CS230

Learning Neural Network Representations of Aircraft Flight Dynamics

Loren J. Newton
Department of Aeronautics and Astronautics
Stanford University
ljnewton@stanford.edu

Abstract

As aircraft design progresses into increasingly complex dynamics and flight envelopes, it is difficult to mathematically capture these complexities with linear models. To improve on this, nonlinear neural networks are trained here in 2 architectures: a feedforward network which is then closed in feedback for simulation, and a recurrent network for explicit time series prediction. While the feedforward network has been favored in literature due to faster training times, the results presented here that despite an almost 2-fold increase in training computation time, the RNN compensates for this with an over 2-fold increase in modeling accuracy.

1 Introduction

A mathematical model of aircraft flight dynamics is an important tool in analysis of flight dynamics. Aircraft system identification refers to the process of using flight data to characterize a system that could have produced that data. The end result, then, is a model that may be used to analytically simulate aircraft behavior, assess handling qualities, and quantify stability. [1] As it becomes available, additional flight data may be applied to progressively update and improve this model. A wide variety of mathematical methods fuel the fitting of models in the system identification context. However, the structure of the model, whether linear or nonlinear, must be fixed beforehand. This report focuses on selecting the architecture of a neural network for nonlinear aircraft system identification.

2 Background and Previous Work

Traditional aircraft system identification schemes fit a linear model to the data. [2] While this process includes several benefits such as simplicity and analytical least-squares fitting, it inherently cannot abstract any nonlinearities in the aircraft dynamics without complex stitching together of different models linearized about different trim points. [3] Unmodeled nonlinearities may otherwise cause the models to deviate significantly from the actual aircraft behavior. Instead of linear methods, neural networks may be used as an architecture to explicitly model nonlinear flight dynamics. The dynamics of an aircraft may be represented with Fig. 1, a recurrent neural network. [4]

Physically, the state of the aircraft at the next time step (the network output) is a function of the state at previous time steps (the feedback connection) and any pilot inputs δ_i at the current time step. This recurrent network models this physical dependency explicitly. The "backpropagation-through-time" algorithm is an augmented form of backpropagation that has been applied to train recurrent neural networks. [5] Traditionally however, [6] this model is trained "open-loop;" (Fig. 2) that is, the next state estimate is simply predicted as a feedforward function of true previous states provided in the

CS230: Deep Learning, Winter 2020, Stanford University, CA. (LaTeX template borrowed from NIPS 2017.)

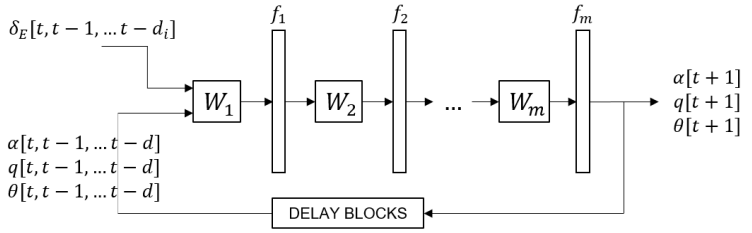


Figure 1: A recurrent neural network representation of flight dynamics.

training data. [7] This is less computationally expensive than training the recurrent network outright, as it does not explicitly account for the feedback relationships between different training outputs. After training, this feedforward network is then closed with feedback to resemble Fig. 1. As it is now estimating states at future time steps as a function of its own previous outputs, it is self-sustaining and now has an infinite prediction horizon.

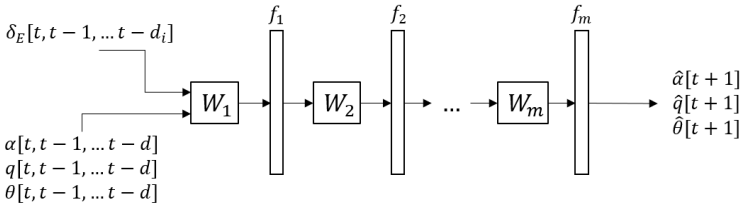


Figure 2: An open-loop training configuration.

This project seeks to model flight dynamics with both approaches, and assess the tradeoff between computation time and network accuracy afforded by each method.

