2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Deep Learning Helicopter Dynamics Models

Ali Punjani and Pieter Abbeel

Abstract— **We consider the problem of system identification of helicopter dynamics. Helicopters are complex systems, coupling rigid body dynamics with aerodynamics, engine dynamics, vibration, and other phenomena. Resultantly, they pose a challenging system identification problem, especially when considering non-stationary flight regimes.**

We pose the dynamics modeling problem as direct high-dimensional regression, and take inspiration from recent results in Deep Learning to represent the helicopter dynamics with a Rectified Linear Unit (ReLU) Network Model, a hierarchical neural network model. We provide a simple method for initializing the parameters of the model, and optimization details for training. We describe three baseline models and show that they are significantly outperformed by the ReLU Network Model in experiments on real data, indicating the power of the model to capture useful structure in system dynamics across a rich array of aerobatic maneuvers. Specifically, the ReLU Network Model improves 58% overall in RMS acceleration prediction over state-of-the-art methods. Predicting acceleration along the helicopter's up-down axis is empirically found to be the most difficult, and the ReLU Network Model improves by 60% over the prior state-of-the-art. We discuss explanations of these performance gains, and also investigate the impact of hyperparameters in the novel model.

I. Introduction and Background

System identification, the modeling of a system's dynamics, is a basic and important part of control. Constructing such a model is often the first step in controller design. Modeling accuracy directly impacts controller success and performance, as inaccuracies appear to the controller as external disturbances.

The aerobatic helicopter has received attention in the past as a system that is difficult to both model and control but which humans can fly effectively, even through complex maneuvers. The helicopter system couples rigid body dynamics with aerodynamic forces, internal control loops and lag, engine dynamics, vibration, etc.[1, 2, 3, 4]. Much of this dynamic coupling involves state variables like airflow which are not easily measured, and thus remain hidden. The resulting system identification problem for aerobatic helicopters poses a challenging problem. This paper focuses on a method to learn a system dynamics model by leveraging captured state-action trajectories from expert demonstrations.

The helicopter is well studied, and simple linear models from aerodynamic first-principles allow construction of controllers for simple trajectories (hover, forward flight) [5, 6, 7]. More complex nonlinear models from first-principles with parameters learned from data have enabled some simple aerobatics (roll, hammerhead, split-S) [8, 9, 10]. Advanced aerobatics, however, have only been successfully flown under autonomous control using Apprenticeship Learning [11]. In the Apprenticeship Learning approach for autonomous helicopter aerobatics, demonstrations of interest are collected, then aligned in time using a variant of Dynamic Time Warping (DTW) [12, 13]. From the aligned trajectories, a target trajectory for control and time-varying dynamics are estimated. Together, the target trajectory and time-varying dynamics around that trajectory allow for successful control of the helicopter through advanced aerobatics, using Model Predictive Control (MPC). The Apprenticeship Learning work demonstrates that in fact helicopter dynamics during aerobatics are predictable enough for MPC to be successful in control around even the most complex trajectories [11]. This suggests that the difficulty in modeling helicopter dynamics does not come from stochasticity in the system or unstructured noise in the demonstrations. Rather, the presence of unobserved state variables causes simple models to be innacurate, even though repeatability in the system dynamics is preserved across repetitions of the same maneuver.

Fig. 1. The dynamical modeling method presented in this work is used to model a helicopter's dynamics as it is flown through aerobatics. A unified model is learned for a rich array of maneuvers, including flips, loops, stop and go, vertical sweeps, circles, dodging, orientation sweeps, gentle freestyle, tictocs, chaos, and aggressive freestyle.

Modeling systems with hidden latent state is often treated as a parameter learning problem in a graphical model [14, 15]. One illustrative example is Expectation Maximization - Extended Kalman Smoothing (EM-EKS) [15] which treats the dynamical system as a latent variable model, and aims to directly estimate the hidden latent state at each timestep along a demonstration trajectory while simultaneously learning a simple dynamics model. In EM-EKS, the form of the dynamics model as well as the number of hidden state variables needs to be prescribed *a-priori*. The Apprenticeship Learning method of [11], on the other hand, deals with hidden state by relying on the assumption that across different demonstrations of the same maneuver, the trajectories of both observed and hidden state variables are similar. For example, [11] expects that the effect of the airflow around the helicopter is similar at the apex of two different loop demonstrations. Using DTW to align annotated loop trajectories and their corresponding state trajectories

Authors are with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, USA {ali.punjani, pabbeel}@berkeley.edu

allows the method of [11] to estimate the dynamics of the helicopter at the apex without explicitly inferring the hidden state.

Although the DTW step in [11] aligns demonstrations based on their entire state trajectory, it is clear that at any particular time, the hidden states (for instance airflow) of the helicopter system depend only on the past. A well known result from dynamical systems theory, Takens Theorem, [16, 17] provides further insight into this viewpoint. Takens Theorem states that for a large class of nonlinear dynamical systems with d-dimensional state space, only $2d+1$ previous measurements of a single system output are required to effectively reconstruct the system state. Naturally, this leads to the idea of directly learning a dynamical model from time-lagged outputs, and skipping the intermediate step of inferring hidden states. This paper aims to exploit this notion in order to construct a global nonlinear model of the helicopter that does not require annotation or alignment of trajectories, has few tuning parameters and that captures dynamics throughout the entire set of flight regimes over which data is collected.

