ANOMALY DETECTION IN AIRCRAFT PERFORMANCE DATA

by

Syam Kiran Anvardh Nanduri

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

| | |
|---|---|
| _____ | Dr. Gheorghe Tecuci, Thesis Director |
| _____ | Dr. Lance Sherry, Committee Member |
| _____ | Dr. Jie Xu, Committee Member |
| _____ | Dr. Sanjeev Setia, Department Chair |
| _____ | Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering |

Date: _____ Semester 2015
George Mason University
Fairfax, VA

Anomaly Detection in Aircraft Performance Data

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Syam Kiran Anvardh Nanduri
Bachelor of Technology
Jawaharlal Nehru Technological University, 2011

Director: Dr. Gheorghe Tecuci, Professor
Department of Computer Science

Semester 2015
George Mason University
Fairfax, VA

# Dedication

I dedicate this thesis at the Lotus Feet of my master Paramahansa Yogananda, whose unconditional love and blessings define what I am today.

# Acknowledgments

I would like to thank Dr. Tecuci for his support and guidance throughout this research. I am very grateful to Dr. Sherry whose constant motivation and insights have helped me during this work. I would like to extend my thanks to Dr. Xu for his valuable feedback and timely advice on various aspects of the research. I express my deepest gratitude for my parents and my brother who have always encouraged me to follow my heart and pursue my interests.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

ANOMALY DETECTION IN AIRCRAFT PERFORMANCE DATA
Syam Kiran Anvardh Nanduri
George Mason University, 2015
Thesis Director: Dr. Gheorghe Tecuci

Detecting anomalous behavior in aircraft both during and after the flight is very important and it is of high interest for aviation safety agencies as well as airliners to ensure safe, efficient and environmentally clean flight operations. In this thesis, we study the capability of artificial neural networks in learning to identify anomalous behaviors in archived multivariate time series aircraft data and also in online data streams. The data is collected from flying 500 approaches to San Francisco International Airport on Boeing 777-200 ER aircraft using the X-Plane aircraft simulation. We analyzed the performance of Deep Autoencoders, Recurrent Neural Networks with Long Short-Term Memory units and Recurrent Neural Networks with Gate Recurrent Neural Networks units on the archived timeseries data. Once trained, the Recurrent Neural Network based algorithms can be applied to either previously collected flight data for retrospective analysis (offline mode) or they can be deployed during the flight to detect the anomalies in real time and alert the crew members (online mode). The performance of these algorithms is compared against MKAD, a Support Vector Machine based algorithm developed at NASA. These algorithms detected the anomalous flight types which MKAD was able to detect and also other anomalous flight types which MKAD was not able to detect.

Experiments were conducted using various parameters combinations for MKAD to see how the resolution of Symbolic Aggregate Approximation encoding and size of alphabet set impact its performance. Similarly, various architectures of autoencoders and recurrent neural networks were designed and their performance was evaluated in terms of precision, recall and $F1$ score. Experimental results show that recurrent neural networks outperformed all other models in overall performance.

# Chapter 1: Introduction

Anomaly detection in aircraft performance data plays an important role in identifying abnormal flight operations which differ from normal flight operations. Analyzing the causal factors which resulted an abnormal flight is significant for understanding the aircraft behavior under special circumstances. It also helps in assessing the human factors related issues, thus facilitating a robust framework to identify and analyze abnormal flight performances and take necessary actions in a proactive manner to avoid or mitigate undesirable similar future events. Machine learning and data mining techniques have been used for solving anomaly detection problem in several domains. Previous algorithms for detecting anomalies in aircraft performance data include SVM based Multiple Kernel Anomaly Detection (MKAD) developed by NASA [1][2] and clustering based Anomaly Detection (ClusterAD) algorithm developed at MIT [3]. In this thesis, we explore the capability of Artificial Neural Networks in detecting anomalies in multivariate, simulated timeseries aircraft performance data. Specifically, we study the performance of autoencoders and two variants of Recurrent Neural Networks, one based on Long Short Term Memory units (LSTM) and other based on Gated Recurrent Units (GRU) in identifying anomalies by training them in a semi-supervised fashion. During training, the models are presented with non-anomalous examples (negative class) and are expected to learn about normal cases. The performance of the trained model is evaluated by their inability to reconstruct the negative examples measured by Root Mean Squared Error (RMSE) values. During testing when presented with other class of anomalous examples (positive class), the models shall output high reconstruction error values indicating the example as an anomaly. We have used MKAD algorithm as the baseline to compare the performance of proposed models.

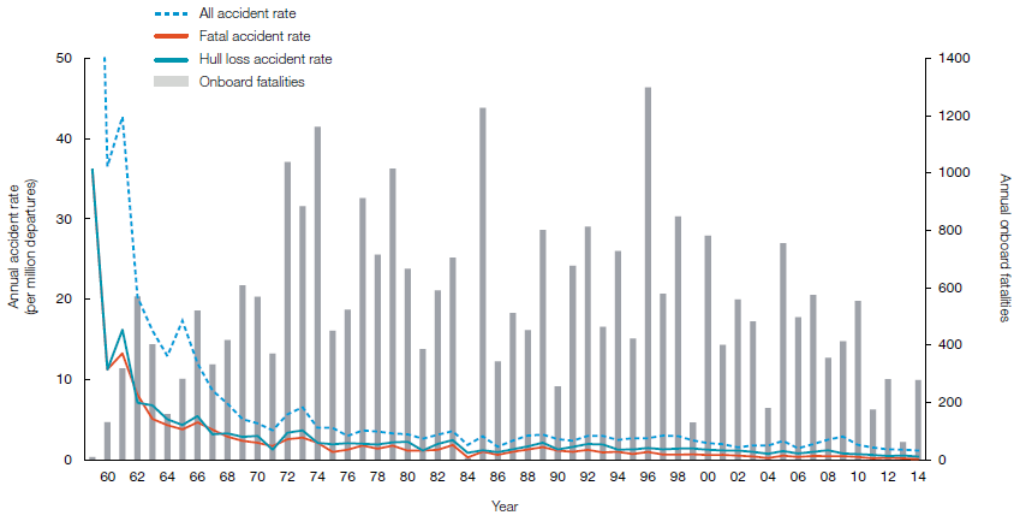



Figure 1.1: Annual Accident Rate per million departures (Source: Boeing, Statistical Summary of Commercial Jet Airplane Accidents Worldwide Operations 1959-2014)

## 1.1 Need for Anomaly Detection in Aviation

Aviation safety and regulatory agencies like International Civil Aviation Organization (ICAO) and National Transport Safety Board (NTSB) remain concerned about the maintanence of aviation safety standards for civilian aircraft. In the United States, the Federal Aviation Administration (FAA), which oversees safety and efficiency of air transport also assumes the responsibility of regulating civil aviation to promote safety. It encourages development of new aviation technology to control aircraft noise and other environmental effects of civil aviation. Because of all the efforts and recent technological advancements along with airliner operators' compliance to safety regulations, the fatal aircraft accident rate has significantly reduced. Figure 1.1 shows the annual accident rate of commercial jet aircraft for worldwide operations collected by Boeing.

But air transportation has experienced an exponential increase since 1970s despite some setbacks in between due to financial instabilities and geopolitical conflicts. The changes in the structure of air transportation networks through the introduction of hubs has led to less direct connections and resulted in longer distances being flown by aircraft. Figure

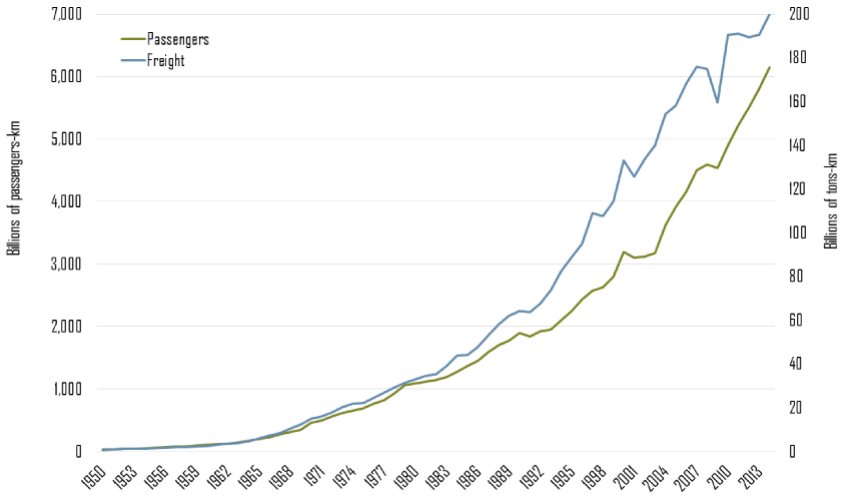



Figure 1.2: World air travel and air freight carried, 1950-2014 (source: Department of Global Studies & Geography, Hofstra University, NY)

1.2 shows the increase in the average distances traveled in air by passengers in terms of passengers-km and freight in tons-km. Thus, given the projected growth in air travel in coming years and resulting exponential increase in number of aircraft requires the aviation agencies to constantly improve the quality and safety standards of their operations, because even at decreasing accident rate the *number* of accidents may significantly increase.

In an effort to proactively identify a potential issue at individual aircraft level and at the fleet level as well, the airlines have to closely monitor the operations and take timely decisions to prevent undesirable events. *Anomaly Detection* in recorded aircraft performance data would certainly help in identification of hidden, unknown abnormalities and thus providing valuable information ahead of time.

## 1.2 Research Objective

Although Artificial Neural Networks have been successfully used in anomaly detection in various domains, much less research has been conducted on using them for anomaly detection in aircraft data. This thesis aims to:

1. Investigate the applicability of Autoencoder Neural Networks for anomaly detection in aircraft performance data.

2. Investigate the applicability of Long Short Term Memory based Recurrent Neural Networks and Gated Recurrent Units based Recurrent Neural Networks in detecting anomalies in aircraft performance data.

3. Evaluate the performance of Autoencoders and Recurrent Neural Networks in detecting anomalies in aircraft performance data.

## 1.3 Contribution to Literature

The Flight Operational Quality Assurance (FOQA) data is highly restricted and thus not easily available for research and academic purposes. But FOQA-like data can be generated through an aircraft simulation tool called X-Plane. We have developed a *plugin* for X-Plane which automatically generates aircraft performance data for various flights while approaching a runway during descent. This automatic data generation capability can be very useful for research community for creating huge amounts of FOQA-like data. A methodology for training Auto encoders on aircraft performance data for anomaly detection is thoroughly discussed and results comparing MKAD and autoencoders are presented. Though MKAD and autoencoders can be used to detect anomalies in archived data, applying them to detect real time anomalies in online data streams is difficult. To address this issue, we have studied and discussed the performance of LSTM and GRU based Recurrent Neural Networks (RNNs) in detecting anomalies in real-time online data streams.

## 1.4 Thesis Outline

The rest of the thesis is organized as follows. In chapter 2, we discuss the previous research conducted to address the anomaly detection problem. Chapter 3 discusses the design of Approach Data Generator Plugin (adgPlugin) for X-Plane software and also discusses how

the normal and anomalous data is collected using it. The characteristics of the normal and anomalous data is presented. Chapter 4 discusses about theoritical aspects of Autoencoders and Recurrent Neural Networks. Chapter 5 compares the performance of MKAD algorithm and autoencoder based anomaly detection. Also, we present the performance of Recurrent Neural Network variants. In Chapter 6, we discuss the work to be done in future and then conclude.

# Chapter 2: Literature Review

Extensive research has been done to solve anomaly detection problem in many domains and a wide gamut of methodologies have emerged based on variety of underlying techniques. This chapter gives a brief overview of past research in anomaly detection. Specifically, we will explore how these numerous techniques have been categorized based on some key factors, then we will discuss how anomaly detection problem is applied to time series data, followed by different model based anomaly detection techniques and conclude this chapter with an overview of existing methodologies for aviation data.

## 2.1 Categorization of Anomaly Detection Techniques

Chandola et al. [4] provided a comprehensive overview of various existing anomaly detection techniques by identifying key characteristics of techniques and classifying them into different categories. They also proposed a taxonomy to categorize any anomaly detection technique based on the type of input data, type of supervision, type of outlier and output of the anomaly detection algorithms. Based on type of *input data* the underlying strategy for anomaly detection techniques can vary. In the simplest case, the data may have represent a single *variable* and the actual data would be different values recorded for that variable. Since the data has only one variable, it is called *univariate* data. Instead, if each data sample records values of multiple variables it is called *multivariate data*. The variables in data instances are sometimes refered to as attributes or features. Also the values of each data instance can be binary, categorical or continuous. If all instances are independent of one another without any consistent pattern, then such data is treated as *point data*. If the data instances follow a sequential pattern occuring one after the other then it is treated as *sequential data*. If the sequence considers time as reference then it is Timeseries data or if

spatial orientation (e.g., latitude longitude values) is considered as reference, it can be treated as spatial data. Data which considers both space and time as reference then such instances constitute a spatial-temporal data. Based on *type of learning*, various machine learning and statistical learning techniques are categorized. If the training data contain labelled instances with both positive class (anomalous) and negative class (non-anomalous) then such learning is considered as supervised. Traditional decision tree classifiers or multilayer perceptrons are typical examples. Whereas if the training data consists of instances belonging to only one class, either positive or negative, and the model is trained with instances of the available class, then it is a semisupervised model. Generally since normal instances are easily available, semisupervised models are trained based on normal data and tested on data with both normal and anomalous data. Examples include One Class SVMs, autoencoders, etc. We can categorize the anomaly detection techniques based on *types of outliers* they are capable of detecting. Some techniques are good at detecting data whose features are inconsistent with features of other instances in a given data set. Such anomalies are called Instantaneous anomalies. Some techniques are good at detecting *contextual anomalies*. These are the instances which are considered abnormal because their presence at that point among given neighbors is questionable. This requires a notion of context, which has to be defined as part of problem definition. Generally it is defined by the neighborhood of a given instance. Chandola et al. also note that these outliers satisfy two key properties: (i) The underlying data has spatial/sequential nature and each data instance consists of two types of features, viz. contextual attributes and behavioral attributes. The contextual attributes define the context of the instance. In spatial data, latitude longitude values can be contextual data. And in a timeseries data, time can be a contextual attribute which defines the context of the instance with respect to other instances in the data set. On the other hand, the behavioral attributes define the noncontextual properties of the data instance. For example in aircraft performance data, latitude and longitude values can be contextual attributes and the speed, altitude and thrust can be behavioral attributes. (ii) Behavioral attributes are used to determine anomalous behavior within a specific context. An instance considered

as a contextual outlier based on its behavioral attributes in a given context, but another identical instance (behavioral attributes) can be treated as normal in a different context. The other type of anomalies are *collective anomalies*. These instances are not abnormal in themselves. Their collective occurance in the given substructure of the data sequence makes the entire substructure questionable.

## 2.2 Instantaneous Anomaly Detection Techniques

In this subsection we discuss various methods based on how the model is trained using known data samples for detecting outliers or anomalies in unseen test samples.

### 2.2.1 Statistical Anomaly Detection

Statistical anomaly detection techniques can be treated as model based techniques, where during the training phase, a probabilistic model is learned by estimating the probability distribution of the training data and during testing phase the learned model is used to compare the test data instances to determine if the instance is an anomaly or normal. In semi-supervised statistical based techniques, the probability distribution is estimated for either normal instances or anomalous instances depending on the availability of the data. In unsupervised statistical techniques, probability distribution is estimated for all data instances and majority class of the observations, which fit to the model are treated as normal. Instances which fall in the low probability region are treated as anomalies or outliers by the model. Model Fitting has been a very important statistical analysis tool [4].

### 2.2.2 Classification Based Anomaly Detection

In classification based anomaly detection techniques the main idea is to train a classification model using the available labeled training data and use it to classify unseen test instances as either normal or anomalous. Thus classification based anomaly detection techniques also have two phases: training phase and testing phase. In *training phase* the model is trained on the labeled data samples and learns about learn all classes of data samples (supervised) or

learns the provided class (semisupervised). In *testing phase* the learned model is presented with unseen test samples and the model predicts the label of the test sample based on the decision boundary it has learned during training phase. Thus again depending on the availability of type of training data available, these models can be classified as follows:

**Supervised Classification** If the training data is available for all classes then the model learns a decision boundary for all classes. Then during testing when presented with an unseen instance, the model classifies the instance and assigns a class label. For supervised models solving anomaly detection problem the data should consist of both normal and anomalous classes with labels. The potential drawback with these techniques is that they may fail to detect unknown and emerging anomalies. Augisteijn et al.[5] compared the performance of the backpropagation neural network with Probabilistic Neural Networks (PNN) architecture to detect novel patterns in remotely sensed imagery. They explored the applicability of different neural network architectures and concluded that PNN shows superior performance as an overall classifier when compared to backprop and also is able to identify novel patterns in the data. Sykacek [6] discussed the problem of outlier detection for neural networks trained by Bayesian inference. It is argued that marginalization is not a good method to get moderated probabilities for classes in outlying regions and the reason why marginalization fails to indicate outliers is analysed and an alternative measure called Equivalent Error Bars, a more reliable indicator for outliers is proposed. Visualizations of an artificial classification problem showed that the equivalent error bar of the classier is a more reliable method for outlier detection than its marginalized output. Vasconcelos et al.[7] investigated the reliability of three variations of Feed Forward Neural Networks, viz., Multi Layer Perceptron (MLP) , Gaussian MLP (GMLP are MLPs with Gaussian as activation function) and Radial Basis Function Network (RBF) in rejection of patterns not belonging to the defined training classes. Networks with different activation functions and propagation rules construct the decision regions in the pattern space differently and affect the network's performance in dealing with anomalous information. A modification to the standard MLP structure is described to enhance its reliability to detect anomalies.

