-k), after removing Q(t)'s linear dependency on all lagged series between t and t-k. The results for input variables selection can be found below in Table 2 and fig.9.

TABLE 2: Cross-Correlation Analysis

	Runoff, Q(t)	
Rainfall	**Godaveri**	**Jardine**
R(t)	0.256	0.355
R(t-1)	0.364	0.404
R(t-2)	0.447	0.424
R(t-3)	0.455	0.419
R(t-4)	0.438	0.402
R(t-5)	0.403	0.385
R(t-6)	0.358	0.377
R(t-7)	0.338	0.372
R(t-8)	0.324	0.37
R(t-9)	0.31	0.372
R(t-10)	0.3	0.37



a. Godaveri Catchment



b. Jardine Catchment

Fig.9. Auto-Correlation and Partial Auto-Correlation Function (ACF and PACF) Analysis

As can be seen in above graphs, for both catchments, Q(t-1) and Q(t-2) have significant PACF values as compared to others and hence were taken as input variables. From cross-correlation values between Q(t) and R(t-k) for k= 0 to 20, it is difficult to interpret a clear difference in correlation strength for any R(t-k). Following model with given input variables was considered for this study but variants with different input variables might also be considered (outside the scope of this report). Although, a comparison between M1's best configuration with a more input neurons model was held with results in section IV.

Model	**Input Variables**	**Output Variables**
M1	Q(t-1), Q(t-2), R(t), R(t-1), R(t-2)	Q(t)

Data Scaling:

ANNs are prone to give bad performance for different inputs with different scales. In unscaled data, input variables with large magnitude tends to decide the outcome of an ANN model irrespective of its nature of relationship with the output variable. To overcome this precedented issue, the data was scaled between 0.1 and 0.9. For demonstration purpose, an ANN model M2 is developed, each, for raw (unscaled) and scaled data of godaveri catchment. Models are evaluated with error metrics AARE, TS1 & TS25. Rectifier activation function in hidden layer neurons was used and the models (5-9-1 ANN model was used which is determined using mentioned above third heuristic rule to estimate the number of hidden layer neurons) were learned by Adam optimization algorithm. The results for ANN model M1-AR-9 are shown below in Table 3. ANN models with scaled data is clearly a better choice with highly positive results when compared to models with raw data. For any other ANN model developed, scaled data was used.

TABLE 3. ANN Model for Scaled and Raw data:

	Training			**Validation**		
ANN model	**AARE**	**TS1**	**TS25**	**AARE**	**TS1**	**TS25**
M1-AR-9 (unscaled)	26.2	6.48	76.6	16.6	9.26	83.5
M1-AR-9	**1.99**	**74.0**	**99.0**	**1.07**	**82.5**	**99.8**

(scaled)						

Optimal ANN Architecture:

There are many experimentally proven rules for selecting the number of hidden layer neurons but the generalization of those rules to different catchments is not evident and can be verified in literature. To verify that, for this study, a comparative analysis for ANN model M1 with architecture 5-y-1 is conducted where x varies from 1 to 20. For finding the optimal structure (y) of ANN model M1, k-cross validation is used with k=3. For ANN model development procedure, the method proposed and evaluated was Adam ReLU (AR).

The results for ANN models M1-AR-x developed are available in section III where x is no. of hidden layer neurons. Parameter x is chosen by validating different hidden layer neurons ANN models. For validation, 500 data points were taken from training data (leaving 2500 and 3000 training data respectively). The analysis was conducted on both catchments. The hidden layer neurons selection analysis can be observed in section III.

Overfitting and Under-fitting Issues of ANN model:

Often, ANN Models over fit to training data during learning and hence, perform unsatisfactorily on test data. During overfitting, despite having low training errors, models underperform because they loses its generalization capacity. Increment in models' biasness results in decrease of variance (generalization power). Overfitting often results due to either more complexity or excessive training of predictive models. Complexity of models developed here was checked while selecting hidden layer neurons (x). Prevention of overfitting was also ensured by choosing experimentally sound no. of epochs for model training. Epoch analysis was done by an iterative error visualization graphical technique. In addition to above methods, overfitting can also be subjected to number of input variables (neurons) for a model. This is shown by a comparison of M1 and M2. The epoch graph and model comparison can be observed in section III.

III. CASE STUDIES

A. GODAVERI:

In Godaveri catchment, for ANN model M1-AR-x, parameter x was found out by a simple iterative methodology with x = 1 to 20 and the results can be observed in fig.10. For modelling, hyper-parameters used were epochs = 100 and batch size = 10.

From fig.10, M1-AR-x models with x = 4, 5, 9, 15 and 17 clearly performed better in comparison to others on the basis of validation data RMSE. Correlation, R and RMSE values for above x values are listed below in Table 4 for a microscopic numerical analysis. Although model with 15 hidden layer neurons performed better during training compared to others, it failed to produce better results on validation data set. This is because as complexity of a model increases, bias (or generalization capability) also increases (decreases). ANN model with x = 5 was considered best among all M1 ANN models based on results. As x increases after 5, chances of overfitting of training data increased with more lower training RMSE (or higher R) while higher validation RMSE (or lower R). Further, from fig.10, overall trend of training RMSE can be seen as decreasing while its gap with validation RMSE increasing. This accounts to underperformance of more complex models (higher x).

