Neural Network based Modeling and Simulation for the Optimization of Safety Logic

by

Neeraj Agarwal

M.S., Environmental Engineering (2001)
University of Connecticut

B.Tech., Civil Engineering (1999)
Indian Institute of Technology, Kanpur

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Civil and Environmental Engineering

at the
Massachusetts Institute of Technology, June 2002

Signature of Author...
Department of Civil and Environmental Engineering
May 10, 2002

Certified by..................................
Dr. Amar Gupta
Co-Director, Productivity From Information Technology (PROFIT) Initiative
Thesis Supervisor

Certified by..
Dr. Ruaidhri M. O' Connor
Assistant Professor
Thesis Reader

Accepted by.. ....
Oral Buyukozturk
Chairman, Department Committee on Graduate Studies

Neural Network based Modeling and Simulation for the Optimization of Safety Logic

by

Neeraj Agarwal

Submitted to the Department of Civil and Environmental Engineering
On May 13, 2002 in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Civil and Environmental Engineering

Abstract

This thesis focuses on a knowledge based approach to reduce the incursion situation in an airport. In an effort to reduce the number of *runway incursions* that occur on an airport surface and the surrounding areas, the Federal Aviation Administration (FAA) is actively investigating automated systems to provide the air traffic controllers with early warnings about impending incidents. The Airport Movement Area Safety System (AMASS) is the first ground surveillance safety system that is being deployed at the 34 busiest airports in the United States. The optimization of the parameters, which control the safety logic algorithms that generate warnings of possible incursions, is a labor and time intensive endeavor. Currently AMASS uses over 200 safety parameter values for all airports independent of the organization or traffic flow of a specific airport.

The thesis focuses on how neural networks are used to organize alert types in a semi-autonomous process that will provide alerts to air traffic controllers as early as expediently. The methodology developed provides a generalized system that can be optimized independently for each airport and still generate warnings of possible incursions. Several networks exhibit good learning shown using different statistical parameters and could eventually be integrated with AMASS. Based on the current operations of neural networks it is projected that neural networks will use the AMASS log data and data filter software to extract necessary information. In the future the technologies discovered in these neural network models can help generate new versions of safety logic software for the advanced surveillance systems.

Thesis Supervisor: Dr. Amar Gupta
Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative

Acknowledgments

First of all, I would like to thank God for giving me strength and perseverance needed to complete this thesis and program. I would also like to thank my parents and my family for standing by me during these times.

In addition, I would extend my thanks to my thesis advisor for his high standards and continued support. His guidance and support helped keep me pointed in the right direction and always moving forward.

I would also like to thank my research project group for their input and valuable discussions.

Lastly, my thanks go out to my new friends in the course 1 M.Eng. program. With these friends, I knew I could accomplish anything. The friends I've made will stay with me always, reminding me that where ever I go.

Table of Contents

List of Figures

1. Introduction

The advancements in the information systems have given rise to a new trend to identify and generate meaningful information from the historical data. Due to the incorporation of this information, it has become a key to gaining an advantage over the existing methods. The value of data mining is to proactively seek out or predict the trends and thus to provide research or industry a better understanding.

Over the past few years, there had been an increasing interest in using Neural Networks to mine the large historical data and extract meaningful patterns and relationships. Neural Network based data mining has been applied to multitude of fields like inventory reduction [1] [2] [3], finance [4], weather prediction [5], electronic commerce [6], medicine [7], image recognition etc. Neural Networks provide a good tool in prediction of complex processes.

This thesis focuses on the use of Neural Network based data mining in reducing the runway incursion situations on an airport. Federal Aviation Administration (FAA) is actively investigating automated early warning systems to reduce the number of runway incursions that occur on an airport surface and the surrounding areas. These systems are designed to provide the air traffic controllers, who are responsible for traffic management on the runways and taxiways, with early warnings about impending incidents. With this additional information, it is hoped that an air traffic controller can intervene in time to avoid the actual incursion.

One such system is the Airport Movement Area Safety System (AMASS), developed by the Northrop Grumman Norden Systems (NGNS) under contract with the FAA. AMASS is the first ground surveillance safety system that is being deployed at the 34 busiest airports.

Optimization of the parameters that control the safety logic algorithms that generate warnings of possible incursions is labor and time intensive. AMASS has approximately 400 different safety cells that require a distance parameter, a time parameter or a combination of both [8]. Of these 200 safety cells, 71 cells are available to warn air traffic controllers of a possible incursion. The average time required to optimize the entire AMASS system including different components like Terminal Automation Interface Unit (TAIU), Airport Safety Detection Equipment (ASDE), etc. is 6 months. The safety parameters that have been optimized are common for all 34 AMASS systems currently deployed. Minor variations to these safety parameters are made based on airport operating conditions and procedures. The FAA has focused approximately two years on the optimization of these 71 safety parameters; the remaining safety parameters values have yet to be determined.

As successful as the AMASS program has been over the last two years, we propose that some form of an autonomous process is needed to fully exploit the AMASS system capability to maximize its effectiveness. This thesis focuses on the initial development and testing of Neural Networks (look into Appendix 1-3 for more details) on flight data. Neural networks can be used as a potential solution to organize alert types in an autonomous process that will provide air traffic controllers earlier warning times while at the same time, introduce the availability of a larger set of alerts than currently available in AMASS. The proposed process can be integrated with the existing AMASS system. The objective of the research is to provide a safety system that is optimized according to air traffic controller actions for each and every individual airport.

The thesis is organized as follows. Chapter 2 will give details about the AMASS and its operation, Chapter 3 will detail the dataset used for the study, Chapter 4 outlines the Neural Network (NN) based approach, Chapter 5 details the statistical parameters used to check the robustness of the NN, then analysis and results are followed by the conclusion and future studies.

2. AMASS Operations

The structure of AMASS and its associated safety logic are shown in Figure 1. The upper left corner of Figure 1 depicts the Airport Surface Detection Equipment radar (ASDE-3) that feeds into the AMASS. The lower left corner shows the Airport Surface Radar (ASR-9) and the Automated Terminal Radar System (ARTS, not depicted) that are used to feed the Terminal Automation Interface Unit (TAIU) providing AMASS with airborne target information. In short, the AMASS cabinet processes this information through safety algorithms and outputs tracked targets onto the existing ASDE-3 Operator Display Unit (ODU) seen in the upper right corner of Figure 1. The ODU is located in the airport tower cab for controller use. The ODU displays target information and text alerts. Aural alerts are broadcast over a set of speakers which are also located in the tower cab.

Aircraft velocity, acceleration and heading parameters are derived parameters that can be defined in terms of raw data collected by radar. No unique aircraft information is available to the controller except for ARTS arrival tag information. In other words, AMASS does not have or use aircraft performance characteristics. For instance, the controller does not know the weight or the type of the aircraft.

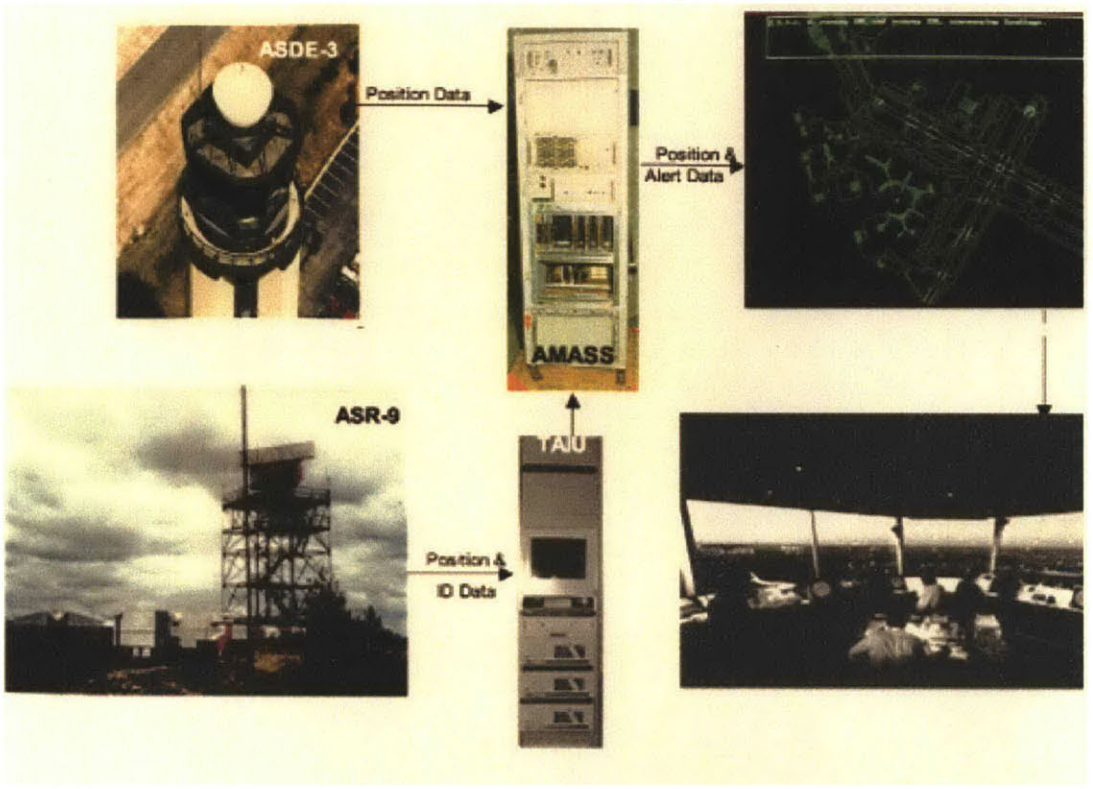


Figure 1. Integrated AMASS System

3. Preliminary Data Set

For the purpose of this study, we use data for aircrafts using runways at one of the 34 busiest airports. The airport was selected where there was frequent use of the same runway for both departures and arrivals. In order for neural networks to operate effectively, a data set needs to be provided to the network so that it can be trained on the normal operation of the traffic. It is desirable, but not necessary, for the data set to provide examples of incursion situations. The reasoning for this hypothesis is provided in the subsequent sections of this thesis. We noted that only one "AMASS" incursion situation was recorded in the analysis data set used for the initial study.

Figure 2 highlights the one "AMASS" alert situation recorded in the analysis data. This figure schematically shows one aircraft departing on a runway and another aircraft landing on the same runway. The rest of the data recorded are similar instances of this one scenario except that they do not generate an alert.

The above scenario may give the impression that one is looking for events that are happening on a single runway at an airport. The real environment being analyzed is much more complex as one is looking simultaneously at multiple runway, and also areas in proximity to such runways. This includes, for example, taxiways leading up to different runways.

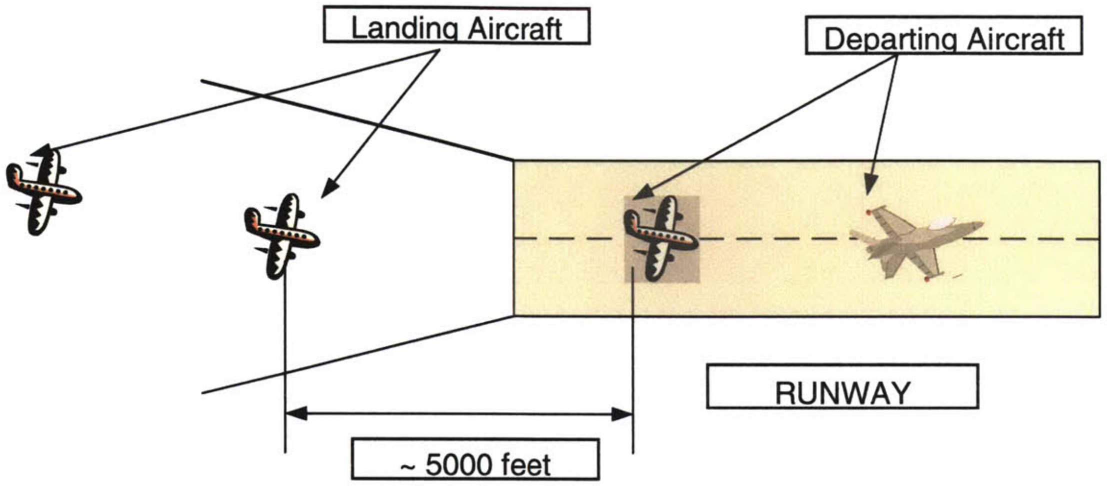


Figure 2. Arriving aircraft is encroaching too close to the departure aircraft that is physically located on the runway. This scenario results in an AMASS "Go-Around" alert.

In addition, one needs to distinguish between and analyze several different scenarios [8], such as:

(i) Lander behind Lander Scenario (LL): In this case, two planes are coming in to land on the same runway, one behind the other. If the separation distance between these two aircrafts is predicted to fall below a nominal threshold, one could try to delay the landing of the second aircraft. However, this may not always be possible, especially if there is a third aircraft coming in to land after the first two aircrafts;

(ii) Lander behind Departure Scenario (LD): In this scenario, one plane is landing and another is taking off, both from the same runway. Again, there are intricate interdependencies on how to maintain high operational throughput from that runway while still maintaining adequate separation distances and safety margins;

(iii) Intersecting Runways: In this scenario, there are two (or more) runways that physically intersect with each other. Analysis of the point of intersection requires detailed data from each of the relevant runways. Accordingly, it becomes hard to model such runways using conventional techniques;

(iv) LASHO: Long Arrival Short Hold: This is a special case of intersecting runways where the runway is long enough to allow the landing aircraft to come to a complete halt just before the point of intersection. It is like stopping at a "STOP" sign on a busy intersection, and then proceeding further after one has verified that there is no traffic on the perpendicular road. Situations like these cannot be analyzed easily with the current version of AMASS software;

(v) Taxiing Aircraft(s) Approaching Active Runways: In this case the objective is to maximize the number of takeoffs and landings while maintaining order and safety in the relevant parts of the airport.

One should note that there is wide variety in the way airports and constituent runways are laid out in the U.S. In many cases, it becomes necessary to incorporate the characteristics of the specific airport and runway into the model. However, one wishes to make the models as generic as possible, with respect to broad applicability. This makes the problem of predicting the alert situation more challenging.

4. Neural Networks Approach

Neural Networks (NN) are complex mathematical models that are nonlinear and input-output mapping adaptive. NN offer uniformity of analysis and design that make them easy and efficient to use for problems such as pattern recognition, optimization, system modeling, and data compression, among other applications (Haykin, 1999). In general NN models are composed of individual processing units called nodes. The nodes are interconnected by links or weights. A node connection arrangement may range from full to sparse or locally connected. A NN generally contains multiple layers of nodes interconnected with other nodes of same or different layers. These layers could be an input layer, hidden layer(s), or an output layer. The inputs to each layer and the weights associated with the links are processed by a weighted summation function to produce a sum, which is subsequently passed to an activation function and the result from there is the output for that node. This is illustrated in Figure 3. For more details about neural networks, the reader can refer to Bishop (1995) or Haykin (1999).

