Multivariate Aviation Time Series Modeling: VARs vs. LSTMs

Hardik Goel* Igor Melnyk† Nikunj Oza‡ Bryan Matthews‡ Arindam Banerjee*

Abstract

Multivariate time-series modeling and forecasting constitutes an important problem with numerous applications. In this work, we consider multivariate continuous time series modeling from aviation, where the data consists of multiple sensor measurements from real world flights. While traditional approaches such as VAR (vector auto-regressive) models have been widely used for aviation time series, recent years have seen significant advances in modeling sequences using LSTMs (long short term memory models). In this paper, we do a careful empirical comparison between VAR and two types of LSTMs on multivariate aviation time series. Surprisingly, VAR is seen to significantly outperform LSTMs on real flight data, as well as synthetic data generated from VAR models and LSTM models. The results suggest that VAR is more suitable for multivariate continuous data especially from aviation where there may not be much utility for modeling long term memory which is a strength of LSTMs.

1 Introduction

Multivariate time-series modeling and forecasting constitutes an important problem with numerous applications in several real-world domains such as healthcare, finance, climate, and aviation [15, 20, 29]. In the aviation industry, the United States alone can expect an increase of more than 60% in its commercial air traffic the over the next two decades [2]. Such growth can lead to increased congestion on the ground and in the air, creating safety issues leading to possible accidents. In response to this problem, air transportation authorities are focusing their efforts towards enhancing existing air traffic control system, in part by improving the processing and analysis of the flight information (known as Flight Operations Quality Assurance (FOQA) data [3]) to find unsafe flight patterns, detect aircraft operation issues, identify pilot-aircraft interaction problems, etc. Accurate modeling and prediction of the FOQA data is therefore of great importance and is the motivation behind the current work. Each flight is represented as a *multivariate, continuous time-series*, describing the time evolution of many on-board sensors sampled at 1 Hz, and has a duration varying from 30 minutes to several hours. The major challenge with modeling such data lies in capturing the proper temporal variations for each flight sensor as well as the possible correlations between them at the each time step.

In recent years, several model-based approaches were proposed for the analysis of the flight data. For example, [23] proposed a dynamic Bayesian network to model the continuous flight sensors as well as discrete pilot commands. In [11] the authors used a specially designed linear regression model to describe the aerodynamic forces acting on an aircraft, and [25] similarly used regression approach to model and predict fuel consumption in jet engines. More recently, the work in [21, 20] proposed several Vector Auto-Regressive (VAR)-based approaches to model the FOQA data and used them to detect abnormal activities in the flights. In parallel, significant recent advances have been made in modeling sequential data using Recurrent Neural Networks (RNNs). In particular, the Long-Short Term Memory (LSTM) model, an extension of RNN, has shown great promise in several tasks [12, 28]. However, LSTMs have not been carefully explored as an approach for modeling multivariate aviation time series.

In this paper, we do a careful empirical comparison between VAR and LSTMs for modeling multivariate aviation time series. We consider two different LSTM architectures (see Sections 3.2 and 3.3) and compare their performance with VAR for making one-step-ahead and multi-step-ahead predictions. Through experiments on real flight data, we find that VAR significantly outperforms both types of LSTMs for both one-step- and multi-step-ahead predictions. We also evaluate their performance on synthetic datasets generated from VAR and LSTM models, and VAR outperforms LSTMs on both types of synthetic datasets. In spite of the popularity of LSTMs in recent years, the findings illustrate that in certain domains traditional linear models such as VAR can still significantly outperform LSTMs. The results are somewhat consistent with the findings of Gers et al. in [9], where they found that traditional, sim-

*Department of Computer Science and Engineering, University of Minnesota, MN {goelx033, banerjee}@cs.umn.edu

†IBM Research, T. J. Watson Research Center, Yorktown Heights, NY igor.melnyk@ibm.com

‡NASA Ames Research Center, Moffett Field, CA {nikunj.c.oza,bryan.l.matthews}@nasa.gov

pler approaches can outperform LSTM in predicting the Mackey-Glass time-series [18]. Our results indicate that more careful study of LSTMs, especially their utility in modeling multivariate continuous time-series, is needed. Further, since LSTMs are highly non-linear models with several choices for their architecture, identifying a suitable architecture for a problem domain can be more challenging compared to VAR models. Finally, the relative performance of VAR and LSTMs may have to do with properties of aviation data. While one of the main strengths of LSTMs is the ability to model long term dependencies, multivariate flight sensor data may not have such long term dependencies, making LSTMs unsuitable for the domain.

The rest of the paper is organized as follows. In Section 2 we discuss the related work of VAR- and LSTM-based methods in aviation and other domains. The standard LSTM and the proposed two architectures are given in Section 3, while the VAR model is shown in Section 4. Experiments are presented in Sections 5 and 6; and in Section 7 we conclude the paper.

2 Related Work

In this section we review the relevant literature for the two broad approaches for modeling time series considered in this paper: Vector AutoRegressive (VAR) and Long-Sort Term Memory (LSTM) models.

2.1 Vector Auto-Regressive Models (VARs) VAR models [17] arguably are the most widely used family of multivariate time series statistical approaches. These models have been applied in a wide variety of applications, ranging from describing the behavior of economic and financial time series [31] to modeling dynamical systems [16] and estimating brain function connectivity [32], among others. Recent work has successfully applied VAR models to analyze multivariate aviation time series data, i.e., multiple sensor measurements for flights [19, 21, 20]. By modeling the time evolution of multiple on-board sensors, these approaches aimed at accurate prediction of aircraft behavior to discover anomalous events from a large dataset of flight information. Extensive experimental evaluations illustrated that VAR models have good performance in modeling flight data and detecting various types of anomalies, outperforming the existing state-of-the-art approaches.

2.2 Long Short Term Memory Models (LSTMs) LSTM is a special kind of Recurrent Neural Network (RNN), originally introduced by Hochreiter & Schmidhuber [13]. Because of their ability to learn long term patterns in sequential data, they have recently been applied to diverse set of problems, including handwriting recognition [12], machine translation [28] and many others.

The simplest LSTM model that takes in a sequence of inputs and produces a single output (i.e., performing 1-step-ahead prediction) has found its use mainly in sequence classification tasks such as text classification [5] or in medical diagnosis [15]. The common property of these problems is that the data is a discrete sequence; the continuous time series modeling has not found its way in much of LSTM work. Few examples are the work of [34] for emotion recognition or [33] for online music mood regression, where such models were applied on the continuous-valued datasets. Apart from this, the work of [24] has used LSTM-based prediction model on the Mackey Glass time-series, achieving promising results. Similar to the above work, in this paper we use the standard LSTM model (see Section 3.3) on continuous-valued data to make a one-step-ahead prediction, given the past sequence of inputs.

Another LSTM-based model that we explore in this work is the sequential encoder-decoder, which was first introduced in [4], where they used traditional RNN and applied it on the task of statistical machine translation. Similar work, but based on LSTM, was also done in [28] and [26], achieving good results. We also found that using encoder-decoder architecture is more suitable for multiple-step-ahead prediction as opposed to using the standard model, treating its own current output as input at the next step and causing inaccurate predictions due to error accumulation.

In the aviation domain, very limited work was done in applying LSTMs to the flight data. An exception is the work of [8], where they explore the use of such models for modeling aircraft engine vibrations. The authors explore different parameters for LSTMs and compare the results. However, the presented results were not compared against any other baseline models; moreover, it was not clear if the dataset was similar to FOQA, as is used in the current work.

3 Long Short Term Memory Models

In this section, we give an introduction to the LSTM model and discuss its two specific variants, viz. the standard LSTM (LSTM-STD) and LSTM Sequential Encoder-Decoder (LSTM-SED) models.

