An Adaptive NARX Neural Network Approach for Financial Time Series Prediction

BY

PARASHAR CHANDRASHEKHAR SOMAN

A thesis submitted to the

Graduate school – New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

written under the direction of

Dr. Ivan Marsic

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October 2008

ABSTRACT OF THE THESIS

An Adaptive NARX Neural Network Approach for Financial Time Series Prediction

By Parashar Chandrashekhar Soman

Thesis Director: Dr. Ivan Marsic

There has been increasing interest in the application of neural networks to the field of finance. Several experiments have been carried out stating the success of neural networks for time series prediction.

Most of the existing systems recommend single neural network architecture to be used for a particular time series. Our experiments have shown that a fixed architecture may not be the best approach across different time horizons. The thesis proposes a new methodology where multiple NARX (nonlinear autoregressive network with exogenous inputs) networks with different architectures are generated and evaluated before the beginning of a new time horizon. A network is selected from this set and employed to make predictions. This selection is based on past datasets only – making the system completely applicable to real world scenarios.

A framework of functions was built in MATLAB® to customize the Neural Network Toolbox ® for financial applications. This framework provides for all the basic

functions required by a financial neural network system. An adaptive system that uses technical indicators and some external time series as inputs was built. Different rules were developed and tested for selecting the best performing neural networks.

The new approach was tested on 5 currencies and the gold series. Our results show that high realized values of returns in the past, along with generalization is the best parameter to select a network for the future. A system with adaptive approach performs better than one with a fixed architecture. Our adaptive system out performed not only the fixed architectures but also other benchmarks like technical indicators, linear regression and baseline buy or sell strategies.

Acknowledgements:

I would like to thank my advisor Dr. Ivan Marsic for his constant help, guidance and encouragement throughout my thesis. I would like to thank him for motivating me to look into real life applications of various concepts that I learnt in his courses. I would like to thank Dr. Manish Parashar and Dr. Zoran Gajic for their valuable time and suggestions regarding my thesis. I am also thankful to Dr. Simi Kedia whose coursework and guidance helped me to obtain the necessary domain knowledge required for this thesis. I would also like to thank the staff of Center for Advanced Information Processing (CAIP) and the Electrical and Computer Engineering (ECE) department at Rutgers for providing the facilities and support required for this research.

Table of contents:

List of Tables:

List of Figures:

Chapter 1: Introduction

With the ever increasing acceptance or rather dependence of the financial world on modern technology, several neural network models have been proposed and tested in various fields of finance. Networks were shown to be effective in areas like forecasting, classification of bonds, bankruptcy prediction, risk assessment, financial evaluation etc.

The outputs of such systems are used to take decisions about various investments in the financial world. Different statistical and technical tools have been historically used to predict trends in time series and aid investment decisions. Due to the criticality of these decisions and the volume of investments involved there is considerable interest in modification and advancement of prediction methodologies.

Different types of neural network based systems have been employed in the past to predict different time series. Such research has usually been limited to few datasets in a time series. Our experiments have shown that an excellent performing neural network may not be able to continue that performance over different time horizons. The selection of architecture of these networks has been done usually by using some thumb rules (heuristics). The thumb rules may provide a good guideline, however cannot be followed in entirety [18].

Researchers have performed sensitivity analysis to determine the optimum architecture. Such analysis uses heuristics based on parameters like error [6] which our experiments show are not a very true indicator of network behavior. Additionally they may use behavior on the test set [5] to determine the best architectures. Using information about the test set may give very good idea about the best network for that data range; however in real life situations it is not possible to implement these heuristics. However if decisions were to be made for a real time training methodology we will have to depend only on the available past data and not the future data.

The challenge in our work is that we first need to identify a network that will make good predictions – to add to it we have to use only the past data. To begin with we need to determine our range of architectures to be tried out. We will have to determine what kinds of networks are to be employed, what kind of data is to be provided and how to preprocess it. We need to decide which networks are to be employed and how they are to be evaluated.

Once these networks are available we will have to set up rules on how to evaluate the performance results that will help us determine the network that is likely to demonstrate a good behavior in the coming time frame.

We will be following guidelines from several papers [9] [16] and perform our own tests to determine the selection of each parameter. We will be conducting

simulations on various time series across different time frames and evaluate the methodology.

In order to evaluate the overall performance of the methodology we need to compare it with certain benchmarks. We will compare the system performance with the performance of an “all buy”, “all short” an “all out of market” strategies. The best of these shall serve as the baseline. We will also test it against the performance of some market technical indicators. Since we are using a non linear regression technique we need to show that it performs better than similar linear regression. To advocate the need of changing architectures at different time horizons we will also benchmark the system against those static architectures that performed best.

We will use a framework of functions to generate the training, evaluation and test sets. We massage the available data to encode targets for supervised learning. We generate several NARX (nonlinear autoregressive network with exogenous inputs) networks and train them on the training data. We evaluate these networks on an evaluation dataset for determining the networks with generalized behavior. We will then evaluate the behavior of the selected network on the test set. We will perform this procedure on various datasets across a financial time series. We will also try out various different time series to evaluate the performance with the system. Standard predictive systems like linear, RSI system, linear stochastic system etc. will also be implemented in MATLAB ®. We will finally benchmark

the system of our performance with that of several neural network systems and Technical analysis tools.

The effectiveness of our system can be claimed only if it is able to perform better than other aforesaid methodologies. It is expected that our new approach will provide better returns than other systems.

In the following sections we discuss the related work in this field. We will discuss about the guidelines and findings of several researchers, which helped in deciding various parameters of the system. It will be followed by our technical approach and implementation of the system.

Finally we present our experimental findings which state that NARX networks are quick to train and simulate, making them appropriate for our problem rather than recurrent networks. We also noted that better results are obtained by following a pattern recognition approach rather than a function approximation problem. Our final simulations show that using the value of realized returns in the past is the best parameter to select a network for the future. We find out that an adaptive system with this approach outperforms the specified benchmarks.

Some specific contributions of our work can be stated as below:

- Introducing NARX neural networks to financial time series applications.

- Comparing the function approximation and pattern recognition approach in financial time series applications.
- Comparing various criteria for identifying a network that will perform well in future.
- Generating an adaptive system that outperforms technical indicators, static systems and financial baselines.

Chapter 2: Related work

With the advent of neural network research in early fifties, it has been put to use in various applications. They have been employed as decision making systems and filters in control systems, they have aided classification and prediction in wide fields ranging from psychology to environmental sciences.

There has been an increasing interest in the application of neural networks to finance [8]. They have been employed to solve problems like financial forecasting [18] [25], classification of bonds [6] [29], Bankruptcy prediction [22] [23] [24], Risk assessment [2], financial evaluation [1] [17] etc.

Forecasting time series, especially those in financial markets like stock markets has recently caught interest of several researchers [18] [25]. These applications include predicting the prices/movements of stock prices [6] [30], currencies [5] etc.

At a higher level of abstraction most of the systems seem similar in their approach. All of them use related input parameters, provide target to predict and train to make predictions

However a deeper analysis of these systems reveals that the systems can be very sensitive to selection of individual parameters. In course of our research we have compared performance of similar systems that use different proportions of

data (section 5.2), different data preprocessor (section 4.3). We notice that changing a single parameter causes a large change in the overall performance. Selection of different parameters, different methods of weight initialization, training algorithms, training conditions, network architectures etc change the behavior of the network greatly and effectively lead to a new system.

Apart from the academic and published research, there has been an increasing interest in financial prediction applications by users of various neural network environments like JOONE ® and MATLAB®. The JOONE® wiki [14] financial forecast tutorial earlier hosted an example about stock market predictions using feed forward networks. This example used a dynamic approach similar to our system to find the ideal number of neurons. The official JOONE® documentation [20] also provides an additional example for financial forecasting of stock trends using neural networks. Various crucial parameters of these examples have been implemented quite arbitrarily and lack sufficient verification. Analysis from our implementation section (Chapter 5) will show how the inputs, learning algorithms, preprocessing schemes etc. of these examples are not poised for optimum performance. Our experiments show that true benefits of dynamism can be realized only when the system has the option to select entire architecture including choices about inputs. Nevertheless they serve as good examples and contribute novel ideas on several fronts.

Individual systems usually reflect an individual designer's opinions and analysis. These systems might use parameters specific to the application that the system deals with. Researchers have published survey papers that provide an overall picture of the research in this field. Notably Fadalla [8] tried to analyze the applications in finance and outlined the similarities and differences of different networks on various parameters. Authors like Klimasauskas [16] and Gallo [9] have tried to provide a basic framework or methodology to create a neural network based system for financial applications. These survey and methodology papers serve as a good indicator of the standard practices and guidelines for building a new system.

Many current papers which implement neural networks for financial forecasting recommend a single architecture that is determined by sensitivity analysis based on factors like final value [5] or error [6]. Our findings in Chapter 6 show that fixed architectures may not be efficient across various time horizons. Moreover such systems use data from test set [5] which hinders their real life implementation as test set (future data) is not available in real life scenarios. These reasons point out the need to have an adaptive system that uses only past data for identifying networks likely to make profits in future.

Chapter 3: Usage scenario

The objective of this research was to come up with an adaptive architecture system to make financial predictions. Before we proceed with discussion of design constraints and implementation we will discuss the usage scenario of the system.

The data available for this research from our data vendor Global financial database (GFD) https://www.globalfinancialdata.com/ was of daily frequency. Hence all the analysis or simulations possible in the research would have a constraint of using frequency no higher than once a day. The target user of this system is envisioned to be a person with the willingness and the ability to trade once a day as the system will generate a decision once every day.

The person wanting to use this system in the future will be required to have access to a data vendor like the one mentioned above. Else the user can use data from free sources like Yahoo finance and Google finance and follow the format specified in section 5.3 to train and simulate the networks.

The system does not make any presumptions about the financial knowledge of the system user. However as mentioned in section 5.2 and section 6.3, basic financial knowledge would be very helpful to avoid certain special cases that may lead to losses.

The simulations here used a selected system for a period of 4 months. If the user is to follow this duration, computing resources required by the user would be very low. If however the user prefers training after shorter durations, the requirements will increase accordingly.

A daily predictive system implemented using our methodology when provided with today's and past data will give advice for the next day – "Buy", "short" or "out of market". If the user buys the asset and the price goes up, the user earns profit. If it goes down the user makes a loss. The effects are vice versa for short.

We implement a system incorporating buy as well as short in order to utilize the potential of the network to predict positive as well as negative trends.

To analyze the performance of this system we will simulate the decisions on the test set and evaluate the returns if the advice of the system were to be followed. The returns of this system will then be benchmarked against the returns of various other system and financial baselines like all-buy, all-sell, all-out of market.

Chapter 4: Technical approach

4.1 Neural network basics

We are going to discuss the basic terminologies and concepts of neural networks that will help us in understanding the forthcoming sections easily.

H. Demuth [4] states that

> "Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. You can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements."

Each such 'single element' is termed as a neuron.

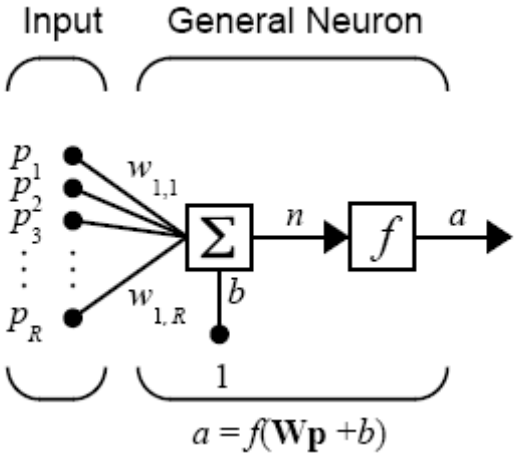

Figure 1: A general neuron structure of a neural network. [4] pg: 5-8

In figure 1:

f: transfer function of the neuron

R: number of inputs

W: weight

b: bias

The output of the neuron is compared with the ideal output and changes are made to the weight and in such a way that the ideal output would be achieved.

The neuron receives inputs from one or more inputs. The output of this neuron depends upon the ‘activation function’ or ‘transfer function’ of the neuron.

The basic transfer functions that we will be talking about in the coming chapters are the sigmoid (or the log-sigmoid transfer function) and the tan based (tan-sigmoid) transfer function. They are described as follows:

Log sigmoid transfer function:

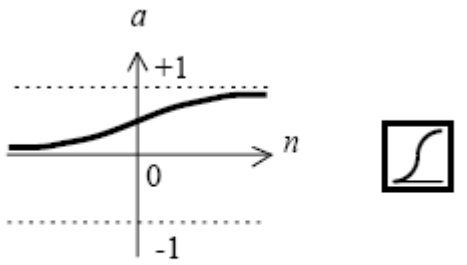

Figure 2: Log sigmoid transfer function. [4] pg: 5-8

This network can receive inputs from negative infinity to positive infinity and always generates an output between 0 and 1

Tan sigmoid transfer function:

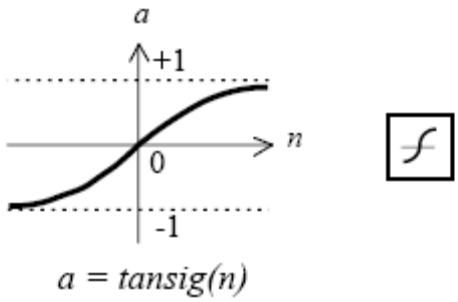

Figure 3: Tan sigmoid transfer function. [4] pg: 5-9

This transfer function receives inputs from negative infinity to positive infinity and gives an output between -1 and 1.

The arrangement of these neurons leads to different types of neural networks. Broadly they can be classified as static and dynamic. Static networks do not have any feedback loops (outputs of a neuron fed back to some previous neuron) or taps (delay lines that feed the network with past values of inputs). Dynamic networks may have one of these two. Dynamic networks are preferable for time series as they have memory in the form of loops or delay lines [4].

The tapped delay line (TDL):

The taps or the delay line is used to feed the network with the past values of inputs. In figure 4 we see that input enters from the left and goes through N-1 delay elements to generate a vector of N outputs

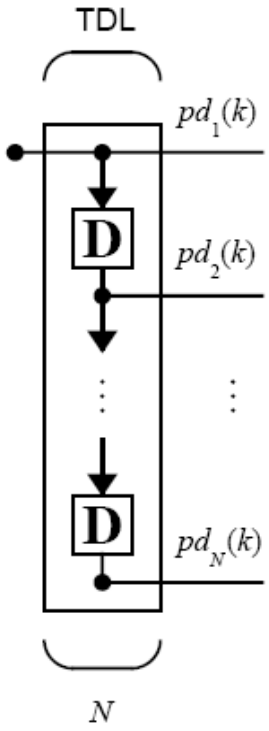

Figure 4: Tapped delay line. [4] pg: 4-10

We will now discuss in brief about the NARX network that we propose to use in this research. The NARX network uses the past values of the actual time series to be predicted and past values of other inputs (like currencies of other nations and technical indicators in our case) to make predictions about the future value of the target series. These networks are again classified as series and parallel architecture

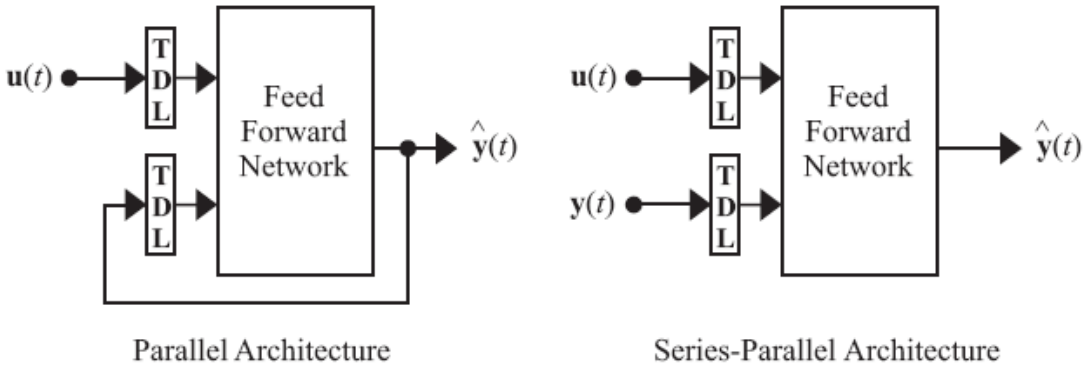

Figure 5: parallel and series parallel architectures of NARX networks. [4] pg: 6-18

In figure 5 u(t) represents the past exogenous values (currencies of other nations and technical indicators in our case) y(t) represents the past values of the actual series to be predicted. ŷ(t) indicates the predicted values. If past values of actual series are not being recorded, they will not be available to the system. In such situations the networks uses its past predicted values. In our case we will have the actual past values; hence we prefer to use them instead of our predictions. Thus we are able to base the model on actual values which are more reliable than our predictions

4.2 Selection of neural network type

Neural networks can in general be divided into two categories – static and dynamic. Static networks have no feedback elements and no delays. The output is calculated directly from the current inputs. Such networks assume that the data is concurrent and no sense of time can be encoded. These networks can thus lead to instantaneous behavior.

Dynamic networks may be difficult to train but are more powerful than static networks. As they have memory in form of delays or recurrent loops, they can be trained to learn sequential or time varying patterns. This makes them networks of choice for various applications like financial predictions, channel equalization, sorting, speech recognition, fault detection etc.

Since we are dealing with a time series it is necessary to use dynamic networks. Dynamic networks can be of two types ones with feed forward connections and taps and those with feedback or recurrent networks.

At this stage two breeds of networks were considered the NARX (Nonlinear Autoregressive Neural Network) and recurrent networks. NARX networks use taps to set up delays across the inputs and also incorporates the past values of the output. Recurrent networks have loops within intermediate layers and incorporate memory via these loops.

Both the networks have been employed in dynamic applications .The difficulties and the time required in training the recurrent networks is a known problem. To evaluate the resources requires by each of these networks, a batch of 40 Networks (1 neuron to 40 neurons) was trained under the same conditions for same datasets and training parameters (MATLAB® Neural Network Toolbox). It was observed that recurrent networks took almost 17 times more time to complete training and simulations as compared to the NARX batch.

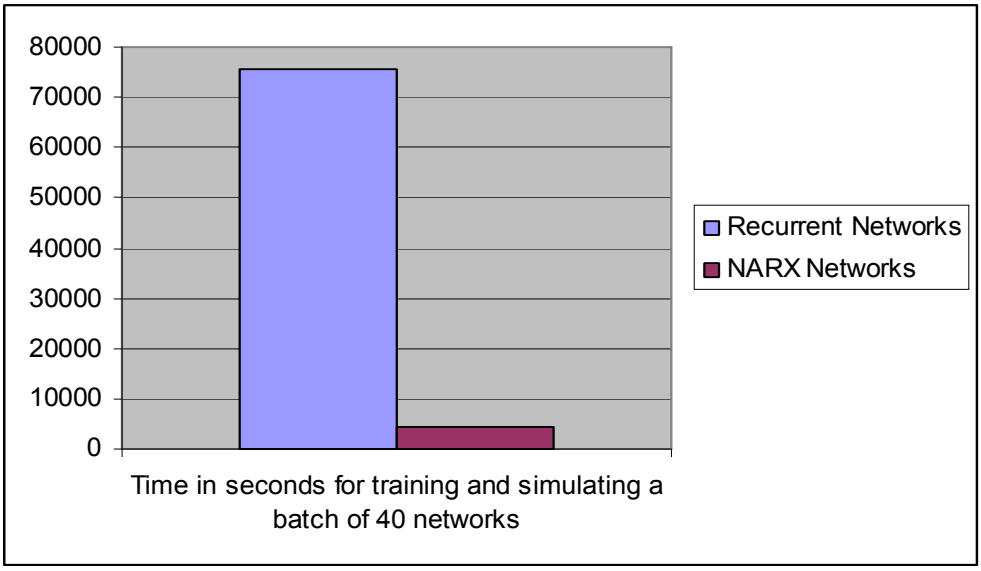

Figure 6: Comparison of training and simulation times of a batch of 40 neurons on 175 Opteron processor 16 Gb RAM (Shared) Recurrent Vs NARX

Our approach is that of simulating several networks, we will have to simulate several networks ranging up to a few hundred neurons for each time frame. This ratio of required time would further be aggravated when we would train networks with hundreds of neurons.

Dunis and Williams [5] who have evaluated neural network application to financial predictions have stated the problems in using recurrent networks. They state that there is no theoretical proof or reason why outputs of layers should looped back to the layers additionally the difference in the performance of recurrent and simpler models is marginal. The neuron structure of the NARX is similar to simple feed forward network making it a simple model.

