Sparse Identification of Nonlinear Dynamical systems (SINDy)

Aayushi Shrivastava
aayushis@umich.edu

Pratik Shiveshwar
spratik@umich.edu

***Abstract*—Discovering the governing equations from scientific data becomes easier using data-driven approaches. Sparse regression enables the tractable identification of both the structure and parameters of a nonlinear dynamical system from data. The resulting models have the fewest terms necessary to describe the dynamics, balancing model complexity with the descriptive ability and thus promoting interpretability and generalizability. In this work, we design a custom autoencoder to discover a coordinate transformation into a reduced space where the dynamics may be sparsely represented. We combine the strength of the autoencoder for coordinate representation in a reduced state of the system and sparse identification of nonlinear dynamics (SINDy) for parsimonious models. We implemented this method on the planar pushing task and compared it against a globally linear, Embed to Control(E2C) latent space model.**

***Index Terms*—autoencoder, sparse, non-linear modeling**

I. Introduction

Non-linear and high-dimensional systems are traditionally difficult to analyze and control using classical dynamical systems theory. Neural networks can learn the underlying governing equations of a system. But the high-dimensionality of the input space like images can make modeling difficult. Autoencoders parameterized by neural networks can be effective for nonlinear dimensionality reduction. In the latent space, different methods can be used to learn the latent space dynamics of the system. A central tension in model discovery is the balance between model efficiency and descriptive capabilities like a linear model loses information about the system and thus makes its control less accurate. Parsimonious models strike this balance, having the fewest terms required to capture essential interactions and not more. Parsimonious models tend to be more interpretable and generalizable.

The sparse identification of nonlinear dynamics (SINDy) [2] algorithm is a mathematical regression technique for extracting parsimonious dynamics from time-series data. It performs a sparse regression on a set of candidate basis functions, with the aim of finding the equations that govern the system. SINDy is particularly useful when the system is complex, nonlinear, and high-dimensional, which makes it difficult to model using traditional approaches. The algorithm leverages sparsity to provide a concise and meaningful representation of the system's dynamics, which is valuable in scientific applications. Additionally, SINDy is computationally efficient and can handle noisy and incomplete data, making it a powerful tool for discovering complex systems using data.

For example, [3] shows how SINDy can be used for modeling the dynamics of a system defined by partial differential equations. [4] states Hybrid-Sparse Identification of Nonlinear Dynamics, which identifies separate nonlinear dynamical regimes employ information theory to manage uncertainty and characterize switching behavior.

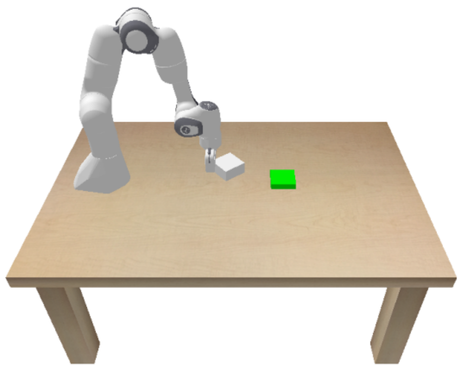
Fig. 1. Gym Environment

II. Implementation

A. Model Environment

We use Gym Environment to simulate a physical system.

As shown in Figure 1, the physical world is a table with a robot arm pushing a block. The green block is the target position of the block. The robot pushes the block at a single point of contact.

Figure 2 shows the action space. The robot pushes the block at distance p, and pushes it l meter, with a pushing angle of ϕ.

Each action $\mathbf{u} = \begin{bmatrix} p & \phi & \ell \end{bmatrix}^\top \in \mathbb{R}^3$ is composed by:

- $p \in [-1, 1]$: pushing location along the lower block edge.
- $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ pushing angle.
- $\ell \in [0, 1]$ pushing length as a fraction of the maximum pushing length. The maximum pushing length is 0.1 m

The state space is a 32×32 image from an overhead camera.

