EE 5239: Estimating Aerodynamic Coefficients using Feed-forward Neural Network (FFNN)

Kerry Sun

December 21, 2017

Abstract

This final report presents an implementation of estimating longitudinal aerodynamic coefficients of an aircraft using artificial neural network. Specially, Feed-forward Neural Network is used to model the lift, drag and pitching moment coefficients of an Ultrastick120e UAV research aircraft from AEM UAV lab. The data was generated using an in-house nonlinear aircraft Simulink model. Different tuning parameters were explored extensively for fast convergence. It was shown FFNN has a good predicting capability estimating aerodynamic coefficients when appropriate tuning parameters were properly chosen.

1 Introduction

Aircraft system identification is traditionally done based on the physical insight. Developing such models require a good understanding of the underlying physics. Artificial Neural Networks (ANNs) provide an alternative approach to model building. The ANNs are neuroscience-inspired computational tools and are often used for pattern recognition. They provide a general framework for nonlinear functional mapping of the input-output subspace, and as such are part of the system identification methodology which deals implicitly with the measured input-output neural networks.

The unique feature of ANNs is its highly interconnected structure spread over multiple layers, each with a large number of simple processing elements. A typical ANN consists of an input layer, one or more hidden layers , and output layer. The number of nodes in the input and output layers is automatically fixed through the numbers of input-output variables. The processing elements in the hidden layer are invariably continuous, nonlinear functions such as sigmoidal (S-shape) functions. The ANNs are trained to adjust the free parameters, namely the weights associated with the various nodal connections so that error between the measured output and network output to the same input will be minimized, thus amounts a classical optimization problem.

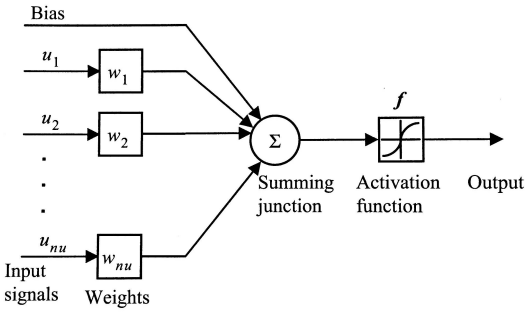
Several types of neural networks result from different ways of interconnecting the neurons and the way the nodes function in hidden layers. There are three networks most commonly used in system identification: 1) feedforward neural network (FFNN), 2) recurrent neural network (RNN), and 3) radial basis function (RBF) neural network. In this project, FFNNS will be used. FFNNs process the information in only one direction through the various layers. They are the simplest of the NNs.

In the case of flight vehicles, NNs are used for a variety of different applications such as structural damage detection and identification [1], modeling of aerodynamic characteristics from flight data [2], autopilot controllers and advanced control laws for applications such as carefree maneuvering [3]. Although FFNNs have been used for the applications mentioned above, engineers well versed with the so-called conventional approach always had in the past some hesitations in applying these techniques. This is partly because of the nature of NNs; it is empirical and has a black-box structure without explicit formulation of dynamic system equations. Nevertheless, we will demonstrate FFNN modeling capabilities by modeling lift, drag and pitch moment coefficients from the simulated flight data.

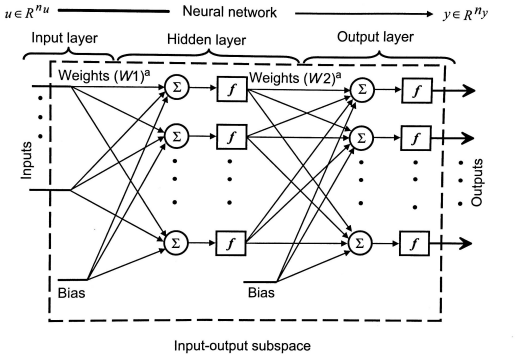
2 Basics of Neural Network

The basic building block of neural network comprises a computational element called neuron, shown in Fig. 1(a). Each neuron receive inputs from multiple sources, denoted by $u(n_u \times 1)$ vector. These inputs are multiplied by effectiveness factors called weights (W_i) and added up to yield the weighted sum which passes through an activation function f to provide an output. Thus, the input-output characteristic of a neuron depends on the weights W and the activation function f.

