Using NeuralODE to Learn Panda Arm Dynamics

Sarvesh Mayilvahanan, Jack Fenton
{smayil, fentonj}@umich.edu

***Abstract*—Learning dynamics in the context of robotics is a difficult problem, especially for complex systems, due to the number of components involved in a system. It may not be possible to model these systems using physics-based models, and thus model-based methods may be less practical than learning-based methods. Typical neural networks can learn the dynamics of a system using collected data and have shown to be effective in doing so for somewhat complex systems. However, in the case of a highly complex system, even these methods may prove ineffective. NeuralODEs are a neural network architecture that use ordinary differential equations to represent the dynamics. This paper includes an in-depth study of the effects of the (i) number of prediction steps, (ii) integration algorithm, and (iii) model expressivity on the NeuralODE's performance and how NeuralODE compares to more traditional neural network architectures for a Panda robot arm.**

Our code is publicly available at https://github.com/sarveshmayil/neuralODE-panda

I. Introduction

Robotics has been one of the most rapidly growing fields, with robots being developed to perform a wide range of tasks, from assembling complex machinery to assisting in surgical procedures. A crucial aspect of robotics is the ability to model and predict the dynamics of the robot, which involves understanding how the robot responds to different inputs and disturbances. Accurate modeling of robot dynamics is essential for tasks such as control, planning, and optimization.

Learning the dynamics of a robot is a challenging task, as it involves modeling the complex interactions between its different components, such as actuators, sensors, and controllers. It is often impractical to sufficiently model these complexities explicitly with mathematical and physics-based approaches, so purely model-based methods are seldom used. Conversely, purely data driven approaches, frequently called an absolute or "know nothing" model, have their own shortcomings. While learning a state representation directly from observed data, these approaches fail to incorporate domain knowledge like the relation between current and subsequent states. A simple method that bridges the gap between model-based and learning-based approaches is to learn residual dynamics, where the network produces the next state by learning an update term to be added to the current state.

Recent years have seen further development towards incorporating domain knowledge of dynamical systems into learning-based approaches. For example, [1] used long-short-term memory (LSTM) [2] networks to learn the inverse dynamics model of a highly complex KUKA arm, outperforming the previously state-of-the-art Gaussian Processes (GPs).

One recent development in this area is the use of NeuralODE [3], a neural network architecture that uses ordinary differential equations (ODEs) to represent the dynamics of a system. By parameterizing the derivative of the hidden state using a neural network, NeuralODE has been shown to be effective in learning the dynamics of complex systems. The output of the network is computed using any one of their implemented blackbox differential equation solvers. Backpropagation through ODE solvers has proven difficult and memory intensive in the past, but NeuralODE's framework uses an *adjoint sensitivity method* [4] to compute gradients, applicable to all ODE solvers and providing low memory cost and linearly scaling with the problem size.

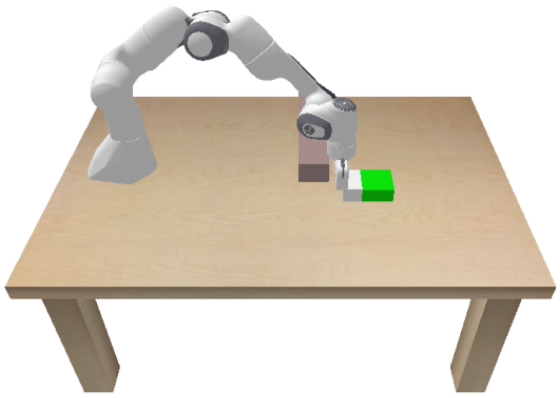

Fig. 1. Panda arm successfully pushing block to goal state.

In this paper, we explore the use of NeuralODE to represent the dynamics of a Panda robot arm simulated in the gym environment used in homework assignments. We compare the performance of the absolute and residual dynamics models previously used in this course with absolute NeuralODE and residual NeuralODE models, applied to learning the dynamics of the panda robot arm and demonstrate the effectiveness of NeuralODE in accurately predicting the behavior of the robot. Additionally, we perform an ablation study of different design decisions for NeuralODE and compare how those changes effect the models' performances. Parameters varied include (i) number of predictions steps, (ii) ODE solver, and (iii) model

size. Our results show that NeuralODE, though not best suited to external input environments as discussed later, can provide a powerful framework for modeling the dynamics of complex robotic systems and can lead to improved control, planning, and optimization of robot behavior.