Support Vector Machines (SVM) proposed by Vapnik [8] have been extensively used in various domains for classification based anomaly detection techniques. Mukkamalla et al.[9] have used SVMs for for the task of Intrusion Detection in networks. They have formulated the problem as binary classification task and compared the performance of SVMs and Neural Networks, and found that both give high accuracies and SVMs train in notably shorter durations. They concluded that whether to use SVMs or neural networks in implementing an intrusion detector depends on anomaly or misuse that is under watch, as well as other security policy requirements.

**Semisupervised Classification** But in some cases only one class of labeled data may be available. In such cases, during training phase the model learns about available class of data instances and during testing phase, if the model accepts the instance then it is considered belonging to the same class of instances on which the model is trained, otherwise if the test instance is rejected it is considered as sample of the other class. Since in any domain the normal data is easily available relative to anomalous data samples, the semisupervised techniques which use normal (negative) samples for training the models are more predominant than models trained on abnormal samples. One of the drawbacks of semisupervised techniques is that they may exhibit high false alarm rate as the previously unseen (yet normal) data records may be falsely categorized as anomalies.

**Unsupervised Classification** If the available data is unlabeled and it generally consists of a large number of positive sample and small number of negative data samples anomaly detection methods make use of unsupervised learning techniques like density estimation or clustering to segregate both classes. Steinwart et al.[10] discussed how to use SVMs for unsupervised learning of anomalies from unlabeled data. They define anomalies by saying that anomalies are not concentrated , which leads to the problem of finding level sets for the data generating density. The learning problem is treated as a binary classification problem and compare the corresponding classification risk with the standard performance measure for the density level problem. They propose that the empirical classification risk can serve as an

empirical performance measure for the anomaly detection problem which allows comparison of different anomaly detection algorithms empirically, with the help of a test set. By the above interpretation they give a strong justification for the well known heuristic of artificially sampling labeled samples, provided that the sampling plan is well chosen. And it enables them to propose a SVM for anomaly detection for which universal consistency can be easily established. They compare this SVM with other commonly used methods including the standard One Class SVMs.

## 2.3 Anomaly Detection in Time Series Data

Data collected in various domains is in form of sequences or timeseries. For example, the aircraft performance data collected by Flight Data Recorder consists of data from various sensors and instruments at regular time intervals. In financial organizations, stock prices of various companies are recorded at regular time intervals. Similarly, sequential data is collected in various other domains like network intrusion monitoring, medical health monitoring systems, fraud detection systems and so on. Depending on number of variables or parameters in the data it can be classified as either univariate time series data or multivariate time series data. Examples of univariate data can be, the heartbeat data from Electrocardiogram (ECG) which records only one variable the heartbeat rate of the patient.

### 2.3.1 Modelling Time Series Data using Deep Neural Networks (DNN)

Recently a lot of active research is concentrated in the domains of speech recognition, Natural Language Processing and signal processing which demonstrate the effectiveness of Deep Neural Networks in solving various problems. A deep neural network (DNN) is a feedforward, artificial neural network that has more than one layer of hidden units between its inputs and its outputs. All these domains work with timeseries data and since this thesis deals with similar timeseries data, it is useful to present brief overview of some of these methodologies.

### 2.3.2 Neural Networks Based Anomaly Detection in Time Series Data

Using Neural Networks, especially Recurrent neural networks is a currently active area of research for solving problem of anomaly detection in time series data. For example, Malhotra et al.[11] have used Long Short Term Memory (LSTM) networks and shown that these are very effective in learning sequences containing longer term patterns of unknown length, due to their ability to maintain long term memory. Stacking recurrent hidden layers in such networks also enabled learning of higher level temporal features. They have used stacked LSTM networks for anomaly/fault detection in various univariate time series datasets. A network trained on nonanomalous data can be used as a predictor over a number of time steps. The resulting prediction errors are modeled as a multivariate Gaussian distribution, which is used to assess the likelihood of anomalous behavior. Staudemeyer [12] applied Long Short Term Memory (LSTM) based Recurrent Neural Networks to Intrusion Detection problem as supervised classification. They trained long short-term memory (LSTM) recurrent neural networks with the training data and to identify suitable LSTM RNN network parameters and structure we experimented with various neural network architectures and found optimal design that works best for the dataset. They also compared the performance of LSTM RNN networks trained with all features against RNN networks trained on extracted minimal feature sets. Performance is measured in terms of mean-squared error, confusion matrix, accuracy, ROC curve and the corresponding AUC values.

## 2.4 Anomaly Detection for Aviation Data

### 2.4.1 Previous Algorithms for analyzing Aircraft Data

**Morning Report** [13][14] is a clustering based algorithm that detects atypical flights over a set of aircraft and identifies the contributing anomalous parameters and flight phases in which anomalies have been detected. The algorithm calculates statistical signatures across the parameters of a given flight and clusters the flights based on the multivariate signatures. Similar flights are grouped together and atypical flights are considered to be far away from a

cluster and therefore have higher scores. Based on the Mahalanobis distance from centroids of the multivariate cluster the distribution of the anomaly scores are determined. The results provide both the degree of the anomalous flight along with the contributing parameters, which can be useful for the analysts.

**Orca** by Bay et al.[15] uses k-nearest neighbor based algorithm for detecting anomalies in both continuous and discrete (binary format) data in vector space. For continuous data, Orca takes a nominal reference data set and calculates the nearest neighbors using euclidean distance to all test points in the original vector space. For binary data points the hamming distance is used. Orca is a nested loop structure in complemented with randomization and simple pruning rule. Algorithm is tested on very large multidimensional data and it is observed to be performing in almost linear time with respect to input data. Authors claim that Pruning used in the algorithm helps in achieving near linear time performance with high dimensional data. Furthermore they observe that for an average case analysis under certain conditions, the time to process nonoutliers, which are the large majority of points, does not depend on the size of the data set. If one looks at the local neighborhood and finds that the test points are relatively close, then the examples are considered normal or else unusual. One major drawback of this algorithm is that each data point is scored independently and therefore anomalies in the temporal domain are undetectable.

**Inductive Monitoring System (IMS)** by Iverson [16], automatically builds knowledge bases from nominal data and uses clustering to group sets of consistent parameter values found in the training data in an unsupervised fashion. System parameter values are arranged as a vector (an ordered list of parameters). These vectors define the points in Ndimensional space that are grouped by the IMS clustering algorithm. To use the cluster knowledge base during monitoring phase, IMS formats the real time data into vectors and queries the knowledge base to locate the cluster that is closest to each input vector. The fastest monitoring schemes require that the input data vectors be contained inside at least one of the knowledge base clusters and also all parameter values are within the ranges specified

by the cluster limits. This eliminates the need to perform distance calculations. A more informative monitoring technique will locate the cluster in the monitoring knowledge base that is closest to the input vector, and report the distance of that vector from the cluster. This will give the user a sense of degree to which the system behavior is anomalous. If a vector is not contained within a cluster, the distance between the vector and the closest point in the bounding rectangle defined by the nearest cluster is reported as anomalous score. However, IMS evaluates each point independently and therefore has the same drawback as orca of not being able to detect anomalies in the temporal domain.

**Sequence Miner** Budalakoti et al.[17] developed SequenceMiner to address the problem of detecting and describing anomalies in large sets of high dimensional symbol sequences such as recordings of switch sensors in the cockpits of commercial aircraft. SequenceMiner also uses an unsupervised clustering algorithm to cluster the sequences using the normalized longest common subsequence (LCS) as a similarity metric. Once the clusters are defined anomalies can be detected using the LCS as the distance measure. In this context anomalies are determined to have low similarities between the clusters of other sequences and are defined to be far away from a cluster. In order find the degree of anomaly, SequenceMiner applies a genetic algorithm to modify the anomalous sequence to draw it closer to the cluster and keeps track of the changes made to the sequence. It reports back the missing and extra symbols giving the user some context of the anomaly. Since SequenceMiner focuses on the sequential nature of the anomalies it can find anomalies that other algorithms such as Orca and IMS are unable to detect, however it is ineffective at handling continuous parameters without somehow drastically changing the nature of the data [2].

### 2.4.2 State-of-the-art Anomaly Detection Algorithms

In this subsection we discuss three current state-of-the-art algorithms for anomaly detection in aircraft performance data. The Multiple Kernel Based Anomaly Detection (MKAD) [1][2] is developed at NASA, the Cluster based Anomaly Detection (ClusterAD) [3] developed at ICAT, MIT and the Exceedance based algorithm (FAA, 2004) which is being used by

the airlines and it is the current industry standard algorithm. This thesis uses MKAD as the baseline algorithm and we compare its performance in detecting the anomalous cases presented in chapter 3 against the performance of new proposed algorithms discussed in chapter 4 and chapter 5.

**Multiple Kernel Based Anomaly Detection (MKAD)**

MKAD is motivated from the ability of Multiple Kernel Learning MKL [18][19] to simultaneously incorporate kernel functions of different types. Thus the ability to incorporate both discrete and continuous sequences simultaneously using different kernels for anomaly detection is the core idea behind MKAD. Consider two data points (feature vectors) $\vec{x}_i$, $\vec{x}_j$ each having equal number of symbols given by $l_x$, then the resultant kernel used in MKAD is a convex combination of two seperate kernels over multiple features and takes the form of,

$$k(\vec{x}_i, \vec{x}_j) = \eta K_d(\vec{x}_i, \vec{x}_j) + (1 - \eta) K_c(\vec{x}_i, \vec{x}_j) \tag{2.1}$$

where,

1. $K_d$ is a kernel over discrete sequences is based on the normalized Longest Common Subsequence (nLCS) measure [17] given by,

$$K_d(\vec{x}_i, \vec{x}_j) = \frac{|LCS(\vec{x}_i, \vec{x}_j)|}{\sqrt{l_{\vec{x}_i} l_{\vec{x}_j}}} \tag{2.2}$$

   If $X$, $Y$ and $Z$ are three sequences, $Z$ is called a *subsequence* of $X$ if removing some symbols from $X$ produces $Z$. $Z$ is a common subsequence of $X$ and $Y$, if $Z$ is a subsequence of both $X$ and $Y$. The longest such subsequence of $X$ and $Y$ is called the Longest Common Subsequence and is denoted by LCS($X$,$Y$) and its length is denoted by |LCS($X$,$Y$)|.

2. $K_c$ is a kernel over continuous sequences. It makes use of the Symbolic Aggregate

approXimation (SAX) representation [20]. SAX is a dimensionality reduction technique which compresses a feature vector $\vec{x}_i$ with $m$ continuous variables and $n$ values for each variable (since each variable sampled $n$ times) into a vector with $m$ variables with only $r$ values per variable, where $r \leq n$. Each of these $r$ values represent the mean of that variable in $r$ consecutive time windows. Thus the length of the time window or window size determines the resolution of the compressed data. Thus a larger window size results in more compressed encoding on the input data (low resolution output) and smaller window sizes results in less encoded data (high resolution output) but with relatively more dimensions than with larger window sizes. Consider any variable $a$ in data point $\vec{x}_i$. This data point contains the values of $a$ (along with other variables) observed and sampled at regular time intervals. Now, let the window size be $w$, given by $\lfloor \frac{n}{r} \rfloor$, then at time interval $t$, the mean for $w$ contiguous sample values of variable $a$,

$$\bar{\vec{x}}_{iat} = \frac{\sum_{k=w(t-1)+1}^{wt} \vec{x}_{iak}}{w} \tag{2.3}$$

where $\vec{x}_{iak}$ is the $k$th time point for variable $a$ of data point $\vec{x}_i$. Once the values of all variables are compressed by calculating means, a normal distribution is fit to all the training data for each variable. A value for *number of bins* $c_a$ is chosen which becomes the *alphabet set size*. Then equiprobable bins are found with breakpoints $\beta_{a,1}, \beta_{a,2}, \ldots, \beta_{a,c_a-1}$ such that area under normal density function is $\frac{1}{c_a}$ for each $x \leq \beta_{a,1}$, $x \in [\beta_{a,k}, \beta_{a,k+1}] \forall k \in 1, 2, \ldots, c_a - 2$ and for $x \geq \beta_{a,c_a-1}$. All equiprobable bins are assigned a label chosen from alphabet set and each of the $\bar{\vec{x}}_{iat}$ is replaced with corresponding labels. $K_c(\vec{x}_i, \vec{x}_j)$ is inversely proportional to the distance between the SAX encodings of $\vec{x}_i$ and $\vec{x}_j$ given by,

$$K_c(\vec{x}_i, \vec{x}_j) = \frac{1}{|LCS(SAX(\vec{x}_i), SAX(\vec{x}_j))|} \tag{2.4}$$

3. $\eta$ is an adjustable parameter which controls the contribution of discrete kernel and continuous kernel. Default value of 0.5 results in equal contribution from both kernels.

One-class Support Vector Machine [21][22] is used as the anomaly detection method in MKAD. The One-class SVM, a semi-supervised method, constructs an optimal hyperplane in the high dimensional feature space by maximizing the margin between the origin and the hyperplane. This is accomplished by solving an optimization problem [22] whose dual form is given as,

$$\begin{aligned} &\text{minimize } Q = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j k(x_i, x_j) \\ &\text{subject to } 0 \leq \alpha_i \leq \frac{1}{lv}, \sum_i \alpha_i = 1, \rho \geq 0, v \in [0,1] \end{aligned} \tag{2.5}$$

where $\alpha_i$ is Lagrange multiplier, $v$ is adjustable parameter gives upper bound on training error and lower bound on the fraction of training points that are support vectors. Solving this optimization problem yields atleast $vl$ training points whose Lagrange multipliers are greater than zero and these data points are called *support vectors*. $\rho$ is a bias term and $k$ is the kernel given in equation 2.1. These Support vectors $x_i : i \in [l], \alpha_i > 0$ are either marginal $\zeta_m = i : 0 < \alpha_i < 1$ or non-marginal $\zeta_{nm} = i : \alpha_i = 1$. Once support vectors $\vec{\alpha}$ are obtained, the following decision function given by following equation and is used to determine if a test data point is normal or anomalous. Data points with negative values are classified as anomalous and points with positive values are treated as normal.

$$f(\vec{x}_j) = sign(\sum_{i \in \zeta_m} \alpha_i k(\vec{x}_i, \vec{x}_j) + \sum_{i \in \zeta_n m} \alpha_i k(\vec{x}_i, \vec{x}_j - \rho) \tag{2.6}$$

### 2.4.3 Clustering based Anomaly Detection (ClusterAD)

ClusterAD [3] initially converts the raw data into time series data. In order to map data into comparable vectors in the high dimensional space, these time series data from different flights are anchored by a specific event to make temporal patterns comparable. Then every

flight parameter is sampled at fixed intervals by time, distance or other reference from the reference event. All sampled values are arranged to form a high dimensional vector for each flight in the following form:

$$[x^1_{t_1}, x^1_{t_2}, \ldots, x^1_{t_n}, \ldots, x^m_{t_1}, x^m_{t_2}, \ldots, x^m_{t_n}]$$

where $x_{itj}$ is the value of the $i$th flight parameter at sample time $t_j$, $m$ is the number of flight parameters and $n$ is the number of samples for every flight parameter. Thus the total dimensionality of every vector is $m$x$n$. Each dimension represents the value of a flight parameter at a particular time. PCA is applied on this high dimensional feature vectors to reduce the number of dimensions. The similarity between flights can be measured by the Euclidian distance between these low dimensional vectors.

### 2.4.4 Exceedance based Method

Exceedance detection is the traditional flight data analysis method that is widely used in the airline industry. It involves checking if particular flight parameters exceed the predefined limits under certain conditions. Domain experts set the list of flight parameters to be monitored and their limits. The list of rules is always chosen to match with the airlines standard operating procedures. For example events such as the pitch at takeoff, the speed at takeoff climb, the time of flap retraction can be monitored. Therefore, this approach requires a predefined list of of key parameters under certain operational conditions and also require precisely defined limits for each parameter. Though many of the known, predefined safety issues can be accurately examined by Exceedance Detection, the unknown and unspecified conditions cannot be undetected.

### 2.4.5 Limitations of Current Methods

Following limitations have been observed for both MKAD and ClusterAD,

**Need for Dimensionality Reduction** Both MKAD and ClusterAD convert sequential data of entire flight into a high dimensional timeseries data. They rely on dimensionality

reduction techniques to map the high dimensional data to a low dimensional feature space. For example, MKAD uses Symbolic Aggregate Approximation (SAX) and ClusterAD uses Principal Component Analysis (PCA) as dimensionality reduction techniques during data preprocessing step in their methodologies. Moreover, ClusterAD requires the feature vectors of multiple flights to be aligned with respect to a specific event for meaningful comparisons. Thus it may be difficult to use these algorithms in real time anomaly detection.