A feedforward neural network, shown in Fig. 1(b), consists of a large number of neurons which are interconnected and arranged in multiple layers. In Fig. 1(b), one hidden layer is used for FFNN. The internal behavior of the network is characterized by the inaccessible (hidden) layer. In order to cater any general nonlinear problem, it is required that activation function of the hidden layer must be nonlinear, whereas that in the output layer may be linear or nonlinear.



(a) Schematic of a single neuron [4].



(b) Schematic of a feedforward neural network with one hidden layer[4].

Figure 1: Schematic of a Neural Network

The performance of neural network depends on the size of networks, the number of hidden layers, and the number of neuron in each hidden layer. However, the larger the size is, the larger the computational effort. In general, it has been shown that an FFNN with a single hidden layer and any continuous sigmoidal activation function can arbitrarily well approximate any given input-output subspace [5]. The practical investigations reported in the past have also shown a single hidden layer FFNN with adequate number of neuron is sufficient for aerodynamic modeling from flight data [4].

The choice of number of neurons in the hidden layer problem-dependent, and it also depends on the training algorithm. In the NN literature several rules of thumb have been suggested for n_h, the number of neuron in the hidden layer, such as: 1) between n_u and n_y, 2) two-thirds of $(n_u + n_y)$, or 3) less than $2n_u$. However, those rules are doubtful because they do not account for issues such as noise and complexity. Too little leads to under-fitting, yet too many nodes lead to over-fitting. The best approach is based on numerical investigation of trying out many networks with different number of hidden neurons.

There are 2 parts in the application of FFNN given a set of data: training and prediction. During the first part, the network is exposed to given data set, and the weights are continuously adjusted until some convergence criterion is met. This process essentially captures the input-output characteristics. The second part involves checking prediction capability with the fixed weights using the same or different set of data.

3 Training Algorithm

In order to determine the weights shown in Fig. 1(b), Back-propagation (BP) is the most commonly used method. The basic idea behind this method is view error as a function of network weights and perform gradient descent in the parameter space to search for a minimum error between the desired and

estimated values. BP provides a means to compute the gradients, starting at output layer and going backwards. The minimization is based steepest descent method. The training algorithm comprises two steps: 1) forward pass, that is, keeping the weights fixed the input signal is propagated through the network; 2)backward pass, which propagates the error backwards through various layers, computes the gradients, and updates the weights. We use a recursive BP method to update the network weights. Fig. 2 is a schematic of forward pass and back-propagation computations. The details will be explained in the next section.

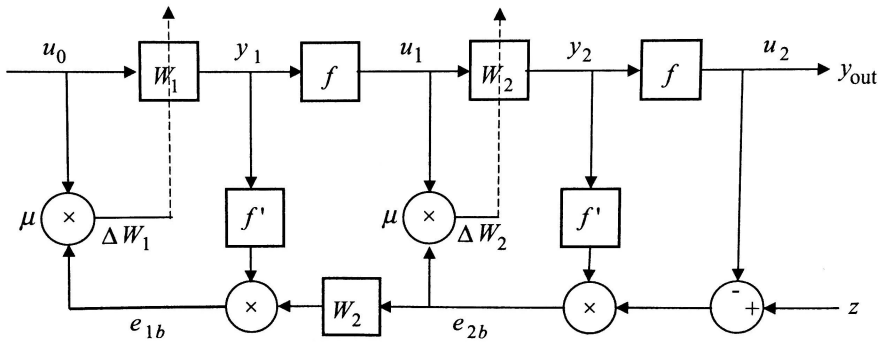


Figure 2: Schematic of a forward pass and back-propagation [4].

3.1 Forward Propagation

Let's denote the weights of a FFNN with a single hidden layer shown in Fig. 1(b):

W_1: weight matrix between input and hidden layer ($n_h \times n_u$)

W_{1b}: bias weight vector between input and hidden layer ($n_h \times 1$)

W_2: weight matrix between hidden and output layer ($n_y \times n_h$)

W_{2b}: bias weight vector between hidden and output layer ($n_y \times 1$)

where n_u is number of nodes in the input layer, n_h is the number of nodes in the hidden layer, and n_y is the number nodes in the output layer. For a given input vector u_0, propagation from input layer to hidden layer yields:

$$y_1 = W_1 u_0 + W_{1b} \tag{1}$$

$$u_1 = f(y_1) \tag{2}$$

where y_1 is the vector of intermediate variables, u_1 is the vector of node outputs at the hidden layer and f is the vector of nonlinear sigmoidal node activation functions. The hyperbolic tangent function is chosen for FFNN in this project:

$$f_i(y_1) = tanh(\frac{\gamma_1}{2} y_1(i)) = \frac{1 - e^{-\gamma_1 y_1(i)}}{1 + e^{-\gamma_1 y_1(i)}} \tag{3}$$

where γ_1 is the slope factor of the hidden layer activation function. All the nodes at any particular layer have same slope factor generally.