3.1 From RNNs to LSTMs We start by reviewing the standard Recurrent Neural Network (RNN), followed by the discussion of the key shortcoming in RNNs, namely the vanishing/exploding gradient problem, limiting the ability to learn long-term data dependencies and present the LSTMs, which address this challenge.

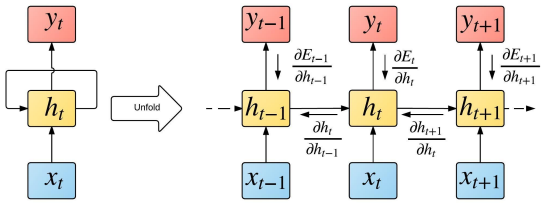


Figure 1: Recurrent Neural Network (RNN). The left side of the figure illustrates the feedback loop of the model, where x_t, h_t and y_t are the input, hidden state and output. The right side shows RNN unfolded in time, which also illustrates the flow of the gradients during the backpropagation step of the training; E_t is the error in y_t at time t.

3.1.1 Why LSTMs? Given a time-series data $x_1, x_2, \ldots, x_t$, the RNN is defined by the following recurrent relation

$$h_t = \sigma(Wx_t + Uh_{t-1} + b), \tag{3.1}$$

where, $x_t \in \mathbb{R}^d$ is the input at time t, $W \in \mathbb{R}^{n\times d}$, $U \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ are the hidden state parameters, h_t and $h_{t-1} \in \mathbb{R}^n$ are the hidden state vectors at times t and $t-1$, respectively, and σ is the logistic sigmoid function. Figure 1 shows an example of RNN model, whose main part is the feedback loop, generating the hidden state h_t at time step t from the current input x_t and previous hidden state h_{t-1}.

The main issue with RNN model is that during training, while applying the backpropagation-through-time (BPTT) algorithm, the error gradients can quickly vanish/explode [22]. This is due to the application of the chain rule of the differentiation at each time step. The gradient of the hidden state at time t with respect to the hidden state at some step $t-k$ is a product of the form $\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$. If the gradient at each time step is greater than or less than 1, the product of such values can grow or vanish exponentially fast. This prevents the network from capturing the long-term dependencies in the data. LSTMs, an extension of RNNs, were introduced as a mechanism to prevent these issues by backpropagating a constant error gradient.

3.1.2 Inside LSTMs To address the vanishing/exploding gradient problem of RNN, the LSTM defines a new hidden state, called a cell. Each cell has its own *cell state*, which acts like a memory, and various control mechanisms, called gates, enable modification of the cell memory. Through the use of such mechanisms, the LSTM can effectively learn when to forget the old memories (*forget gate*), when to add new memories from the current input (*input gate*) and what memories to present as output from the current cell (*output gate*).

Each gate is a single layer neural network whose weights are the extra parameters to be learned during training. The gates have sigmoidal activation in their outputs, squashing the output to $[0, 1]$ range. These values indicate, as fractions, how much reading, writing or forgetting to perform, giving increased learning power to the model. More detailed description of the LSTM is presented below.

An LSTM cell, at time t, receives a *cell state* $c_{t-1} \in \mathbb{R}^n$ and $h_{t-1} \in \mathbb{R}^n$, from the previous time step $t-1$, where n is the dimension of the hidden state. The cell also receives the current input $x_t \in \mathbb{R}^d$. The value of the *forget gate* is then calculated as

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{3.2}$$

where $W_f \in \mathbb{R}^{n\times d}$, $U_f \in \mathbb{R}^{n\times n}$, $b_f \in \mathbb{R}^n$ are the *forget gate* parameters and $f_t \in \mathbb{R}^n$ is the output of the gate. This output determines what to erase from the previous *cell state*. The *input gate* is also applied to determine what parts of the current input are to be added to the *cell state*. This is accomplished with the following two operations

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.3}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{3.4}$$

where $W_i \in \mathbb{R}^{n\times d}$, $U_i \in \mathbb{R}^{n\times n}$, $b_i \in \mathbb{R}^n$ are the *input gate* parameters and $i_t \in \mathbb{R}^n$ is the output of the gate in (3.3) and $W_c \in \mathbb{R}^{n\times d}$, $U_c \in \mathbb{R}^{n\times n}$, $b_c \in \mathbb{R}^n$ in (3.4) are the parameters for selecting a *candidate state*, $\tilde{c}_t \in \mathbb{R}^n$. The candidate state along with the input and forget gates together determine the current state of the cell, $c_t \in \mathbb{R}^n$, according to

$$c_t = i_t \odot \tilde{c}_t + f_t \odot c_{t-1}, \tag{3.5}$$

where $\odot$ represents the element-wise Hadamard product. The final step is the *output gate*, which determines the output from the current cell state

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.6}$$

$$h_t = o_t \odot \tanh(c_t), \tag{3.7}$$

where $W_o \in \mathbb{R}^{n\times d}$, $U_o \in \mathbb{R}^{n\times n}$, $b_o \in \mathbb{R}^n$ are the *output gate* parameters and $o_t \in \mathbb{R}^n$ is used to calculate the output $h_t \in \mathbb{R}^n$ in (3.7). Note that h_t itself can be used as the final output of the network. If h_t does not have the desired dimension, one can use a fully connected feed-forward neural network to convert it to an output y_t with the desired dimensions. Figure 2 illustrates the entire flow inside the LSTM cell.

3.1.3 LSTMs vs. RNNs Having seen the architecture of an LSTM cell, we briefly discuss why it addresses the vanishing/exploding gradient problem, introduced in Section 3.1.1. The main reason LSTM does not suffer from the vanishing gradient problem is because *cell*

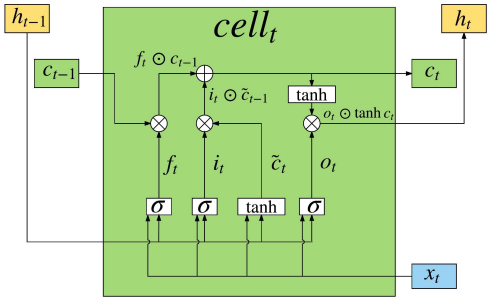


Figure 2: An LSTM cell at time t. The color mapping is similar to Figure 1, showing that h_t here is equivalent to hidden state h_t of RNN and c_t is a new component that is added to the LSTM cells. The h_t here can be used as the cell output or converted using a feed-forward connection to an output y_t. The σs represent the gates of the cell, which are single layer neural networks with sigmoidal activation.

state c_t is updated using only addition operation, as opposed to RNN, which involves a sigmoidal transformation (3.1). During backpropagation-through-time, the partial derivative $\frac{\partial c_t}{\partial c_{t-1}}$ depends on the value of the *forget gate* f_t, as can be seen in (3.5). This means that if the *forget gate* always outputs a vector of 1s, only a constant error is propagated back to every step. Similarly, LSTM does not suffer from exploding gradient because the sigmoidal *forget gate* cannot output values greater than 1. Compared to RNN, where the error gradient was multiplied by the parameter matrix and a sigmoidal derivative at each time step, it is evident that LSTM is able to learn long-range data dependencies.

3.2 Standard LSTM (LSTM-STD) The first LSTM architecture used in this work is the standard LSTM model, that takes a fixed-length sequence of vectors $x_1, x_2, \ldots, x_t \in \mathbb{R}^d$ as input and generates a single output vector $\hat{x}_{t+1} \in \mathbb{R}^d$ using a final feed-forward connection on top of h_t. The vector $\hat{x}_{t+1}$ represents the model's prediction for the future. In the next step, it takes as input the next sequence from the data x_2, ... x_{t+1} and generates $\hat{x}_{t+2}$ as output. We refer to this model as LSTM-STD, since it is the basic LSTM model without any add-ons. Equation (3.8) shows the relation between $\hat{x}_{t+1}$ and h_t

$$\hat{x}_{t+1} = \sigma(W_{ff} h_t + b_{ff}), \tag{3.8}$$

where $W_{ff} \in \mathbb{R}^{d \times n}$ and $b_{ff} \in \mathbb{R}^d$ are the parameters of the final fully-connected, feed-forward layer and $h_t \in \mathbb{R}^n$ can be generated using x_t, c_{t-1} and h_{t-1} as presented in Section 3.1.2.

