27 Case Study 5: Prediction

Objectives

This chapter presents a case study in using neural networks for prediction. Prediction is a kind of dynamic filtering, in which past values of one or more time series are used to predict future values. Dynamic networks, such as those described in Chapter 10 and Chapter 14, are used for filtering and prediction. Unlike the previous case studies, the input to these dynamic networks is a time sequence.

There are many applications for prediction. For example, a financial analyst might want to predict the future value of a stock, bond, or other financial instrument. An engineer might want to predict the impending failure of a jet engine. Predictive models are also used for system identification (or dynamic modeling), in which we build dynamic models of physical systems. These dynamic models are important for analysis, simulation, monitoring and control of a variety of systems, including manufacturing systems, chemical processes, robotics and aerospace systems. In this chapter we will demonstrate the development of predictive models for a magnetic levitation system.

Theory and Examples

This chapter presents a case study in using neural networks for prediction. In this case study, the predictor neural network is used to model a dynamic system. This dynamic modeling from data is referred to as system identification. System identification can be applied to a variety of systems: economic, aerospace, biological, transportation, communication, manufacturing, chemical process, etc. For this case study, we will consider a simple magnetic levitation system. Magnetic levitation has been used for many years in transportation systems. In our simple maglev system, we will suspend a magnet above an electromagnet. A maglev train works on a similar principle.

Description of the Magnetic Levitation System

The objective of this maglev system is to control the position of a magnet suspended above an electromagnet, where the magnet is constrained so that it can only move in the vertical direction, as shown in Figure 27.1.

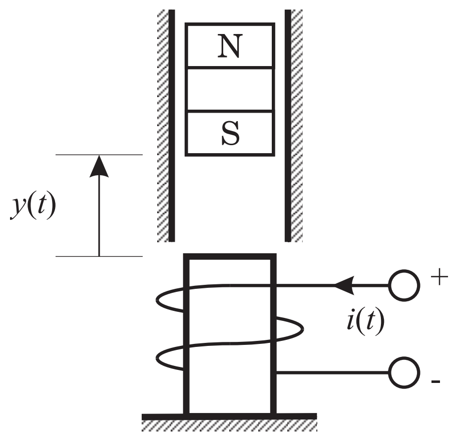

Figure 27.1 Magnetic Levitation System

The equation of motion for this system is

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M}\frac{i^2(t)\,\mathrm{sgn}[i(t)]}{y(t)} - \frac{\beta}{M}\frac{dy(t)}{dt} \qquad (27.1)$$

where $y(t)$ is the distance of the magnet above the electromagnet, $i(t)$ is the current flowing in the electromagnet, M is the mass of the magnet, and g is the gravitational constant. The parameter β is a viscous friction coefficient that is determined by the material in which the magnet moves, and α is a field strength constant that is determined by the number of turns of wire on the electromagnet and the strength of the magnet. For our case study, the parameter values are set to $\beta = 12$, $\alpha = 15$, $g = 9.8$, $M = 3$.

The objective of the case study will be to develop a dynamic neural network model that can predict the next value of the magnet position, based on previous values of the magnet position and the input current. Once the model has been developed, it can be used to find a controller that can determine the correct current to apply to the electromagnet, so as to move the magnet to some desired position. We won't go in to the control design in this case study, but the reader is referred to [HaDe02] and [NaMu97].

Data Collection and Preprocessing

For this case study, we did not build the maglev system of Figure 27.1. Instead, we created a computer simulation to implement Eq. (27.1). We used Simulink® to implement the simulation, but any simulation tool could be used. For our simulations, the current was allowed to range from -1 to 4 amps. The data were collected every 0.01 seconds.

To develop an accurate model, we need to be sure that the system inputs and outputs cover the operating range for which the system will be used. For system identification problems, we often collect training data while applying random inputs which consist of a series of pulses of random amplitude and duration. (This form of input is sometimes called a skyline function, because of its resemblance to a city skyline.) The duration and amplitude of the pulses must be chosen carefully to produce accurate identification. Figure 27.2 shows the input current, and the corresponding magnet position for our data set. A total of 4000 data points were collected.