II. Methods

The process for learning state dynamics with neural networks is similar across different ML approaches. Training data consists of large quantities of trajectories, representing the explorable state space of the robot by applying random actions to from initial states. Training then consists of learning the state that results from any particular state when an action is taken. This can be evaluated using a single-step or multi-step loss, representing whether learning considers a single state at a time or tries to produce predictions consistent with multiple steps in the trajectory. Once this dynamics model is trained, it can be used by planning and prediction models like MPPI [5] to find a path towards a goal. These dynamics models are responsible for producing good estimates of what state will result from each possible action, which allows the controller to more accurately and reliably produce paths towards a goal.

In this course we previously implemented both absolute dynamics and residual dynamics models inside Open AI's Gym environment for the Panda robot arm. The absolute dynamics model takes inputs consisting of the current state and action being taken concatenated together and predicted a resulting state. The Panda environment has a 3 dimensional state and 3 dimensional action space, resulting in a network architecture of 6 dimensional input, 2 hidden layers of size 100, and an output of dimension 3. The residual dynamics model consists of the same network architecture but produces a result with different meaning. The residual model's output serves as an update term that gets added to the current state to produce a next state.

NeuralODE takes the idea of residual updates to its limit, where step sizes approaching 0 leads to differentiable dynamics. By parameterizing the continuous dynamics of hidden units using an ODE, NeuralODE is able to learn more accurate dynamics. The resulting continuous function produced by NeuralODE has the potential to greatly increase the effectiveness of its prediction, achieving results consistent with much larger architectures.

NeuralODE has been implemented as a complete Python package called *torchdiffeq* [6], built in Pytorch with their adjoint method used for backpropagation. One main interface is provided as a container for general-purpose algorithms for solving initial value problems. ODE solvers like *dopri5* and *rk4*, different versions of Runge-Kutta, can be selected along with a set of times that much be explicitly stepped to, forcing the solver to learn particularly complex functions faster. One limiting factor of the NeuralODE framework is that it requires inputs and outputs to be of the same dimension, which can prove troublesome when attempting to predict a state using both a state and an action.

Our implementation of NeuralODE replaces the dynamics models used previously. Like those previous models, the neural network used consists of an MLP with 2 hidden layers of size 100, though in our ablation study we show the effects of changing the size of the hidden dimension. As stated previously, the framework requires that inputs and outputs be the same shape, so both state and action are concatenated for input and the output is sliced to produce the resultant state. In each step of a dynamics roll out a single state and action are input into the model to produce the next state. That resultant state is then returned, whereupon a new action is applied to it. This is the important thing to note about our application compared to those mentioned in their paper and documentation; a continuous integration to predict the input's evolution over time cannot be made when there is an external action being applied. This action is unable to be modeled because actions in the dataset being learned on are taken at random. Instead of integrating for multiple time steps into the future, we set a parameter for number of steps to break the single prediction into; though only the value of the last step gets returned, these intermediate steps enforce further constraints on the solver that increase the accuracy of the learned ODE. This limits the power of NeuralODE, as without observations of those intermediate states their contribution to the loss cannot be accurately accounted for, but in practice it still proves to improve the performance over other methods. Two different version of the NeuralODE model were implemented: the first, called Absolute NeuralODE, solves for the full next state and the second, called Residual NeuralODE, solves for a residual to update the current state.

III. Results

In order to evaluate the effect of the hyperparameters (i) number of prediction steps, (ii) integration algorithm, and (iii) model expressivity, we perform a grid search on these parameters using collected data of the Panda arm. We chose to experiment with a range of prediction steps from 2 to 10, where 2 is the minimum required number of steps. We also chose two integration algorithms: *dopri5* and *rk4*, which are adaptive and fixed solvers respectively, in order to evaluate the effects of that difference. For model expressivity, we maintained the same number of hidden layers as the previously trained standard NNs, but varied the hidden dimension size on a range from 4 to 100. This was done in order to evaluate how NeuralODE would perform with a very lightweight network and with a very expressive network. All models were trained for 100 epochs at an initial learning rate of 1E-3. The ODE integrators were given absolute and relative tolerances of 1E-8.

Figure 2 shows the results of this grid search for an Absolute NeuralODE. We see fairly similar performance between the two integration algorithms, having a similar loss in the best and worst performing models. There is also a clear trend observed between the number of prediction steps and hidden dimension size. As the number of prediction steps increases, the loss decreases and as the hidden dimension size increase, the loss also decreases. This is intuitive as an increased

number of prediction steps corresponds to a larger number of smaller integration steps, which allows for more accuracy in state predictions. Additionally, the increased dimension size allows for more information to be learned, as there are more parameters in the network.

