Traffic Prediction Using Neural Networks

Edmund S. Yu and C.Y. Roger Chen

Edmund S. Yu, School of Computer and Information Sciences, Syracuse University, Syracuse, New York
C.Y. Roger Chen, High Performance Distributed Computing Lab, Department of Electrical and Computer Engineering, Syracuse University, Syracuse, New York

Abstract Broadband ISDN has made possible a variety of new multimedia services, but also created new problems for congestion control, due to the bursty nature of traffic sources. Traffic prediction has been shown to be able to alleviate this problem in [1, 2]. The traffic prediction model in their framework is a special case of the Box-Jenkins' ARIMA models. In this paper we would like to go one step further and propose a new approach, the neural network approach, for traffic prediction. A (1, 5, 1) back-propagation feedforward neural network is trained to capture the linear and non-linear regularities in several time serieses. A comparison between the results from the neural network approach and the Box-Jenkins approach is also provided. The non-linearity used in this paper is chaotic. We have designed a set of experiments to show that neural networks' prediction performance is only slightly affected by the intensity of the stochastic component (noise) in a time series. We have also demonstrated that a neural network's performance should be measured against the variance of the noise to gain more insight into its behavior and prediction performance. Based on the experimental results we then conclude that the neural network approach is an attractive alternative to traditional regression techniques as a tool for traffic prediction.

I. Introduction

The advent of broadband integrated services digital networks (B-ISDN) has made possible a variety of new multimedia services, but it has also created new problems for congestion control. Most of the problems arise because of the bursty nature of the traffic sources. It has been argued that traditional reactive congestion control is not suitable for broadband integrated networks due to the effects of high-speed channels[3, 4]. Recently, Lazar et al. have proposed a new congestion control scheme, called proactive control, to overcome this difficulty [1, 2]. The core of their proactive congestion control scheme lies in the traffic predictor. The traffic predictor in their framework is a seasonal auto-regressive (AR) model, which is a special case of the Box-Jenkins' auto-regressive integrated moving average (ARIMA) models.

Among traditional regression techniques the ARIMA models are said to be optimal forecasts [5]. They are optimal in the sense that no other univariate forecasts have a smaller mean squared error (MSE). However, this comparison is valid only for those univariate models that are linear combinations of the past values in the time series, with fixed coefficients. The seasonal AR model used in [1, 2] likewise can only capture the linear relationship among the cell arrivals of traffic sources. Non-linear regression techniques do exist, but they require much more computational and intellectual efforts. This fact severely limits their practicality. In this paper we would like to propose a new approach, the neural network approach, for traffic prediction. It is well known that neural networks are capable of performing non-linear mappings between real-valued inputs and outputs. As a matter of fact, mathematical theorems have proved that a three-layer feedforward neural network, with sigmoidal units in the hidden layer, can approximate a given real-valued, continuous multi-variate function to any desired degree of accuracy [6, 7, 8]. Furthermore, the consistency property of three-layer feedforward networks has also been established in [9], which , in turn, implies that this kind of neural networks possess non-parametric regression capabilities.

An ARIMA model is normally represented as a triple (p, d, q), where p represents the order of the AR terms in the model, q represents the order of moving average (MA) terms, and d is the number of differences of the original series necessary to remove any inherent trends and make the time series stationary. The AR component can be written as the following form:

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \dots + a_p y_{t-p} + \varepsilon_t \qquad (1)$$

where a_i are coefficients and ε_t is the white noise. On the other hand, the multi-layer feedforward neural networks, when applied to time series forecasting, can be represented as:

$$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-p}) + \varepsilon_t \qquad (2)$$

where f is a linear or non-linear function, which, in non-linear regression analysis, is normally called the expectation function. Figure 1 is a conceptual diagram of the neural network model, when applied to time series forecasting. The inputs are the past values of the time series $y_{t-1}, y_{t-2} \dots y_{t-p}$, where p is the order in AR models and is the number of input units (nodes) in neural networks. The single output, y_t is the future value to be predicted,

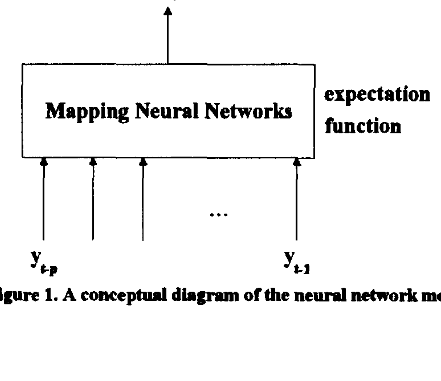

Figure 1. A conceptual diagram of the neural network model.

