A Neural Network Based Approach To Determine Stability Of A Dynamical System

Pritam Ghoshal

Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kalyanpur, Kanpur, 208016, Uttar Pradesh, India

Abstract

The Lavenberg-Marquardt neural network algorithm is used to predict the dominant eigenvalue of a multi-dimensional dynamical system. The aim is to find the optimal number of hidden layers and hidden neurons required to find the dominant eigenvalue. Also, a classification problem using the scaled conjugate gradient method has been solved to classify a given fixed point of the system as stable or unstable.

1. Introduction

Artificial Neural Network [1] is a data processing system that aims at predicting certain target parameters based on its training history. During its training, input parameters are fed into the system along with the collected or calculated target parameters that are associated with the input. The Neural Network morphs and adapts itself to match the target parameters based on the given input and some non-linear functions (sometimes linear) inbuilt in it. Neural network is based on the biological brain, which is a type of biological Neural Network consisting of numerous neurons. Due to its high accuracy of learning, good robustness and non-linear mapping, Neural Network is used in numerous areas such as medicine, weather prediction, sales forecasting, speech recognition, etc.

One of the most significant problems facing researchers is the selection of the type and number of hidden neurons for a specific problem. The ultimate aim is to design a Neural Network that can predict the required output with minimum error and, if possible, minimum time. Overfitting [2] is one of the main issues faced by Neural Network designers. The issue arises because the Neural Network matches the data so closely that it looses its generalization ability over the test data. Therefore, finding the optimum type and number of hidden neurons is necessary.

Following are the list of problems a researcher might face while developing a Neural Network for any kind of problem [3]

1. How many hidden layers should be used?
2. How many hidden neurons should be used in each layer?
3. How many training pairs should be used?

August 24, 2021

4. Which training algorithm should be used?

5. What neural network architecture should be used?

The stability of a Neural Network is measured through error calculation methods such as Root Mean Square Error (RMSE), Mean Relative Error (MRE), Mean Absolute Error (MAE), etc. Minimal error reflects better stability and higher error reflects worst stability.

This particular paper is concerned with predicting the dominant eigenvalue of a given Jacobian matrix. This has immense use in understanding the stability of a dynamical system (illustrated in the next section). Out of the five problems mentioned above, we tackle the first two problems. It is important to notice here that the input to the Neural Network for this problem is a matrix, which is typically not the case. Usually, it is just a bunch of parameter values (for instance see K. Gnana Sheela and S. N. Deepa [3]). Finally, we use the RMSE technique to study the stability of the developed Neural Network. Based on trial and error method, this paper seeks to find the optimum number of hidden layers and number of neurons in each hidden layer for a 4x4, 5x5, 6x6, 7x7 and 8x8 matrix. Also, a simpler Neural Network classification problem has been tackled at the end that can predict if the dynamical system is stable or unstable based on its Jacobian matrix.

2. Theory

In this section, we describe the motivation for finding the eigenvalues of a dynamical system followed by a brief description of the Lavenberg-Marquadt algorithm used in this paper.

2.1. Stability Of A Dynamical System

Let us consider a system with differential equations of the form

$$\dot{x} = f(x) \tag{1}$$

Here $x \in \mathbb{R}^n$ is the current state of the system and $\dot{x}$ represents the derivative of the current state with respect to time.

Let $q \in \mathbb{R}^n$ be a fixed point of the system such that $\dot{q} = 0$. We consider a small perturbation ξ about this fixed point. To see whether this perturbation grows or decays we linearise the system about this point. Expanding (1) about q we get,

$$\dot{q} + \dot{\xi} = f(q) + \left.\frac{\partial f}{\partial x}\right|_{x=q} \xi + \mathcal{O}(\xi^2) \tag{2}$$

or,

$$\dot{\xi} = J\xi \tag{3}$$

where J is the Jacobian matrix evaluated at the fixed point and defined by,

$$J = \begin{vmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{vmatrix} \tag{4}$$

The eigenvalues of J give an insight into the dynamics of the system. The eigenvalues may be real or complex. If the eigenvalues are real and all negative, the fixed point is stable and all solutions decay. If any of the eigenvalue is positive, any typical trajectory will approach the unstable manifold as $t \to \infty$. If the eigenvalues are complex, then the fixed point may be a center or a spiral. If the real part of the eigenvalue is 0, then the fixed point is a center for a linear system(further analysis is needed for a linearised system). If the real part is positive, then it is an unstable spiral, and if the real part is negative, then it is a stable spiral.