The same model is used for 5-step-ahead prediction as well. The first input is a data sequence $x_1, x_2, \ldots, x_t \in \mathbb{R}^d$ and the output is $\hat{x}_{t+1} \in \mathbb{R}^d$. The model then takes in the sequence $x_2, \ldots, x_t, \hat{x}_{t+1}$ to generate the output $\hat{x}_{t+2}$. Thus the model uses its own output as part of the next input to predict any number of steps into the future. The LSTM-STD model is illustrated in Figure 3.

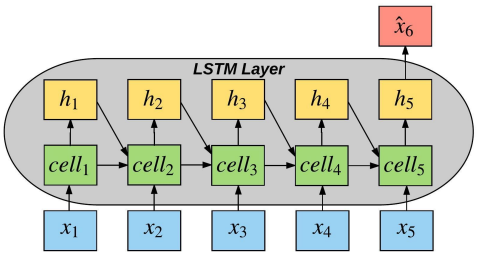


Figure 3: LSTM-STD model. Blue boxes (bottom layer) are the data inputs, green boxes (second from below) are the LSTM cells, yellow boxes (third from below) represent the cell outputs, and finally the red box is the predicted output. We also added a dense feed-forward layer at the final output, converting the high-dimensional h_5 to $\hat{x}_6$ of input-dimension d. Also, note that a network with an input sequence length of 5 is shown here just for illustration (in experiments, we used input sequence length of 10).

3.3 LSTM Sequential Encoder-Decoder (LSTM-SED) The second LSTM architecture that we used is a sequential autoencoder-based model, inspired from the work of Srivastava et al. [26]. This model consists of two LSTMs, where the first one (called Encoder LSTM) works like the LSTM-STD model, presented in Section 3.2. It takes in a fixed-length sequence of vectors $x_1, x_2, \ldots, x_t \in \mathbb{R}^d$ as input and produces the output $h_t \in \mathbb{R}^n$, where n is the hidden state dimension. We call this vector h_t, the encoded version of the sequential input. This vector is then fed as input to the second LSTM (called Decoder LSTM) at each time-step. The decoder LSTM generates a sequence of outputs $\hat{x}_{t+1}, \hat{x}_{t+2}, \ldots, \hat{x}_{t+k} \in \mathbb{R}^d$ that represent the model prediction, where k is the number of predicted steps. The LSTM-SED model is illustrated in Figure 4.

There are certain similarities and differences between our model and the ones that [26] have used, which are worth mentioning. Our model is the same as theirs in that both of them accept sequential input data, produce encoded representation and then use it to make predictions. The difference, however, is in the way the encoded representation is used by the decoder. [26] handle it in two different ways. In the first one, they provide the encoded vector as input only to the first cell of the decoder LSTM, while the rest of the cells do not receive any input (*unconditional* model). In the other variant, the encoded vector is provided as input only to the first cell of the decoder LSTM, while for the remaining cells the output generated from the previous decoder cell is used as input (*conditional* model). They present arguments for and against both of these models, an im-

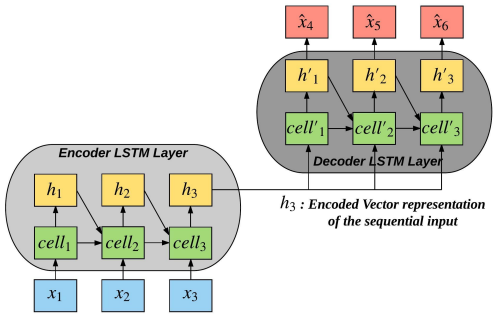


Figure 4: LSTM-SED model. Blue boxes are the data inputs. The left block is the Encoder LSTM, and the right block is the Decoder LSTM. The Encoder LSTM produces the encoded vector h_3, which is fed in as inputs to Decoder LSTM. Fully-connected, feed-forward networks are then used to convert each cell's output (yellow boxes in Decoder LSTM) into the final predictions $\hat{x}$s (red boxes at the top). Note that a network with input and output sequence length of 3 is shown here just for illustration (in experiments, we used input and output sequence lengths of 10 and 5).

portant one being that conditioning the decoder cells on the previous cell's output does not force it to look deep inside the encoder for valuable information. In our case, once we generate the encoded representation, we send this vector as input to *each* cell of the decoder. This ensures that the model only uses the actual inputs $x_1, \ldots, x_t$ to predict the outputs $\hat{x}_{t+1}, \ldots, \hat{x}_{t+k}$. This is important because the errors in prediction do not accumulate. From the learning point of view, providing the encoded representation as input at each step of the decoder helps it learn by exploiting short-range correlations, rather than trying to use the encoded representation from k steps ago.

4 Vector Autoregressive Models

In this Section we present the details of the Vector AutoRegressive model and discuss the method to preform parameter estimation. VAR is a popular statistical approach to model linear dependencies among multiple features that evolve in time [16, 17]. In general form, the k-th order VAR model can be written as

$$x_t = A_1 x_{t-1} + \ldots + A_k x_{t-k} + \epsilon_t, \tag{4.9}$$

where $A \in \mathbb{R}^{d \times d}$ are the matrices of coefficients, $x \in \mathbb{R}^d$ is the vector of parameters, $\epsilon \in \mathbb{R}^d$ is the zero-mean white noise, and $t = \max(k+1), \ldots, T$, where T denotes the length of time series. The subscript k determines the lag of the model, i.e., the degree to which the data in the current time step depends on the data in the past.

To estimate the VAR parameters, the model in (4.9) is usually transformed into the form suitable for a least-square estimator. Specifically, let $(x_0, x_1, \ldots, x_T)$ denote the $T+1$ samples generated by the stable VAR model in (4.9), then stacking them together we obtain

$$\begin{bmatrix} x_k^T \\ x_{k+1}^T \\ \vdots \\ x_T^T \end{bmatrix} = \begin{bmatrix} x_{k-1}^T & x_{k-2}^T & \ldots & x_0^T \\ x_k^T & x_{k-1}^T & \ldots & x_1^T \\ \vdots & \vdots & \ddots & \vdots \\ x_{T-1}^T & x_{T-2}^T & \ldots & x_{T-k}^T \end{bmatrix} \begin{bmatrix} A_1^T \\ A_2^T \\ \vdots \\ A_k^T \end{bmatrix} + \begin{bmatrix} \epsilon_k^T \\ \epsilon_{k+1}^T \\ \vdots \\ \epsilon_T^T \end{bmatrix}$$

which can also be compactly written as

$$Y = XB + E, \tag{4.10}$$

where $Y \in \mathbb{R}^{N \times p}$, $X \in \mathbb{R}^{N \times kp}$, $B \in \mathbb{R}^{kp \times p}$, and $E \in \mathbb{R}^{N \times p}$ for $N = T - k + 1$. Vectorizing (column-wise) each matrix in (4.10), we get

$$\begin{aligned} \mathrm{vec}(Y) &= (I_{p \times p} \otimes X)\mathrm{vec}(B) + \mathrm{vec}(E) \\ \mathbf{y} &= Z\boldsymbol{\beta} + \boldsymbol{\epsilon}, \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^{Np}$, $Z = (I_{p \times p} \otimes X) \in \mathbb{R}^{Np \times kp^2}$, $\boldsymbol{\beta} \in \mathbb{R}^{kp^2}$, $\boldsymbol{\epsilon} \in \mathbb{R}^{Np}$, and $\otimes$ is the Kronecker product. Consequently, the least-squares estimator then takes the form

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{kp^2}}{\mathrm{argmin}} \frac{1}{N} ||\mathbf{y} - Z\boldsymbol{\beta}||_2^2. \tag{4.11}$$

Note that if the domain problem entails certain structure in the paramaters A, e.g., sparsity, low-rank structure, etc., then the estimator (4.11) can be regularized with different regularization norms, e.g., L_1 norm, nuclear norm [14], etc. Note that matrix Z in (4.11) can become very tall in cases when the length of time series is large. The standard approaches of estimating $\boldsymbol{\beta}$, based on regular QR decomposition [10], become impractical. For this purpose, in practice, we use the approach of [6] based on Tall and Skinny QR (TSQR), which enables to perform QR of a tall matrix in a block-by-block manner.