II. The neural network model

The neural network model we have adopted in this paper is a three-layer feedforward neural network (FFNN) with the back-propagation learning algorithm [10]. Three-layer FFNNs are a subset of the multi-layer back-propagation neural networks, originally called multi-layer perceptrons (MLP). In this paper, we will consistently call them FFNNs. Their mathematical properties have already been mentioned in Section I. A three-layer FFNN consists of an input layer, an output layer and a hidden layer. Each of these layers consists of one or more neurons (processing units). Figure 2 is an example of a three-layer FFNN.

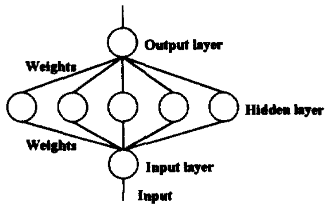

Figure 2. A (1, 5, 1) three-layer FFNN.

Its hidden layer has five processing units, while there is only one input unit and one output unit. Actually, this is the neural network we used throughout the experiments reported in this paper. Each processing unit in a layer is fully connected with all the processing units in the neighboring layers, but not with those in the same layer. Therefore, the number of units in each layer completely determine the topology of an FFNN and the FFNN in Figure 2 can simply be represented by a triple (1, 5, 1), where the first number refers to the number of processing units in the input layer, the second number the hidden layer, and so on. Those connections are uni-directional. Each unit receives inputs from the units one layer below it and sends outputs to the units one layer above it. Each connection is associated with a weight, which, conceptually, represents the strength of the connection between those two units. For all units except the input units, the net input is a weighted sum of all the inputs plus a bias,

$$\text{net}_i = \sum_j (\text{weight}_{ji} \cdot \text{input}_j) + \text{bias}_i$$

Mathematically, the bias term in this equation can be treated just like ordinary input terms, with its input value fixed to 1 and the bias is just the weight associated with this input. Originally, it was called the threshold, but the term bias is more generally applicable and has been widely adopted in literature. The output of a unit is determined by applying a transfer function f to the net input, $\text{output}_i = f(\text{net}_i)$. The transfer function can be any differentiable non-linear function. The commonly used transfer functions are the sigmoidal functions, such as the logistic function, $f(x) = 1/(1 + e^{-x})$, which is the function we use in our experiments. The input units function just like fan-out units. They simply broadcast the external inputs to the hidden layer. There is no computation involved. Hence, given a set of weights and biases, the entire network can be thought of as a mapping, from a set of input vectors to a set of output vectors. So, if we again use equation (2) to model a time series, the prediction problem is just to find a set of weights and biases that can minimize ε_t. This set of weights and biases in turn defines the expectation function. The back-propagation learning algorithm is a gradient descent method for finding such a set of weights and biases [10].

III. Video traffic

We have selected a video traffic source as the starter experiment. Video sources are expected to be the major traffic components in B-ISDN and their traffic characteristics are generally considered to be much more complex (bursty) than other types of sources like data sources or voice sources. In real world situations the characteristics of the input traffic is assumed to be unknown. A neural network is then trained to learn the regularities in the traffic patterns. The traffic must be able to be represented as a discrete-time time series with a fixed lag time; otherwise, both the Box-Jenkins and neural network approaches are not applicable.

Our traffic simulator generates a time series for video sources following the AR Markov model described in [11],

$$y_t = ay_{t-1} + bw_t \tag{3}$$

where a = 0.8781, b = 0.1108 and w_t is a Gaussian white noise with mean = 0.572 and variance = 1. The lag time is 1/30 second, which is the frame generation rate. The unit of y_t here is 1 bit/pixel, which corresponds to 7.5 Mbits/s.