**Poor sensitivity towards short term anomalies** Past studies by [3] found that both MKAD and ClusterAD are not sensitive to anomalous patterns which occur for short durations. One of the reasons could be that, due to data compression during dimensionality reduction some of these nuances would have been lost. The Recurrent Neural Networks based on LSTM and GRU units (presented in chapter 4) do not suffer from above limitations as RNNs are by definition capable of handling multivariate sequential data without any modifications and treat it as timeseries data.

**Inability to detect anomalies in Latent Features** Li et al. [3] discuss that both MKAD and ClusterAD cannot detect anomalies in features that are not explicitly present in the feature vector, although these latent features are derivable from existing features. For example, they find that both algorithms failed to detect *abnormal pitch rate during landing* as the pitch rate was not part of the feature vector. The dataset included pitch value as one of the features. In this thesis, we study the performance of autoencoders and RNNs in detecting such anomalies in latent features.

## 2.5 Anomaly Detection in This Research

In this thesis, we develop neural networks based *semisupervised* models and train these models on the normal non-anomalous instances. The input data is *multivariate time series* flight performance data and the aim of our models is to detect contextual, pattern based and instantaneous anomalies in the test data. This is summarized in Table 2.1

Table 2.1: Characteristics of Anomaly Detection Techniques in this Research

| **Property** | **Value** |
|---|---|
| Type of Training | Regression |
| Mode of Training | Semi-supervised |
| Type of Data | Multivariate time-series |
| Type of Anomalies | Contextual, Pattern-based, Instantaneous |

In the next chapter we will discuss how the data used in this research is collected and present some statistical properties of the data.

# Chapter 3: Aircraft Performance Data Generation

The Distributed National FOQA (Flight Operations Quality Assurance) Archive (DNFA) contains data from flight data recorders of over two million flights and covers over 10 major carriers. NASA established this archive in 2007 and data from most of the major carriers in the US is collected. Typical FOQA parameters consist of both continuous and discrete (categorical) data from the avionics, propulsion system, control surfaces, landing gear, the cockpit switch positions, and other critical systems. These sets can have up to 500 parameters and are sampled at 1 Hz. Due to proprietary and legal issues, these data are not shared between airlines or directly with the government. And thus it is not available for public for research purposes. Development of state-of-the-art aircraft simulation software like X-Plane[1] which provide their own SDKs, allows development of external plugins to tweak the underlying flight model for modified performance or to customize the whole simulation to suit the needs. Interestingly, this feature allows us to automate the generation of aircraft performance data which closely resembles FOQA data.

## 3.1 Simulation Setup for Approach Data Collection

For the purpose of this research, we have collected performance data of Boeing 777-200 ER aircraft in XPlane for 500 approaches into San Francisco (KSFO) airport. A C++ plugin called adgPlugin (Approach Data Generator Plugin) has been developed which automates the approach phase of the flight into any desired airport. Though it allows us to collect huge number of aircraft parameters, we restricted ourselves to 20 most important performance parameters during these 500 flights. It is important to note that, these 500 flights include both normal and anomalous. There are nearly 485 normal flights and 15 anomalous flights.

[1] developed by Laminar Research http://www.xplane.com

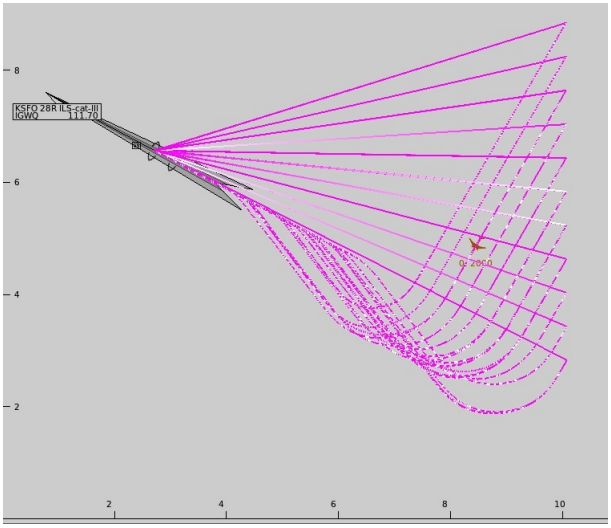



Figure 3.1: Lateral Flight Paths for multiple approaches into KSFO Runway 28R ILS, CAT III approach (ILS Frequency 111.7 Hz) by adgPlugin on Boeing 777-200 ER

While the adgPlugin automates the normal flights according to the predefined trajectory, to simulate anomalous flights we have to manually intervene and perform some actions (e.g., toggle a switch, pull back throttles column) to model the anomaly. It is interesting to note that in normal flights there are the three kinds of stochastic variations as follows:

1. Initial point of Lateral Flight Path: From a predefined set of latitude longitude pairs the plugin chooses a pair for each new approach and initializes the aircraft at that position at a fixed altitude as shown in Figure 3.1. Because of this some of the flights show variations in their lateral paths while approaching the specified runway.

2. Fuel and weight: We begin the simulation with fuelled aircraft and during each approach the fuel is burned gradually. As of now the adgPlugin doesnt refuel after each flight. This makes the aircraft lighter and lighter during and after each approach. Because of this we can expect some inevitable variations in the normal data collected.

3. Wind and turbulence: The simulation enables to specify predefined wind speed or random amounts of wind speed during the flight. We have specified random amount of winds (between 0-8 kts) for some of the flights. Because of this change in wind patterns

we can observe some variations in the normal data recorded. Furthermore, this enables the data to be more realistic. Lastly, KSFO has 4 different runways available and our approaches include only two of them, either 28L or 28R. Approximately half of the flights approached runway 28L and while remaining approached runway 28R.

## 3.2 Details of Recorded Parameters

Though for adgPlugin hundreds of parameters are accessed and programmed, for assessing the performance of each flight we have recorded following 21 important parameters once every 2 seconds. Of these *continuous parameters* are Latitude, Longitude, Airspeed, Vertical Speed, Flap Position, Gear, Pitch, Altitude, Thrust, N1, LD, Pitch Target, Roll Target. The *discrete parameters* are AT, VNAV, Heading Mode, Pitch Mode, AP1, AP2, FD1, FD2. Time is considered as the reference during all our analysis. The details of these parameters is given in Table 3.1.

Table 3.1: Details of Performance Parameters Recorded by adgPlugin

| **Parameter(units)** | **Description** |
|---|---|
| **Continuous Parameters** | |
| Latitude(deg) | Current latitude position of aircraft |
| Longitude(deg) | Current longitude position of aircraft |
| Airspeed(knots) | Current speed of the aircraft |
| Vertical Speed(ft/min) | Rate at which aircraft is descending or ascending |
| Flap Position(deg) | Position of the flaps control surfaces on the aircraft wings |
| Gear(real) | Position of landing gear. 0 if fully retracted (up), 1 if completely extended (down) and anything in between during transition |
| Pitch(deg) | Current pitch of the aircraft |
| Altitude(feet) | The height of aircraft above sea level as given by altimeter |
| Thrust(pounds) | Forward force provided by aircraft engines |
| N1 (real) | The rotational speed of the low pressure spool also called fan speed of an engine given as percentage of rated speed |
| LD(real) | Lift to Drag ratio |
| Pitch Target | Target Pitch (along Lateral axis of aircraft) calculated by automation |
| Roll Target | Target Roll (along Longitudinal axis of aircraft) calculated by automation |
| **Discrete Parameters** | |
| AT, AP1, AP2 | Status of Auto Throttle, Auto pilot Buttons in cockpit. 1 for ON, 0 for OFF |
| VNAV | Status of VNAV button in cockpit. 1 for ON, 0 for OFF |
| Heading Mode | FMA heading mode used by auto pilot |
| Pitch Mode | FMA pitch mode used by auto pilot |
| FD1, FD2 | Positons of Flight Director switches in cockpit. 0 for ON, 1 for OFF |

## 3.3 Data Generation for Normal Flights

If not intervened adgPlugin generates normal flight sequences as predefined in the code. A normal flight approach is generated by algorithm 1:

**Algorithm 1** Automatic Approach Data Generation Algorithm

**procedure** PLUGINSTART
    InitDataRefs()  ▷ Reading Data from Xplane and for modifying XPlane Model
    Allocate Resources
    INITIALIZEAIRCRAFT
    RegisterFlightLoopCallback (pluginRuntimeCallback, 2.0, NULL)
**end procedure**
**procedure** INITIALIZEAIRCRAFT
    $\langle Lat, Long \rangle \leftarrow Rand(\{\langle Lat_i, Long_i \rangle\})$
    $CurrentAltitude, TargetAltitude \leftarrow 1800$ ft
    $CurrentSpeed, TargetSpeed \leftarrow 190$ kts
    $Heading \leftarrow 210^{\circ}$
    $Flaps \leftarrow 5^{\circ}$
    $Gear, Throttle \leftarrow 0$
    $AP1, AP2, AT \leftarrow 1$  ▷ Buttons set to ON state
    $FD1, FD2 \leftarrow 0$  ▷ Switches set to ON state
    $VNAV \leftarrow 0$  ▷ Button set to OFF state
    $FlightCount \leftarrow FlightCount + 1$  ▷ Increment the count once initialized
**end procedure**

Figure 3.2 shows how various continuous and discrete parameters vary with respect to altitude and time. Each unit on horizontal axis represents 2 seconds of time. Secondary vertical axis shows altitude in feet. It can be observed that at the beginning of the approach though the aircraft is programmed to *hold and maintain altitude* of 1800 ft, it climbs to 2000 ft followed by a rapid descent to 1600 ft and then reaches 1800 ft to maintain that altitude. This behavior of the simulation was attributed to the pitch up condition during flare at the end of previous flight. Because of the positive pitch at end of the flight, the simulation continues to maintain the attitude even if repositioned at intial approach point. Thus it climbs to 2000 ft before its descent because of lack of thrust. Since autothrottle and autopilot are engaged, the aircraft is brought back to designated target altitude of 1800 ft. Since this behavior was commonly observed in all flights, we have assumed it as normal behavior for the purpose of this study.

Figure 3.3 depicts how the altitude and target airspeed vary in a typical flight as the

```
19: procedure pluginRunTimeCallback
20:     Record Parameters                ▷ Record performance data from X plane Data Refs
21:     while (FMAPitchMode != FLARE) do
22:         if LocalizerArmed = False AND ApproachArmed = False then
23:             LocalizerButton ← 1                          ▷ Button set to ON state
24:             LocalizerArmed = True
25:         end if
26:         if RollMode = LOC then
27:             ApproachButton ← 1                           ▷ Button set to ON state
28:         end if
29:         if PitchMode = GS AND Flaps < 15° then
30:             TargetSpeed ← 170 kts
31:             Command ⟨FlapsDown⟩                          ▷ extend from 5°to15°
32:         end if
33:         if Gear = 0 AND RollMode = LOC AND PitchMode = GS
34: AND Flaps > 15°  then
35:             Gear ← 1                                     ▷ extend landing gear
36:             TargetSpeed ← 150 kts
37:         end if
38:         if AFDSMode = AP  then
39:             TargetSpeed ← 138kts
40:         end if
41:     end while
42:     InitializeAircraft          ▷ Reinitialize aircraft if it reaches FLARE mode
43: end procedure
44: procedure pluginStop
45:     UnregisterFlightLoopCallback(pluginRuntimeCallback, NULL)
46:     Release Resources
47: end procedure
```

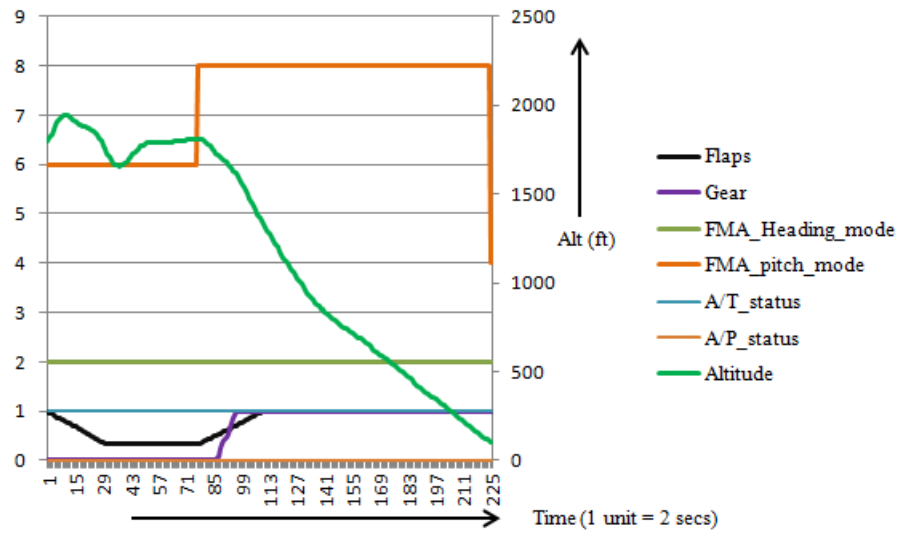



Figure 3.2: Characteristics of Normal Flight Parameters

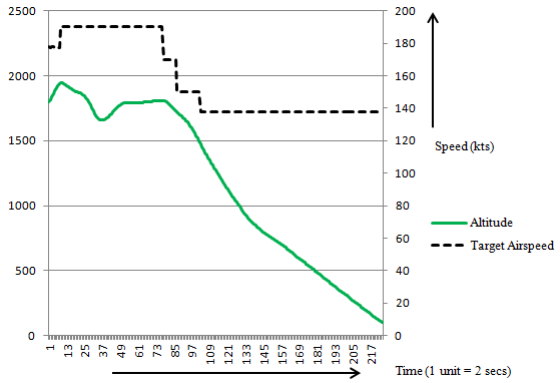



Figure 3.3: Characteristics of Normal Actual Altitude and Target Airspeed

aircraft approaches the runway. Primary vertical axis has altitude in feet and secondary vertical axis shows speed in knots.

## 3.4 Operational Characteristics of Anomalous Flights

We simulate anomalous flights in order to build the test dataset. For reproducing anomalous cases, adgPlugin allows user to override and manually control aircraft when needed. Thus we manually intervene and perform abnormal actions like making aircraft pitch up and slow down by pulling the control column, toggling a switch at an inappropriate time during the flight, increasing the thrust or decreasing the thrust abnormally for short durations and so on. Li et al.[3] have identified many anomalous flight types among which around 10 significant anomalous flight types have been observed during approach and landing flight phases. In this thesis, we studied those anomalous cases and reproduced most of them in Xplane and recorded the performance data. We have augmented these cases with few other common anomalies which in the past have resulted in fatal Controlled Flight Into Stall (CFIS) accidents. These anomalous flights along with other normal flights constitute the test data and we use this data for evaluating the performance of baseline and proposed algorithms. We now present the details of the operational characteristics of anomalous data collected as part of the experiments.

### Very High Airspeed Approach (Rushed and Unstable Approach)

This is a very high speed ILS approach which is a type of unstable approach as shown in Figure 3.4. Because of high energy state of the aircraft, the engines were always *idle* which resulted in significantly low n1 values (anomalous n1) throughout the later part of the approach. Also because of high speed, the approach took relatively less time than normal approaches.

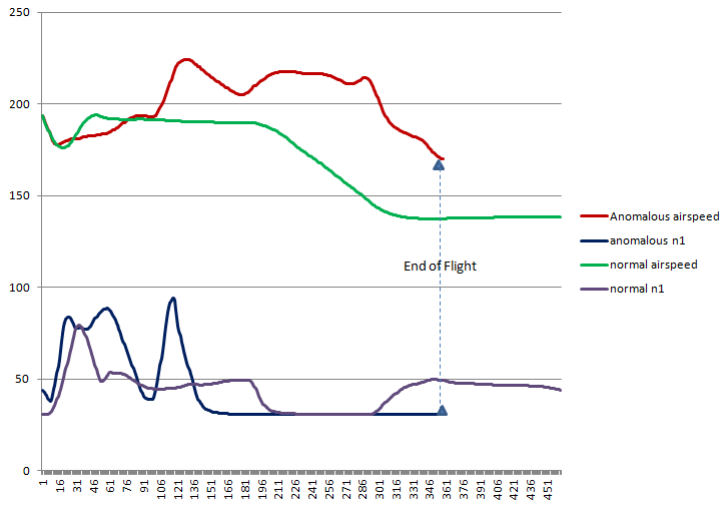



Figure 3.4: Very High Airspeed Approach (Rushed and Unstable Approach)

### Landing Runway Configuration Change

This type of anomalies have been observed in FOQA data by previous algorithms because of change in destination runway during *final approach*. We have considered two cases of this type, wherein the first case as detailed in Figure 3.5, the landing runway is changed from 28R to 28L after the aircroft crosses the ILS outer marker. We rely on deviations in latitude-longitude and target roll parameters to detect the runway configuration changes.