Our limited computational resources do not allow us to train and simulate such networks. However for applications where these gains outweigh the cost of computation, recurrent networks should be tried out.

Due to the above mentioned finding and the limited computing resources available for this research it was decided that NARX networks will be used for our prediction system.

The basic NARX network is used for multi step predictions. It is assumed that actual past values of target are not available and the predictions themselves are fed back to the network. Since we will have access to the actual past values we will provide those values instead of our past predictions. This helps the system train on actual values rather than predictions. This is achieved by using the series-parallel version of the NARX network which is described in section 4.1.
Thus a series parallel NARX dynamic network will be used as a basis of our system.

4.3 Training and data preprocessing

Different researchers have different take about the method of preprocessing the data and training a neural network for such applications. Roughly speaking, they can be called as the pattern recognition approach and function approximation approach. We will evaluate both the approaches; our implementation of each of them has been discussed below.

4.3.1 Function approximation:

Here we will assume that we can feed the returns series value at each instance and we can then try to predict the return value at each day.

We just map the targets within the neuron range and train the network using an algorithm that is suited for such an application. The MATLAB® Neural Network Toolbox [4] provides a detailed survey of algorithms appropriate for various applications. The results therein state that Levenberg-Marquardt (LM) is a good algorithm for small function approximation problems. Scaled conjugate gradient algorithm (SCG) shows good performance for problems of all sizes, and is especially good for large networks. Hence we decide to use scaled conjugate gradient algorithm for training in this approach.

We will train the NARX network over all scaled values and SCG algorithm. If the value of this predicted signal is positive and above a threshold, we anticipate a positive return and buy, if it is negative and below a threshold we short. Else we stay out of the market. These thresholds are set to remove some small values from the decision making process as they are deemed to be unreliable.

Table 1 below summarizes how “buy”,” short” or “out of market” decisions are generated:

Output of the neural network	Decision of system
greater than (+ threshold)	Buy
Less than (- threshold)	Short
Between (threshold) and (- threshold)	Out of market

Table 1: Generation of “Buy”, “Short” or “Out of market” signals by the system

4.3.2 Pattern recognition:

In the function approximation approach we provided the network with the actual returns and expected the network to predict the actual returns. In the function approximation approach we do not provide actual returns but we provide the network with a pattern of movements on returns. We encode highly upward price movements as 1 and highly downward movements as -1. We then ask the network to predict occurrences of such patterns.

We encode the information about training targets as follows.

- When the value goes more than the average positive movement by a threshold the system encodes +1
- When the value goes more than the average negative movement by a threshold the system encodes -1
- otherwise it is zero

The range is selected to comply with the ranges of tansig neurons which are going to be used

The network is trained using this data and predictions are made about the trend each day. If the value of this predicted signal is positive and above a threshold, we anticipate a positive return and buy, if it is negative and below a threshold we short. Else we stay out of the market. These thresholds are set to remove some small values from the decision making process as they are deemed to be

unreliable. Values of movements that are greater than roughly about 1.5 times the average were used to encode the 1 and -1. Predicted values less than roughly about +/- 0.2 were ignored in the decision making. These thresholds were set/fine tuned manually by observing the behavior of individual time series.

It was not possible to perform complete sensitivity analysis with respect to these thresholds. A complete analysis would require training and simulation of several combinations of these thresholds and analyzing them. This was not possible to the limited computing resources. Hence like in most of the neural network systems we make manual designing decisions with logical reasoning.

Analysis of equivalent adaptive systems trained as function approximation and pattern recognition approaches reveals that the performance of pattern recognition systems is much better as compared to function approximation system. This can be seen in figure 6. This is understandable because the assumption made by a function approximation system is that there is an underlying function by which exact values can be predicted. Pattern recognition does not try to find exact predicted value but tries to find a trend. Marrone [20] expresses problems with function approximation approach and suggests pattern recognition instead.

Comparison of the realized returns using Function approximation and pattern recognition systems is outlined in table 6:

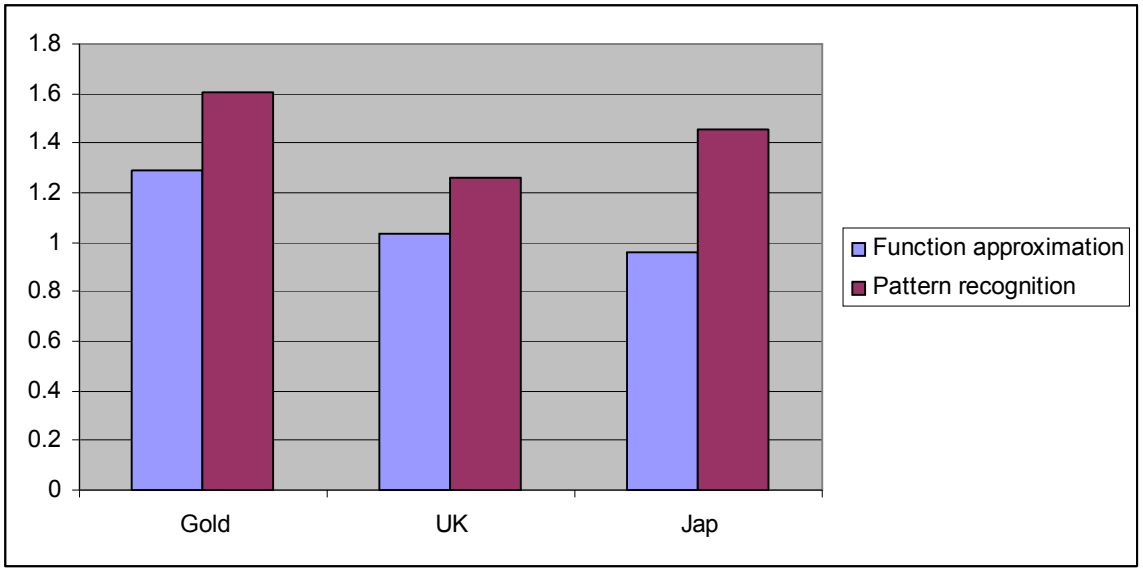

Figure 7: Comparison of two equivalent NARX systems using the function approximation and pattern recognition approach for massaging data and training

The following can be inferred about the two proposed approaches:

- Function approximation has performed poorly as compared to the other approach.
- This network has values available at all the points, it is not possible to take extreme stands and the network prefers to display an averaged behavior.
- The algorithm also encourages an averaged behavior

Hence we conclude that this approach is not fit for predictive systems. More importantly we can infer that it is difficult to predict actual movements daily. We are better off trying to predict the trends by using the second approach – pattern recognition.

These findings lead us to using the pattern recognition approach of massaging and training towards our problem. Henceforth whenever we talk about our

system, we are referring to systems with the pattern recognition approach for massaging and training

4.4 Network architectures to be considered

On the basis of our earlier findings we have decided to use NARX networks for our application. Now we determine the other parameters of the network architecture.

The NARX networks will have a linear input layer of neurons (default by MATLAB®) for the hidden and the output layers we will use the Tansig neurons. Tansig neurons and sigmoid neurons have been extensively used in most of the neural network applications. Both of them are similar in behavior and have a similar looking transfer function except for the range of outputs that they can generate. A sigmoid layer has a range form 0 to 1 whereas the range for Tansig transfer function ranges from -1 to 1. Klimasauskas [16] suggests that sigmoid neurons be preferred to determine average behavior and Tan based layers to find deviations from normal. The application at hand also prompts us to use the Tansig layer. Additionally our development environment the MATLAB® Neural Network Toolbox also recommends Tansig layers for pattern recognition problems and provides it as the default layer. The Tansig activation function has been described in the section 4.1

4.4.1 Number of layers:

We have a linear layer of linear neurons at the input; the number of neurons in this layer will be equal to the number of inputs that we have provided to the networks. Using linear neurons at the input is a standard practice and is used merely as an interface between the inputs and the hidden layers. In fact MATLAB® Neural Network Toolbox does not count it as an independent layer.

There is no certain figure for the number of hidden layers to be used. Cybenko [3] Hornik et al. [11] [12] show how a single layer of hidden neurons is capable of adapting to complex functions. Survey papers on this field [8] [30] also reveal how a single hidden layer of neurons is the most preferred option. Using additional layers adds up complexities to the model and increases the time required for training and simulation.

The framework described earlier has capabilities to generate, simulate networks with multiple hidden layers. It can be seen from the code that the framework can generate the number of layers specified by the user, and use user specified number of layers for each layer. However the limited processing resources available did not permit us to perform tests with several layers, hence we too decided to use single layer of hidden neurons for this research.

The inputs were mapped in the input range of -1 to 1 by our massager. The outputs were also to be mapped within this range. Naturally Tansig was the

output neuron of choice since it maps the inputs to this range. A single output neuron was thus used at the output.

4.4.2 Number of Taps and Hidden Neurons:

The tapped delay line of the NARX network allows passing of past values to the network. They make the data sequential unlike the original concurrent dataset. Thus these taps set up a sense of time and correlation of past values.

The numbers of neurons are supposed to be related to the complexity of the application at hand as each neuron in the hidden layer contributes weights and flexibility to the network. There is no standard method to determine the number of neurons to be used and several thumb rules are used. Mehta [21] has stated how the architecture depends on these thumb rules and it is not necessary that they work well.

Regarding the numerosity of neurons to be used in a network different thumb rules have different opinion. Klimasauskas [16] suggests that there at least be 5 examples per weight. For our application of using 10 inputs and upto 4 taps with 240 training points – we have 240 examples hence we will be allowed upto 48 total weights and biases. The number of weights and biases in our structure can be calculated as

Number of weights and biases = (I + 1) * H + (H + 1)* O

Where,

I = number of inputs (10 to 40)

H = Number of hidden neurons (to be determined)

O= outputs (1 i.e. daily prediction)

This would limit us to a range of 1 to 4 neurons.

The rule mentioned by Gallo [9] would suggest using twice the number as inputs i.e. upto 80 neurons.

Bayesian regularization is a method that uses the LM algorithm and tries to optimize the error and number of neurons both. This method id not optimized to minimize error alone, also is more suited to function approximation applications. However we ran jobs over it to get a rough number of neurons that it would suggest. It used around a single neuron.

Different researchers have used 5 [5] or 20 [4] neurons in their applications.

Some have used extremely large number of neurons in their application [6]

Some researchers have used some kind of sensitivity analysis to determine the number of neurons. These researchers use errors [6] or analyze behavior on the test set to settle for a fixed architecture [5]. We are going to follow an adaptive approach in determining the number of taps and neurons in this research. Various policies for selecting a network in the adaptive system have been discussed in the following section.

4.5 Policies for network selection

As discussed in the introduction section (Chapter 1) we propose an adaptive system where the system will first train a set of networks on the training set, then evaluate performance on the evaluation set. Then we use the results obtained on the evaluation set to identify networks that are likely to make profits on the test set.

We believe that a network that behaves satisfactorily on this evaluation data will be a good candidate to make predictions for the test set (future). Now we need to decide which performance parameter of the evaluation set will we actually use to select a network. Popular performance parameters like mean square error, hit rate or realized value can be used. Hence at this stage we decide to frame three policies for selection of a network for the future. The policies are as follows: policy 1: We will select a network that has minimum error on the evaluation set to make predictions on the test set (future)

Policy 2: We will select a network that has maximum realized value on the evaluation set to make predictions on the test set (future)

Policy 3: We will select a network that has maximum hit rate on the evaluation set to make predictions on the test set (future)

We will simulate the system behavior for different time series and different time horizons. We will analyze the behavior of each of these policies and determine the best policy for identifying a network that would make profit in the future.

Chapter 5: Implementation

5.1 Comparison and selection of platforms

The research to be conducted will require an environment where different kinds of neural networks can be generated, trained and evaluated. This environment should provide for all the basic network architectures, training algorithms and other features required by neural networks. It was preferable if this platform could provide for several options for each of these features. This would allow us to test the performance of the network across various settings and analyze the improvement/degradation in the network behavior. Having these functionalities in built would help us channel our resources on our problem at hand rather than implementing existing standard algorithms and architectures.

Since we are also required to benchmark our network performance with other standard methods like linear regression, technical indicators etc. it would be helpful if he selected environment would allow us to easily encode the above mentioned methods too.

The two environments that were considered for the task were the Java Object Oriented Neural Network Environment - JOONE® [19] and MATLAB® Neural Network Toolbox®

JOONE® is an open source Java framework to generate and run neural network based applications. It consists of functions written in Java which are modularized to provide flexibility and hence can be used across wide range of applications. It also provides for elementary distributed support for training networks in parallel. JOONE® also provides a GUI for generating, training and running neural networks. However it does not allow customization to the degree required by our research.

However JOONE® has a very limited set of architectures, algorithms and other functionalities as compared to MATLAB® Neural Network Toolbox®. It was observed that JOONE® based system takes longer training times as compared to MATLAB® Neural Network Toolbox® systems. JOONE® does not have an inbuilt NARX network unlike MATLAB®, which meant that additional network structure would have to be constructed using the basic feed forward network. Additionally there has been no update to JOONE® since January 2007. Several bugs and errors pointed by users have not been fixed and many such issues remain pending. There is no official support for JOONE® and the Wiki and FAQ pages have been unavailable for quite some time. The documentation [20] lacks detailed information on several fronts.

MATLAB® Neural Network Toolbox® on the other hand is a commercial, professionally built framework. It has a large set of available architectures, algorithms and related features. These have been implemented in an optimized

fashion and help in reducing training times as compared to JOONE®. MATLAB® also has a dedicated Distributed Computing Toolbox® which can be used to train and evaluate networks in parallel. MATLAB® also provides an elaborate users manual explaining various function implementations and specifications. The code of each of these functions is well commented and helps in improving the understanding of their implementation. MATLAB® has excellent support for their products in the form of helpdesk and user community.

Considering the factors mentioned above it was decided to use MATLAB® Neural Network Toolbox® for our research.

Within MATLAB® we are going to write an additional framework of functions which will help us customize the neural networks to our financial applications. This framework of functions will provide abstraction from the internal implementation and allow the user to set vital parameters while calling these functions (as arguments). We will implement and evaluate our proposed methodology via these functions. A detailed description of these functions and their utility has been discussed in chapter 5 - Implementation.

5.2 Data for simulations:

5.2.1 Source:

Global Financial Database was chosen as the provider for data. Rutgers University has an institutional subscription with the data vendor this would allow us unlimited downloads of various time series. The vendor has a tool by which we can organize various time series into a single worksheet and then download it as csv or excel. Since Rutgers University has a subscription with the vendor it was easy to get good customer and technical support – something that would not have been difficult with free data vendors.

However the functions that have been written use the data from the `.mat` extension files. Hence the user may choose to select data from any source including free sources like Yahoo finance and arrange it in a format specified in section 4.2. This data can now be used with the functions coded for this research.

As a part of this research, when data was first simulated for the currency of France, the NARX system along with the other systems made heavy losses. On analyzing, we realized that the data provided for the exchange rate which corresponded to that test set was faulty. Our data vendor promptly fixed the data and the new sets for France were simulated and the results have been incorporated in this research. This however brings to light a rare probability that the data might be faulty and if not looked at carefully might lead to losses.

5.2.2 Time series:

Currency prices were used as target prices of inputs. Currency prices were used as they are considered to be stable and not easily fluctuated by rumors like the stock prices. This however does not mean that currency movements are easy to predict. Dunis and Williams [5] have demonstrated how a returns series for currency shows similarities to white noise and is not easy to predict. Our research involves comparison of our system with several systems, hence we select currency series as the value is highly dependent on the parameters that we provide and not affected randomly by parameters like rumors that have not been provided to these systems. This provides a fair competition ground for all the systems involved.

We used the time series of exchange rates of the following countries United Kingdom, France, India, and Canada. We used technical indicators and 2 other currencies as inputs for each time series. Exchange rates of Japan and Australia were used as exogenous inputs for all the series.

To add variation 2 new sets were created which would use relationships between Gold series. A gold series was also tested. This series used gold price as target and used the technical indicators, exchange rates of United Kingdom and Japan as inputs. A series was also tested for Japanese exchange rate and used the technical indicators, Gold price and Exchange rates of Japan as exogenous inputs.

5.2.3 Data for training, evaluation and testing:

The performance of the system was tested over 6 datasets of 4 months each, thus encompassing each series for 2 years. We used the most recent 2 years for all the testing (May 2006 – April 2008)

Thus we tested the system over 6 time series of 6 data sets each (total of 36 datasets)

For predicting each of these test sets, we would reserve some amount of past data for evaluation and some amount of data prior to that as the training data. Thus as we move into consecutive test sets we would have to maintain a sliding window approach to extract the training and evaluation data.

Regarding the ideal ratios for training, evaluation and testing data, the MATLAB® Users guide [4] suggests a ratio of (60%:20%:20%). Gallo [9] Suggests that (60%:20%:20%) and (60%:30%:10%) are the most popular ratios. We used the (60%:20%:20%) proportion for our system. It was observed that adding more training data did not improve results. Adding more data to the evaluation data too deprived the training set of vital recent data and showed acute fall in performance. Following is a chart representing the fall in the realized value of the system when the same NARX system used 2:1 ratio between evaluation and test data volume as compared to the standard 1:1

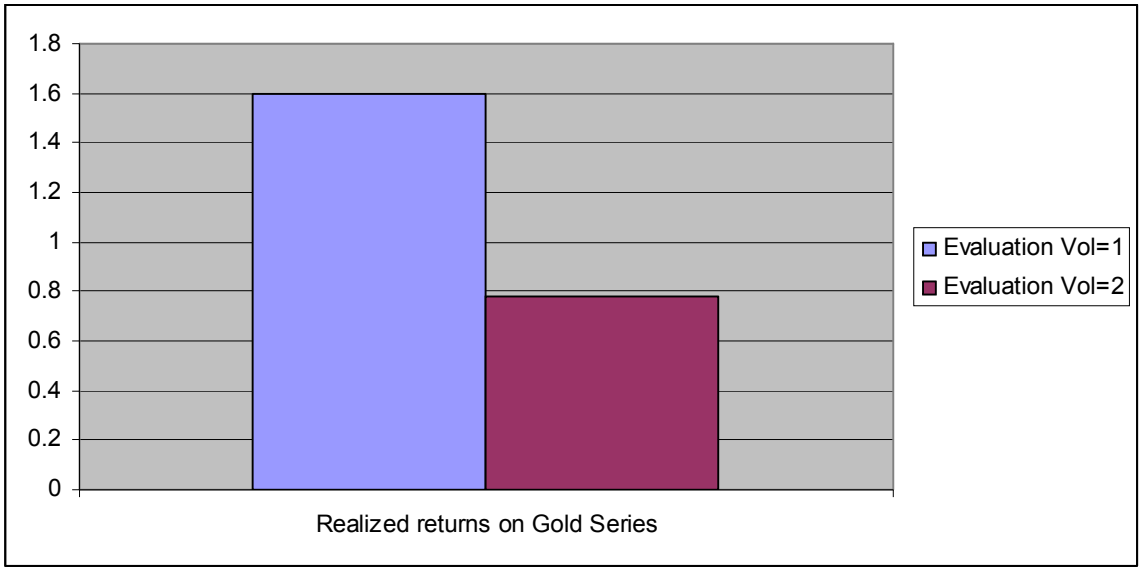

Figure 8: Realized returns on Gold Series: 3:2:1 NARX Vs 3:1:1 NARX system

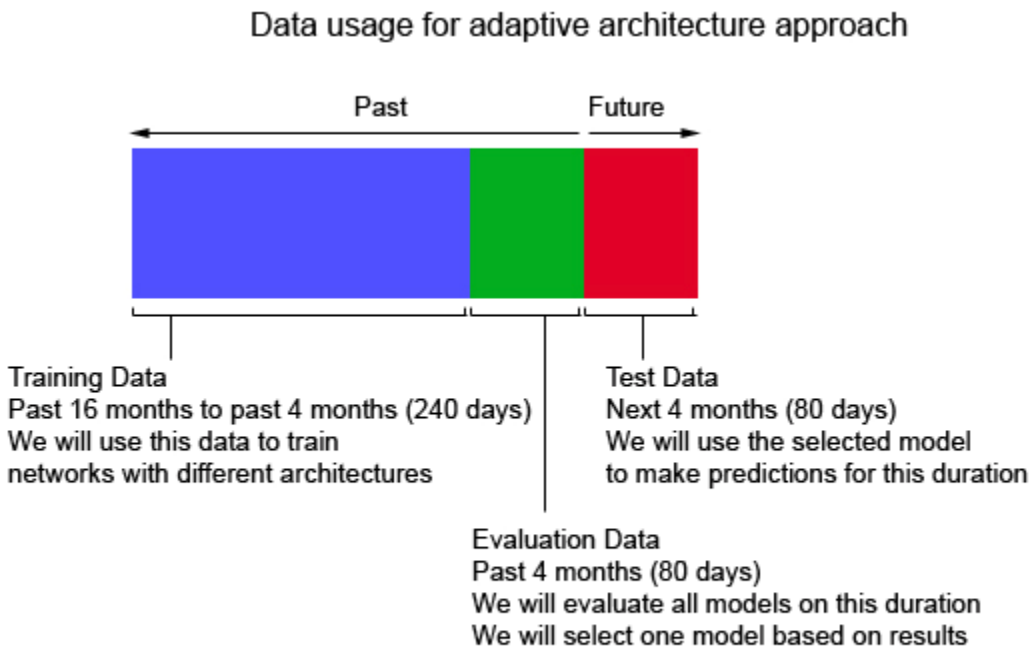

Figure 9: Data Usage for adaptive architecture

5.2.4 Targets for training: stationary or returns series?

Regression and prediction systems always use stationary data. The time series that we have chosen for our research – like most of the time series are not stationary but have some upward or downward trend. Such targets are not ideal for training and predictions. Hence we use the returns series where each entry is (current value-previous value)/previous value. A set of Gold datasets was simulated using the stationary data seta and it was observed that it shows degraded performance as compared to a system trained on returns series. Even Dunis and Williams [5] have documented similar problems related to stationary in exchange rates.