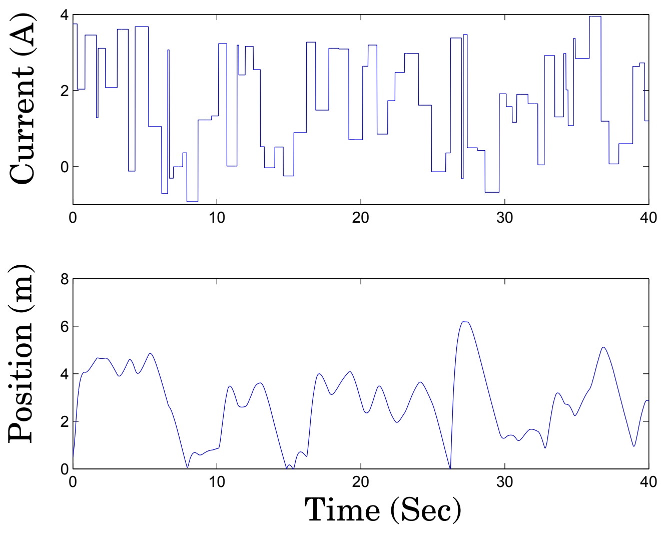

Figure 27.2 Magnetic Levitation Data

The skyline form of input function has the advantage that it can explore both transient and steady state operation of the system. Because some of the pulses are long, the system will approach steady state at the end of those pulses. Shorter width pulses explore transient operation of the system.

When steady state performance is poor, it is useful to increase the duration of the input pulses. Unfortunately, within a training data set, if we have too much data in steady state conditions, the training data may not be representative of typical plant behavior. This is due to the fact that the input and output signals do not adequately cover the region that is going to be controlled. This will result in poor transient performance. We need to choose the training data so that we produce adequate transient and steady state performance. This can be done by using an input sequence with a range of pulse widths and amplitudes.

After the data has been collected, the next step is to divide the data into training, validation and test sets. In this case, because we will be using the Bayesian regularization training technique, we do not need to have a validation set. We did set aside 15% of the data for testing purposes. When the input is a time sequence, it is useful to have the testing sequence consist of a contiguous segment of the original data set. For our tests, we used the last 15% of the data as the testing set.

The data were scaled using Eq. (22.1), so that both the inputs and the targets were in the range [-1,1]. The resulting scaled data is shown in Figure 27.3

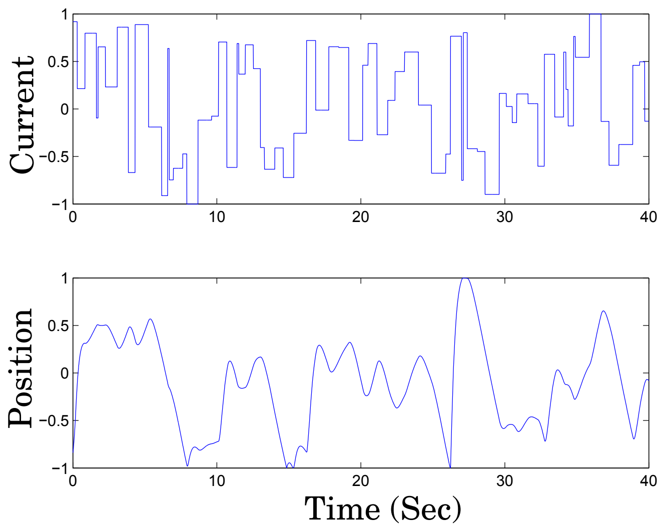

Figure 27.3 Scaled Data

Selecting the Architecture

There are many dynamic network architectures that can be used for prediction. A popular architecture is the nonlinear autoregressive network with exogenous inputs (NARX) network, which was discussed in Chapter 22. The NARX network is a recurrent dynamic network, with feedback connections enclosing several static layers of the network. The NARX model is based on the linear ARX model, which is commonly used in time series modeling.

The defining equation for the NARX model is

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n_y), u(t-1), u(t-2), \ldots, u(t-n_u)) , \quad (27.2)$$

where the next value of the dependent output signal $y(t)$ is regressed on previous values of the output signal and previous values of an independent (exogenous) input signal $u(t)$. (For our application, $y(t)$ is the position of the magnet, and $u(t)$ is the current going into the electromagnet.) We can implement the NARX model by using a feedforward neural network to approximate the function $f(\ )$. A diagram of the resulting network is shown in the Figure 27.4, where a two-layer feedforward network is used for the approximation. The output of the last layer of the network is the prediction of the next value of the magnet position. The network input is the current into the electromagnet.