Table 4. Training and Validation errors for M1-AR-x ANN models on Godaveri Catchment:

	Training		Validation	
Hidden Neurons, x	**R**	**RMSE**	**R**	**RMSE**
4	0.958	0.0371	0.924	0.0394
5	0.945	0.0434	**0.934**	0.0392
9	0.975	0.0285	0.930	0.0405
15	**0.987**	**0.0207**	0.928	**0.0387**
17	0.984	0.0232	0.925	0.0403

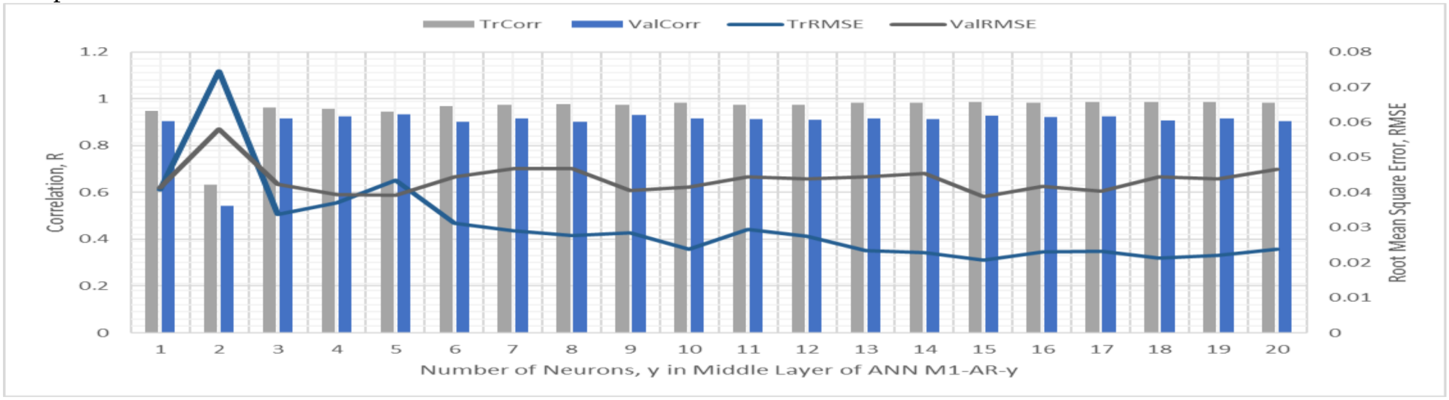


Fig.10. Hidden Layer Neurons Selection for M1-AR-x ANN Modelling in Godaveri Catchment

To validate overfitting by excessive training in M1-AR-5 model, fig.11 shows a graph between epochs and training & testing RMSE after each epoch. Only training (without validation) and testing data was considered for this overfitting validation. Error metric RMSE was calculated for evaluation and from below graph, it can be observed that there are many regions, larger one between 136 and 156, where fluctuations are infinitesimal. Overall, in this case, there is not a clear sign of overfitting during model training for larger epoch value. But as a precaution for overfitting and also to give ample time for model training, epoch value, 100 was considered. An inference for this behavior of M1-AR-5 is explained during epoch analysis of Jardine.

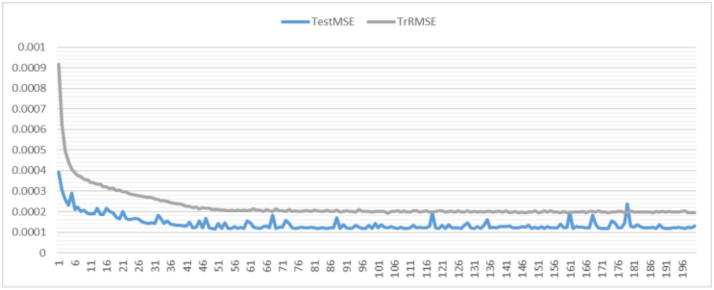


Fig. 11. Graph for different epochs chosen for model M1-AR-5 training vs RMSE (Godaveri catchment)

B. JARDINE:

Similar to Godaveri, best value of x for M1-AR-x ANN model is found out by k cross-validation with k=3. Only training data was used for this calculation of x. In fig.12, bias-variance tradeoff phenomenon of predictive models can be clearly observed as complexity of model (or x) increases, training metrics (R and RMSE), comparatively, perform better than validation. However, at x = 18, there is a sudden increment in validation correlation (ValCorr) which can be attributed to regularization feature of rectifier linear unit (ReLU) activation function. Due to ReLU, as its composition is, a large number of hidden layer neurons must be inactivated and as a result, transmission of input from only highly uncorrelated and influential neurons transmitted must be taking place to output neuron. At lower values of x, a constant improvement in both training and validation metrics can be seen till 4 and then a sudden decadence at x = 5 with further improvement to x = 6. Moving forward from x = 6 to 8, validation performance is not substantial and a precedent decrease can be seen. Now, following the previous trend similar to lower values of x, performance starts improving till x = 12. After x=12, training metrics are showing inverse behavior to validation metrics, a sign of overfitting. Finally, x = 4, 6, 11, 12 and 18 are taken for detailed numerical evaluation in Table.5.

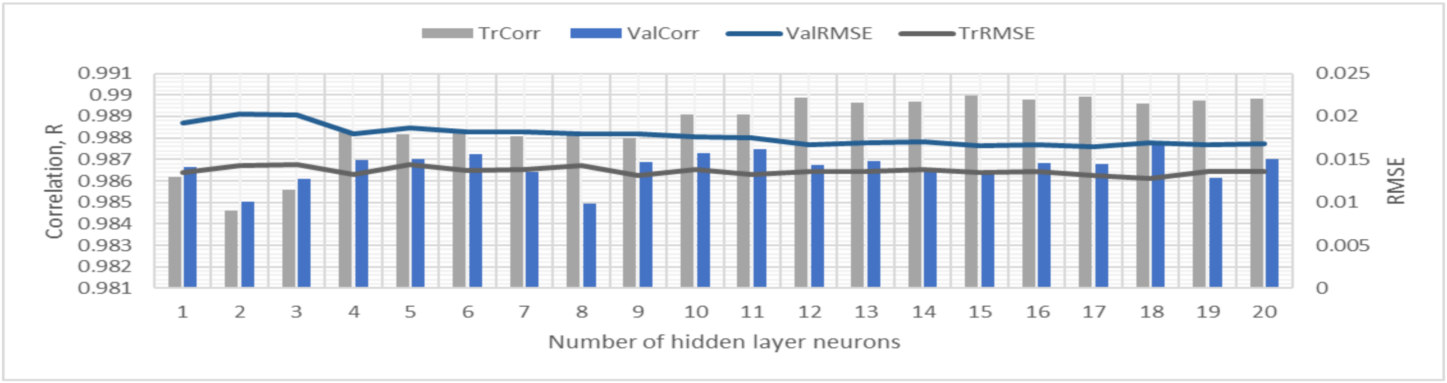


Fig.12. Hidden Layer Neurons Selection for M1-AR-x ANN Modelling in Jardine Catchment

Table 5. Training and Validation performance for M1-AR-x ANN models on Godaveri Catchment:

	Training		Validation	
Hidden Neurons, x	**R**	**RMSE**	**R**	**RMSE**
4	0.98824	0.013193	0.986978	0.01794
6	0.988211	0.013702	0.987266	0.018262
11	0.989098	0.013275	0.987481	0.017547
12	**0.98987**	0.013563	0.986753	**0.016682**
18	0.989608	**0.012806**	**0.987831**	0.016928

For model M1-AR-x, from above table, x=11 was considered the chosen number of hidden layer neurons.