5.2.5 Inputs for training:

We use a mix of auto regression (past inputs), technical indicators and other currencies as inputs for our system. They are listed as follows:

- Past returns of 2 exogenous values:
    - Exchange rates of 2 other nations used as inputs. Many neural network systems use associated series as additional inputs
- Past values of Tech Indicators
    - Standard RSI
    - Standard Fast Stochastic
        - Both of them are well known technical indicators
- Past values of actual asset and preprocessing data

- Past values provide a sense of autocorrelation. Past values believed to have effect on current values.

The details of the individual inputs for each series can be found in appendix.

5.3 Data format and organization:

The data obtained from the data vendor Global financial data (https://www.globalfinancialdata.com/) was in the form of excel/csv (comma separated values) this data was converted into the MATLAB® compatible data format namely .mat format. All the excel/csv (comma separated values) data was converted to the MATLAB® format by using the `xlsread` command of the MATLAB® environment.

Since the system was to be used over different assets like gold series and currencies and the framework was to be made compatible for future research, it was decided that the data should follow a fixed organization. The first two columns allow the user to provide any two related time series that are to be used as exogenous inputs to our network. We have used closing prices of other exchange rates or gold price in our examples. The last 4 columns should represent the open, high, low and close price of the asset to be predicted. Having this predetermined data format and organization made rapid development and compatibility of all framework functions possible. Users who want to simulate/modify the work in the future can stay abstracted with the function implementations and just alter the statistical/network parameters.

The sequence of columns in the data to be provided is as follows:

1. Related time series 1
2. Related time series 2
3. Opening price of time series to be predicted
4. High price of time series to be predicted
5. Low price of time series to be predicted
6. Close price of time series to be predicted

The functions coded for this research are extendible for experiments involving more than 2 external time series. As long as the last 4 columns stay as specified, the user can add as many series as his or her computational resources are able to handle.

5.4 Data handling:

In this section we describe the functions that have been coded using MATLAB® to implement our system.

`dataCutter:`

The NARX networks require training, validation and testing data for their functioning, this data again needs to be provided as inputs and targets. Initial data to be filled in the tapped delay lines needs to be provided separately as tap data. The tap data also has to be split into inputs and targets.

As per our findings in section 5.2.3 we are going to use the 3:1:1 ratio for these datasets. However we do not limit the future user of these functions to these

ratios alone. The user may prefer to use other ratios subject to the constraints mentioned in 5.2.3

This function gives the user the freedom to cut the datasets from the available dataset on the basis of these ratios. It cuts the test data forward from the test point and the other 2 sets backwards. The data are output as sequential data hence can be used in static as well as dynamic networks.

The function receives the following arguments. `diffSet` is a two dimentional array as specified in the section 5.2. other arguments are integers.

- `diffSet` = The original data provided by user as described in section 5.2
- `testPoint` = The starting point for the test set (used to delimit the train, evaluation and test data)
- `testDur` = The number of elements in test set (duration of testing)
- `tap` = Number of taps in the delay line
- `trainVol` = ratio of number of training data points to test data points
- `valVol` = ratio of number of evaluation data points to number of test data points

The entire block below represents the input data. As specified in 5.2 the input data is in the form of a two dimensional matrix. Different columns are different time series, the last column is the time series to be predicted. Different rows represent different instances in time.

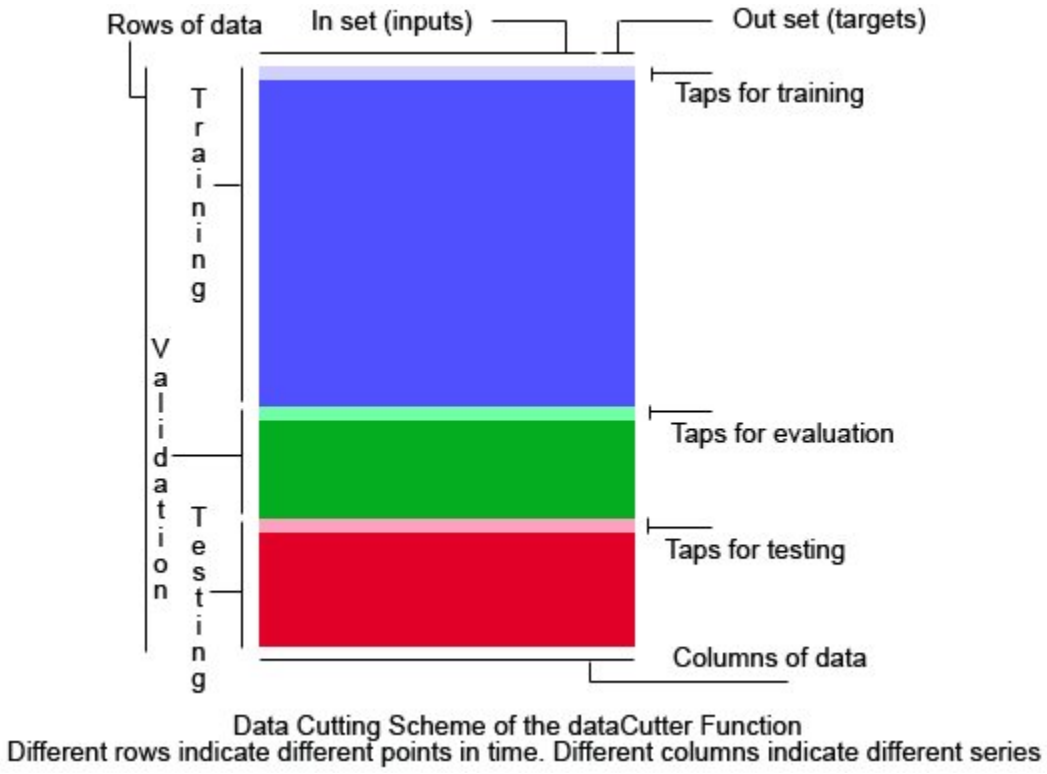

Figure 10 : The data cutting scheme of the data Cutter function

The function will generate all the datasets according to the specifications of the user and return them back to the calling function.

A standard naming scheme has been followed for the datasets to provide abstraction from the actual generation process. The naming scheme is as follows:

- name consists of 4 segments (e.g. 'train"Tap"In"S')
- segment 1 :
    - `train` - refers to training set
    - `val` - refers to validation set

    - `test` - refers to validation set
- segment 2:
    - `Tap` - implies it is data for tapped delay line
    - `Set` - implies it is data for actual set
- segment 3:
    - `In` - implies it is network input
    - `Out` - implies it is network output
- segment 4:
    - `empty` - Implies it is matrix data
    - `S` - implies it is sequential data

Thus the function returns 12 datasets in sequential format that can be used conveniently with the dynamic networks in MATLAB®.

`mainSetCleaner:`

This is a simple function used to eliminate not a number (NaNs) values from the targets. This function is optional and may not be used by some network implementations. Complete implementation and code available in the appendix.

`diffGenerator:`

This is a simple function used to create a difference or returns series of the provided datasets. The outputs of this function are used by the preprocessor and other functions. Complete implementation and code available in the appendix.

`massage1/massageP` (data preprocessors):

These functions are used for data preprocessing. They are named after the colloquial term ‘massaging’ which is used for data preprocessing. The `massage1` function implements the pattern recognition approach described in the methodology section, while the `massageP` function is used to implement the function approximation approach. Complete implementation and MATLAB® code of the function can be found in the appendix.

5.5 The technical indicator generators:

These functions generate Relative strength index (RSI), fast stochastic and exponential moving averages (EMA) required by our research. To calculate these values it is necessary to know the frequency or periodicity of data. This can be provided by using the integer argument known as `statP`. If the data contains 5 values in a week (weekdays only) we provide the value 5. For assets where data for all the 7 days is available, the user may provide 7. Thus instead of hard coding these generators to our application or data, these functions receive the statistical period (`statP`) of the data where the user can provide information regarding the nature/periodicity of the data passed to it.

`rsiGenerator:`

RSI or the Relative strength index is a famous technical indicator used for financial predictions [27]. We write this function to provide the RSI signal to our

neural network system, later this function shall also be used to analyze the performance of RSI based predictive systems. The detailed description of RSI and the associated formulas are provided under the Benchmarks section (6.2) of this document.

This function receives the mainSet data as an argument. As described earlier, the first two columns of mainSet provide two associated time series that serve as the exogenous inputs to the NARX networks. It also receives the `statP` namely the statistic period for the calculation. This is used to customize the RSI generator for the frequency and nature of the data provided. Here we are going to use the value 5 as our data has 5 entries per week for 5 working days. If 7 days data is available for an asset, it can be customized to it by using the value of 7. Different frequencies like weekly, monthly data etc can also be taken care of by using this parameter.

The detailed formulas for implementation can be found in the benchmarks section (6.2) and the complete implementation of the function can be found in the appendix

`fstGenerator:`

This function is used to generate the fast stochastic [28] of a given time series. The concept of stochastic oscillator and fast stochastic is described in section 6.2

It receives the `mainSet`, and `statP`. Last 4 columns of `mainSet` which are the open, high, low and close price of the asset to be predicted. These are used to generate the intermediate values like %K, %D etc [28]. These parameters are also discussed in section 6.2. The formulas and description for the implementations is available in the benchmark section. This function provides the following outputs

- KfstSet: Set of %K values for the time series
- DfstSet: Set of %D values for the time series (K smothered for 3 values)
- NfstSet: Set of %D values for the time series (K smothered for statP values - Not used)
- fstSet: Set of calculated fast stochastic

We use only first, second and the last values in our system. The third value was generated just as an add-on and is not a part of the standard definition of fast stochastic. Complete implementation and MATLAB® code of the function can be found in the appendix

`macdGenerator:`

This function was coded in order to generate the exponential moving averages (EMA) for 1 and 2 statistical periods (1 week and 2 week) of the provided data. Regarding EMA stockcharts.com [26] states that

> “In order to reduce the lag in simple moving averages, technicians often use exponential moving averages (also called exponentially weighted moving averages). EMA's reduce the lag by applying more weight to

> recent prices relative to older prices. The weighting applied to the most recent price depends on the specified period of the moving average".

The formula for moving average from the same source [26] is:

EMA (current) = ((Price (current) - EMA(prev)) x Multiplier) + EMA(prev)

Where "Multiplier" is equal to 2 / (1 + N) where N is the specified number of periods.

We generate and use EMA for 1 and 2 statistical periods – in our case 5 and 10 days. Complete implementation and MATLAB® code of the function can be found in the appendix.

5.6 Evaluation function:

`nseval:`

We will evaluate the signal of the network on the basis of several statistical measures in order to implement the strategies mentioned in section 4.5. It was decided to generate a dedicated function for this purpose – `nseval`. This function receives the Network Signals as inputs and performs statistical EVALuation of the quality of signal.

In this research we have decided to use the pattern recognition approach where the user shall invest according to "buy", "short" or "out of market" signals given by the network. If however a user never wishes to come out of the market he may continue to assume his previous "buy" or "short" signal whenever he gets out of

market signal. This policy however is not very safe and has not been tested. However we have provided the user of the function to use this strategy by setting the stratG input argument. We call such a strategy “never out of market” strategy as the user is always either in “buy” or “short” mode.

The function receives the following input arguments:

- `predSig` = the predicted signal of a network (e.g. ny)
- `masSig` = the massaged signal (e.g. testSetOut) (used only to calculate mse and mae)
- `actSig` = the actual signal values (e.g. testSetOutAct) (used to calculate rest of statistics)
- `tmargin` = defines the thresholds described in section 4.3
- `stratg` = defines the satrategy (0=quick trading-used in this research and explained in chapter 3 (Usage scenario) 1=never out of market - explained above)

The function performs various operations and returns the following results:

- npu = number of predicted ups (total buy signals generated)
- hu = upward hit rate (hit rate of buy signals)
- npd = number of predicted downs (total sell signals generated)
- hu = downward hit rate (hit rate of sell signals)
- npd = number of total predictions (total of buy and sell signals)
- ht = total hit rate

- netVal =net value at end of trade (realized value of $1 investment at end of test period)
- rmset = total rmse between predicted and actual preprocessed value
- maet = total mae between predicted and actual preprocessed value

MATLAB® uses a matrix based framework hence it is recommended to use matrix operations and avoid loops. In order to optimize performance we have used matrix operations instead of loops wherever possible. Complete implementation and MATLAB® code of the function can be found in the appendix

5.7 Performance optimization:

As discussed in the introduction section our approach in this research is that of generating several networks and selecting one network to make predictions in the test set.

Since several networks were to be evaluated, keeping the networks in memory would have occupied a lot of RAM space. To eliminate this problem each network was generated the results on evaluation set are stored and then killed. These results are stored in a two dimensional matrix. This two dimensional matrix is analyzed based on policies described in section 4.5 and 6.1 and a network is selected. The selected network is retrained using the same training data and architecture. (In this research we do not retrain the network again – we already have the result on the test set recorded. However if the system is to be deployed the actual network can be retrained easily)

In order to generate and evaluate all the specified approach we use batch files as specifies in the appendix. These batch files provide all the specified parameters to the `batchEl3` function. The `batchEl3` function uses the various supporting functions including the ones mentioned earlier and returns the 'report' on the validation and test data. It additionally returns the validation and test signals to allow the user to develop newer policies other than those mentioned in 4.5 and 6.1 if required in the future.

The batch file will call `batchEl3` in parallel loops thus allocating the task to different processors. Great speed ups can be obtained by using this approach. This was achievable because we store only the results and do not hold the networks in the memory and also because we are concerned with the final 'report card' and sequence of generation is not important – thus we need not wait for a particular architecture to complete its simulations before initiating the process for the next one. (All our files were not run on parallel mode due to server privilege issues). The chart below compares the times required by a batch of 40 networks under normal and parallel (4 processing modules) executions on a 175 Opteron processor 16 Gb RAM (Shared). The Opteron processor mentioned here has 4 cores and MATLAB® starts 4 processing modules which can simultaneously run processes that are not interdependent. Thus the tests for parallel execution were run on a quad core server and used the distributed computing toolbox of MATLAB®. The time required is reduced to about one third (not one fourth

because of the processing overheads). Additional speedups can be obtained by using more than 4 labs (MATLAB® processing modules – see glossary).

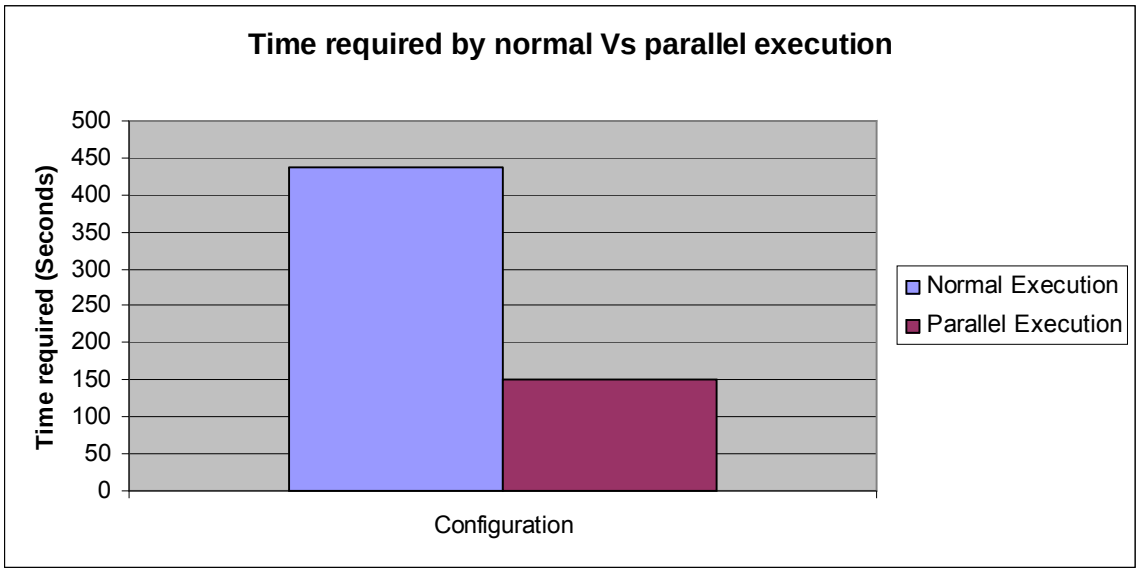

Figure 11: Comparison of times required by normal and parallel (4 labs) execution

This is followed by execution of evaluation files which analyze the evaluation set results as per the policies mentioned in 4.5 and 6.1 and return us a report stating what results would have been received if those policies were adopted.

5.8 Function interaction, integration and execution

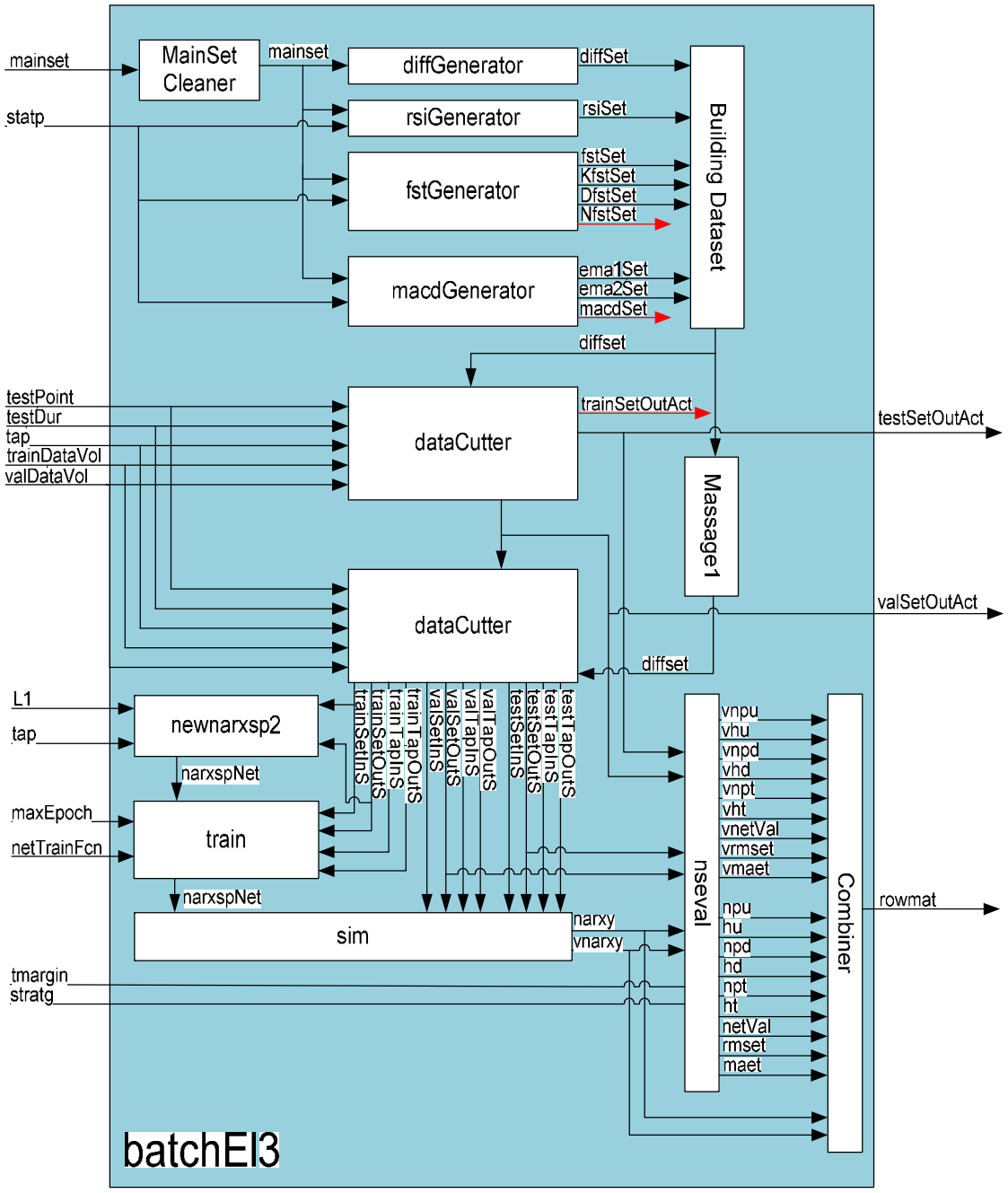

Figure 12: Integration and Integration diagram for the batchEl3 – The master function that generates, trains, simulates and evaluates neural network.

