Neural ODE-based Model Predictive Control

Luke Roberto
Khoury College of Computer Sciences
lroberto@ccs.neu.edu

Anupam Kumar
Department of Mathematics
kumar.anupa@husky.neu.edu

1 Introduction

In the real world, humans learn many complex tasks and general skills from a stream of high-dimensional sensory inputs with little external guidance. This is often without specific external rewards and without knowing the exact physics of the skills and objects that they interact. Providing robots the ability to learn complex and generalizable skills directly from raw visual and sensory inputs with little external intervention is highly valuable. Training robots for each task separately with current state-of-the-art approaches is very computationally expensive, and we want them to function in unstructured and diverse environments.

Modern approaches to the robotic planning problem tend to use neural networks to learn models of the environment, known as model-based approaches. For example, the authors of the paper **?** have proposed a self-supervised model-based approach they called Visual Model Predictive Control (VMPC) which is shown to learn a predictive model for a wide range of tasks such as picking, placing, arranging, and folding deformable objects. This predictive model is in the form of next-frame image predictions for a sequence of actions. Their method consists of three phases: unsupervised data collection (autonomously collecting data by applying random actions sampled from a pre-determined distribution), model training (video prediction taking input as images at different time-steps and action-sequence), and test time control (finding action-sequence that minimizes the cost function). The major contribution the of VMPC method is that it can generalize to never-before-seen-objects–both rigid and deformable–and can solve new user-defined object interaction tasks using the same model.

In order to reduce the scope of the project, we wanted to hone in on a particular aspect of neural network-based planning papers such as the one described above. Many of these authors use RNN-based algorithms to train their models. We believe this is a critical drawback. The models that are trying to be learned are inherently continuous, but the RNN discretizes time into fixed-size bins. An improvement that can be made to these model learning algorithms could be an inductive bias added to the model architecture that allows it to directly model the continuous nature of the environment. A candidate method for this type of algorithmic prior is a Neural Ordinary Differential Equation (NODE) **?**. In this work we trained a neural network-based environment model based off of NODE, and then tested its performance on a standard planning algorithm called model predictive control (MPC).

The main contributions of of our work are as follows:

- novel algorithm, using a combination of NODE-based model with MPC
- novel insight into inductive bias on the model architecture to be used for robotic control

2 Methodology

We wanted to develop a similar methodology to **?**, where authors first trained a dynamics model and then used that learned model for control. Fig. **??** shows a condensed version of their approach.

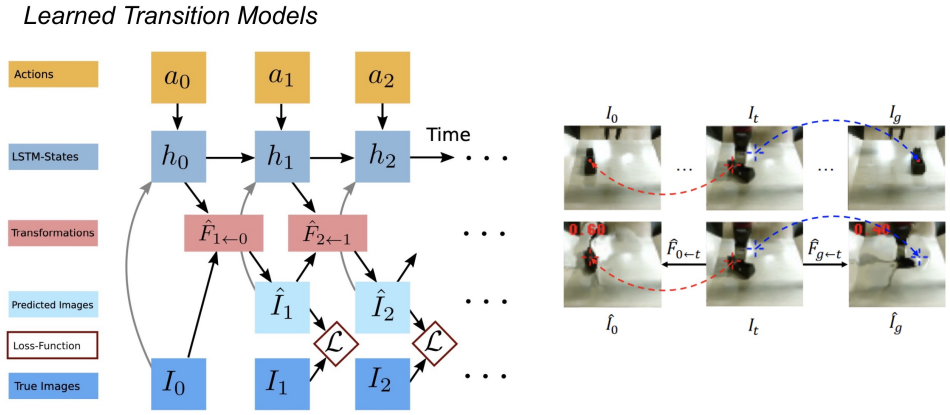


Figure 1: Left: the network architecture used to perform image prediction. The bottom left shows I_0, the input image, and then gradually transforms this image with a sequence of actions. Each of the intermediate images are evaluated with a loss, an used to train the network. Right: Control task, the network uses its prediction model to evaluate various sequences of takes to to see if an action sequence will bring it close to the goal image I_g, starting from I_0. Figure reference: **?**.

The authors use a flow based model that gradually transforms an input image over each time step based on the current action. We propose using a neural network to directly learn the continuous dynamical mode. We believe that the model learned by the neural ODE will be much more expressive and robust **?**, while also encoding an inductive prior that our system is a continuous dynamical system - rather than discrete. In order to build this system, we had to train the prediction models and then test with MPC.