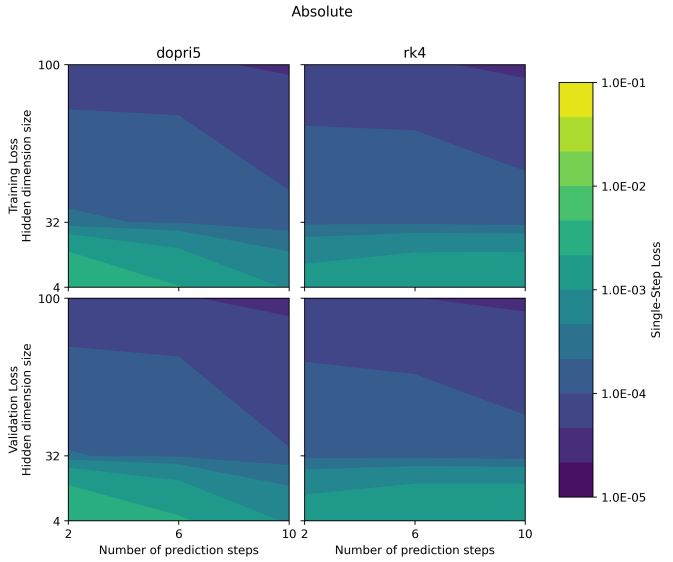

Fig. 2. Grid search of absolute NeuralODEs to learn dynamics of Panda arm using a single-step loss.

Figure 3 shows the results of the grid search for a Residual NeuralODE. We observe a fairly similar trend to that seen for the absolute models. At low number of steps and low hidden dimension, the model does not appear to learn very well as opposed to high number of steps and high hidden dimension. This trend is exacerbated at low hidden dimension, where there are a very few number of parameters. However, as the model expressivity increases, the larger number of parameters is able to account for the low number of prediction steps to provide good performance.

We also study the effects of these hyperparameters when training the dynamics models using a multi-step loss, which helps in preventing drift in model roll outs. Figure 4 shows the results of this ablation study where we once again observe the same trends as those in Figures 2 and 3. We see that a smaller model with a large number of prediction steps could match the performance of a large model with fewer number of prediction steps.

In order to evaluate the effectiveness of NeuralODE and its performance in state prediction, we compare the validation loss of two standard NN architectures (absolute and residual) to two NeuralODE models. All four models that are being compared have the same NN architecture and same number of trainable parameters. Figure 5 shows the results of this comparison, which illustrates that the NeuralODE models overall match out outperform the standard NNs. In particular, we see that the Absolute NeuralODE achieves 2x better loss than the standard Absolute model. However, the Residual NeuralODE only achieves comparable performance to the

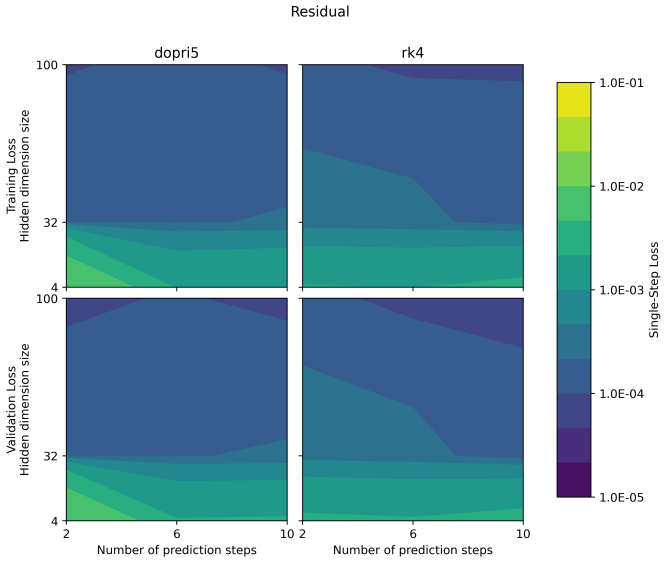

Fig. 3. Grid search of residual NeuralODEs to learn dynamics of Panda arm using a single-step loss.