Any NN has to be calibrated, or trained, before its application. The training can be of three types - supervised, unsupervised, or reinforcement type. Training is basically a procedure of adjusting the NN weights to best represent the problem solution. Supervised learning may conceptually be thought as a teacher having knowledge of the environment, and that knowledge is gained by a set of input-output pre-acquired data. When the NN and the teacher are exposed to a training vector drawn from that environment, the teacher by virtue of its prior knowledge is able to provide NN with a desired response for that vector. By desired response is meant that the set of weights are altered, and this adjustment is carried out iteratively until the NN emulates the teacher; the emulation is presumed to be optimum in some statistical sense using objective criteria. This way the whole training data set is ingested and NN trained which is subsequently ready to deal with the environment by itself. In this research the backpropagation training scheme is employed (see Appendix 2 for complete description). In the unsupervised scheme there is no external teacher to check on the learning. Rather, provision is made for a task-independent measure of the quality of measurement that NN has to learn, and the free parameters of the network are optimized with respect to that measure. After the NN is tuned to the statistical regularities of the input data, it is ready for application. More detailed information on NN calibration procedures can be found in Haykin (1999).

In general, the advantageous characteristics of the NN approach can be summarized as follows: (1) the addressed problem or task does not have to be clearly defined and it does not require prior knowledge of the problem; (2) one may not explicitly recognize all the existing complex relationships as during the training NN is able to form these relationships (non-parametric method); (3) NN almost always converges to an optimal (or sub-optimal) solution and need not run to any pre-specified condition; (4) no prior solution structure is assumed or enforced on the NN development; and (5) large amount of data can be handled quite efficiently. These advantageous characteristics of the NN fit very well with the particular requirements of prediction of separation distances, as we are dealing with large dataset and there is no well-defined relationship between data and the separation distances in future.

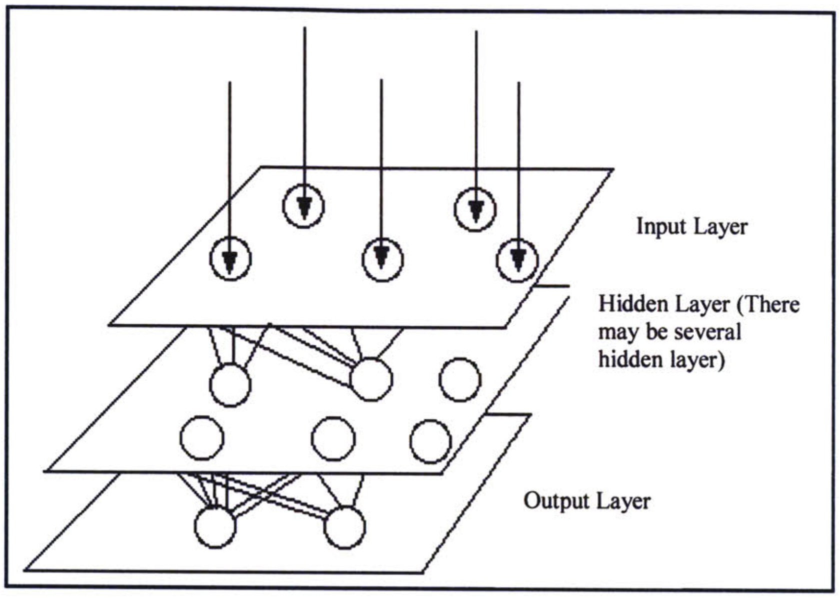


Figure 3. Iterative process of designing a Neural Network

To determine the number of hidden layers in the network, one needs to often make this crucial decision by trial and error. In order to get optimal results, if one increases the number of hidden layers, one gets into a state of "over fit" in which the network has problems of generalization. The training set of data will be memorized, making the network ineffective when dealing with new data sets.

There are several different types of NNs. The more popular techniques employ the Multilayer perceptron, which is gradually trained with the backpropagation of error algorithm; Recurrent Neural Network, which is trained with recurrent algorithm (i.e. implement feedback) depending on how data are processed through the network.

The data provided by the AMASS included absolute positions of each plane (x, y and z co-ordinates), heading and velocities. Accordingly, a data extraction tool was developed to extract specific features of interest.

The output of the data extraction tool was processed on a sequential basis as depicted above. In the training phase of the NN, since NN is untrained initial errors are high and correlation is low. If the NN is training well the errors come down and finally stabilize at some low value. The correlation also stabilizes at value close to 1 that shows that NN has learned the outputs. On the other hand, if the training results show bumps and valleys it means NN is unable to learn. Figure 4 shows the overall approach for the study.

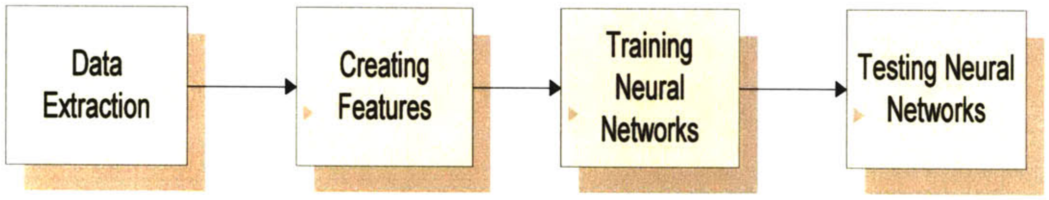


Figure 4. Overall approach for use of Neural Networks in optimization of safety logic

The initial endeavor was to utilize a breadth first approach that involved applying several different NN techniques to perform preliminary analysis of the underlying dataset. In conjunction with the breadth first approach, several models for predicting alert conditions were tried on two distinct cases: landing – landing (L-L: Pair of successive flights landing on the runway) and landing – departure (L-D: Pair of successive flights landing on and departing from the same runway). Detailed analysis of these models is presented in the next section. It is important to note that the models varied significantly based on the type of *Case* (L-L or L-D), *Inputs* (absolute values such as x, y coordinates and derived parameters such as Separation distance, Separation velocity and Separation time), *Outputs* (Separation distance in future), *Hidden layers* (1 through n) and the particular *NN technique*.

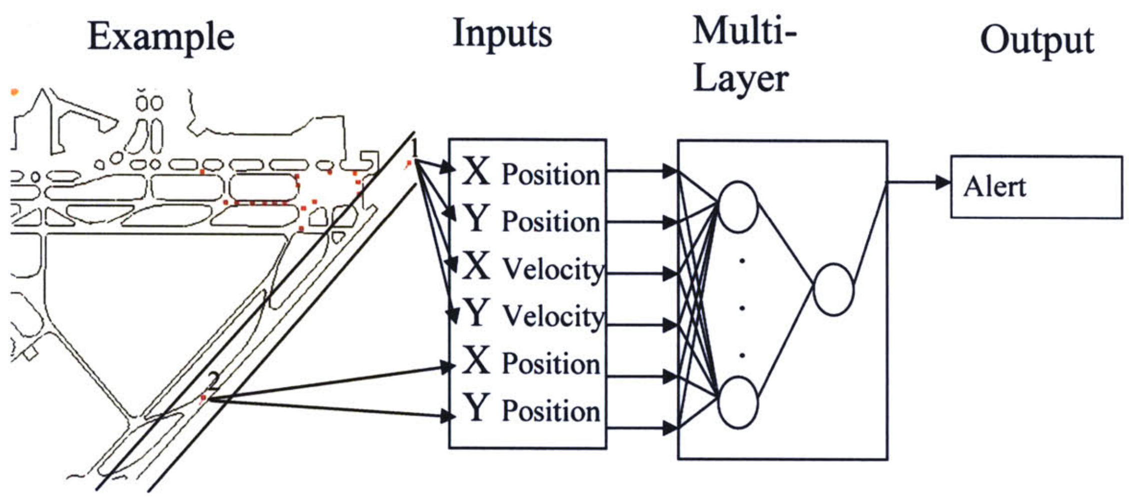


Figure 5. NN model with binary output (Alarm vs. Non-Alarm)

Initially the output of the models was binary: Alert versus Non-Alert conditions. The models showed very good training results i.e. the average errors very nominal and the model showed high correlation with the actual values. The very low number of Alert cases (2 records) made it difficult to accurately determine if the model is indeed learning the true characteristics for an Alert condition. For example, if the model predicted Non-Alarm outputs for all the cases including the two alert cases in the underlying dataset, the performance metrics would show high accuracy while the results would hold little relevance. This is because even if we have a trivial model which predicts every case as Non-Alert, the overall rate of errors computed to be low due to few cases of Alerts.

To overcome the above problem, the output of the network was changed to calculate separation distances at different times in future. In such a case, the low number of alarm cases does not present a challenge to the training phase of the model. The output estimates of the separation distance can be utilized to develop a post processor that could trigger an alarm or near alarm situation. The overall system suggested would then take parameters like absolute positions and velocities to accurately predict the alarm condition sooner than what is possible with the current version of AMASS.

5. Statistical Parameters

In a range of hundreds of neural networks, it is extremely difficult to find which types and classes of neural networks are suitable for predicting alarm conditions. To objectively evaluate the performance of different types of networks, three different statistical indicators were used. These indicators are: *Pearson Correlation Coefficient (PC), Normalized Mean Square Error (NMSE),* and *Average Error (AE)* [2].

Each of the performance indicators represent a precise method of measuring how well a simulation performed. No one indicators can tell how well one simulation fared against others; instead all of these three numbers should generally be considered together. The *Pearson Correlation Coefficient (Equation 1)* shows how well trends, i.e., bumps and valleys, were picked up. The *Pearson Correlation* is a number ranging between -1 and 1. If the simulation predicts bumps and valleys perfectly, then the corresponding *Pearson Correlation* would be 1.

$$P.Correlation = \frac{\sum_{1}^{N} X_i Y_i - \sum_{1}^{N} X_i \sum_{1}^{N} Y_i}{\sqrt{N\sum_{1}^{N} X_i^2 - \left(\sum_{1}^{N} X_i\right)^2}\sqrt{N\sum_{1}^{N} Y_i^2 - \left(\sum_{1}^{N} Y_i\right)^2}}$$

Equation 1: P.Correlation, where X and Y are different NN input features and N is the number of points in the data set.

The *Normalized Mean Square Error (NMSE) (Equation 2)* is a method to compare the mean of a series against the predicted values. If the NMSE is greater than 1, then the predictions are worse than the series mean. If the NMSE is less than 1, then the forecasts are better than the series mean. The NMSE is widely used measure in academic journals to evaluate how well a Neural Network has performs.

$$NMSE = \frac{\sum \frac{(X_{\Pr edicted} - X_{Ouput})^2}{(X_{Output} - \overline{X})^2}}{N}$$

Equation 2: NMSE, where X is NN output and N is the number of points in the data set.

The *Average Error (AE) (Equation 3)* is another way to compare the forecasts with the actual values. It indicates, as a percentage value, the average difference between the predicted and actual value. For instance, an AE of 0.30 means that the neural network will provide predictions which on the average are within plus or minus 30% of the actual values. Unfortunately the AE tells one nothing of how well the computer predicts trends and for that one must use *Pearson Correlation* factor.

$$AE = \frac{\sum (X_{Output} - X_{\Pr edicted})}{N}$$

Equation 3: AE, where X is NN output and N is the number of points in the data set.

6. Analysis and Results

6.1 Single Layer Perceptron (SLP)

SLP is the simplest form of a neural network. It is used for classification of patterns that are linearly separable (i.e., patterns that lie on the opposite sides of a hyper plane). For instance in two dimensions it is analogous to linear regression. An SLP consists of a single neuron or a node with adjustable weights and bias. For details refer to Appendix 1. This can only be used to distinguish between two classes: in our case, Alert and Non-Alert situations.

This model is intended to test the viability of predicting alert conditions from the separation distance, the separation velocity and the separation time information, relative to many sets, each relating to a pair of planes. These features were calculated using the raw position and the velocity data. The NN based model was built using the derived parameters from the raw data set; the hypothesis was that such a model could be employed for multiple runways with no further training needed. The input features (SD, SV and ST) are the separation distance, the separation velocity and the separation time for a pair of planes on the runway. The output for the model is a single value representing the likelihood of an alert occurring between the pair of planes. A zero represents an Alert condition and a one represents a Non-Alert condition.

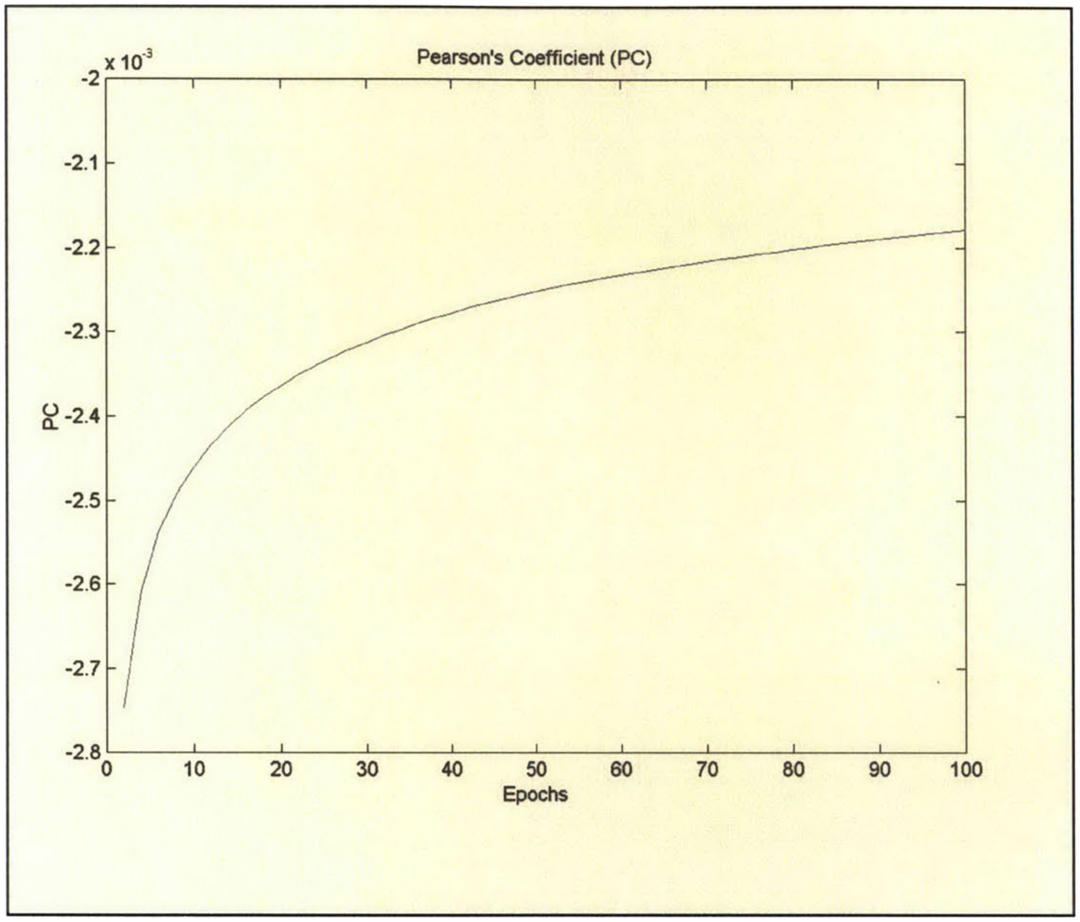


Figure 6. Pearson's Correlation for a single runway

For the training phase of the model, the alert output corresponded to the pair of planes that caused an AMASS alert in the log. This alert condition was associated with the pair of planes for all instances of alerts that were utilized in the model.

