CS229 Project: Learning Dynamic Models from Data
Project Report

Alex Maynard (06040102) and Jason Anderson (06063625)

I. Introduction

Control in robotics applications requires knowledge of how the robot dynamically changes with time in response to environmental factors and actuator inputs. Existing methods generally assume dynamic models of the system, such as transfer functions or state space equations. These methods are useful for systems approximated as deterministic, linear, and time-invariant, and can be modified to account for stochasticity and nonlinearity. However, these models are often inadequate for environments with complex time-variance, especially if the nature of the dynamics' change over time is unknown *a priori*. For example, consider a self-driving car that uses models based on summer driving conditions and doesn't perform well in winter because of icy road conditions; a space exploration robot that experiences wear-and-tear that gradually alters its dynamics over many years; or an autonomous delivery drone that passes through unexpected, inclement weather conditions *en route* to its destination. In all of these scenarios, the autonomous system would quickly stop being useful if it retained the same control scheme and dynamic model as it used when initially deployed.

The challenges time-variance presents to more traditional control methods motivate adaptive algorithms that continually update the robot's dynamic models to minimize the discrepancy between expected and actual observations. With this project, we will explore the application of machine learning techniques to identify those models from data.

In particular, we investigate the challenge of system identification for controls tasks. System identification is a task in which an algorithm designs a model for a dynamical system based on collected data. Such data typically consists of a series of control inputs and the corresponding observed outputs. System identification is generally the first step in designing robotic controllers. We use methods from two different families of system identification algorithms as baselines: autoregression with exogenous terms (ARX), which represents system dynamics as a transfer function between inputs and outputs; and numerical subspace state space system identification (N4SID), which represents system dynamics as a state-space model. The goal of this project is to compare the performance of ARX and N4SID to that of a neural network for several tasks, in particular: modelling a synthetic, 2-dimensional, single-input-single-output (SISO), linear time-invariant (LTI) system; and modelling a nonlinear, time-variant, simulated racecar. We also investigated which inputs provided the best training data for dynamic models.

This report is organized as follows. The remainder of this section reviews relevant background for ARX, N4SID, and neural networks. Section II discusses the details of our implementations for each algorithm. The experiment details and results are described in section III. Finally, we review conclusions and contributions in sections IV and V respectively.

A. ARX

Consider the typical linear-time-invariant ("LTI") $\mathcal{H}$ defined with Equation (1), where z is the delay operator, n_p is the number of poles, and n_z is the number of zeros.

$$\begin{aligned} a(z)y_t &= b(z)u_t \\ a(z) &= 1 + \alpha_1 z^{-1} + \cdots + \alpha_{n_p} z^{-2} \\ b(z) &= \beta_1 z^{-1} + \cdots + \beta_{n_z} z^{-2} \end{aligned} \tag{1}$$

Let input and output data sets from which to learn $\mathcal{H}$ be U and Y, respectively. To compute an ARX model, we set up the following linear regression problem.

$$Y = \begin{bmatrix} 0 & 0 & \cdots & | & U & 0 & \cdots & | \\ -Y & 0 & \cdots & 0_{n_p} & | & U & \cdots & 0_{n_z} \\ | & -Y & \cdots & | & | & | & \cdots & | \\ | & | & \cdots & -Y & | & | & \cdots & U \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{n_p} \\ \beta_1 \\ \vdots \\ \beta_{n_z} \end{bmatrix} \tag{2}$$

For brevity of notation in Equation (1), we did not explicitly notated that the shifted Y and Us must be truncated at the bottom of the data matrix of the regression problem.

B. Subspace Methods

Consider the typical LTI system $\mathcal{S}$ defined with Equation (3).

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t \in \mathbb{R}^n \\ y_t &= Cx_t + Du_t + v_t \in \mathbb{R}^p \\ u_t &\in \mathbb{R}^m \\ t &\in 0...N \end{aligned} \tag{3}$$

We desire to learn the best set of parameters of $\mathcal{S}$, which are matrices A, B, C, D from the input and output data sets U and Y, respectively.