We are using the tan-sigmoid transfer function in the hidden layer, and a linear output layer. As with the standard multilayer network, the number of neurons in the hidden layer, S^1, will depend on the complexity of the system being approximated. We will discuss this choice in the next section.

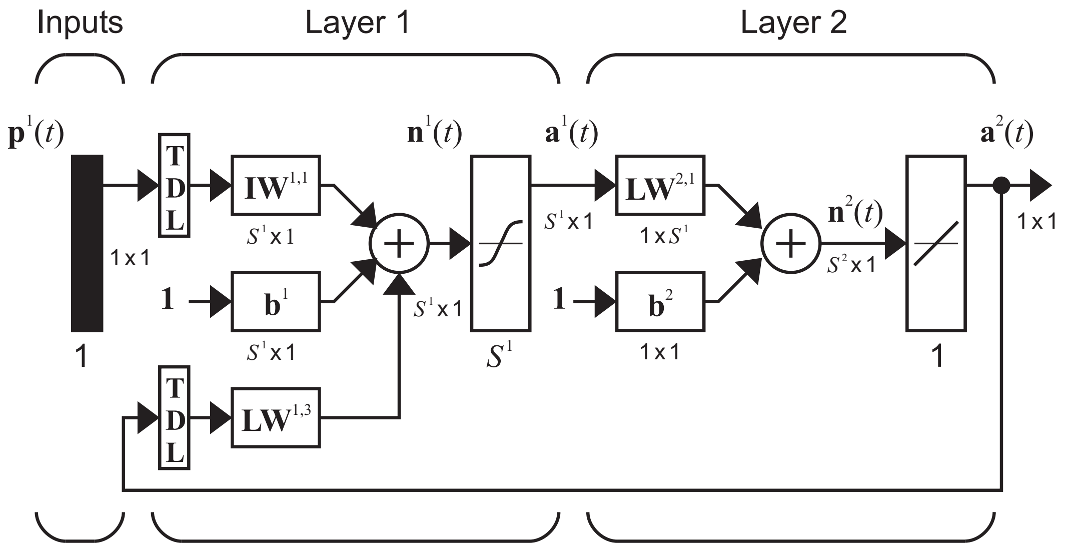

Figure 27.4 NARX Network Architecture

To define the architecture, we also need to set the length of the tapped-delay lines. The TDL for the inputs will contain the variables $u(t-1), \ldots, u(t-n_u)$, and the TDL for the outputs will contain the variables $y(t-1), \ldots, y(t-n_y)$. The TDL lengths n_u and n_y need to be defined. Because the defining differential equation in Eq. (27.1) is second order, we will start with $n_y = n_u = 2$. Later, we will investigate other possibilities.

Before demonstrating the training of the NARX network, we need to present an important configuration that is useful in training. We can consider the output of the NARX network to be an estimate of the output of the nonlinear dynamic system that we are trying to model. The output is fed back to the input of the feedforward neural network, as part of the standard NARX architecture, as shown on the left side of Figure 27.5. Since the true output is available during the training of the network, we could create a series-parallel architecture (see [NaPa90]), in which the true output is used instead of feeding back the estimated output, as shown on the right side of Figure 27.5. This has two advantages. The first is that the input to the feedforward network will be more accurate. The second is that the resulting network has a purely feedforward architecture, and static backpropagation can be used for training.