A sequence of time-lagged system outputs and controls grows in dimensionality as the time horizon is increased. Thus, learning a direct high-order dynamics model requires a method for regression of a nonlinear function in a high-dimensional input space. Recently, Deep Learning (regression using hierarchical neural network representations) has had much success in high-dimensional learning problems in domains like image understanding [18], natural language processing [19, 20], and most relevantly with time-series inputs as in handwriting [21] and speech recognition [22]. In Deep Learning, the lowest layers in the hierarchical representation partition the input space, by computing a transformation of the inputs where each component is only active in a certain region. This makes the regression task of the next layer in the hierarchy, taking the transformed components as input, much simpler. This is a similar notion to Hybrid System Identification [23], where a system is modeled using a set of simple sub-models, along with definitions of regions in state-control space where each sub-model is active. We believe the power of Deep Learning can be brought to the system identification task through this connection; deep learning methods can be used to learn sub-models and the definitions of their corresponding active regions jointly.

Our main contribution in this work is proposing a novel Rectified-Linear Unit (ReLU) Network Model, providing details about parameter initialization and optimization methods. We compare the model against three baseline methods, and show that it performs well in modeling helicopter dynamics across all flight regimes for which data was collected. The range of maneuvers covered includes forward/sideways flight, vertical sweeps, inverted vertical sweeps, stop-and-go, flips, loops, turns, circles, dodging, orientation sweeps, orientation sweeps with motion, gentle freestyle, tic-toc, chaos, and aggressive freestyle. We describe the ReLU Network Model in Section IV after defining baseline models in Section III for performance comparison. Section II gives details of the helicopter system on which our experiments are carried out. Results and discussion follow in Section V.

II. The Helicopter System

Consider a dynamical system with state vector s and control inputs u. The task of system identification is to find a function F which maps from state-control space to state-derivative:

$$\dot{s} = F(s,u,\theta)$$

Typically F is restricted to be from a certain family of functions $\mathscr{F}$ parameterized by θ. The system identification task then becomes to find, given training data, a vector θ (and the corresponding F) which minimizes prediction error on a set of held-out test data. In our problem setting, training and test data sets consist of state-control trajectories, with no additional labels.

The helicopter state s is represented by a total of twelve degrees of freedom, three for each of position r, attitude q, linear velocity v, and angular velocity ω. The control space u for the helicopter consists of four inputs. Cyclic pitch controls u_1 and u_2 cause the helicopter to pitch forward-back or sideways respectively. Tail rotor collective pitch u_3 modulates tail rotor thrust, causing yaw. The main rotor collective pitch control u_4 modulates the pitch angle of blades in the main rotor, and consequently impacts the total thrust of the main rotor in the up-down direction.

Following standard practice in system identification and helicopter modeling in particular [24, 11], we make an informed choice about the family of functions $\mathscr{F}$. Knowledge of gravity and the kinematics of rotation are both directly incorporated into $\mathscr{F}$, so that only the dynamics particular to the helicopter are encoded in θ. Consider a fixed world reference frame (1) and a frame (2) fixed to the center of mass of the helicopter. Rotation matrix C_{12} (parameterized by quaternion q) rotates vectors from frame 2 to frame 1, and the velocity and angular velocity of the helicopter can be expressed in frame 2 as v and ω. Then the kinematics of rotating frames, along with the differential equation for quaternion rotations [25] provide the form of F:

$$\dot{s} = \begin{bmatrix} \dot{r} \\ \dot{q} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = F(s,u,\theta) = \begin{bmatrix} C_{12}v \\ \frac{1}{2}\hat{\omega}q \\ C_{12}^{\mathrm{T}}g - \omega \times v + f_v(s,u,\theta) \\ f_\omega(s,u,\theta) \end{bmatrix} \quad (1)$$

Here g is the known acceleration of gravity, and $f = (f_v, f_\omega)$ is the unknown linear and angular acceleration that the helicopter undergoes, measured in the helicopter frame, as a function of state, control, and model parameters. The system identification task for the helicopter is now to find θ, given observed values of f, s, and u. In this work we minimize mean squared prediction error over a training set of demonstration data, solving

$$\min_{\theta} \sum_{t=1}^{T} \frac{1}{T} \|\tilde{f}_t - f(s_t,u_t,\theta)\|^2 \quad (2)$$

where $\tilde{f}_t$ are the observed values of f along a demonstration trajectory. Equation 2 results in a linear or nonlinear least squares problem depending on the form of $f(s,u,\theta)$.

III. Baseline Models

A simple dynamics model can be constructed by selecting f from the class of affine functions, and fitting the linear coefficients θ to the observed data with least squares regression. This Linear[1] Acceleration Model (originally proposed in [24]) serves as our baseline model, and has been shown to achieve state-of-the-art performance in our problem setting [24]. In the alternate setting of learning trajectory specific models, where maneuver labels are given at training and test time, Apprenticeship Learning provides state-of-the-art models [11].

In our work, the Linear Acceleration Model is defined as

$$f = Px_t + Qy_t \tag{3}$$

Where $x = \begin{bmatrix} v & \omega & 1 \end{bmatrix}^{\mathrm{T}}$, $y = \begin{bmatrix} u & \dot{u} \end{bmatrix}^{\mathrm{T}}$, and P, Q are matrices of parameters which are learned.

Note that we include the control derivatives $\dot{u}$ in the model. This is important especially in the case of yaw-rate control u_3, due to the presence of an internal gyroscope and control loop in the helicopter. The input u_3 modulates the setpoint of the internal control loop rather than directly controlling the yaw torque, so the helicopter acts more like a first-order system than a second-order system. Thus the rate of change of the yaw input is a better predictor of the yaw acceleration than the control input value itself.