The above model did not exhibit characteristics of good learning as the Pearson's Correlation (Figure 6) was almost zero although the AE of the model with respect to the desired output drops to a low value. These factors indicate that the model does not show learning capabilities. As such, one focused more on MLPs and RNNs, which are described next.

6.2 Multiple Layer Perceptron (MLP)

An MLP can be thought of as a cluster of many SLPs. Unlike SLPs, an MLP can emulate any real continuous function. MLPs consist of a set of source nodes that constitute the input layer, one or more hidden layers of computation nodes, and an output layer. The input signal propagates through the network in a forward direction, on a layer-by-layer basis.

MLP has two distinguishing characteristics from SLPs. First, all its nodes or neurons utilize a nonlinear activation function. The most commonly used activation function is the sigmoid function (Equation 4). The presence of non-linearity enables MLPs to map the input to the output using any real function. Second, the hidden layer or layers enable the network to learn complex tasks. For more details refer to Appendix 2.

$$g_j(z) = \frac{1}{1+e^{-bz_j}}$$

Equation 4. The sigmoid function used as activation function.

The Multiple Layer Perceptron model used derived parameters. The input features were the separation distance; the separation velocity and the separation time for a pair of flights at time t_0 and time t_n. The output for the model was separation distances into the future between the pair of flights.

MLP based models were deployed for the following cases –

- Single Runways
- Multiple Runways

Single Runways:

This case is used for airports where we consider the air traffic only for individual runways and the taxiways, which lead to them. Figures 7, Figure 8 and Figure 9 show the different error statistics for the MLP model. Figure 7 represents the Average Errors in separation distances in feet versus the epochs. We see that the model is learning well as the errors are stabilizing to a low value. Similar conclusions can be drawn from Figure 8 and Figure 9.

Multiple Runways:

This case is used for airports where there are runways, which intersect each other. In this case we have to consider the air traffic across all the intersecting runways as well as the taxiways, which lead to them. Figures 10 - 12 show the different error statistics for the MLP model. Figure 10 represents the Average Errors in separation distances in feet versus the epochs. We see that the model is learning well as the errors are stabilizing to a low value. Similar conclusions can be drawn from Figure 11 and Figure 12.

6.3 Recurrent Neural Network

RNNs are connected networks from output nodes to hidden layer and/or input layer nodes, and they also allow interconnectivity between nodes of the same layer, specifically between the nodes of hidden layers.

Various training algorithms for RNN have been proposed by several researchers such as Jordan (1986) [14]; Rumelhart, Hinton, and Williams (1986) [15]; Williams and Zipser (1989) [16]; and Elman (1990) [17]. Elman training algorithm is being used in this research.

RNN is deployed for the cases of single runways and the results are shown in figures 13 through 15. Figure 13 represents the Average Errors in separation distances in feet versus the epochs. We see that the model is learning well as the errors are stabilizing to a low value. Similar conclusions can be drawn from Figure 14 and Figure 15.

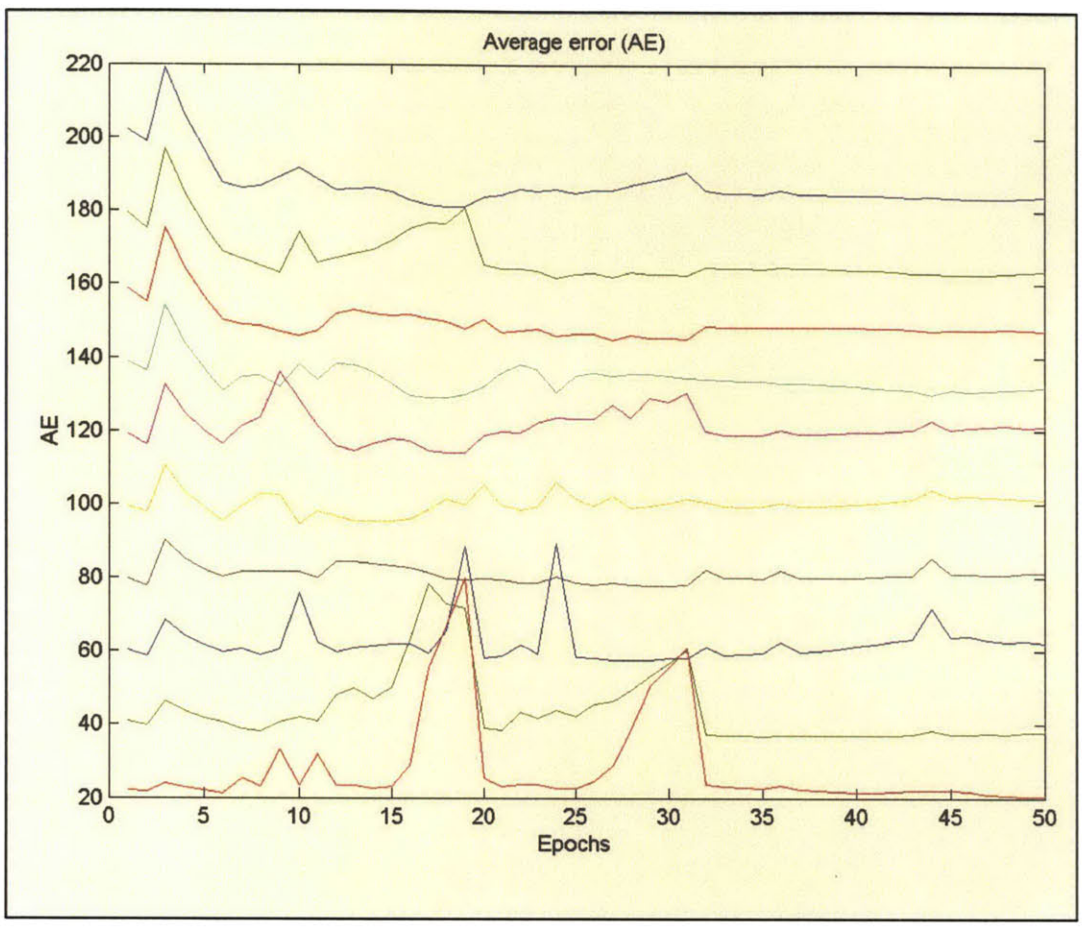


Figure 7. Average Error for a single runway case. Each line in the graph represents the output for a second in the future with the topmost line being 10 seconds through 1 second (the bottom most line) into the future.

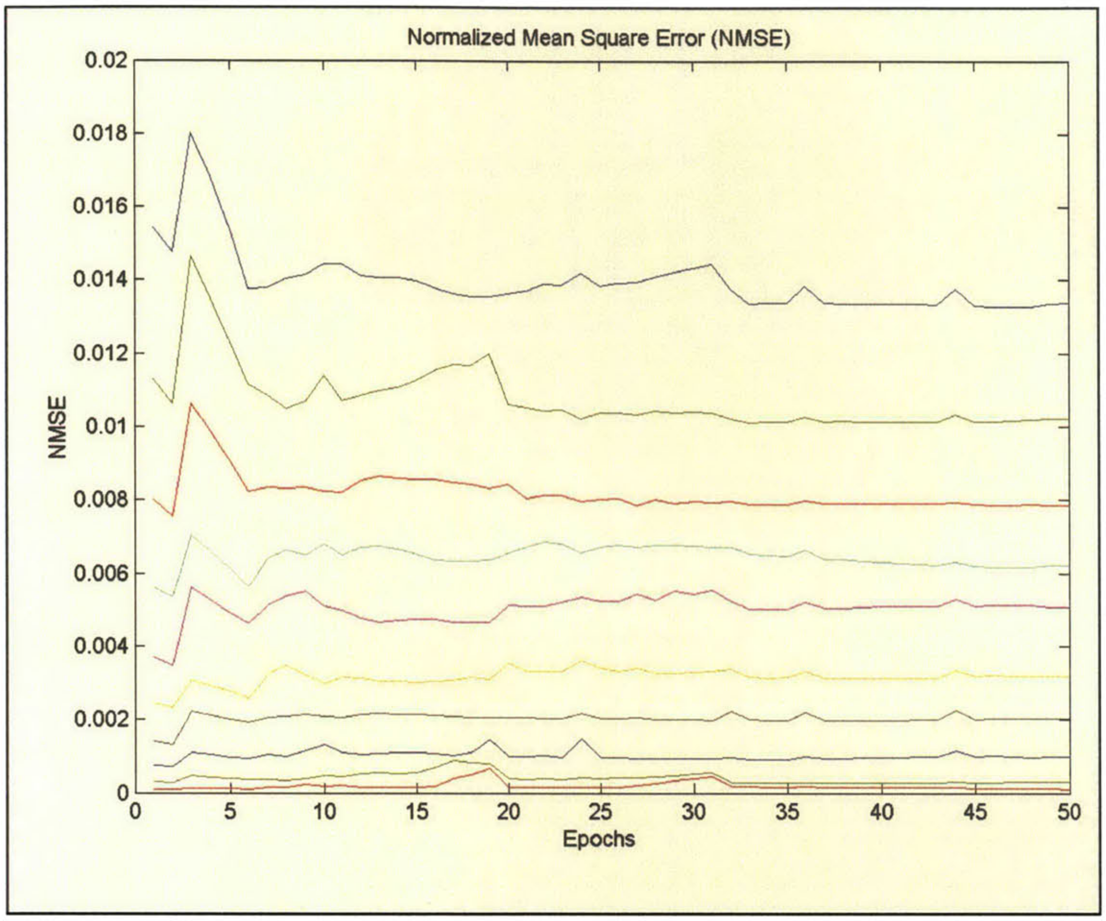


Figure 8. Normalized Mean Square Error for a single runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

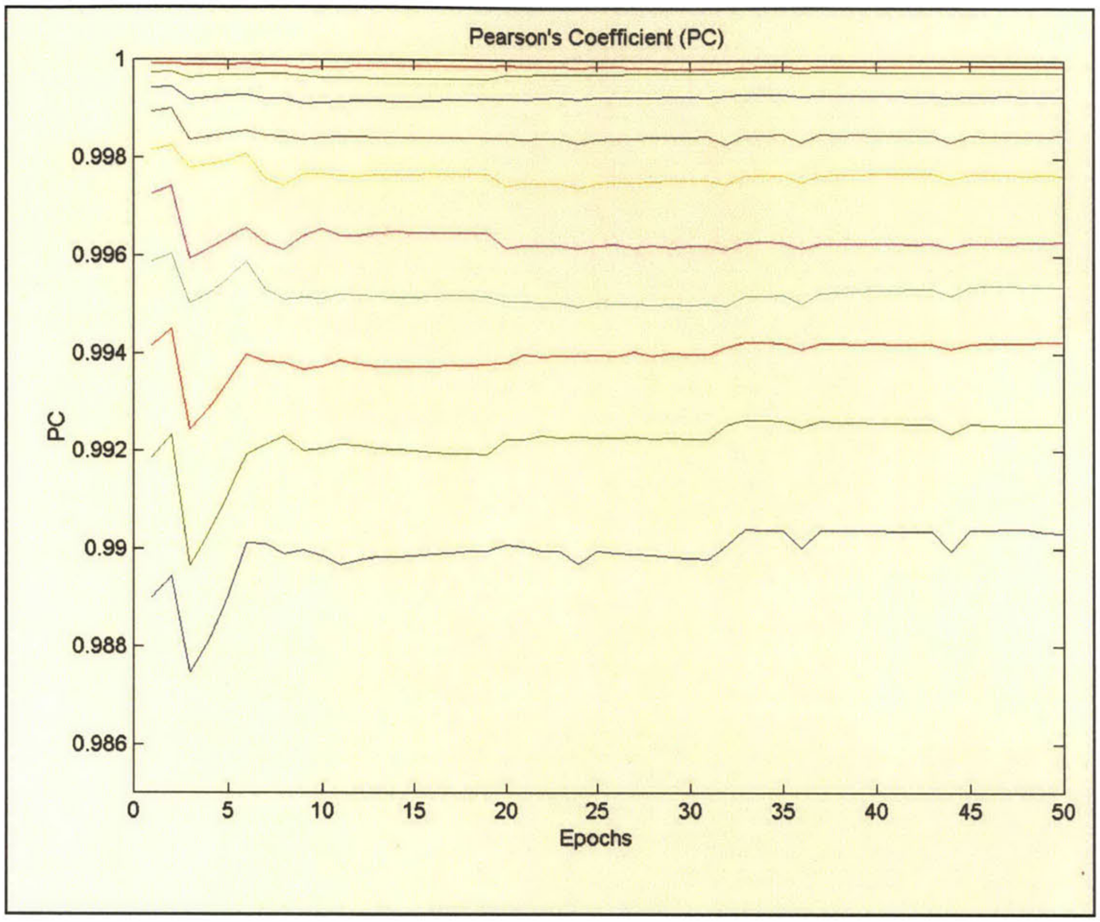


Figure 9. Pearson's Coefficient for a single runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