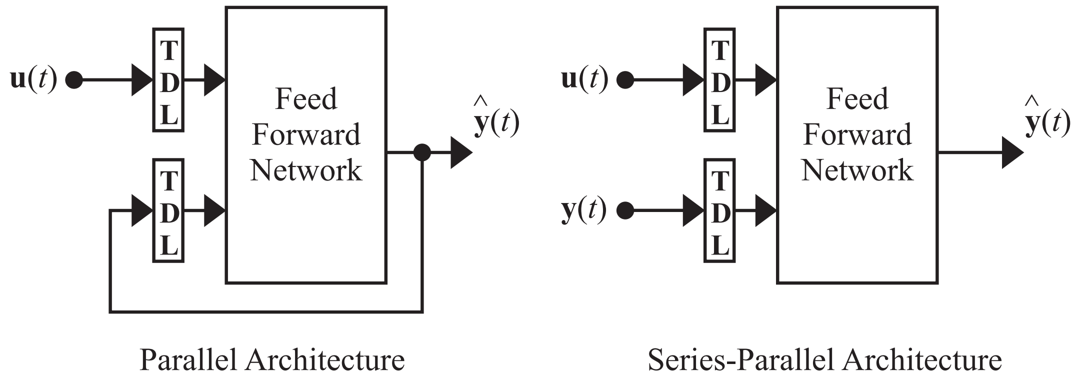

Figure 27.5 Parallel and Series-Parallel Forms

Using the series-parallel form, we can actually use a standard multilayer network to implement the NARX model. We can create an input vector that consists of previous system inputs and outputs:

$$\mathbf{p} = \begin{bmatrix} u(t-1) \\ u(t-2) \\ y(t-1) \\ y(t-2) \end{bmatrix}. \tag{27.3}$$

The target is then the next value of the output:

$$\mathbf{t} = \begin{bmatrix} y(t) \end{bmatrix}. \tag{27.4}$$

Training the Network

We used the Bayesian regularization training algorithm, described in Chapter 13, to train the NARX network, after the weights were initialized using the Widrow/Nguyen method (see page 22-13). The prediction problem is similar to the function approximation problem, which was demonstrated in Chapter 23, and the Bayesian regularization method is effective for both applications.

Because we have 4000 data points, and the number of network weights and biases will be less than 100 (as we will see later), the chances of overfitting are very small. We do not need to use Bayesian regularization (or early stopping) in this case. However, because it can tell us the effective number of parameters, we like to use it whenever it is appropriate.

Figure 27.6 illustrates the sum square error versus iteration number, while using Bayesian regularization. We used a network with 10 neurons in the hidden layer ($S^1 = 10$) for this case. The network was trained for 1000 iterations, at which time the performance was changing very little. Several different networks were trained with different initial conditions, and the final SSE was similar for each, so we can be confident that we did not reach a local minimum.

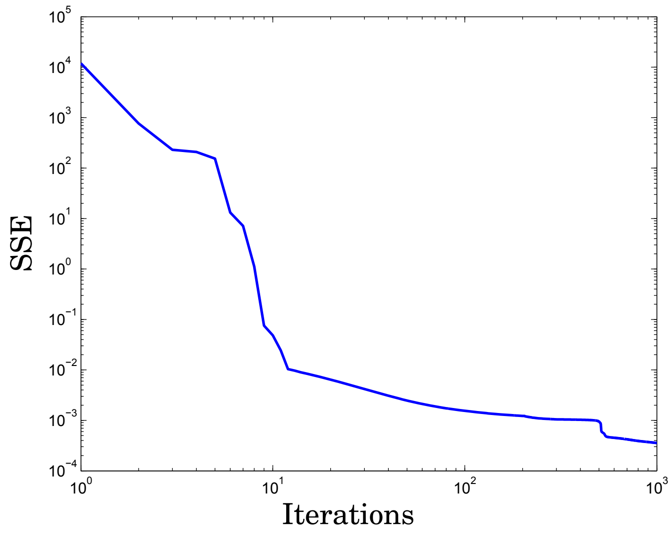

Figure 27.6 Sum Squared Error vs. Iteration Number ($S^1 = 10$)

In Figure 27.7, we can see the variation of the effective number of parameters γ during training. It eventually converges to 39. There are a total of 61 parameters in this 4-10-1 network, so we are effectively using less than 2/3 of the weights and biases. If the effective number of parameters was close to the total number of parameters, then we would increase the number of hidden neurons and retrain the network. That is not the case here.

There is no need to decrease the number of neurons, since the network computation time is not critical for this application. The only other reason for reducing the number of neurons would be to prevent overfitting. In terms of preventing overfitting, having 39 effective parameters is equivalent to having 39 total parameters. That is the beauty of using the Bayesian regularization technique. This method chooses the correct number of parameters for each problem, as long as we have a sufficient number of potential parameters in the network.