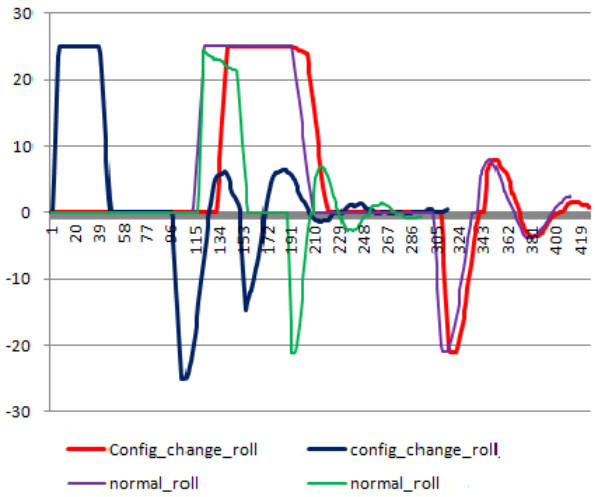



Figure 3.5: Landing Runway Configuration Change: Case 1

As shown in Figure 3.6, in this case the landing runway is changed from 28R to 28L similar to case 1, but when aircraft is very close to the destination. It has to be noted that, both these anomalies are very subtle and are not considered severe by the Exceedance detection algorithm[3].

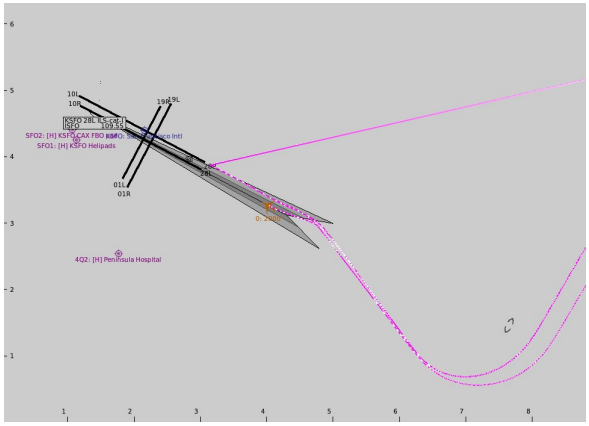


Figure 3.6: Landing Runway Configuration Change: Case 2

### Auto Land without Full Flaps(Unusual Auto Land Configuration)

Due to poor visibility and low ceiling altitude, visual landing may not be possible and automation is delegated to perform the landing of aircraft. There are strict requirements on both ground and airborne instruments for executing autoland operation. Generally this operation is performed with fully extended flaps and both auto pilots engaged. All the normal approaches executed by adgPlugin have full flaps configured with both autopilots engaged, during the auto landing mode. This anomalous case has flaps set to a configuration where flaps are not fully extended when the aircraft is in auto land mode as shown in Figure 3.7. The AFDS mode *LAND3* is the autoland mode for all the flights considered in this study.

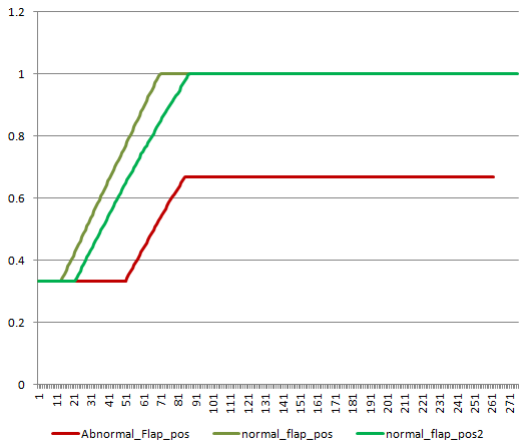



Figure 3.7: Auto Land without Full Flaps (Unusual Auto Land Configuration)

### Auto Land with Single Auto Pilot (Unusual Auto Land Configuration)

In this abnormal case, while flaps are set to full during autolanding, through out the flight, only single AP is engaged which constitutes an unusual auto land configuration.

### High Energy Approach (Too High or Too Fast or both)

This is an example of high energy flight because the airspeed was too high as shown in Figure 3.8. Once glideslope is caprtured in order to achieve rapid deceleration to target speed, the pitch was increased momentarily (as shown by anomalous pitch). This also resulted in abnormal vertical speeds (not shown). Thus the anomaly was result of multiple continuous parameters in this case.

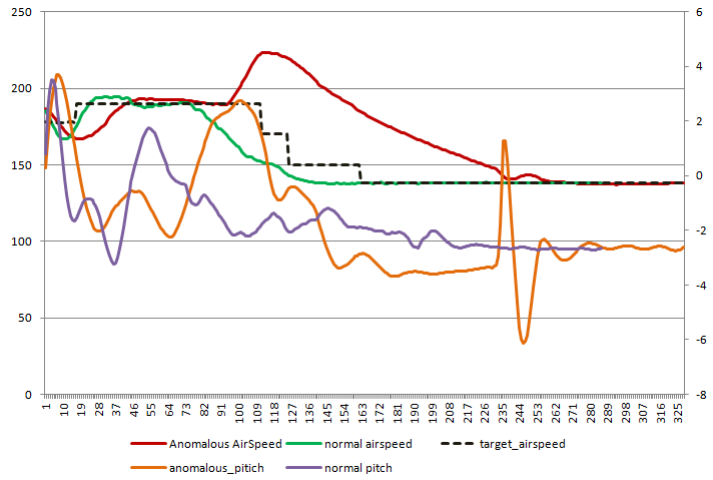



Figure 3.8: High Energy Approach (Too High or Too Fast or both)

### Recycling FDIR

As shown in Figure 3.9, the flight director switches are toggled (switched off and switched on) momentarily. Though this should ideally result in disconnect of auto pilots the simulation did not disconnect the automation. Nevertheless, the momentary toggle of FDIR switches is recorded in the discrete parameter data.

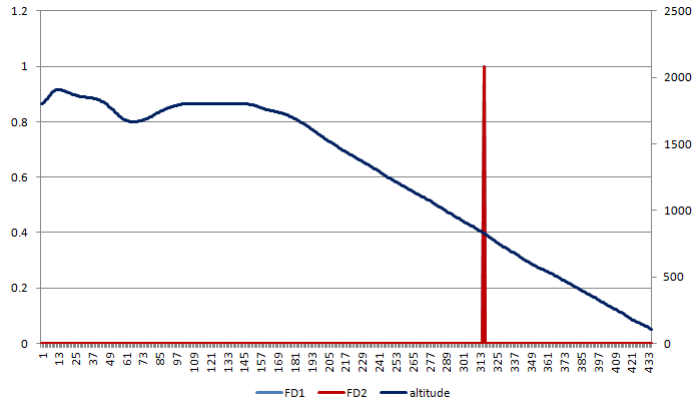



Figure 3.9: Recycling FDIR

### Influence of Wind

As shown in Figure3.10, there is a significant turbulence throughout this flight. The rapid fluctuations in the continuous parameter (airspeed) is recorded in the data for this anomalous flight. Though most of the flights are subjected to wind and turbulence this case is abnormal as the influence of wind is significantly higher.

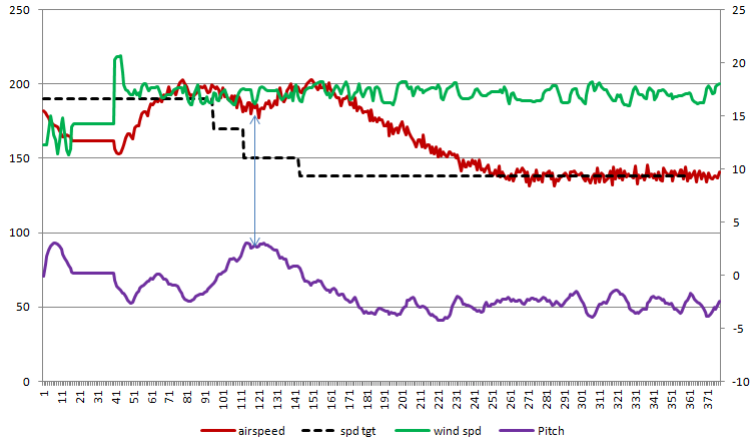



Figure 3.10: Influence of Wind

### High Pitch Rate for Short Duration

This flight is anomalous because of slight abnormalities in pitch just before landing as shown in Figure 3.11. Since the pitch is abnormal for only short durations, this anomaly is hard to detect.

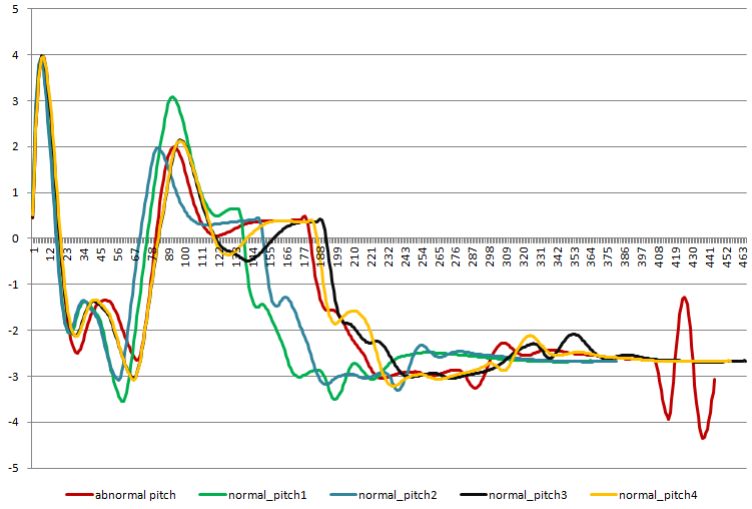



Figure 3.11: Characteristics of High Pitch Rate During Landing

### High Airspeed for Short Duration

This anomaly is related to high airspeed for very short duration. As shown in Figure 3.12, the airspeed was high for two short periods. The increase in airspeed was the result of anomalous pitch angle as shown in the Figure, but these are immediately rectified by appropriate actions. Since the deviations occur for short durations, these kinds of anomalies are difficult to be detected by the algorithms.

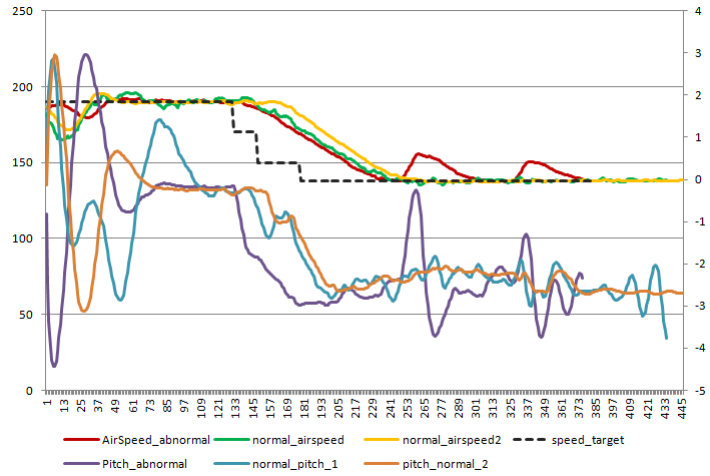



Figure 3.12: High-Airspeed for Short Durations

### GS missed from Below, Captured from Above with VS mode

In this case, the aircraft missed to capture the 3° glide slope path from below which is the case for normal flights in this study. The FMA pitch mode had to be changed to Vertical Speed in order for the aircraft to descend and capture the glideslope path from above. Once it is captured the Pitch mode automatically changes to GS from VS. This anomaly records the abnormalities in discrete parameter (FMA pitch mode) and also in a continuous parameter (Vertical Speed) as shown in Figure 3.13. Though the test set does not include this case and we do not present results for this anomaly, the proposed algorithms were able to detect this anomaly.

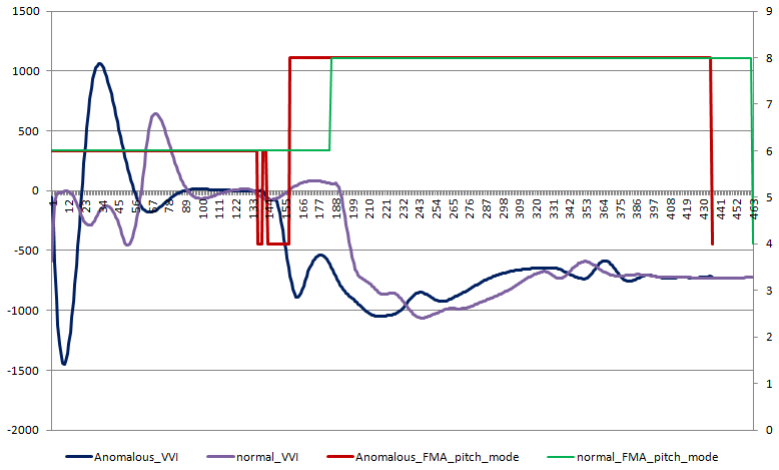



Figure 3.13: G/S missed from Below, captured from Above with V/S

### Low Energy Approach

This anomalous flight is the another case of unstable apporach but because of low energy state. As seen in Figure 3.14, the airspeed during the end of the approach is way less than the normal flights. Low energy unstable approaches are one of the major contributors of Controlled Flight into Stall/ Controlled Flight into Terrain accidents observed in the past.

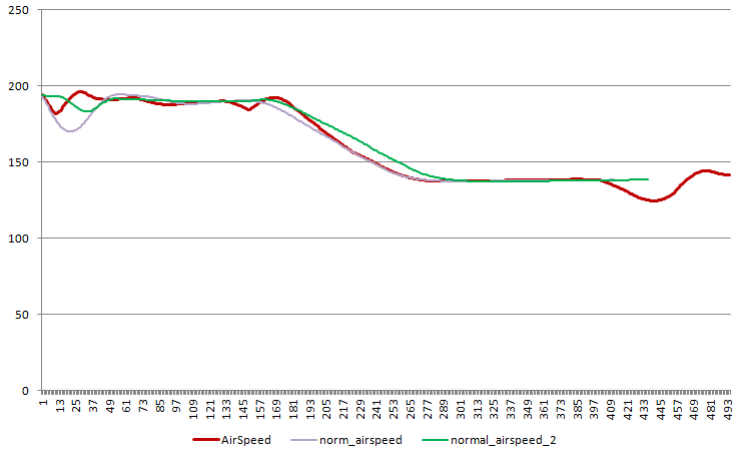



Figure 3.14: Characteristics of Low Energy Approach

# Chapter 4: Auto Encoders and Recurrent Neural Networks

In this chapter we shall discuss the theory behind the two kinds of neural networks we are using in this thesis: Autoencoders and Recurrent Neural Networks. Specifically, we briefly present the underlying algorithms viz., *gradient descent* using error back propagation for learning in autoencoders and *backpropagation through time* (BPTT) based on which recurrent neural networks learn.

## 4.1 Autoencoders[1]

An autoencoder neural network is an unsupervised learning algorithm that learns efficiently encodings. It is trained to reconstruct its own inputs through error backpropagation. In other words, it learns an approximation to the identity function so that output $\hat{x}$ is similar to input $x$. Figure 4.1 depicts an autoencoder with three layers. The layer $L_1$ is the input layer, $L_2$ is the hidden layer and $L_3$ is the output layer. The network has 6 input neurons also called input units, 3 units in hidden layer, and 6 units in output layer. It has same number of neurons in input layer and output layer. Units labeled as +1 are called *bias units*, equivalent to intercept in a regression model [24]. The presence of bias units is crucial for effective learning and these terms can be learned just like other weights. They also help in faster learning by enabling faster convergence.

Before more rigorous analysis of autoencoders a brief mention of the terminology would be helpful. Let $n_l$ denote number of layers in the network and let $L_l$ denote any of the layers in the neural network. The neural network has parameters $(W, b)$. For autoencoder neural net in Figure 4.1 $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$. $W_{ij}^{(l)}$ specifies the parameter or weight

[1]The concepts and equations in this section are adapted from Section 2.2 and Chapter 3 of Sparse Encoders by Andrew Ng [23]

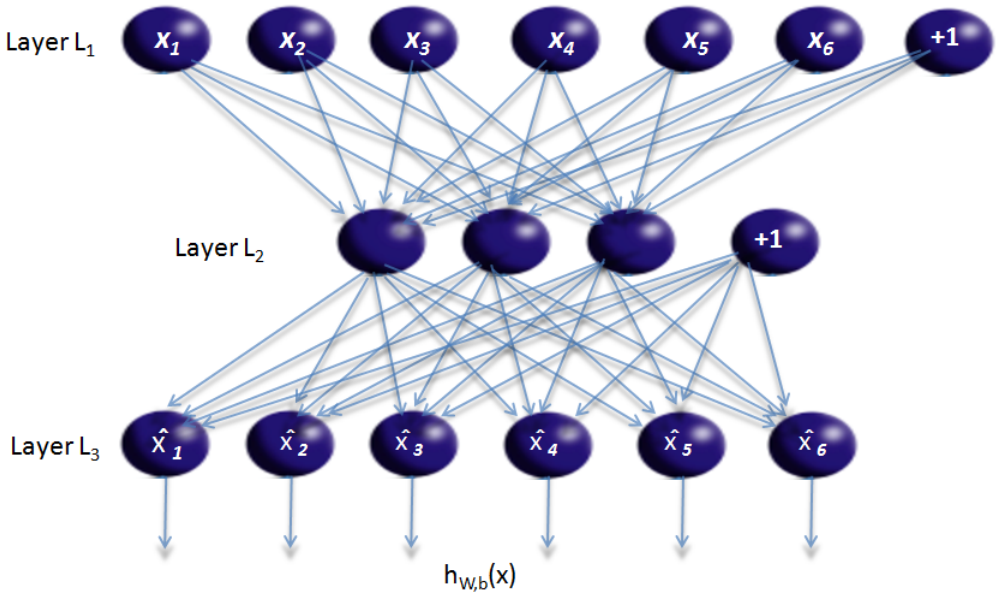



Figure 4.1: An autoencoder with one hidden layer with bias units (shown by +1)

associated with the connection between unit $j$ in layer $l$, and unit $i$ in layer $l+1$. $b_i^{(l)}$ is the bias associated with unit $i$ in layer $l+1$. For autoencoder in Figure 4.1, $W^{(1)} \in \mathbb{R}^{3x6}, W^{(2)} \in \mathbb{R}^{6x3}$. Also let $k_l$ denote the number of nodes in layer$l$ without counting the bias unit. Bias units do not have any inputs or incoming connections. The activation of a unit $i$ in layer $l$ is denoted by $a_i^{(l)}$. Thus each of the input units $x_i$ can be represented by $a^(1)_i$. Given a neural network with parameters $W, b$ and unlabeled training examples $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots\}$, where $x^{(i)} \in \mathbb{R}^n$ then the output of autoencoder given by $y^{(i)}$ is equal to input $x^{(i)}$. Thus $y^{(i)} = x^{(i)}$ and it learns a hypothesis $h_{W,b}(x) = x$. Though learning an identity function may seem easy, by enforcing some restrictions on the network such as limiting the number of hidden neurons, autoencoders can learn interesting features about the input data thus acting Feature Detectors. If the input data has features that are correlated then autoencoder will learn these correlations as well, thus acting in its simplest form as a dimensionality reduction algorithm similar to PCA.