More about stability can be found in Strogatz[4]. For systems with large number of dimensions, it becomes difficult to solve the eigenvalue problem. Hence we seek an alternative method using neural network approach. It is evident from the above discussion that we are not interested in all the eigenvalues, rather only the maximum of the real part of the eigenvalues.

2.2. The Levenberg-Marquardt Backpropagation Algorithm

The primary aim of Levenberg-Marquardt Algorithm is to minimize the function

$$\phi(x) = ||F(x)||^2 \tag{5}$$

where $F(x) : R^n \to R^m$ is assumed to be continuously differentiable. Let z be a point in R^n. Provided x is sufficiently close to z, we can approximate $F(x)$ to be (linearization)

$$F(x) \approx \hat{F}(x; z) = F(z) + DF(z).(x - z) \tag{6}$$

where $DF(z)$ is the Jacobian of $F(x)$ evaluated at $x = z$. Here z basically represents our best guess or the current iterate. Let the iterates be represented as

$$x^{(1)}, x^{(2)}, x^{(3)}....x^{(k)}$$

At iteration k, from equation (6),

$$\hat{F}(x; x^{(k)}) = F(x^{(k)}) + DF(x^{(k)}).(x - x^{(k)})$$

We try to find $x^{(k+1)}$ such that it minimizes

$$||\hat{F}(x; x^{(k)})||^2 + \lambda^{(k)}||x - x^{(k)}|| \tag{7}$$

where λ is a parameter and $\lambda > 0$. In other words $x^{(k+1)}$ is a solution to the optimization problem

$$\text{minimize} \quad \left[||F(x^{(k)}) + DF(x^{(k)}.(x - x^{(k)})||^2 + \lambda^{(k)}||x - x^{(k)}||^2\right] \tag{8}$$

The solution to equation (8) is found to be

$$x^{(k+1)} = x^{(k)} - (DF(x^{(k)})^T DF(x^{(k)}) + \lambda^{(k)} I)^{-1} DF(x^{(k)})^T F(x^{(k)}) \tag{9}$$

Note here that since $\lambda^{(k)} > 0$, inverse always exists.

From equation (9), $x^{(k+1)} = x^{(k)}$ only when $DF(x^{(k)})^T F(x^{(k)}) = 0$. If this happens, optimality condition holds.

Update mechanism:

i) If $||F(x^{(k+1)})||^2 > ||F(x^{(k)})||^2$, accept new x and reduce λ by some factor. For instance,

$$\lambda^{(k+1)} = 0.7\lambda^{(k)}$$

ii) If $||F(x^{(k+1)})||^2 \leq ||F(x^{(k)})||^2$, increase λ and do not update x. For instance

$$\lambda^{(k+1)} = 1.5\lambda^{(k)}$$

3. Methodology

The primary objective is to train a Neural Network to predict the maximum of the real part of the eigenvalues of the Jacobian matrix of a dynamical system. If predicted correctly to a certain degree of accuracy, this value is extremely useful in knowing the nature of stability of the dynamical system both quantitatively and qualitatively.

This problem is divided into five parts. Part one consists of training Neural Networks to predict dominant eigenvalues of 4x4 matrices. The subsequent parts are devoted to training Neural Networks for increasingly larger square matrices, i.e., 5x5, 6x6, 7x7 and 8x8. Every part is then divided into sub-parts. Every sub-part containing increasingly larger number of hidden layers. The number of neurons in these hidden layers were then varied while keeping the same number of neurons in each layer.

3.1. Creation of Input Values

30,000 random matrices containing normally distributed numbers were created to construct the input and target values for training the Neural Network. The columns of individual matrices were then stacked one over the other creating 30,000 column matrices. These column matrices were then arranged side by side which resulted in a 16x30000 matrix for 4x4, 25x30000 matrix for 5x5 and so on. There are some other rigorous techniques to deal with matrix type inputs to Neural Network tools which may result in carrying the information, that the input is indeed a matrix, in a better way to the trained network. But, those have not been implemented here. The reader can find them in Daniusis and Vaitkus [5].