In addition to knowing U and Y, suppose also knew the the state history X as well. Then learning $\mathcal{S}$ becomes the simple regression problem of Equation (4).

$$\begin{bmatrix} x_{t+1} \\ y_t \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} w_t \\ v_t \end{bmatrix} \tag{4}$$

The solution to Equation 4 is derived via the following.

$$\begin{aligned} \begin{bmatrix} x_{t+1} \\ y_t \end{bmatrix} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} w_t \\ v_t \end{bmatrix} \\ \mathcal{Y}^\top &= \Theta^\top \mathcal{X}^\top \\ \mathcal{Y} &= \mathcal{X}\Theta \\ \Theta &= (\mathcal{X}^\top \mathcal{X})^{-1} \mathcal{X}^\top \mathcal{Y} \\ \Theta^\top &= \mathcal{Y}^\top \mathcal{X} (\mathcal{X}\mathcal{X}^\top)^{-1} \\ \begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \sum_t \begin{bmatrix} x_{t+1} \\ y_t \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^\top \left(\begin{bmatrix} x_t \\ u_t \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^\top \right)^{-1} \end{aligned}$$

The novelty of N4SID, is how it computes X from Y and U via a combination of LQ decomposition and SVD [1], for which we adapt and summarize here.

First, let us define Hankel Matrices on the data as follows.

$$U_{0|k-1} = \begin{bmatrix} u_0 & u_1 & \cdots & u_{N-1} \\ u_1 & u_2 & \cdots & u_N \\ \vdots & \vdots & \ddots & \vdots \\ u_{k-1} & u_k & \cdots & u_{N+k-2} \end{bmatrix} \in \mathbb{R}^{km \times N}$$

$$Y_{0|k-1} = \begin{bmatrix} y_0 & y_1 & \cdots & y_{N-1} \\ y_1 & y_2 & \cdots & y_N \\ \vdots & \vdots & \ddots & \vdots \\ y_{k-1} & y_k & \cdots & y_{N+k-2} \end{bmatrix} \in \mathbb{R}^{kp \times N}$$

$$\begin{aligned} U_p &= U_{0|k-1} \\ U_p &= U_{k|2k+1} \\ Y_p &= Y_{0|k-1} \\ Y_p &= Y_{k|2k+1} \\ W_p &= \begin{bmatrix} U_p \\ Y_p \end{bmatrix} \end{aligned}$$

Then we perform the follow LQ decomposition.

$$\begin{bmatrix} U_f \\ W_p \\ Y_f \end{bmatrix} = \begin{bmatrix} R_{11} & 0 & 0 \\ R_{21} & R_{22} & 0 \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} Q_1^\top \\ Q_2^\top \\ Q_3^\top \end{bmatrix}$$

Then we perform an oblique projection.

$$\xi = R_{32} R_{22}^\dagger W_p$$

Then we perform an SVD.

$$\xi = U \Sigma V^\top$$

Finally, we arrive at the state vector.

$$X = \Sigma^{1/2} V^\top \tag{5}$$

We see that to learn the LTI system, subspace methods use LQ decomposition, SVD, and linear regression on input and output data.

C. Neural Networks

Control theorists rely on especially strong mathematical guarantees and first-principles derivations for their controllers and algorithms to work. As such, neural networks were an unpopular tool in the era of model-based control. However, a new paradigm of data-driven control [2] has made neural networks increasingly popular for controls tasks.

Most work on the intersection between machine learning and control theory has been on deep reinforcement learning algorithms, such as deep Q-networks [3] or actor-critics [4]. Additionally, control tools such as filters and adaptive controllers operate on algorithms similar to machine learning that iteratively estimate system parameters. We focus instead on the application of supervised learning to system identification. Neural networks are well-suited for this task because simulated systems can be used to generate a wealth of training samples with little effort. For this project, we consider a network designed to learn a mapping from past inputs and outputs to future outputs of a dynamical system.

II. Implementation

For the ARX and N4SID methods, we used an open source python implementation called SIPPY available on GitHub [5]. This implementation depends on several other open source libraries, such as the Python Control Systems Library.