B. Model Architecture

The model consists of a state encoder, which maps images into latent vectors, a SINDy model which represents the

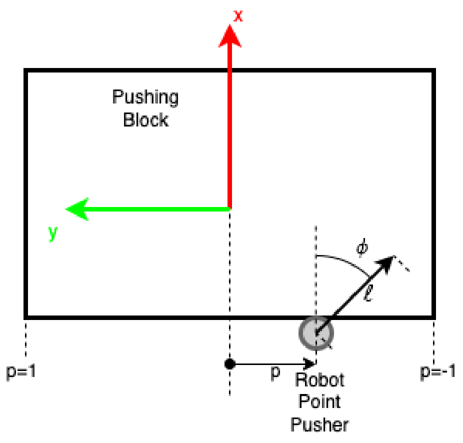

Fig. 2. Planar Pushing

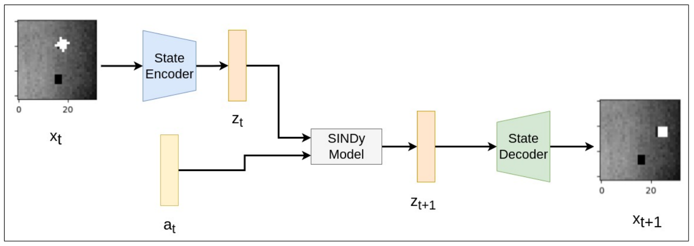

Fig. 3. Model Architecture

dynamics in the latent space and then a decoder which maps the latent vector make to images.

Figure 3 shows the simplified block diagram of the Model Architecture. The State Encoder encodes the data from a 32x32 pixel image into a latent space of size 16 as shown in fig 4. The input action is of size 3, which is used to predict

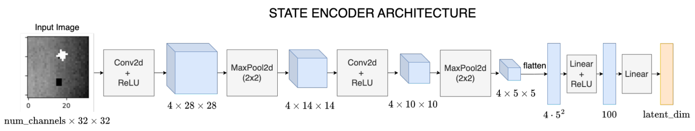

Fig. 4. State Encoder Architecture

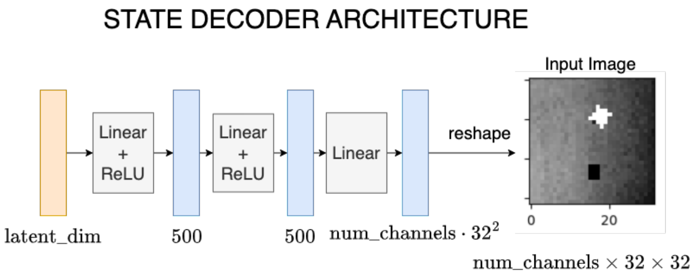

Fig. 5. State Decoder Architecture

the next state. Fig. 5 shows the architecture for decoder which converts the predicted next latent state to the image of 32x32 pixels.

We used a multi-step loss function to train the autoencoder and the latent dynamics model end-to-end. A single training trajectory is given by a sequence of images $x_0, x_1, ..., x_T$, and controls $a_0, ..., a_{T-1}$. We encode every image to a latent vector z with the encoder to produce $z_0, z_1, ..., z_T$. In other words, $z_t = \Delta(x_t)$.

From the initial latent state z_0 we apply the latent dynamics model recursively, i.e. $\hat{z}_1 = f_{latent}(z_0, a_0)$, $\hat{z}_2 = f_{latent}(\hat{z}_1, a_1)$ and so on. This produces predictions $\hat{z}_1, ..., \hat{z}_T$. Similarly, we do the same for states. Starting at x_0, we propagate state dynamics to produce $\hat{x}_1, ..., \hat{x}_T$, where $\hat{x}_{t+1} = f_{\text{dyn}}(\hat{x}_t, a_t)$, and $\hat{x}_1 = f_{\text{dyn}}(x_0, a_0)$.

The full loss is then:

$$L_{recon} = \sum_{t=0}^{T} \frac{1}{T+1} ||x_t - \nabla(z_t)||^2 \tag{1}$$

$$L_{pred} = \sum_{t=1}^{T} (\frac{\lambda_1}{T} ||x_t - \hat{x}_t||^2 + \frac{\lambda_2}{T} ||z_t - \hat{z}_t||^2) \tag{2}$$

$$L_{reg} = \lambda_3 ||\psi||_1 \tag{3}$$

$$L_{tot} = L_{recon} + L_{pred} + L_{reg} \tag{4}$$

where ψ is the SINDy coefficient matrix. λ_1, λ_2, and λ_3 are hyperparameters that give weightage to different losses like state prediction loss and latent state prediction loss, and SINDy coefficient matrix sparsity loss. This loss term has three components, and the first is a standard reconstruction loss. The second loss is on the latent dynamics prediction and the last one is the SINDy coefficient matrix sparsity loss.