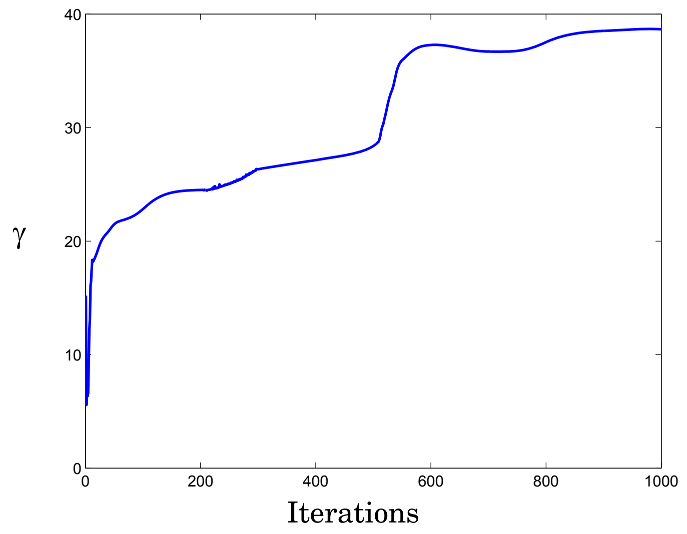

Figure 27.7 Effective Number of Parameters ($S^1 = 10$)

Validation

As we discussed in previous chapters, an important tool for network validation is a scatter plot of network outputs versus targets, as shown in Figure 27.8 (in normalized units). The figure on the left shows the training data, while the figure on the right shows the testing data. Because the testing data fit is as good as the training data fit, we can be confident that the network did not overfit.

For prediction problems, there is another set of tools for model validation. These tools are based on two basic properties of accurate prediction models. The first property is that the prediction errors,

$$e(t) = y(t) - \hat{y}(t) = y(t) - a^2(t), \tag{27.5}$$

should be uncorrelated with each other from one time step to the next. The second property is that prediction errors should be uncorrelated with the input sequence $u(t)$. (See [BoJe86].)

If the prediction errors were correlated with each other, then we could use that correlation to improve the predictions. For example, if prediction errors one time step apart had a positive correlation, then a large positive prediction error at the current time point would suggest that the prediction error at the next time point would also be positive. By lowering our next prediction, we could then reduce the next prediction error.

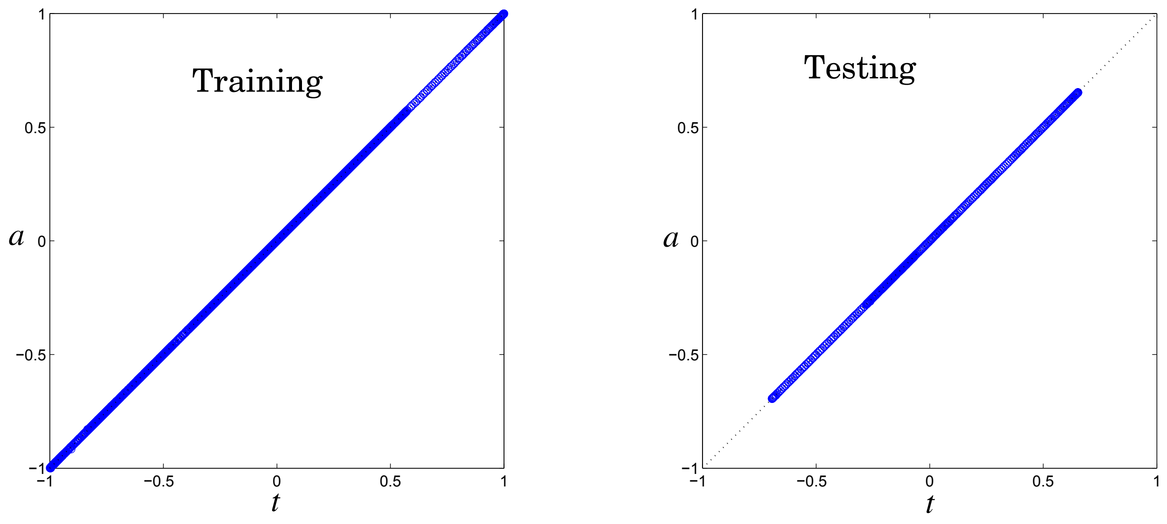

Figure 27.8 Scatter Plots of Network Outputs vs. Targets - Training and Testing Sets

The same argument holds for correlation between the input sequence and the prediction error. For accurate prediction models, there should be no correlation between the input and the prediction error. If there was correlation, then we could use this correlation to improve the predictor.