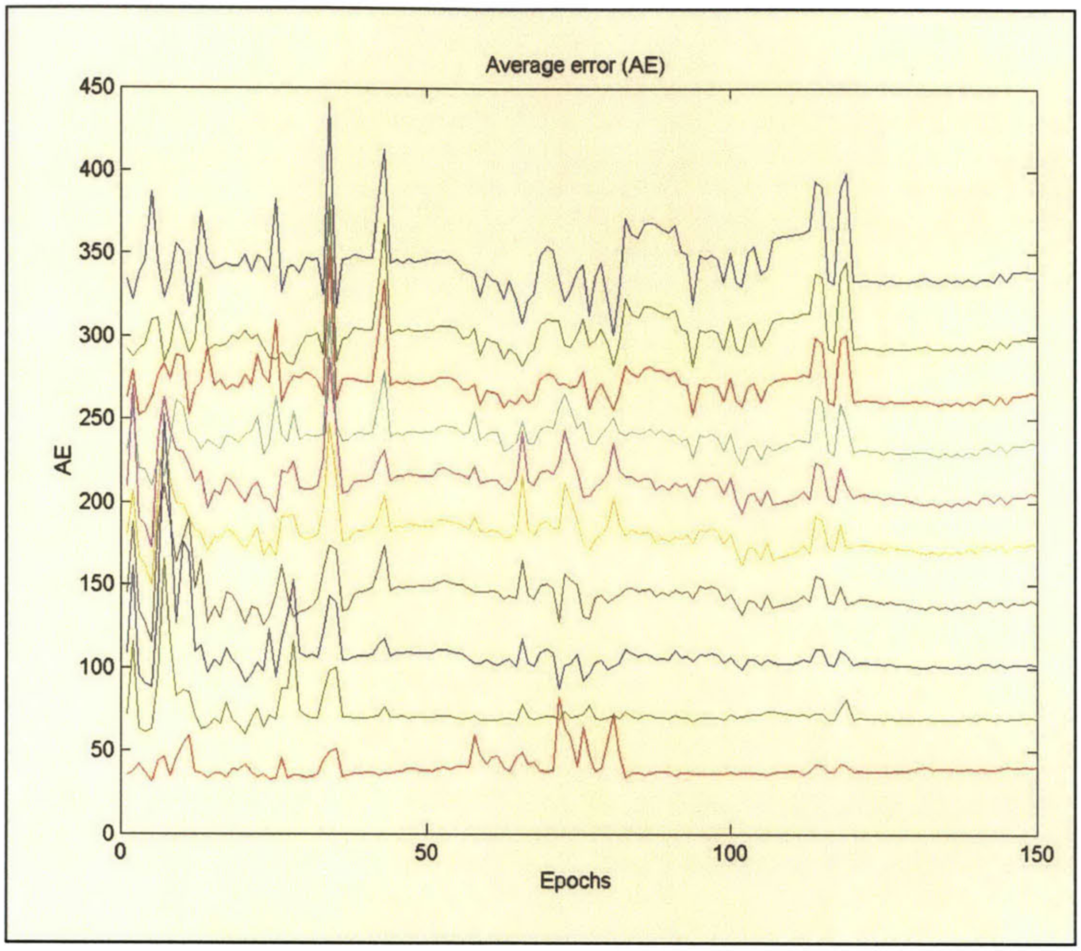


Figure 10. Average Error for a multiple runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

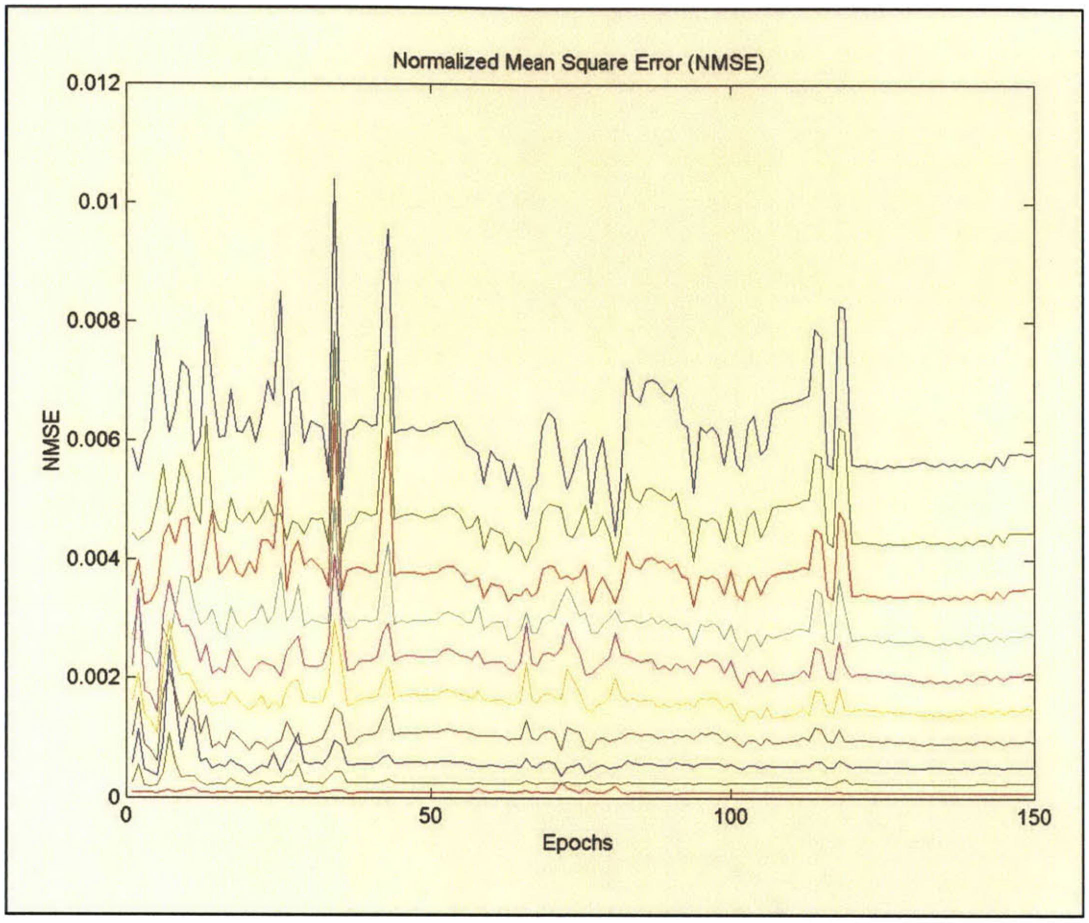


Figure 11. Normalized Mean Square Error for a multiple runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

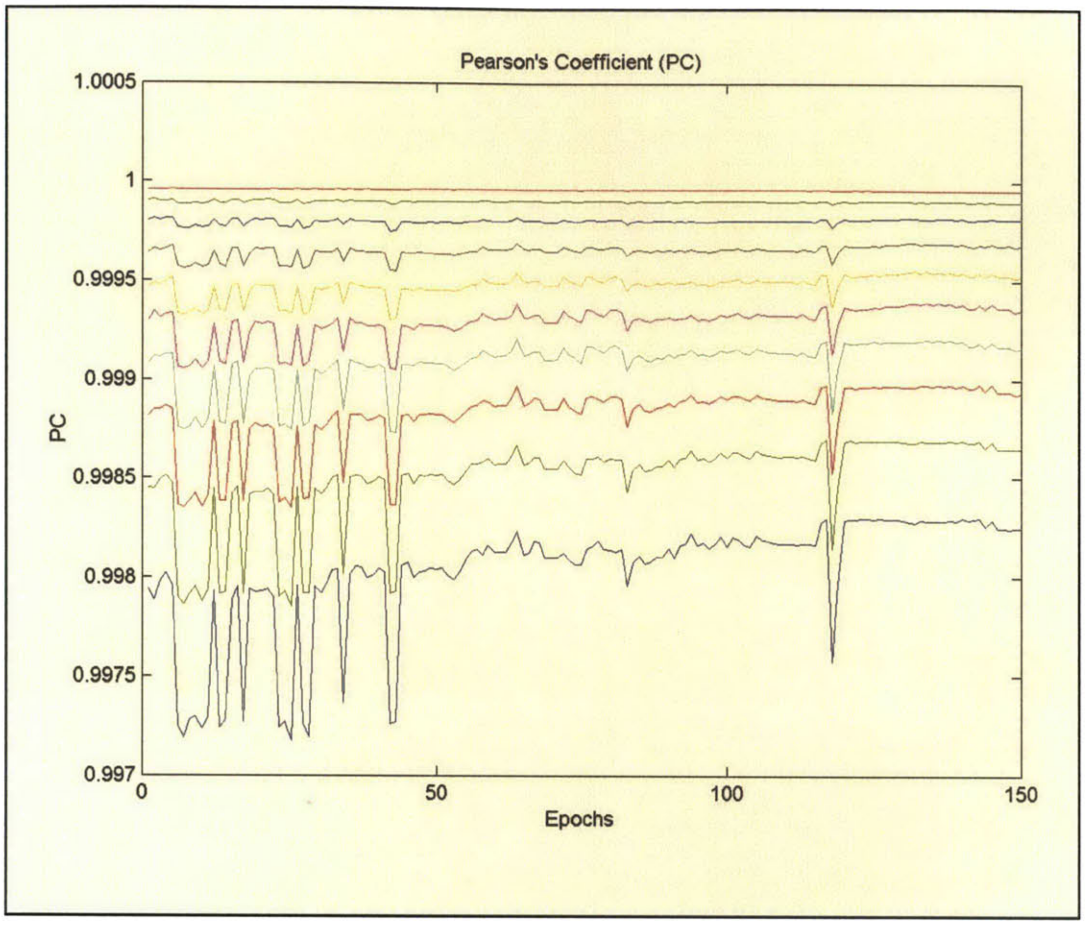


Figure 12. Pearson's Coefficient for a multiple runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

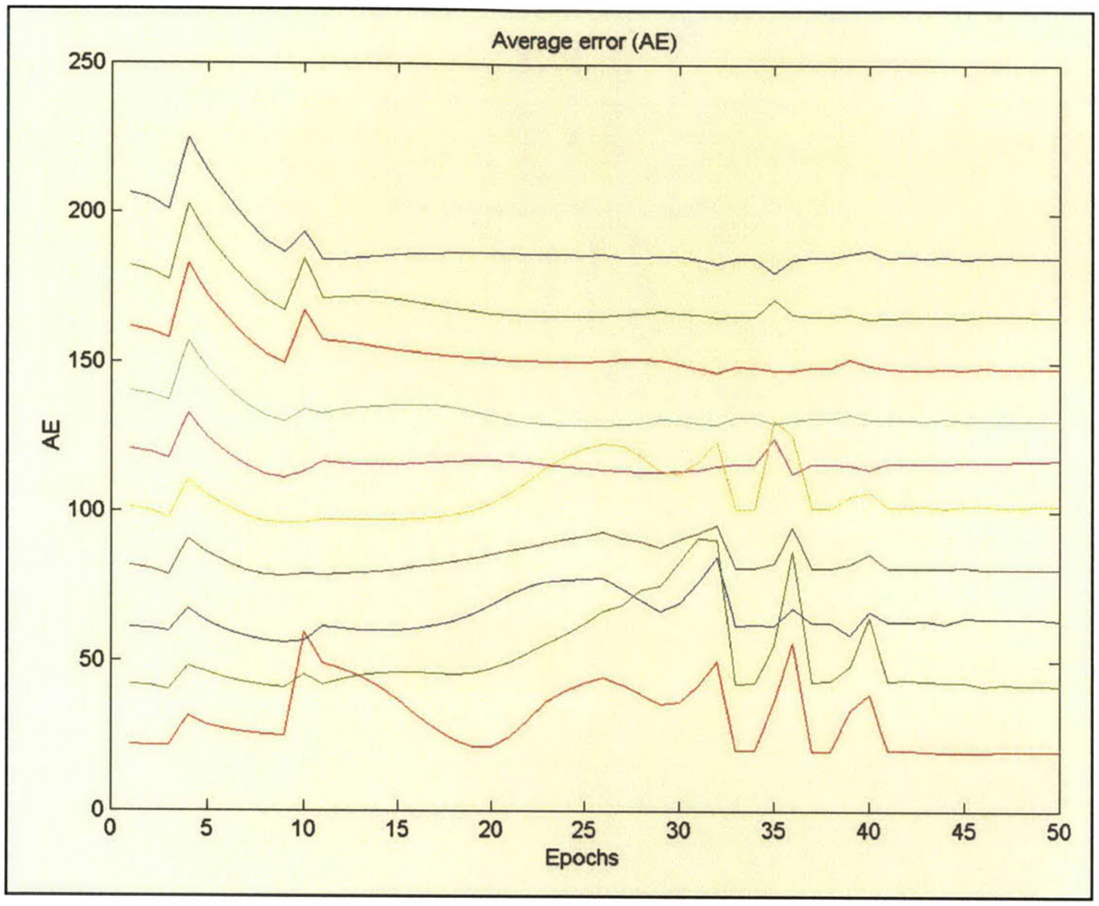


Figure 13. Average Error for a single runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

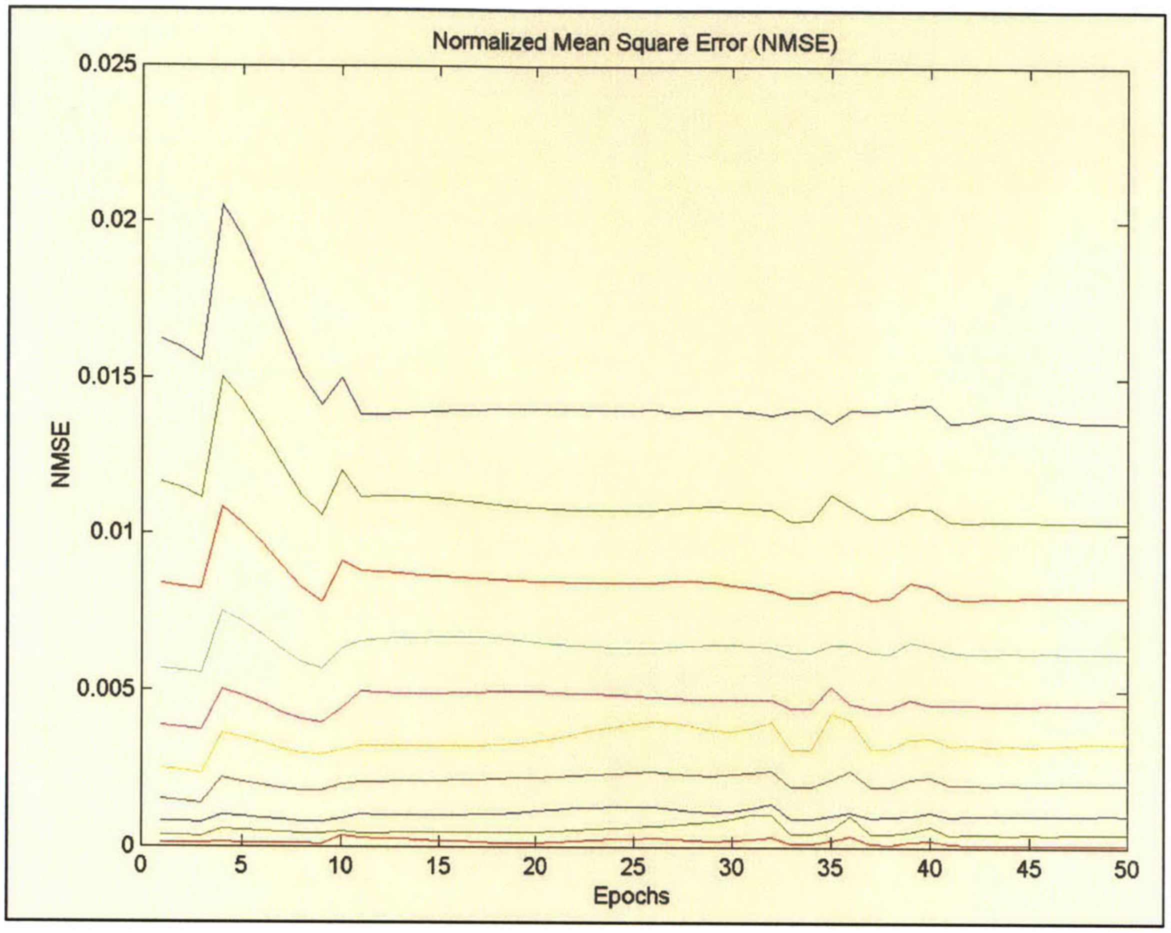


Figure 14. Normalized Mean Square Error for a single runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