The node outputs u_1 at the hidden layer from the inputs to the next layer. Propagation of u_1 from the hidden layer to the output layer yields

$$y_2 = W_2 u_1 + W_{2b} \tag{4}$$

$$u_2 = f(y_2) \tag{5}$$

where y_2 is the vector of intermediate variables (output of the summing junctions), u_2 is the vector node outputs at the output layer and f is the vector of sigmoidal functions (same as the one in Eq. 3).

Thus, propagation of the inputs u_0 through the input, hidden and output layers using known (fixed) weights yields the network estimated outputs u_2. Note the bias is not shown explicitly in Fig. 2. The goal of optimization is to find optimal values for the weights such that u_2 match with the measured system responses z.

3.2 Backward Propagation

The backward propagation algorithm is based on minimizing the local output error cost function, which is the sum of the squared errors given by:

$$E(k) = \frac{1}{2}[z(k) - u_2(k)]^T[z(k) - u_2(k)] = \frac{1}{2}e^T(k)e(k) \tag{6}$$

In order to minimize Eq. 6, we apply the steepest descent method [6] which yields:

$$W_2(k+1) = W_2(k) + \mu(-\frac{\partial E(k)}{\partial W_2}) \tag{7}$$

where μ is the stepsize or learning rate parameter and $\frac{\partial E(k)}{\partial W_2}$ is the local gradient of the error cost function with respect to W_2. A positive constant stepsize is used to ensure reasonable convergence rate. Substituting Eq. 4 and 5 in Eq. 6, and the partial differentiation with respect to the element of W_2 yield the gradient of the cost function:

$$\frac{\partial E(k)}{\partial W_2} = -f^{'}[y_2(k)][z(k) - u_2(k)]u_1^T(k) \tag{8}$$

where $f^{'}[y_2(k)]$ is the derivative of the output node activation function shown in the following:

$$f^{'}(y_i) = \frac{\gamma_i}{2}[1 - tanh^2(\frac{\gamma_l y_i}{2})] = \frac{2\gamma_l e^{-\gamma_l y_i}}{(1 + e^{-\gamma_l y_i})^2} \tag{9}$$

where l (= 1 or 2) is the index for the input-to-hidden and hidden-to-output layers.

Now substituting Eq. 8 into Eq. 7, we obtain the weight-update rule for the output layer:

$$\begin{aligned} W_2(k+1) &= W_2(k) + \mu f^{'}[y_2(k)][z(k) - u_2(k)]u_1^T(k) \\ &= W_2(k) + \mu e_{2b}(k)u_1^T(k) \end{aligned} \tag{10}$$

Similarly, by substituting Eq. 1 and 2 in Eq. 6, we obtain the partial differentiation with respect to W_1:

$$\frac{\partial E(k)}{\partial W_2} = -f^{'}[y_1(k)]W_2^T e_{2b}(k)u_0^T(k) \tag{11}$$

Therefore, the weight update rule for the hidden layer is:

$$\begin{aligned} W_1(k+1) &= W_1(k) + \mu f^{'}[y_1(k)]W_2^T e_{2b}(k)u_0^T(k) \\ &= W_1(k) + \mu e_{1b}(k)u_0^T(k) \end{aligned} \tag{12}$$

To summarize, we apply forward and backward propagation to update the weights in both hidden and output layers recursively for N, where N is number of data points. At the end of the recursive loop, the mean square error (MSE) is computed as the following:

$$\sigma^2 = \frac{1}{Nn_y} \sum_{N}^{k=1} \sum_{n_y}^{j=1} [z_j(k) - u_j(k)]^2 \tag{13}$$

where n_y is number of output, N is number of data points. z is the output and u is the input.

The algorithm will stop when the MSE satisfies some convergence criterion, for example the relative change from iteration to iteration less than a specified value or until the maximum number of iterations is reached.