3 Dataset

3.1 Data Source: Aircraft Simulation

In a system identification task in a real flight test campaign,the aircraft in question would be flown through a series of maneuvers designed to excite certain dynamic modes. [8] (The maneuver nature is very important to system identification, see Appendix A). The data time histories could then be applied to a system identification algorithm to build a mathematical model from the data. In lieu of having (typically proprietary) flight test data for this project, a simulation of the Rockwell Commander 700 aircraft's longitudinal dynamics was constructed and "flown" in MATLAB Simulink to produce data for the system identification task. It is simple but nontrivial: a published [9] linear aircraft model with rate limit and saturation nonlinearities on the input, and a simple gain/delay pilot model in the feedback control loop. (Fig. 3)

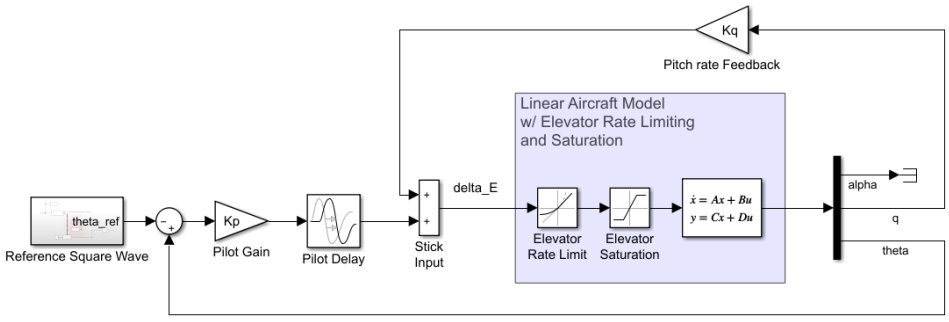


Figure 3: Block diagram of the Commander 700 nonlinear longitudinal dynamics simulation. The blue box contains the aircraft dynamics to be modeled with neural networks.

The aircraft state includes the angle of attack α, pitch rate q, and pitch angle θ. The single pilot input commanded the deflection of the elevator control surfaces, δ_E. This allowed the construction of a dataset with high-excitation maneuvers typical of a flight test campaign geared towards system identification. In each data recording, the closed loop system was tasked with tracking a square-wave reference in the pitch angle. Aircraft state time histories were measured for 980 different permutations of pilot model gain and lag and pitch reference period and amplitudes. Note that varying these parameters changes the nature of the pilot command and thus the states realized- however, it does not change the dynamics of the aircraft itself. The state and input time histories were sampled from each "flight" at 100 samples per second. Each maneuver was simulated for 24 seconds, giving a total of 2.4 million total samples.

3.2 Data Pre-Processing

The time history outputs of the simulation needed to be processed into appropriate formats for each of the network architectures. For the feedforward network, this step packaged each training input as the state at a specified number of previous time steps and the input at a (potentially different) number of time steps. This was paired with the state at the appropriate future time step as the training output. These "state delay" and "input delay" numbers were network hyperparameters to be tuned. For the RNN, the states and inputs at a certain identical number of time steps comprised each training input. The training output consisted of the states at a certain number of future time steps. The number of time steps in the training inputs and outputs were hyperparameters as well. For both cases, 884 (90%) of the 980 maneuvers were selected at random as the training set; 48 were the development set and 48 went to the test set. The overall training set of about 1.9 million training pairs was shuffled and used to train each network in minibatches of 1000 samples each. The validation and test data sets were *not* shuffled, in order to preserve the time history visualization there.

4 Constructing and Training Networks

Both neural network architectures were implemented in Python using the Tensorflow framework; network hyperparameters were tuned through manual trial-and-error iteration. The general philosophy for sizing hyperparameters (such as number of layers, number of neurons, number of delays, etc.) was to start small and incrementally increase each parameter until no additional test set performance was seen. The philosophy with regularization constants was to start at zero and gradually increase them until the early-stopping flag (discussed in section 4.1) was not thrown in the first ten iterations. Finally, the strategy for tuning the training learning rates was to start high and gradually decrease until no additional test set performance was seen.