5 Experimental Setup

Our experiments focus on comparing VAR and LSTM models for multivariate time-series prediction on both the real aviation data as well as synthetic datasets. In this section we discuss the different architectures considered and the datasets used.

5.1 Model Parameters and Evaluation Metrics

VAR. For the VAR model, we used an input sequence of length 10 (i.e., 10th order VAR), which means that for $d = 42$ dimensional input (this is the number of continuous-valued sensors we used in the flight datasets, as described in Section 5.2), it would have $10d^2 = 17,640$ parameters to learn.

LSTM-STD. For the standard LSTM model, we again used an input sequence length of 10 and 1 hidden layer with $n = 64$ units in each LSTM cell. The hidden

layer was used with the ReLu non-linearity. For the final target prediction, a dense feedforward connection was used to transform the 64-dimensional hidden layer output into a 42-dimensional vector. Sigmoidal units were used as the final non-linearity. In all, this makes for about 30,442 parameters. More about the reason for selecting these specific parameters is discussed in Section 6.1.3.

LSTM-SED. For the encoder-decoder LSTM model, the encoder had the same parameters as the standard model (LSTM-STD) above. However, instead of having a dense connection at the output, the 64-dimensional vector was fed into the decoder LSTM. This LSTM used a sequence length of 5 and 1 hidden layer with $n = 64$ units in each cell. It produced five 64-dimensional output vectors that were transformed through dense, feedforward connections into five 42-dimensional final prediction vectors. This model contains about 74,706 parameters. Due to the model's complexity, it takes more time to train LSTM-SED for the same number of epochs as in the case of LSTM-STD.

All of the above models were then used to make predictions for the data in the test set. Given 10 time steps of data as input at each step, the models performed one-step- or five-step-ahead predictions, which were evaluated against the true values based on the *root mean squared error (RMSE)*. The RMSE was measured for each test file and their mean was reported as the evaluation metric.

We used the deep learning library Lasagne [7], which is built on top of Theano [30], for training the LSTM models. The mean training/test set errors were tracked at each epoch. The training was done until the test set error had largely flattened and no further gains were noticed.

5.2 Datasets We used one real and two synthetic datasets for our experiments.

Real Flight Data. The real-world data is the Flight Operations Quality Assurance (FOQA) dataset from NASA [1]. This data contains air traffic flight sensor information and is being used in the research community to detect issues in aircraft operation due to mechanical, environmental or human factors [20, 27]. The data contains over a million flights, each having a record of about 300 (multivariate) time series measurements, sampled at 1 Hz over the duration of the flight. These parameters include both discrete and continuous readings from control switches (like thrust, autopilot, flight director, etc.) and sensors (like altitude, angle of attack, drift angle, etc.). For our experiments, we selected 42 features representing all the continuous sensor measurements from 110 flights of the same type of aircraft, landing at the same airport. The focus was on a portion of the flight below $10,000$ feet until touchdown (duration 600-1500 timestamps), which makes for about 77,000 time-steps of data in all. We then split it into train-test sets with a ratio of 10:1.

Synthetic VAR-generated data. A 10th order VAR model was used to generate data similar to the real flight data with 42 continuous variables. We initialized the model with parameters learned from the real flight data. This model was then fed with 10 random initial inputs and was made to predict the next values, using its own output as inputs in successive steps. The initial 50 values were discarded as burn-in time, and the rest 700 steps were considered one flight. In all, data corresponding to about 77,000 timesteps (≈ 110 flights) was generated. This was then split into train-test sets with a ratio of 10:1.

Synthetic LSTM-generated data. Similar to VAR data above, an LSTM with an input sequence length of 10 was used to generate synthetic data that resembled the real flight data. This LSTM was also initialized with parameters learned from the real data and generated new data in the same way as VAR above.

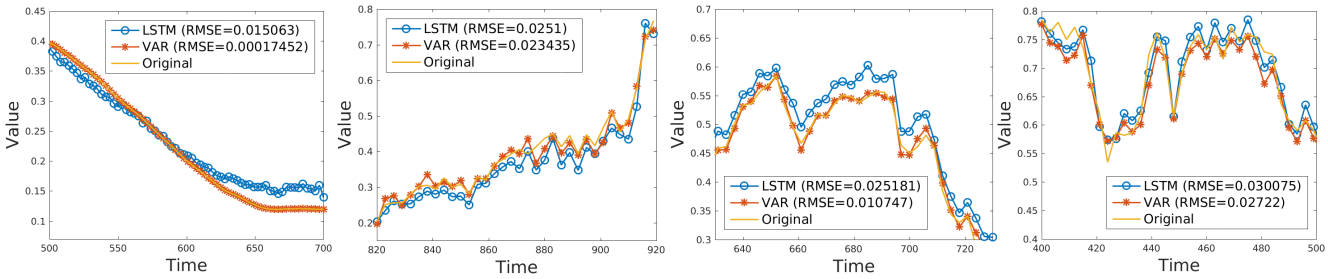
The data from all the above datasets was normalized using "min-max" scaling on a per-feature basis to lie in the interval $[0, 1]$, i.e., $x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$.

6 Experimental Results

6.1 Aviation data We start with a comparison of VAR and LSTM on the FOQA dataset with model parameters as discussed in Section 5.

6.1.1 One-step-ahead Prediction The one-step-ahead predicted values from VAR and LSTM-STD on the test set are shown in Figure 5 for 4 out of 42 selected features. Qualitatively, VAR performs slightly better than LSTM-STD for all the features, closely modeling the data in both the smooth and highly oscillating scenarios. It is also interesting to note that if LSTM-STD overshoots the high peaks in the data oscillations, it tends to keep overshooting (Figure 5c); similarly, if it undershoots the low peaks, it keeps undershooting (Figure 5b). For a quantitative comparison, the average RMSE on the test sets are shown in Table 1 (first column)—VAR achieves a lower RMSE which is statistically significant based on a Kolmogorov-Smirnov (K-S) test.

6.1.2 Multi-step-ahead Prediction We tested three models to perform multi-step-ahead prediction into the future, given the past 10 inputs. The first model is the same 10th order VAR model that was trained earlier in section 6.1.1, i.e., use $x_1, \ldots, x_{10}$ as input to predict $\hat{x}_{11}$. Subsequently, we used inputs $x_2, \ldots, x_{10}, \hat{x}_{11}$ to predict $\hat{x}_{12}$, and so on. The second model is also the



(a) Altitude (b) Angle Of Attack 1 (c) Drift Angle (d) Flight Path Acceleration

Figure 5: Plots of portions of selected features from a file in the original aviation data and the 1-step predictions made by LSTM-STD and VAR. Note that the x-axis and y-axis differ in scale for each feature because different zoomed-in sections of the same flight are shown for clarity. VAR follows the original data more closely as compared to LSTM. In the first figure, VAR has completely overlapped the original data. Also, notice that LSTM consistently overshoots/undershoots the peaks.

same LSTM-STD model from section 6.1.1, but made to predict multiple steps into the future using the immediate predicted values similar to the VAR model. The third model is the LSTM-SED model that uses inputs $x_1,\ldots,\ x_{10}$ to predict $\hat{x}_{11},\ldots,\ \hat{x}_{10+k}$ at once for k-step prediction. We expected VAR and LSTM-STD to make more errors as the number of future steps increases because the models use their own outputs as inputs in the next step, which can cause the errors to accumulate and grow. Since LSTM-SED is trained to use only the true inputs to predict all the future steps at once, it may not suffer from error accumulation. However, LSTM-SED has the representational challenge of encoding all the information from data input sequence into one vector of limited dimension, and use only this vector to predict multiple future steps.