3.3 Modified Backward Propagation with Momentum Term

It was found out the convergence rate of the steepest descent optimization is slow, affected by poor initial weights. To overcome this problem, we modified backward-propagation that is commonly used in the NN applications; we add a momentum term [6] in Eq. 10 and 12 and obtain the following:

$$W_2(k+1) = W_2(k) + \mu e_{2b}(k) u_1^T(k) + \Omega[W_2(k) - W_2(k-1)] \tag{14}$$

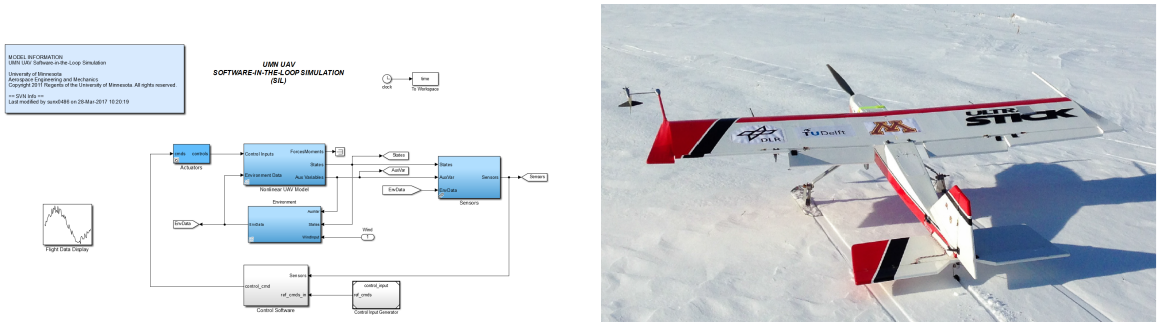
$$W_1(k+1) = W_1(k) + \mu e_{1b}(k) u_0^T(k) + \Omega[W_1(k) - W_1(k-1)] \tag{15}$$

where Ω is called the momentum factor. The different $[W_l(k) - W_l(k-1)]$ makes the optimization move along the temporal average direction used in the past iteration. It helps to damp out parasitic oscillations in the weight update. As Ω is chosen greater than zero and less than 1, it approximately amounts to increasing the stepsize from μ to $\mu/(1-\Omega)$.

4 Modeling of Longitudinal Data with Noise

4.1 Aircraft Modeling

An in-house nonlinear aircraft Simulink model shown in Fig. 3(a) was used for numerical analysis. The Simulink model was developed to include the full nonlinear model of the aircraft, sensors, actuators and environment (e.g. wind condition). The parameters of the aircraft in Simulink model came from Wind Tunnel testing on the in-house 120 UltraStick 120e Testbed shown in Fig. 3(b).

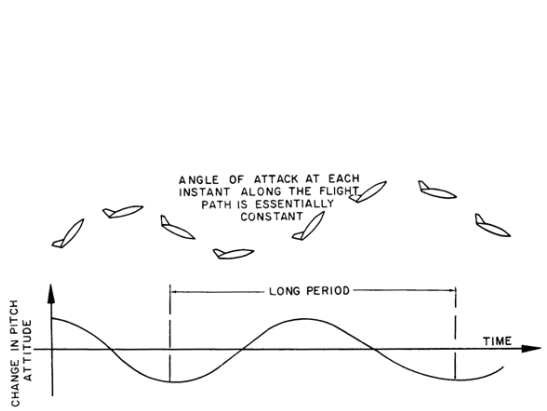


(a) Overview of Simulink Modeling.

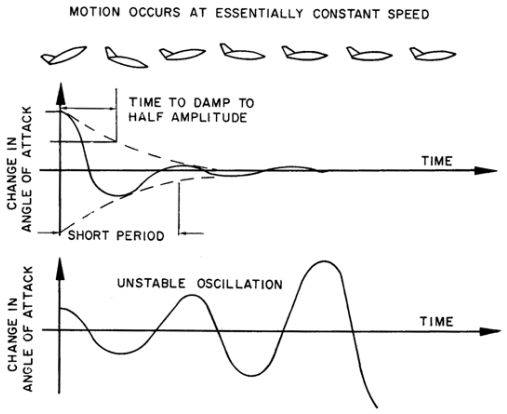
(b) University of Minnesota Ultra Stick 120e UAV Testbed.

Figure 3: Aircraft Modeling Using Simulink.