Similar to Godaveri catchment, validation of overfitting by excessive training in M1-AR-5 model is studied. From fig.13, it can be observed that both training and testing RMSE decreases by an order and then remains fluctuating in a very small bound. In this case, there is no sign of overfitting during training. This result of no overfitting during model optimization, in both catchments, can be attributed to Adam optimization algorithm's use of previous moments (gradients) during training. For precautionary purposes, epoch value, 50 was considered to let models learn for a considerate time.

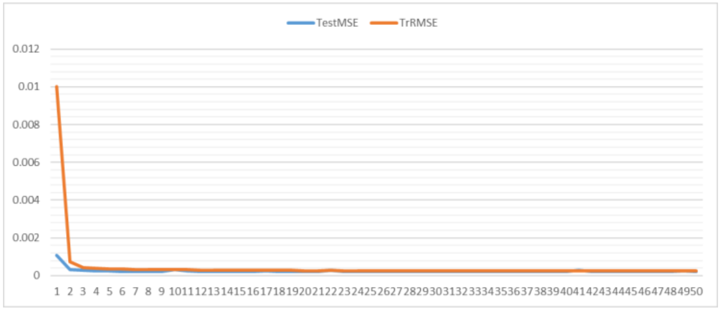


Fig.13. Graph for different epochs chosen for model M1-AR-11 training vs RMSE (Jardine catchment)

IV. RESULTS, DISCUSSION AND CONCLUSION

The results were obtained during training and testing in the form of error statistics, mentioned earlier, for various models developed and are presented in table.6. Starting with conceptual models, AWBM is a better model compared to TANK. AWBM and TANK both have almost similar predictive power. During testing, these two produced similar values of E and R. AWBM comprehensibly outperforms TANK with a comparatively very low AARE and high TS100.

For purpose of assessing more dependent input variables' contribution during RR modelling, another model M2 was developed with similar configurations of M1-AR-x models developed above. This ANN model M2-AR-x has two additional input variables Q(t-3) and R(t-3) with hidden layer neurons 5 (and 11) for Godaveri (and Jardine). M2's results are compiled in table 6 with errors indicating a slight improvement over M1 in case of Godaveri with better MAE, AARE (%), RMSE, NMBE (%), and TS1 (%). Other error statistics are almost similar to be decisive. Distinctively, Jardine showed a decrease in performance when M2 model was used. Apart from that fig.14 shows the scatter plots which indicate a model's goodness of fit performance. M2 have slope closer to 1 compared to that of M1 when testing data of both catchments are predicted. Time Series plots of both M1 and M2 (fig.15) are almost identical. These results can be broadened into two points. One is that RR modelling is a spatial based physical process and requires different models for different locations. Other is that an experimental exploration is required to overcome RR modelling's conservative approach of less complexity. The improved metrics are a clear sign that ANN models with more input variables (M2) can perform better compared to other lower dimensional family models like M1 if configured with properties which can overcome the bias-variance tradeoff of predictive models. Adam and ReLU are few of those functional properties which can help make model training fast (less number of epochs required) and balance the tradeoff for complex models. ReLU's ability to shutoff neurons getting input below a threshold value is beneficial for maintaining parsimony principle. ReLU activation function is, as evident from its author's results, a good parameter for high dimensional processes. It can use sparse parameters (like rainfall) for predictive modelling and produce better results which otherwise would have not improved in case of traditional predictive models like conceptual. In the knowledge of author, there were no descriptive literature available for development of Deep Neural Networks (DNN) for RR modelling. Using more properties like ReLU, a dedicated study can be conducted for a DNN model development and compared with other ANN models which had been proven to perform better.

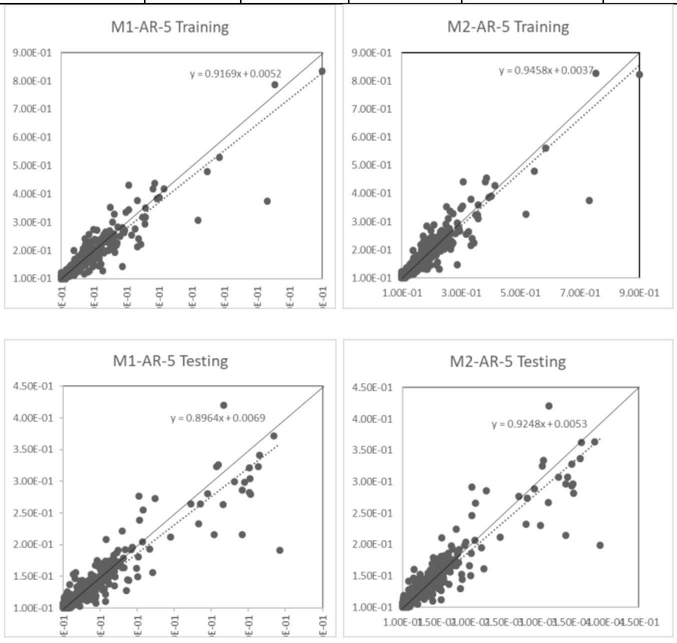
In table given below, it can be evidently seen that ANN models performed better than conceptual models with a very substantial improvement. For both training and testing of each catchment, both ANN models M1 and M2 outperforms AWBM and TANK on basis of almost error metric. Extremely low values of E and R shows the lesser predictive power of conceptual models. Fig.15 shows the time series plots of observed and predicted testing data for different models developed here. These plots clearly shows how much better ANN models are for RR modelling. Both ANN models are far superior in terms of accurately predicting the runoff. Observed and predicted runoff of ANN models are identical while conceptual models were unable to level up with such performance. These unchallenging results of conceptual models are a clear sign for an upward shift in use of ANN models over conceptual models for water utility systems and other purposes. As far as their non-reliability for application purposes is concerned, it can be the result of their parameters' non-interpretability factor.