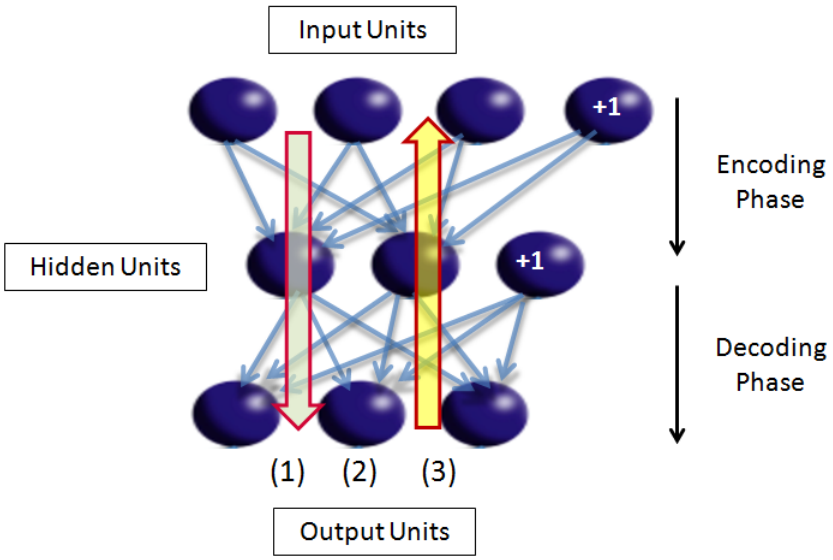



Figure 4.2: An Autoencoder learns to reconstruct inputs. (1) Forward propagate inputs $y = f(\sum(w.x))$ (2) Calculate errors, (predicted output - desired output) (3) Back propagate Errors from output units

### 4.1.1 Learning the structure of inputs

As shown in Figure 4.2, training of autoencoder consists three steps:

1. Forward propagate the inputs
   (a) Encoding phase
   (b) Decoding phase
2. Calculate the error or cost function
3. Back propagate the calculated errors

Consider the network shown in Figure 4.2. For this autoencoder $n_l = 3$. Let $L_1$ be the input layer, $L_2$ is hidden layer, $L_3$ is the output layer with $k_1 = 3, k_2 = 2$ and $k_3 = 3$. The feedforward values forward propagated by the network (1) are computed as below: Computations during the encoding phase decide the activation of hidden units in $L_2$ are given as follows:

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \tag{4.1}$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \tag{4.2}$$

Above equations can be rewritten compactly as,

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f(z^{(2)}) \end{aligned} \tag{4.3}$$

Decoding phase computations determine the activations of output units in $L_3$ and are given as follows

$$a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + b_1^{(2)}) \tag{4.4}$$

$$a_2^{(3)} = f(W_{21}^{(2)}a_1^{(2)} + W_{22}^{(2)}a_2^{(2)} + b_2^{(2)}) \tag{4.5}$$

$$a_3^{(3)} = f(W_{31}^{(2)}a_1^{(2)} + W_{32}^{(2)}a_2^{(2)} + b_3^{(2)}) \tag{4.6}$$

Above equations can be rewritten compactly as,

$$\begin{aligned} z^{(3)} &= W^{(2)}x + b^{(2)} \\ h_{W,b}^{(x)} &= f(z^{(3)}) \end{aligned} \tag{4.7}$$

Thus more generally, the activations in layer $l+1$ can be computed by using the activations in layer $l$ as,

$$\begin{aligned} z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\ a^{(l+1)} &= f(z^{(l+1)}) \end{aligned} \tag{4.8}$$

Where $f$ is the activation function. It is the transformation function which determines the activation behavior of the neuron. Generally, sigmoid or hyperbolic tangent activation functions are used. The plots of these functions are shown in Figure 4.3. Sigmoid function

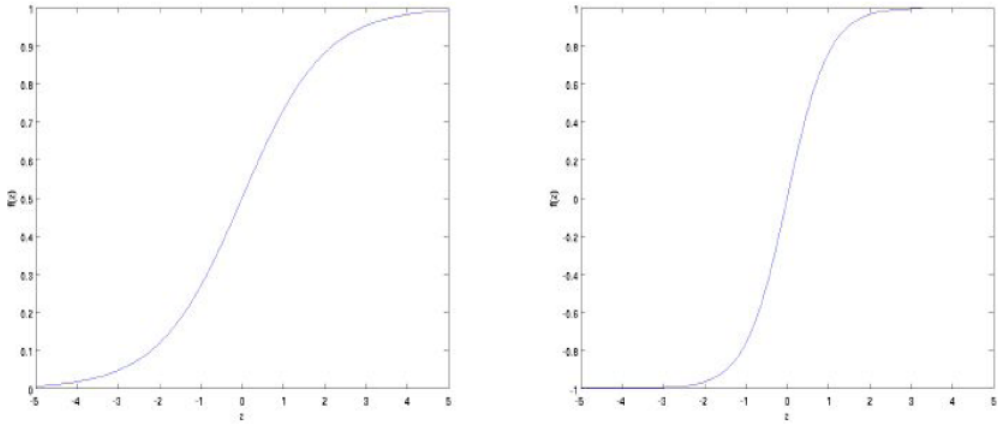


Figure 4.3: sigmoid (left) and tanh (right) activation functions

$\sigma(z) = \frac{1}{1+\exp(-z)}$. The output of the sigmoid function lies in the range of [0,1]. Thus sigmoid transforms any negative input value to 0 and all positve inputs are limited by +1. Sigmoids units are only sensitive to values between 0 and 1 and left unchanged. Similarly, the output of *tanh* function lies in the range of [-1,1] and is given as: $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. Thus all negative inputs less than $-1$ to $-\infty$ are transformed into $-1$ and positve values from $+1$ to $+\infty$ are limited by $+1$. It is sensitive to only inputs between $-1$ and $+1$.

**Important properties of Activation Functions**

The sigmoidal and hyperbolic tangent activation function discussed above have two important properties:

1. NonLinearity: Nonlinear neural networks are more powerful than linear networks because they can find nonlinear decision boundaries and can model nonlinear functions. Any combination of linear operators is itself a linear operator,which means that any Multi Layer Perceptron (MLP) network with multiple linear hidden layers is exactly equivalent to some other MLP with a single linear hidden layer. This contrasts with nonlinear networks, which can gain considerable power by using successive hidden layers to rerepresent the input data [25][26].

2. Differentiability: Both the activation functions are differentiable which allows the networks to be trained with gradient descent. Below given are the first order differentials of tanh and sigmoid functions respectively.

   $\frac{\partial tanh(x)}{\partial x} = 1 - tanh(x)^2$

   $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

### 4.1.2 Calculating the error and backpropagating the error

Suppose we have following unlabeled set of training instances $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$ We can consider this set of training instances as $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots$, where $y^{(i)} = x^{(i)} \forall i$. For a single training example, the cost function or error is given as:

$$E(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2 \tag{4.9}$$

Over a set of $m$ training examples, it is calculated as:

$$\begin{aligned} E(W, b) &= \left[ \frac{1}{m} \sum_{i=1}^{m} E(W, b; x^{(i)}, y^{(i)}) \right] + \frac{\gamma}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{k_l} \sum_{j=1}^{k_{l+1}} \left( W_{ji}^{(l)} \right)^2 \\ &= \left[ \frac{1}{m} \sum_{i=1}^{m} \left( \frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\gamma}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{k_l} \sum_{j=1}^{k_{l+1}} \left( W_{ji}^{(l)} \right)^2 \end{aligned} \tag{4.10}$$

The first term is an average sum of squared errors term. The second term is a regularization term (also called weight decay) term which tends to decrease the magnitude of weights and helps the network to avoid overfitting to the training examples. Weight decay parameter $\gamma$ controls the relative significance of each of the two terms. The goal is to minimize $E(W, b)$ as a function of $W$ and $b$. We begin the training by initializing the weights $W_{ij}^{(l)}$ and $b_i^{(l)}$ to small random values close to zero. This helps to prevent all of the hidden units learning same function of the input.

### 4.1.3 Backpropagation Algorithm

This is the algorithm using which the errors are backpropagated in the network, so that updates are made to the weights iteratively till the autoencoder network learns to approximate the provided input data. Given a training example, we will first run a *forward pass* to compute all the activations throughout the network, including the output value of the hypothesis $h_{W,b}(x)$. Then, for each node $i$ in layer $l$, we compute an error term $\delta_i^{(l)}$ that measures how much that node was responsible for any errors in the output. For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to calculate $\delta_i^{n_l}$ where $n_l$ is the output layer. But we compute $\delta_i^{(l)}$ based on a weighted average of the error terms of the nodes that uses $a_i^{(l)}$ as an input. The backpropagation algorithm is given below

**Algorithm 2** Error Backpropagation

1: **procedure** BACKPROP(TrainingExample $m_i$)

2: Compute activations for layers $L_2, L_3, \ldots, L_{n_l}$

3: For each output unit $i$ in layer $n_l$ (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|h_{W,b}(x) - y\|^2 = -(y_i - a_i^{(n_l)}).f'(z_i^{(n_l)})$$

4: For each node $i$ in hidden layer $l = n_l - 1, n_l - 2, \ldots, 2$, set the error term

$$\delta_i^{(l)} = \left( \sum_{j=1}^{k_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

5: Compute the desired partial derivatives as

$$\frac{\partial}{\partial W_{ij}^{(l)}} E(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} E(W, b; x, y) = \delta_i^{(l+1)}.$$

6: **end procedure**

### 4.1.4 Gradient Descent using Back propagation

The basic idea of gradient descent is that to calculate the partial derivative of loss function with respect to each of the network weights and then adjust the weights in the direction of negative gradient.

**Algorithm 3** Gradient Descent

**procedure** GRADIENTDESCENT(TrainingExamples $m_1, m_2, \ldots$)
  $\Delta W^{(l)} \leftarrow 0, \Delta b^{(l)} \leftarrow 0$
  **for all** $m_i$ in TrainingExamples **do**
    Use Backprop Algorithm 2 to compute partial deravatives

$$\nabla_{W^{(l)}} E(W, b; x, \hat{x})$$

$$\nabla_{b^{(l)}} E(W, b; x, \hat{x})$$

    $\Delta W^{(l)} \leftarrow \Delta W^{(l)} + \nabla_{W^{(l)}} E(W, bx, \hat{x})$
    $\Delta b^{(l)} \leftarrow \Delta b^{(l)} + \nabla_{b^{(l)}} E(W, bx, \hat{x})$
  **end for**
  Update parameters:

$$W^{(l)} = W^{(l)} - \alpha \left[ \left( \frac{1}{m} \Delta W^{(l)} \right) + \gamma W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[ \frac{1}{m} \Delta b^{(l)} \right]$$

**end procedure**

Where $\alpha$ is the learning rate which determines the size of the steps to take along the error surface during gradient descent. The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight tuning iterations increases. Repeatedly perform Gradient Descent to reduce the Error Function $E(W, b)$. The trouble with training autoencoders and neural networks in general is that the Error function $E(W, b)$ is a *non-convex* function and thus the gradient descent over the error surface is susceptible to local minima.

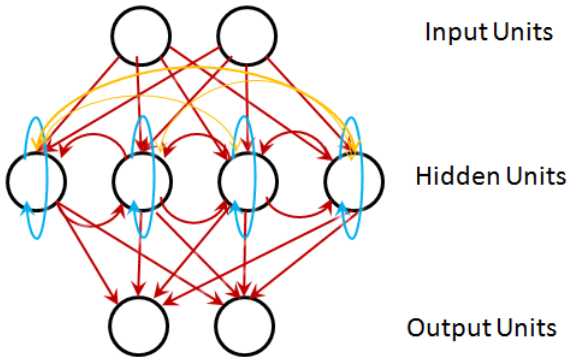



Figure 4.4: A Recurrent Neural Network with single hidden layer

## 4.2 Recurrent Neural Networks[2]

As opposed to Feed forward neural networks which are acyclic, Recurrent Neural Networks are artificial neural networks with cyclic connections. Figure 4.4 shows a recurrent neural network (RNN) with a single, self connected hidden layer. It is interesting to observe that though the difference between feedforward neural network like an autoencoder and RNN may seem trivial, but by virtue of having cycles in the network allow RNNs to preserve valuable information from the past inputs. The recurrent connections allow a memory of previous inputs to persist in the networks internal state. This valuable temporal information can be used to influence the current output of the network. It can also be noted that the restriction of having equal number of units in output layer as that of input layer is no longer enforced in RNNs. But when we want to predict features or sequence of features based on current inputs and previous time steps inputs, we design the network such that number of units in input layer equals number of units in output layer.

### 4.2.1 Learning in Recurrent Neural Networks

The training of recurrent neural networks is similar to feedforward autoencoder networks presented in previous section, but with certain variations. In the forward pass the only

[2]Equations in this section are adapted from Section 3.2 of Supervised Sequence Labelling with Recurrent Neural Networks by Alex Graves [27]

difference between recurrent neural networks and feedforward neural networks is that for hidden layers, the connections come from both external inputs and also from hidden layer activations from previous timesteps. Consider a sequence $x$ of length $T$ be provided as input to an RNN with $I$ input units, $H$ hidden units, and $R$ output units. Let $x_i^t$ be the value of input $i$ at time $t$ , and let $a_j^t$ and $z_j^t$ be the network input to unit $j$ at time $t$ and the activation of unit $j$ at time $t$ respectively. Note the subtle difference in notations from previous section. Now during forward pass for hidden unit $h$, the inputs at time $t$ can be given as

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} z_{h'}^{t-1} \tag{4.11}$$

By applying the chosen nonlinear differentiable activation function $f$ on the inputs, the activation of hidden unit at time $t$ can be given as,

$$z_h^t = f(a_h^t) \tag{4.12}$$

The complete sequence of hidden activations are calculated by starting at $t = 1$ and recursively applying Equations 4.11 and 4.12 with increments of $t$ at each step. The initial values for hidden units for initial time step $t_0$ can be either set to zero or to nonzero initial values as in [28]. Using the activations of hidden units we can calculate the inputs for the output units using Equation 4.13

$$a_r^t = \sum_{h=1}^{H} w_{hr} z_h^t \tag{4.13}$$

### 4.2.2 Backward pass using Backpropagation Through Time (BPTT)

Given the partial derivatives of some differentiable error function E with respect to the network outputs we have to determine the derivatives with respect to the weights. This can be easily accomplished if the activation function chosen is easily differentiable. Two algorithms are well known to efficiently calculate weight derivatives for RNNs,

1. Real Time Recurrent Learning (RTRL) [29]

2. Backpropagation Through Time (BPTT) [30][31]

Backpropagation Through Time is discussed here as it is more relevant for this thesis. Since the error function depends on the activation of the hidden layer not only through its influence on the output layer, but also through its influence on the hidden layer at the next timestep, the error for hidden layer $h$ at time $t$ is given by

$$\delta_h^t = f'(a_h^t)\left(\sum_{r=1}^{R}\delta_r^t w_{hr} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}\right) \tag{4.14}$$

where $\delta_j^t = \frac{\partial E}{\partial a_j^t}$ is the error at time $t$ for unit $j$ given as deravative of error function with respect to inputs at unit $j$ at time $t$. The complete sequence of terms can be calculated by starting at $t = T$ and recursively applying Equation 4.14, decrementing $t$ at each step. Now, since the same set of weights can be reused at every timestep, we can sum over the whole sequence over the time to obtain the derivatives with respect to the network weights as follows by applying the chain rule,

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T}\frac{\partial E}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T}\delta_j^t b_i^t \tag{4.15}$$

### 4.2.3 Long Short-Term Memory (LSTM)

The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections. This occurs as the error is perturbed by the current irrelevant inputs to the network. This effect is often referred to in the literature as the vanishing gradient problem or exploding gradient problem [32] [33] depending on whether the gradients decay exponentially or explode.