3.1.1. Normalization

Before feeding it into the training tool, normalization of these input matrices is carried out. Normalization scales the values of the matrix within the range 0 to 1. This step is essential as this improves the accuracy of subsequent numeric computation which results in better output. The min-max normalization technique is used. This is illustrated here

$$X_i' = \left(\frac{X_i - X_{min}}{X_{max} - X_{min}} \right) (X'_{max} - X'_{min}) + X'_{min} \tag{10}$$

where X_i, X_{min} and X_{max} are the actual input data, minimum input data and maximum input data respectively. X'_{min} and X'_{max} are the minimum and maximum target value.

3.2. Creation of Target Values

The target values are constructed by calculating the dominant eigenvalues of all the 30,000 random matrices created in the first step. All of them are then arranged in a column vector and normalization is carried out. Lastly, this is fed into the training tool.

3.3. Performance Analysis

Inside the training tool, the columns of the input matrix are divided into two parts. 70% out of the 30,000 columns are used to actually train the Neural Network while the remaining 30% is used for validation and performance checking.

The main criterion used for checking the performance of the Neural Network was Root Mean Square Error (RMSE). The formula for this is

$$RMSE = \left(\frac{1}{N} \sum_{i=1}^{N} (Y_i' - Y_i)^2 \right)^{1/2} \tag{11}$$

where Y_i in the predicted output, Y_i' is the actual output and N is the number of samples. All the RMSE values for a particular number of hidden layers are plotted against number of neurons in each hidden layer.

4. Observation

The RMSE values are plotted as data points in the graph versus the number of neurons in each layer. Cubic fitting is used to find a trend in the data points as the number of hidden neurons increases.

It can be inferred from figure 1 that for 2 layers the RMSE data points continuously dip till about 30 neurons per layer. After this point, error remains more or less the same. Important thing to note here is that the cubic fitting curve dips at the end because of the fact that it is a cubic curve. The actual data does not dip. So we can consider that 30-35 neurons per layer for a 2 layer neural network gives the optimum results.

Similarly, for 3,4 and 5 layers, data dips till 28 neurons per layer. Notice here that error does not decrease in general as we increase the number of layers. Therefore, 2 layers with 30-35 neurons gives us enough accuracy for a 4x4 matrix. In addition to that, 2 layered Neural Networks can be trained faster that 3 or more layered Neural Networks. From figure 2, data dips till 41 neurons per layer for (a), 42 neurons per layer for (b) and (c), and 36 neurons per layer for (d). But here it can be observed that error is minimum for 3 layers.

From figure 3, optimum number of neurons per layer for 2 layers is 49, for 3 and 4 layers is 41 and for 5 layers is 40. Here also, the error is least for 3 layers.

From figure 4, optimum number of neurons per layer for 2 layers is 52 and for 3 layers is 46. The data for 4 and 5 layers is extremely chaotic in this case even though the cubic fitting decreases. So, no conclusions can be made. It is better to use 2 layers as the error is less and the data is somewhat disciplined.

From figure 5 it can be seen that our model completely collapses. None of them are conclusive figures as the data is very chaotic and widespread.