For future scope, a comparative study of Adam with benchmarked LM and ReLU with different activation functions for RR Neural Networks (NN) is encouraged.

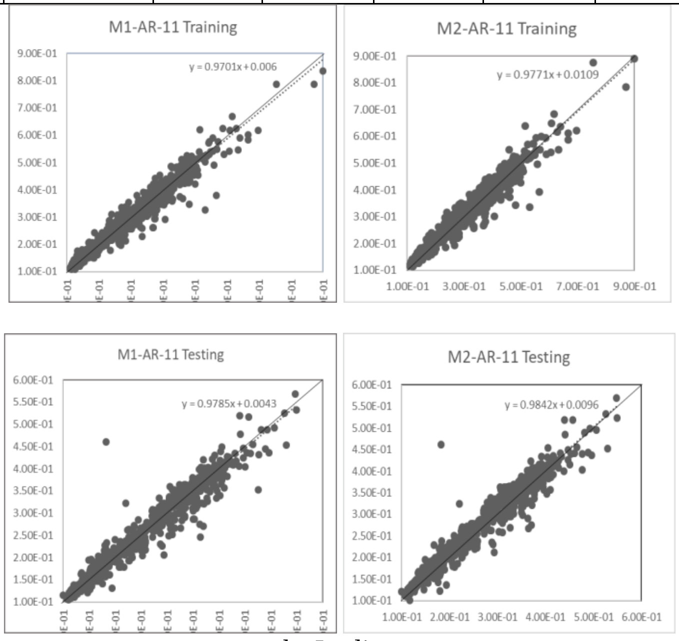
Table 6. Error Statistics of Rainfall-Runoff models

Godaveri Catchment											
Training											
Models	**E**	**R**	**MAE**	**AARE%**	**RMSE**	**NMBE%**	**TS1%**	**TS25**	**TS50**	**TS100**	**MF%**
AWBM	0.429	0.709	-	117.41	5247.35	-2.7	0.25	7.86	15.5	90.94	-47.4
TANK	0.492	0.737	-	208.66	4936.76	-20.4	0.10	7.76	14.0	23.53	-61.1
M1-AR-5	0.897	0.952	0.0074	5.69	0.01443	3.82	5.2	99.0	99.96	100	-7.20

M2-AR-5	0.898	0.951	0.0066	4.96	0.01433	2.25	5.7	99.0	100	100	-8.46
Testing											
AWBM	0.245	0.558	-	100.23	4183.12	64.9	0.288	3.31	7.14	98.13	-45.2
TANK	0.309	0.577	-	179.05	4011.11	29.3	0.216	3.82	8.80	19.32	-32.2
M1-AR-5	0.866	0.942	0.0070	5.74	0.01194	4.26	3.82	99.0	99.93	100	-51.2
M2-AR-5	0.877	0.943	0.0061	4.88	0.01142	2.82	5.42	99.1	100	100	-49.3
Jardine Catchment											
Training											
AWBM	0.762	0.875	-	39.1	1.34	1.27	1.61	37.7	75.10	95.03	-18.2
TANK	0.741	0.861	-	34.7	1.38	5.25	1.92	49.6	80.58	95.78	16.8
M1-AR-11	0.979	0.989	0.0065	2.44	0.015	-0.04	54.4	99.6	100	100	-7.05
M2-AR-11	0.975	0.989	0.0108	5.39	0.016	-3.24	4.13	99.6	100	100	-1.04
Testing											
AWBM	0.717	0.875	-	38.9	1.26	8.82	1.36	36.0	70.27	96.54	1.99
TANK	0.754	0.875	-	37.7	1.12	13.1	1.82	40.0	70.13	95.89	-17.8
M1-AR-11	0.971	0.985	0.0058	2.51	0.014	-0.36	57	99.5	99.96	99.96	-2.82
M2-AR-11	0.965	0.986	0.0101	5.74	0.015	-3.96	2.65	99.6	100	100	-4.54