C. SINDy Model

The sparse identification of nonlinear dynamics (SINDy) [2] algorithm is a mathematical regression technique for extracting parsimonious dynamics from time-series data. The method takes snapshot data $x(t) \in R^n$ and attempts to discover a best-fit dynamical system with as few terms as possible:

$$\frac{dx(t)}{dt} = f(x(t))$$

The snapshots represent measurements of the state of the system in time t, and the function f constrains how the dynamics of the system evolve in time. We seek a parsimonious model for the dynamics, resulting in a function f that contains only a few active terms: it is sparse in a basis of possible functions. SINDy frames model discovery as a sparse regression problem. Assuming snapshot derivatives are available or can be calculated from data, the snapshots are stacked to form data matrices. Although f is unknown, we can construct an extensive library of p potential candidate basis functions $\Theta(X) = [\theta_1(X)...\theta_p(X)] \in R^{mxp}$, where each θ_j is a candidate basis function or model term. We assume $m >> p$ so that the number of data snapshots is much larger than the number of candidate library functions. The basis functions are polynomials in x as these are elements of many canonical models of dynamical systems. We also included $sin(x)$ functions in the library. The library is used to formulate a regression problem to approximately solve the overdetermined linear system of equations [1].

$$\dot{X} = \Theta(X)\psi \tag{5}$$

where the unknown matrix ψ =$[\zeta_1\zeta_2...\zeta_n] \in R^{pxn}$ is the set of coefficients that determine the active terms from $\Theta(X)$ in the dynamics f.

D. Controller

We used Model Predictive Path Integral (MPPI), a trajectory optimization algorithm fro controlling the Panda arn. This method uses a dynamics model (a simulator, a learned model, or an analytical model) to roll out trajectories. This method has been shown to be more efficient and useful for high-dimensional spaces than other model predictive control (MPC) approaches [6]. MPPI propagates the dynamics in the latent space. Moreover, the cost function is formulated in latent space, comparing how close states are to the goal in latent space. The steps in MPPI Controller are as follows:

Rollout Dynamics: Given an initial state and a set of perturbed actions, it computes the resultant states.

Compute $\mathbf{z}_0, \ldots, \mathbf{z}_T$ for each of the K trajectories, where

$$\mathbf{z}_t^k \leftarrow \mathbf{F}(\mathbf{z}_{t-1}^k, \mathbf{u}_{t-1} + \epsilon_{t-1}^k)$$

where ϵ_t^k are the perturbation applied to the nominal action sequence $\mathbf{u}_0, \ldots \mathbf{u}_{T-1}$ (actions without perturbations).

Compute Trajectory Cost: Given K different trajectories and control perturbations ϵ, it computes the cost associated with each of the K trajectories. For the state cost, here we used a quadratic state cost with no terminal cost, i.e.

$$\text{cost}^k = \sum_{t=1}^{T} (\mathbf{z}_t^k - \mathbf{z}_{\text{goal}})^\top \mathbf{Q} (\mathbf{z}_t^k - \mathbf{z}_{\text{goal}}) + \lambda \mathbf{u}_{t-1}^\top \Sigma^{-1} \epsilon_{t-1}^k$$

Nominal Trajectory Update: Once we have the trajectory costs computed, we will update the nominal trajectory.

$$\begin{gathered}
\beta \leftarrow \min_k[\text{cost}^k] \\
\gamma^k \leftarrow \exp(-\tfrac{1}{\lambda}(\text{cost}^k - \beta)) \\
\eta \leftarrow \textstyle\sum_{k=1}^{K} \gamma^k \\
\omega^k \leftarrow \tfrac{1}{\eta}\gamma^k \\
u_t \mathrel{+}= \textstyle\sum_{k=1}^{K} \omega^k \epsilon_t^k
\end{gathered}$$

Where λ, Σ and Q are hyperparameters. λ controls how 'peaked' the weights are around the lowest cost trajectory. Σ is the covariance used to sample the control perturbations. Larger values mean you will sample larger perturbations. Q weighs the relative importance of each state towards the cost computation. We execute the first action and iterate through the above steps until we reach the goal position.