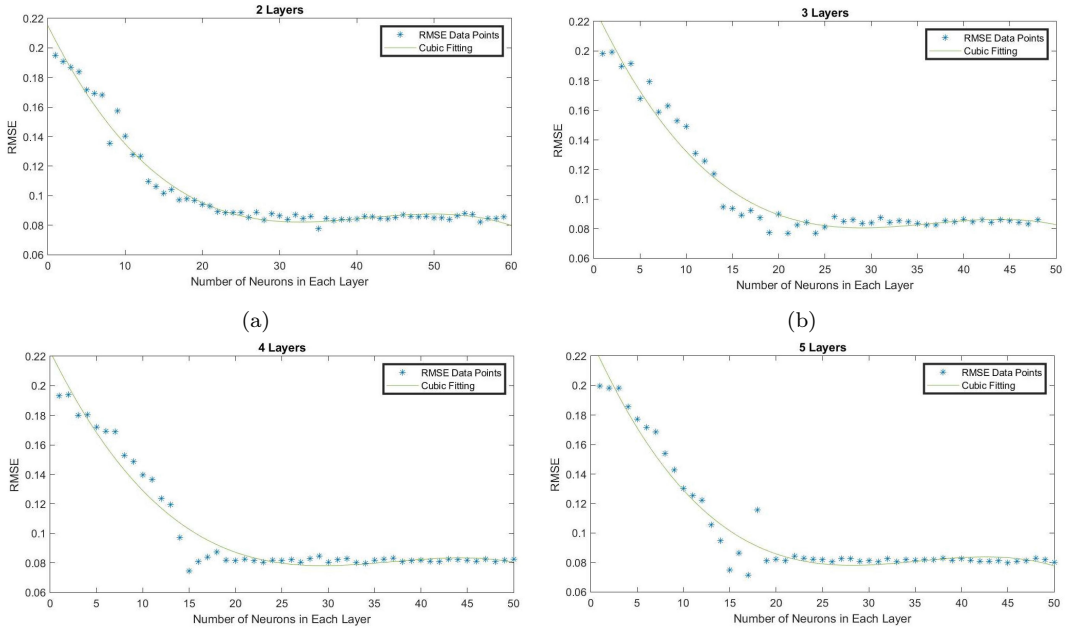

Figure 1: 4x4 Matrices

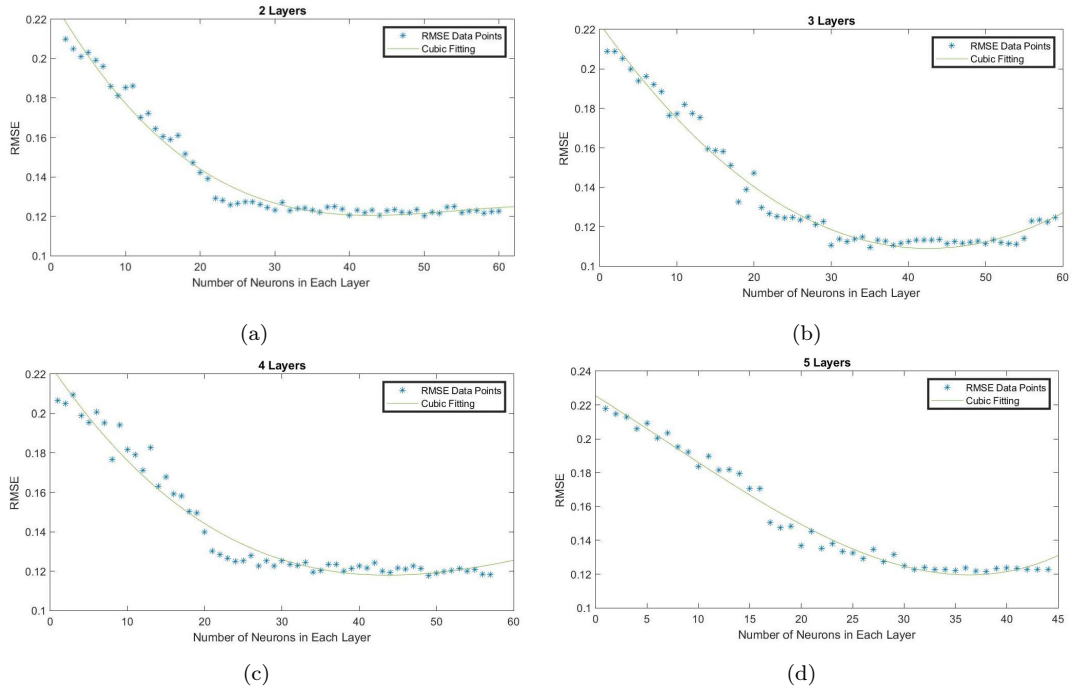

Figure 2: 5x5 Matrices

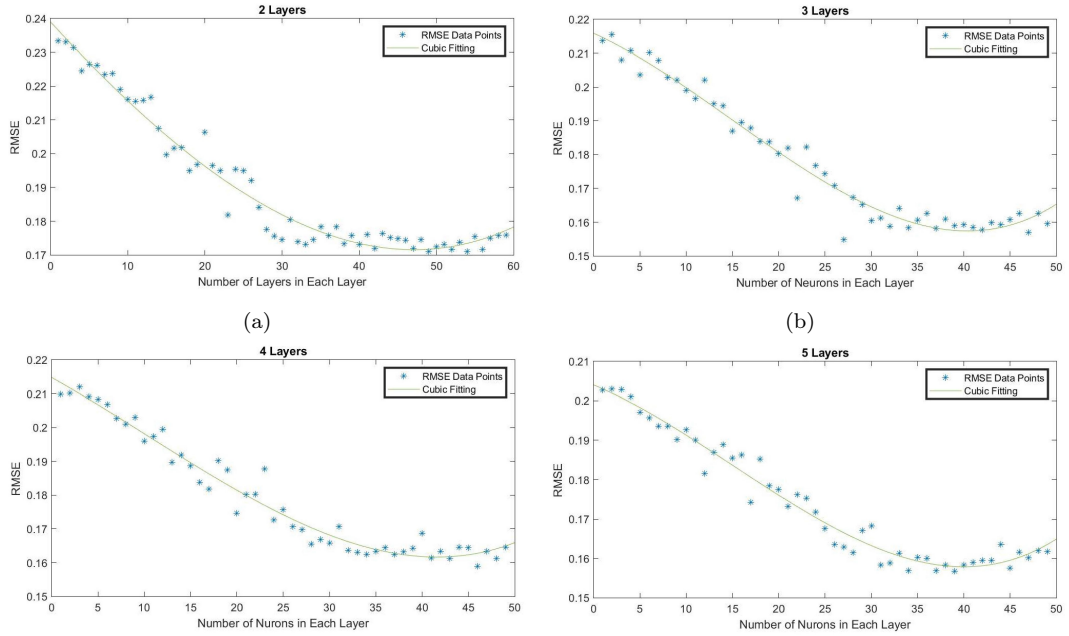

Figure 3: 6x6 Matrices

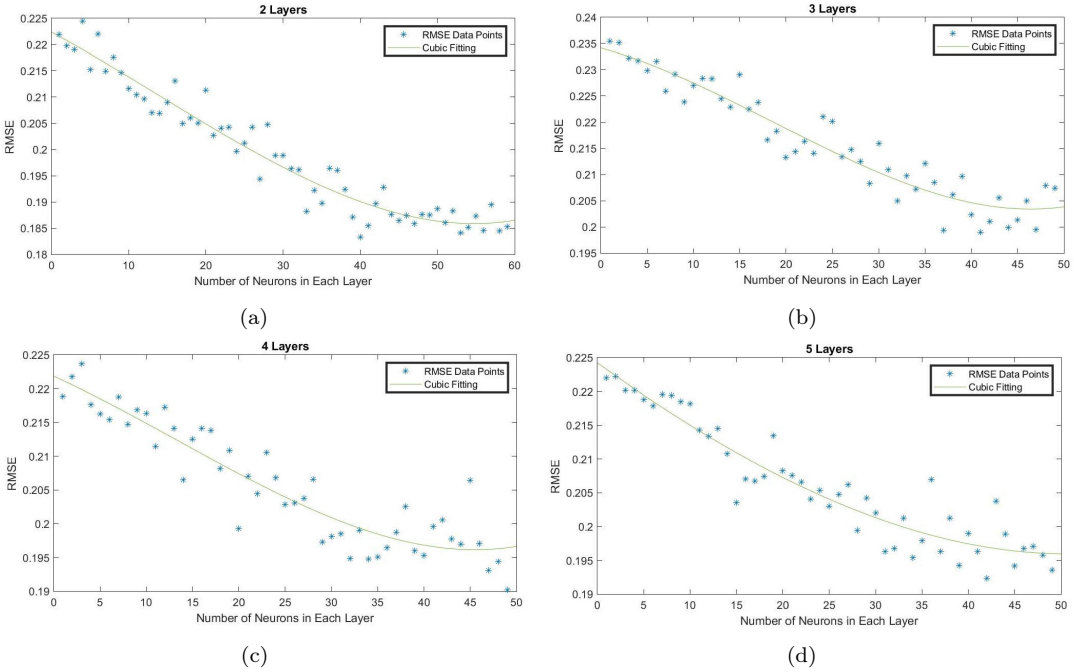

Figure 4: 7x7 Matrices

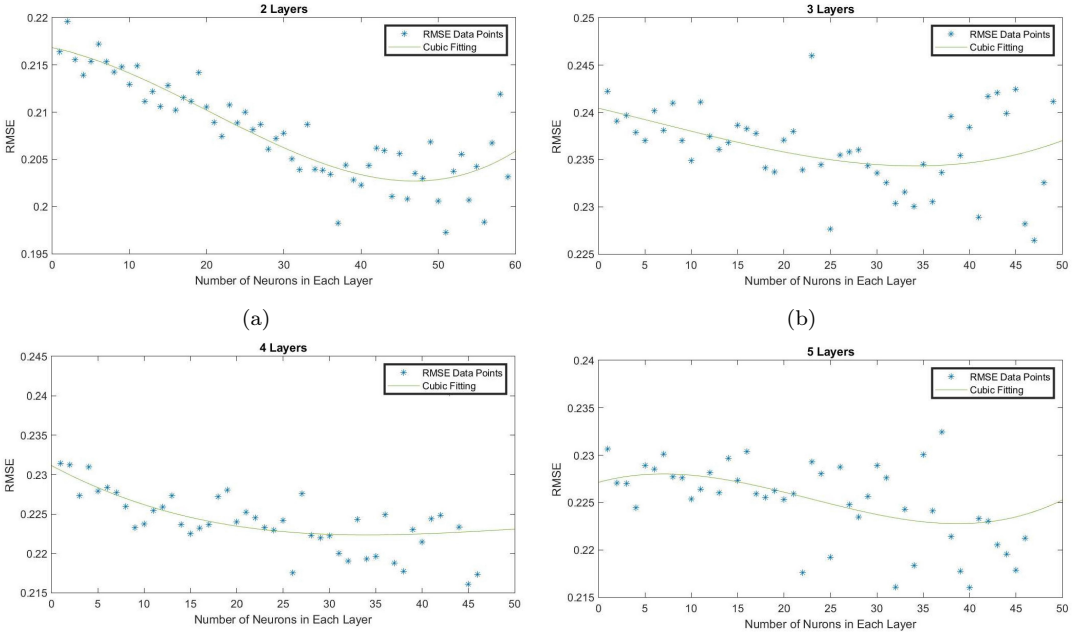

Figure 5: 8x8 Matrices

5. The Classification Problem

The classification problem is solved using a set of 600000 data points for a 5×5 matrix. The neural network architecture is shown in Fig.6. The input layer consists of 25 nodes corresponding