The time series data thus generated is not yet ready to be used by our neural networks. Since currently we are using the logistic function for all units in the hidden layer and the output layer, we have to normalize the traffic data so that all the y values fall between 0 and 1. Next, we have to set up the training patterns and the test patterns. The actual form of these patterns depends on the numbers of the input and output units. In this experiment the number of input and output units is fixed to 1, respectively. Therefore, each training pattern takes the form (y_{t-1}, y_t). As to how many training patterns are necessary, ideally, we should use as many patterns as possible and stop when increasing the size of the training patterns does not significantly improve the results (mean squared errors). But practical considerations may prevent us from doing so, either because the training time becomes too long or we do not have so many training patterns. Throughout our experiments we have fixed the size of training patterns to 200. We have also set up a test set for each experiment, which also consists of 200 patterns, to cross-validate the results. Besides using the neural network approach, we have also applied the Box-Jenkins approach to this time series. The resulting model is an AR Markov (AR(1)),

$$y_t - \mu = 0.8393\,(y_{t-1} - \mu) + a_t, \tag{4}$$

where $\mu = 0.6372$ is the (estimated) mean of the time series and a_t is a Gaussian white noise with mean = 0 and variance = 0.01203. Equation (4) then can be used for prediction. Figures

3 and 4 show the one-step prediction results from both approaches. Figure 3 is for the training set, while Figure 4 is for the test set. As you can see, the prediction results from both approaches are very close. This is not surprising because the time series was generated by a linear model. Table I shows the numerical results.

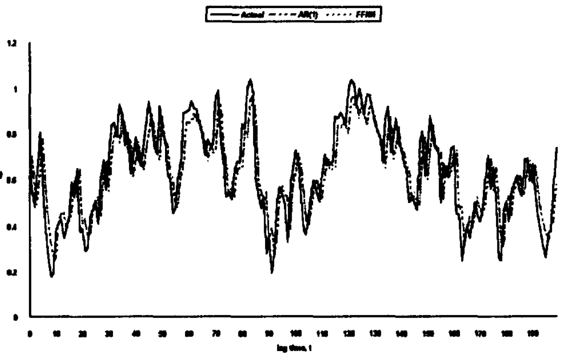
Figure 3. One-step prediction results for the training set generated by (3).

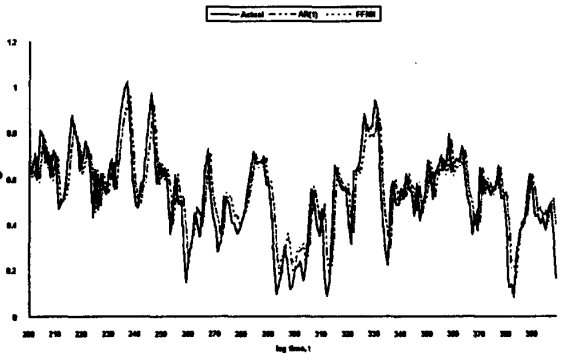
Figure 4. One-step prediction results for the test set generated by (3).

Table I. Comparison between the NN and Box-Jenkins approach.

	AR(1) Training Set	FFNN Training Set	AR(1) Test Set	FFNN Test Set
MSE	0.01208	0.01223	0.01290	0.01268

IV. A chaotic time series

In this experiment we have decided to use a chaotic time series. There are two reasons for making this decision. First, recently researchers have pointed out that the concepts of chaotic systems can provide new tools for understanding and modeling natural phenomena and can be used in many scientific and engineering applications, including communications. Secondly, it is very difficult for the traditional regression techniques to handle chaotic time series [12]. A chaotic time series generated by a simple deterministic equation, called the logistic map,

$$y_t = 4y_{t-1}(1 - y_{t-1}) \qquad (5)$$

can pass virtually every test of randomness [13]. The actual traffic depicted in Figure 7 is the time series generated by the logistic map with a seed $y_0 = 0.2$. Please note that the y values are all in the range [0, 1]. This is one of the examples considered in [12], but in that paper the authors use 1000 training patterns to train the neural network and the *tanh* function as the transfer function for the processing units. In our experiments we have found out that 200 training patterns and the logistic transfer function are enough to obtain comparable results. Figures 5 and 6 show the auto-correlation function (ACF) and partial auto-correlation function (PACF) for the first 200 points in the time series, respectively.