To measure the correlation in a time sequence, we use the autocorrelation function, which can be estimated by

$$R_e(\tau) = \frac{1}{Q-\tau} \sum_{t=1}^{Q-\tau} e(t)e(t+\tau) . \tag{27.6}$$

The autocorrelation function of the prediction error (in normalized units) of our trained network for the maglev problem is shown in Figure 27.9.

For the prediction error to be uncorrelated (termed “white” noise), the autocorrelation function should be an impulse at $\tau = 0$, with all other values equal to zero. Because Eq. (27.6) provides only an estimate of the true autocorrelation function, the values at $\tau \neq 0$ will never be exactly equal to zero. If the error sequence is white noise, then we can find confidence intervals for the $R_e(\tau)$ (see [BoJe86]) defined by

$$\pm 2\frac{R_e(0)}{\sqrt{Q}}. \tag{27.7}$$

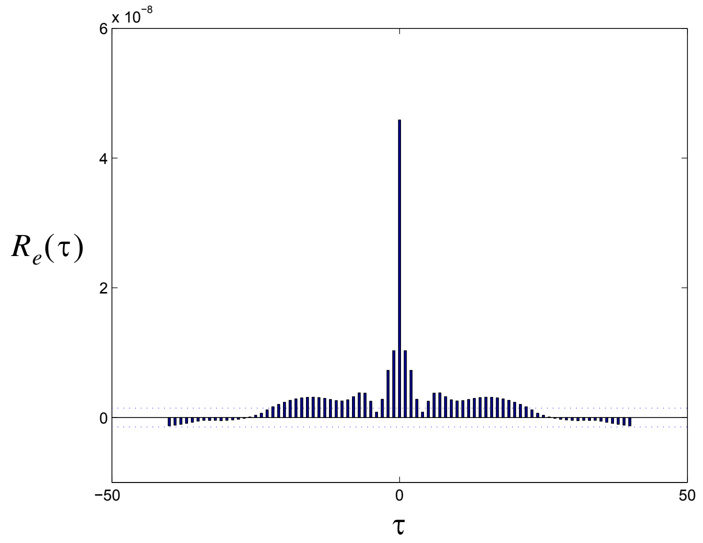

Figure 27.9 $R_e(\tau)$ ($n_y = n_u = 2$, $S^1 = 10$)

The dashed blue lines in Figure 27.9 indicate these confidence bounds. We can see that the estimated autocorrelation function for the prediction errors falls outside these bounds at a number of points. This indicates that we may need to increase n_y and n_u.

To measure correlation between the input sequence $u(t)$ and the prediction error $e(t)$, we use the cross-correlation function, which can be estimated by

$$R_{ue}(\tau) = \frac{1}{Q-\tau}\sum_{t=1}^{Q-\tau} u(t)e(t+\tau). \tag{27.8}$$

The cross-correlation function $R_{ue}(\tau)$ (in normalized units) of our trained network for the maglev problem is shown in Figure 27.9.

As with the estimated autocorrelation function, we can define confidence intervals to determine if the cross-correlation function is near zero

$$\pm 2\frac{\sqrt{R_e(0)}\sqrt{R_u(0)}}{\sqrt{Q}}. \tag{27.9}$$

The dashed blue lines in Figure 27.10 represent these confidence bounds. The cross-correlation function remains within these bounds, so it does not indicate a problem.