to the elements of the matrix. There are 2 hidden layers with 10 nodes each.The output layer consists of 1 node corresponding to 0 or 1 for unstable and stable fixed points respectively. The activation function for the input and hidden layers is the tangent sigmoid function and that of the output layer is a purely linear function. The performance of the network is evaluated using the cross-entropy function which is a function of the weights and biases. The scaled conjugate gradient training algorithm is used to train the network.

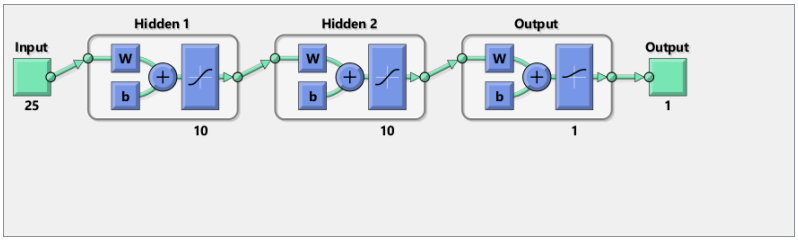

Figure 6: Neural Network Architecture

The network performance is shown in Fig.7. It can be seen that the error function has converged below 0.1 for all three sets of data. This suggests that the network has been

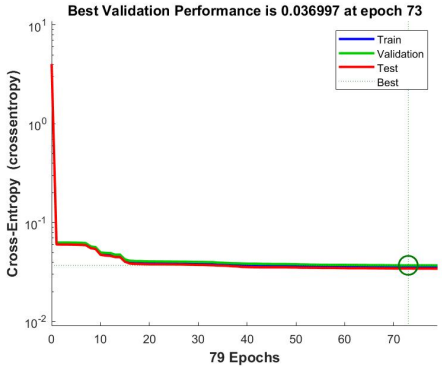

Figure 7: Performance Plot

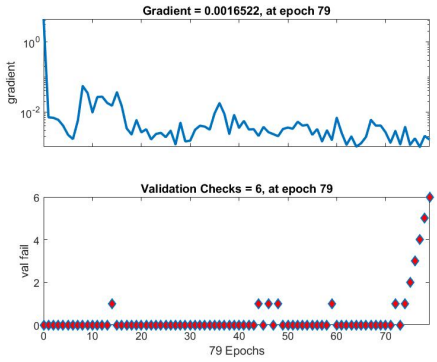

Figure 8: Gradient Plot

generalised and the weights obtained after training using the sample data set will be equally applicable to any new data. Thus there is no over-fitting.