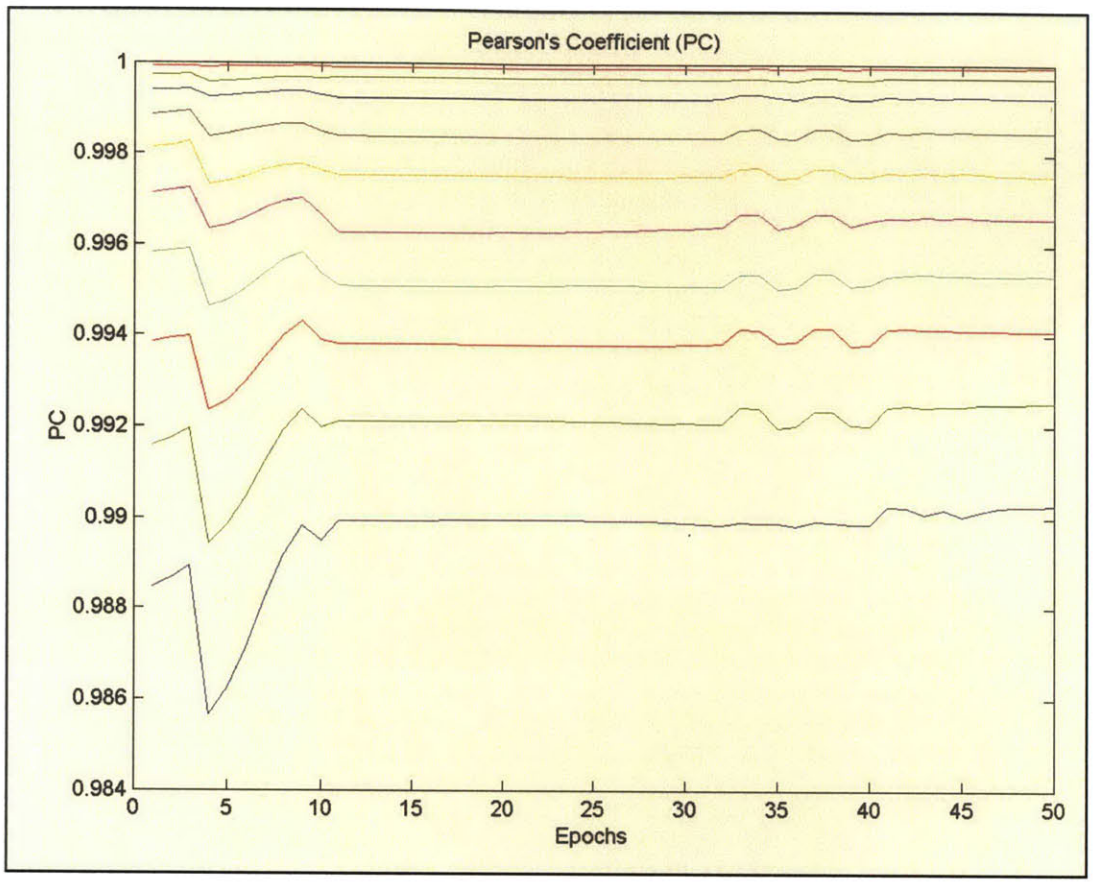


Figure 15. Pearson's Coefficient for a single runway case. Each line in the graph represents the output for a second in the future with the top most line being 10 seconds through 1 second (the bottom most line) into the future.

6.4 Comparison of AMASS vs. NN

The current AMASS system uses physically based rules to predict the future position of each aircraft using its current position and its position at previous instances in time. These positions are used to compute the velocity and the acceleration. This process provides reasonably good estimates, but it totally disregards the actions that are likely to be taken by the pilot or the human controller on the ground. The Neural Network based approach, on the other hand, places heavy emphasis on past behavior and embodies the expected actions and reactions. As such, it becomes difficult to compare the results. In the analysis below, we have compared several cases, each involving a pair of aircraft.

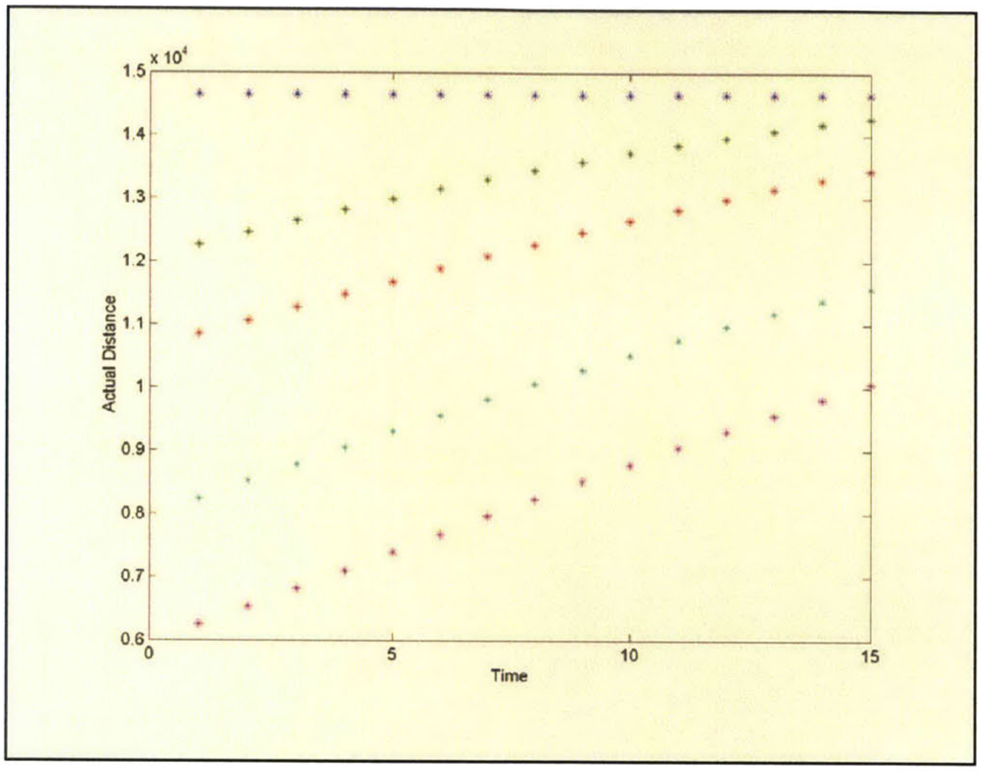


Figure 16. Actual Separation Distance (in feet) (Calculated from the Log files of AMASS)

Figure 16 shows the actual separation distances (in feet) versus Time (in seconds) for 15 seconds in the future as evaluated from the present the current report time by ASDE radar. Each color in the graph shows the separation distances for each of 5 pairs of planes.

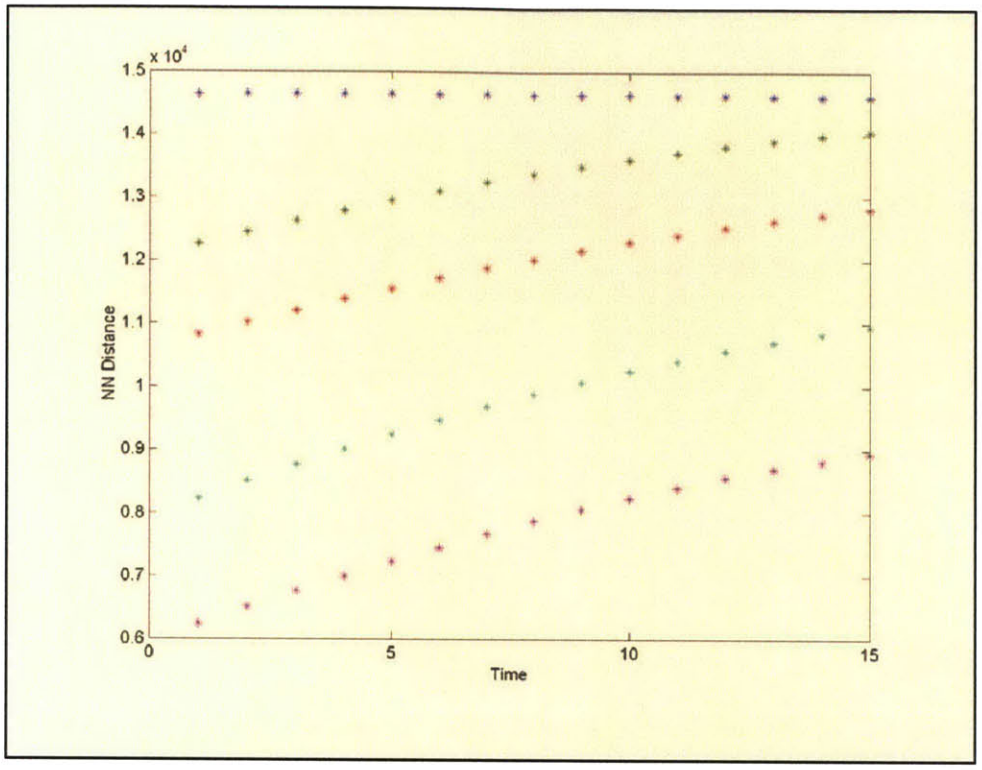


Figure 17. NN Separation Distance (Predicted Separation Distance using NN)

Figure 17 shows the predicted separation distance (in feet) versus time (in seconds) for 15 seconds in future with the neural network system using information from the past 5 seconds for each pair of plane as the input to the neural network. The figure shows the NN predicted separation distance from 1 second through 15 seconds in the future. (Please note that the current time stamp is assumed to be time 0). Each color in the figure shows the separation distances for each of the 5 pair of planes.

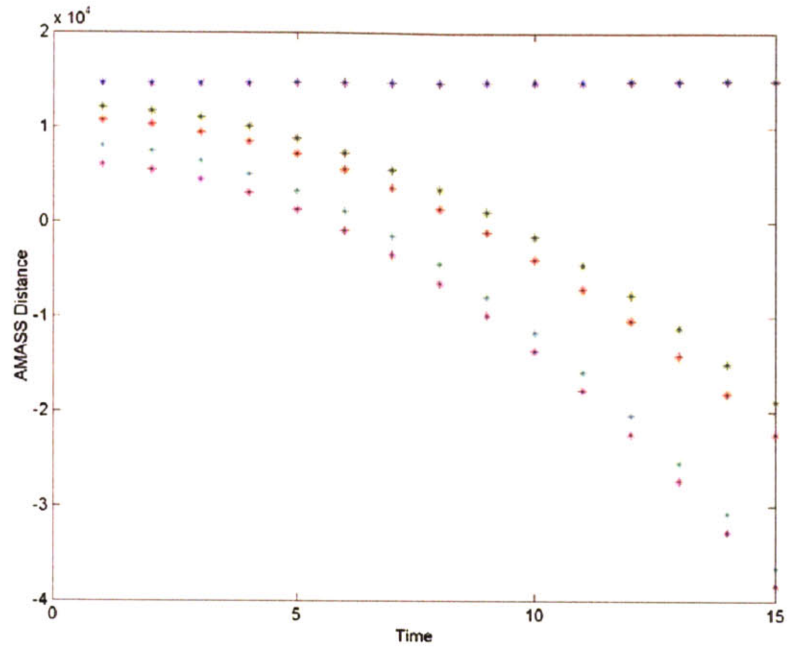


Figure 18. AMASS predicted separation distance

Figure 18 shows the AMASS predicted separation distances (in feet) versus. Time (in seconds) from 1 second through 15 seconds in the future. Here separation distances are calculated using 5 seconds of history for each plane and the present time stamp or current report. The results do not depict the output from actual AMASS system as such output was not available in the form needed for this comparison. Instead, we have simulated the AMASS approach to enable this comparison study.

Again, each color in the graph shows the separation distances for each of the 5 pair of planes.