III. Results

Our aim is to implement the Autoencoder-based SINDy model on the planar pushing task using the Panda arm on the pybullet gym environment. as shown in fig.1. The dataset consists of a dictionary for states and actions for different trajectories for pushing the block using the Panda arm. The dataloader can load the state and action pair for multiple steps for a training step so we can predict states for future trajectory steps as well. We used AdamW optimizer with a learning rate of 1e-3 for total epochs of 3000. Momentum parameters of AdamW were fixed at (0.9, 0.999). Batch size was set to 500.

To obtain parsimonious dynamical models, we use a sequential thresholding procedure that promotes sparsity on the coefficients in the SINDy model, which represent the dynamics on the latent variables z [1]. Every 300 epochs, we set all coefficients in the SINDy model with a magnitude of less than 0.001 to 0, effectively removing these terms from the SINDy model. This is achieved by using a coefficient mask, consisting of 1s and 0s, that determines which terms remain in the SINDy model. Once a term has been thresholded out during training, it is permanently removed from the SINDy model. Therefore, the number of active terms in the SINDy model can only be decreased as training continues.

We experimented with different polynomial orders of the SINDy model, where the polynomial order of the model represents the maximum polynomial order of the latent state in the basis function library. Sinusoidal functions of the latent states z are added to the basis library $\Theta(z)$. The accuracy and other comparison metrics can be seen in Table 1. The training loss is minimum, but the test loss is high for polynomial order 3. Moreover, we found that using the MPPI controller sometimes fails and produces NAN values with the polynomial order 3 and 4 SINDy models, so we used the SINDy model with polynomial order 2. The figures 6 and 7 show the training and validation loss history over the training process for different SINDy models.

We used a single-step dynamics model because a multi-step dynamics model will require temporal signals to produce the correct non-linear dynamics model. Fig 8 shows the reconstructed image over single-step and multi-step. For a single step, the reconstructed image matches the original image, but for the multi-step reconstruction, the image is very blurry and is different from the original image which is due to the incorrect multi-step non-linear model as mentioned in the previous paragraph.

Fig 9 and 10 show the active SINDy coefficients for the model of polynomial order 1 and 2, respectively. The polynomial order 1 SINDy model has only nine active terms, whereas the polynomial order 2 SINDy model has 102 active terms. The rows represent the 16 latent states, and the columns represent the candidate functions like constant function, sin(z) and z, and $z_1 * z_2$ functions depending on the polynomial order. The sparsity percentage decreases with higher polynomial order.

We choose the linear Embed to Control (E2C) model as the

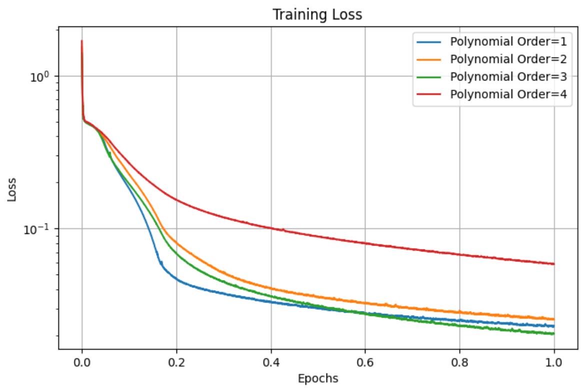

Fig. 6. Training Loss for different Polynomial Order SINDy model

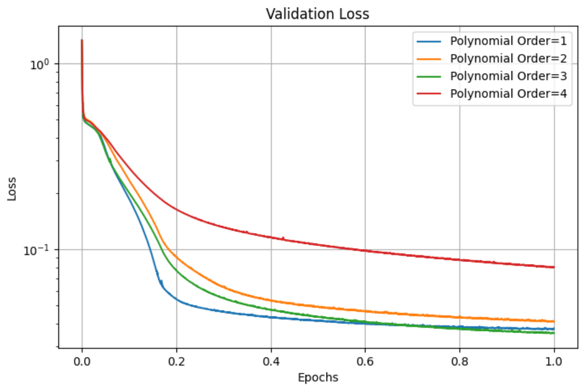

Fig. 7. Validation Loss for different Polynomial Order SINDy model

baseline model for comparison, and we can see the SINDy model has better test accuracy than the linear model but takes more steps to reach the goal position when run with a latent space MPPI controller. We ran experiments with a single-step dynamics SINDy model. We also compared the latent space MPPI controller versus the image space MPPI controller on our learned SINDy model. Image-space MPPI controller failed 3 out of 10 times with the SINDy model, but it also failed with the linear model with the same accuracy. So we decided to use a latent space MPPI controller which gives higher accuracy.