For demonstration purpose, we only make use of longitudinal motion of the aircraft. We will estimate a few aerodynamic coefficients of the aircraft. Those parameters are often used as a measure of the stability and controllability of an aircraft. For example, the dynamic stability of an aircraft refers to how the aircraft behaves after it has been disturbed following steady non-oscillating flight. Longitudinal motion consists of two distinct oscillations, a long-period oscillation called a phugoid mode and a short-period oscillation referred as the short-period mode. The phugoid mode was shown in Fig. 4(a) and the short-period mode was shown in Fig. 4(b). For short-period mode, the unstable oscillation would result aircraft to crash if not controlled properly. Thus, a typical flight experiment involves exciting those 2 modes on the aircraft to analyze the stability and controllability.



(a) Longitudinal Dynamic 1st mode: Phugoid mode.

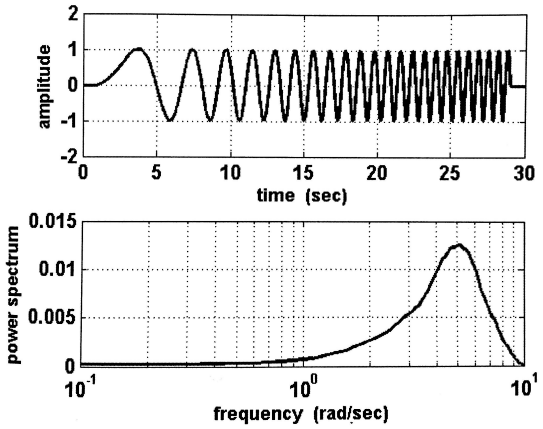


(b) Longitudinal Dynamic 2st mode: short-period mode.

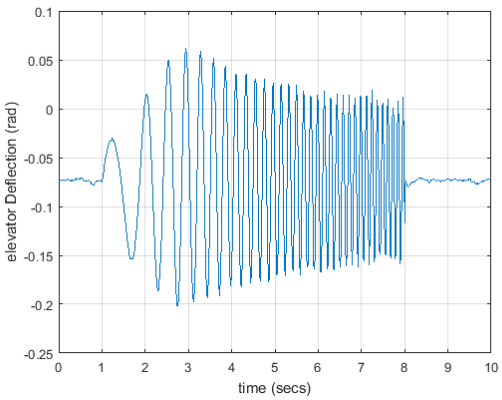
Figure 4: Modes of Longitudinal Dynamics [7].

4.2 Experiment Design

In our experiment, a chirp signal (frequency sweeps) was designed to excite those 2 modes. The idea behind this input is to apply a continuous sinusoid with the frequency increasing in time, so that the frequency content of the input covers a frequency band of interest. Fig. 5(a) shows an example of linear frequency sweep as well as its power spectrum. Fig. 5(b) shows the output of the elevator of the aircraft in response to its command from our simulation. Note the elevator output is passed through the actuator, that's why it doesn't have the same amplitude over time.



(a) An example of linear freqency sweep [8].



(b) An example of elevator output.

Figure 5: An example of Chirp signal: input and output.

Given the designed input, we can sufficiently excite the system to estimate the longitudinal modes. Furthermore, the lift, drag and pitching moment coefficients can be estimated if those modes are excited

by our input. Of course, the main goal is to model the lift, drag and pitching moment coefficients of the aircraft using a FFNN.

4.3 Longitudinal Data with Sensor Noise

Both input and output variables were generated through Simulink simulation. The total running time was 10 seconds. The chirp signal was used to excite the phugoid and short-period modes. Data was sampled at 50Hz. In particular, the following are the input variables and output variables for the FFNN simulation:

Number of independent variables (input): 4

Independent variables: δ_e, α, q, V

Number of dependent variables (output): 3

Independent variables: C_L, C_D, C_m

δ_e is the elevator (one of the flight control surfaces), α is angle-of-attack (angle between airflow and heading of the aircraft), q is pitch rate, and V is airspeed of the aircraft. C_L, C_D, C_m are lift, drag and pitching moment coefficients of the aircraft.

Two sets of data were generated: one without sensor noise for training, one with sensor noise for prediction. For the second set, the inputs are corrupted by sensor noise and dynamics of the actuator to make the problem more realistic. Note, there is no wind, gust and turbulence during this flight simulation.