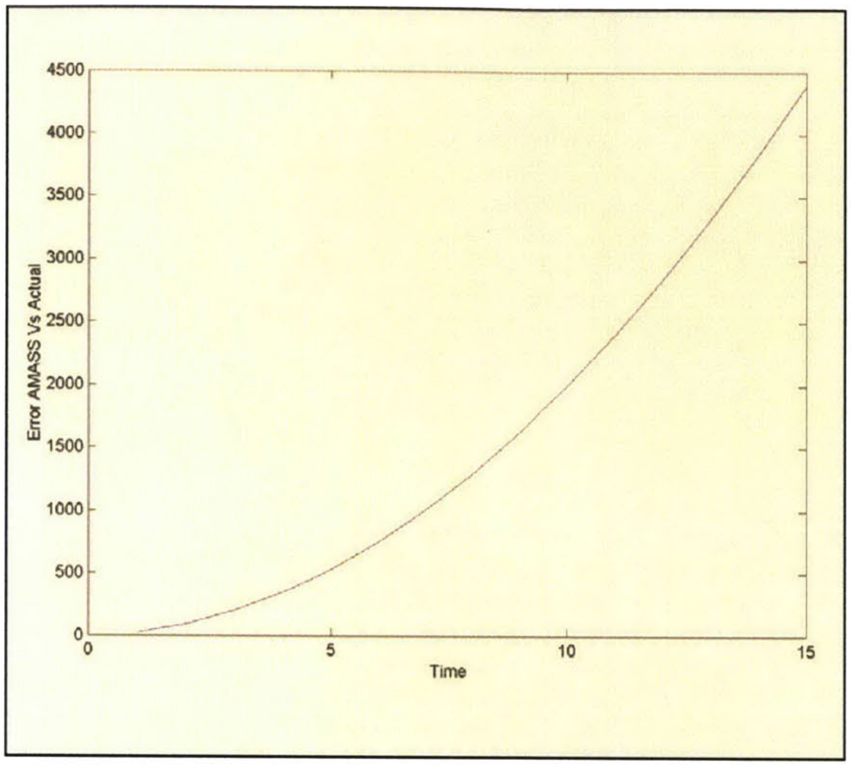


Figure 19. Actual Distance Predictions obtained using a Simulated Version of AMASS approach

Figure 19 shows that the Average errors between the AMASS predictions and actual distances increases as a quadratic function with the independent variable being time in future.

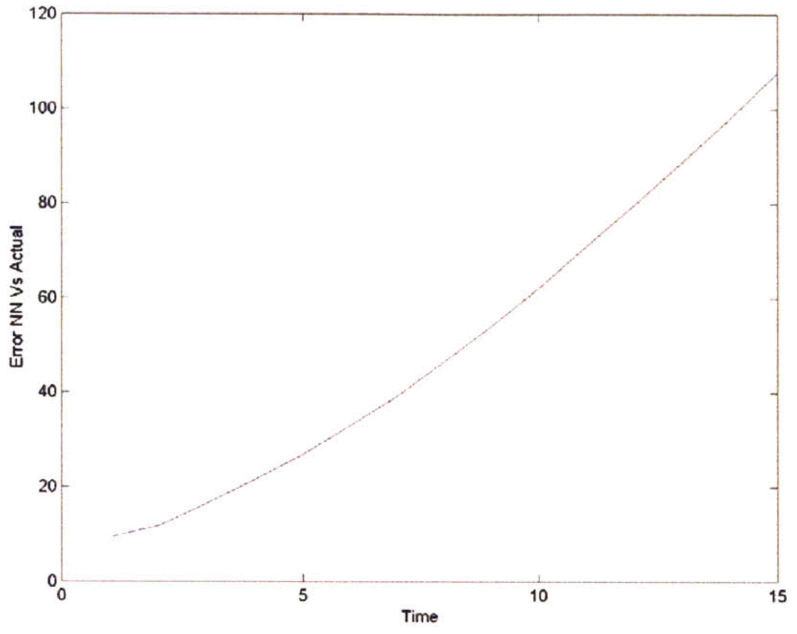


Figure 20. NN Vs Actual Distance - Overall Comparison

From the above preliminary comparative study of the results obtained using the AMASS and NN models, one can see that NN models perform significantly better compared to AMASS with respect to predicting future separation distances. AMASS shows biases in predicting the separation distances as prediction is based on quadratic distance formulae. The longer one looks out in the future, the poorer is the quality of the predicted values provided by AMASS. This is expected, because AMASS does not take into account the changes in flight speeds or directions made by the concerned pilots in future. These impacts of such changes magnify over time. As aircrafts approach the airport, the pilots reduce the speeds. Such expected changes are taken into account by the Neural Network based system.

7. Conclusions

This thesis examines the ability of different feed-forward neural networks to predict the alert situations on an airport based on the input from the airport radar sites. The input parameters reflect the current and past conditions of a pair of aircraft. Analysis using different type of neural networks is presented. This thesis also analyzes the performance of the neural networks based on some statistical parameters. The accuracy of the prediction depends on the number of nodes in hidden layers and the training algorithm used in the network. Different configurations of neural networks have been exploited to best represent the airport scenario.

Three types of NN based data mining techniques were applied to predict the runway incursion situations from the AMASS raw data. Both RNN and MLP models exhibit the least average errors. A drawback to RNN models, however, is that these models take a long time to train as compared to MLP models. On the other hand, SLP models take a long time to train. SLP models also failed to emulate the different complex airport scenarios and were hence not useful for this research.

The MLP and RNN based models were used to analyze the underlying AMASS data. Several models that focused on predicting the future separation distance as the output. These models yielded very good results. The average errors were well below 1000 feet for predictions of up to 10 seconds in the future. The normalized mean square errors were also fairly low and the Pearson's coefficients stabilized close to a value of 1 which represents perfect correlation between the predictions and the actual value of separation distances. Combing all three performance metrics it is concluded that these models can be applied for future studies to integrate NN based models into AMASS.

Some MLP based models used derived parameters like separation distances, velocities and separation time while others were based more heavily on absolute parameters taken directly from the raw data. Derived parameters like separation distance and separation velocity have the same connotations for all runways and airports. As a result, such models ideally do not require the training exercise to be repeated for each runway or airport under constant traffic and runway configuration. Due to different runway configuration and airport traffic this does not hold true and hence the models have to be trained for each airport.

In general, neural network techniques have significant training costs, both in terms of computing time and personnel. A single layer perceptron can take several days of computing time for the initial training process, which involves the use of huge amounts of historical data to compute the weights. (This is a process similar to determining the coefficients of the various variables in a regression equation.) Once the network has been trained, typically using huge amount of data from a runway, the model can be run on commodity personal computers to make accurate predictions of separation distances between two aircrafts. These trained NNs can be ideally run on real time basis with the direct input from the ASDE, the airport radar.

In an operational environment, input data for neural networks will come from relevant radar sites. Such data are currently utilized to update the AMASS system on an ongoing basis. NN based models will use such data from radar sites. Appropriate data filters will need to be incorporated to enable direct feed of relevant subsets of data to the neural network. These models will then perform predictions for separation distances, on a real-time and continued bases, and provide such information to the human controllers, either directly or via AMASS.

It is clear that the application of neural network data mining has been able to model the airport scenarios pretty accurately. Section 6.4 gives a detailed comparison study between the safety systems, AMASS that is currently used as opposed to the NN based models. The results show an enormous amount of prediction capability of NNs in terms of errors as well as an early prediction time.

Several networks demonstrate learning and could eventually be integrated with AMASS. However, these networks have different applications as some predicts Alert and Non-Alert situation and some predict separation distances between two aircrafts and thus would be applied accordingly. Based on the current operations of neural networks it is initially projected that the neural networks would be used to examine AMASS log data (offline) to generate safety parameters. No modifications to the AMASS software would be necessary. The neural networks use the AMASS log data and data filter software to extract necessary information. Offline analysis will allow an individual to crosscheck the safety parameters on the AMASS playback platform before they are installed on the

AMASS system. In the future, the technologies discovered in these neural network models can help generate new versions of safety logic software for the latest surveillance systems.

References

[1] Bansal Kanti, Vadhavkar Sanjeev and Dr. Amar Gupta, "Neural Network Based Data Mining Applications for Medical Inventory Problems", *Data Mining and Knowledge Discovery*, 2 97-102, 1998, Kluwer Academic Publishers.

[2] Reyes, C., A. Ganguly, G. Lemus, and A. Gupta. A Hybrid Model Based on Dynamic Programming, Neural Networks, and Surrogate Value for Inventory Optimization Applications. *Journal of the Operational Research Society*, 49, 1998, 1-10.

[3] Bansal, K., S. Vadhavkar, and A. Gupta. Neural Networks Based Data Mining Applications for Medical Inventory Problems. *International Journal of Agile Manufacturing*. 1-2, 1998, 187-200.

[4] Prokhorov, D., E. Saadm and D. Wunsch. Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks. *IEEE Transactions on Neural Networks*, 9-6, 1998.

[5] Agarwal, N., E. N. Anagnostou. Investigating Improvements in Precipitation Classification from Ground Based Weather Radar Observations. *International Symposium on Hydrological Applications of Weather Radar*. 2001 Kyoto, Japan.

[6] Gupta, A., S. Vadhavkar and S. Au. Data Mining for Electronic Commerce. *Electronic Commerce Advisor*. 4-2. 1999, 24-30.

[7] Penny, W. and D. Frost. Neural Networks in Clinical Medicine. Medical Decision Making: An International Journal of the Society for Medical Decision Making. 16-4, 1996.

[8] Neural Network for Optimization of AMASS. Prepared for FAA. 2002.

[9] Avelino J. Gonzalez & Douglas D. Dankel, "The Engineering of Knowledge-based Systems", 1993 Prentice-Hall Inc. ISBN 0-13-334293-X.

[10] Daniel Klerfors & Dr. Terry Huston, "Artificial Neural Networks", 1998 (http://hem.hj.se/~de96klda/NeuralNetworks.htm)

[11] Avelino J. Gonzalez & Douglas D. Dankel, "The Engineering of Knowledge-based Systems", 1993 Prentice-Hall Inc. ISBN 0-13-334293-X

[12] Haykin Simon, "Neural Networks", 1994 Macmillan College Publishing Company Inc. ISBN 0-02-352761-7

[13] Bishop Chris, "Neural Networks for Pattern Recognition", 1995 Oxford University Press, ISBN 0-19853-8642

[14] Jordan, M.I., 1986. Serial Order: A Parallel, Distributed Processing Approach. In Advances in Connectionist Theory: Speech, eds. J.L. Elman and D.E. Rumelhart. Hillsdale: Erlbaum.

[15] Elman, J.L. 1990. Finding Structure in Time, Cognitive Science 14, 179-211.

[16] Williams, R.J. and D. Zisper. 1989. A Learning Algorithm for Continually Running Full Recurrent Learning Algorithm. Connection Science 1, 87-111.

[17] Rumelhart, D.E., G.E. Hinton, and R.J. Williams. 1986. Learning Representations by Back-Propagating errors. Nature 323, 533-536. Reprinted in Anderson and Rosenberg [1988].

Appendix 1

SINGLE-LAYERED PERCEPTRON

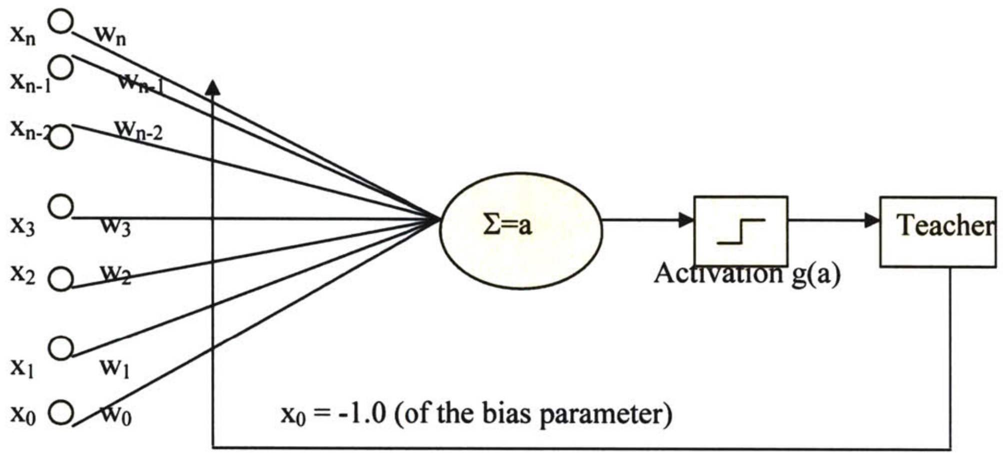


Figure 1. A Single Layered Perceptron Network.

The above figure shows the structure of a perceptron network [3] [4]. The values are as follows:

- x_0 is permanently set to -1.0 for the bias parameter w_0.
- The weight vector **w** exists for each class C_k and is a n dimensional vector.
- $\Sigma = \mathbf{w}^T\mathbf{x}$ (inner product that serves as the argument to the activation function.)
- Activation Function g(a) : The anti-symmetric version of the logistic function such that
  $g(a) = 1$ when $a \geq 0$
  $g(a) = -1$ when $a < 0$

Algorithm:

Step 1. Initialize w_j^1 – the weight vector for each class C_j.
Step 2. Start iteration, ii = 1, 100 (say 100 epochs)

```
DO ii =1, 1000
  N_error=0
    DO n =1, $N_{train}$   ($N_{train}$ is the total number of patterns)
      Pick a training data $\mathbf{x}^{n}_{train}$ whose class argument j is known.
      Compute $\mathbf{w}^{n}_{j}{}^{T}\mathbf{x}^{n}_{train}$ for each class $C_j$. (The weight vector exist for each class
      Compute k, where k= arg max i $(\mathbf{w}^{n}_{i}{}^{T}\mathbf{x}^{n}_{train})$
    If j=k do nothing (go to the next pattern, n=n+1) (right classification)
    Else
        N_error=N_error+1
        $w_j^{n+1}= w_j^{n} + \eta\mathbf{x}^{n}_{train}$                                        (7)
        $w_k^{n+1}= w_k^{n} - \eta\mathbf{x}^{n}_{train}$                                        (8)
    ENDIF
    ENDDO
 ENDDO  (End iteration once % Misclassification rate stabilizes to a nominal
        value ( i.e., becomes constant)
```

Appendix 2

MULTI-LAYER PERCEPTRON

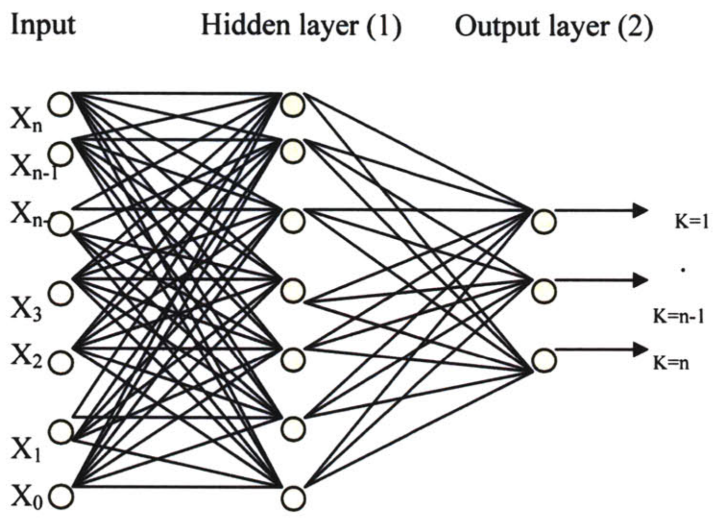


Figure 1. A Two Layered Perceptron Network.

Here the following should be noted.