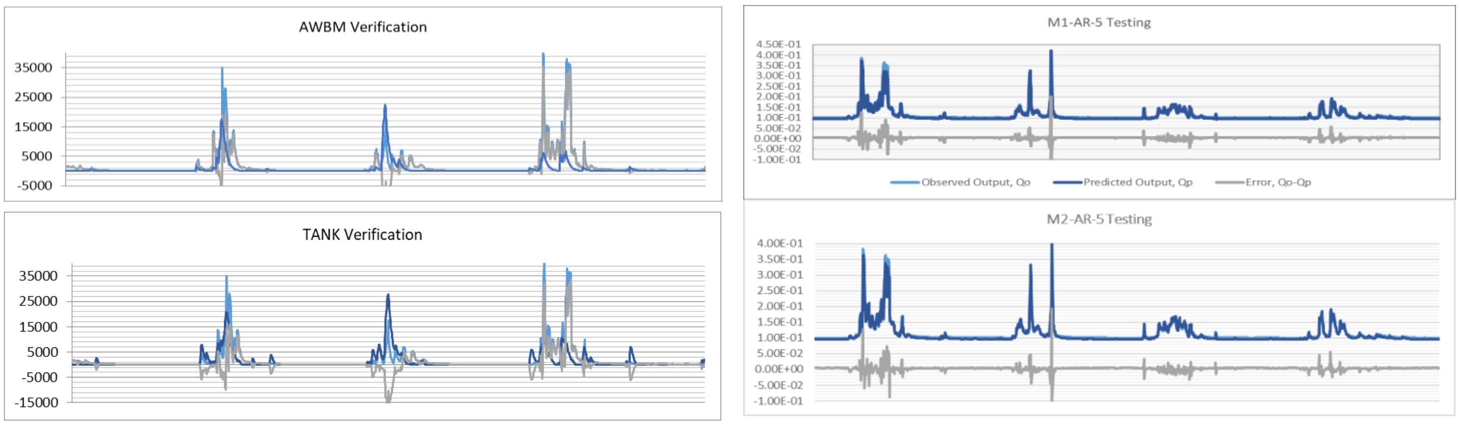


a. Godaveri

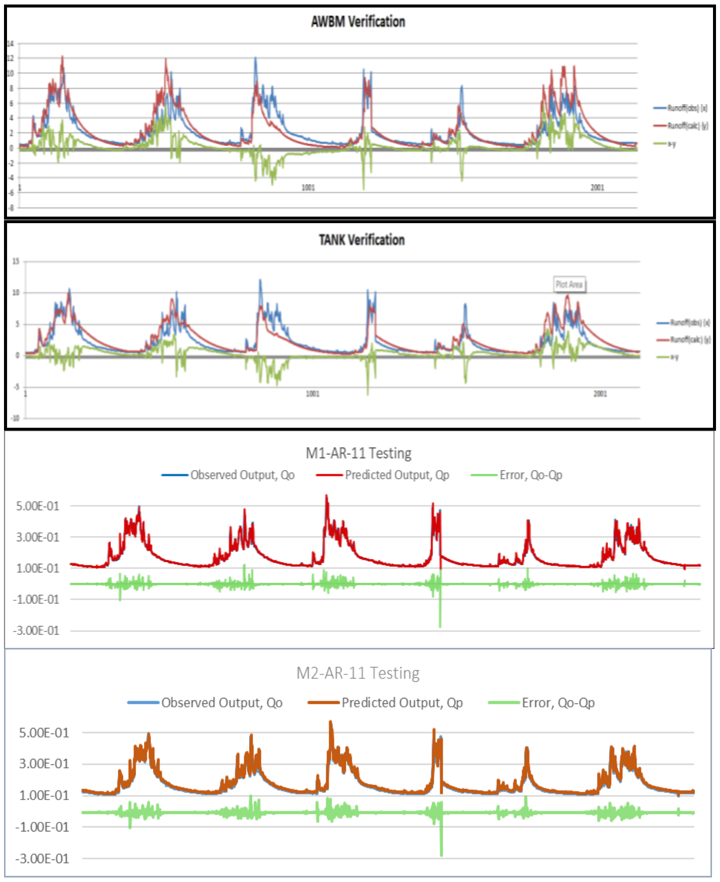


b. Jardine

Fig.14. Scatter plots of observed and predicted testing data for different models



15a. Godaveri



15b. Jardine

Fig.15 Time series plots of observed, predicted output and error for different models

V. Acknowledgement

I would like to thank Dr. Ashu Jain, Department of Civil Engineering, Indian Institute of Technology Kanpur for his enriching insights and helpful guidance which has hugely contributed in this undergraduate project (UGP). And, this wouldn't have been completed without assistance of Ms. Shubhanshi Singh and Mr. Anshul Yadav who performed conceptual modelling in this study. Their contribution is immense and possess huge value in the successful completion of this UGP.

VI. References

1. Bellie Sivakumar, Ronny Berndtsson, Jonas Olsson & Kenji Jinno (2001). *Evidence of chaos in the rainfall-runoff process*, Hydrological Sciences Journal, 46:1, 131-145. http://dx.doi.org/10.1080/02626660109492805

2. Huynh Ngoc Phien and Pramod S.S. Pradhan, Asian Institute of Technology, P.O. Box 2734, Bangkok 10301, Thailand. *The TANK Model in Rainfall-Runoff Modelling.*

3. McCulloch, W.S. & Pitts, W. Bulletin of Mathematical Biophysics (1943) 5: 115. https://doi.org/10.1007/BF02478259

4. Minsky, Marvin; Papert, Seymour (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press. ISBN 0-262-63022-2.

5. Narain, Seema; Jain, Ashu (2010). *Modelling Hydrological Process using Conceptual, Neural System, and Hybrid Approaches.* PhD thesis, IIT Kanpur.

6. http://www.igbp.net/download/18.62dc35801456272b46d4b/1398850074082/NL82-Deltas_infographic.pdf

7. Hargreaves, George; F.ASCE; G. Allen, Richard (2003). *History and Evaluation of Hargreaves Evapotranspiration Equation, Journal of Irrigation and Drainage Engineering.* DOI: 10.1061/(ASCE) 0733-9437(2003) 129:1(53)

8. Jani Fathima Jamal and Ashu Jain (2011). *Comparison of Conceptual and Neural Network Models for Daily Rainfall- Runoff Modelling.* International Conference on Chemical, Ecology and Environmental Sciences. http://psrcentre.org/images/extraimages/25.%201211237.pdf

9. https://toolkit.ewater.org.au/Tools/RRL

10. Boughton, W.J. (2004) The Australian water balance model, Environmental Modelling & Software, vol. 19, pp. 943-956.

11. R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung (2000). *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*. *Nature*. 405. pp. 947–951.

12. LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". *Nature*. 521(7553): 436-444. Bibcode:2015Natur.521..436L. doi:10.1038/nature14539. PMID 26017442

13. Marquardt, Donald (1963). "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". *SIAM Journal on Applied Mathematics*. 11 (2): 431–441. doi:10.1137/0111030.

14. Duchi, John; Hazan, Elad; Singer, Yoram (2011). "Adaptive subgradient methods for online learning and stochastic optimization" (PDF). *JMLR*. 12: 2121–2159.

15. Kingma, Diederik P.; Ba, Jimmy Lei. *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980, 2015.

VII. Model Development Tools

For preparing ANN RR models, various tools are available for easy deployment of ANN model's complex procedures. However, because of restrictive functionalities of toolbox (limitation of activation functions and optimization algorithms) like MATLAB, et al. and author's personal choice, Keras API [https://keras.io/] was used for developing ANN models in addition to MATLAB NN ToolBox [https://in.mathworks.com/products/neural-network.html]. Keras is a high-level neural networks API, written in Python programming language. It was developed with a focus on enabling fast experimentation for research with high computationally expensive neural networks variants like CNNs, RNNs et al. Another advantage of Keras or other open-source programming platforms for modelling different processes is their fast implementation of state-of-the-art technologies which makes modelling process less time-consuming and resourceful.