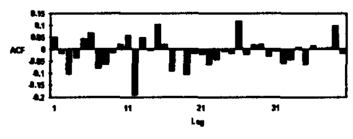
Figure 5. ACF for the first 200 points generated by (5)

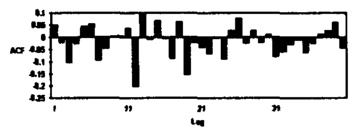
Figure 6. PACF for the first 200 points generated by (5)

The autocorrelations and partial autocorrelations at all lags except at lag 12 are smaller than 1.96 times the standard error, which can be approximated by $n^{-1/2}$, where n is the number of observations, 200 in this case. If we apply the Box-Jenkins identification techniques, this time series will easily be treated as a uncorrelated random sequence or as a seasonal AR process with period 12, which is a very inappropriate model. On the other hand neural networks have no problem identifying the non-linear regularities in the time series. One-step prediction results for the training and test sets are shown in Figures 7 and 8, respectively, and the stabilized weights and biases are shown in Figure 9. The number attached to a connection is the weight, while the number within a circle (processing unit) represents its bias. These number are all rounded down to one digit after the decimal point to make it more readable. We trained the neural network 60,000 cycles (epochs) to get this set of weights and biases.

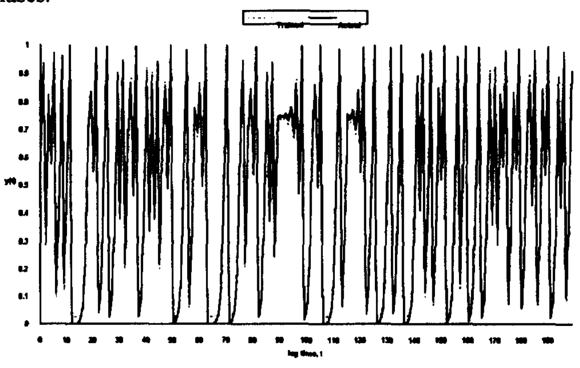
Figure 7. One-step prediction results for the training set generated by (5).

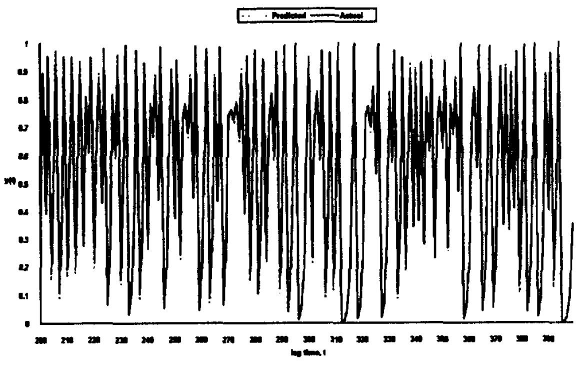

Figure 8. One-step prediction results for the test set generated by (5).

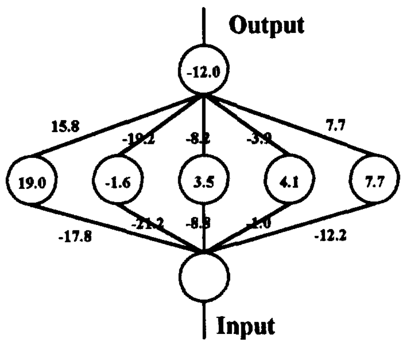

Figure 9. A neural network that can predict a chaotic time series.

V. Noises vs. MSE

This set of experiments has been designed to show how an FFNN behaves with respect to noises. This is important because we have to know if an FFNN can still detect the regularities in a noisy environment. If it has the tendency to absorb too much noise instead of filtering it out, then it is not useful for stochastic time series and the prediction results can not be trusted.