- x_0 is permanently set to -1.0 for the bias parameter w_0.
- The weight vector **w** exists for each class C_k and is a n dimensional vector.
- $\Sigma = \mathbf{w}^T\mathbf{x}$ (inner product that serves as the argument to the activation function.)
- Activation Function g(a) : The logistic sigmoid function such that
  $g(a) = 1$ when $a \geq 0$
  $g(a) = 0$ when $a < 0$
- $\mathbf{w}_{ji}^{(L)}$ refers to the weight in layer (L) connecting the ith node of the (L-1)th layer to the j th node of the Lth layer.
- The same number of hidden units (n) were chosen for the hidden layer.

Algorithm:

(Training of Weights)

Step 1. Initialize w_{ji}^{L} – the weight vector for each class C_j of layer L.
Step 2. Start iteration, ii = 1, 1000 (say)

Step 3. Forward Propagation:

i) Pick a training data $\mathbf{x}_{train}$

ii) Compute a_j at hidden layer for each of the hidden nodes as

$$a_j = \Sigma\, w_{ji}^{(1)}\, x_j$$

iii) Compute activation $g(a_j)$ for each node as,

$$g(a_j) = \frac{1}{1+e^{-a_j}}$$

iv) Set $z_j = g(a_j)$ at the hidden layer

v) Compute ak for each output node as

$$a_k = \Sigma\, w_{ji}^{(2)}\, z_i$$

vi) Compute the output activation y_k using the activation function as in step iii). ($y_k = g(a_k)$)

vii) Compute the error signal e_k for each output node as

$$e_k = y_k - t_k$$

where t_k is the target value for class k (i.e. if $\mathbf{x}_{train}$ belongs to class k then t_k is 1.0. It is zero for the other classes). Summing up squares of e_k over k and then halving gives the value of the error function E at iteration i.

$$E = \frac{1}{2}\sum_{k=1}^{3}(y_k - t_k)^2$$

Step 4. Backward propagation (Computing local gradients at each node)

i) Compute δ_k at each output node. It is the same as the error signal e_k.

ii) Back propagate δ s to the hidden layer as

$$\delta_j = z_j(1 - z_j)\sum_{k=1}^{3} w_{kj}\,(2)\delta_k$$

iii) Compute the derivative of the Error function for the 1st and 2nd layer as

$$\frac{\partial E}{\partial w_{ji}} = \delta_j \, x_i$$

$$\frac{\partial E}{\partial w_{kj}} = \delta_k z_j$$

iv) Update the weights by an increment (at each layer) Δw_{ji} as

$\Delta w_{ji} = - \eta\delta \, x_i$ and $\Delta w_{kj} = - \eta\delta \, z_j$

OR apply a suitable gradient scheme (conjugate gradient, incremental gradient or memory less quasi Newton method) to update the weights

Step 5. Repeat steps 3 to 4 till the Error function value minimizes.

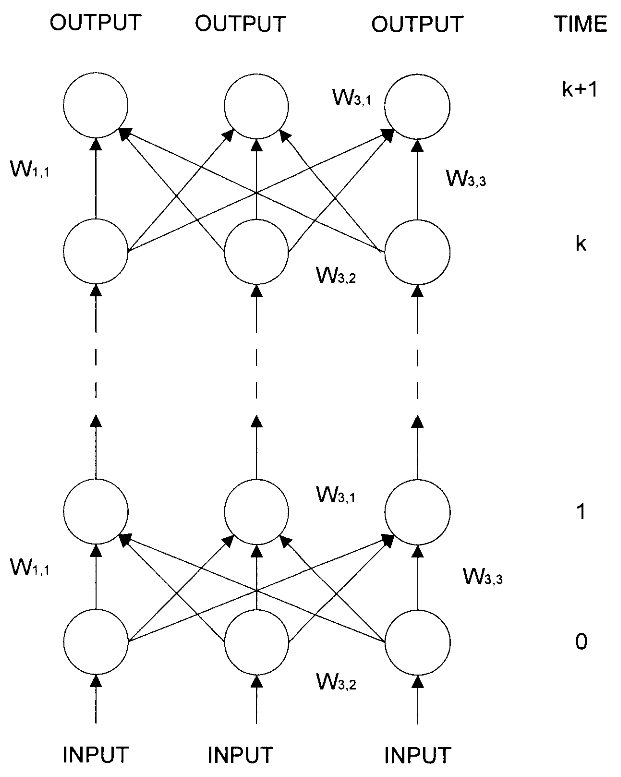
Appendix 3

<u>Recurrent Neural Network (RNN)</u>

RNNs are connected networks from output nodes to hidden layer and/or input layer nodes, and they also allow interconnectivity between nodes of the same layer, specifically between the nodes of hidden layers.

Various training algorithms for RNN have been proposed by several researchers such as Jordan (1986); Rumelhart, Hinton, and Williams (1986); Pineda (1989); Williams and Zipser (1989); and Elman (1990). Elman training algorithm is being used in this research.

Each layer is considered of as representing a time delay in the network. Figure below shows the fully connected NN with k hidden layers. Since a unique weight is associated with any connection in the RNN, connection weights in the "equivalent" feed forward network cannot be completely arbitrary and weights connecting nodes from one layer to next layer are identical for all layers of weights as shown.



Algorithm

Step 1: Start with randomly chosen weights.

Step 2: Let time t = 0, and assume the initial condition

$$\frac{\partial y_k}{\partial w_{i,j}} = 0, \quad \forall\, i, j, k.$$

Step 3: While MSE is unsatisfactory and computational bounds are not exceeded, do

Step 4: Compute the output of each node at the next time instant:

$$y_k(t+1) = f\left(\sum_{l \in U \bigcup I} w_{k,l} z_l(t) \right)$$

Step 5: Compute the error $\sum_k \left(d_k(t) - y_k(t)\right)^2$,

Step 6: for nodes $k \in U$ with a specified target value $d_k(t)$.

Step 7: Modify the targets:

$$\Delta w_{i,j}(t) = \eta \sum_k (d_k(t) - y_k(t))^2,$$

Step 8: For use in the next iteration, compute

$$\frac{\partial y_k(t+1)}{\partial w_{i,j}} = f_k^{'}(net_k(t))\left[\sum w_{i,j} \frac{\partial z_k(t)}{\partial w_{i,j}} + \delta_{i,j} z_j(t)\right]$$

Step 9: Increment time counter t;

Step 10: end while.

Appendix 4

All the codes are written in MATLAB 6.0

Data Extraction Codes:

```
function main(filename, varargin)

clear global;
global gCurrentRecordTime gFileHeader
gRecordHeader;

%% Initialize
%% =======

%% Open file
gFileHeader = opencsv(filename);
lLastByteReport = 0;
lLastPercentReport = 0;

stdout = 1;

%% What columns do we want in
records
gRecordHeader.labels = {'Time_Sec',...
      'PID','Tgt_Type',...
      'X_Pos','Y_Pos',...
      'X_Vel','Y_Vel' };

%% What column numbers do these
represent
gRecordHeader.columnNumbers = ...
   findcolumns( gRecordHeader.labels,...
   gFileHeader.labels);


%% Load filters to use
lFilters = getfilters('config.txt');
for iter = 1:length(lFilters)
   lFilters{iter,2}{1} = ...
      findcolumns( lFilters{iter,2}{1},...
      gRecordHeader.labels);
end % for
fprintf(stdout,'Loaded filters:\n',[]);
disp(lFilters);

%% Process file
%% =======

%% CSV file --> Records
%% =======

[lCurrRecord, lEOF] =
getrecord(gFileHeader,gRecordHeader.c
olumnNumbers);
% While still more records in file
fprintf(stdout,'Running\n',[]);
fprintf(stdout,'(0%%)',[]);
while ( ~ lEOF )

   %% Filter records
   %% =======

   % For each filter
   for lFilterIter = 1:size(lFilters,1)

      % Run the filter
      eval( ['lCurrRecord = '...
            lFilters{lFilterIter,1}...
            '( lCurrRecord,'...
            ' lFilters{lFilterIter,2} );'] );

      % If filtered out
      if ( isempty( lCurrRecord ) )
         % Can stop filtering this record
         break;
      end % If filtered out

   end % For each filter

   lCurrentByte = ftell(gFileHeader.fid);
   if (lCurrentByte > lLastByteReport +
100000)
      lLastByteReport = lCurrentByte;
```

```
        fprintf(stdout,'.',[]);
        lPercentDone =
round(100*(lCurrentByte /
gFileHeader.size));
        if (lPercentDone >
lLastPercentReport)
            lLastPercentReport =
lPercentDone;
            fprintf(stdout, [ '\n('
num2str(lPercentDone) '%%)' ],[]);
        end % if
    end % if

    % Get new record
    [lCurrRecord, lEOF] =
getrecord(gFileHeader,gRecordHeader.c
olumnNumbers);

end % While still more records in file

fclose(gFileHeader.fid);
```

```
function rFilters = getfilters(configfile)

rFilters = { };

if 0
% 1. Filter by time
rFilters = [...
        rFilters;...
        {...
            'filtertime',...
            {...
                {'Time_Sec'}....
                [...
                    959710808 959712008; ...
                    959720408 959724008 ...
                ]...
            }...
        }...
    ];
end

if 0
% 2. Filter 14R
rFilters = [...
        rFilters;...
        {...
            'filterconstraint',...
            {...
                {'X_Pos' 'Y_Pos'}...
                [...
                    -0.7590 -0.6511 ...
                ]...
                [...
                    -2513.0 ...
                ]...
            }...
        };...
        {...
            'filterconstraint',...
            {...
                {'X_Pos' 'Y_Pos'}...
                [...
                    0.7503 0.6611 ...
                ]...
                [...
                    3076.0 ...
                ]...
            }...
        }...
    ];
end

if 0
% 2. Filter 34R for SEA
rFilters = [...
        rFilters;...
        {...
            'filterconstraint',...
            {...
                {'X_Pos' 'Y_Pos'}...
                [...
                    1.0 0.0 ...
                ]...
                [...
                    1608.0 ...
                ]...
            }...
        };...
        {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                -1.0 0.0 ...
            ]...
            [...
                -968.0 ...
            ]...
        }...
    }...
  ];
end

if 0
% 2. Filter 34L for SEA
rFilters = [...
    rFilters;...
    {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                1.0 0.0 ...
            ]...
            [...
                2332.0 ...
            ]...
        }...
    };...
    {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                -1.0 0.0 ...
            ]...
            [...
                -1664.0 ...
            ]...
        }...
    }...
  ];
end

if 0
% 2. Filter 28L for SFO
rFilters = [...
    rFilters;...
    {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                0.4725 0.8813 ...
            ]...
            [...
                -1.6319e+003 ...
            ]...
        }...
    };...
    {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                -0.4574 -0.8893 ...
            ]...
            [...
                1.7141e+003 ...
            ]...
        }...
    }...
  ];
end

if 0
% 2. Filter 1R for SFO
rFilters = [...
    rFilters;...
    {...
        'filterconstraint',...
        {...
            {'X_Pos' 'Y_Pos'}...
            [...
                0.8839 -0.4678 ...
            ]...
            [...
                -2.2906e+003 ...
            ]...
        }...
        };...
        {...
          'filterconstraint',...
          {...
            {'X_Pos' 'Y_Pos'}...
            [...
              -0.8864 0.4630 ...
            ]...
            [...
              2.5129e+003 ...
            ]...
          }...
        }...
    ];
end

if 0
% 2. Filter Runway 22L for ORD
rFilters = [...
        rFilters;...
        {...
          'filterconstraint',...
          {...
            {'X_Pos' 'Y_Pos'}...
            [...
              0.7421 -0.6703 ...
            ]...
            [...
              -5788.1 ...
            ]...
          }...
        };...
        {...
          'filterconstraint',...
          {...
            {'X_Pos' 'Y_Pos'}...
            [...
              -0.7289 0.6846 ...
            ]...
            [...
              6787.4 ...
            ]...
          }...
        }...
    ];
end

if 1
% 2. Filter Runway 17L for DFW
rFilters = [...
        rFilters;...
        {...
          'filterconstraint',...
          {...
            {'X_Pos' 'Y_Pos'}...
            [...
              1 0 ...
            ]...
            [...
              -2481 ...
            ]...
          }...
        };...
        {...
          'filterconstraint',...
          {...
            {'X_Pos' 'Y_Pos'}...
            [...
              -1 0 ...
            ]...
            [...
              2832 ...
            ]...
          }...
        }...
    ];
end


if (1)
% Filter planes
rFilters = [...
        rFilters;...
        {...
          'filterplane'...
          {...

{'Time_Sec','PID','Tgt_Type',...

'X_Pos','Y_Pos','X_Vel','Y_Vel'}...
          }...
        }...
```

```
   ];
end % if

% 3. Output record
rFilters = [...
      rFilters;...
      {...
         'filteroutputrecord'...
         {...
            {}
            'outputfile.txt'...
         }...
      }...
   ];
```

```
function [a, b, c] = normalform2d(p1,
p2)
% Normal form of line passing through
p1 and p2
%   ax + by + c = 0, a^2 + b^2 = 1

dp = p2 - p1;

% Initial values
a = dp(2);
b = - dp(1);
c = - p1(1)*dp(2) + p1(2)*dp(1);

nFactor = (a^2 + b^2)^(1/2);

a = a/nFactor;
b = b/nFactor;
c = c/nFactor;
```

```
function rHeader = opencsv(filename)
% opencsv
% Opens file and creates a header struct
% Assumes that file is a CSV file and
that the first line
%   contains the column labels
% rHeader:
%   .fid: file id for file
%   .labels: cell array of strings

% open file
rHeader.fid = fopen(filename);
if (-1 == rHeader.fid)
   error(['opencsv: Unable to open file '
filename])
end


fseek(rHeader.fid,0,'eof');
rHeader.size = ftell(rHeader.fid);
fseek(rHeader.fid,0,'bof');

% get column labels
lLine = fgetl(rHeader.fid);
rHeader.labels = {};
[lLabel, lLine] = strtok(lLine,',');
while( ~ isempty(lLabel) )
   rHeader.labels = [ rHeader.labels
{lLabel} ];
   [lLabel, lLine] = strtok(lLine,',');
end

disp(['Opened ' filename ': ('
num2str(rHeader.size) ' Bytes)']);
```

```
function [rRecord, varargout] =
getrecord( fileHeader, varargin )
% getrecord: Get next valid record from
file
% [rRecord [, rEOF]] = getrecord(
fileHeader [, desiredColumns] )
%   rRecord: array of desired columns
%   rEOF: 1 iff reach end of file
%   fileHeader: must have (.fid, .labels)
%   desiredColumns: array of file
columns to put in rRecord


% Check Arguments
try
   lFid = fileHeader.fid;
   if (0 < length(varargin))
      desiredColumns = varargin{1};
   else
```

```
      desiredColumns =
[1:1:length(fileHeader.labels)];
   end
catch
   error(['getrecord.m: Invalid
arguments'])
end
rRecord = [];
varargout{1} = 0;

% Get next valid record
while ( isempty(rRecord) & ~
varargout{1} )

   % Get Line from file
   lLine = fgetl(lFid);
   if (-1 == lLine)
      % reached EOF
      varargout{1} = 1;
      warning('getrecord: EOF');
      continue;
   end
   % tokenize line
   lRow = {};
   while ( ~isempty(lLine) )
      [lToken,lLine] = strtok(lLine,',');
      lRow = [lRow {lToken}];
   end

   % Get elements of record and convert
to number
   try
      for iter = 1:length(desiredColumns)
         lToken = str2num(
lRow{desiredColumns( iter )} );
         if (isempty( lToken ))
            nToken = 0;
         end
         % Save the value
         rRecord(iter) = lToken;
      end
   catch
      warning(['getrecord: Error while
getting record']);
      warning([' ' lasterr]);
      rRecord = [];
   end
end
```