The functions mentioned in the previous chapters are integrated first into the function `batchEl3`. The diagram below represents this function and all the sub functions that are called during its execution. Each of the blocks represents a function (Except building a dataset and combiner – they are single line MATLAB® commands). Arrows pointing to a block are the arguments that it receives. Arrows that come out of it represent the outputs. The functionality/algorithm of these individual functions and that of `batchEl3` has been explained in the implementation chapter. The sub functions in the diagram are called from the top to the bottom. The arrows additionally help in interpreting the sequence. Codes of each of these functions are available in the appendix.

The function `batchEl3` as the name suggests is an element of a batch. This function can be passed arguments including the number of taps and neurons and receive a report of the behavior of that network. We can then use files like `batch2_06.m` to set up for loops that can help us try out all the permutation and combinations of number of taps and the number of neurons. The actual code can be found in the appendix. Such a batch file helps us define a range for number of taps and number of neurons. Running the `batchEl3` within the loops as shown in the code allows us to generate a report matrix which we save as `.mat` files. Such a report matrix is then processed by functions that select a network based on our policies and displays its behavior on test set. `B2res` is one such function that implements the policy 4 that is described in section 6.1 in the results section (Chapter 6). This is the policy which we finally select for our application.

The files can be executed in any standard MATLAB® installation that has the Neural Network Toolbox®. The execution procedures for executing the files used for this research are as follows:

The sequence of events explained above can be summarized as follows:

1. The batch file like `batchCombined` is executed.
2. `batchCombined` calls batchEl3 for all combinations in the predefined range
3. Each `batchEl3` performs the following actions
    a. `batchEl3` calls `mainSetCleaner` to clean the data
    b. `diffGenerator` is called to generate the returns series
    c. `rsiGenerator` is called to generate RSI, %D and %K
    d. `fstGenerator` s called to generate fast stochastic
    e. `macdGenerator` is called to generate moving averages
    f. Data is organized
    g. The data is preprocessed using the function `masage1`
    h. `dataCutter` is called to generate the required datasets for training, evaluation and testing
    i. Neural network for the specified configuration is generated using the function `newnarxsp2`
    j. The network is trained using the `train` function
    k. Network behavior is simulated on the evaluation and test sets using the `sim` function
    l. The simulated behavior is evaluated using the `nseval` function

m. All the outputs are combined returned to `batchCombined`

4. Values returned from several such `batchEl3` executions are stored in a two dimensional matrix.
5. This matrix is analyzed by function like `B2res`. This function returns us results that would have been obtained on the test set by following policy 4 (Se section 4.5 and 6.1)

Windows/Linux GUI:

Type in `BatchAll` in the command window of MATLAB®. This will execute the `batch2_06, batch2_07` and B2res sequentially. As the networks are being generated and simulated, the screen will display their number of taps, neurons etc to serve as progress indicators. Finally the results of the simulation will be available on the screen.

UNIX MATLAB® command prompt:

Typing in `matlab` at the command prompt of UNIX brings up the MATLAB® command prompt. The same procedure discussed above for windows GUI can be followed here.

UNIX Server:

On Unix server you cannot use the `BatchAll` as it points to three different files. They can be combined into a file like `batchCombined` which is simply the

concatenation of the code contained in three files. The other option is to use a bash script. The combined file can be executed as follows:

Type `matlab <BatchCombined.m> batchComb.out` and on the command prompt

`matlab` indicates that this process is to be executed by the MATLAB® engine. `<BatchCombined.m >` specifies the source file which is to be executed in this process. `batchComb.out` is the name of the output file where the output is to be redirected and indicates that this process is to be put in the background thus returning the prompt to allow the user to do other activities.

Codes for all the files and functioned in this chapter are available in the appendix.

Chapter 6: Results

6.1 Comparison of network selection policies

Instead of adhering to a single architecture we are going to define a range of taps and neurons and going to train and test networks all permutation and combinations of taps and neurons within this range.

Taking into consideration the need for adaptability and the resources available we decide to use taps up to 4 (4 taps would mean last 4 days behavior directly related with the 5^{th} day – we have a 5 day week) and up to 80 neurons. This would give us 320 possible networks for each dataset.

A decision needs to be made how frequently the system is going to update its architecture. High frequency will ensure that all the latest data has been accounted for in the model however will require large computational capabilities.

Not changing a model for a long time period will cause the system to utilize less recourses however will not be very efficient as it has been trained and evaluated on data that is very old.

Looking at our computing constraints, we decided to train a network on the training set (16 months in past to 4 months in past of test set) evaluate it over an evaluation set (past 4 months from test set) and finally present results on the test set (4 months).

Networks were generated and trained over the specified range. Now it was necessary to set up rules that would help us determine networks that perform well on our future (test) sets.

We tried out three policies mentioned in section 4.5 – Policies for network selection. They are as follows:

- Policy 1: We will select a network that has minimum error on the evaluation set to make predictions on the test set (future)
- Policy 2: We will select a network that has maximum realized value on the evaluation set to make predictions on the test set (future)
- Policy 3: We will select a network that has maximum hit rate on the evaluation set to make predictions on the test set (future)

If each of the following policies were to be followed, we would get the following performance on various datasets.

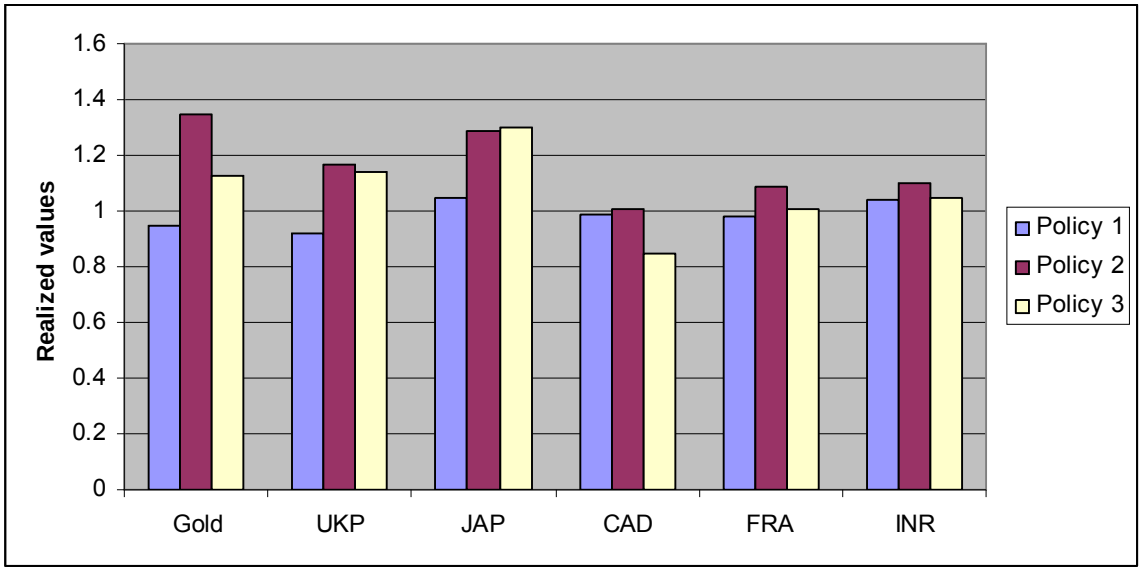

Figure 13: comparison of realized values on the six assets by using Policy 1, 2 and 3

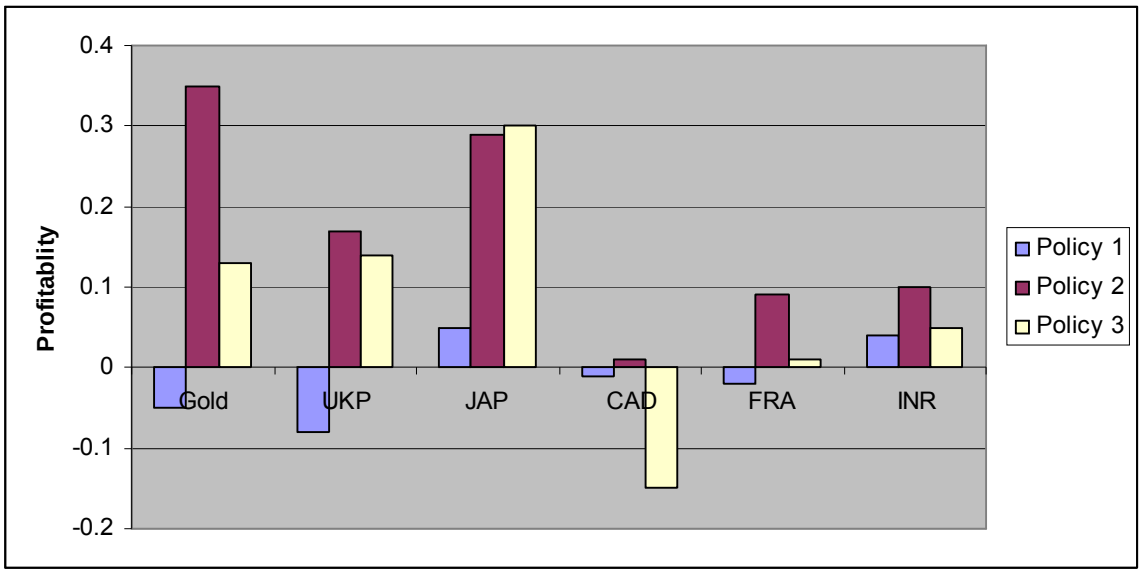

Figure 14: comparison of profits on the six assets by using Policy 1, 2 and 3

Looking at the realized returns and the profitability we see that only the past realized returns are a fair indicator of the expected behavior on the test set. Behavior of hit rate is better than error, but still lagging to realize value.

Thus traditional statistical measures like error and Hit rate are not a very good measure for comparison or indicator for future behavior of a network .Kaastra and Boyd [15] have also stated that low error and profits are not synonymous in trading.

Results show that realized value on the evaluation data is the best indicators of behavior on the test data. However there are still some problems with this approach. Since we are selecting the network with the maximum returns on the evaluation data, we at times get networks which have made huge profits on the evaluation sets – but are not generalized well. This means that they may make huge profits on a certain section of the evaluation data and not fit well on one part at all. Such networks if selected plainly considering their profit would not guarantee good behavior on the next set.

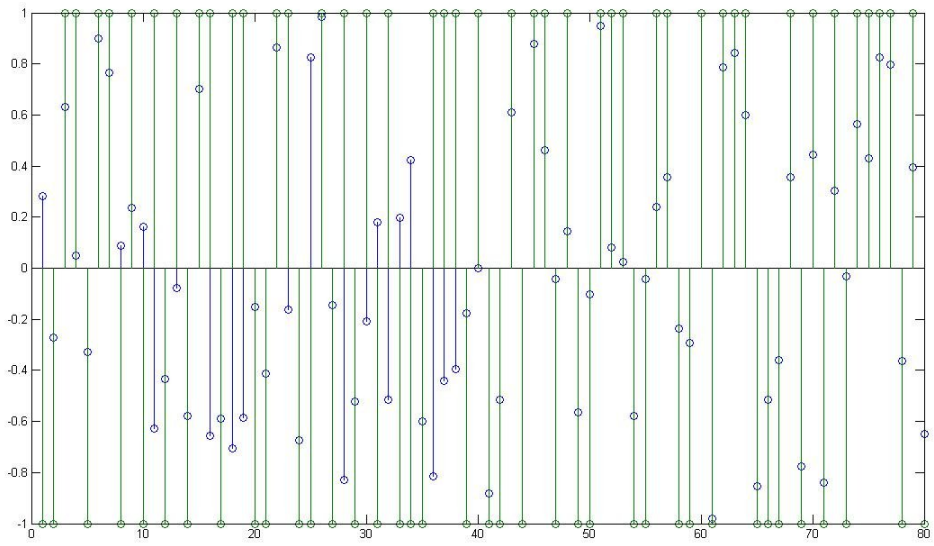

Figure 15: A sample representation to demonstrate a possible non-generalized output that makes large profits

In this diagram, blue dots represent the actual market movement, Green lines and dots represent the predicted signal. Wherever we observe both the dots on the same side, the network makes a profit and vice versa.

This output is very good on the later half of the evaluation set; however the predictions on the first half are random. Thus this network would make large profits, yet not be a good network to select for future predictions.

The following sketch is a sample representation of a network output that would make high profits in one section and not much on the other. Such a network would be selected to make predictions for the future, but would not give great results on the test set.

In order to eliminate such behavior we decided to split the evaluation set into two halves. Each of the networks in the set of 320 would receive a rank between 0.99 and 0 depending on its performance. The best performing network would receive 0.99 and the worst performing would receive 0 for each of the halves. We call the ranking policies based on these ranks as ‘Percentile Ranking Policy’.

- Policy 4: Rank the networks on the basis of realized value on both the halves of evaluation set and add up these ranks. Select the network with maximum total to make predictions on the testing set.
- Policy 5: Rank the networks on the basis of realized value and hit rates on both the halves of evaluation set and add up these ranks. Select the network with maximum total to make predictions on the testing set.
- Policy 6: Rank the networks on the basis of realized value, upward hit rates and downward hit rates on both the halves of evaluation set and add up these ranks. Select the network with maximum total to make predictions on the testing set.

The realized values and profits earned by following each of the generalization rules are:

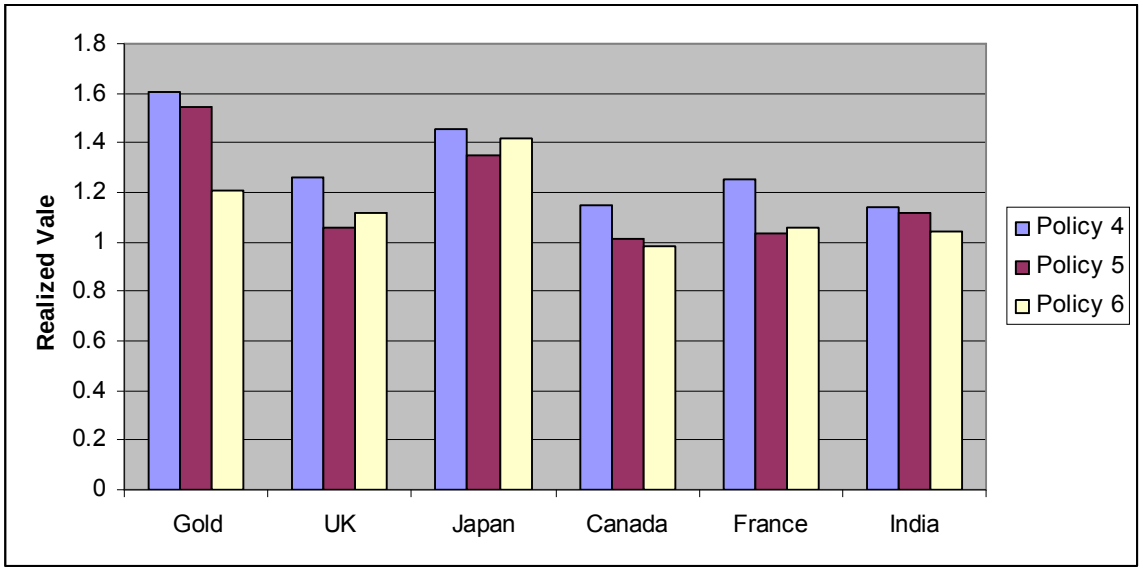

Figure 16: Comparison of realized value the six assets by using Policy 4, 5 and 6

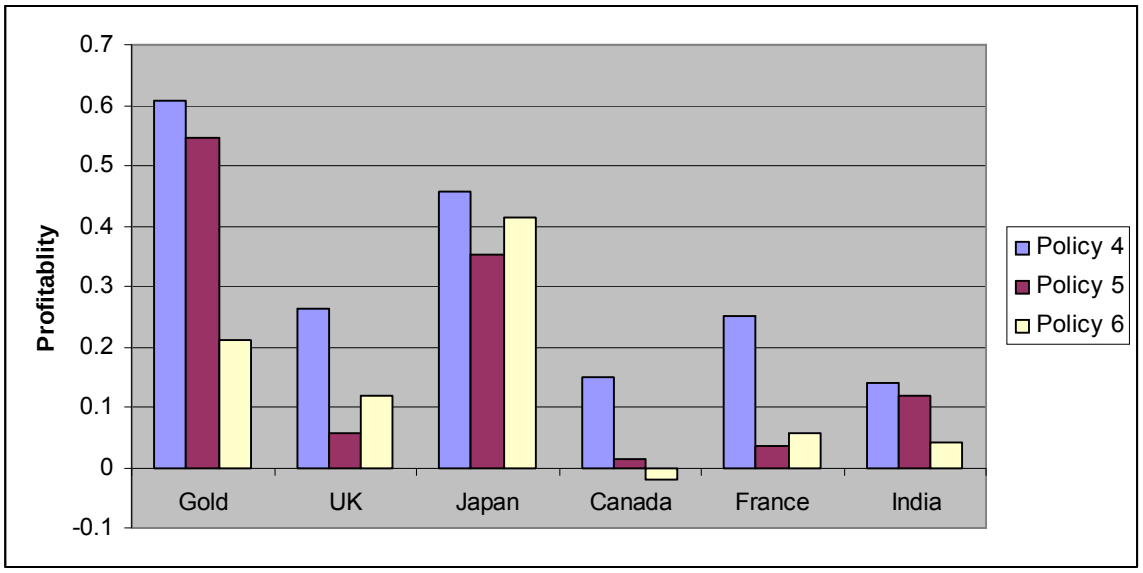

Figure 17: comparison of profits on the six assets by using Policy 4, 5 and 6

Thus we observe that using the percentile ranks of realized values alone was sufficient. Adding hit rates does not improve the results and makes them unstable. Sufficient generalization was achieved by splitting the evaluation set into two. Hence jobs with higher number of sets for generalization were not run..

The individual tables displaying the performance at intermediate stages are available in the appendix.

6.2 Benchmarks:

6.2.1 Performance criteria:

As observed earlier, Thus traditional statistical measures like error and Hit rate are not a very good measure for comparison or indicator for future behavior of a network .Kaastra and Boyd [15] have also stated that low error and profits are not synonymous in trading. Since we are developing a neural network based system for trading its true performance needs to be observed in the form of financial terms and not statistical error etc. It is finally the money making ability of the network that is going to serve as its final test. Hence we are going to compare the realized value - The total value of investment (1$) at the end of all test periods or profitability – The total profit earned at the end of all the periods at the end of all test sets (on an investment of 1$)

6.2.2 Benchmarks systems:

6.2.2.1 Linear regression:

Since we use neural networks (which are non-linear regression models), we need to compare their performance with standard linear models to justify the need for non linear models. We will use the same data as used by the neural

network for this linear model. This linear model will be used to make predictions just like our network. The targets for the linear system for training will be the return series. The inputs will be the same inputs like our network we use the standard function ‘regress’ to predict the values. Our very same train set is used to set the regression weights, and this regression function is used to generate predicted values. We use these values to generate buy and short decisions.