We use PyTorch [6] to implement a neural network for observation prediction. The network takes in a series of past observations and actions and outputs a prediction of the observation at the next timestep. Each hidden layer uses a ReLU activation function, the loss function is Mean-Squared Error (MSE), and the optimizer is Adam. The size of the network changes slightly for different experiments.

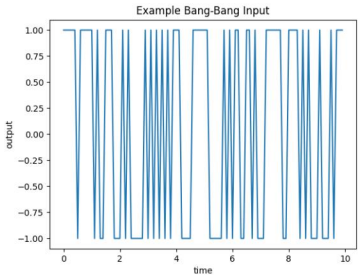

Fig. 1. An example bang-bang input used to simulate a system to generate training data.

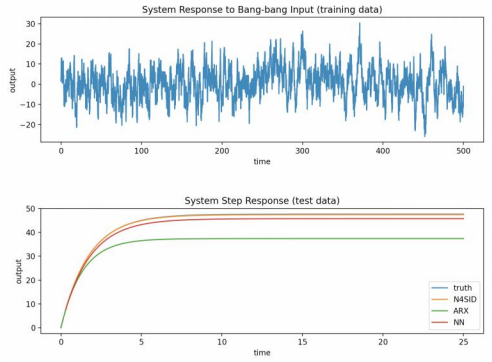

Fig. 2. (top) Pseudorandom data use to train each model. (bottom) Each trained model's step response compared to the true system's step response.

III. Experiments

A. Step Response of LTI System

To evaluate the methods presented here, we generate data sets via simulation. Since the underlying system is a difference equation with process and measurement noise, we can arbitrarily generate training and test data. The efficacy of the subspace methods depends on the underlying rank of matrices in the procedure summarized in Section I-B, the best training data would be to simulate the system under a psuedorandom input. From prior experience of the authors, the best psuedorandom input is a bang-bang input. That is, to generate our train data, we simulated the system we desire to learn under a psuedorandom input the switches from a maximum and minimum input. An example bang-bang input is depicted in Figure 1. In total, we generated 5000 input-output samples to train each model on. For this experiment, the neural network had a single 20-unit layer and is trained on an action-observation history 3 timesteps long. The neural network was trained over 130 epochs, in each of which the data was shuffled and separated into minibatches of size 64. We used a learning rate of 0.0001 for all training.

To evaluate our models, we compare their predictive performance for a step response. A step response is a common metric for dynamical systems in which the system is brought to a stable equilibrium and the following control law is applied:

$$u_t = \begin{cases} 0 & \text{for } t < 0 \\ 1 & \text{for } t \geq 0 \end{cases} \tag{6}$$

The step response is considered a standard evaluation metric for dynamical models and their controllers—if a model cannot accurately represent a step response, it is indicative of underlying problems, and can be used to troubleshoot design flaws. For this project, we simply use it as a comparison metric, as each algorithm should ideally be able to model the step response accurately.

Fig. 2 shows that while N4SID is indistinguishable from the true system response, the neural network is comparable. ARX's performance is noticeably worse than the other two.

B. Selecting Random Training Data

In order for N4SID to arrive at a learned system, the training data must contain amount of entropy. Using training data, holding the system at a static value will cause the SVD operation to break. Supplying entropic training data also adds to the efficacy of the neural network model. This leads to the question of what type of random data should be used for the training data.

This work studies 3 type of psuedorandom signals: bang-bang, normal, and uniform. A bang-bang input derives from a pseudo-random binary sequence. When that sequence is 0, the control input is set to minimum, and when the sequence 1, the control input is set to maximum. This sequence should provide the maximum entropy to the system, since it switches from maximum and minimum control inputs, but it also poses the greatest risk of destructive testing. A normal sequence samples from a normal distribution. And a uniform sequence derives from a uniform distribution. Both the bang-bang and uniform inputs, the distributions have maximum and minimum values of the system control inputs. For the normal input, the distribution is approximated normal but setting the 3-sigma value to be the maximum and minimum values. Values exceeding the maximum and minimum control inputs are clipped, but this occurs less than 1% of the time.