All the codes are written in MATLAB 6.0

Code for Calculating performance Metrics and plotting them:

```
function [ae,nmse,pc] =
testnet_v2(net,epochs,times,trainIn,train
Out,testIn,testOut)
% Calculates performance metrics for
network

global hAePlot hNmsePlot

prevEpochs = net.trainParam.epochs;
net.trainParam.epochs = epochs;
prevShow = net.trainParam.show;
net.trainParam.show = NaN;


nTest = size(testOut,2);
meanTrainOut = mean(trainOut,2);

fprintf(2,'Computing performance for
%d trainings of %d
epochs\n',[times,epochs]);


ae = [];
nmse = [];
pc = [];

for iter=1:times

   % Train batches of 500
   for jter = 1:size(trainOut,2)/500
      first = (jter-1)*500 + 1;
      last = (jter)*500;
      net =
train(net,trainIn(:,first:last),trainOut(:,fir
st:last));
   end %for

   simOut = sim(net,testIn);

   [currAe,currNmse,currPc] =
performance(meanTrainOut,testOut,sim
Out);
   ae = [ae,currAe];
   nmse = [nmse,currNmse];
   pc = [pc,currPc];

   fprintf(2,'%d...',[iter]);

end % times

hAePlot = figure;
figure(hAePlot);
plot([epochs:epochs:iter*epochs],[ae]);
xlabel('Epochs');
ylabel('AE');
title('Average error (AE)');

hNmsePlot = figure;
figure(hNmsePlot);
plot([epochs:epochs:iter*epochs],[nmse]
);
xlabel('Epochs');
ylabel('NMSE');
title('Normalized Mean Square Error
(NMSE)');

hPcPlot = figure;
figure(hPcPlot);
plot([epochs:epochs:iter*epochs],[pc]);
xlabel('Epochs');
ylabel('PC');
title('Pearson''s Coefficient (PC)');


fprintf(2,'...done\n',[]);
```

```
function [ae,nmse,pc] =
performance(meanTrainOut,testOut,sim
Out)

% Number of outputs and test set size
nOutputs = size(testOut,1);
nTest = size(testOut,2);
```

```
% For each output, measure performance
for iter = 1:nOutputs

   ae(iter,1) = sum(abs(testOut(iter,:)-
simOut(iter,:)))/nTest;

   nmse(iter,1) = sum(((testOut(iter,:)-
simOut(iter,:)).^2))./sum(((testOut(iter,:)
-meanTrainOut(iter,:)).^2));

   pc(iter,1) =
pearson(simOut(iter,:),testOut(iter,:));

end
```

```
function correlationXY = pearson(x,y)

if (size(y,1)~=1)
   x = x';
   y = y';
end

% input/output as row vectors
n = size(y,2);
sum_x = sum(x,2);
sum_y = sum(y,2);
sum_sqx = sum(x.^2,2);
sum_sqy = sum(y.^2,2);
dotp_xy = x*y';

correlationXY = (n*dotp_xy -
sum_x*sum_y) ./...
   ( sqrt(( n*sum_sqx - sum_x.^2 ) * (
n*sum_sqy - sum_y.^2 )) );
```

Code to make pairs of planes and prepare input to NN.

```
function
[trainIn,trainOut,testIn,testOut,sd_amass
_trainOut,sd_amass_testOut,dataSize,tot
al_info,history]=
loadtrain_v3_AMASS_mod_andrewSt(fi
lein,tFuture)

% Load train/test for series of positions
(version 2)
% i.e. the positions for pair of planes
over time.
% input will be tPast Separation
distance,velocity,time for each plane
% output: distances for tFuture seconds


%% Initialize
%% -------

% Open file
tPast=5;
lFileHeader = opencsv(filein);
lRecordHeader.labels =
{'Time_Sec','PID','X_Pos','Y_Pos','X_Ve
l','Y_Vel'};
eTimeSec = 1; ePID = 2;
eXPos = 3; eYPos = 4; eXVel = 5;
eYVel = 6;
lRecordHeader.columns =
findcolumns(lRecordHeader.labels,lFile
Header.labels);


%% Load training set
%% -------

totaltrainIn = [];
totaltrainOut = [];
testIn = [];
testOut = [];
sd_amass_totalout = [];
total_info = [];
%total_history = {};
% Pi2 = [];
% Pj2 = [];


% Info on planes
% Rows of (PID, last seen, total time on
radar)
info=[];
% History of positions for planes
history={};
```

```
fl = 1;

% Get first record
[lRecord, lEOF] = getrecord(
lFileHeader, lRecordHeader.columns );
while (~lEOF)

   % Get one second's worth of planes

   lCurrentTime = lRecord(eTimeSec);
   while ( ~lEOF &...
         ( lRecord(eTimeSec) ==
lCurrentTime ) )

      % Add this record

      % Check if this PID has been
loaded before
      RecordIndex = [];
      if (~isempty(info))
         RecordIndex =
find(info(:,1)==lRecord(ePID));
      end % if

      if (isempty(RecordIndex))
         % This plane has not been loaded
before

         % Add plane to info and history
         info =
[lRecord(ePID),lCurrentTime,1;info];
         history =
[{lRecord([eXPos,eYPos,eXVel,eYVel]
)};history];
      else

         % Update time
         info(RecordIndex,2) =
lCurrentTime;
         info(RecordIndex,3) =
info(RecordIndex,3) + 1;

         % Update history
         history{RecordIndex} =
[lRecord([eXPos,eYPos,eXVel,eYVel]);
history{RecordIndex}];
      end % if

      % Get next record
      [lRecord, lEOF] = getrecord(
lFileHeader, lRecordHeader.columns );
   end % while one second

   % Delete old planes
   % Planes are old if they are not on
radar at current time
   if (~isempty(info))
      OldPlanes = find(info(:,2) ~=
lCurrentTime);
      info(OldPlanes,:) = [];
      history(OldPlanes,:) = [];
   end % if
   tPresent = tPast + 1;

   if (~isempty(info))
      MatchablePlanes =
find(tFuture+tPast <= info(:,3));


      for iter =
1:length(MatchablePlanes)
         for jter =
iter+1:length(MatchablePlanes)

            total_info = [total_info;
info(MatchablePlanes(iter),:)];
            total_info = [total_info;
info(MatchablePlanes(jter),:)];

            Pi =
history{MatchablePlanes(iter)};
            Pj =
history{MatchablePlanes(jter)};
            Pi1 =
flipdim(Pi(1:tPast+tFuture,:),1);
            Pj1 =
flipdim(Pj(1:tPast+tFuture,:),1);
```

```
%           if (iter==1 & jter == 2 & fl
== 1)
%              Pi2 = Pi1
%              Pj2 = Pj1
%              fl = 0;
%           end %if

         %SD According to AMASS
using userdefined projection time (time)
         dist_amass =
(Pi1([tPresent],1:2)-
Pj1([tPresent],1:2)).^2;
         dist_amass =
sqrt(dist_amass(:,1) + dist_amass(:,2));

         % Target i
         % Calculates the velocity,
acceleration and general direction

         %Present target ie at time 6
         xi = Pi1([tPresent],1);
         yi = Pi1([tPresent],2);
         gendir_i = position(xi ,yi);
         %END Present target ie at
time 6

         cum_dist_i(1)=0;
         for ii=2:tPresent
            dist_i = (Pi1([ii-1],1:2) -
Pi1([ii],1:2)).^2;
            dist_i = sqrt(dist_i(:,1) +
dist_i(:,2));
            cum_dist_i(ii)=dist_i +
cum_dist_i(ii-1);
         end%for

         vi = ((2*cum_dist_i(6)) +
(cum_dist_i(2)) - (cum_dist_i(4)) -
(2*cum_dist_i(5))) / 10;
         ai = ((2*cum_dist_i(6)) -
(cum_dist_i(2)) - (2*cum_dist_i(3)) -
(cum_dist_i(4)) + (2*cum_dist_i(5))) /
7;


         % Target j
         % Calculates the velocity,
acceleration and general direction

         xj = Pj1([tPresent],1);
         yj = Pj1([tPresent],2);
         gendir_j = position(xj ,yj);
         cum_dist_j(1)=0;
         for jj=2:tPresent
            dist_j = (Pj1([jj-1],1:2) -
Pj1([jj],1:2)).^2;
            dist_j = sqrt(dist_j(:,1) +
dist_j(:,2));
            cum_dist_j(jj)=dist_j +
cum_dist_j(jj-1);
         end%for

         vj = ((2*cum_dist_j(6)) +
(cum_dist_j(2)) - (cum_dist_j(4)) -
(2*cum_dist_j(5))) / 10;
         aj = ((2*cum_dist_j(6)) -
(cum_dist_j(2)) - (2*cum_dist_j(3)) -
(cum_dist_j(4)) + (2*cum_dist_j(5))) /
7;

         % Gets the distances traveled
in projection time

         for time=1:tFuture

            si(time) = vi*time +
(0.5*ai*(time^2));
            sj(time) = vj*time +
(0.5*aj*(time^2));

            %

         end %for

            %for CHASE situation
            %returns 1->N 2->E 3->S
4->W
            % General direction is north

            if (gendir_i == 1) | (gendir_j
== 1)
               if (yi >= yj)
```

```
                    ds = si - sj;
                 else
                    ds = sj - si;
                 end%if

              % General direction is south
              elseif (gendir_i == 3) |
(gendir_j == 3)
                 if (yi >= yj)
                    ds = sj - si;
                 else
                    ds = si - sj;
                 end%if

              % General direction is East
              elseif (gendir_i == 2) |
(gendir_j == 2)
                 if (xi >= xj)
                    ds = si - sj;
                 else
                    ds = sj - si;
                 end%if

              % General direction is West
              elseif (gendir_i == 4) |
(gendir_j == 4)
                 if (xi >= xj)
                    ds = sj - si;
                 else
                    ds = si - sj;
                 end%if
              end%if

              is = dist_amass + ds;


           % output value calculated here
           % Find closest distance for
tFuture seconds of history
           distance =
(Pi1([tPresent:tFuture+tPast],1:2)-
Pj1([tPresent:tFuture+tPast],1:2)).^2;
           distance = sqrt(distance(:,1) +
distance(:,2));
```

```
           % Input values calculated here
            % Get series SD,SV
            SD = (Pi1([1:tPast],1:2)-
Pj1([1:tPast],1:2)).^2;
            SD = sqrt(SD(:,1)+SD(:,2));
            size(SD);
            SV = (Pi1([1:tPast],3:4)-
Pj1([1:tPast],3:4)).^2;
            SV = sqrt(SV(:,1)+SV(:,2));


               totaltrainIn = [totaltrainIn,...
                  [SD;SV]];
               totaltrainOut =
[totaltrainOut,...
                  distance];
               %is=flipdim(is,1);

sd_amass_totalout=[sd_amass_totalout,i
s'];
%              end %if

         end % for jter
      end % for iter
   end % if
end % while

%for getting the OUTPUT of Seperation
Distance in 2 files
%_testOut and _trainOut
d = size(totaltrainOut)
d1 = size(totaltrainIn)
da=round((d(2)*80)/100);
fprintf(1,'\noriginal da=%d',da);
dataSize = da;
if(da>10000)
  da=10000;
end
fprintf(1,'\nnew da=%d',da);

trainOut=totaltrainOut(:,1:da);
sd_amass_trainOut=sd_amass_totalout(:,
1:da);
%
%dlmwrite(strcat(fileout,'_trainOut.csv'),
trainOut,',')
```

```
%
testOut=totaltrainOut(:,da+1:d(2));
sd_amass_testOut=sd_amass_totalout(:,
da+1:d(2));
%
%dlmwrite(strcat(fileout,'_testOut.csv'),t
estOut,',')
%
%******************************
************************
%
% %for getting the INPUT of SD;SV;ST
in 2 files
% %_testIn and _trainIn
%
trainIn=totaltrainIn(:,1:da);

%******************************
************************


epochs=1;
times=70;
lRange = minmax(trainIn);
lLayerNumber = [10,15];
lLayerType =
{'purelin','purelin','purelin'};
lTrainType = 'trainlm';%'trainlm';
%net = newff(lRange, lLayerNumber,
lLayerType, lTrainType);
net = newff(lRange, lLayerNumber,
lLayerType, lTrainType);
[ae,nmse,pc,amass_nn_ae,amass_nn_nm
se,amass_nn_pc] =
testnet_v2_new(net,epochs,times,trainIn,
trainOut,testIn,testOut,sd_amass_trainO
ut,sd_amass_testOut)


% Close file
fclose(lFileHeader.fid);

% SubFunction to calculate the general
direction
%returns 1->N 2->E 3->S 4->W
function gen_dir = position(x, y)

% Finds N and E
if(x >= 0 & y >= 0)
  if(abs(x) >= abs(y))
    gen_dir=2;
  else
    gen_dir=1;
  end %if
end %if

% Finds S and E
if(x>=0 & y<0)
  if(abs(x) >= abs(y))
    gen_dir=2;
  else
    gen_dir=3;
  end%if
end%if

% Finds N and W
if(x<0 & y>=0)
  if(abs(x) >= abs(y))
    gen_dir=4;
  else
    gen_dir=1;
  end%if
end%if

% Finds S and W
if(x<0 & y<0)
  if(abs(x) >= abs(y))
    gen_dir =4;
  else
    gen_dir=3;
  end
end
%end function
```