Based on the Linear Acceleration Model, we construct a slightly more advanced baseline model in which f is a function of the current state and controls, and also of a sequence of past control inputs. This modification results in the Linear Lag Model, and serves to allow the model to capture control lag which can be seen as a simple but important type of hidden system state. The Linear Lag Model is defined as

$$f = Px_t + \sum_{i=0}^{H} Q_i y_{t-i} \tag{4}$$

where H is a horizon into the past, and P, $Q_0 ... Q_H$ are matrices of parameters to be learned.

We develop a third baseline model which extends the Linear Lag Model to capture some simple nonlinearities in the helicopter dynamics. In particular, it is known from physics that many aerodynamic forces (for instance drag) grow approximately quadratically in velocities. Furthermore, these forces depend on the frontal shape of the moving body and are thus expected to be assymetric with respect to direction of motion (for instance up versus down). To incorporate this knowledge, we define a feature transformation $\beta(z) = \begin{bmatrix} z & \max(0,z)^2 & \min(0,z)^2 \end{bmatrix}^{\mathrm{T}}$. This transformation is applied elementwise to x and y in the Linear Lag Model to arrive at the Quadratic Lag Model:

$$f = P\beta(x_t) + \sum_{i=0}^{H} Q_i \beta(y_{t-i}) \tag{5}$$

The transformation β provides the model with flexibility to capture quadratic nonlinearities in both state variables and lagged control inputs.

In our experiments, we show that our novel modeling method (the ReLU Network Model, presented in Section IV) significantly outperforms these baseline models in prediction accuracy. The Linear Acceleration Model serves as a direct state-of-the-art performance baseline [24]. Comparison with the Linear Lag Model serves to show the impact of additionally accounting for control lag, and the Quadratic Lag Model serves to show the impact of directly accounting for some nonlinear dependence on the current state and controls. The ReLU Network Model, presented next, accounts for all of these system features and also attempts to extract information about the hidden state of the helicopter system from state trajectories, which we show provides a significant performance increase. Note that the baseline models presented here might appear simple, but are actually nontrivial nonlinear dynamics models when composed with Equation 1.

IV. ReLU Network Model

Consider the space $\mathscr{P}$ of state-control trajectory segments, each of length H. Each point $p_t = (x_{t-H}, y_{t-H}, x_{t-H+1}, y_{t-H+1}, ..., x_t, y_t)$ in this space corresponds to a small part of a helicopter maneuver, H timesteps long, ending at time t. Takens Theorem suggests that if H is large enough, then from any segment p_t in $\mathscr{P}$, the entire hidden state z_t of the helicopter at the end of p_t can be reconstructed. Assuming that the mapping from p_t to z_t is smooth, we can expect that two points in $\mathscr{P}$ which are similar will also have similar hidden state, and thus similar dynamics. This leads to an implicit notion of neighborhoods in $\mathscr{P}$, focused around regions that have been explored in trajectories where system dynamics are similar. For example, the apex of a loop might correspond to one neighborhood, while the beginning of a roll might correspond to another. This same notion is the principle behind the method of [11], where alignment of similar trajectories is used to define neighborhoods.

In this work, we do not use any explicit grouping or alignment of similar trajectories, and we aim to learn these neighborhoods directly from training data. In a similar vein as feature learning [26], and using ideas from recent advances in computer vision and speech due to Deep Learning [18, 22], we propose a Rectified Linear Unit Network Model specifically for system identification. Our ReLU Network Model is a combination of the Quadratic Lag Model and a two-layer neural network using rectified-linear units in the hidden layer. Algebraically, the model can be written

$$f = P\beta(x_t) + \sum_{i=0}^{H} Q_i \beta(y_{t-i}) + \eta(p_t;\theta) \tag{6}$$

$$\eta(p_t;\theta) = w^{\mathrm{T}} \phi(W^{\mathrm{T}} p_t + B) + b \tag{7}$$

$$\phi(\cdot) = max(0,\cdot) \tag{8}$$

[1] The Linear Acceleration Model is not a linear dynamics model, as it only predicts accelerations measured in the helicopter frame as linear functions of state variables measured in the helicopter frame. The entire dynamics model, given by Equation 1, is highly nonlinear.

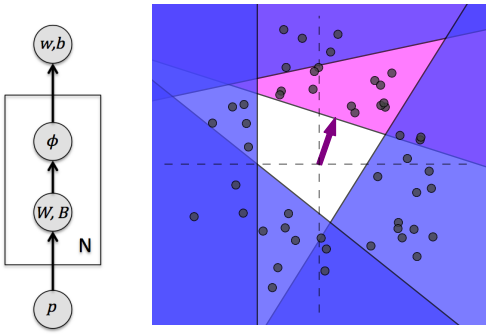

Fig. 2. **Left:** A diagrammatic representation of the neural network used in the ReLU Network Model. Input p is sent to each of N hidden units. Each unit computes the inner product of p and W_i, the weight vector for that unit. A bias B_i is added, and the result sent to the unit activation function, $\phi(\cdot) = max(\cdot, 0)$. This rectified-linear activation function simply zeroes its input if negative. The outputs of all hidden units are then linearly mixed by the output unit, with weights w and bias b. **Right:** A pictorial representation of the flexibility of the ReLU Network Model. Each hidden unit can be thought of as defining a hyperplane (line) in the 2D input space pictured. The data points (grey) each fall somewhere in input space and each has a value we wish to regress (not pictured). Consider the hidden unit i, drawn in purple. The purple arrow points in the direction of weight vector W_i, and has length according to B_i. Points on one side of this line do not activate unit i, while points on the other side (shaded) cause positive output. This effectively partitions the input space, and the partitions generated by considering many hidden units (blue) together split the space into regions. These regions give the model flexibility to capture structure in the input data. In the ReLU Network Model, the partitions are learned from data.