2.1 Model Learning

2.1.1 RNNs

As mentioned earlier, RNNs will be used as the baseline. The model will take a sequence of inputs and unroll a sequence of outputs, while having some internal "memory" about the current state of the network. In our case, we are using gated recurrent units (GRUs), as seen in figure **??**, which help to mitigate the vanishing gradient problem that arises in long time horizon tasks.

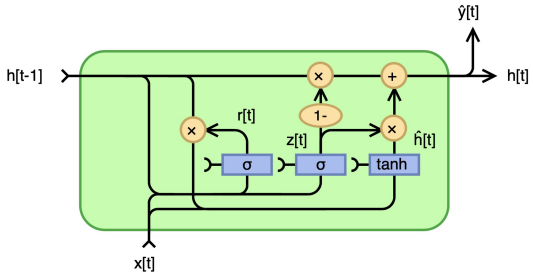


Figure 2: GRUs architecture

For the purposes of robotic control, we have state-action (s_t, a_t) pairs as input and the next state s_{t+1} as outputs in our training setup. Fig. **??** shows how a trained RNN will be used as a transition model.

The idea is to predict the intermediate and final output state based on a state and a sequence of actions as input.

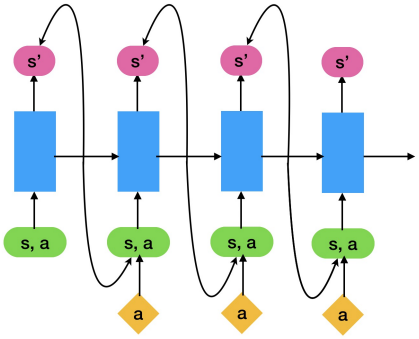
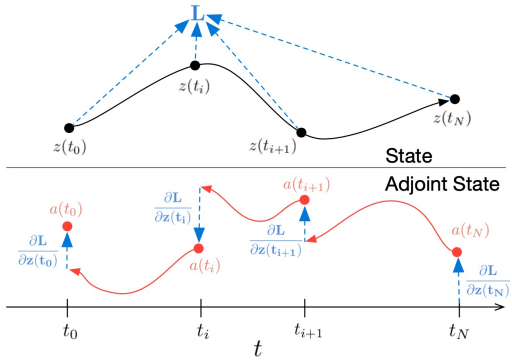


Figure 3: RNN Transition model: The blue blocks represent the hidden state of the RNN unrolled through time moving left. The inputs are the state and action vector that the network combines with its hidden state to predict the next state in the series. If we are given multiple actions (shown in yellow), the RNN can be cascaded, using the output as the input for the next state, in order to predict many steps into the future.

2.1.2 Neural ODEs

In the VMPC case, the dynamical model is the dynamics of the pixels in image space. We can think of dynamics as defining a differential equation, $\dot{x} = f(x, t)$, for the evolution of our state x given some initial conditions. In the neural ODE paper, they mention how Residual networks are like an Euler forward scheme over an ODE. If you fix the blocks to be the same, you can represent a ResNet as repeated applications of this block, $x_{t+1} = x_t + f(x_t)$. This is exactly an Euler forward scheme with time step $\Delta t = 1$. Fig. **??** shows a simple 1D example of the effects of discretization.

Similarly, in the VMPC paper, they use an RNN to learn the dynamics. We can think of a simple RNN as iteratively applying the function to the current hidden state, $x_{t+1} = f(x_t)$ **?**. This is just like the ResNet application, but we have the ability to unroll the RNN forward in time as far as we would like. This not a great model for the system dynamics because the time step is fixed and we are also throwing away much of the knowledge we have about solving differential equations away.



(a) Euler method is used to calculate derivatives for back-propagation

(b) The adjoint sensitivity method solves an augmented ODE backwards in time. Figure reference: **?**.

Figure 4: Discretization error is a very well-studied phenomenon in differential equations research, using a tool like NODEs allows us to use numerical solvers to learn models that fit the given data points quite well.

In our implementation we essentially use the Neural ODE to perform an extrapolation from the current state to a state into the future given a sequence of actions (we call it a controller). Figure **??** visually depicts this process.