The complete set of results are presented in Table 1, along with error bar plots of RMSE (Standard Deviation) as a function of delay in Figure 6. Surprisingly, VAR outperforms both the LSTM models for 5-step-ahead prediction! We did additional experiments to go up to 10-step-ahead prediction, and VAR continues to dominate both the LSTM models, and the improvement is statistically significant.

For both VAR and LSTM-STD, the errors grow fast as we increase the number of predicted steps into the future. LSTM-STD starts with more accurate predictions as compared to LSTM-SED at 1-step, and then the errors quickly grow to exceed the LSTM-SED at 3-steps. The VAR also starts at significantly lower error than LSTM-SED but then it comes close to LSTM-SED at 5-steps. Also, note that the standard deviation at step-5 is smaller for LSTM-SED as compared to VAR. This could mean that the LSTM-SED model is more stable in terms of learning as the variability in test error is smaller. These observations motivated us to train another LSTM-SED to perform 10-step-ahead prediction to see if the trend changes. As can be seen from the error bar plot in Figure 6 and Table 1 (last column), the trend continues to be similar, though the gap between the VAR and LSTM-SED stops decreasing with VAR still outperforming LSTM-SED.

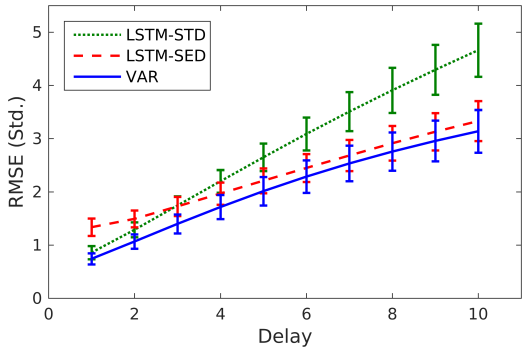


Figure 6: Error bar plots for RMSE (Standard Deviation) on the flight data for VAR, LSTM-STD and LSTM-SED for all delay values between 1 and 10. The RMSE for LSTM-STD errors grow rapidly as compared to the other two. Note that the gap between VAR and LSTM-SED becomes narrower for higher delays.

6.1.3 Exploring LSTM-STD Architectures To verify that we used the best LSTM architecture, we conducted experiments by varying some of the different configurable parameters such as the number of hidden units n, the input sequence length and the number of layers. These experiments were done on the aviation data. The models were trained for 800 epochs or a fixed time of 10 hours, whichever ended earlier. The results presented in Table 2 indeed showed that our configuration of 64 hidden units, input sequence length of 10 and 1 hidden layer gives the best performance within a reasonable training time. Note that the results for 2 hidden layers are not shown in the table, because

Model	Delay					
	1	2	3	4	5	10
VAR	**0.752*(0.103)**	**1.093*(0.130)**	**1.439*(0.173)**	**1.767*(0.220)**	**2.073*(0.263)**	**3.138*(0.402)**
LSTM-STD	0.867 (0.120)	1.304 (0.137)	1.769 (0.169)	2.236 (0.205)	2.694 (0.246)	4.663 (0.501)
LSTM-SED	1.105 (0.160)	1.319 (0.163)	1.607 (0.185)	1.914 (0.209)	2.209 (0.231)	3.329 (0.375)

Table 1: RMSE (Standard Deviation) values for different future prediction delays for aviation data - VAR vs LSTM. VAR beats LSTM based models in terms of error performance. However, for larger delays the standard deviation for LSTM-SED is the best, which indicates it is more stable than the other two models. Also, note that the * on VAR RMSE values indicate that these results are statistically significant as compared to the LSTM values, based on the Kolmogorov-Smirnov test.

Input	Hidden Units		
Length	32	64	128
5	1.024 (0.169)	0.942 (0.144)	0.919 (0.118)
10	1.041 (0.183)	**0.878 (0.118)**	0.913 (0.118)
20	1.038 (0.172)	0.896 (0.121)	0.912 (0.133)

Table 2: RMSE (Standard Deviation) summary for 1-step predictions by varying LSTM-STD parameters of input sequence length and number of hidden units. Our primary model with 1 LSTM layer, input sequence length of 10 and 64 hidden units performs the best. Note that for the aviation data, higher RMSE for 32 hidden units shows that within-feature correlation is important to capture.

we only tested 64 hidden units with 2 layers and obtained worse performance than with 1 hidden layer.

Some other observations about training LSTMs are as follows. If the data has important correlation within its multiple variables and has no long-term dependencies, it is better to use more hidden units and smaller sequence length. This allows the backpropagation-through-time algorithm to be faster as the gradient is to be calculated at smaller number of time steps. Also, having more hidden units can help capture the correlation between features. Another observation is that the deeper networks are also not always better. When another layer is added for the data which is not of high complexity, it becomes harder for the model to reach the local minimum because it has to estimate twice the number of original parameters.

6.2 VAR-generated Data We looked at performance of VAR and LSTM on synthetic data generated from a VAR model. The average 1-step-ahead prediction RMSE and standard deviation values on the test sets are shown in Table 3. Since the VAR-generated data is relatively simple, LSTM-STD's performance is better than on the aviation data. However, VAR still outperforms LSTM-STD by a substantial margin. In synthetic VAR-generated data, one would expect VAR to perform well, but it was surprising that LSTM-STD was outperformed by such a large margin.

Data	VAR	LSTM-STD
VAR-generated	**0.160**	0.445
LSTM-generated	**1.161**	1.590

Table 3: RMSE summary for 1-step predictions for both VAR-generated and LSTM-generated data. VAR outperforms LSTM by a good margin for both the datasets. The good performance of VAR on synthetic datasets as well leads to the conclusion that VAR is better than LSTM at predicting continuous time series data.

6.3 LSTM-generated data. For a fair comparison, we looked at performance of VAR and LSTM on synthetic data generated from a LSTM model. The average 1-step-ahead prediction RMSE and standard deviation values on the test sets are shown in Table 3. Surprisingly, we found VAR outperformed LSTMs even on LSTM-generated data! The result suggests that VAR may be better at predicting continuous multivariate time-series, especially ones with short term dependence, compared to LSTM. One reason to explain this behavior could be as observed by Gers et al. in [9]. In their work, time-window based simple, feed-forward neural networks outperform LSTMs in the prediction of the Mackey-Glass time-series [18]. The observation is that LSTM's main strength - the ability to remember significant events from a long time ago - is not of much use in the time-series where the next values can be predicted by just looking at some previous time-steps. Therefore, in such data, it might be more useful to use simpler, traditional approaches, such as VAR in our case. It is also possible that all the datasets that we have used in our experiments do not need events from long back to be remembered for modeling, therefore using linear approaches like VAR produce better results than LSTM.