TABLE I
COMPARISON METRICS

Metrics	SINDy Polynomial Order				E2C Model
	1	*2*	*3*	*4*	
Training Loss	0.0232	0.0255	0.0206	0.0587	0.0286
Validation Loss	0.0376	0.0409	0.0355	0.0804	0.0574
Test Loss	0.0631	0.0597	0.0647	0.1018	0.2212
Steps to Goal	15	15	15	15	5
Sparsity Percent	98.3%	96.3%	96.0%	94.6%	NA

Figure 9 and 10 shows the active coefficients of SINDy, $\psi = [\zeta_1 \zeta_2 ... \zeta_n] \in R^{pxn}$ that determine the active terms from

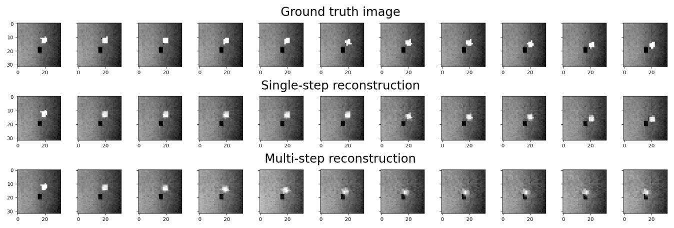

Fig. 8. Reconstructed Image for single step and multi-step for SINDy model with polynomial order 2

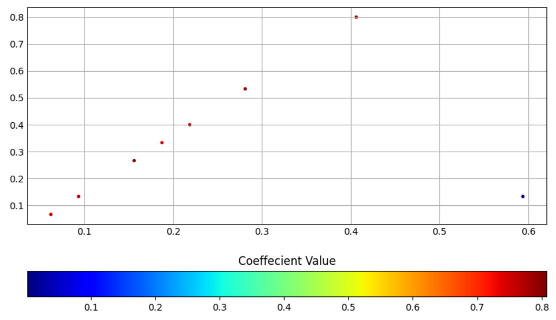

Fig. 9. Active Coefficients for SINDy model with polynomial order 1

the basis library in the dynamics. Coefficients that are active generally for linear and sinusoidal basis functions.

IV. Conclusion

We designed a SINDy model combined with a state autoencoder to learn the non-linear dynamics of the planar pushing task for the panda arm. The learned model is parsimonious, making it a generalizable and interpretable model. Experiments show that the SINDy model has better accuracy than the linear E2C model but is slower than the linear model when combined with the MPPI controller. Future work includes making the SINDy model faster.

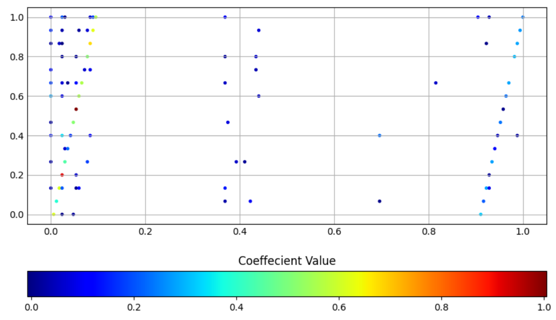

Fig. 10. Active Coefficients for SINDy model with polynomial order 2

References

[1] Kathleen Champion, Bethany Lusch, J. Nathan Kutz1, Steven L. Brunton, ”Data-driven discovery of coordinates and governing equations,” Proc. Natl. Acad. Sci. U.S.A. 116, 22445–22451 (2019).

[2] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz, ”Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” PNAS, vol. 113 (15), pp. 3932–3937, 2016.

[3] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. Science Advances, 3(e1602614), 2017.

[4] N. M. Mangan, T. Askham, S. L. Brunton, J. N. Kutz and J. L. Proctor, ”Model selection for hybrid dynamical systems via sparse regression,” Proc. R. Soc. A 475:20180534, 2019.

[5] K.P. Champati, B. Engquist. (2020). SindyAutoencoders [GitHub repository]. Retrieved from `https://github.com/kpchamp/SindyAutoencoders`

[6] G. Williams et al., ”Information theoretic MPC for model-based reinforcement learning,” 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 1714-1721.