The code and implementation details can be found in the appendix.

6.2.2.2 Technical Indicator – Relative strength Index:

Relative strength index (RSI) is a famous technical indicator for time series analysis

According to investopedia.com [13] RSI is defined as

> “A technical momentum indicator that compares the magnitude of recent gains to recent losses in an attempt to determine overbought and oversold conditions of an asset”

It is calculated by the formula

RSI = 100 – (100/ (1+Rs))

The formulas are also available on stockcharts.com [27]. Stockcharts.com states

> “Developed by J. Welles Wilder and introduced in his 1978 book, *New Concepts in Technical Trading Systems*, the Relative Strength Index (RSI) is an extremely useful and popular momentum oscillator. The RSI compares the magnitude of a stock's recent gains to the magnitude of its recent losses and turns that information into a number that ranges from 0 to 100. It takes a single parameter, the number of time periods to use in the calculation.”

Stockcharts.com provides the formulas for calculation for Rs and related terms. We have used those formulas to calculate the RSI indicators, these formulas

have been customized to our 5 day week data instead of 7 day week. The neural network uses these RSI values as inputs; hence we should compare the performance of plain RSI system and the neural network approach.

The code and implementation details can be found in the appendix.

6.2.2.3 Technical Indicator – Fast Stochastic:

Fast stochastic is also a very famous technical indicator. Stockcharts.com [28] states

> "Developed by George C. Lane in the late 1950s, the Stochastic Oscillator is a momentum indicator that shows the location of the current close relative to the high/low range over a set number of periods. Closing levels that are consistently near the top of the range indicate accumulation (buying pressure) and those near the bottom of the range indicate distribution (selling pressure)."

The website also provides formulas for calculating the individual parameters like %D, %K etc. Explanations for these factors can be found in section 5.5 and glossary.

% K is calculated as

100 * ((Recent close - Lowest low) / (Highest high – Lowest low))

%D is calculated as 3 period moving average of %K

We have implemented the basic oscillator that uses the difference between these parameters to generate signals. We feed our network with this signal. We will also benchmark our system against this stochastic oscillator.

6.2.2.4 Neural Networks systems:

Best case fixed architecture hypothetical NARX systems:

We compare the performance of our system with the performance of fixed architecture neural networks that adhere to the specified thumb rules. As per the discussion above, we will compare our dynamic system with NARX systems which have 1, 5, 20, 80 neurons. The taps of these systems will be the taps that have given best performance in the test sets with that number of neurons. Please note that these numbers of taps have been determined by analyzing the test sets and were not known before simulation on the test set. Hence we consider these systems as best case fixed architecture hypothetical systems.

Fixed architecture static systems:

To evaluate the performance against fixed architecture static systems (no delay lines) we simulate the behavior of 1, 5, 20, 80 neuron networks which have only single past value (no time series delay line). Comparison with these systems will show us if it is beneficial to use our system as compared to static systems.

6.2.2.5 Financial Baselines:

To examine the financial validity of our model w e will compare the performance with financial baselines like all buy (If a buyer had bought that asset for the entire duration what would the realized value be) and all short (If a buyer had short that

asset for the entire duration what would the realized value be). We will compare our performance with the outcomes of these financial policies.

6.3 Performance analysis:

We hereby present the results of our experiments executed by using the methodology and framework mentioned above.

In table 2 we compare the performance of our system with the various benchmarks mentioned in the benchmarks section over all the test sets spanning duration of two years.

	NARX	All Buy	All Short	fst	RSI	Lin	N1	N5	N20	N80	F1	F5	F20	F80
Gold	1.61	1.53	0.47	1.55	1.06	0.57	0.85	1.32	1.25	1.23	0.82	1.28	1.21	0.61
UK	1.26	1.13	0.87	0.92	1.06	0.90	0.98	1.08	1.20	1.00	0.99	0.89	1.20	0.97
JAP	1.46	1.16	0.84	1.02	0.95	1.08	1.26	1.28	1.21	1.15	1.33	1.19	1.07	0.92
CAD	1.15	1.13	0.87	1.06	0.92	0.82	0.94	0.97	1.09	1.09	1.07	0.93	1.04	0.99
FRA	1.25	0.78	1.22	0.82	0.81	0.92	0.83	0.99	1.01	1.09	1.02	0.88	0.98	1.07
IND	1.14	0.89	1.11	0.91	0.92	1.05	1.03	1.02	1.06	0.97	1.04	0.98	1.06	0.97

Table 2: Comparison of realized values by using various predictive systems

The table on the previous page (Table 2) gives that realized value at the end of two years by the use of various strategies. The tables showing period wise progressions and returns of individual periods are available in the appendix.

We have plotted the period wise progression of realized value in the form of 3D charts just to compare the performance with different systems. Different charts have been generated for each of the asset.

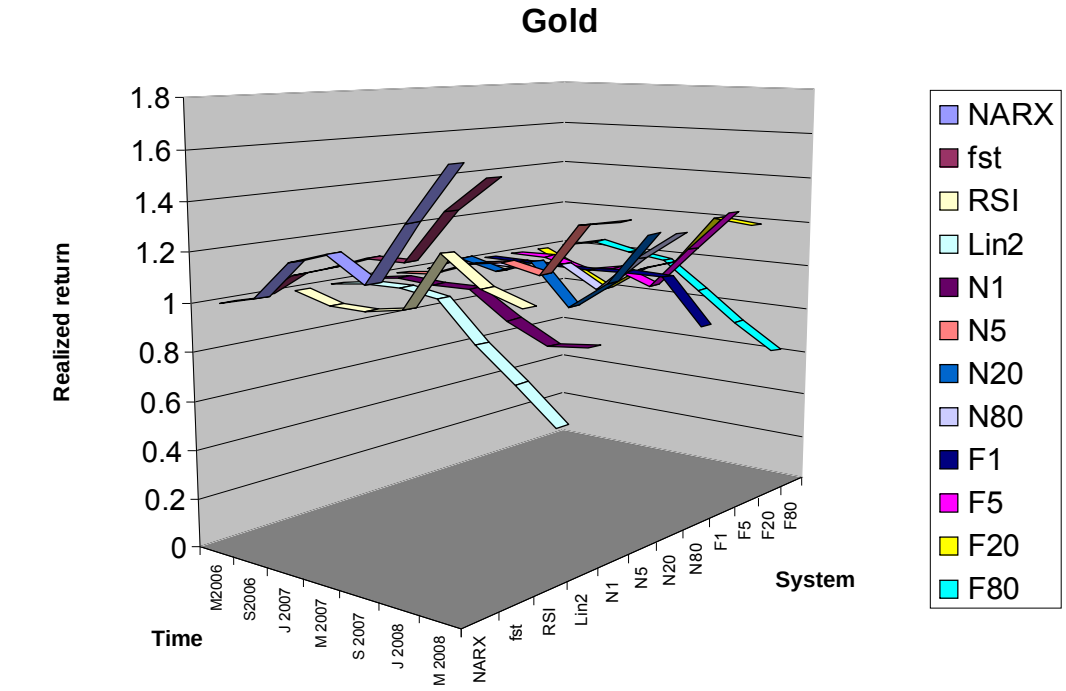

Figure 18: Period wise realized value for the investment in gold.

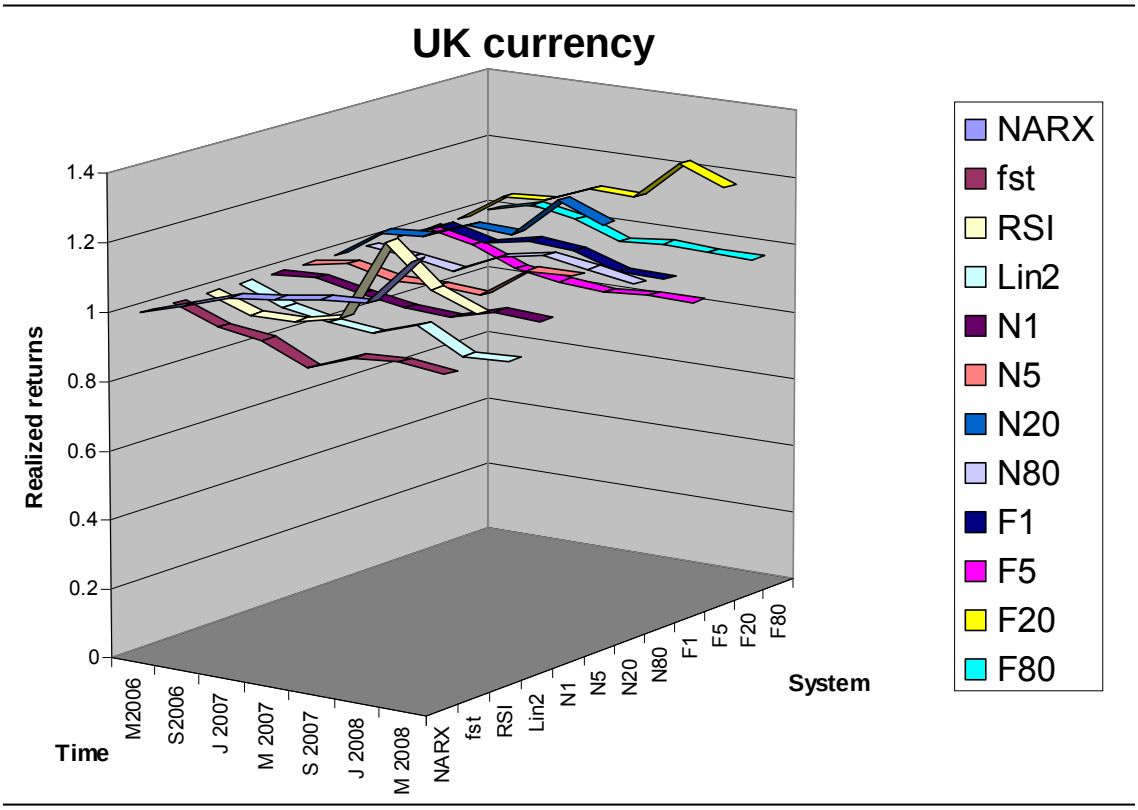

Figure 19: Period wise realized value for the investment in UK currency.

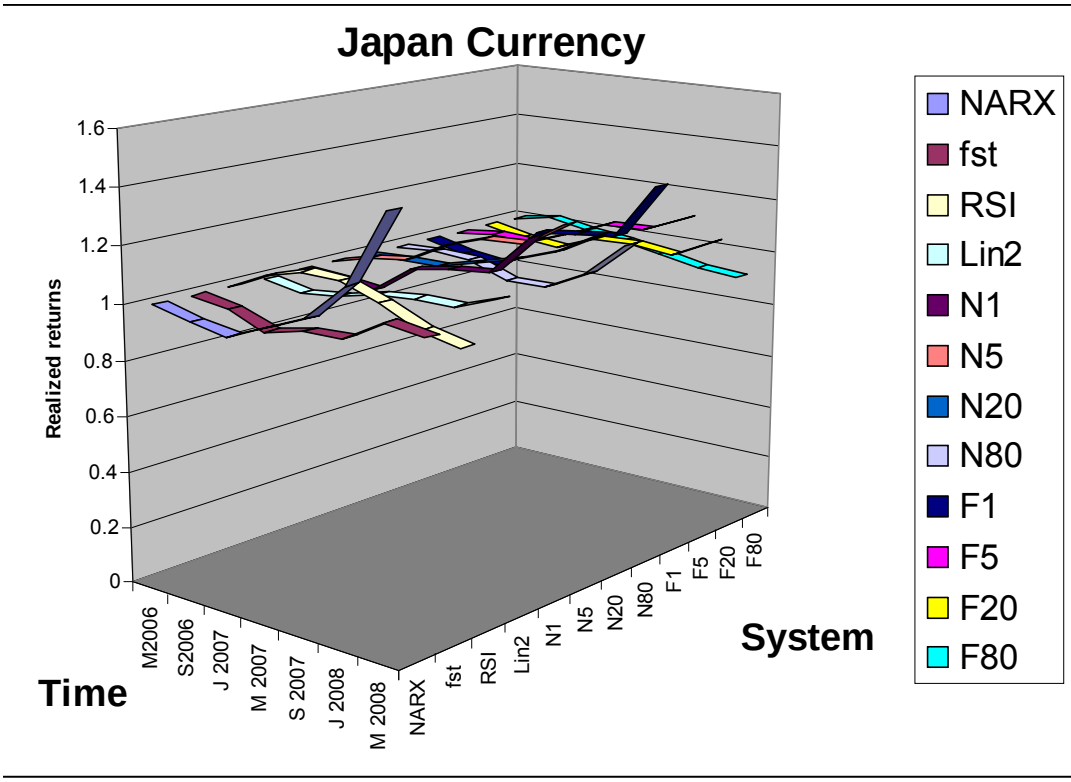

Figure 20: Period wise realized value for the investment in UK currency.

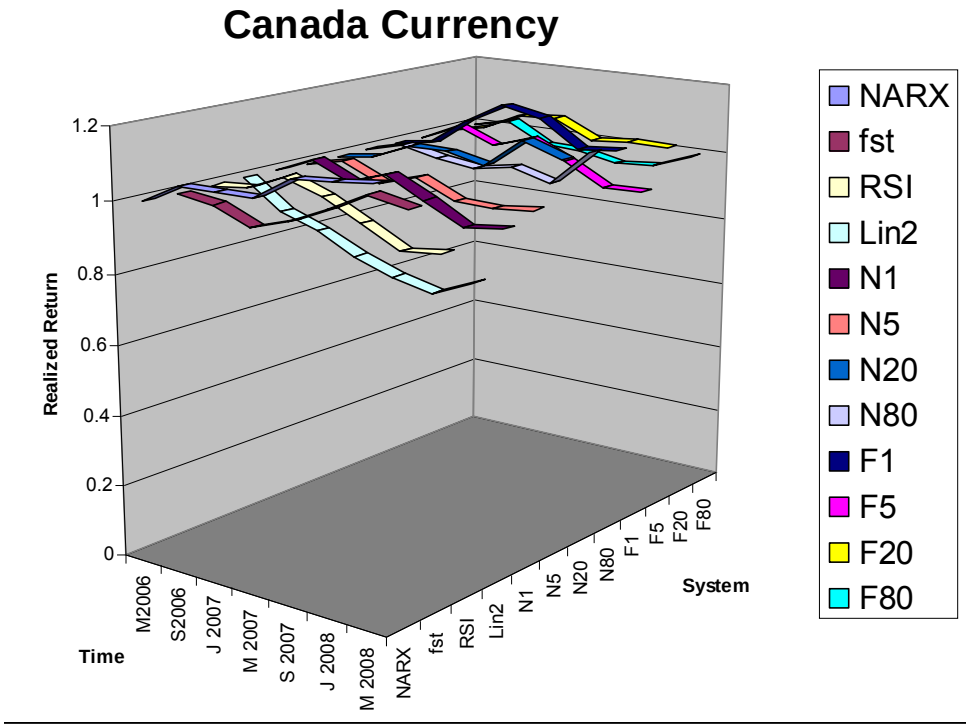

Figure 21: Period wise realized value for the investment in Canada currency

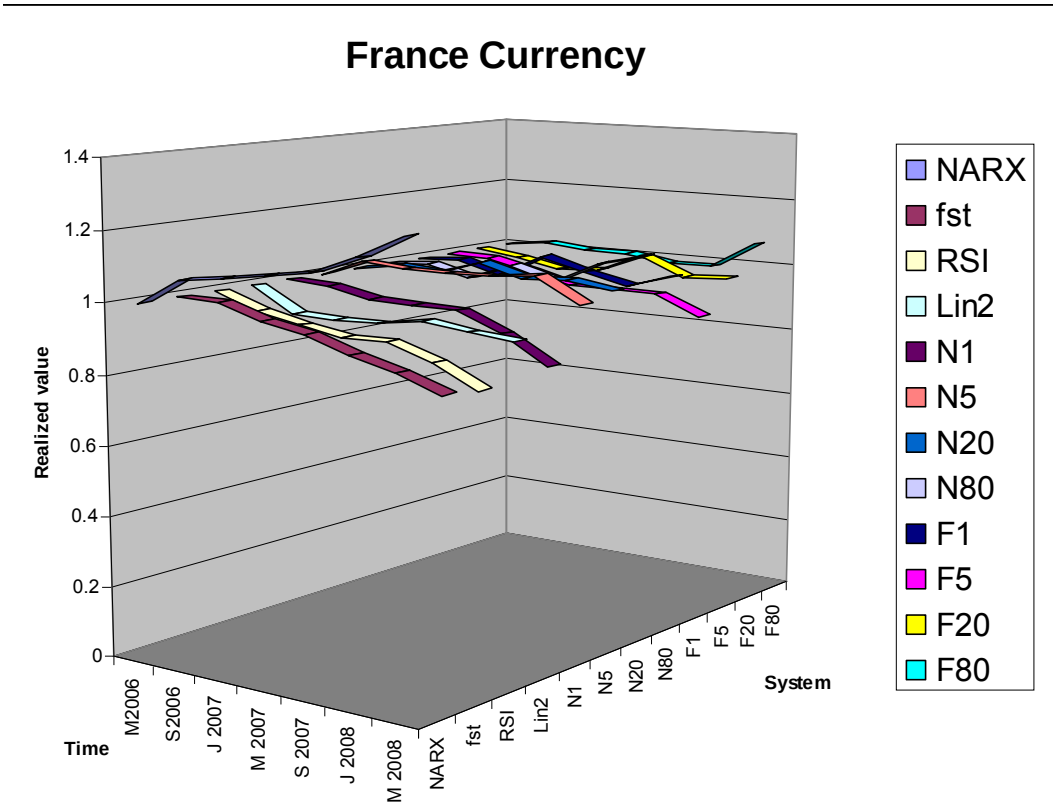

Figure 22: Period wise realized value for the investment in France currency.

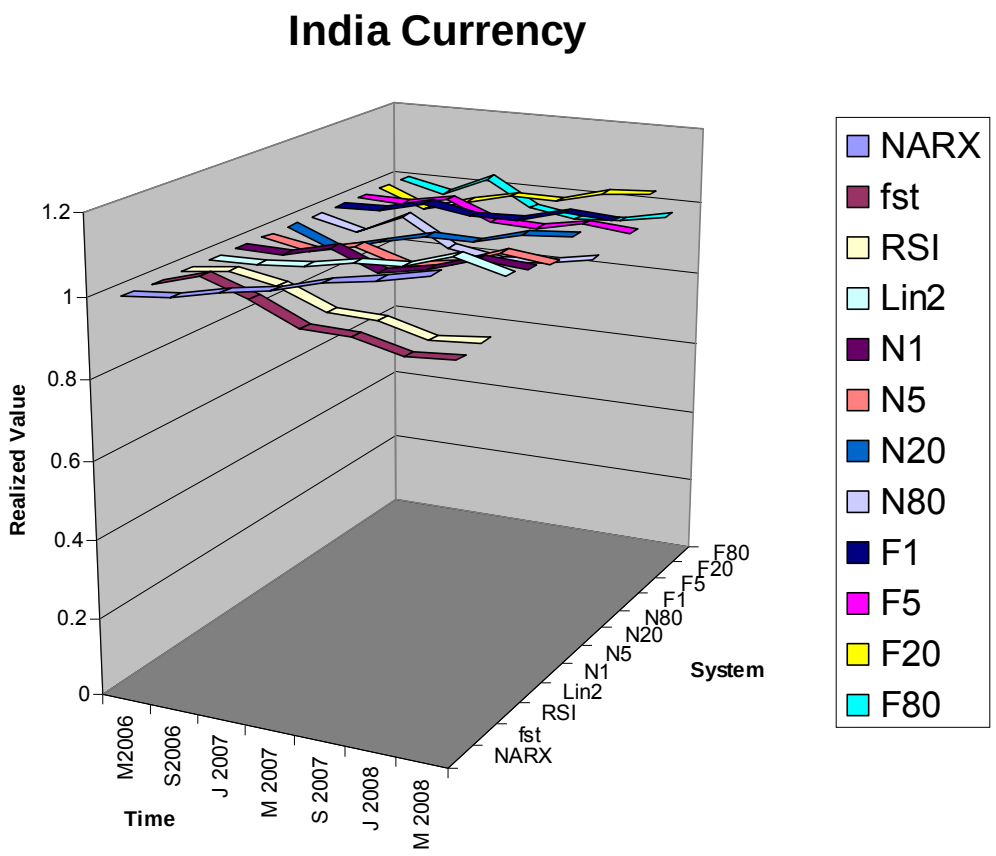

Figure 23: Period wise realized value for the investment in India currency.