7 Conclusions

In this work, we presented a comparison of VAR and LSTM on time-series prediction on the multivariate, continuous-valued aviation data. Two variants of LSTM were considered - standard LSTM predictor and a sequential encoder-decoder model for multi-step prediction. The experiments on real and synthetic data show

that VAR is better than LSTMs at predicting such time-series data. We attribute this to the hypothesis that for data where the future values can be predicted by just looking at a few previous time-steps, it might be better to use simpler linear models like VAR. Though LSTMs show promise with continuous-valued data, they do not currently outperform the state-of-the-art in this setting.

References

[1] NASA Flight Dataset. Available at https://c3.nasa.gov/dashlink/projects/85/.

[2] Federal Aviation Administration. Terminal area forecast summary. `https://www.faa.gov/data_research/aviation/taf/media/TAF_Summary_FY_2015-2040.pdf`, 2015.

[3] Federal Aviation Administration. Next generation air transportation system. `https://www.faa.gov/nextgen/`, 2016.

[4] K. Cho, B. V. Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *EMNLP*, pages 1724–1734, 2014.

[5] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NIPS*, pages 3079–3087, 2015.

[6] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing*, 34(1):206–239, 2012.

[7] S. Dieleman et al. Lasagne: First release., August 2015.

[8] A. ElSaid, B. Wild, J. Higgins, and T. Desell. Using lstm recurrent neural networks to predict excess vibration events in aircraft engines. In *IEEE Conference on eScience*, 2016.

[9] F. A. Gers, D. Eck, and J. Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *International Conference on Artificial Neural Networks*, pages 669–676. Springer, 2001.

[10] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[11] D. Gorinevsky, B. Matthews, and R. Martin. Aircraft anomaly detection using performance models trained on fleet data. In *CIDU*, pages 17–23, 2012.

[12] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552. 2009.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[14] M. Jaggi and M. Sulovsk. A simple algorithm for nuclear norm regularized problems. In *ICML*, pages 471–478, 2010.

[15] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell. Learning to diagnose with lstm recurrent neural networks. In *ICLR*, 2015.

[16] L. Ljung. *System identification: theory for the user*. Springer, 1998.

[17] H. Lutkepohl. *New introduction to multiple time series analysis*. Springer, 2007.

[18] M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

[19] I. Melnyk and A. Banerjee. Estimating structured vector autoregressive models. In *ICML*, 2016.

[20] I. Melnyk, B. Matthews, A. Banerjee, and N. Oza. Semi-Markov switching vector autoregressive model-based anomaly detection in aviation systems. *Knowledge Discovery and Data Mining*, 2016.

[21] I. Melnyk, B. Matthews, H. Valizadegan, A. Banerjee, and N. Oza. Vector autoregressive model-based anomaly detection in aviation systems. *JAIS*, 13(4):161–173, 05 2016.

[22] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML*, 28:1310–1318, 2013.

[23] M. Saada and Q. Meng. An efficient algorithm for anomaly detection in a flight system using dynamic bayesian networks. In *ICONIP*, pages 620–628, 2012.

[24] J. Schmidhuber, D. Wierstra, and F. Gomez. Evolino: Hybrid neuroevolution / optimal linear search for sequence learning. *IJCAI*, pages 853–858, 2005.

[25] A. N. Srivastava. Greener aviation with virtual sensors: a case study. *Knowledge Discovery and Data Mining*, 24(2):443–471, 2012.

[26] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

[27] I. C. Statler and D. A. Maluf. Nasa's aviation system monitoring and modeling project. Technical report, SAE Technical Paper, 2003.

[28] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[29] S. J. Taylor. *Modelling financial time series*. World Scientific Publishing, 2007.

[30] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[31] R. S. Tsay. *Analysis of financial time series*, volume 543. John Wiley & Sons, 2005.

[32] P. A Valdés-Sosa, J. M Sánchez-Bornot, A. Lage-Castellanos, M. Vega-Hernández, et al. Estimating brain functional connectivity with sparse multivariate autoregression. *Philosophical Transactions of the Royal Society*, 360(1457):969–981, 2005.

[33] F. Weninger, F. Eyben, and B. Schuller. On-line continuous-time music mood regression with deep recurrent neural networks. In *IEEE ICASSP*, pages 5412–5416, 2014.

[34] M. Wöllmer, F. Eyben, et al. Abandoning emotion classes-towards continuous emotion recognition with modelling of long-range dependencies. In *INTERSPEECH*, volume 2008, pages 597–600, 2008.

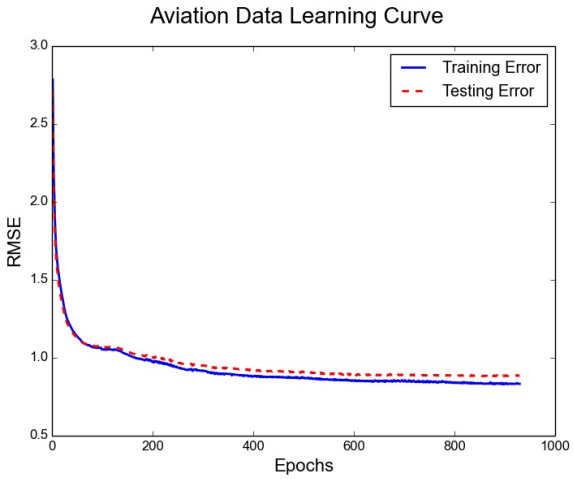
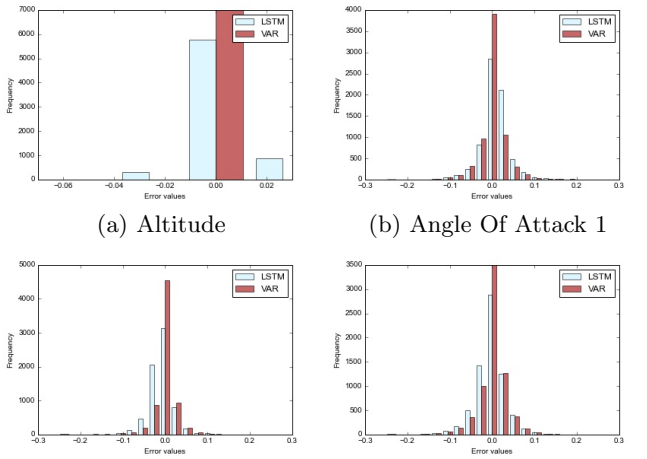


Figure 7: Learning curve of LSTM-STD on **flight data**. The plot shows the RMSE values on the training data and test data as a function of the number of epochs over the training data. The test set error flattens out after a while, with no overfitting.



(a) Altitude (b) Angle Of Attack 1

(c) Drift Angle (d) Flight Path Acceleration

Figure 8: Histogram plots of errors made by LSTM-STD and VAR for selected features (same as Figure 5) from a file in the **real flight data**. The errors are well distributed following a bell curve for both VAR and LSTM.

A Additional Results

A.1 Real Flight Data

A.1.1 LSTM-STD Learning Curve The learning curve for LSTM-STD with input sequence length of 10 and 64 hidden units is shown in Figure 7. This curve is for the real flight data. Note that, the test set error closely follows the training error and no over-fitting is noticed. The test set error flattens out after a while with no significant changes.

A.1.2 1-step-ahead predictions Since there were 42 continuous features in all that we used in our experiments, the plots of 1-step-ahead predictions for selected portions of a flight are shown in Figures 9 and 10. These plots are for the remaining 38 features (the rest 4 were in Figure 5). Note that the discussion in Section 6.1.1 can be seen again in the plots here, that is if LSTM-STD overshoots some peaks, it keeps overshooting; similarly if it undershoots, it keeps undershooting further peaks as well.

To gain a better understanding of the errors made by LSTM-STD and VAR in 1-step prediction, we present the error histograms for the same selected features in Figure 8. VAR is seen to have many errors close to zero compared to LSTM-STD, and LSTM-STD can sometimes have large errors.