Here, η is the neural network of N hidden units with weights W and biases B, and a linear output layer with weights w and biases b. The hidden unit activation function ϕ is a soft threshold function. Matrices P, Q are parameters of the Quadratic Lag Model to be learned. Figure 2 shows a diagrammatic representation of the network η.

A. Interpretation

The neural network η can be interpreted as follows. The input layer takes in a point p from $\mathscr{P}$, representing a segment of state-control trajectory with length H. Each of N hidden units computes the inner product of p and one of the columns of W. Each of these columns, or weight vectors, is also a point in $\mathscr{P}$. The hidden units add a bias to the inner product and rectify this value at zero. The output layer is a linear combination of the hidden units, plus a final bias.

The inner product computed by hidden unit i can be thought of as a measurement of similarity between the input p and the hidden unit's weight vector W_i. This is made more concrete by noting that the inner product computes the projection of p onto the direction W_i. The nonlinearity $\phi(\cdot)$ from Equation 8 simply sets the hidden unit's output to zero if the projection is less than the unit's bias B_i. This means that any input p that does not have sufficient projection along W_i will be ignored by the hidden unit, while those inputs that are sufficiently similar to W_i will cause an output which is proportional to their similarity. In the ReLU Network Model, both the projection directions W and biases B are learned from training data. Each hidden unit linearly partitions the input space into two parts based on W and B; In one part the unit is inactive with zero output, while in the other it is active with positive output. This allows for learning of regions around training examples, where each region is defined by the set of hidden units which are active in that region. Figure 2 shows a pictorial representation of this notion, where each hidden unit separates the space into an active and inactive part, and the overlap of these parts partitions the data. In each of these regions, the model has flexibility to learn a correction to the Quadratic Lag Model which better represents the dynamics locally. Furthermore, the total output of the network is piecewise linear in inputs and continuous over all regions in the input space since all the hidden units are piecewise linear and continuous in the inputs.

B. Optimization

For all the models presented in this work, the optimization problem in Equation 2 can be split into six subproblems, one for each component of $\tilde{f}$. These six subproblems are solved independently. In the case of the ReLU Network Model, this means training six different neural networks. Empirically we find that the component of $\tilde{f}$ corresponding to the up-down acceleration of the helicopter is the most difficult to predict.

Unlike the baseline models, Equation 2 for the ReLU Network Model is non-convex. We solve this optimization problem by first fitting the P, Q matrices with the output of η set to zero, using simple least-squares regression. This corresponds to fitting a Quadratic Lag Model. We then fix P and Q, and optimize over the weights and biases of the neural network (W, B, w, b) using Stochastic Gradient Descent (SGD) [27]. This entire procedure can be seen as one iteration of Block-Coordinate Descent.

Stochastic Gradient Descent is the standard optimization method for training recent neural network models [18, 22, 28] and is very simple to implement. In each iteration, the objective in Equation 2 is computed over a small subset of training data, called a minibatch. The gradient of the objective over the minibatch with respect to the parameters of the neural network is computed using Backpropagation [29]. An update step is computed as an average of the past update step and the current gradient, where the past step is weighted by a hyperparameter called momentum (typically set to 0.9). Parameters (weights and biases) are updated by the new update step, scaled by a learning rate. A new minibatch is randomly selected from training data and the next iteration proceeds. The learning rate is decreased during optimization. Section V-B provides specific details used in our experiments.

V. Experiments

A. Real-World Data

In order to investigate the efficacy of the baseline models and the ReLU Network Model, we use data from the Stanford Autonomous Helicopter Project [11]. The data was collected from a Synergy N9 model helicopter weighing 4.85 kg with a 720 mm main rotor blade length, shown in Figure 1. The