The gradient plot is shown in Fig.8. At the point where the algorithm stops, the value of the gradient is 0.00165. This signifies that the network has stopped very close to the actual target. Fig.8 also shows that the algorithm has stopped because the validation performance has failed to increase for 6 consecutive iterations.

The error histogram is shown in Fig.9. The error ranges from -0.6801 to 0.9568. This range has been subdivided into 20 bins. Thus the width of each bin is 0.0818. It can be seen that the majority of the samples have an error lying in the range of -0.0727 to 0.0909. This implies that the error in the network is very less.

The confusion matrix for all three sets of data is shown in Fig.10. For the training set, the network classifies the output correctly for 415414 samples with an accuracy of 98.9%. Similar accuracy is obtained in case of the validation and test set. Thus the network has generalised well.

The receiver operating characteristic curve is shown in Fig.11. The ROC is a metric for evaluating the performance of the classifier network. For each class of the classifier, the ROC applies a threshold value across the interval $[0, 1]$ to the outputs. For each threshold, the true positive and the false positive ratio are calculated. More the curve hugs the axes, better

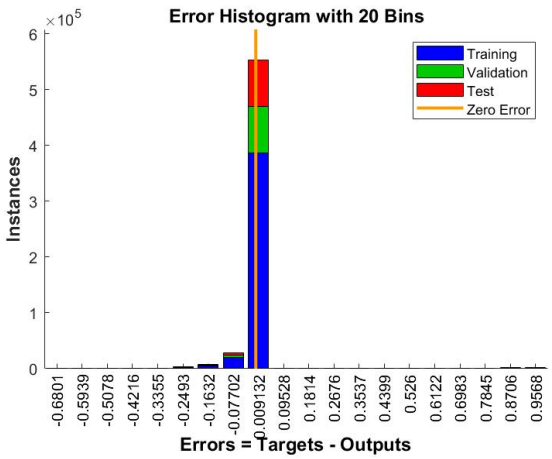

Figure 9: Error Histogram

is the classification. It can be seen that there is a slight deviation from the axes,although within acceptable range. This can be improved by tweaking the number of layers and nodes. Thus the neural network does a reasonable job.