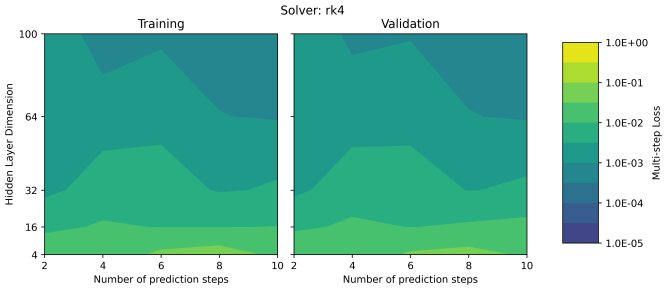

Fig. 4. Grid search of absolute NeuralODEs using fixed *rk4* solver to learn dynamics of Panda arm using a multi-step loss.

standard Residual model. This is due to the key idea that NeuralODE is built upon. NeuralODEs are formulated by definition to act as a residual model, with the starting state being the current state, which is evolved over the prediction steps to the next state. Therefore, when a Residual NeuralODE is used, the model must start out with the current state and evolve it to become the residual instead of the next state. This simply makes it harder to learn than the Absolute NeuralODE, which already encodes the model's residuals.

In order to validate the NeuralODE model performance on a control task, we use MPPI to push a block towards an goal state with an obstacle in the way. The cost function used for MPPI is defined as follows

$$\text{Cost}(\mathbf{x}_1, ..., \mathbf{x}_T) = \sum_{t=1}^{T} ((\mathbf{x}_t - \mathbf{x}_{goal})^T Q (\mathbf{x}_t - \mathbf{x}_{goal}) + 100 \texttt{in\_collision}(\mathbf{x}_t))$$

where $\mathbf{x}$ is the state, Q is a weighting matrix, and $\texttt{in\_collision}(\mathbf{x}_t)$ is a function that determines whether the block is in collision with an object.

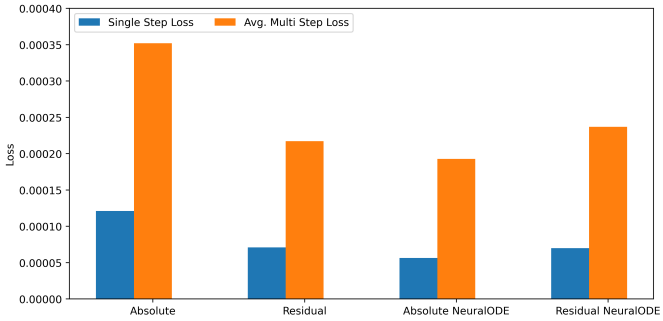

Fig. 5. Comparison of best performing model of each model type for single-step and multi-step loss averaged by trajectory length.

We observe that the Panda arm is able to successfully push the block to the goal state consistently as shown in Figure 1, indicating that the NeuralODE was able to accurately learn the dynamics of the system to an extent at which MPPI was able to very consistently move the block around the obstacle.

From our testing, we found that the arm was able to push the block to the goal state 95+% of trials using the Absolute NeuralODE model.

IV. Conclusion

Through our testing, we found that NeuralODEs were at least as accurate as any standard neural network with an equal number of parameters. This can be attributed to the incremental nature of NeuralODEs, which integrates the ODE, represented by a neural network, to reach the next state.

The ablation study that was conducted for different number of prediction steps, ODE integration solvers, and model expressivity in the form of hidden dimension size concluded that an increased number of prediction steps and an increased hidden dimension size aided in the accuracy of state predictions. This is intuitive as more prediction steps means the model is able to take smaller, more accurate steps during ODE integration. Additionally, a large hidden dimension size equates to more trainable parameters, making it easier for the model to (express the complexity of the system dynamics) learn the dynamics. One worthwhile finding is that in most cases a smaller network with more prediction steps could achieve results as good as larger networks that used fewer steps, which could point towards a way of limiting required memory for model parameters by using more prediction steps. We also find that the adaptive and fixed solvers tested perform similarly in terms of accuracy when trained for a sufficient number of epochs. However, during training, we found that the adaptive solver *dopri5* was significantly slower (about 10x) than the fixed solver *rk4*. Therefore, due to the similar accuracy performance, we would suggest using the fixed solver for general purposes.

Generally, we found that the package *torchdiffeq* was not very well optimized and overall performance was fairly slow. Additionally, using a GPU to train as opposed to a CPU did not show any speed up which one would expect.

When compared to standard neural networks used to learn dynamics, NeuralODEs have a clear advantage in accuracy, performing at least as well as a similar size neural network. However, there are clear drawbacks in efficiency and training/evaluation time which need to be considered before employing a NeuralODE. In the case of a highly complex dynamical system that a simple neural network is not able to learn, NeuralODE could be applied to provide better performance.

References

[1] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, "Learning inverse dynamics models in o (n) time with lstm networks," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 811–816.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[4] L. S. Pontryagin, *Mathematical theory of optimal processes*. CRC press, 1987.

[5] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[6] R. T. Q. Chen, "torchdiffeq," 2018. [Online]. Available: https://github.com/rtqichen/torchdiffeq