4.1 Learning Algorithm

Conventional backpropagation was applied to train the feedforward network, and backpropagation-through-time was applied for the RNN; in both cases, the Adam optimizer was used to specify each weight change. The success of convergence was extremely sensitive to Adam's learning rate hyperparameter; a value of $\alpha = 5 \times 10^{-5}$ was found to provide reliable convergence for the feedforward network and a value of $\alpha = 1 \times 10^{-5}$ worked well for the RNN. The hyperparameters β_1, β_2, and ϵ were not changed from their accepted default values of 0.9, 0.999, and 10^{-8}, respectively. The cost function was selected to be mean-squared error between the network's state estimates and the true states from the training data (augmented by L_2 regularization terms, as are discussed below). Early stopping was implemented, at the first observed increase in development set loss.

4.2 Converted Feedforward Network Approach

The feedforward neural network design contained 3 hidden layers with 50, 30, and 30 units respectively. Network inputs were normalized; batch normalization was also implemented in each hidden layer. The 3 hidden layers contained ReLU activation functions; the output layer, however, had a linear activation to allow the outputs to take on a full range of values to match the data. To prevent overfitting, L_2 regularization was applied to the weights of the hidden layers with regularization constant 0.1; inverted dropout was similarly implemented on all 3 hidden layers, with dropout frequencies of 0.2, 0.2, and 0.1 respectively. It was found that adding additional state delays and input

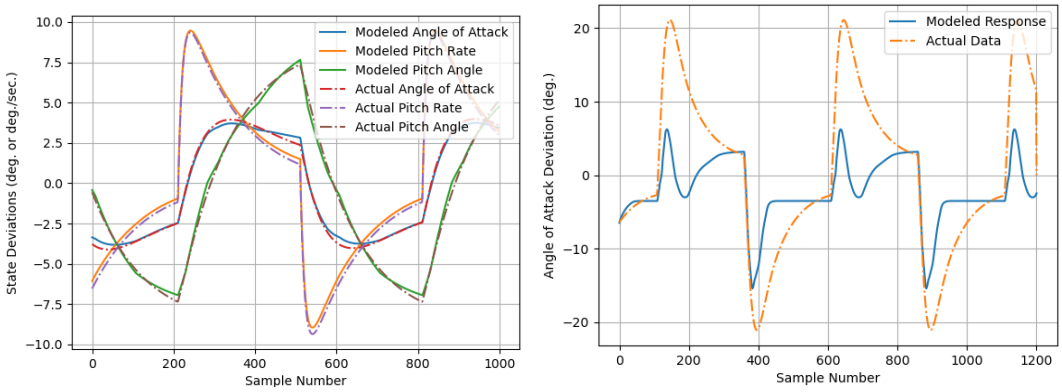
delays beyond a single time step each did not increase the fitting accuracy of the network. As such, the feedforward network had 4 inputs: the pilot input and the 3 state values at the current time step. The output was 3-dimensional, matching the state at the next time step.

4.3 Closed-loop Recursive Neural Network Approach

The RNN representation took on a canonical encoder-decoder architecture with LSTM blocks. The decoder was fixed to have 20 LSTM blocks, equivalent to a 0.2-second prediction horizon. This was due to the fact that the pilot delay discussed in section 3.1 was simulated at a maximum of 0.2 seconds: if the neural network model replaced the plant in Fig. 3, any errors beyond a 0.2-second prediction horizon would not be fed back into the modeled pilot reaction and would therefore be inconsequential. This delay is a conservative estimate, at the higher extreme of most pilot reaction delay models. [10] After some iteration, 30 encoder LSTM blocks were used for both training and testing, similar to the feedforward architecture where each output state was dependent upon one input. Here, a *slight* dependence on more than one previous input was identified, thus 30 encoder blocks and not simply 20. The LSTM blocks received L_2 regularization with a scaling constant of 0.1. For each decoder block, the output was fed through a time-distributed dense layer with a linear activation function (again to match the unbounded training target data).

5 Results, Analysis, and Conclusion

Training loss at each epoch can be seen for both networks in Appendix B. The feedforward network performed exceptionally well at testing time when operating in open-loop form: being presented with true data at the current time step and only predicting the state at a single future time step (Fig. 4a). However, when the feedforward network loop is closed to resemble Fig. 1, the network generally does not do as good of a job at modeling the system dynamics for longer prediction windows. Fig. 4b illustrates predictions for angle of attack as compared to the aircraft's actual response.



(a) Evaluating the feedforward network on feedforward testing pairs.

(b) Evaluating the feedforward network in closed-loop operation.

Figure 4: Randomly selected examples of feedforward network modeling performance.