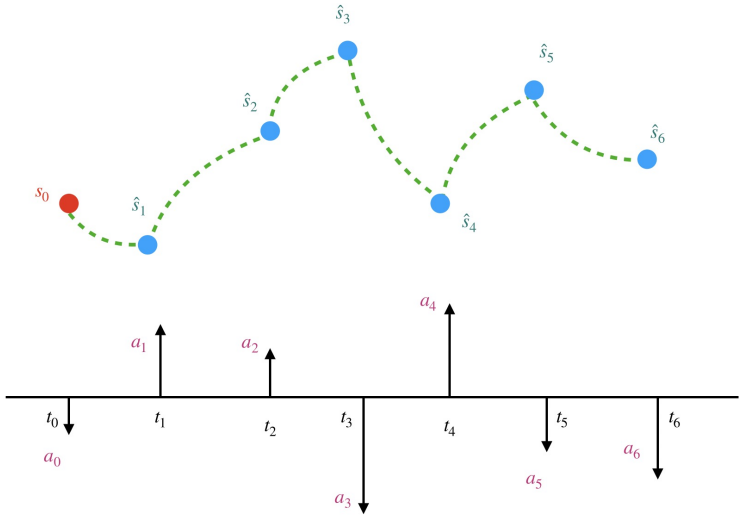


Figure 5: Representation of our transition model with a Neural ODE. This is a 1 dimensional state space, with state on the vertical axis and time on the horizontal. The initial state s_0 and a list of actions $[a_0, ..., a_n]$ are integrated over time in order to generate the intermediate state predictions.

2.2 Model Predictive Control

A simplified version of the VMPC algorithm from **?** was implemented for our system. It differs in the action sampling step, as the environments we chose to test on all had discrete action spaces. The pseudo-code is provided for our implementation below:

Algorithm 1: Model Predictive Control

Input : Predictive model g, planning cost function c
output : a_t, predicted best action
for $t = 0... \ T-1$ **do**
  for $i = 0... \ n_{iter} - 1$ **do**
    if $i == 0$ **then**
      Sample M actions over the horizon $a^{(m)}_{t:t+H-1}$ from a uniform distribution over our discrete action space ;
    else
      Sample M actions over horizon from updated distribution over action space. ;
    end
    Check if actions are valid, otherwise resample. ;
    Use g to predict future state sequence $\hat{s}^{(m)}_{t:t+H-1}$. ;
    Evaluate each action sequence with cost function, c. ;
    Update action distribution with lowest cost action distribution. ;
  end
  return a^*_t, first action from best sequence;
end

3 Experiments

All code for the following experiments can be found at this github link.

3.1 Experimental Setup

We used the Acrobot environment for our experiments. It's a double pendulum dynamical system. We chose it because it is a simple environment that can be represented by 4 states: $[\theta_0, \theta_1, \dot{\theta}_0, \dot{\theta}_1]$, but the dynamics are relatively complex since it is a chaotic system. The action is either applying +1, 0 or -1 torque on the joint between the two pendulum links. Fig. **??** graphically depicts the acrobot environment.

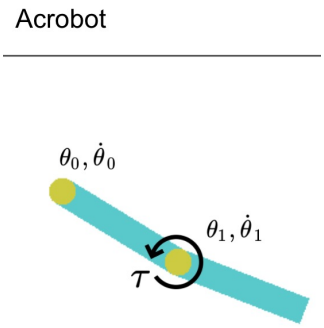


Figure 6

3.2 Experimental Results

We randomly generated 50k data samples with features [current-state, action, next-state] for training purposes. There are 3 parts of the project that we covered: model learning, model predicting horizon, and MPC swing-up experiment.