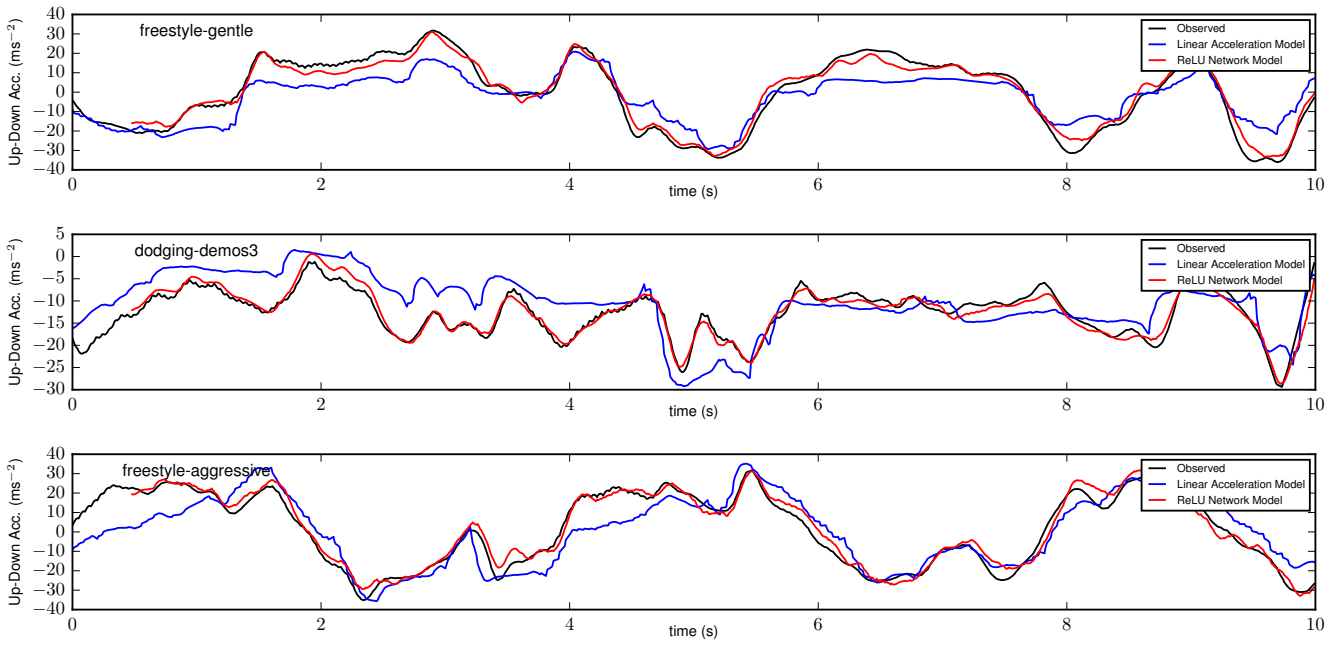

Fig. 3. Observed and predicted accelerations in the up-down direction over three difficult test-set trajectories corresponding to different aerobatic maneuvers. In all cases, the baseline Linear Acceleration Model captures the gross pattern of the acceleration over the maneuver, but performs very poorly compared to the novel ReLU Network Model.

helicopter is powered by a two-stroke engine. The helicopter was flown repeatedly through a variety of aerobatic maneuvers by an expert pilot. The recorded trajectories include smoothed estimates of the helicopter state s and controls u at 100Hz sample rate. Using these and Equation 1 we compute observed values of $\tilde{f}$ along the trajectories. In total there are 6290 seconds of flight time recorded, across 19 maneuvers.

We slice each trajectory into ten second long parts, resulting in 629 parts. A randomly selected 318 of these are used as the training set, 152 as a validation set, and 159 as a testing set. Although each trajectory part comes from a known maneuver, these labels are not used in training, and a single, global model is learned that is applicable across all maneuvers. The training set is used to optimize model parameters and the validation set is used to choose hyperparameters.

B. Optimization

For the baseline models, solving Equation 2 requires simply solving a linear least-squares problem. For the ReLU Network, we use SGD (see Section IV-B) with a momentum term of 0.95, a minibatch size of 100 examples, and train for 200,000 iterations. The learning rate is set at 1×10^{-8} and is halved after 100,000 iterations. The training set consists of 302,546 examples in total, meaning optimization makes about 66 passes through the dataset. To compute gradients of the objective in Equation 2, we use an automatic differentiation library called Theano [30].

C. Initialization

When optimizing neural networks in general, initialization is an important and often tricky problem. In the case of the ReLU Network Model, we provide a very simple, effective

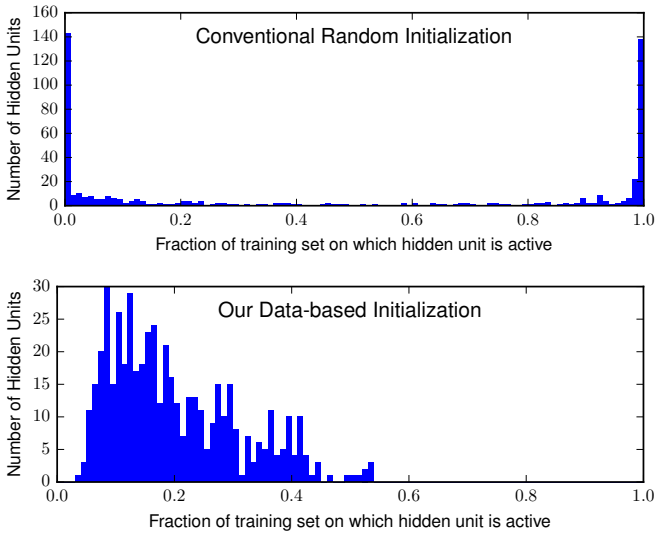

Fig. 4. Histograms of hidden unit activation over the training set, showing the advantage of our data-based initialization method. Hidden units which fall at either end of the histogram, being either always active or always inactive, are not useful to the ReLU Network Model. Our initialization method does not produce any units which are not useful, and requires no tuning or setting of hyperparameters.

and tuning-free method for initializing the weights and biases before beginning SGD. First, note that in the ReLU Network, any hidden unit is not useful if it is always active or always inactive over the space of trajectory segments. If the unit is always inactive, the error gradients with respect to the unit's weights will always be zero, and no learning will occur. If the unit is always active, its contribution to the network output will be entirely linear in the inputs, which will not allow for learning of any interesting structure not already captured by the Quadratic Lag Model. Conventional initialization tech-

niques for neural networks would call for initializing weight vectors randomly from a zero-mean Gaussian distribution. Empirically we find that this produces a very large fraction of hidden units which are always on or always off over the entire training set. Our initialization strategy is designed to solve this problem and remove the need for tuning or adjustment. We simply select examples randomly from the training set and use these as initial weight vectors. This corresponds to setting each hidden unit to compare the input with a direction in $\mathscr{P}$ *known* to contain some training points. Furthermore, we set the bias of each hidden unit uniformly at random so that the threshold between inactive and active lies somewhere between the origin of $\mathscr{P}$ and the selected exemplar. This ensures that the unit is unlikely to be always active. Our data-based initialization method is motivated by the use of data exemplars to initialize dictionaries in the context of sparse coding [31].