This is perhaps expected. In any long-horizon prediction scheme, model errors propagate and are summed in each future time step. In linear aircraft system identification, this error accumulation tends to cause steady drift away from the truth. In a neural network representation however, there is more potential for strange error propagation behavior due to greater computational graph complexity and nonlinearities, as seen here where the network seems to represent completely different dynamics. The RNN, however, demonstrated superior time series modeling as anticipated. Figure 5 illustrates an example of the RNN's prediction of 0.2 seconds of future states, based on 0.3 seconds of previous states and control inputs. Overlaid on this plot is the feedforward network's prediction of the same time window, as well as the actual aircraft behavior truth source.

Clearly the RNN is performing better in this example. The error metric for the RNN training scheme- the MSE between the 0.2-second prediction window and the true aircraft state in that window- was

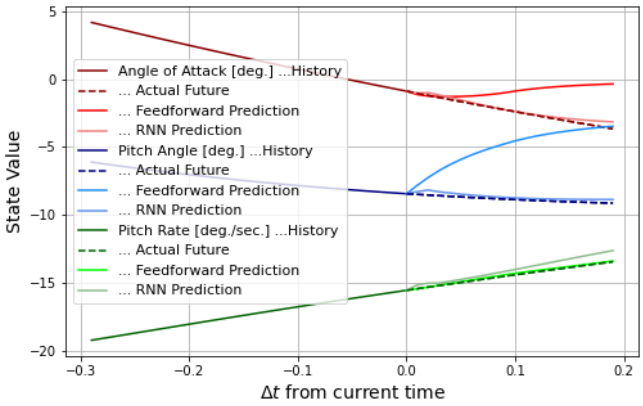


Figure 5: Randomly selected example 0.2 second time series prediction, comparing the two networks.

applied to the feedforward network as well for comparison. For each time step in the testing data (the data from 48 flights concatenated together), a 0.2-second prediction was made with each of the networks. The MSE of these errors is illustrated in Fig. 6.

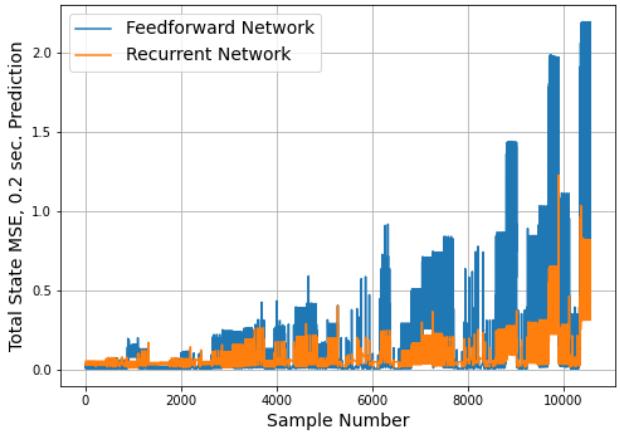


Figure 6: Comparison of MSE for a 0.2 second prediction for each network.

Note that the errors plotted in Fig. 6 seem to be stratified to roughly constant values every several hundred samples: each of the strata represent a different flight within the test set. On average, the RNN returned only 39.5% of the error of the feedforward network. Training the final RNN required 946 seconds on an AWS p2.xlarge GPU. The feedforward network required only 511 seconds, but **at this scale the increase in computational time is certainly worth the superior RNN performance.**

Finally, the networks proved insensitive to several other developments. Regularization, even for constants much larger than were used in training the final networks, seemed to have little effect on the results. Since the training, development, and testing datasets are all drawn from similar distributions- the same aircraft flight envelope- perhaps overfitting would be more difficult to achieve without larger networks. This would explain the marginal impact of regularization. In an attempt to improve the feedforward network's closed loop performance, noise was added to the training data to include some robustness or tolerance to modeling error into the network. When trained on this data, the feedforward network's open-loop performance decreased; its closed-loop performance did not change in having large and unpredictable errors. The RNN, when trained on this data, did not experience any appreciable closed-loop performance changes either. Regardless, favorable results were achieved with the RNN and this architecture is recommended for future, aircraft system identification work: an improved nonlinear tool for modeling aircraft flight dynamics.

Contributions

Loren Newton is the sole author of this report and all of the work described herein.