Three time series were generated by the following equations, respectively,

$$y_t = 0.9 * 4y_{t-1}(1 - y_{t-1}) + 0.1 * \text{Noise}_t \quad (6)$$
$$y_t = 0.7 * 4y_{t-1}(1 - y_{t-1}) + 0.3 * \text{Noise}_t \quad (7)$$
$$y_t = 0.5 * 4y_{t-1}(1 - y_{t-1}) + 0.5 * \text{Noise}_t \quad (8)$$

where Noise_t is a uniformly distributed random noise between 0 and 1. This set of experiments has been designed in such a way that we can easily see the relative strength between the deterministic part and the stochastic part. Again, 200 training patterns have been used to train the neural network. Once the neural network has been sufficiently trained, another 200 test patterns are used to cross-validate its prediction performance and generalization capability.

When the random noise becomes severe, the effects of over-training must be taken into consideration. Over-training occurs when the neural network tries to fit the training data too closely. One solution to this problem is to use the test set to cross-validate. Normally when the MSE for the training set decreases during training, so does the MSE for the test set. But once a neural network hits the over-training point, the situation reverses. Figures 10, 11 and 12 show the effects of over-training for the time series generated by (8).

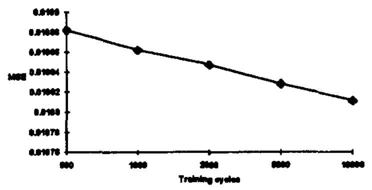

Figure 10. MSE vs. training cycles for the training set.

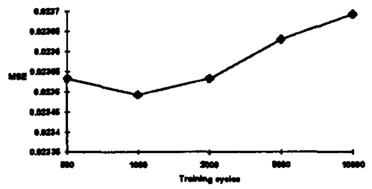

Figure 11. MSE vs. training cycles for the test set.

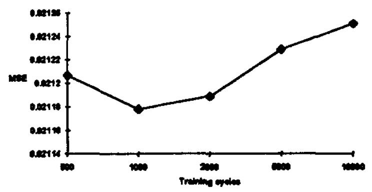

Figure 12. Average MSE vs. training cycles for the entire set.

As we can see from the curves, 1,000 training cycles is the time we should stop training the neural network. After that, further training not only increases the MSE for the test set, it also increases the average MSE of the two sets. For the time series generated by (7), the over-training point is around the 4,000th cycle, while for the time series generated by (6), it is later than the 60,000th cycles and does not have much impact on the results. The following table summarizes the numerical results of this set of experiments.

Table II. Numerical results of time series (6), (7) and (8).

	Training set (6)	Test set (6)	Training set (7)	Test set (7)	Training set (8)	Test set (8)
Mean	0.6385	0.6260	0.6644	0.6723	0.6139	0.6300
SD	0.2548	0.2680	0.1936	0.1751	0.1476	0.1321
MSE	0.000778	0.000916	0.006951	0.008648	0.01885	0.02349
RMSE	0.02789	0.03016	0.08337	0.09299	0.1373	0.1533
RE	0.1095	0.1125	0.4306	0.5310	0.9302	1.1602
Variance of noise	0.000779	0.000881	0.007007	0.007926	0.01946	0.02202
MSE/ Variance of noise	0.9987	1.0397	0.9920	1.0911	0.9687	1.0668

To measure the prediction performance, we can use the relative error suggested in [14], which is the ratio between the root MSE (RMSE) and the standard deviation (SD) of the time series. If the ratio is 1, it means the prediction results are as bad as using the mean of the time series to predict (guess) each and every one of the future values. The closer it is to 0, the better the prediction results. Thus, we can say the neural network predicts pretty well for the time series (6), bad for time series (7) and as bad as guessing for time series (8). But we know this measurement is not fair when there is a stochastic component (noise) present in the time series. The intensity of the noise affects the prediction results. Therefore, we need a new measurement to measure its performance. If we compare the MSEs (row 3 of Table II) with the variances of the noise (row 6 of Table II), we will see they are pretty close. The ratios are listed in row 7 of Table II. The variance of the noise can be thought of as the MSE of the optimal prediction. Take equation (6) as an example, the optimal prediction is,

$$y_t = 0.9 * 4y_{t-1}(1 - y_{t-1}) + 0.1 * 0.5 \qquad (9)$$

Using the mean of noise, which is 0.5, to predict the noise is the best we can do. Then, the MSE of this prediction is just the variance of the sequence $0.1*\text{Noise}_t$. From this point of view, we can say that the prediction results of the neural network are close to the optimal. Even in the case of (8), when the noise level is relatively high, the ratio is still not far away from 1. We, therefore, conclude that FFNNs indeed have the capability of filtering out (most) of the noise and capturing the regularities buried within it.