The final realized investment of each of the systems is presented asset wise in the graphs below

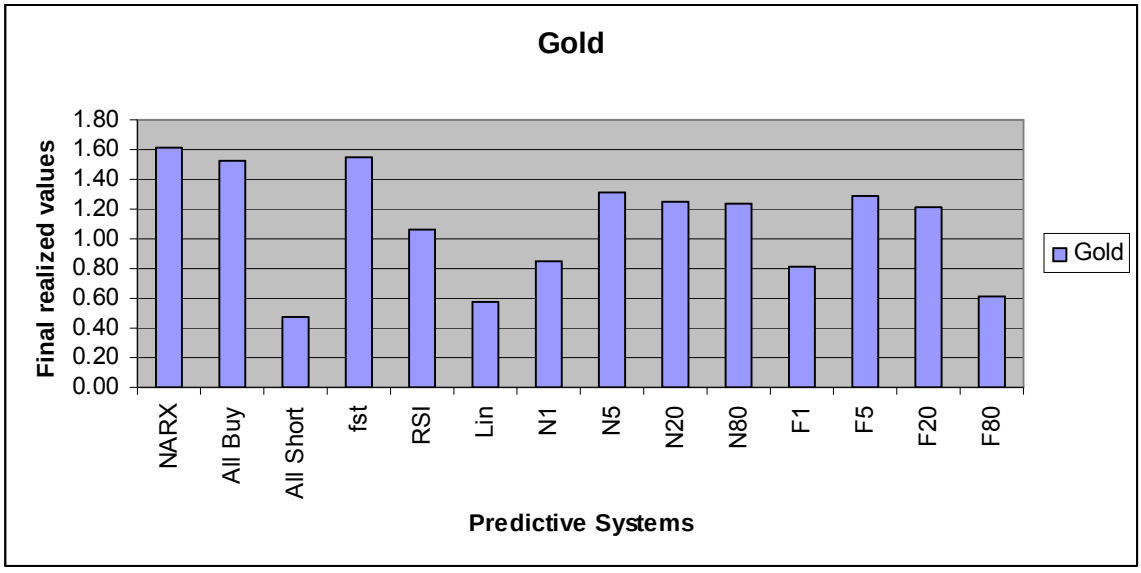

Figure 24: Final realized values in Gold investment, across various systems

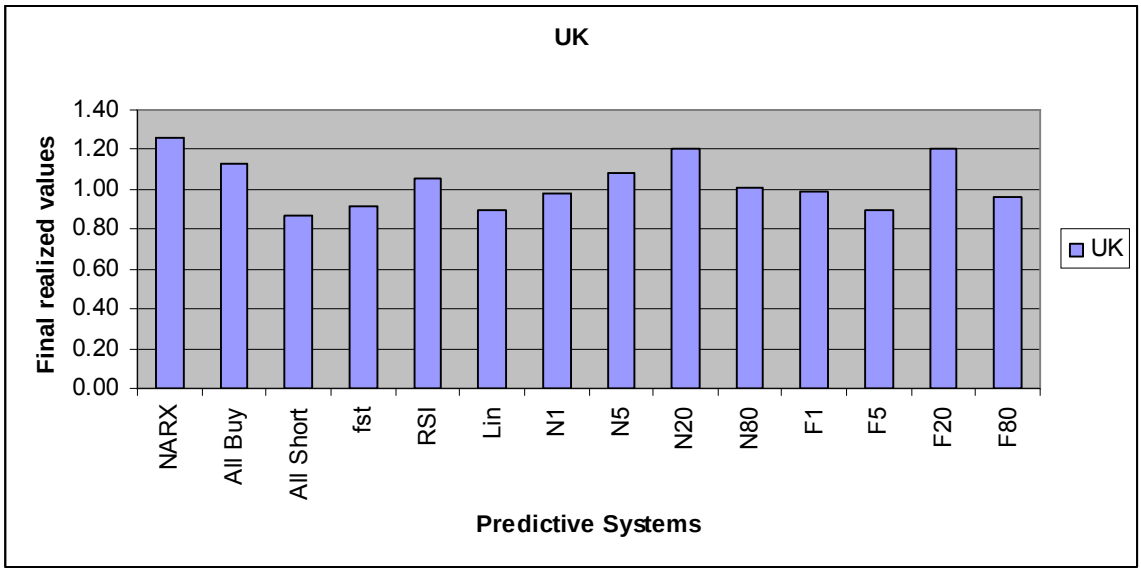

Figure 25: Final realized values in currency of UK, across various systems

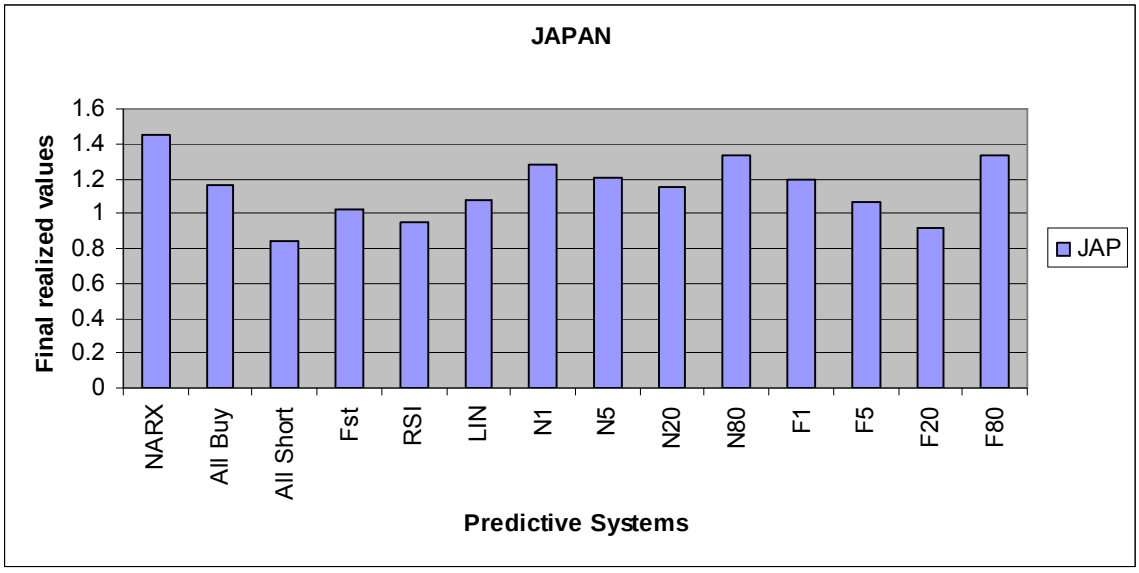

Figure 26: Final realized values in currency of Japan, across various systems

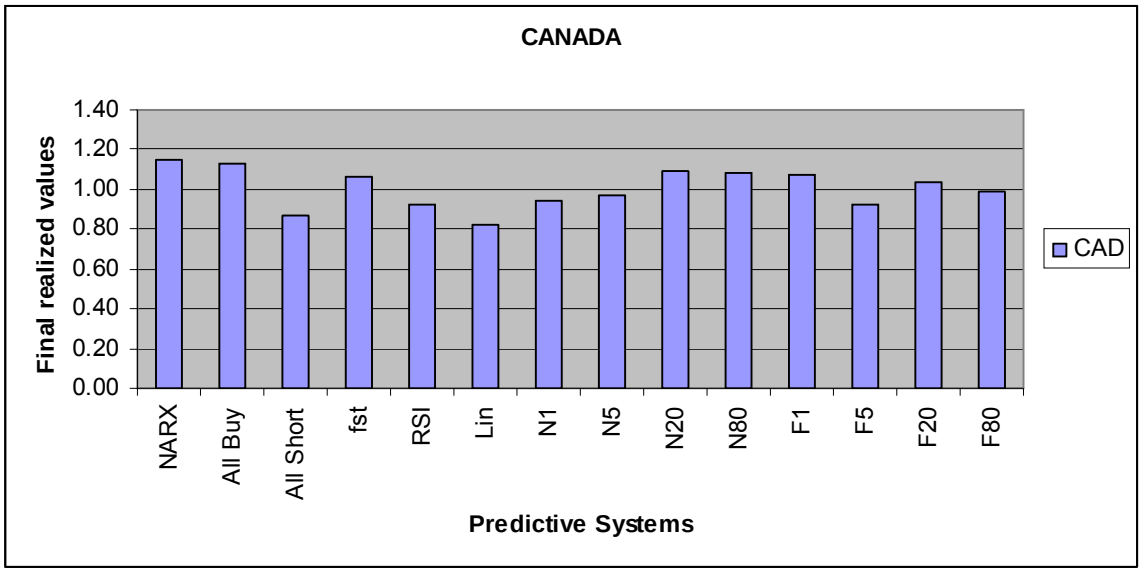

Figure 27: Final realized values in currency of Canada, across various systems

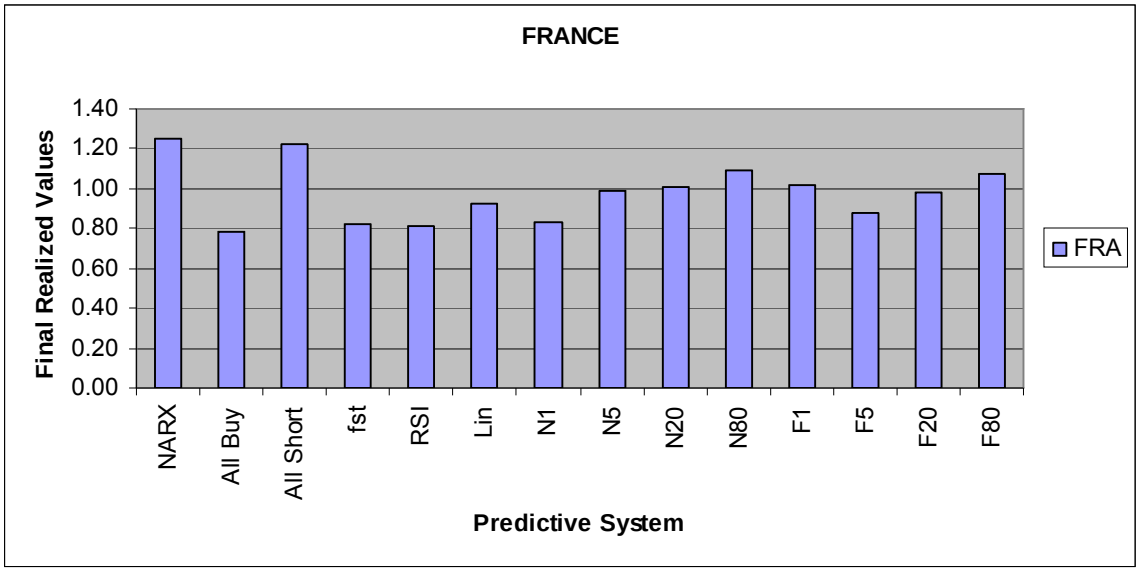

Figure 28: Final realized values in currency of France, across various systems

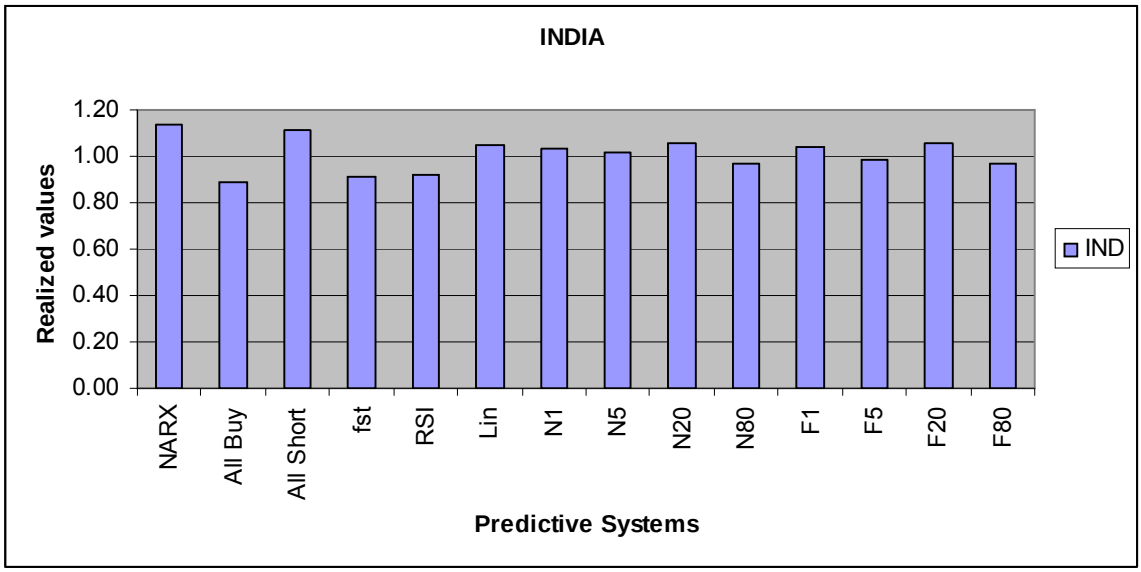

Figure 29: Final realized values in currency of India, across various systems

In order to give an overview of performance, we consider an investment whereby one sixth of the initial investment (1$/6) was invested in each of the systems. The

final total realized value by following each of the following systems would be as follows:

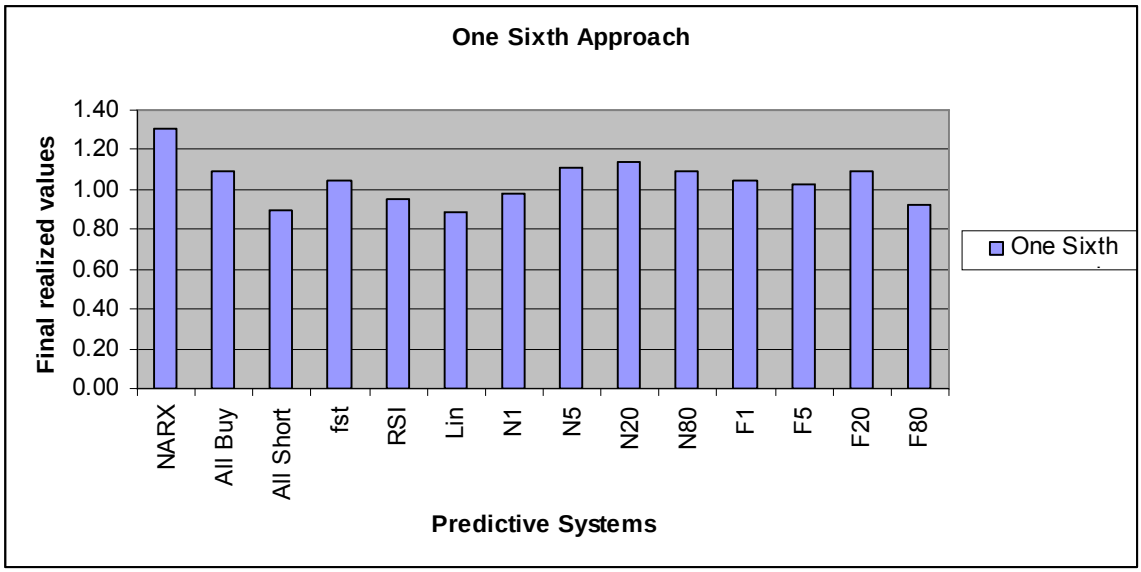

Figure 30: Final realized values by investing an all 6 assets, across various systems.

By looking at the above charts it is clear that the NARX based adaptive system that we presented performs better than the other benchmark systems that we defined earlier.

While comparing with financial baselines like the All Buy and All Short it should be noted that these systems invest your money in the market for all 80 days in each of our test set. Whereas our NARX based system roughly gives out of market signal for 10 days in each test set. The individual figures can be found in the table in the appendix.

I we were to equate the returns of our system for 80 days of investment, the comparison of the NARX system and financial baselines would look as follows:

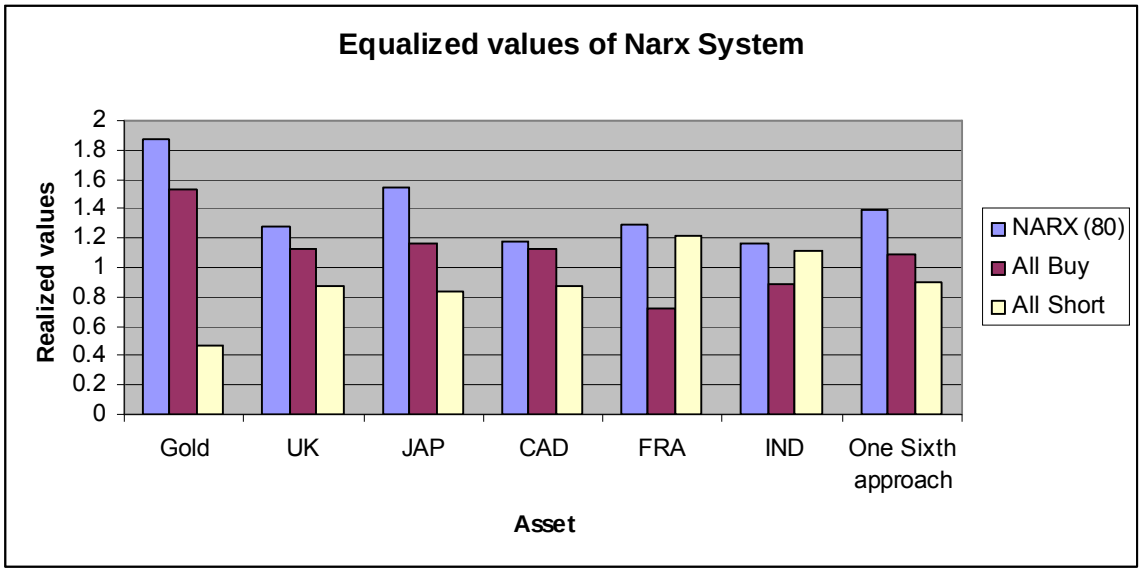

Figure 31: Comparison of 80 day equalized values of NARX system with financial baselines

This supports and validates our claim that NARX systems hold financial viability and perform better than the baselines.

The individual period wise tables in the appendix demonstrate that the NARX system selected networks with different number of taps and neurons whenever possible. Thus the system found different combinations to be best suited for the data at different time frames. Lower numbers of neurons are required when less computation is required to establish a pattern and more neurons are required when more computation is required. Thus it is believed that large numbers of neurons refer to a time frame where the underlying data is very complex. Future

work will involve associating the number of neurons selected with parameters like volatility.

6.4 A word of caution:

As mentioned in section 5.2 faulty time series or a time series with random behavior might lead to losses. Though in our simulation set there has never been sudden random behavior, in case of extreme conditions like war etc. it is possible that the time series behaves randomly. It is hence advisable that the user of the system has some basic knowledge in the field of finance. This system is to be preferably used as an advisory tool and the results should not be followed blindly.

Chapter 7: Conclusion and future work

7.1 Conclusion

We had focused our research on developing neural network based methodology which could earn profits via trading on different assets. As opposed to the standard static approach we decided to follow a strategy where the system could select a network from amongst multiple trained networks and use them in the predictions of the future sets.

The results that we have presented in the section above allow us to draw the following inferences.

The NARX based adaptive strategy is financially viable and outperforms the financial baselines.

The NARX based adaptive strategy outperforms standard implementations of technical indicators like fast stochastic, relative strength index, and linear regression

The NARX based adaptive strategy that we proposed performs better that the usage of a fixed architecture NARX system and non dynamic feed forward networks

Using NARX was beneficial as it allowed us to alter input taps as well as neurons. Simulations have revealed that an adaptive feed forward network that varies the number of neurons alone [14] would not have performed as well as

NARX system that we proposed. This can be inferred from last three tables in the appendix.

The results of this system should be monitored and not be followed blindly. Our experience on the corrupted data series mentioned earlier demonstrates possible faults. Hence they should be decision making tools and not final decision makers.

7.2 Future work:

Portfolio optimization: Rules can be derived to identify which of the generated signals is more reliable and invest on that asset in portfolio of multiple assets. Also the system is out of market about 1/8 th period – adding opportunities for portfolio optimization.

Generating multiple networks and Generalization rules for networks with different thresholds, Multilayer architectures, large number of taps, multiple exogenous inputs – this effectively implies pushing the boundaries of current research.