5 Numerical Simulation Results

5.1 Optimal Tuning Parameters

Since standard Backward Propagation (BP) or BP with a momentum is characterized by a slow convergence and high sensitivity to parameter like learning rate and initial weights, we will explore the choices of tuning parameters in our simulation. Here are several interrelated parameters that influence the FFNN performance: 1) input and output data scaling, 2) initial network weights, 3) number of hidden nodes, 4) learning rate, 5) momentum parameter and 6) slope factors of the sigmoidal function. Based on numerical investigation in this project as well as literature reading for aerodynamic coefficient prediction, the following are used [4]:

Tuning Parameter		
Number of hidden layers	1	(1)
Number of hidden nodes	6	(5-8)
Data Scaling range	-0.5-0.5	(-0.5-0.5)
Nonlinear function slopes	0.85/0.6	(0.6-1.0)
Learning rate parameter	0.125	(0.1-0.3)
Momentum parameter	0.5	(0.3-0.5)
Initial random weights	0.3	(0.0-1.0)

where the typical values are provided in the third column.

As previous mentioned, one hidden layer is sufficient enough for aerodynamic modeling. Also, fewer than eight neurons in the hidden layer is sufficient for good predictive capability. Also, it was found out the error function in Eq. 13 is affected by the numerical values of the input-output data, particular if the values differ much in magnitude. Therefore, scaling was used reduce the error. The scaling method

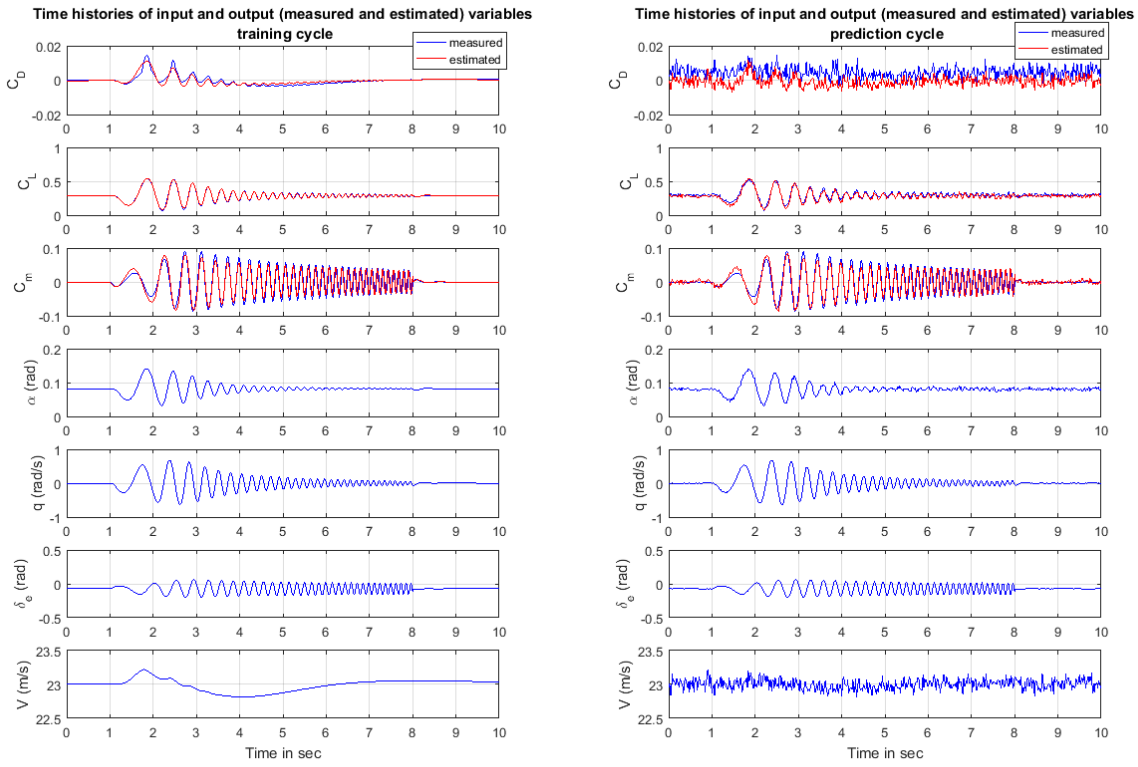
used here is to arrange the values between the chosen lower and upper limits such that all the variables have similar orders of magnitude. Note the scaling also helps improve convergence rate significantly for FFNN. The initial weights are set to small random numbers to avoid saturation in neurons.