In our experiment, we trained N4SID and neural network models on each training data type. The output of the simulated data included random normal noise on the output so that the models received noisy simulated data. The models were tested on a isolated test set that was a combination of periodic functions and step inputs. Figure 3 shows a comparison of the trained model on the test

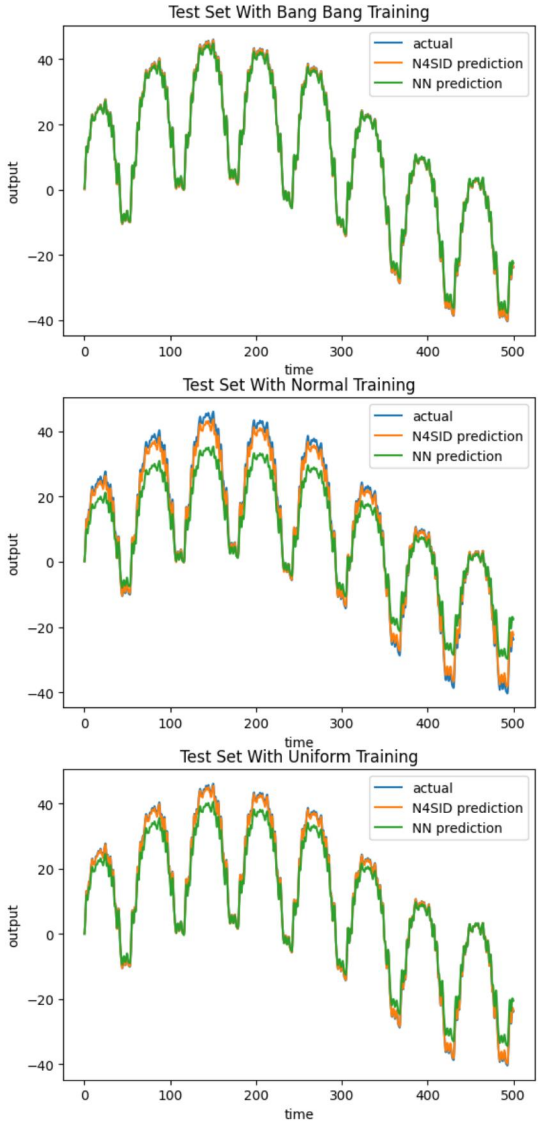

Fig. 3. Comparison of predictions between models trained on different random inputs.

set, and Table I provides the final test prediction errors. These errors are the RMS errors on the actual simulated data from the true model.

For the N4SID models, as expected, the bang-bang input produced the best results. This derived from the bang-bang input supplying the most entropy for the SVD operation within the method. However, these differences are almost indistinguishable compared to the neural network results. A much larger error difference was observed among the neural network models, with the bang-bang outperforming the others. Therefore, we conclude the best training data is the bang-bang input.

Model	Bang-Bang	Normal	Uniform
N4SID Test Error	0.0203	0.1860	0.0407
NN Test Error	0.1142	0.8249	0.4207

TABLE I
FINAL RMS TEST ERRORS AGAINST ACTUAL SIMULATED MODEL FOR FIGURE 3.

C. Modelling Nonlinear, Time-variant Dynamics

Our last experiment was to compare the three algorithms' performance when modelling a system with nonlinear, time-variant dynamics. We used OpenAI Gym's CarRacing environment [7]. In this environment, the 6 observed state variables are the car's position (x, y), linear velocity ($\dot{x}$, $\dot{y}$), orientation (θ), and angular velocity ($\dot{\theta}$). The control input consists of a 3-dimensional vector whose elements represent the turning signal, which ranges from -1 (left) to +1 (right); the acceleration, which ranges from 0 (none) to +1 (full throttle); and the brake, which ranges from 0 (none) to +0.8 (full brake). The car not only experiences friction (making its dynamics nonlinear), but also experiences different magnitudes of friction from different surfaces in the environment. In particular, there is a road with low coefficient of friction on a patch of grass with high coefficient of friction. As such, the car dynamics vary over time as the car drives on and off the road. This makes the environment especially challenging for traditional system identification algorithms.