Using second set of networks for network selection – investigation into various complex statistical systems or neural network based systems to select a neural network from the set of pre-trained networks.

Improving performance of current system if possible by fusing various input selection methodologies.

References:

1) D. Barker. “Analysing financial health: Integrating neural networks and expert systems”, PC AI, 1990

2) E. Collins, S. Ghosh, C. Scofield, “An application of a multiple neural network learning system to emulation of mortgage underwriting judgments”, 1988., IEEE International Conference on Neural Networks, 1988

3) G. Cybenko, “Approximation by superpositions of a sigmoidal function”. Mathematics of Control, Signals, and Systems (MCSS) – Springer, 1989

4) H. Demuth, M. Beale, M. Hagan,”MATLAB Neural Network Toolbox 5, Users Guide”, 2007

5) C. Dunis, M. Williams, “Modeling and Trading the EUR/USD Exchange Rate: Do Neural Network Models Perform Better?”, Liverpool Business School and CIBEF, 2002

6) G. Dutta, P. Jha, A. Laha, N. Mohan, “Artificial Neural Network Models for Forecasting Stock Price Index in the Bombay Stock Exchange”, Journal of Emerging Market Finance, 2006.

7) S. Dutta, S. Shekhar, “Bond rating: a nonconservative application of neural networks Neural Networks, 1988.” IEEE International Conference on neural networks, 1988

8) Fadalla, L. Chen-Hua, “An Analysis of the Applications of Neural Networks in Finance”, Interfaces - Volume 31 , Issue 4 (July 2001) ISSN:0092-2102

9) C. Gallo, “Artificial Neural Networks in Finance Modelling”, ideas.repec.org, 2005

10) S. Ghosh and C. Scotfield, “An application of a multiple neural-network learning system to emulation of mortgage underwriting judgments,” in Proc. IEEE Conf. Neural Networks, 1988.

11) K. Hornik, “Approximation capabilities of multilayer feedforward networks”, Neural Networks - portal.acm.org, 1991

12) K. Hornik, Stinchcombe et al., “Multilayer feedforward networks are universal approximators”, ACM, 1989

13) investopedia.com, “Relative strength index (RSI)”, at: http://www.investopedia.com/terms/r/rsi.asp

14) Java Object Oriented Neural Engine, “The JOONE financial forecast tutorial on Joone wiki”, at: http://www.jooneworld.com/ (exact author/link cannot be provided as the Wiki is now unavailable)

15) Kaastra, M. Boyd, “Designing a Neural Network for Forecasting Financial and Economic Time Series”, Neurocomputing, 1996.

16) CC Klimasauskas, “Applying neural networks. Neural Networks in Finance and Investing”, 1993

17) RA. Marose, “A financial neural-network application”, AI Expert, 1990

18) L. Marquez, T. Hill, R. Worthley, W. Remus, “Neural network models as an alternative to regression”, Proceedings of the 24th Annual Hawaii International Conference on System Sciences, IEE Comput. Soc. Press, Kauai, Hawaii, Vol. 4 pp.129-35.1991

19) P. Marrone, Java Object Oriented Neural Engine, at: http://www.jooneworld.com/

20) P. Marrone, “JOONE complete guide”, at: http://www.jooneworld.com/

21) M. Mehta, “Foreign Exchange Markets”, 176-198, Neural Networks in the Capital Markets, John Wiley, Chichester, 1995.

22) MD. Odom, R. Sharda, “A neural network model for bankruptcy prediction”, 1990 IJCNN International Joint Conference on neural networks, 1990.

23) W. Raghupathi, L. Schkade, BS. Raju, “A neural network approach to bankruptcy prediction”, Neural Networks in Finance and Investing, 1991

24) E. Rahimian, S. Singh, T Thammachote, R Virmani. “Bankruptcy prediction by neural network”, Neural Networks in Finance and Investing, 1993

25) R. Sharda, RB. Patil, “Connectionist approach to time series prediction: an empirical test Journal of Intelligent Manufacturing”, Springer, 1992

26) stockcharts.com, “Moving averages”, at: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_averages

27) stockcharts.com, “Relative strength index”, at: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_index_rsi

28) stockcharts.com, “Stochastic oscillators (Fast, Slow and Full)”, at: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator

29) AJ. Surkan, JC. Singleton, “Neural networks for bond rating improved by multiple hidden layers. Neural Networks”, 1990 IJCNN International Joint Conference, 1990

30) M. Zekic. “Neural Network Applications in Stock Market Predictions”, University of Josip Juraj Strossmayer in Osijek, Croatia, 1998

Glossary of terms:

- **%D**: It is the 3 day simple moving average of %K. Explained in section 6.2
- **%K**: It is the statistical parameter used to calculate the fast stochastic. Explained in section 6.2
- **EMA**: Exponential moving average. Explained in section 5.5
- **Fast stochastic**: A technical indicator used to forecast financial time series. Explained in section 5.5
- **Labs** (MATLAB®): refers to the processing modules started by MATLAB®. The parallel and distributed toolbox can start multiple MATLAB® processing modules (labs) that can process non-dependent processes simultaneously
- **MAE**: mean absolute error
- **MSE**: Mean square error
- **NARX**: nonlinear autoregressive network with exogenous inputs. Explained in section 4.1
- **Policy**: The rule that we set up for selection of a network for the future. Policies have been discussed in section 4.5 and 6.1
- **RSI**: Relative strength index. Explained in section 5.5
- **Sigmoid transfer function**: A type of neural network transfer function or activation function. Explained in section 4.1
- **Statistical period** (`statP`): an integer input received by some functions in this research which helps the network to determine the periodicity of the input data. If the input data has values for 5 working days only use 5. If data is available for all 7 days use 7
- **Tansig transfer function**: A type of neural network transfer function or activation function. Explained in section 4.1
- **Taps**: delayed inputs to a neural network. Explained in section 4.1
- **Technical indicators**: Statistical parameters that are calculated to analyze a financial time series. They can be plotted on the time series charts for trend analysis. (`www.stockcharts.com`)

Appendix A:

Function: dataCutter

```
function
[trainTapInS,trainTapOutS,trainSetInS,trainSetOutS,valTapInS,valTapOutS
,valSetInS,valSetOutS,testTapInS,testTapOutS,testSetInS,testSetOutS]=da
taCutter(diffSet,testPoint,testDur,tap,trainVol,valVol)

% RATIONALE:
% - This function is used to cut data into the data sets required for
neural network training, validation and testing.
% - Different papers suggest different ratio of
training:validation:testing data.
% - This function gives the user the freedom to cut the datasets from
input data on the basis of these ratios. It cuts the test data forward
from the test point and the
% other 2 sets backwards.
% - The data are output as sequential data hence can be used in static
as well as dynamic networks


% Cuts the data into the sets required for the network

%INPUTS
%diffSet = The original set to cut
%testPoint = The starting point for the test set
%testDur = The number of elements in test set (duration of testing)
%tap = Number of taps in the delay line
%trainVol = Volume of training data (training elements :test elements)
%valVol = Volume of validation set (validation elements:test elements)

%OUTPUTS
% name consists of 4 segments
% (e.g. 'train''Tap''In''S')
% segment 1 :
% train - refers to training set
% val - refers to validation set
% test - refers to validation set
% segment 2:
% Tap - implies it is data for tapped delay line
% Set - implies it is data for actual set
% segment 3:
% In - implies it is network input
% Out - implies it is network output
% segment 4:
% empty - Implies it is matrix data
% S - implies it is sequential data

testPoint;
valPoint=testPoint-(valVol*testDur);
trainPoint=valPoint-(trainVol*testDur);
```

```
trainTapIn=diffSet(trainPoint-tap:trainPoint-1,1:end-1);
trainTapOut=diffSet(trainPoint-tap:trainPoint-1,end:end);

trainSetIn=diffSet(trainPoint:valPoint-1,1:end-1);
trainSetOut=diffSet(trainPoint:valPoint-1,end:end);

valTapIn=diffSet(valPoint-tap:valPoint-1,1:end-1);
valTapOut=diffSet(valPoint-tap:valPoint-1,end:end);

valSetIn=diffSet(valPoint:testPoint-1,1:end-1);
valSetOut=diffSet(valPoint:testPoint-1,end:end);

testTapIn=diffSet(testPoint-tap:testPoint-1,1:end-1);
testTapOut=diffSet(testPoint-tap:testPoint-1,end:end);

testSetIn=diffSet(testPoint:testPoint+testDur-1,1:end-1);
testSetOut=diffSet(testPoint:testPoint+testDur-1,end:end);

% Transpose data and convert to sequential
trainTapInS = con2seq(trainTapIn');
trainTapOutS = con2seq(trainTapOut');

valTapInS = con2seq(valTapIn');
valTapOutS = con2seq(valTapOut');

testTapInS = con2seq(testTapIn');
testTapOutS = con2seq(testTapOut');


trainSetInS = con2seq(trainSetIn');
trainSetOutS = con2seq(trainSetOut');

valSetInS = con2seq(valSetIn');
valSetOutS = con2seq(valSetOut');

testSetInS = con2seq(testSetIn');
testSetOutS = con2seq(testSetOut');

end
```

Function: mainSetCleaner

```
function mainSet = mainSetCleaner(mainSet)
% RATIONALE: Data cleaner to remove NAN's from the targets.

% INPUT: mainSet (raw)
% OUTPUT: mainSet (clean)

outSet=mainSet(:,end);
mainSet=mainSet';
mainSet(:,isnan(outSet'))=[];
```

```
mainSet=mainSet';

end
```

Function:diffGenerator

```
function diffSet=diffGenerator(mainSet)
% RATIONALE: Generates the returns series for the mainSet
% INPUT: mainSet
% OUTPUT: Corresponding diffSet

diffSet=diff(mainSet)./mainSet(2:end,:);
end
```

Functions massage1/massageP

```
function diffSet=massage1(diffSet,valSetOutAct)

% RATIONALE: This function is used to encode the Targets (Pattern
recognition)

% INPUTS:
% diffSet: The returns series;
% valSetOutAct: The actual targets of evaluation set.

% OUTPUTS:
% diffSet: Returnes the massaged diffSet



% Setting thresholds
Set=diffSet(:,end);
ucutavg=mean(valSetOutAct(valSetOutAct>0))*1.5;
dcutavg=mean(valSetOutAct(valSetOutAct<0))*1.5;

% information encoding - generating the targets
Set=zeros(size(diffSet(:,end)));
a=find(diffSet(:,end)>ucutavg);
Set(a)=1;
b=find(diffSet(:,end)< (dcutavg));
Set(b)=(-1);
Set=(mapminmax(Set'))';
diffSet(:,end)=Set;

% Removing constants and fixing unknowns
diffSet=diffSet';
diffSet=fixunknowns(diffSet);
diffSet=removeconstantrows(diffSet);
diffSet=diffSet';

end
```

```
function diffSet=massageP(diffSet,valSetOutAct)
% RATIONALE: This function is used to encode the Targets (Function
approximation)

% INPUTS:
% diffSet: The returns series;
% valSetOutAct: The actual targets of evaluation set.

% OUTPUTS:
% diffSet: Returnes the massaged diffSet


% Removing outliers
Set=diffSet(:,end);
trainVal=[valSetOutAct'];
ucutavg=mean(trainVal(trainVal>0))*20;
dcutavg=mean(trainVal(trainVal<0))*20;

% Encoding information
a=find(diffSet(:,end)>ucutavg);
Set(a)=1;
b=find(diffSet(:,end)< (dcutavg));
Set(b)=(-1);
Set=(mapminmax(Set'))';
diffSet(:,end)=Set;

% Removing constants and fixing unknowns
diffSet=diffSet';
diffSet=fixunknowns(diffSet);
diffSet=removeconstantrows(diffSet);
diffSet=diffSet';

end
```

function: rsiGenerator

```
function rsiSet= rsiGenerator(mainSet,statP)

% RATIONALE: provides the RSI for the provided set

% INPUTS:
% mainSet - the input dataset. OHLC are the last 4 columns
% statP - statisical period e.g. 5 for 5 day week, 7 for 7 day week, 4
for 4 week month etc.

% OUTPUTS:
% rsiSet - the RSI values

Set=diff(mainSet(:,end));
rsiSet=zeros(length(Set),1);
ADrsiSet=Set.*(Set>0);
```

```
DErsiSet=abs(Set.*(Set<0));
GrsiSet=zeros(length(Set),1);
LrsiSet=zeros(length(Set),1);
GrsiSet(statP)= mean(ADrsiSet(1:statP));
LrsiSet(statP)= mean(DErsiSet(1:statP));
rsiSet(statP)=100-(100/(1+(GrsiSet(statP)/LrsiSet(statP))));

for c=(statP+1):length(Set)
    GrsiSet(c)= ((GrsiSet(c-1)*(statP-1))+ADrsiSet(c))/statP;
    LrsiSet(c)= ((LrsiSet(c-1)*(statP-1))+DErsiSet(c))/statP;
    RS=GrsiSet(c)/LrsiSet(c);
    rsiSet(c)=100-(100/(1+RS));
end

rsiSet=(mapminmax(rsiSet'))';
end
```

Function: fstGenerator

```
function [fstSet KfstSet NfstSet DfstSet]=fstGenerator(mainSet,statP)
% RATIONALE: provides the Fast Stochastic values for the provided set

% INPUTS:
% mainSet - the input dataset. OHLC are the last 4 columns
% statP - statisical period e.g. 5 for 5 day week, 7 for 7 day week, 4
for 4 week month etc.

% OUTPUTS:
% KfstSet - K values of Fst
% DfstSet - D values of Fst (k smoothed for 3 values)
% NfstSet - Highly smoothed K (over statP values)
% DfstSet - The fast stochastic


Set=diff(mainSet(:,end));
cl=mainSet(2:end,end);
lo=mainSet(2:end,end-1);
hi=mainSet(2:end,end-2);
KfstSet=zeros(length(Set),1);
DfstSet=zeros(length(Set),1);
NfstSet=zeros(length(Set),1);


for c=(statP):length(Set)
    KfstSet(c)=100*((cl(c)-min(lo(c-statP+1:c)))/(max(hi(c-statP+1:c))-
min(lo(c-statP+1:c))));
    DfstSet(c)=mean(KfstSet(c-2:c));
    NfstSet(c)=mean(KfstSet(c-statP+1:c));
end

fstSet=KfstSet-DfstSet;
fstSet=(mapminmax(fstSet'))';

end
```

Function: macdGenerator

```
function [ema1Set ema2Set macdSet] = macdGenerator(mainSet,statP)
% RATIONALE: provides the exponential moving average values for the
provided set

% INPUTS:
% mainSet - the input dataset. OHLC are the last 4 columns
% statP - statisical period e.g. 5 for 5 day week, 7 for 7 day week, 4
for 4 week month etc.

% OUTPUTS:
% ema1set - ema over 1 statistical period
% ema2set - ema over 2 statistical period
% macdSet - Difference between ema1 and ema2 . Not used by the neural
network

Set=diff(mainSet(:,end));
cl=mainSet(2:end,end);
ema1Set=zeros(length(Set),1);
ema2Set=zeros(length(Set),1);

ema1Set(statP)= mean(cl(1:statP));
ema2Set(statP*2)=mean(cl(1:statP*2));

for c=(statP+1):length(Set)
    ema1Set(c)=ema1Set(c-1)+((2/(statP+1))*(mainSet(c)-ema1Set(c-1)));
end


for c=(2*statP+1):length(Set)
    ema2Set(c)=ema2Set(c-1)+((2/(2*statP+1))*(mainSet(c)-ema2Set(c-
1)));
end

macdSet=ema2Set-ema1Set;
macdSet=(mapminmax(macdSet'))';

end
```

Function: nseval

```
function [ema1Set ema2Set macdSet] = macdGenerator(mainSet,statP)
% RATIONALE: provides the exponential moving average values for the
provided set

% INPUTS:
% mainSet - the input dataset. OHLC are the last 4 columns
```

```
% statP - statisical period e.g. 5 for 5 day week, 7 for 7 day week, 4
for 4 week month etc.

% OUTPUTS:
% ema1set - ema over 1 statistical period
% ema2set - ema over 2 statistical period
% macdSet - Difference between ema1 and ema2 . Not used by the neural
network

Set=diff(mainSet(:,end));
cl=mainSet(2:end,end);
ema1Set=zeros(length(Set),1);
ema2Set=zeros(length(Set),1);

ema1Set(statP)= mean(cl(1:statP));
ema2Set(statP*2)=mean(cl(1:statP*2));

for c=(statP+1):length(Set)
    ema1Set(c)=ema1Set(c-1)+((2/(statP+1))*(mainSet(c)-ema1Set(c-1)));
end


for c=(2*statP+1):length(Set)
    ema2Set(c)=ema2Set(c-1)+((2/(2*statP+1))*(mainSet(c)-ema2Set(c-
1)));
end

macdSet=ema2Set-ema1Set;
macdSet=(mapminmax(macdSet'))';

end
```

Function: batchEl3

```
%% || SHREE ||

function [rowmat valSetOutAct testSetOutAct]=
batchEl3(mainSet,testPoint,L1,tap,statP,trainDataVol,valDataVol,tmargin
,testDur,netTrainFucn,maxEpoch,lock,stratg)

% RATIONALE:
% - This function helps in building a customized NARX network for time
series prediction
% - The function internally calls in function to perform the following
%       clean the data provided by the user
%       generate the diffSet
%       generate indicators like RSI, Fast Stochastic, MACD
%       Cut the dataset as per the user specifications

% INPUTS:
% mainSet       : The entire time series provided by the user
% test point    : Specifies the row on the mainSet from where
predictions are to be made
```

```
% L1            : Number of neurons in the first layer of the neural
network
% tap           : Number of taps that the Narx network should have in
the input as well as the output delay line
% statP         : the statistical period for calculating the tech
indicators (e.g. 5 for a week)
% trainDataVol  : Ratio of training to testing data
% valDataVol    : Ratio of validation to testing data
% tmargin       : Trading margin
% testDur       : No of instances to predict
% netTrainFucn  : Training function for the neural net
% maxEpoch      : Max epoch for net training
% lock          : Specifies if the weights are to be initialized on
random seed or a fixed seed
% stratg        : Specifies the trading strategy 0: quick strategy 1:
never out of market

% OUTPUT:
% rowmat        : A row report of the behavior of the desired network
[Vnseval(9) Tnseval(9) NaN(2) Vnarxy(valDataVol*testDur) NaN(2)
narxy(testDur)]
% valSetOutAct  : The vector of the actual market values during the
validation period
% testSetOutAct : The vector of the actual market values during the
test period

% originally rowmat= batchEl1(1200,15,5,5,3,1,0.7,30,'trainrp',2000,0)

%% ||Shree||

neuronsByLayer=L1;                % Layer matrix

%% Selecting parameters
%mainSet=mainSet(:,[4 6 10 12 15 17 22 26 28 33 20]);   % Selecting
parameters of mainSet (optional - all selected if commented)

%% Cleaning The main Set
mainSet = mainSetCleaner(mainSet);

%% generate diff set
diffSet=diffGenerator(mainSet);

%% Calculating RSI
rsiSet= rsiGenerator(mainSet,statP);

%% Calculating Fast Stochastic
[fstSet KfstSet NfstSet DfstSet]=fstGenerator(mainSet,statP);

%% calculating MACD
[ema1Set ema2Set macdSet] = macdGenerator(mainSet,statP);

%% Add tech indicators to data set
outSet=diffSet(:,end);
diffSet=[mainSet(2:end,[1 2 end]) rsiSet KfstSet DfstSet fstSet ema1Set
ema2Set outSet];
```

```
diffSet=diffSet(2*statP+1:end,:);                          % Cut
undetermined values

%% preserve actual values for later calculations
[trainTapInS,trainTapOutS,trainSetInS,trainSetOutS,valTapInS,valTapOutS
,valSetInS,valSetOutS,testTapInS,testTapOutS,testSetInS,testSetOutS]=da
taCutter(diffSet,testPoint,testDur,tap,trainDataVol,valDataVol);
trainSetOutAct=cell2mat(trainSetOutS)';
valSetOutAct=cell2mat(valSetOutS)';
testSetOutAct=cell2mat(testSetOutS)';

%% Data Massaging
diffSet=massage1(diffSet,valSetOutAct);

%% Plot of original signal and signal presented to NW
%Set=diffSet(:,end);
%plot(1:length(Set),diffSet(:,end),1:length(Set),Set(:));

%% Create Train/Val/Test Sets and Delays
[trainTapInS,trainTapOutS,trainSetInS,trainSetOutS,valTapInS,valTapOutS
,valSetInS,valSetOutS,testTapInS,testTapOutS,testSetInS,testSetOutS]=da
taCutter(diffSet,testPoint,testDur,tap,trainDataVol,valDataVol);

%% Lock the random number generation (optional)
if(lock)
    randn('state',0);
    rand('state',0);
end

%% Create NARX Network
narxspNet =
newnarxsp2(trainSetInS,trainSetOutS,[1:tap],[1:tap],neuronsByLayer,{'ta
nsig','tansig'});
narxspNet = init(narxspNet);

%%  Set training parameters
narxspNet.divideFcn ='';
narxspNet.performFcn='mse';
narxspNet.trainParam.goal=0;
narxspNet.trainFcn = netTrainFucn;
narxspNet.trainParam.show = Inf;
narxspNet.trainParam.epochs = maxEpoch;

%% Create Validation and Test structures
VV.P = [valSetInS;valSetOutS];
VV.T = valSetOutS;
VV.Pi = [valTapInS;valTapOutS];

TV.P = [testSetInS;testSetOutS];
TV.T = testSetOutS;
TV.Pi = [testTapInS;testTapOutS];


%% Train the network
narxspNet.inputs{1}.processFcns = {'mapstd','mapminmax','processpca'};
```

```
narxspNet.inputs{1}.processFcns;
narxspNet.outputs{(narxspNet.numLayers)}.processFcns =
{'mapstd','mapminmax'};
narxspNet.outputs{(narxspNet.numLayers)}.processFcns;
[narxspNet,narxTr,Y,E,Pf,Af] =
train(narxspNet,[trainSetInS;trainSetOutS],trainSetOutS,[trainTapInS;tr
ainTapOutS],[],[],TV);
narxspNet.trainParam;               % outputs training parameters if
required

%% Simulate the network

[narxy,nPf,nAf,nE,nPerf] =
sim(narxspNet,[testSetInS;testSetOutS],[testTapInS;testTapOutS],[],test
SetOutS);
[vnarxy,vnPf,vnAf,vnE,vnPerf] =
sim(narxspNet,[valSetInS;valSetOutS],[valTapInS;valTapOutS],[],valSetOu
tS);

%% Analyse results

ny = cell2mat(narxy');
vny = cell2mat(vnarxy');

testSetOut=cell2mat(testSetOutS);
valSetOut=cell2mat(valSetOutS);

[vnpu,vhu,vnpd,vhd,vnpt,vht,vnetVal,vrmset,vmaet]=nseval(vny,(valSetOut
)',valSetOutAct,tmargin,stratg);
[npu,hu,npd,hd,npt,ht,netVal,rmset,maet]=nseval(ny,(testSetOut)',testSe
tOutAct,tmargin,stratg);

rowmat
=[vnpu,vhu,vnpd,vhd,vnpt,vht,vnetVal,vrmset,vmaet,npu,hu,npd,hd,npt,ht,
netVal,rmset,maet,NaN NaN vny' NaN NaN ny' ];

end
```