Python code for MLPNN, a sequential ANN model (feedforward or backward) is made available in this section for model redevelopment purpose. Further semantic details of the code below can be explored at: https://matplotlib.org/, https://keras.io/, https://pandas.pydata.org/, and https://scikit-learn.org/

Below is the python code used to develop ANN model for this study:

```
#Importing required modules
import pandas as pd
from pandas import concat
import numpy as np
import matplotlib.pyplot as plt

'''Before proceeding format the
catchment file as following:
    column1: Date,
    column2: Rainfall,
    column3: ET or blank (because of
physical models)
    column4: Runoff'''

#CHOOSE CATCHMENT FILE (godaveri.csv
or jardine.csv)
datafile = 'godaveri.csv'
#Choose training data length (3000
for godaveri.csv and 3500 for
jardine.csv)
ntrain = 3000
#Choose no. of lagged steps backward
(n_in) and forward (n_out) for data
preparation
n_in = 2
n_out = 1
#Choose no. of epochs (or
iterations) for modelling
epoch = 100
#Choose validation(0), testing(1) or
cross validation (2)
#test = 2 is for selection of hidden
layer neurons by cross-validation
test = 1

#Choose validation data length
(choose ~500 for both Godaveri and
Jardine)
nval = 498
#Choose no. of hidden layer neurons
nneuron = 5


#Read the data
data =
pd.read_csv("C:\\Users\\ShubhM\\Desk
top\\CE491A\\Project Files\\Data for
project\\%s" %(datafile),
                   sep = ",",
index_col = 0, usecols =[0,1,3],
                   parse_dates =
True)
data[data == -999] = pd.np.nan
array = data.values


#Checking Statistics of data
#print(data.describe())
#print(data.isnull().sum())


#Adding lagtime series of Q and R
#Method and functions for converting
data to the time-lag form
def lagvariable(data, n_in, n_out):
    """
    Frame a time series as a
supervised learning dataset.
    Arguments:
        data: Sequence of
observations as a DataFrame.
        n_in: Number of lag
observations as input (X).
```

```
        n_out: Number of
observations as output (y).

    Returns:
        Pandas DataFrame of lagged
series of an input variable.
        """
    colname = data.columns[0]
    cols, names = [], []
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(data.shift(i))
        names += [('%s(t-%d)' %
(colname, i))]
    # forecast sequence (t, t+1, ...
t+n)
    for i in range(0, n_out):
        cols.append(data.shift(-i))
        if i == 0:
            names += [('%s(t)' %
(colname))]
        else:
            names += [('%s(t+%d)' %
(colname, i))]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    return agg

def lagdata(data, n_in, n_out):
    """
    Arguments:
        Read above method's comments
for understanding the terms
        of this method.
        n_vars: No. of variables

    Returns:
        Pandas DataFrame of lagged
series of all variables.
    """
    n_vars = 1 if type(data) is list
else data.shape[1]
    lag_data = []
    for i in range(n_vars):
        data1 =
pd.DataFrame(data[data.columns[i]])

lag_data.append(lagvariable(data1,
n_in, n_out))
    lag_data = concat(lag_data,
axis=1)
    return lag_data


#Time Series of Q and R
'''data['Q'].plot(title='Daily
Runoff Time Series',
      sharex = False, figsize =
(30,10), color = 'blue')
plt.legend(loc='best')
plt.show()
data['R(mm)'].plot(title='Daily
Rainfall Time Series',
      sharex = False, figsize =
(30,10), color = 'red')
plt.legend(loc='best')
plt.show()
data['ET(mm)'].plot(title='Daily ET
Time Series',
      sharex = False, figsize =
(30,10), color = 'green')
plt.legend(loc='best')
plt.show()'''


#Auto & Partial Correlation Graphs
from statsmodels.tsa.stattools
import acf, pacf
def autopartcorr(data, c):
    data = data.dropna(axis = 0)
    f = pd.DataFrame(acf(data[c]),
columns = [c])
    g = pd.DataFrame(pacf(data[c]),
columns = [c])
    index =
f.iloc[1:20,:].index.values
    f1 = f.iloc[1:20,:].values
    g1 = g.iloc[1:20,:].values
    plt.bar(index, f1,
align='center', color = 'blue',
alpha=0.7, label='ACF')
    plt.bar(index, g1,
align='center', color = 'green',
label='PACF')
    plt.show()
    return f, g


#Uncomment below cell to calculate
auto-corr and partial auto-corr
functions values
'''c = data.columns[1]
```

```
acf1, pacf1 = autopartcorr(data,
c)'''

#Dividing train and testing data
#Dropping Missing Values
data = data.dropna(axis = 0)

#method for Dividing Training &
Testing data
def split(data, ntrain):
    n = data.shape[0]
    traindata =
data.iloc[0:ntrain,:]
    ntest = n - ntrain
    testdata =
data.iloc[ntrain+1:(ntrain+ntest),:]
    return traindata, testdata

#method for feature Scaling
from sklearn.preprocessing import
MinMaxScaler
def scaledata(data, ntrain):
    scaler =
MinMaxScaler(feature_range=(0.1,
0.9))
    data =
pd.DataFrame(scaler.fit_transform(da
ta), index=data.index,
columns=data.columns)
    train, test = split(data,
ntrain)
    return train, test

#method for converting train, test
into lagged data
def finallag(data, ntrain, n_in,
n_out):
    train, test = scaledata(data,
ntrain)
    lag_train = lagdata(train, n_in,
n_out)
    lag_test = lagdata(test, n_in,
n_out)
    return lag_train, lag_test

#Method or function for input and
output split for train and test data
def XYsplit(data, ntrain, n_in,
n_out):
    train, test = finallag(data,