Figure 4 is a pair of histograms showing the distribution of hidden unit activations over the training set. With conventional random initialization, 56% of the hidden units are not useful after initialization, while with our data-based initialization, all the units have activations covering between 5% and 60% of the training set, making them all useful.

We find that after randomly initializing 5000 hidden units, training leads to an RMS prediction error of 4.27ms^{-2} on the held out validation set, while using our data-based initialization leads to an error of 2.70ms^{-2}. This performance increase is achieved without modifying learning rates or any other hyperparameters. Though it is likely that the performance of random initialization could be improved by further tuning the variances of random noise used, the tedious and expensive tuning process is eliminated entirely in our data-based initialization method.

D. Performance

We train the three baseline models (see Section III) and the ReLU Network Model on the training set, and report performance as Root-Mean-Square (RMS) prediction error on the held-out test set. RMS error is used because it normalizes for the length of trajectories and has meaningful units for each of the components of $\tilde{f}$ which are regressed. In all models, we use a horizon $H = 0.5$ s, and downsample our observed data to 50 Hz. We use $N = 2500$ hidden units for each of six neural networks in the ReLU Network model. The three baseline models are fit to the training set first. Each takes about 10 minutes to train over the entire training set, for all components. The ReLU Network Model is trained with SGD over 200,000 iterations, taking less than 1 hour for each component on a 6-core Intel i7 server with 32GB RAM.

Empirically, we find that the up-down acceleration direction is the most difficult component of $\tilde{f}$ to predict (see Figure 5), likely because the main rotor thrust is the strongest force on the helicopter. For this reason, we focus on the up-down component in displaying prediction results. The RMS error in up-down acceleration is measured in units of ms^{-2}, and results should be interpreted with the yardstick

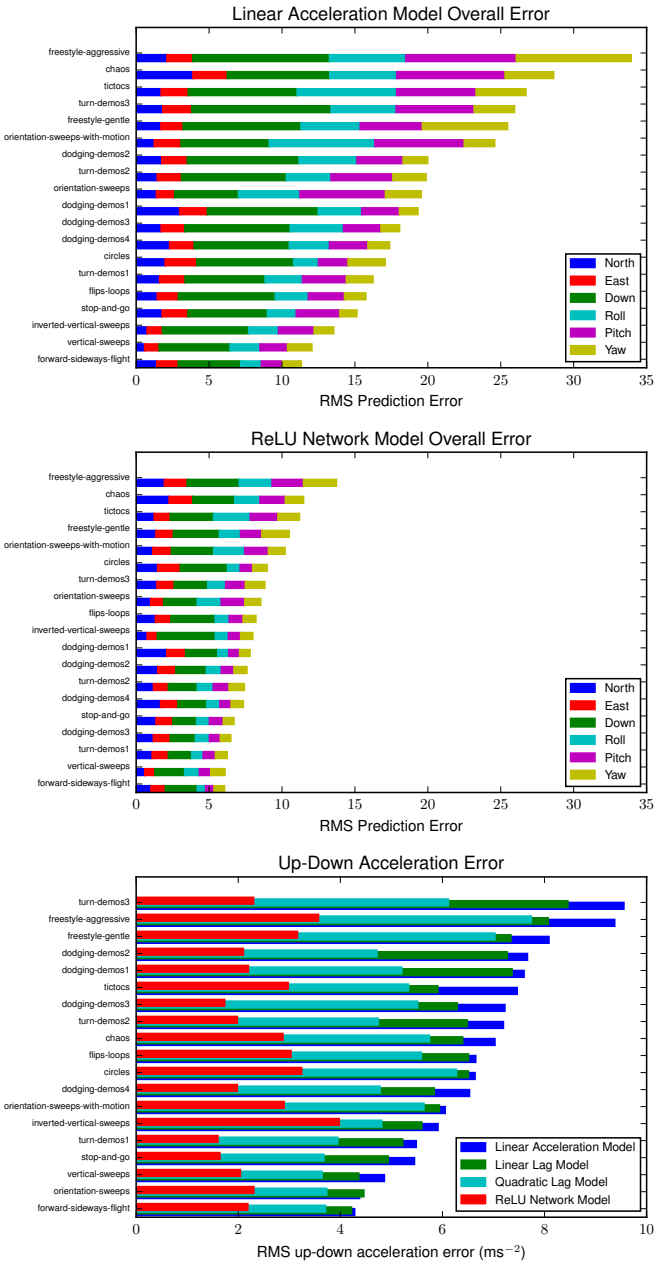

Fig. 5. Performance Comparison of the ReLU Network Model and baseline models from Section III. **Top:** Overall RMS prediction error of the Linear Acceleration Model over all acceleration components, each denoted by a different color. **Middle:** Overall RMS prediction error of the ReLU Model over all acceleration components. Accross all maneuvers, this result represents a **58% improvement** in RMS error over the Linear Acceleration Model above. **Bottom:** RMS Error in predicting the up-down acceleration of the helicopter on test-set trajectories, which is empirically found to be the most difficult component to predict. The Linear Acceleration Model (blue) is outperformed successively by the Linear Lag Model which accounts for control lag, and the Quadratic Lag Model which accounts for some simple nonlinearities. The ReLU Network Model significantly outperforms all baselines, demonstrating the importance of accounting for hidden system state. Performance is displayed separately for each maneuver, but the labels are not used during training. Across all maneuvers, the ReLU Network Model reduces up-down RMS error of the Linear Acceleration Model by **60%**. Note that the Linear baseline models actually correspond to nontrivial nonlinear dynamics models (see Section III).