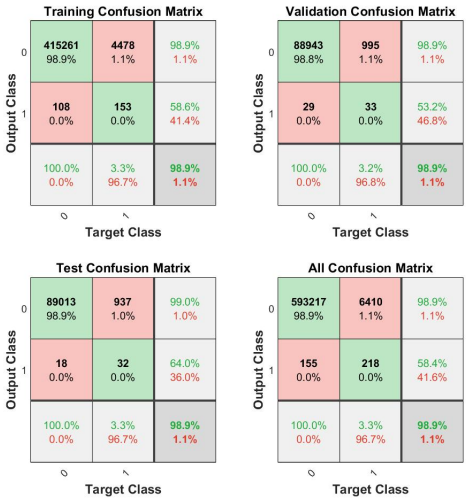

Figure 10: Confusion Matrix

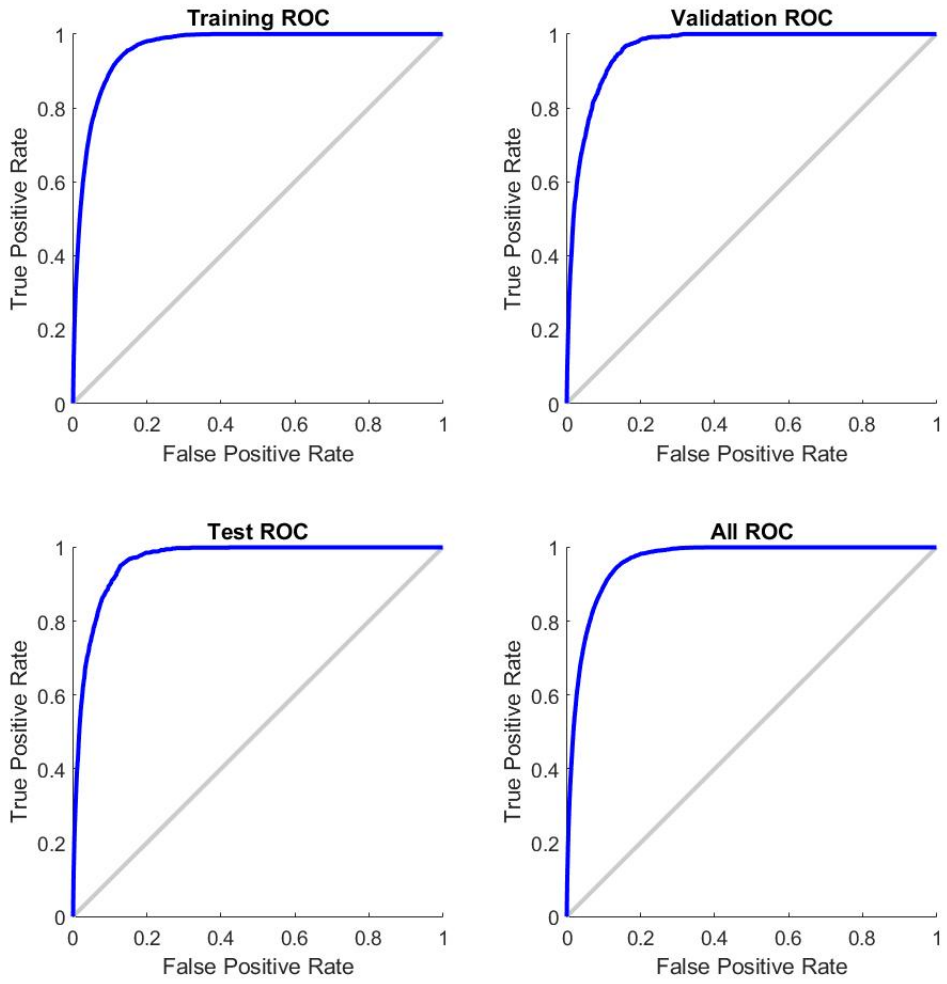

Figure 11: Receiver Operating Characteristic

6. Conclusion

In this paper, numerous Neural Networks are trained using the Levenberg-Marquardt Backpropagation Algorithm to find out the optimum number of hidden layers and hidden neurons to calculate the dominant eigenvalue of the Jacobian of a dynamical system. 30,000 matrices along with their dominant eigenvalues are used, out of which 70% are used to train the Neural Networks and 30% are used to validate the results of the trained networks. RMSE is used to analyse the performance of the Neural Network and identify which combination of hidden layers and hidden neurons are best for a particular matrix size. This is done by plotting the RMSE values against the number of hidden neurons for a particular number of hidden layers and a particular matrix size. The following conclusions are made:

1. For 4x4 matrices, 2 layers with 30-35 neurons per layer gives the optimum results.
2. For 5x5 matrices, 3 layers with 42 neurons per layer gives the optimum results.
3. For 6x6 matrices, 3 layers with 41 neurons per layer gives the optimum results.
4. For 7x7 matrices, 2 layers with 52 neurons per layer gives the optimum results.
5. For 8x8 matrices, no conclusions could be made as the data was too widespread and chaotic.

At the end, a simple classification problem is solved to find out whether a system is stable or unstable based on its Jacobian matrix.

References

[1] S. N. D. S. N. Sivanandam, S. Sumathi, Introduction to Neural Networks Using Matlab 6.0, Tata McGraw Hill, 2008.

[2] L. X. K. Jinchuan, Empirical analysis of optimal hidden neurons in neural network modeling for stock prediction, Proceedings of the Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008.

[3] S. N. D. K. Gnana Sheela, Review on Methods to Fix Number of Hidden Neurons in Neural Networks, Hindawi Publishing Corporation, 2013.

[4] S. H. Strogatz, Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering, Westview Press, 2000.

[5] P. V. Povilas Daniusis, Neural Networks with Matrix Inputs, INFORMATICA, 2008.