To overcome this problem various solutions have been proposed as simulated annealing and discrete error propagation by Bengio et.al[33] and explicitly introduced time delays Lang et al. [34] among other solutions. But the Long Short-Term Memory proposed by Hochreiter et al [35] is explored in this thesis. LSTMs have solved several problems that remain impossible with any other RNN architecture. LSTM has been capable of solving problems requiring the use of long range contextual information. It has been successfully applied for protein secondary structure prediction [36] [37], music generation [38], reinforcement learning [39], speech recognition [40] [41] and handwriting recognition [42] [43]. In short, the design of LSTMs enable them to learn *long term dependencies* in the input. The LSTM architecture consists of a set of recurrently connected structures called *memory blocks*. Each block contains one or more self-connected memory *cells* each with an associated *cell state*. The memory block is built around the cell(s) which ensure **constant error flow** through them by using an identity function and always getting incoming unit weights. The central feature of LSTM is based on the concept of achieving constant error flow through a given unit $i$ having a single connection to itself [35]. The back propagated error for unit $i$ is

$$\delta_i^{(t)} = f_i'(a_i^{(t)})\delta_i^{(t+1)}w_{ii} \tag{4.16}$$

For constant error flow we need

$$f_i'(a_i^{(t)})w_{ii} = 1.0 \tag{4.17}$$

Differentiating Equation **??**, we get

$$f_i(a_i^{(t)}) = \frac{a_i^{(t)}}{w_{ii}}. \tag{4.18}$$

Thus when the activation function is identity function such that $f_i : f_i(x) = x, \forall x$ and setting $w_{ii} = 1$, the error can be backpropagated constantly forever, in theory. This is referred to as **Constant Error Carousel (CEC)**. In reality unit $i$ will be connected to

other units other than to itself. And because of this there would be difficulties in learning due to conflicting input weights and output weights. Because of this special structures called *gates* are introduced. They protect the stored error from perturbations and allow access to it only when needed. LSTM has three multiplicative gates, *viz.*, the input, output and forget gates and using these, the error flow throughout the network can be regulated. Gates allow the information to pass through when they are open and block the information when closed. Input gate protect the error from perturbing due irrelevant inputs. Likewise, output gate protect other units from currently irrelevant contents stored in this unit. When to open gates and when to close is also learned by the network. Generally, a layer of sigmoid units is used to model the gates. Apart from these *tanh* or *sigmoid* activation functions are used on inputs coming into the cell. Similarly on output from cell one of the *tanh*, *sigmoid* or identity activation functions are applied. Figure 4.5 shows structure of an LSTM unit with single memory cell.

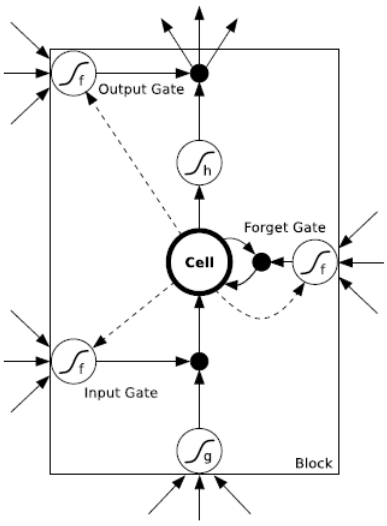



Figure 4.5: A LSTM memory block with single unit with Input gate, Forget Gate and Output Gate. (Source: Section 4.1 [27])

### 4.2.4 Gated Recurrent Unit(GRU)

Gated Recurrent Units are special variants of LSTMs and are introduced by Cho et al. [44]. It merges the forget and input gates into a single update gate along with some other changes resulting in a much simpler model than standard LSTM models. They are used on the tasks of polyphonic music modeling and speech signal modeling by Chung et al. [45] and the performance is found to be comparable to LSTMs. In this thesis we use RNNs based on both GRU and LSTM units and compare their performance. See Section 5 for results.

# Chapter 5: Experiments and Results

In this chapter we discuss the datasets used, experimental protocols followed and the results obtained. We begin with the performance of the baseline algorithm MKAD in detecting anomalous sequences in the test data and compare it with performance of autoencoders on same test dataset. Then, we present the results of Recurrent neural networks in detecting anomalous sequences in the provided test dataset. The neural networks, both autoencoders and RNNs are trained in semi-supervised fashion, wherein the negative class samples with only normal examples are used for training. Test data contains the positive samples (anomalous sequences) we have reproduced (as presented in chapter 3) and also includes some negative samples or normal flights. It is worth mentioning that, real world datasets like FOQA data may contain both normal and abnormal data sequences and we may not know the class labels. Since unsupervised methods do not require any knowledge of labels for the data samples they can operate on this data as-is. But for semi-supervised algorithms proposed here and for MKAD a preprocessing step is needed. It is based on the premise that *anomalies are not concentrated*[10]. Thus during data preprocessing we can apply clustering and consider the most nominal flights belonging to clusters as the training data. Otherwise, when the dataset has huge number of flights the most nominal flights can be identified through a filtering step and use them for training. A similar methodology is followed by Das et al. for MKAD [1]. Since One Class SVM used in MKAD is a semi-supervised learning model this step is needed. Since in this work, we have knowledge of the normal flight sequences for the training set, we did not consider this in our methodology.

## 5.1 Data Used

We have collected the data for 500 flights (approaches into KSFO 28L and 28R) in the X plane simulation as described in Chapter 3. Of these 500 flights 489 are normal flights generated by the adgPlugin with no intervention. Remaining 11 flights are anomalous and are reproduced for the purpose of testing and evaluating the algorithms. The 500 flights are split into training set, validation set and test sets as discussed here. We have designed 3 dataset combinations, with varying sizes of training and test sets in first two combinations while third combination includes a validation set of 50 samples along with training and test sets. These dataset combinations are designed for the in an attempt to evaluate the *robustness* of the models by observing the difference in their performance due to change in training instances. But for observing this, we could have definitely used *k-fold cross validation* technique instead, by randomly selecting the normal data samples to be used in test set and repeating the experiment $k$ times each time with different resulting training and test data. The future work would definitely include k-fold cross validation in the methodology. Though MKAD is run on first two combinations, experiments on Autoencoders used all three combinations. Table 5.1 summarizes all three combinations.

Table 5.1: Different Combinations of Training and Test Data Samples Used

| **DataSet** | **Number of Instances** |
|---|---|
| Train 1 | 355 |
| Test 1 | 145 |
| Train 2 | 450 |
| Test 2 | 50 |
| Train 3 | 355 |
| Validate 3 | 50 |
| Test 3 | 95 |

## 5.2 Evaluation Metrics

For evaluating various models discussed in this thesis, we use precision, recall and $F_1$ score given in Equation 5.1. High precision signifies that model has been able to detect anomalies correctly (true positives) more than falsely classifying normal instances as anomalies (false positives). High recall results in detection of more anomalies correctly than falsely classifying them as normal instances (false negatives). Thus signifying the *sensitivity* of the model in detecting anomalous instances in the given data. $F_1$ score is *Harmonic Mean* of precision and recall.

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \\ F_1 score &= 2.\frac{Precision * Recall}{Precision + Recall} \end{aligned} \tag{5.1}$$

where

**TP** is number of True Positives,

**FP** is number of False Positives,

**TN** is number of True Negatives,

**FN** is number of False Negatives.

## 5.3 Performance Comparison of MKAD and Autoencoders

We trained MKAD algorithm[1][2] with various parameter settings and evaluated the performance of each model on the test dataset. Specifically, we have trained MKAD algorithm with three different window lengths of 30, 60 and 90 which result in different SAX encodings of the same time series data. Also we have used two different alphabet sizes, one with 20 and other with 40. Alphabet size determines the number of symbols used

in encoding the timeseries data. Though we could not cover all possible combinations of settings, we have trained with the following parameter combinations as shown in Table 5.2.

Table 5.2: Different MKAD Models with Corresponding Parameter Combinations

| **Model** | **Window Len** | **Alphabet size** |
|---|---|---|
| MKAD1 | 30 | 20 |
| MKAD2 | 60 | 20 |
| MKAD3 | 90 | 40 |

Similarly, we have used different architectures of autoencoders and tested the models on same data as that of MKAD models. Training autoencoders and neural networks in general is very tricky and involves fine tuning of various parameters for different architectures. Also the number of iterations over the training data (epochs) is also an important factor for successful training because of the problem of overfitting to training data. Table 5.3 describes the architectures of various autoencoder models trained using error backpropagation using gradient descent.

Table 5.3: Different Autoencoder Models with Corresponding Parameter Combinations

| **Model** | **Hidden Layers** | **Activation Function** | **L1 Error** | **Epochs** |
|---|---|---|---|---|
| AutoEncoder1 | (1000,500,1000) | Tanh (with Dropout) | 1E-4 | 3 |
| AutoEncoder2 | (1000,800,400,800,1000) | Tanh (with Dropout) | 1E-4 | 3 |
| AutoEncoder3 | (1000,800,200,800,1000) | Tanh (with Dropout) | 1E-4 | 3 |
| AutoEncoder4 | (1000,500,1000) | Tanh (with Dropout) | 1E-4 | 10 |
| AutoEncoder5 | (1000,500,1000) | Tanh (with Dropout) | 1E-4 | 100 |

### 5.3.1 Methodology for Training Autoencoders

#### Converting Sequences into High Dimensional Time Series Data

The collected data for each flight is a multidimensional data with varying number of sequences, as different approaches may have different durations. As a first step, all the data of 500 individual flights is transformed into a very high dimensional time series data as in[3]. The high dimensional time series data for all the flights are zero padded to ensure each example has equal number of features. The flight with longest duration will considered to decided the dimensionality of the feature vector. This is required as autoencoders can be trained only on examples that have fixed dimensions.

#### Data Normalization

All the data is normalized so that all the inputs lie in range of [0,1]. It is important to note normalization is performed on the whole dataset which includes both training and test data.

#### Training the Autoencoder

The autoencoder model is trained through error backpropagation using gradient descent as discussed in Chapter 4 on the training set provided for specified number of iterations. In some of the models we have used validation dataset (Validate3) during training. It was used to achieve better generalization of the model on unseen examples. This also prevents model from overfitting to the noise in the training examples. When training data set is huge, monitoring validation error and training error can be useful to visualize when the model is beginning to overfit to training examples and thus when to stop further training.

#### Testing the model for anomaly detection

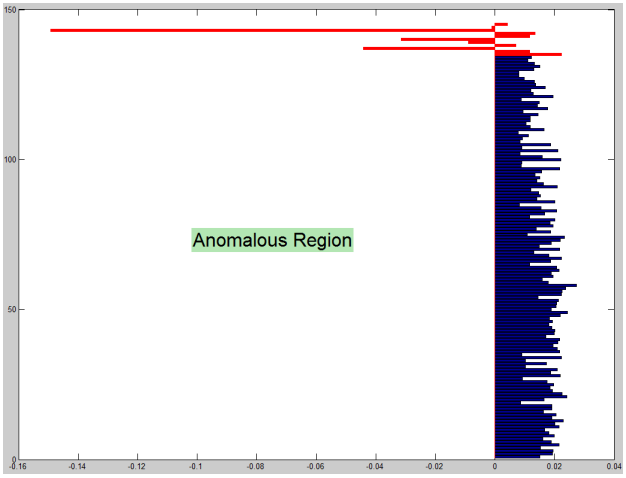
Once the model is trained on normal examples, it is presented with unseen test data containing both normal and anomalous examples. The main idea is that the autoencoder learns the structure of the normal data presented during training and the model should be able reconstruct similar data relatively comfortably which results in a low reconstruction

error. For anomalous cases the reconstruction error should be relatively high. In this work we have considered RMSE as the measure of reconstruction error. Hence the test examples with low RMSE values are treated as normal and examples which have relatively higher values are considered anomalous.

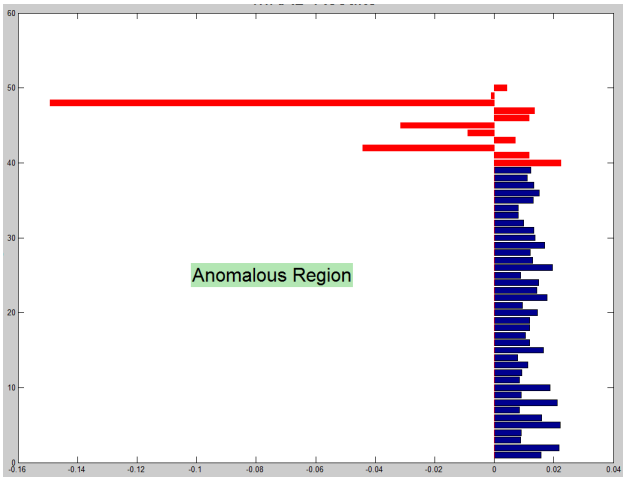
### 5.3.2 Results: MKAD

We here present the results of MKAD with different parameter settings on different dataset combinations. The following is the order of anomalous cases present in the test data for both MKAD and Autoencoder models. For MKAD the first case 'Abnormal Pitch for short duration' is represented by first red column from the bottom in all figures, whereas for Autoencoders the list corresponds to red columns from left to right in the corresponding figures. The list of positive examples in order of their presence in test data is as follows:

1. Abnormal pitch short duration (PTCH)
2. FDIR recycling (FDIR)
3. High airspeed short duration (SHORT)
4. High Energy Approach (HENG)
5. Partial Flaps (FLAP)
6. Single AP approach (1AP)
7. RW configuration change #1 (RW#1)
8. RW configuration change #2 (RW#2)
9. Very High Speed Approach (VHSPD)
10. Influence of Wind (WIND)
11. Low energy approach (LENG)



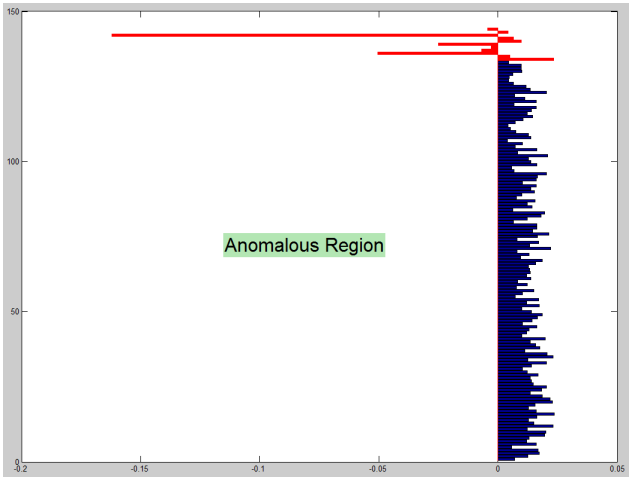

(a) MKAD1 on Test1



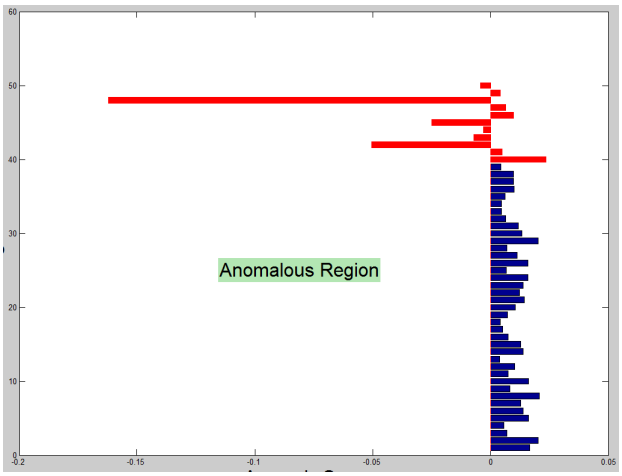

(b) MKAD1 on Test2

Figure 5.1: Performance of MKAD1

**Observations**: MKAD1 is able to detect 5 out of 11 anomalies on both test sets as shown in Figure 5.1. None of the normal flights are classified as anomalous by MKAD1. It has missed Abnormal pitch short duration, FDIR recycling, High Energy, Runway Config Changes both 1 & 2 and Low energy anomalies.