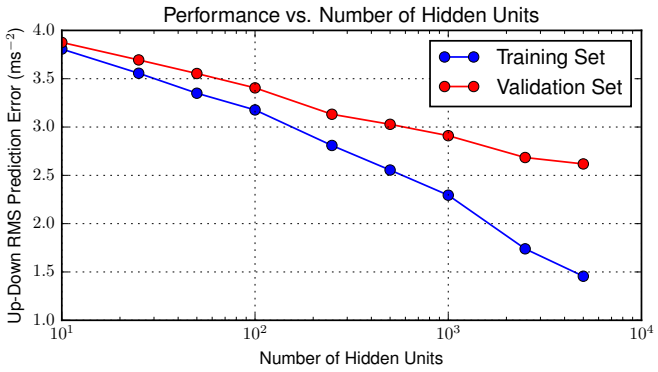

Fig. 6. Training and validation set performance as a function of the number of hidden units (N) used in the ReLU Network Model. Training time increases linearly with N. Training set performance continues to improve as more units are added, but validation set performance levels out indicating that further model complexity is not required. In our performance experiments we use $N = 2500$ hidden units.

of gravity $g = 9.8\text{ms}^{-2}$ in mind. Figure 3 shows the up-down acceleration as measured over three difficult trajectory segments, along with predictions along those trajectories from the Linear Acceleration Model and the ReLU Network Model. The baseline model captures some gross patterns, but the local corrections provided by ReLU Network Model perform much better.

E. Number of Hidden Units

Figure 5 shows the ReLU Network Model's RMS error on the testing set over all components, as well as for the up-down direction in particular. The ReLU Network significantly outperforms the baselines, showing that directly regressing from state-control history space $\mathscr{P}$ to acceleration captures important system dynamics structure. In overall prediction error, the ReLU Network model improves over the Linear Acceleration baseline by 58%. In up-down acceleration prediction, the ReLU Network model has 60% smaller RMS prediction error than the Linear Acceleration baseline, improving from 6.62ms^{-2} to 2.66ms^{-2}. The performance improvements of the Linear Lag and Quadratic Lag Models over the Linear Acceleration Model are also significant, and show that control lag and simple nonlinearities are useful parts of the helicopter system to model. In Figure 5 the errors are shown separately across maneuvers, to illustrate the relative difficulty of some maneuvers over others in terms of acceleration prediction. Maneuvers with very large and fast changes in main rotor thrust (aggressive freestyle, chaos) are the most difficult to predict, while those with very little change and small accelerations overall (forward/sideways flight) are the easiest.

It is important to note that there are more inputs to the ReLU Network Model than to the baseline models. Specifically, the past states $x_{t-H}, x_{t-H+1}, \ldots x_{t-1}$ are included in the input to the ReLU Network Model, while the baseline models only have access to x_t. This raises the question of whether a simpler model given the same input would be able to capture system dynamics as effectively as the

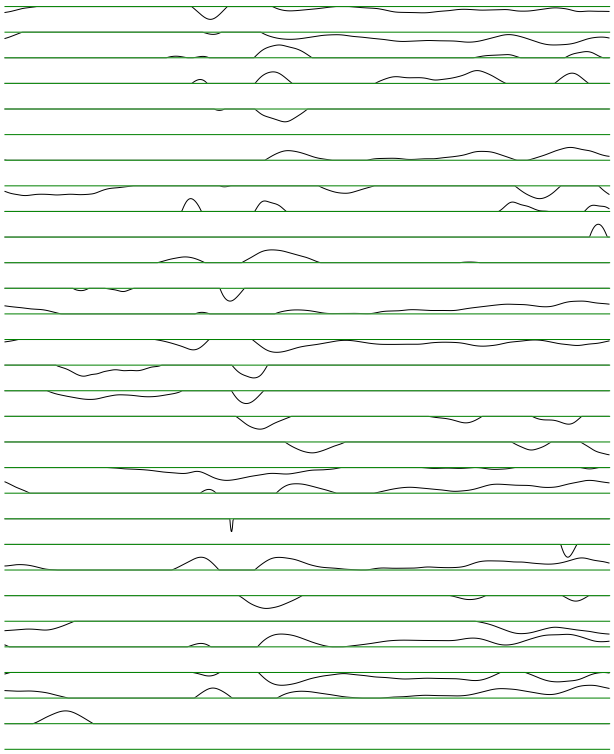

Fig. 7. Activations of a random selection of hidden units, along a test-set trajectory. Green lines are zero activity for each unit displayed. Interestingly, the hidden units are inactive over most of the trajectory, and only become active in particular regions. This observation corresponds well to the intuition that motivated the ReLU Network Model, as explained in Section IV.

ReLU Network Model. To answer this question, we train an extended Linear Lag Model using both past states and past controls as inputs. This model has the same input at prediction time as the ReLU Network Model. We find, however, that this extended Linear Lag Model performs only marginally better than the original Linear Lag Model; performance on up-down acceleration prediction is improved by only 2.3%. Thus the ReLU Network Model's ability to represent nonlinear locally corrected dynamics in different regions of the input space allow it to extract structure from the training data which such simpler models cannot.

Our work in developing the ReLU Network Model was inspired by the challenges of helicopter dynamics modeling, but the resulting performance improvements indicate that this approach might be of interest in system identification beyond helicopters.