References

[1] Klein, V., and Morelli, E. A., "Introduction," *Aircraft System Identification: Theory and Practice*, AIAA, 1-8.

[2] Klein, V., "Estimation of Aircraft Aerodynamic Parameters from Flight Data," *Progress in Aerospace Sciences*, Vol. 26, No. 1, 1989, pp. 1-77.

[3] Berger, T., Tischler, M. B., Hagerott, S. G., Cotting, M. C., and Gray, W. R., "Identification of a Full-Envelope Learjet-25 Simulation Model Using a Stitching Architecture," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 11, 2020, pp. 2091-2111.

[4] Hagan, M. T., Demuth, H. B., Beale M. H., De Jesus, O., "Dynamic Networks," *Neural Network Design*, Oklahoma State University, 520-573.

[5] De Jesus, O. ND Hagan, M. T., "Backpropagation Through Time for a General Class of Recurrent Neural Network," *IJCNN'01. International Joint Conference on Neural Networks Proceedings*, IEEE, Washington, D. C., 2001, 2638-2643.

[6] Ng, B. C., Darus, I. Z. M., Jamaluddin, H., and Kamar, H. M., "Dynamic Modeling of an Automotive Variable Speed Air Conditioning System Using Nonlinear Autoregressive Exogenous Neural Networks," *Applied Thermal Engineering*, Vol. 73, No. 1, 2014, pp. 1255-1269.

[7] Hagan, M. T., Demuth, H. B., Beale M. H., De Jesus, O., "Variations on Backpropagation," *Neural Network Design*, Oklahoma State University, 413-467.

[8] Klein, V., and Morelli, E. A., "Experiment Design," Aircraft System Identification: Theory and Practice, AIAA, 289-311.

[9] Kirkpatrick, K., May Jr., J., and Valasek, J., "Aircraft System Identification Using Artificial Neural Networks," AIAA Report 2013-0878, January 2013.

[10] McRuer, D. T., and Jex, H. R., "A Review of Quasi-Linear Pilot Models," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 3, 1967, pp. 231–249.

[11] Matthews, B., "Flight Data for Tail 683," *NASA DASHlink*, retrieved 19 January 2021.

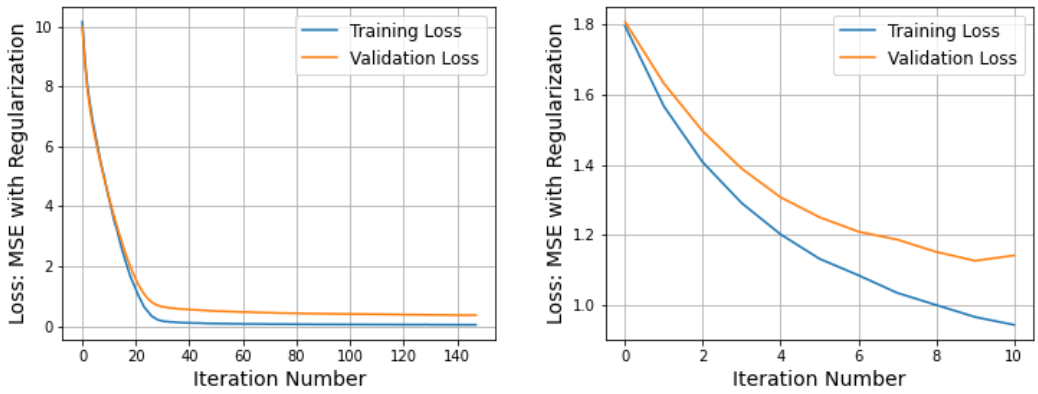
[12] Matthews, B., "Flight Data for Tail 669," *NASA DASHlink*, retrieved 19 January 2021.

Appendices

A Additional Flight Data Discussion

The dataset initially intended to be used consisted of 8,000 NASA-published flight data recordings from Boeing 747 airliner operations. [11][12] However, it was found that this data was inadequate for system identification, as there was not much excitation in each data time history. For an airliner flight, there are generally few high-amplitude changes in aircraft attitude in the interest of a smooth flight; most of the flight is steady state. This reduced the usefulness of the vast majority of the data since the aircraft is not excited for long periods: an important lesson to be kept in mind in aircraft system identification.

B Training Histories



(a) Feedforward network training history.

(b) RNN training history.

Figure 7: Loss at each training epoch for each network.