ntrain, n_in, n_out)
    train = train.dropna(axis = 0)
    test = test.dropna(axis = 0)
    Xtrain = train.iloc[:,:-
1].values
    Ytrain = train.iloc[:,-1].values
    Xtest = test.iloc[:,:-1].values
    Ytest = test.iloc[:,-1].values
    return Xtrain, Xtest, Ytrain,
Ytest

#choose no. of training points
ntrain = ntrain
#Compare statistical properties of
train and test data
def statcompare(data, ntrain):
    traindata, testdata =
scaledata(data, ntrain)
    print(traindata.describe())
    print(testdata.describe())

#statcompare(data, ntrain)
#Change ntrain and do necessary
changes in training and testing
split
#for better predictions

#Uncomment below to save .csv file
of final lagged data and caculate
correlation values
'''lag_data = []
lagtrain, lagtest = finallag(data,
ntrain, n_in, n_out)
lagtrain = lagtrain.dropna(axis = 0)
lagtest = lagtest.dropna(axis = 0)
lag_data.append(lagtrain)
lag_data.append(lagtest)
lagdata1 = concat(lag_data, axis =
0)
lagdata1.to_csv("C:\\Users\\ShubhM\\
Desktop\\CE491A\\Project Files\\Data
for project\\lagjardine.csv",
sep=',')
#Correlation Matrix
corrm = lagdata1.corr()'''


#Defining Correlation coeffcient for
model evaluation
from math import sqrt
def corr(y_true, y_pred):
    '''
        y_true -> true output array
        y_pred -> predicted output
array
        Qobar -> mean of true
outputs
        Qmbar -> mean of predicted
outputs
    '''
    Qobar = np.mean(y_true)
    Qmbar = np.mean(y_pred)
    n = len(y_true)
    covar = 0
    varQo, varQm = 0, 0
    for i in range(n):
        covar += (y_true[i]-
Qobar)*(y_pred[i]-Qmbar)
        varQo += ((y_true[i]-
Qobar)*(y_true[i]-Qobar))
        varQm += ((y_pred[i]-
Qmbar)*(y_pred[i]-Qmbar))
    corr = covar/sqrt(varQo*varQm)
    return corr

#ANN Models' Evaluation Metrics
function
def evalmet(Qo, Qm):
    '''
        mbe -> Mean Bias Error
        mae -> Mean Absolute Error
        TSX -> Threshold Statistics
below X%
        mare -> Mean Absolute
Relative Error
        mape% -> Mean Absolute %
Error
        E -> Nash-Sutcliffe
Efficiency, value=-infinity to 1
        MF% -> relative % error in
maximum value of Qo
    '''
    n = len(Qo)
    be, ae, are, arpe = 0, [], [],
[]
    Qobar = np.mean(Qo)
    for i in range(n):
        be += Qo[i]-Qm[i]
        ae.append(abs(Qo[i]-Qm[i]))
        are.append(abs((Qo[i]-
Qm[i])/Qo[i]))
        arpe.append(abs(((Qo[i]-
Qm[i])*100)/Qo[i]))
    mbe = be/float(n)
    mae = sum(ae)/float(n)
    arpe = np.array(arpe)
    TS1 = (sum(arpe<1))*100/n
    TS25 = (sum(arpe<25))*100/n
    TS50 = (sum(arpe<50))*100/n
    TS100 = (sum(arpe<100))*100/n
    mare = sum(are)/float(n)
    mape = mare*100
    e1, e2 = 0, 0
    for i in range(n):
        e1 += (Qo[i]-Qobar)*(Qo[i]-
Qobar)
        e2 += (Qo[i]-Qm[i])*(Qo[i]-
Qm[i])
    E = (e1-e2)/e1
    Qo = np.array(Qo)
    MF = 100*((Qm[np.argmax(Qo)]-
max(Qo))/max(Qo))
    results = []
    metrics = ['mbe', 'mae', 'mare',
'mape', 'TS1', 'TS25', 'TS50',
'TS100', 'E', 'MF']
    metrics1 = [mbe, mae, mare,
mape, TS1, TS25, TS50, TS100, E, MF]
    j = 0
    for i in metrics:
        i =
pd.DataFrame(metrics1[j], columns =
[i])
        results.append(i)
        j += 1
    results = concat(results,
axis=1)
    return results


#Design ANN MODEL
seed = 7
np.random.seed(seed)
# define a base MLPNN model
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Activation
from sklearn.model_selection import
TimeSeriesSplit
from sklearn.metrics import
mean_squared_error
def baseline_model(n_hidneurons,
n_input):
    # create model
    model = Sequential()
    model.add(Dense(n_hidneurons,
input_dim = n_input))
    model.add(Activation('relu'))
    model.add(Dense(1))
    model.add(Activation('linear'))
    return model


def validatemodel(Xtrain, Ytrain,
Xtest, Ytest, nval, model,
optimizer, test, epoch):
    tbCallBack =
keras.callbacks.TensorBoard(log_dir=
'/tmp/keras_logs',

write_graph=True)

    #Compile model
    model.compile(loss='mse',
optimizer=optimizer)

    if test==2:
        #For cross validation
modelling (hidden layer neurons
selection)
        #Cross-Validation Split
        tscv =
TimeSeriesSplit(n_splits=3)

        trainrmse = []
        valrmse = []
        corrtrain = []
        corrval = []
        for train, test in
tscv.split(Xtrain, Ytrain):
            # Fit the model
            model.fit(Xtrain[train],
Ytrain[train], epochs=epoch,
                      batch_size=10,
verbose=1, callbacks=[tbCallBack])
            # evaluate the model

model.evaluate(Xtrain[test],
Ytrain[test], verbose=0)
            #Preciting the output
values for train and val data
            yhattrain =
model.predict(Xtrain[train])
            yhatval =
model.predict(Xtrain[test])
            #Calculating Correlation
Coefficient for train and val data

corrtrain.append(corr(Ytrain[train],
yhattrain))

corrval.append(corr(Ytrain[test],
yhatval))
            #Calculating Root Mean
Square Error for train and val data

trainrmse.append(sqrt(mean_squared_e
rror(Ytrain[train], yhattrain)))

valrmse.append(sqrt(mean_squared_err
or(Ytrain[test], yhatval)))
        return trainrmse, valrmse,
corrtrain, corrval
    else:
        #For validation and testing
modelling (hidden layer neurons
selection)
        ntrain = Xtrain.shape[0]
        ntrain = ntrain-nval
        Xval =