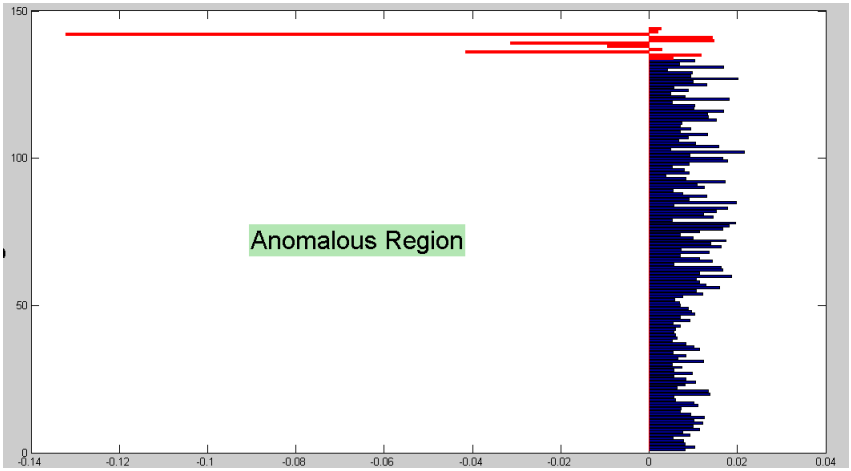

(a) MKAD2 on Test1



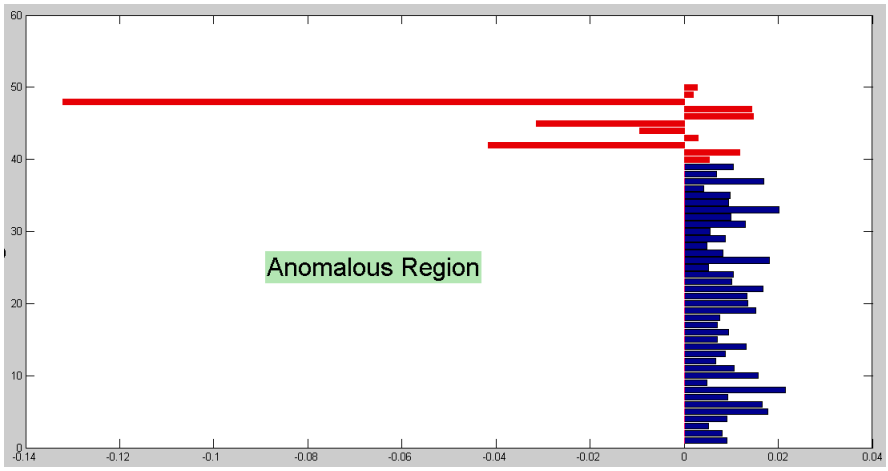

(b) MKAD2 on Test2

Figure 5.2: Performance of MKAD2

**Observations**: MKAD2 is able to detect 6 out of 11 anomalies on both test sets as shown in Figure 5.2. Also, none of the normal flights are classified as anomalous by MKAD2. It has missed Abnormal pitch for short duration, FDIR recycling, both cases of Runway Configuration Changes and Influence of wind anomalies.




(a) MKAD3 on Test1




(b) MKAD3 on Test2

Figure 5.3: MKAD3 Performance

**Observations**: MKAD3 is able to detect the only 4 anomalies on both test sets as shown in Figure 5.3. None of the normal flights are classified as anomalous by MKAD3. It has missed Abnormal pitch for short duration, FDIR recycling, High Energy approach, both cases of Runway Configuration Changes, Influence of wind and and Low energy approach anomalies.

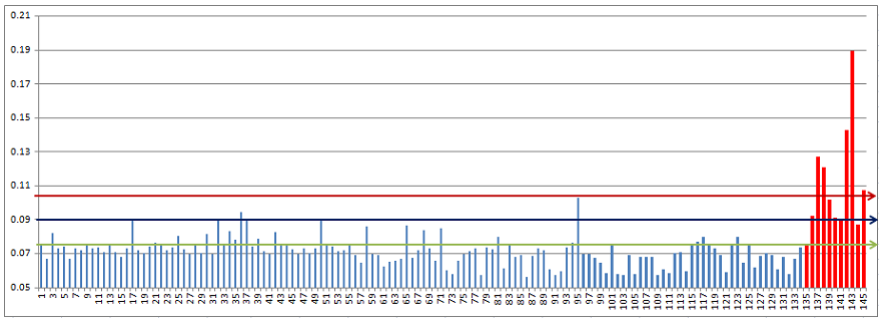
### 5.3.3 Results: Autoencoders

We present the results of three autoencoder models, *viz.,* Autoencoder1, Autoencoder4 and Autoencoder5.

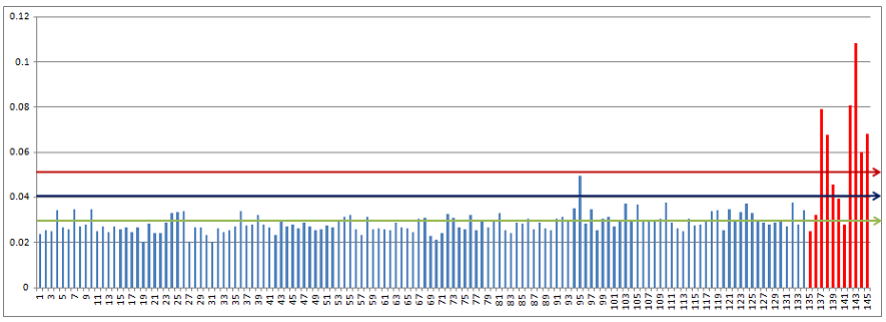
**Observations**: We presented $\mu + 1\sigma$ and $\mu + 2\sigma$ thresholds shown by blue and red arrows respectively. By varying the threshold we can control the sensitivity of the models in identifying the anomalies. While higher thresholds may improve precision values by reducing number of false positives, they negatively impact the recall values as some of the true positives are also missed. Since in this research we propose to use autoencoders only for offline anomaly detection, we prefer higher values of true positives at the expense of some false positives. And since $+2\sigma$ threshold is more restrictive on number of true positives, through rest of the analysis we use $+1\sigma$ as the threshold. Figure 5.4a shows the performance of Autoencoder1 on Test1. The reconstruction errors is relatively high for all anomalous cases (red columns) except in cases 1 & 10 which are abnormal pitch for short duration and influence of wind anomalies respectively. Performance of Autoencoder4 on Test1 is shown in Figure 5.4b. This was able to detect 7 out of 11 anomalies (true positives) with $+1\sigma$ threshold. The abnormal pitch for short duration, FDIR recycling, flight with single Autopilot and runway configuration change 1 were missed (False negatives). There was one normal flight classified as anomalous (false positive). Though performance of Autoencoder5 as shown in Figure 5.4c is similar to Autoencoder4, the RMSE values over all test samples are less. It is interesting to note that since, unlike MKAD, autoencoders have classified some of the normal flights as anomalous, they result in false positives and thus precision values are less than 1.

#### Limitations of Autoencoders

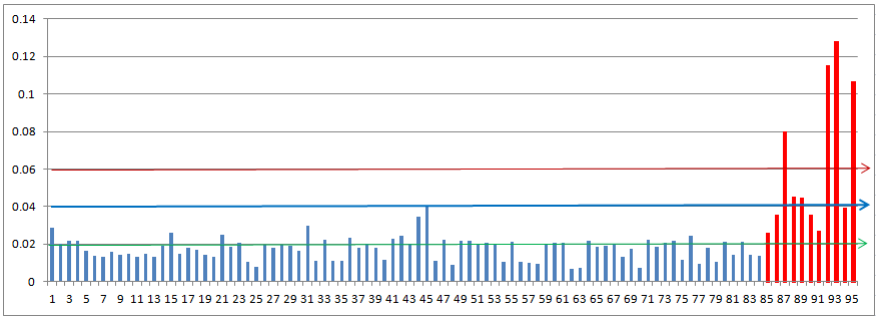
Even for autoencoders we convert sequential data into high dimensional time series data. Since the autoencoders require fixed input size, we had to pad the feature vectors with zeroes to ensure the dimensions of all inputs are consistent. This may be computationally inefficient and prevents this to be applied for online anomaly detection.



(a) Autoencoder1 on Test1: Reconstruction errors on Test1 (RMSE values). Green arrow at 0.075 indicates the average RMSE, Blue arrow indicates the $\mu+\sigma$ and Red arrow indicates $\mu + 2 * \sigma$



(b) Autoencoder4 on Test1: Reconstruction errors on Test1. Green arrow at 0.03 indicates the average RMSE, Blue arrow at 0.04 indicates the $\mu + \sigma$ and Red arrow at 0.05 indicates $\mu + 2 * \sigma$



(c) Autoencoder5 on Test3: RMSE values on test3. The network is trained with train3. validate3 as validation set and tested on test3. Green arrow at 0.02 indicates the average, Blue arrow indicates the $\mu + \sigma$ and Red arrow indicates $\mu + 2 * \sigma$

Figure 5.4: Performance of Autoencoders models on various test sets. Green arrow indicates the average ($\mu$) RMSE value for all test flights. Blue and Red arrows indicate $+1\sigma$ and $+2\sigma$ thresholds given by, $(\mu + 1 * \sigma)$ and $(\mu + 2 * \sigma)$ respectively, where $\sigma$ is standard deviation of RMSE values for all test flights.

## 5.4 Performance of Recurrent Neural Networks

### 5.4.1 Design of Networks

To evaluate the performance of Recurrent Neural Networks we have considered 20 different architectures of RNNs of which 10 are based on Long Short-Term Memory units (LSTM) and remaining 10 use Gated Recurrent Units (GRU). We vary the following parameters to generate various models of RNNs:

1. Number of iterations (epochs) training examples are presented to the network for learning
2. Number of hidden layers and number of hidden units in each layer
3. Number of time steps that RNNs are allowed to look into past
4. The dropout ratio which determines the percent of neurons randomly dropped at that layer before each iteration to improve generalization
5. Batch size which determines number of examples presented at a time to the network during training
6. Validation split which determines percent of *training examples* used to calculate the validation loss of the trained network

Throughout these models, we have used *adam* optimizer[46] with default arguments. The loss or cost function used is Mean Squared Error in all the models. In the input layer and output layers we have 21 neurons each. Thus the predicted feature vector for next time step is the output of the RNN. All the networks are designed and trained in Keras- a Theano based deep learning library. The details of RNN models using GRU are summarized in Table 5.4 and details of RNN models using LSTM are summarized in Table 5.5.

Table 5.4: Details of Parameter Combinations for Various GRU RNN Models

| Model | Timesteps | Dropout | Config | Batch Size | Epochs | Validation |
|---|---|---|---|---|---|---|
| GRU1 | 60 | 0.2 | 30 | 30 | 40 | 0.2 |
| GRU2 | 60 | 0.2 | 30 | 30 | 60 | 0.2 |
| GRU3 | 60 | 0.2 | 30 | 30 | 90 | 0.2 |
| GRU4 | 60 | 0.2 | 30 | 30 | 120 | 0.2 |
| GRU5 | 60 | 0.2 | 60 | 30 | 120 | 0.2 |
| GRU6 | 30 | 0.2 | 60 | 30 | 120 | 0.2 |
| GRU7 | 60 | 0.1, 0.1 | 30, 30 | 30 | 120 | 0.2 |
| GRU8 | 60 | 0.2, 0.2 | 30, 30 | 30 | 120 | 0.2 |
| GRU9 | 60 | 0.2, 0.2 | 30, 30 | 30 | 120 | 0.3 |
| GRU10 | 60 | 0.2, 0.2 | 60, 60 | 30 | 120 | 0.3 |

Table 5.5: Details of Parameter Combinations for Various LSTM RNN Models

| Model | Timesteps | Dropout | Config | Batch Size | Epochs | Validation |
|---|---|---|---|---|---|---|
| LSTM1 | 60 | 0.2 | 30 | 30 | 40 | 0.2 |
| LSTM2 | 60 | 0.2 | 30 | 30 | 60 | 0.2 |
| LSTM3 | 60 | 0.2 | 30 | 30 | 90 | 0.2 |
| LSTM4 | 60 | 0.2 | 30 | 30 | 120 | 0.2 |
| LSTM5 | 60 | 0.2 | 60 | 30 | 120 | 0.2 |
| LSTM6 | 60 | 0.2, 0.2 | 60, 60 | 60 | 40 | 0.2 |
| LSTM7 | 60 | 0.2, 0.2 | 30, 30 | 60 | 40 | 0.2 |
| LSTM8 | 90 | 0.2, 0.2 | 30, 30 | 60 | 40 | 0.2 |
| LSTM9 | 60 | 0.2, 0.2 | 30, 30 | 30 | 40 | 0.3 |
| LSTM10 | 60 | 0.2, 0.2 | 60, 60 | 30 | 40 | 0.3 |

### 5.4.2 Datasets

The dataset is split into 478 training examples and 22 test samples. Among 22 test samples, 11 are positive anomalous examples and other 11 are normal examples.

### 5.4.3 Methodology for Training RNN

We normalize the whole dataset similarly as done for autoencoders so that all the values range in [0,1]. In contrast to autoencoders, the normalized data need not be converted into high dimensional time series vectors of fixed size. The normalized feature vectors sampled at regular time intervals (per 2 secs) with 21 features per vector are presented to RNNs sequentially. We iterated through the training samples number of times as specified by epochs parameter. The output of the recurrent neural network at time t is predicted value at next time step(s) $t+1$. Thus the ideal output from RNN at time $t$ is the actual value (input) at time $t+1$.Thus in order to calculate the error during training, we can set the expected output at time $t$, $Y(t) = X(t+1)$, actual input from next timestep. It is important to note that since during training phase we have knowledge (both temporal and featural) about all the examples we can afford to follow this methodology to calculate the training error. Whereas during testing, we do not need $X(t+1)$ during time $t$. In fact, at time $t+1$, we calculate the error of value predicted at time $t$. Thus during testing if the resultant error is low, it signifies that the current value(s) are normal. On the other hand if the resultant error is relatively high, it indicates the presence of anomaly. It is interesting to note that, during online/realtime testing, as the network receives input values sequentially, ideally both point type and contextual anomalies can be detected.

### 5.4.4 Results

As shown in Figure 5.5 all RNN models are able to detect 8 out of 11 anomalous cases very comfortably. Anomalous flights with abnormal Pitch for short duration and second case with Runway Configuration Change were missed by all the models. The first case of Runway Configuration Change was a close call for all 20 models. Nonetheless, all models were able

to produce slightly higher MSE value for this case as shown in Table 5.6 and Table 5.7 thus marginally classifying it as anomaly.

Table 5.6: Performance of GRU RNN Models on 22 Test Instances (MSE Values)

| **Instance** | **GRU1** | **GRU2** | **GRU3** | **GRU4** | **GRU5** | **GRU6** | **GRU7** | **GRU8** | **GRU9** | **GRU10** |
|---|---|---|---|---|---|---|---|---|---|---|
| FDIR | 0.0449 | 0.0371 | 0.0297 | 0.0271 | 0.0198 | 0.0180 | 0.0270 | 0.0311 | 0.0321 | 0.0224 |
| FLAP | 0.0529 | 0.0465 | 0.0403 | 0.0380 | 0.0337 | 0.0322 | 0.0375 | 0.0413 | 0.0421 | 0.0354 |
| HENG | 0.0569 | 0.0510 | 0.0434 | 0.0396 | 0.0308 | 0.0282 | 0.0395 | 0.0432 | 0.0446 | 0.0351 |
| LENG | 0.0609 | 0.0525 | 0.0467 | 0.0431 | 0.0357 | 0.0330 | 0.0434 | 0.0487 | 0.0494 | 0.0381 |
| 1AP | 0.0904 | 0.0832 | 0.0774 | 0.0740 | 0.0665 | 0.0638 | 0.0731 | 0.0766 | 0.0781 | 0.0688 |
| PTCH | 0.0368 | 0.0300 | 0.0239 | 0.0214 | 0.0167 | 0.0161 | 0.0214 | 0.0242 | 0.0247 | 0.0188 |
| RW#1 | 0.0443 | 0.0358 | 0.0286 | 0.0254 | 0.0177 | 0.0155 | 0.0252 | 0.0296 | 0.0306 | 0.0204 |
| RW#2 | 0.0342 | 0.0280 | 0.0220 | 0.0185 | 0.0135 | 0.0136 | 0.0179 | 0.0215 | 0.0222 | 0.0164 |
| SHORT | 0.0565 | 0.0494 | 0.0441 | 0.0405 | 0.0328 | 0.0315 | 0.0409 | 0.0446 | 0.0455 | 0.0370 |
| VHSPD | 0.1391 | 0.1316 | 0.1297 | 0.1247 | 0.1174 | 0.1081 | 0.1288 | 0.1305 | 0.1300 | 0.1149 |
| WIND | 0.0463 | 0.0400 | 0.0340 | 0.0309 | 0.0227 | 0.0204 | 0.0303 | 0.0347 | 0.0359 | 0.0266 |
| Norm1 | 0.0365 | 0.0299 | 0.0242 | 0.0215 | 0.0146 | 0.0146 | 0.0205 | 0.0236 | 0.0256 | 0.0172 |
| Norm2 | 0.0348 | 0.0283 | 0.0227 | 0.0198 | 0.0144 | 0.0138 | 0.0194 | 0.0227 | 0.0242 | 0.0163 |
| Norm3 | 0.0354 | 0.0292 | 0.0232 | 0.0209 | 0.0142 | 0.0141 | 0.0196 | 0.0228 | 0.0248 | 0.0163 |
| Norm4 | 0.0373 | 0.0294 | 0.0234 | 0.0211 | 0.0153 | 0.0157 | 0.0204 | 0.0235 | 0.0250 | 0.0169 |
| Norm5 | 0.0357 | 0.0293 | 0.0235 | 0.0211 | 0.0148 | 0.0144 | 0.0200 | 0.0231 | 0.0250 | 0.0169 |
| Norm6 | 0.0349 | 0.0286 | 0.0229 | 0.0205 | 0.0147 | 0.0146 | 0.0192 | 0.0223 | 0.0243 | 0.0164 |
| Norm7 | 0.0347 | 0.0282 | 0.0224 | 0.0196 | 0.0143 | 0.0140 | 0.0193 | 0.0224 | 0.0239 | 0.0161 |
| Norm8 | 0.0350 | 0.0286 | 0.0230 | 0.0199 | 0.0144 | 0.0143 | 0.0194 | 0.0228 | 0.0245 | 0.0160 |
| Norm9 | 0.0370 | 0.0292 | 0.0233 | 0.0210 | 0.0154 | 0.0159 | 0.0203 | 0.0234 | 0.0249 | 0.0168 |
| Nrm10 | 0.0330 | 0.0275 | 0.0222 | 0.0195 | 0.0149 | 0.0154 | 0.0189 | 0.0218 | 0.0234 | 0.0166 |
| Nrm11 | 0.0378 | 0.0298 | 0.0243 | 0.0213 | 0.0156 | 0.0164 | 0.0213 | 0.0243 | 0.0255 | 0.0178 |