A.2 VAR-generated data

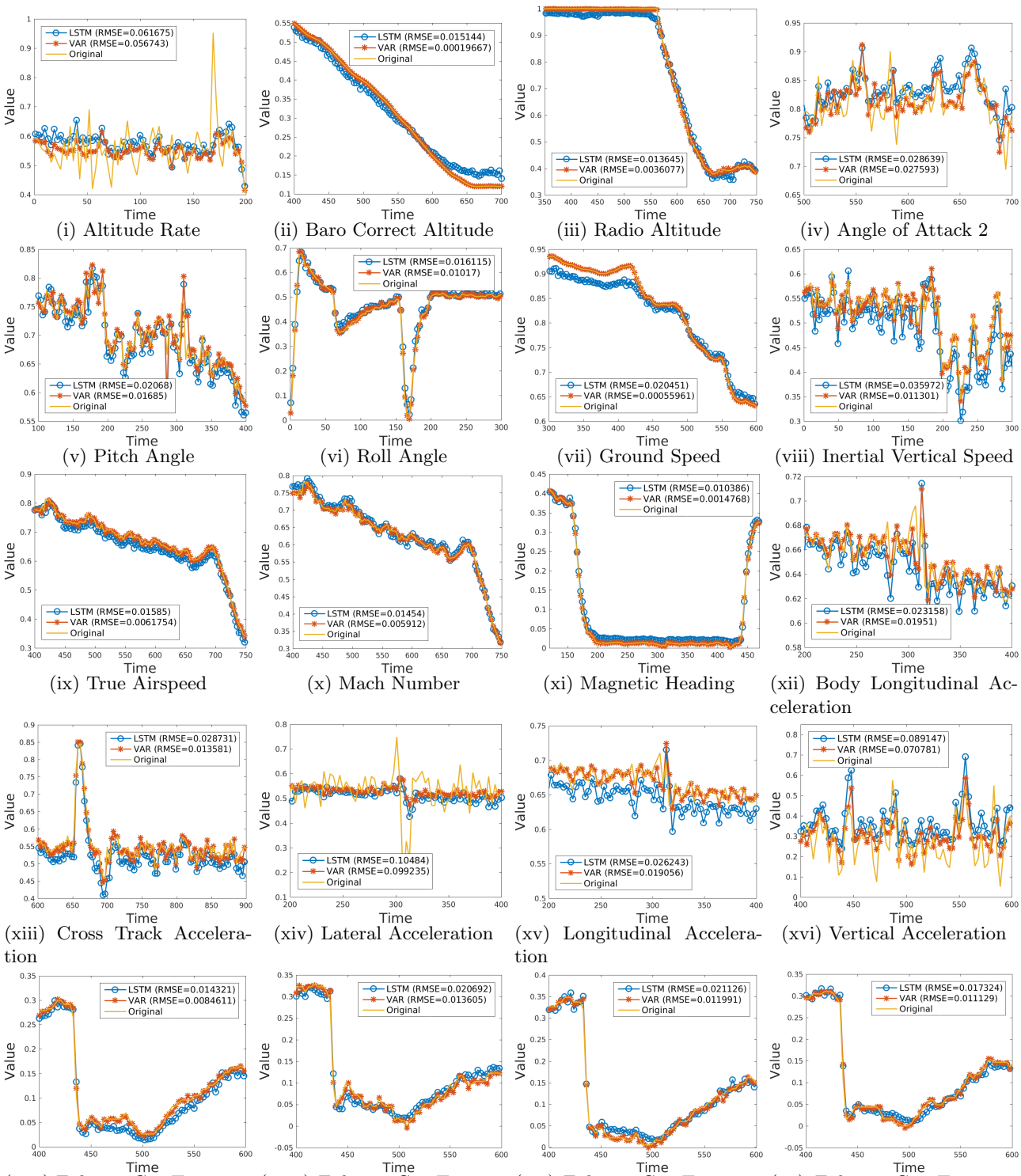
A.2.1 1-step-ahead predictions For the VAR-generated data in Section 5.2, the one-step predicted values on the test set are shown in Figure 11 for selected features. The figure shows that the data is of comparatively less complexity. Note that, the VAR has learned the features almost perfectly, while the LSTM-STD still makes some errors.

To have a better look at the errors made by LSTM-STD and VAR, the error histograms for selected features are shown in Figure 12. Again, the errors seem to be well distributed for both VAR and LSTM-STD as in the case of aviation data.

A.3 LSTM-generated data

A.3.1 1-step-ahead predictions For the LSTM-generated data in Section 5.2, the one-step predicted values on the test set are shown in Figure 13 for selected features. Note that, as compared to the VAR data, the data generated from LSTMs is of comparatively higher complexity. VAR has again outperformed LSTMs here, but not by a very significant measure.

To have a better look at the errors made by LSTM-STD and VAR, the error histograms for selected features are shown in Figure 14. Again, the errors seem to be well distributed for both VAR and LSTM-STD as in the case of aviation data.



(i) Altitude Rate (ii) Baro Correct Altitude (iii) Radio Altitude (iv) Angle of Attack 2

(v) Pitch Angle (vi) Roll Angle (vii) Ground Speed (viii) Inertial Vertical Speed

(ix) True Airspeed (x) Mach Number (xi) Magnetic Heading (xii) Body Longitudinal Acceleration

(xiii) Cross Track Acceleration (xiv) Lateral Acceleration (xv) Longitudinal Acceleration (xvi) Vertical Acceleration

(xvii) Exhaust Gas Temperature 1 (xviii) Exhaust Gas Temperature 2 (xix) Exhaust Gas Temperature 3 (xx) Exhaust Gas Temperature 4

Figure 9: Plots of portions of selected features from a file in the **real flight data** and the 1-step predictions made by LSTM-STD and VAR. Note that the x-axis and y-axis are different for each feature because selected sections of each feature have been zoomed-in for clarity. 20 features are shown here, apart from the 4 in Figure 9, out of the total 42.