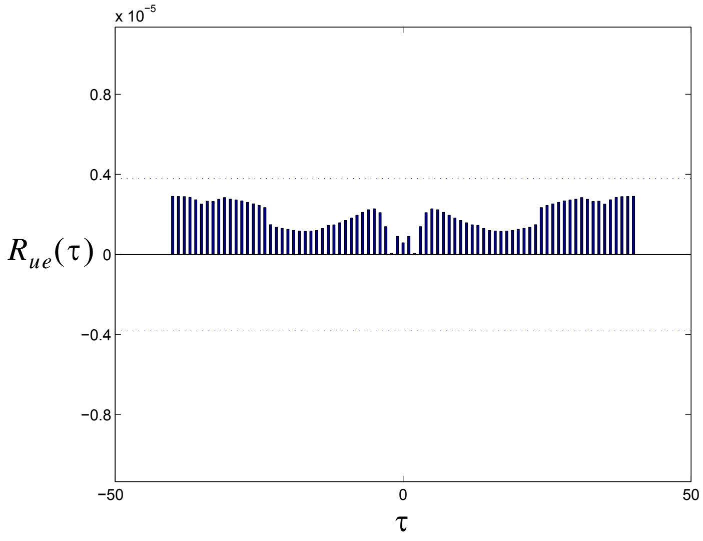

Figure 27.10 $R_{ue}(\tau)$ $(n_y = n_u = 2, S^1 = 10)$

Because the autocorrelation function of the prediction errors in Figure 27.9 indicated correlation in the errors, we increased the n_y and n_u values from 2 to 4 and retrained our neural network predictor. The resulting estimated autocorrelation function is shown in Figure 27.11. Here we can see that $R_e(\tau)$ falls within the confidence bounds, except at $\tau = 0$, which indicates that our model is performing correctly.

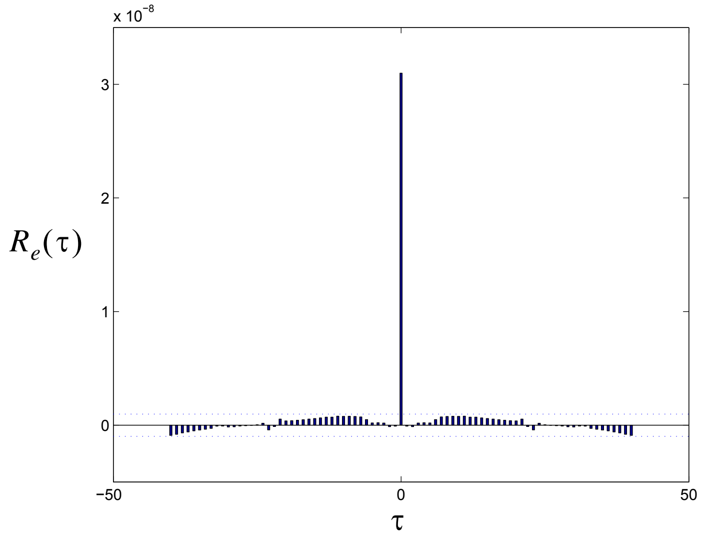

Figure 27.11 $R_e(\tau)$ $(n_y = n_u = 4, S^1 = 10)$

The estimated cross-correlation function, with the increased delay order, is shown in Figure 27.12. All of the points are well within the zero confidence interval. There is no significant correlation between the errors and the input.

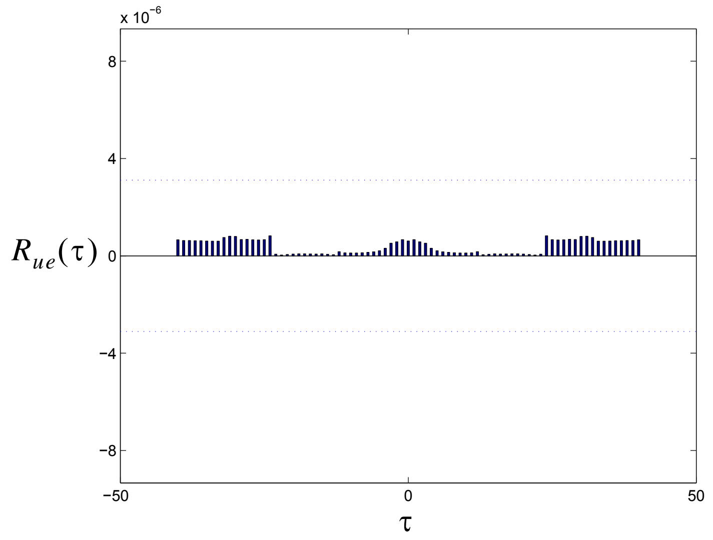

Figure 27.12 $R_{ue}(\tau)$ ($n_y = n_u = 4$, $S^1 = 10$)

With $n_y = n_u = 4$, we have white prediction errors, and there is no significant correlation between the prediction errors and the model input. It appears that we have an accurate prediction model.

The errors for the final prediction model are shown in Figure 27.13. We can see that the errors are very small. However, because of the series-parallel configuration, these are errors for only a one-step-ahead prediction. A more stringent test would be to rearrange the network into the original parallel form and then to perform an iterated prediction over many time steps. We will now demonstrate the parallel operation.