Xtrain[(ntrain+1):(ntrain+nval)]
        Yval =
Ytrain[(ntrain+1):(ntrain+nval)]
        Xtrain = Xtrain[0:ntrain]
        Ytrain = Ytrain[0:ntrain]
        #Fit the model
        History = model.fit(Xtrain,
Ytrain, epochs=epoch, batch_size=10,

validation_data=(Xtest, Ytest),

verbose=1, callbacks=[tbCallBack])
        epocherr = History.history
        #Predicting the output
values for train and val data
        yhattrain =
model.predict(Xtrain)
        yhatval =
model.predict(Xval)
        #Calculating Correlation
Coefficient for train and val data
        corrtrain = corr(Ytrain,
yhattrain)
        corrval = corr(Yval,
yhatval)
        #Calculating Root Mean
Square Error for train and val data
```

```
        trainrmse =
(sqrt(mean_squared_error(Ytrain,
yhattrain)))
        valrmse =
(sqrt(mean_squared_error(Yval,
yhatval)))

        if test==0:
            return epocherr, Ytrain,
Yval, yhattrain, yhatval, trainrmse,
valrmse, corrtrain, corrval
        else:
            return Ytrain,
yhattrain, trainrmse, corrtrain


def modeltesting(data, ntrain, nval,
n_in, n_out, n_hidneurons, test,
epoch):
    #Split data here
    Xtrain, Xtest, Ytrain, Ytest =
XYsplit(data, ntrain, n_in, n_out)
    n_input = Xtrain.shape[1]

    #Choose model
    model =
baseline_model(n_hidneurons,
n_input)

    #Model summary and optimization
function selection
    model.summary()
    #choose optimizer
    #sgd = optimizers.SGD(lr=0.01,
decay=1e-6, momentum=0.9,
nesterov=True)
    adam =
keras.optimizers.Adam(lr=0.001,
beta_1=0.9, beta_2=0.999,
epsilon=1e-08, decay=0.0)
    if test==0:
        #call simulation method
        epocherr, Ytrain, Yval,
yhattrain, yhatval, trainrmse,
valrmse, corrtrain1, corrval =
validatemodel(Xtrain,

Ytrain,

Xtest,

Ytest,

nval,

model,

adam,

test,

epoch)
        trainrmse =
np.mean(trainrmse)
        valrmse = np.mean(valrmse)
        corrval = np.mean(corrval)
        corrtrain1 =
np.mean(corrtrain1)
        return epocherr, Ytrain,
Yval, yhattrain, yhatval,
corrtrain1, corrval, trainrmse,
valrmse

    elif test==1:
        #call simulation method
        Ytrain, yhattrain,
trainrmse, corrtrain1 =
validatemodel(Xtrain,

Ytrain,

Xtest,

Ytest,

nval,

model,

adam,

test,

epoch)
        #Evaluating the model for
testdata (Only use when a best model

#is selected)
        yhattest =
model.predict(Xtest, batch_size=1)
        corrtest = corr(Ytest,
yhattest)
```

```
        rmsetest =
sqrt(mean_squared_error(Ytest,
yhattest))
        weight = model.get_weights()
        return Ytrain, Ytest,
yhattrain, yhattest, corrtrain1,
corrtest, trainrmse, rmsetest,
weight
    else:
        trainrmse, valrmse,
corrtrain1, corrval =
validatemodel(Xtrain,

Ytrain,

Xtest,

Ytest,

nval,

model,

adam,

test,

epoch)
        trainrmse =
np.mean(trainrmse)
        valrmse = np.mean(valrmse)
        corrval = np.mean(corrval)
        corrtrain1 =
np.mean(corrtrain1)
        return trainrmse, valrmse,
corrtrain1, corrval


def HiddenNeuronsSelection(data,
n_in, n_out, ntrain, test, epoch):
    nval = 0
    ntrainrmse = []
    nvalrmse = []
    ncorrtrain = []
    ncorrval = []
    for nneuron in (1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20):
        cvtrainrmse, cvvalrmse,
corrtrain, corrval =
modeltesting(data,

ntrain,

nval,

n_in,

n_out,

nneuron,

test,

epoch)

ntrainrmse.append(cvtrainrmse)
        nvalrmse.append(cvvalrmse)
        ncorrtrain.append(corrtrain)
        ncorrval.append(corrval)
    return ntrainrmse, nvalrmse,
ncorrtrain, ncorrval


if test==2:
    rmsetrain, rmseval, corrtrain,
corrval =
HiddenNeuronsSelection(data,

n_in,

n_out,

ntrain,

test,

epoch)

    #Choose number of hidden layer
neurons with best results from above
metrics
    nneuron = nneuron
#Model Evaluation
elif test==0:
    epocherr, Ytrain, Yval,
yhattrain, yhatval, corrtrain,
corrval, rmsetrain, rmseval =
modeltesting(data,

ntrain,

nval,
n_in,

n_out,

nneuron,

test,

epoch)
    #Error metrics for training &
validation
    trainres = evalmet(Ytrain,
yhattrain)
    valres = evalmet(Yval, yhatval)
elif test==1:
    Ytrain, Ytest, yhattrain,
yhattest, corrtrain, corrtest,
rmsetrain, rmsetest, weights =
modeltesting(data,

ntrain,

nval,

n_in,

n_out,

nneuron,

test,

epoch)
    #Error Metrics for training &
validation
    trainres = evalmet(Ytrain,
yhattrain)
    valres = evalmet(Yval, yhatval)
    #Final Evaluation for best model
    #Evaluating the model for
testdata (Only use when a best model
is selected)
    testres = evalmet(Ytest,
yhattest)
else:
    print("Looks like you have
failed your test of
comprehensibility. Please kindly
change the value! Oh I am sorry! it
must have been tough to decipher the
previous enigmatic text, let me make
it easier for you: choose 0, 1 or 2
only for test variable")


# Plot scatterplots for predicted vs
observed runoff
'''plt.scatter(Ytrain, yhattrain,
c="b", alpha=0.5,label="Predicted vs
True Output")
plt.show()
plt.scatter(Ytest, yhattest, c="b",
alpha=0.5,label="Predicted vs True
Output")
plt.show()'''
#train
'''plt.plot(yhattrain,
label='Predicted Output')
plt.plot(Ytrain, label='True
Output')
plt.legend(loc='best')
plt.show()'''
#test
'''plt.plot(yhattest,
label='Predicted Output')
plt.plot(Ytest, label='True Output')
plt.legend(loc='best')
plt.show()'''
```