VI. Conclusion

In this paper we have shown how to use neural networks to perform traffic prediction. The theoretical justification of this approach is that FFNNs are capable of approximating any continuous function and perform non-parametric regression. When compared with traditional non-linear regression analysis, a single FFNN can provide many function forms to be used to approximate the expectation function. This property makes the neural network approach an attractive alternative to traditional regression techniques. In [10], the author establishes the consistency property of FFNNs by providing a growth rate of the network complexity with respect to the sample size n. However, his theory does not provide any practical method for choosing network complexity, given sample size n. In practice, the network complexity still has to be decided empirically. We believe by using our experimental procedure, with a fixed network complexity and sample size, we can gain more insight into a neural network's behavior and prediction performance.

Our experimental results also confirm that neural networks are capable of learning the non-linear regularities in a time series by using only 200 training patterns. This number is important because a sample of this size is easier to obtain and easier to train. In our case this non-linearity is chaotic and the prediction results are very encouraging. Even with random noises introduced into the time series, the prediction results are still close to the optimal prediction. Of course when the random noises are too severe, the prediction results are not useful. But based on our results, we know the noise intensity affects a neural network's performance only mildly. This again confirms the noise tolerance capability of neural networks. Another interesting phenomenon is that the relative error is also close to the ratio between the weights of the stochastic and deterministic parts. But the implications of this relationship are beyond the scope of this paper.

Three-layer back-propagation FFNNs are by no means the only kind of neural networks that can be applied to time series forecasting. Among others, the recurrent neural networks seem to be more suitable for exploiting temporal relations. However, the complexity of implementing and training this kind of neural networks is so great that it severely limits their practicality. On the other hand, it is easy to implement back-propagation FFNNs on hardware and to put into practical use.

Reference

[1] A.A. Lazar and G. Pacifici, "Control of resources in broadband networks with quality of service guarantees," *IEEE Communications Magazine*, pp. 66-73, October 1991.

[2] J.-T. Amenyo, A.A. Lazar and G. Pacifici, "Cooperative distributed scheduling for ATS-based broadband integrated networks," CTR Technical Report 240-91-21, Center for Telecommunications Research, Columbia University, August, 1991.

[3] J.J. Bae and T. Suda, "Survey of traffic control schemes and protocols in ATM networks," *Proceedings of the IEEE*, vol. 79, No. 2, pp. 170-189, February 1991.

[4] G.M. Woodruff, R.G.H. Rogers and P.S. Richards, "A congestion control framework for high speed integrated packetized transport," *Proceedings of the IEEE Globecom Conference*, pp. 203-207, 1988.

[5] G.E. Box and G.M. Jenkins, *Time Series Analysis: forecasting and control*, Holden-Day, 1976.

[6] K. Hornik, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.

[7] Funahashi, "On the approximate realization of continuous mapping by neural networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[8] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp.303-314, 1989.

[9] H. White, "Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings," *Neural Networks*, vol. 3, pp. 535-549, 1990.

[10] D.E. Rumelhart, G. Hinton and R.J. Williams, "Learning representations by backpropagating errors," *Nature*, 323, 1986.

[11] B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson and J.D. Robbins, "Performance models of statistical multiplexing in packet video communications," IEEE Trans. Communications, vol. 6, no. 7, July 1988.

[12] J.D. Farmer and J.J. Sidorowich, "Predicting chaotic time series," *Physical Review Letters*, series B, vol. 59, no. 8, pp. 845-848, 1987.

[13] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: prediction and system modeling," Tech. Rep. LA/UR 87/2662, Los Alamos National Lab., Los Alamos, NM, 1987.

[14] E. Hartman and J.D. Keeler, "Predicting the future: advantages of semilocal units," *Neural Computation*, vol.3, no. 4, pp. 566-578, 1991.