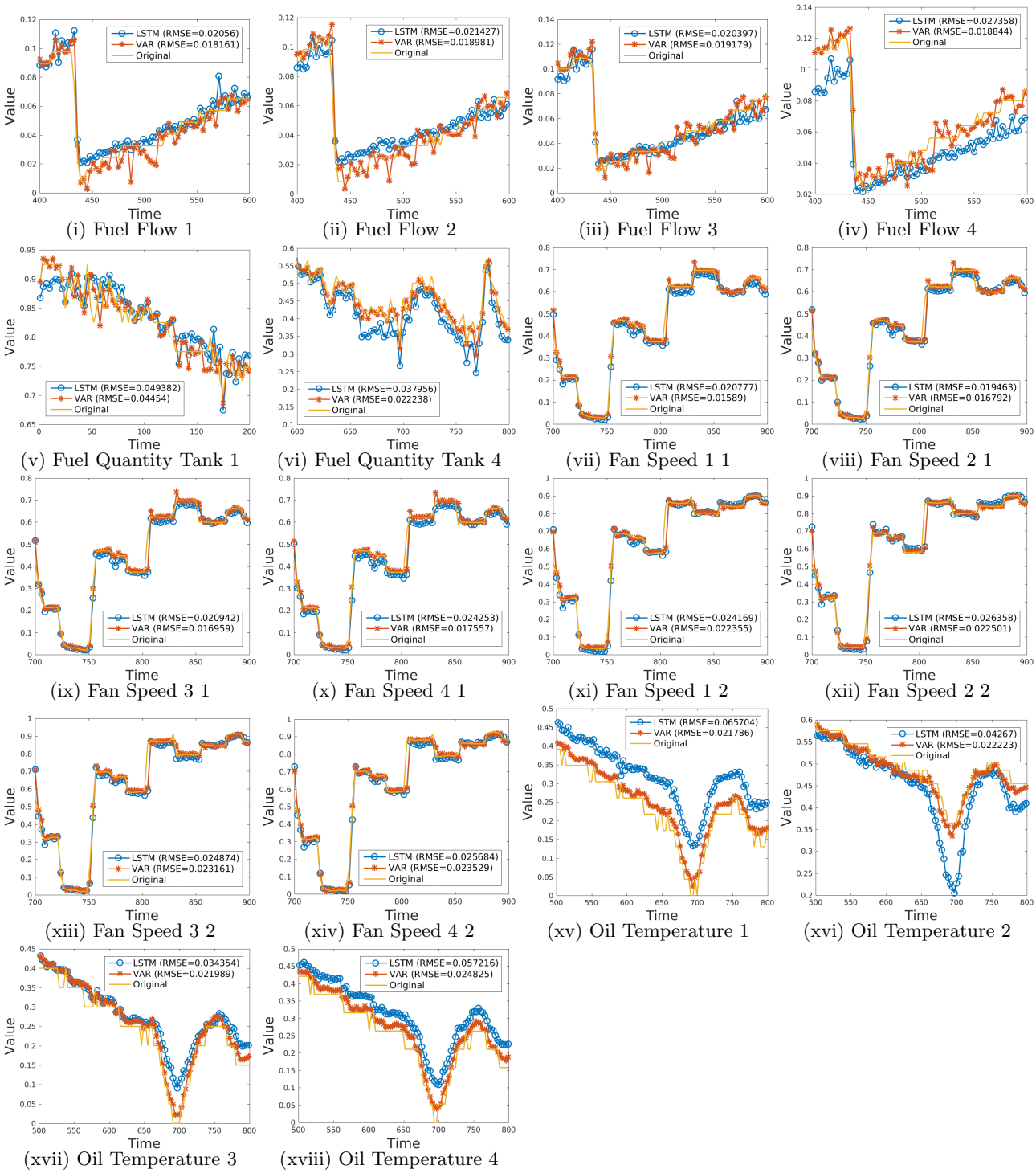


Figure 10: Plots of portions of selected features from a file in the **real flight data** and the 1-step predictions made by LSTM-STD and VAR. Note that the x-axis and y-axis are different for each feature because selected sections of each feature have been zoomed-in for clarity. 18 features are shown here, apart from the 4 in Figure 5, out of the total 42.

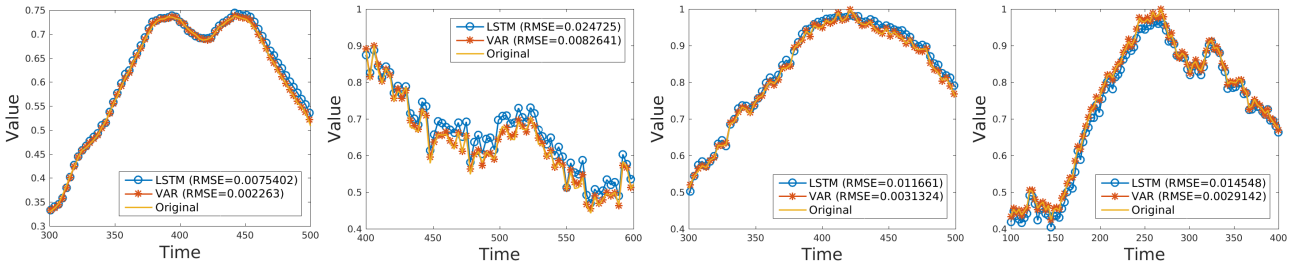
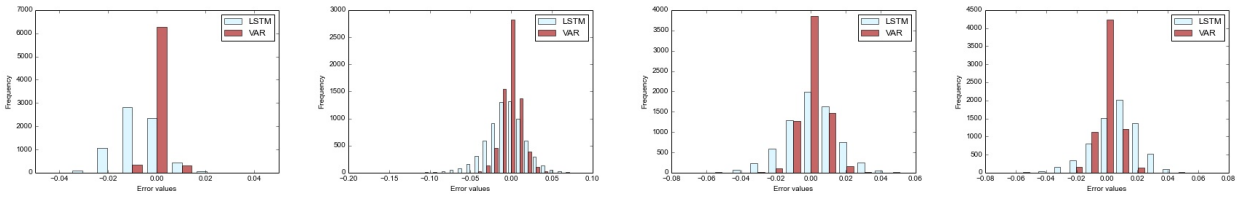


Figure 11: Plots of portions of selected features from a file in the **VAR generated data** and the 1-step predictions made by LSTM-STD and VAR. Note that the x-axis and y-axis are different for each feature because selected sections of each feature have been zoomed-in for clarity. VAR again outperforms LSTM and follows the original plots really well. A small amount of peak overshooting can again be noticed in the LSTM-STD predictions (as for the aviation data).



(i) Feature 1 (ii) Feature 5 (iii) Feature 7 (iv) Feature 17

Figure 12: Histogram plots of errors made by LSTM-STD and VAR for selected features (same as Figure 11) from a file in the **VAR-generated data**. Both VAR and LSTM-STD have a good bell-shaped error distribution.

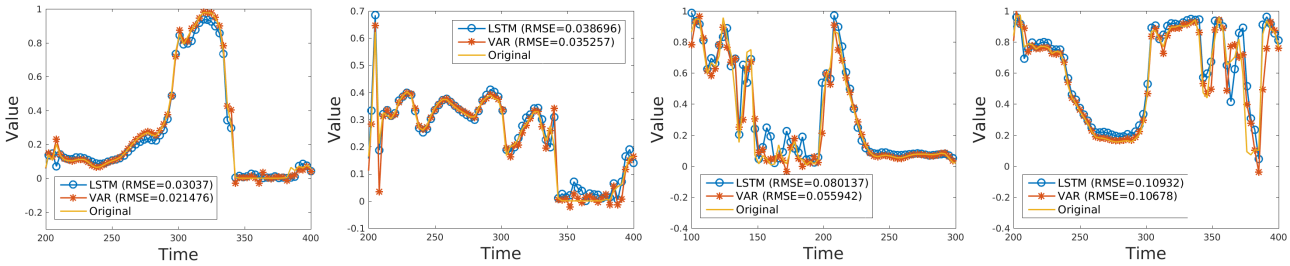
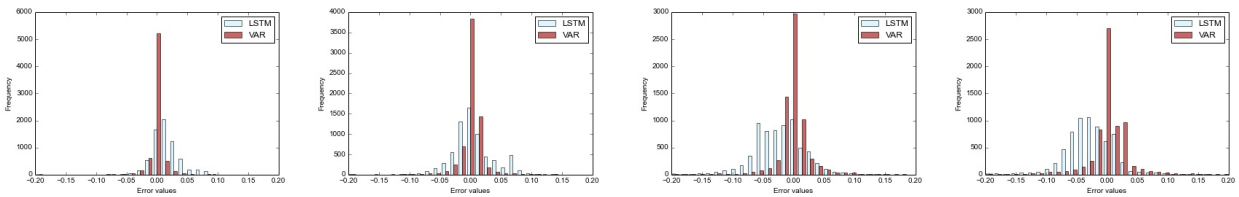


Figure 13: Plots of portions of selected features from a file in the **LSTM-generated data** and the 1-step predictions made by LSTM-STD and VAR. Note that the x-axis and y-axis are different for each feature because selected sections of each feature have been zoomed-in for clarity. VAR again outperforms LSTM-STD though not as significantly, probably due to more osciallations in this data



(i) Feature 1 (ii) Feature 5 (iii) Feature 7 (iv) Feature 17

Figure 14: Histogram plots of errors made by LSTM-STD and VAR for selected features (same as Figure 13) from a file in the **LSTM-generated data**. Both VAR and LSTM-STD have a good bell-shaped error distribution.