Figure 27.14 illustrates the iterated prediction. The solid line is the actual position of the magnet, and the dashed line is the position predicted by the NARX neural network. The network prediction is very accurate - even 600 time steps ahead.

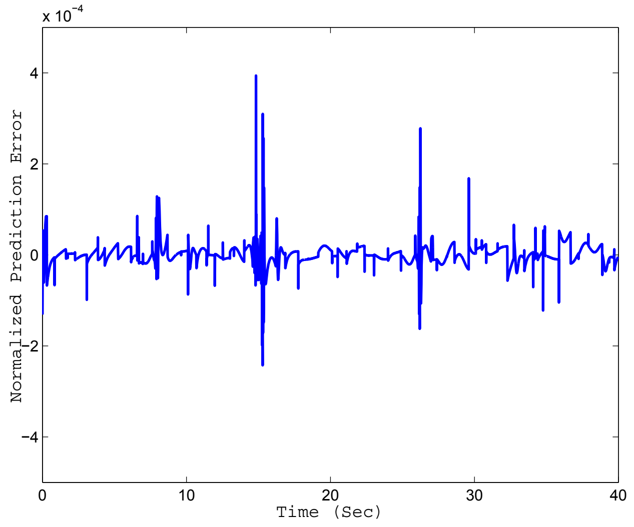

Figure 27.13 Prediction Errors vs. Time

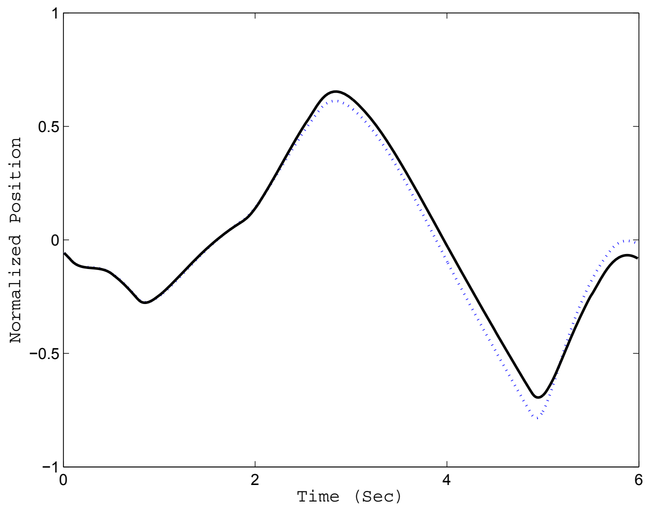

Figure 27.14 Iterated Prediction for the Maglev NARX Network

Data Sets

There are two data files associated with this case study:

- maglev_u.txt — contains the input sequence in the original data set
- maglev_y.txt — contains the output sequence in the original data set

They can be found with the demonstration software, which is described in Appendix C.

Epilogue

This chapter has demonstrated the use of multilayer neural networks for prediction, in which a future value of a time series is predicted from past values of that series and potentially other series. In this case study, the prediction network was used as a model of a magnetic levitation system. This modeling of dynamic systems is referred to as system identification.

A Nonlinear Autoregressive model with eXogenous inputs (NARX) network is well suited to this problem, and Bayesian regularization is an excellent training algorithm to use in this situation.

Further Reading

[BoJe94] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, Fourth Edition, Wiley, 2008.

A classic textbook on time series analysis and the development of prediction models.

[HaDe02] M. Hagan, H. Demuth, O. De Jesus, “An Introduction to the Use of Neural Networks in Control Systems,” *International Journal of Robust and Nonlinear Control*, vol. 12, no. 11, pp. 959-985, 2002.

This survey paper describes some practical aspects of using neural networks for control systems. Three neural network controllers are demonstrated: model predictive control, NARMA-L2 control, and model reference control.

[NaMu97] Narendra, K.S.; Mukhopadhyay, S., “Adaptive control using neural networks and approximate models,” *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 475 - 485, 1997.

This paper introduced the NARMA-L2 model and controller. Once the NARMA-L2 model is trained, it can be easily inverted to form a controller for the identified system.

[NaPa90] K. S. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

Classic early paper on the use of neural networks for the identification and control of dynamical systems.