Larger learning rate leads to a faster convergence, but the resulting NN may not have sufficient prediction capabilities. Also, inappropriate selection of other influencing parameters in combination with a large value of learning rate may lead to oscillations. The effect of momentum term, which was introduced to improved the convergence rate and damp out the larger oscillations in the network weights is not obvious in this example. Hence a general one was chosen.

For the slopes of the nonlinear activation functions of the hidden layer nodes and the output layer nodes, it was observed that very small slope factors close to zero have an adverse effect. It was also found out the performance is not good for the slope factor values of more than 1. In general, the slope factor of the hidden layer will have more influence on network convergence than the other. It has been confirmed by our example in this project.

5.2 Training and Prediction using FFNN

Both input and output are plotted in Fig. 6(a) and 6(b). In Fig. 6(a), the neural network was trained with noiseless data, and the estimated outputs and measured outputs match well. The resultant weights calculated from the training data were then used to predict on a noisy flight data set shown in Fig. 6(b). It can be seen that the estimated lift (C_L) and moment (C_m) match well with the measured output.



(a) Training using FFNN with the noiseless data.

(b) Prediction using the weights from training with the noisy data.

Figure 6: Training and Prediction for aerodynamics coefficient using FFNN with standard backward propagation (momentum term added)

However, the drag coefficient (C_D) is not matched well with the measured. This is perhaps because C_D is prone to noise from different sensors by nature, and magnitude of sensor error is on the same order

as the typical value of C_D. Hence the FFNN could not do a descent job separating the noise and true value.

Fig. 7 is the residual error in Eq. 13 versus number of iteration. We can see it converges in about 20 steps after carefully tuning all the parameters.

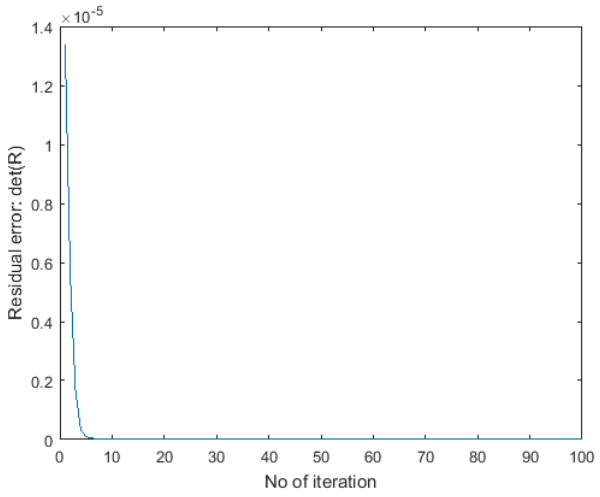


Figure 7: Residual Error vs iteration for FFNN.

6 Conclusion

A feedforward neural network (FFNN) implemented to predict aerodynamic coefficients was studied in this project. A standard backward propagation with momentum was used for updating the weights in the neurons. Different tuning parameters including number of hidden layers, number of hidden nodes, learning rate, nonlinear function slopes, momentum parameters, initial random weights, were carefully examined and chosen for fast convergence. The estimated aerodynamic coefficients match well with the measured except C_D. This is caused by the sensor noise, which is at same magnitude as the C_D. Hence the FFNN could not separate the noise and true value well. NN is a powerful to explore, but it has limitation such as we cannot interpret the weights physically.

References

[1] M.H.H. Shen P. Tsou. Structural damage detection and identification using neural newtorks. In *AIAA Journal*, 1994.

[2] R.A. Hess. On the use of back propagation with feed-forward neural network for the aerodynamic estimation problem. In *AIAA Paper*, 1993.

[3] M. R. Napolitano and M. Kincheloe. On-line learning neural-network controllers for autopilot systems. In *Journal of Guidance, Control and Dynamics*, volume 33, pages 1008–1015, 1995.

[4] R.V. Jategaonkar. Flight vehicle system identification: A time domain methodology. *AIAA Progress in Aeronautics and Astronautics*, No 216.

[5] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. In *Neural Networks*, pages 551–560. 1990.

[6] D.P. Bertsekas. Nonlinear programming. *Athena Scientific, Belmont, Massachusetts.*

[7] C.E. Dole. Flight theory and aerodynamics. *John Wiley Sons Inc New York NY*, 1998.

[8] E.A. Morelli and V. Klein. Aircraft system identification: Theory and practice. *AIAA Education in Aeronautics and Astronautics*.