Function: B2res

```
function avgro=B1res(lim,stratg,gen)


load repBatch1_06
outmat06m=Mcalc2(repBatch1_06_M,lim,80,1,stratg,gen);
outmat06s=Mcalc2(repBatch1_06_S,lim,80,1,stratg,gen);

load repBatch1_07
outmat07j=Mcalc2(repBatch1_07_J,lim,80,1,stratg,gen);
outmat07m=Mcalc2(repBatch1_07_M,lim,80,1,stratg,gen);
outmat07s=Mcalc2(repBatch1_07_S,lim,80,1,stratg,gen);
outmat08j=Mcalc2(repBatch1_08_J,lim,80,1,stratg,gen);
```

```
avgro=(outmat06m(1,:)+outmat06s(1,:)+outmat07j(1,:)+outmat07m(1,:)+outm
at07s(1,:)+outmat08j(1,:))./6;

end
```

Function: batchCombined

```
%% || SHREE ||
%%
tic
load cad1.mat

% uses EL3 - 10 elems
testDur=80;                     % No of instances to predict
NurMax=80;                      % Max no of neurons to be tried in each
matrix
TapMax=4;                       % Max no of taps to be tried in the
experiment
netTrainFucn='trainrp';         % Training function for the neural net
maxEpoch=800;                   % Max epoch for net training



tmargin=0.0;                    % Trading margin
trainDataVol=3;                 % ratio of training to testing data
valDataVol=1;                   % ratio of validation to testing data
statP=5;                        % statistical periods

lock=1;                         % 1=the random function is locked  0=not
locked
stratg=0;                       % 0 = quick trade        1 = buy n hold


pointMat=[4415 4500 4589];          % 2006 points
TN=combvec(1:TapMax,1:NurMax)';     % vector of no of taps and neurons

matRows=(TapMax*NurMax);            % rows of report matrix
matCols=(18+2+valDataVol*testDur+2+testDur);         % cols of report
matrix [Vnseval Tnseval NaN NaN Vnarxy NaN NaN Tnarxy]
repMat= zeros(3*matRows,matCols);
actMat= zeros(3,matCols);
rno=matRows+1;
for i=2:3
    testPoint=pointMat(i);
    for j=1:(TapMax*NurMax)
        tap = TN(j,1);
        L1=TN(j,2);
        [rowmat valSetOutAct testSetOutAct]=
batchEl3(mainSet,testPoint,L1,tap,statP,trainDataVol,valDataVol,tmargin
,testDur,netTrainFucn,maxEpoch,lock,stratg);
        repMat(rno,:)=rowmat;
        [rno testPoint tap L1]
        rno=rno+1;
```

```
    end
    actMat(i,:)= [zeros(1,18) NaN NaN valSetOutAct' NaN NaN
testSetOutAct'];
end
repBatch1_06_J =[repMat(1:matRows,:)' actMat(1,:)' ]';
repBatch1_06_M=[repMat(matRows+1:2*matRows,:)' actMat(2,:)' ]';
repBatch1_06_S=[repMat(2*matRows+1:3*matRows,:)' actMat(3,:)']';

save repBatch1_06.mat repBatch1_06_J repBatch1_06_M repBatch1_06_S
toc



%% || SHREE ||
%%
tic
load cad1.mat

% uses EL3 - 10 elems
testDur=80;                     % No of instances to predict
NurMax=80;                      % Max no of nurons to be tried in each
matrix
TapMax=4;                       % Max no of taps to be tried in the
experiment
netTrainFucn='trainrp';         % Training function for the neural net
maxEpoch=800;                   % Max epoch for net training


tmargin=0.0;                    % Trading margin
trainDataVol=3;                 % ratio of training to testing data
valDataVol=1;                   % ratio of validation to testing data
statP=5;                        % statistical periods

lock=1;                         % 1=the random function is locked  0=not
locked
stratg=0;                       % 0 = quick trade        1 = buy n hold


pointMat=[4675 4761 4850 4936];      % 2007 points
TN=combvec(1:TapMax,1:NurMax)';      % vector of no of taps and neurons

matRows=(TapMax*NurMax);             % rows of report matrix
matCols=(18+2+valDataVol*testDur+2+testDur);         % cols of report
matrix [Vnseval Tnseval NaN NaN Vnarxy NaN NaN Tnarxy]
repMat= zeros(4*matRows,matCols);
actMat= zeros(4,matCols);
rno=1;
for i=1:4
    testPoint=pointMat(i);
    for j=1:(TapMax*NurMax)
        tap = TN(j,1);
        L1=TN(j,2);
        [rowmat valSetOutAct testSetOutAct]=
batchEl3(mainSet,testPoint,L1,tap,statP,trainDataVol,valDataVol,tmargin
,testDur,netTrainFucn,maxEpoch,lock,stratg);
        repMat(rno,:)=rowmat;
        [rno testPoint tap L1]
```

```
        rno=rno+1;
    end
    actMat(i,:)= [zeros(1,18) NaN NaN valSetOutAct' NaN NaN
testSetOutAct'];
end
repBatch1_07_J =[repMat(1:matRows,:)' actMat(1,:)' ]';
repBatch1_07_M=[repMat(matRows+1:2*matRows,:)' actMat(2,:)' ]';
repBatch1_07_S=[repMat(2*matRows+1:3*matRows,:)' actMat(3,:)']';
repBatch1_08_J =[repMat(3*matRows+1:4*matRows,:)' actMat(4,:)']';

save repBatch1_07.mat repBatch1_07_J repBatch1_07_M repBatch1_07_S
repBatch1_08_J
toc

%%
B2res(tmargin,0,0) % Execute the policy number 4 explained in section
6.1
```

Function: batchAll

```
Batch2_06.m % generate report matrix for periods in 06
Batch2_07.m % generate report matrix for periods in 07
B2res(tmargin,0,0) % execute the policy number 4 explained in section
6.1
```

Appendix B:

Performance of adaptive NARX on using twice the evaluation data:

Period	Val
M06 - S06	1.0567
S06 - J07	0.8316
J07 - M07	1.1172
M07 -S07	0.8812
S07- J08	0.8347
J08 - M08	1.0918
Average	0.968867
Realized Value	0.788397

List of Exogenous time series by asset

Asset	Exogenous Input	Exogenous Input
Gold	Japan Currency	UK currency
Japan Currency	Gold Price	UK currency
Canada Currency	Japan Currency	Australia currency
UK currency	Japan Currency	Australia currency
France Currency	Japan Currency	Australia currency
India Currency	Japan Currency	Australia currency

Comparison of decision policies 1, 2 and 3 – period wise results:
Gold:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	1	0.8017	0.91
S06 - J07	0.9983	1.1519	1.0448
J07 - M07	1	1.0419	1.0222
M07 -S07	1.0017	0.9224	0.8256
S07- J08	1	1.2069	1.1511
J08 - M08	0.9549	1.2672	1.2278
Average	0.992483	1.065333	1.03025
Realized Value	0.954897	1.357339	1.13402

UK:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	0.9476	1.0409	1.01
S06 - J07	1	1.0413	1.0722
J07 - M07	0.9783	1.0133	1.0604
M07 -S07	0.9934	0.993	0.9956
S07- J08	1	1.0115	1.0115
J08 - M08	1.0021	1.0649	0.9899
Average	0.9869	1.027483	1.023267
Realized Value	0.922853	1.174754	1.144746

Japan:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	1	1.0053	1.0575
S06 - J07	0.9919	0.9721	0.9936
J07 - M07	1.0079	1.0417	0.9924
M07 -S07	1	1.061	1.0899
S07- J08	1.0028	0.9864	0.9978
J08 - M08	1.0479	1.2195	1.1549
Average	1.008417	1.047667	1.047683
Realized Value	1.050557	1.29927	1.309644

Canada:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	1.0478	1.0591	1.0013
S06 - J07	1.0039	1.0064	0.9908
J07 - M07	1	1.0024	1.0158
M07 -S07	0.9506	1.0619	0.9378
S07- J08	0.9983	0.8923	0.8923
J08 - M08	1	1.0021	1.0092
Average	1.0001	1.004033	0.974533
Realized Value	0.998223	1.014505	0.851053

France:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	0.9973	0.9918	0.9574
S06 - J07	0.9907	0.9652	0.9758
J07 - M07	1	1.0179	1.0314
M07 -S07	1	1.0138	1.0138
S07- J08	1	1.0444	0.9624
J08 - M08	1	1.0556	1.0808
Average	0.998	1.014783	1.0036
Realized Value	0.988025	1.089093	1.016096

India:

Period	Policy 1	Policy 2	Policy 3
M06 - S06	1.0237	1.0115	0.9612
S06 - J07	1.0042	1.0266	1.0306
J07 - M07	1	1.0197	1.0416
M07 -S07	1	1.003	1.003
S07- J08	1.0243	1.0192	1.0009
J08 - M08	0.9882	1.0234	1.0125
Average	1.006733	1.017233	1.0083
Realized Value	1.040555	1.107759	1.048797

Comparison of decision policies 4, 5 and 6 – period wise results:

Gold:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	1.0413	1.1519	0.8941
S06 - J07	1.1519	1.0296	1.0296
J07 - M07	1.0419	0.9224	0.9224
M07 -S07	0.9224	1.1511	1.2069
S07- J08	1.2069	1.155	1.155
J08 - M08	1.155	1.0631	1.0226
Average	1.086567	1.07885	1.038433
Realized Value	1.606901	1.546222	1.210411

UK:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	1.0409	0.9874	0.9874
S06 - J07	1.0413	1.0413	1.0722
J07 - M07	1.0133	1.0365	1.0365
M07 -S07	1.0183	1.0025	1.03
S07- J08	1.0062	1.0002	1.0002
J08 - M08	1.1218	0.9883	0.9883
Average	1.0403	1.009367	1.0191
Realized Value	1.262404	1.056084	1.117252

Japan:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	0.9684	1.0877	1.0546
S06 - J07	0.9758	0.9758	0.9758
J07 - M07	1.0598	1.0598	1.0598
M07 -S07	1.061	1.061	1.061
S07- J08	1.1246	0.9282	1.0023
J08 - M08	1.2195	1.2195	1.2195
Average	1.068183	1.055333	1.062167
Realized Value	1.457252	1.350929	1.414384

Canada:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	1.0591	1.0591	1.0591
S06 - J07	1.0002	1.0002	0.9908
J07 - M07	1.0024	1.0024	1.0024
M07 -S07	1.0619	1.0619	1.0135
S07- J08	1.0078	0.8923	0.9106
J08 - M08	1.0113	1.0086	1.0086
Average	1.023783	1.004083	0.9975
Realized Value	1.149219	1.014795	0.979117

France:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	1.0779	0.9982	1.0349
S06 - J07	1.0141	1.0133	1.0133
J07 - M07	1.0179	1.0179	1.0685
M07 -S07	1.0191	0.9949	0.9949
S07- J08	1.0444	0.9572	1.0444
J08 - M08	1.0556	1.0556	0.9086
Average	1.038167	1.006183	1.010767
Realized Value	1.250108	1.035004	1.057864

India:

Period	Policy 4	Policy 5	Policy 6
M06 - S06	1.0115	1.0115	1.0115
S06 - J07	1.0266	1.0161	1.035
J07 - M07	1.0197	1.0197	0.9533
M07 -S07	1.0308	1.0308	1.0308
S07- J08	1.0192	1.0215	1.0009
J08 - M08	1.0234	1.0125	1.0125
Average	1.021867	1.018683	1.007333
Realized Value	1.138463	1.117333	1.042548

Individual period wise behavior of adaptive NARX

Gold:

Period	up	Dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	0.00	70.00	70.00	0.47	1.04	42.00	2.00	11.00
S06 - J07	39.00	16.00	55.00	0.60	1.15	272.00	4.00	68.00
J07 - M07	34.00	28.00	62.00	0.55	1.04	173.00	1.00	44.00
M07 -S07	14.00	44.00	58.00	0.47	0.92	159.00	3.00	40.00
S07- J08	68.00	0.00	68.00	0.56	1.21	138.00	2.00	35.00
J08 - M08	51.00	12.00	63.00	0.52	1.16	186.00	2.00	47.00
Average	34.33	28.33	62.67	0.53	1.09			
Realized Value	1.61							
80 day Equ	1.88							

UK:

Period	up	Dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	42.00	33.00	75.00	0.51	1.04	15.00	3.00	4.00
S06 - J07	71.00	3.00	74.00	0.51	1.04	197.00	1.00	50.00
J07 - M07	51.00	26.00	77.00	0.48	1.01	262.00	2.00	66.00
M07 -S07	63.00	14.00	77.00	0.55	1.02	271.00	3.00	68.00
S07- J08	71.00	6.00	77.00	0.56	1.01	96.00	4.00	24.00
J08 - M08	44.00	27.00	71.00	0.69	1.12	150.00	2.00	38.00
Average	57.00	18.17	75.17	0.55	1.04			
Realized Value	1.26							
80 day Equ	1.28							

Japan:

Period	up	Dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	57.00	16.00	73.00	0.51	0.97	65.00	1.00	17.00
S06 - J07	48.00	24.00	72.00	0.44	0.98	165.00	1.00	42.00
J07 - M07	69.00	5.00	74.00	0.50	1.06	294.00	2.00	74.00
M07 -S07	71.00	6.00	77.00	0.52	1.06	229.00	1.00	58.00
S07- J08	11.00	51.00	62.00	0.61	1.12	296.00	4.00	74.00
J08 - M08	63.00	5.00	68.00	0.60	1.22	134.00	2.00	34.00
Average	53.17	17.83	71.00	0.53	1.07			
Realized Value	1.46							
80 day Equ	1.54							

Canada:

Period	up	dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	29.00	40.00	69.00	0.48	1.06	88.00	4.00	22.00
S06 - J07	51.00	19.00	70.00	0.47	1.00	86.00	2.00	22.00
J07 - M07	20.00	53.00	73.00	0.55	1.00	191.00	3.00	48.00
M07 -S07	25.00	53.00	78.00	0.59	1.06	75.00	3.00	19.00
S07- J08	61.00	17.00	78.00	0.49	1.01	129.00	1.00	33.00
J08 - M08	5.00	48.00	53.00	0.49	1.01	29.00	1.00	8.00
Average	31.83	38.33	70.17	0.51	1.02			
Realized Value	1.15							
80 day Equ	1.17							

France:

Period	up	dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	34.00	35.00	69.00	0.64	1.08	242.00	2.00	61.00
S06 - J07	19.00	49.00	68.00	0.53	1.01	104.00	4.00	26.00
J07 - M07	55.00	17.00	72.00	0.53	1.02	226.00	2.00	57.00
M07 -S07	18.00	55.00	73.00	0.52	1.02	57.00	1.00	15.00
S07- J08	3.00	71.00	74.00	0.64	1.04	103.00	3.00	26.00
J08 - M08	12.00	50.00	62.00	0.53	1.06	258.00	2.00	65.00
Average	23.50	46.17	69.67	0.56	1.04			
Realized Value	1.25							
80 day Equ	1.30							

India:

Period	up	dn	tot	hit	val	Net no	taps	Neurons
M06 - S06	9.00	66.00	75.00	0.39	1.01	171.00	3.00	43.00
S06 - J07	8.00	61.00	69.00	0.58	1.03	180.00	4.00	45.00
J07 - M07	16.00	56.00	72.00	0.44	1.02	303.00	3.00	76.00
M07 -S07	18.00	44.00	62.00	0.56	1.03	23.00	3.00	6.00
S07- J08	25.00	45.00	70.00	0.49	1.02	99.00	3.00	25.00
J08 - M08	40.00	36.00	76.00	0.50	1.02	55.00	3.00	14.00
Average	19.33	51.33	70.67	0.49	1.02			
Realized Value	1.14							
80 day Equ	1.16							

The last two columns indicate the architecture of the neural network selected by the system for making predictions on the test set. The tables show that the system changed the number of neurons and taps very frequently. Section 6.3 describes the possible inference of these architectures & associated future work

Performance of fixed taps adaptive feed forward networks

Gold:

Period	tap =1	taps = 2	taps = 3	taps =4
M06 - S06	0.9712	1.1555	1.07	1.07
S06 - J07	1.005	0.9666	0.99	0.84
J07 - M07	1.03	1.0243	0.96	0.8566
M07 -S07	0.9	1.0554	0.85	1.0583
S07- J08	0.94	1.1649	1	0.9405
J08 - M08	1.24	0.91	1.021	0.8413

Average	1.014367	1.046117	0.981833	0.93445
Realized Value	1.054639	1.279944	0.882541	0.644703

UK:

Period	tap =1	taps = 2	taps = 3	taps =4
M06 - S06	0.9507	0.9798	0.9744	0.9654
S06 - J07	0.9942	0.9532	1.0002	1.008
J07 - M07	1.0144	1.0378	0.988	1.056
M07 -S07	0.9734	1.0129	1.0065	0.9965
S07- J08	1.041	1.0267	1.0909	1.0074
J08 - M08	1.0102	0.9709	0.9785	1.0047
Average	0.997317	0.996883	1.006417	1.006333
Realized Value	0.981468	0.978633	1.034524	1.036448

Japan:

Period	tap =1	taps = 2	taps = 3	taps =4
M06 - S06	0.9645	0.9732	0.9732	1.005
S06 - J07	1.0013	0.9873	0.99	1.012
J07 - M07	1.0036	1.0394	1.003	1.021
M07 -S07	1.0481	0.9767	0.959	1.029
S07- J08	1.0023	1.0442	0.997	1.164
J08 - M08	1.0348	1.0128	1.0838	1.075
Average	1.0091	1.0056	1.001	1.051
Realized Value	1.05362	1.031579	1.001385	1.337055