In Figure 6 we show the results of an experiment evaluating the impact of changing the number of hidden units in the ReLU Network Model. We find that, as expected, having more hidden units improves training set performance. Performance on the held out validation set, however, improves more slowly as N is increased, and there are diminishing returns. Gradient computation time for training is linear in N, which is why we chose $N = 2500$ as a suitable number of hidden units for the experiments in Section V-D.

F. Sparse Activation

The ReLU Network Model outperforms the Linear Acceleration Model baseline by a large margin. Some of the improvement is due to accounting for control lag and simple nonlinearity (as in the Linear and Quadratic Lag Models), and the remainder is due to the flexibility of the neural network. Along with quantifying the performance of the ReLU Network Model, it is important to investigate the intuition motivating the model, namely that the hidden units of the neural network partition the space $\mathscr{P}$ and their pattern of activations corresponds to different regions which have similar dynamics. A simple way to understand the influence of each hidden unit is to visualize the activations of the hidden units along a particular trajectory. Figure 7 shows normalized activations of a random subset of hidden units along a trajectory from a single maneuver.

Interestingly, each hidden unit is mostly inactive over the trajectory shown, and smoothly becomes active in a small part of the trajectory. We find this to be true in all the trajectory segments we have visualized. This observation fits well with the motivating intuition behind the ReLU Network Model.

VI. Conclusion and Future Work

In this work, we define the ReLU Network Model and demonstrate its effectiveness for dynamics modeling of the aerobatic helicopter. We show that the model outperforms baselines by a large margin, and provide a complete method for initializing parameters and optimizing over training data. We do not, however, answer the question of whether this new model performs well enough to enable autonomous control of the helicopter through aerobatic maneuvers. Future work will aim to answer this question, either through simulation with MPC in the loop, or through direct experiments on a real helicopter. We will also investigate the efficacy of this model for modeling other dynamical systems.

Acknowledgment

This research was funded in part by DARPA through a Young Faculty Award and by the Army Research Office through the MAST program.

References

[1] J. M. Seddon and S. Newman, *Basic helicopter aerodynamics*. John Wiley & Sons, 2011, vol. 40.

[2] J. G. Leishman, *Principles of Helicopter Aerodynamics with CD Extra*. Cambridge university press, 2006.

[3] G. D. Padfield, *Helicopter flight dynamics*. John Wiley & Sons, 2008.

[4] W. Johnson, *Helicopter theory*. Courier Dover Publications, 2012.

[5] M. B. Tischler and M. G. Cauffman, "Frequency-response method for rotorcraft system identification: Flight applications to BO 105 coupled rotor/fuselage dynamics," *Journal of the American Helicopter Society*, vol. 37, no. 3, pp. 3–17, 1992.

[6] B. Mettler, M. B. Tischler, and T. Kanade, "System identification of small-size unmanned helicopter dynamics," *Annual Forum Proceedings- American Helicopter Society*, vol. 2, pp. 1706–1717, 1999.

[7] J. M. Roberts, P. I. Corke, and G. Buskey, "Low-cost flight control system for a small autonomous helicopter," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 546–551.

[8] M. La Civita, W. C. Messner, and T. Kanade, "Modeling of small-scale helicopters with integrated first-principles and system-identification techniques," *AHS International, 58 th Annual Forum Proceedings-*, vol. 2, pp. 2505–2516, 2002.

[9] V. Gavrilets, I. Martinos, B. Mettler, and E. Feron, "Control logic for automated aerobatic flight of miniature helicopter," *AIAA Guidance, Navigation and Control Conference*, no. 2002-4834, 2002.

[10] ——, "Flight test and simulation results for an autonomous aerobatic helicopter," in *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, vol. 2. IEEE, 2002, pp. 8C3–1.

[11] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, 2010.

[12] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 26, no. 1, pp. 43–49, 1978.

[13] J. Listgarten, R. M. Neal, S. T. Roweis, and A. Emili, "Multiple alignment of continuous time series," in *Advances in neural information processing systems*, 2004, pp. 817–824.

[14] Z. Ghahramani and G. E. Hinton, "Parameter estimation for linear dynamical systems," Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science, Tech. Rep., 1996.

[15] S. Roweis and Z. Ghahramani, "An EM algorithm for identification of nonlinear dynamical systems," 2000.

[16] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*. Springer, 1981, pp. 366–381.

[17] J. C. Robinson, "A topological delay embedding theorem for infinite-dimensional dynamical systems," *Nonlinearity*, vol. 18, no. 5, p. 2135, 2005.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[20] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. A. Ranzato, and T. Mikolov, "DeViSE: A deep visual-semantic embedding model," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2121–2129.

[21] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernndez, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2008, pp. 577–584.

[22] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and others, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[23] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal, "Identification of hybrid systems a tutorial," *European journal of control*, vol. 13, no. 2, pp. 242–260, 2007.

[24] P. Abbeel, V. Ganapathi, and A. Y. Ng, "Learning vehicular dynamics, with application to modeling helicopters," in *Advances in Neural Information Processing Systems*, 2005, pp. 1–8.

[25] S. A. Whitmore, *Closed-form integrator for the quaternion (euler angle) kinematics equations*. Google Patents, May 2000, US Patent 6,061,611.

[26] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 215–223.

[27] L. Bottou, "Online learning and stochastic approximations," *On-line learning in neural networks*, vol. 17, p. 9, 1998.

[28] O. Bousquet and L. Bottou, "The tradeoffs of large scale learning," in *Advances in neural information processing systems*, 2008, pp. 161–168.

[29] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Mller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[30] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Austin, TX, Jun. 2010, oral Presentation.

[31] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *ICML-11*, pp. 921–928.