Table 5.7: Performance of LSTM RNN Models on 22 Test Instances (MSE Values)

| **Instance** | **LST1** | **LST2** | **LST3** | **LST4** | **LST5** | **LST6** | **LST7** | **LST8** | **LST9** | **LST10** |
|---|---|---|---|---|---|---|---|---|---|---|
| FDIR | 0.0213 | 0.0168 | 0.0136 | 0.0132 | 0.0099 | 0.0232 | 0.0345 | 0.0362 | 0.0354 | 0.0272 |
| FLAP | 0.0302 | 0.0253 | 0.0231 | 0.0230 | 0.0190 | 0.0351 | 0.0441 | 0.0491 | 0.0448 | 0.0379 |
| HENG | 0.0304 | 0.0248 | 0.0208 | 0.0199 | 0.0150 | 0.0309 | 0.0438 | 0.0284 | 0.0462 | 0.0357 |
| LENG | 0.0329 | 0.0263 | 0.0206 | 0.0201 | 0.0139 | 0.0348 | 0.0472 | 0.0549 | 0.0490 | 0.0395 |
| 1AP | 0.0952 | 0.1010 | 0.1019 | 0.1055 | 0.1081 | 0.0843 | 0.1023 | 0.0963 | 0.1007 | 0.0854 |
| PTCH | 0.0185 | 0.0150 | 0.0128 | 0.0126 | 0.0089 | 0.0190 | 0.0260 | 0.0293 | 0.0276 | 0.0218 |
| RW#1 | 0.0199 | 0.0156 | 0.0120 | 0.0119 | 0.0081 | 0.0215 | 0.0329 | 0.0351 | 0.0339 | 0.0257 |
| RW#2 | 0.0159 | 0.0124 | 0.0101 | 0.0100 | 0.0074 | 0.0160 | 0.0237 | 0.0196 | 0.0252 | 0.0187 |
| SHORT | 0.0321 | 0.0242 | 0.0213 | 0.0201 | 0.0150 | 0.0336 | 0.0456 | 0.0455 | 0.0470 | 0.0395 |
| VHSPD | 0.1080 | 0.0885 | 0.0719 | 0.0649 | 0.0384 | 0.1136 | 0.1243 | 0.1339 | 0.1255 | 0.1189 |
| WIND | 0.0240 | 0.0198 | 0.0179 | 0.0173 | 0.0132 | 0.0248 | 0.0363 | 0.0402 | 0.0386 | 0.0293 |
| Norm1 | 0.0172 | 0.0132 | 0.0113 | 0.0102 | 0.0079 | 0.0178 | 0.0275 | 0.0301 | 0.0292 | 0.0221 |
| Norm2 | 0.0167 | 0.0129 | 0.0115 | 0.0105 | 0.0078 | 0.0174 | 0.0262 | 0.0295 | 0.0278 | 0.0207 |
| Norm3 | 0.0168 | 0.0128 | 0.0109 | 0.0098 | 0.0078 | 0.0175 | 0.0264 | 0.0297 | 0.0282 | 0.0217 |
| Norm4 | 0.0170 | 0.0129 | 0.0112 | 0.0104 | 0.0077 | 0.0182 | 0.0289 | 0.0333 | 0.0301 | 0.0221 |
| Norm5 | 0.0171 | 0.0131 | 0.0111 | 0.0100 | 0.0079 | 0.0177 | 0.0265 | 0.0301 | 0.0282 | 0.0219 |
| Norm6 | 0.0171 | 0.0129 | 0.0111 | 0.0100 | 0.0079 | 0.0178 | 0.0258 | 0.0302 | 0.0276 | 0.0216 |
| Norm7 | 0.0167 | 0.0128 | 0.0115 | 0.0104 | 0.0079 | 0.0173 | 0.0262 | 0.0295 | 0.0278 | 0.0207 |
| Norm8 | 0.0168 | 0.0129 | 0.0115 | 0.0104 | 0.0078 | 0.0176 | 0.0257 | 0.0288 | 0.0277 | 0.0207 |
| Norm9 | 0.0169 | 0.0130 | 0.0113 | 0.0104 | 0.0078 | 0.0182 | 0.0288 | 0.0332 | 0.0300 | 0.0221 |
| Nrm10 | 0.0172 | 0.0130 | 0.0116 | 0.0106 | 0.0081 | 0.0179 | 0.0249 | 0.0298 | 0.0269 | 0.0207 |
| Nrm11 | 0.0175 | 0.0136 | 0.0120 | 0.0110 | 0.0081 | 0.0183 | 0.0279 | 0.0325 | 0.0294 | 0.0221 |

Out of 11 normal examples, all of them were successfully classified as negative by all LSTM and GRU RNN models. Hence RNNs gave zero false positives and thus resulting in high values *precision* equal to 1. Since majority of anomalies are classified correctly, they also resulted in high recall values and thus high $F_1$ scores. The details are presented in Tables 5.8 and 5.9. All models using RNNs with LSTM units as well as GRU units performed equally well. All the models with various configurations yielded similar results with high precision and high recall values. Figure 5.5 depicts reconstruction error values of first 4 LSTM and GRU models.

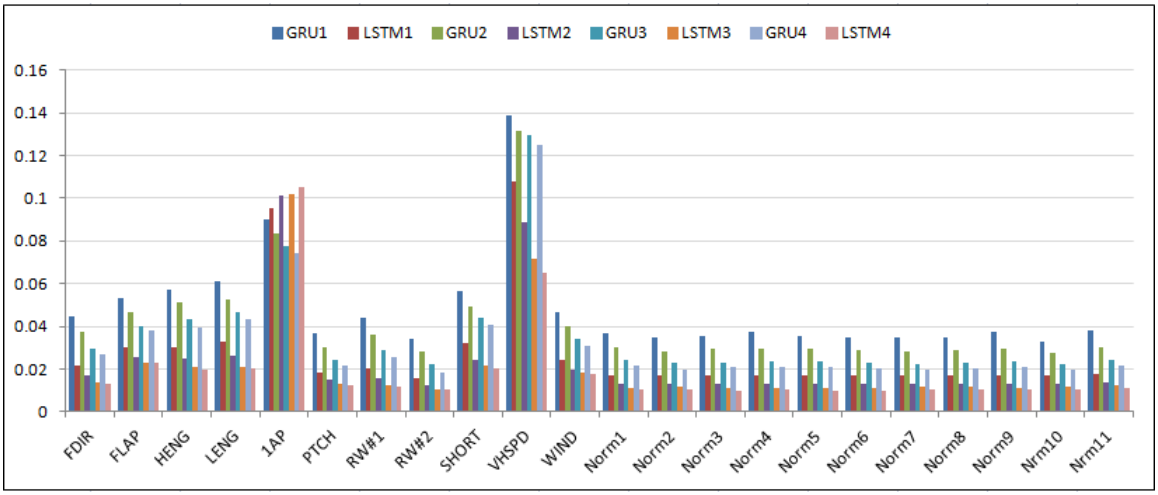



Figure 5.5: RNN: Reconstruction errors on test data (RMSE values)

## 5.5 Summary of Results

Table 5.8: Summay of results: Anomalies detected and missed by various MKAD, autoencoder and RNN models

| Anomaly | MKAD1 | MKAD2 | MKAD3 | Auto1 | Auto4 | Auto5 | LSTM | GRU |
|---|---|---|---|---|---|---|---|---|
| PTCH | No | No | No | No | No | No | No | No |
| FDIR | No | No | No | Yes | No | No | Yes | Yes |
| SHORT | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| HENG | No | Yes | No | Yes | Yes | Yes | Yes | Yes |
| FLAP | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| 1AP | Yes | Yes | Yes | Yes | No | No | Yes | Yes |
| RW1 | No | No | No | Yes | No | No | Yes | Yes |
| RW2 | No | No | No | Yes | Yes | Yes | No | No |
| VHSPD | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| WIND | Yes | No | No | No | Yes | Yes | Yes | Yes |
| LENG | No | Yes | No | Yes | Yes | Yes | Yes | Yes |

Autoencoders and RNNs are able to detect all the anomalous cases detected by MKAD and they are also able to detect some cases missed by MKAD. Since all RNN models yielded similar results instead of presenting individual performance for all models we presented the overall results for LSTMs and GRUs.

Table 5.9: Performance of MKAD, Autoencoders and RNNs

| **Model** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|
| **MKAD1** | **1** | 0.454 | 0.624 |
| **MKAD2** | **1** | 0.545 | 0.706 |
| **MKAD3** | **1** | 0.363 | 0.534 |
| **Auto1** | 0.6 | **0**.8181 | 0.691 |
| **Auto4** | 0.88 | 0.7272 | 0.8 |
| **Auto5** | 0.875 | 0.63 | 0.709 |
| **RNN-LSTM** | **1** | **0**.818 | **0**.899 |
| **RNN-GRU** | **1** | **0**.818 | **0**.899 |

Since $F1$ score considers both precision and recall values, it best represents the overall performance of the models. As shown in Table 5.9, RNN-LSTM and RNN-GRU outperformed both MKAD and autoencoders in terms of all three metrics. Though MKAD is able to achieve high precision values, as a result of false negatives their overall performance was poor. On the other hand, as autoencoders suffered from false positives their precision was poor but as were successful in detecting more anomalous cases than MKAD, their recall values and also overall performance was better.

# Chapter 6: Conclusion and Future Work

In this thesis the performance of autoencoders and RNNs in detecting anomalies in aircraft performance data was studied and it was compared with performance of MKAD algorithm. The models in this work were trained in semi-supervised fashion, wherein the negative class samples with only normal examples were used for training. Various autoencoders and RNN models were trained and their performance in terms of precision, recall and $F1$ score was compared with the performance of MKAD using various combinations of datasets. Data was collected by reproducing various anomalous and normal flights using adgPlugin developed for X-Plane simulation. Though using the current methodology autoencoders could not be used for real time anomaly detection, experimental results showed that they detected anomalies that MKAD was able to detect and also in addition detected some anomalies missed by MKAD. Recurrent Neural Networks, because of their better overall performance in detecting anomalies and their capability to handle multivariate timeseries data as input in its original form, they can be ideal candidates for online anomaly detection in aircraft data.

## 6.1 Future Work

As discussed, as part of future work, we plan to train autoencoders using k-fold cross validation. In this work we have collected and considered a fixed set of features in all the experiments. As part of future work, we plan to collect data for various other parameters and evaluate the performance of proposed models using various feature combinations. Also, as observed in the experiments, RNNs have missed identifying runway change configuration and abnormal pitch anomalies. Experiments with varying feature combinations may be valuable in assessing the performance of recurrent neural networks in detecting even the subtlest anomalies in the dataset.

# Bibliography

# Bibliography


[1] S. Das, B. L. Matthews, A. N. Srivastava, and N. C. Oza, "Multiple kernel learning for heterogeneous anomaly detection: Algorithm and aviation safety case study," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10, 2010, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/1835804.1835813

[2] S. Das, B. Matthews, and R. Lawrence, "Fleet level anomaly detection of aviation safety data," in *Prognostics and Health Management (PHM), 2011 IEEE Conference on*, June 2011, pp. 1–10.

[3] L. Li, M. Gariel, R. Hansman, and R. Palacios, "Anomaly detection in onboard-recorded flight data using cluster analysis," in *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, Oct 2011, pp. 4A4–1–4A4–11.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[5] M. F. Augusteijn and B. A. Folkert, "Neural network classification and novelty detection," *International Journal of Remote Sensing*, vol. 23, no. 14, pp. 2891–2902, 2002. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01431160110055804

[6] P. Sykacek, "Equivalent error bars for neural network classifiers trained by bayesian inference," in *In Proc. ESANN*, 1997, pp. 121–126.

[7] G. C. Vasconcelos, M. C. Fairhurst, and D. L. Bisset, "Investigating feedforward neural networks with respect to the rejection of spurious patterns," *Pattern Recogn. Lett.*, vol. 16, no. 2, pp. 207–212, Feb. 1995. [Online]. Available: http://dx.doi.org/10.1016/0167-8655(94)00092-H

[8] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[9] A. Sung and S. Mukkamala, "Identifying important features for intrusion detection using support vector machines and neural networks," in *Applications and the Internet, 2003. Proceedings. 2003 Symposium on*, Jan 2003, pp. 209–216.

[10] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection," *J. Mach. Learn. Res.*, vol. 6, pp. 211–232, Dec. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1046920.1058109

[11] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series, european symposium on artificial neural networks."

[12] R. C. Staudemeyer and C. W. Omlin, "Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data," in *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, ser. SAICSIT '13. New York, NY, USA: ACM, 2013, pp. 218–224. [Online]. Available: http://doi.acm.org/10.1145/2513456.2513490

[13] B. Amidan, A. Swickard, R. Allen, and F. T. A., "Identifying in-close-approach-changes in air traffic control (atc) data," 2002.

[14] T. R. Chidester, "Understanding normal and atypical operations through analysis of flight data," in *In Proceedings of the 12th International Symposium on Aviation Psychology*, 2003.

[15] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 29–38. [Online]. Available: http://doi.acm.org/10.1145/956750.956758

[16] D. L. Iverson, "Inductive system health monitoring," in *In Proceedings of The 2004 International Conference on Artificial Intelligence (IC-AI04), Las Vegas.* CSREA Press, 2004.

[17] S. Budalakoti, S. Budalakoti, A. Srivastava, M. Otey, and M. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 39, no. 1, pp. 101–113, Jan 2009.

[18] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 6–. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015424

[19] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *J. Mach. Learn. Res.*, vol. 5, pp. 27–72, Dec. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1005334

[20] P. Patel, E. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, 2002, pp. 370–377.

[21] D. M. J. Tax and R. P. W. Duin, "Support vector domain description," *Pattern Recognition Letters*, vol. 20, pp. 1191–1199, 1999.

[22] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001. [Online]. Available: http://dx.doi.org/10.1162/089976601750264965

[23] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, 2011.

[24] K. Hornik, "Some new results on neural network approximation," *Neural Netw.*, vol. 6, no. 8, pp. 1069–1072, Jan. 1993. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(09)80018-X

[25] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1162/neco.2006.18.7.1527

[26] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1561/2200000006

[27] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2012, vol. 385. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24797-2

[28] H. G. Zimmermann, R. Grothmann, A. M. Schaefer, and Ch, "Identification and forecasting of large dynamical systems by dynamical consistent neural networks," in *New Directions in Statistical Signal Processing: From Systems to Brain*, S. Haykin, J. Principe, T. Sejnowski, and J. Mcwhirter, Eds. MIT Press, 2006, pp. 203–242.

[29] A. J. Robinson and F. Fallside, "The utility driven dynamic error propagation network," Cambridge University Engineering Department, Cambridge, Tech. Rep. CUED/F-INFENG/TR.1, 1987.

[30] R. J. Williams and D. Zipser, "Backpropagation," Y. Chauvin and D. E. Rumelhart, Eds. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995, ch. Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pp. 433–486. [Online]. Available: http://dl.acm.org/citation.cfm?id=201784.201801

[31] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[32] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[33] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

[34] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural Netw.*, vol. 3, no. 1, pp. 23–43, Jan. 1990. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(90)90044-L

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[36] S. Hochreiter, M. Heusel, and K. Obermayer, "Fast model-based protein homology detection without alignment," *Bioinformatics*, vol. 23, no. 14, pp. 1728–1736, 2007. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/23/14/1728.abstract

[37] J. Chen and N. S. Chaudhari, "Protein secondary structure prediction with bidirectional lstm networks," in *Post-Conference Workshop on Computational Intelligence Approaches for the Analysis of Bio-data (CI-BIO)*, Montreal, Canada, August 2005.

[38] D. Eck and J. Schmidhuber, "Finding temporal structure in music: blues improvisation with lstm recurrent networks," in *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, 2002, pp. 747–756.

[39] B. Bakker, "Reinforcement learning with long short-term memory," in *In NIPS*. MIT Press, 2002, pp. 1475–1482.

[40] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, pp. 5–6, 2005.

[41] A. Graves, S. Fernndez, and F. Gomez, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *In Proceedings of the International Conference on Machine Learning, ICML 2006*, 2006, pp. 369–376.

[42] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *In Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[43] A. Graves, H. Bunke, S. Fernndez, M. Liwicki, and J. Schmidhuber, "Unconstrained online handwriting recognition with recurrent neural networks," in *in Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

[44] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: http://arxiv.org/abs/1409.1259

[45] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

# Curriculum Vitae

Anvardh Nanduri received his Bachelor of Technology in Information Technology from Jawaharlal Nehru Technological University, India in 2011. Before pursuing his masters, he was with Honeywell Technology Solutions, Bangalore, where he was a developer for Next Generation Flight Management System for over two years. He has been a Research Assistant in Center for Air Transportation Systems Research, George Mason University for past two years.