A pseudo-random control sequence was generated over 2000 timesteps and fed into 2 different environments: one that was exclusively grass and one that was exclusively road. We trained all 3 algorithms on the grass and road data separately to construct 6 models in total: one grass and one road model for each system identification algorithm. The neural network consisted of 2 hidden layers of 20 units, which we trained on the past 10 timesteps of action-observation data. Furthermore, the neural network was outfitted with a classifier that determined the most likely model at any given timestep based on which model was closest to the true observations over the past 3 timesteps. We refer to these models in combination with the classifier as the bimodal system, as it models both the grass and road dynamical modes simultaneously.

Figure 4 shows the performance of the neural network models after 300 epochs with a learning rate of 2e-4. The models evidently learned the dynamics very well, and the full bimodal system was reasonably capable of predicting

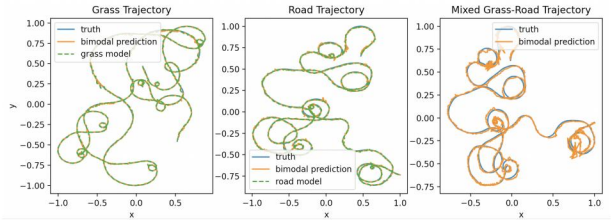

Fig. 4. Performance of the neural network car models on road, grass, and a mixed course.

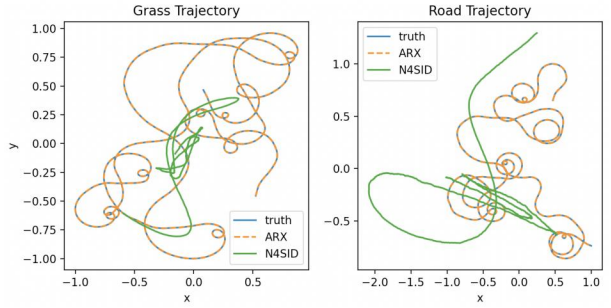

Fig. 5. Performance of the ARX and N4SID car models on road and grass.

future observations given the changing dynamics of the car's environment.

Figure 5 shows the performance of the more traditional ARX and N4SID algorithms given the same training data. While ARX was evidently capable of capturing the underlying car dynamics on grass and road, N4SID struggled substantially. This is likely due to high nonlinearity in the car model, as N4SID would be expected to work the best of the 3 algorithms on a linear model.

While ARX appears to be the best algorithm for predicting individual models, the neural network is overall more capable because we can also apply machine learning to determine whether the vehicle is on grass or road at any given time.

IV. Conclusions

This project compared the performance of system identification algorithms from 3 different paradigms of control: classical (ARX transfer functions), modern (N4SID state-space models), and data-driven (neural networks). We performed 3 experiments in total. The first experiment was a comparison between all 3 models for identification and step response of a linear system. N4SID performed the best, but the neural network's performance was comparable. The second experiment we determined that psuedorandom bang-bang input provided the best training data. The third experiment compared the performance of all 3 algorithms for a more challenging simulation that had nonlinear, time-varying dynamics. In this case, the neural network was overall the most robust model.

While neural networks cannot give the kind of guarantees and high-level performance we see from more traditional algorithms in LTI systems, they appear overall more robust for more realistic tasks.

V. Contributions

Jason reviewed the necessary literature for implementing N4SID and ARX, designed the data collection and step response experiment, and performed the random training data comparison. Alex designed the infrastructure for the neural network, trained it for the step response task, and performed the car model experiment. Both authors collaborated to write the final report.

References

[1] T. Katayama and H. Tanaka, "An approach to closed-loop subspace identification by orthogonal decomposition," *Automatica*, vol. 43, no. 9, pp. 1623–1630, 2007.

[2] "Data-driven control and learning systems," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 4070–4075, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[5] G. Armenise, M. Vaccari, R. B. Di Capaci, and G. Pannocchia, "An open-source system identification package for multivariable processes," in *2018 UKACC 12th International Conference on Control (CONTROL)*, pp. 152–157, 2018.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.