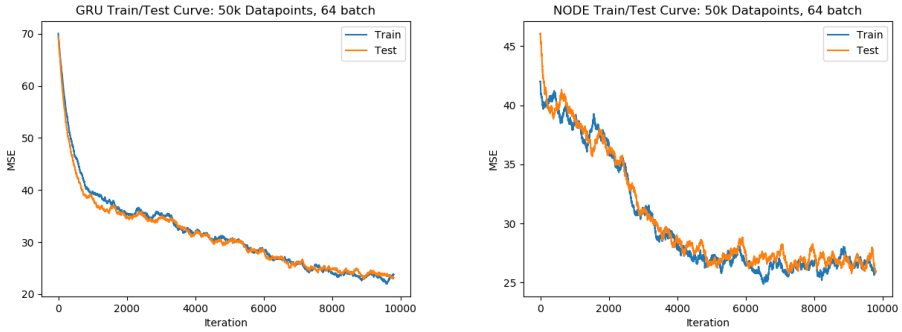
3.2.1 Model learning

We want the model to learn to predict next-state given the current state and a sequence of actions. We used the Mean Squared Error (MSE) cost between states as criterion to train the network. The units for this cost are not well defined as state is defined by two different values θ (radians) and $\dot{\theta}$ (radians/s). We are basically trying to reduce the L_2 distance between the predicted state and the true 4-dimension state vector. The learning plot for GRU and Neural-ODE is show in Fig. **??**. The learning is over 50k data points, 80% of the data reserved for training and remaining for test, 10k iterations of stochastic batch gradient descent, and a batch size of 64 samples.

3.2.2 Prediction over 100 steps horizon

The MSE loss observed in Fig. **??** quantitatively shows quite poor loss, even near the end of training. The models have become better at predicting the next state, but what we really want is to know how good the model is at predicting multiple steps into the future. Fig. **??** shows the qualitative results of the future state prediction for both models. Both GRU and Neural ODE are performing poorly for predicting even over next 5-10 steps horizon. We spent a lot of time working on improving the performance of these models, including implementing a simple 2 layer MLP as a test for correctness. Our hypothesis is that one reason for poor prediction over horizon is the dynamics are chaotic, so small perturbations in the input make the horizon quite different. This large variance in the next state could make it quite hard for a network to learn these dynamics well enough to track a single trajectory.

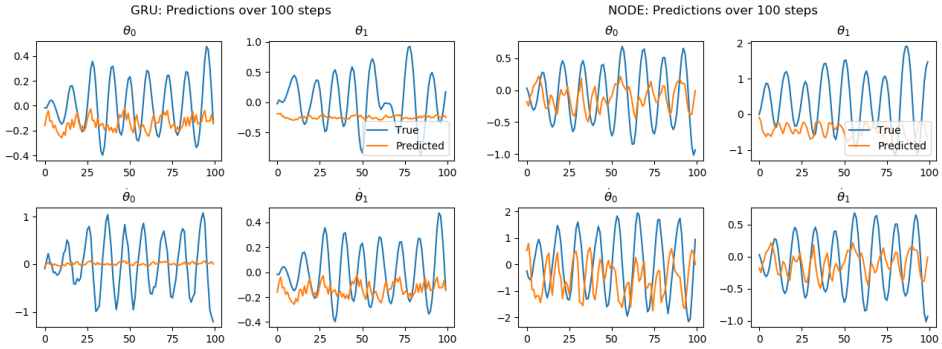
Hence, our result is that predicting far into the future is quite difficult, even for this environment. Still, Neural ODE has qualitatively and quantitatively better performance than GRU.



(a) GRU model training

(b) Neural ODE model training

Figure 7



(a) GRU prediction

(b) Neural ODE prediction

Figure 8

3.2.3 MPC Swing-up Experiments

The goal of the MPC swing-up experiment is to roll out some horizon steps to calculate the cost in terms of swinging the double-pendulum past a horizontal line, and choose next action based on smallest loss for corresponding roll-out. Since our model performed poorly to predict states over horizon, our model did not learn to swing the double pendulum up, although we can see that it is doing better than if random actions would have been selected. Here is the mpc_gif link of the resulting policy that we want our model to learn.

4 Conclusion and Discussion

For this final project, we wanted to validate whether encoding a continuous inductive bias would improve model learning for robotic planning. We built a framework to test on a variety of standard Reinforcement Learning and Robotic Planning benchmarks, while also benchmarking our model against other models in the literature that solved similar problems. We learned a lot from the work we did the past 3 months. Learning dynamical models is not an easy task (as can be seen in the literature as well). Choosing the Acrobot environment proved to be difficult due to its chaotic nature and it being under-actuated. It surprised us that the GRU and evan a vanilla MLP-based transition model performed so poorly in our testing, as we thought they would just overfit if we gave them enough training data. It was encouraging to see the NODE agent performing at least qualitatively well, although not well enough to have reasonable performance on the MPC-based controller.

While we did not find conclusive evidence that our approach achieved the results that we are looking for, we do believe it opens up an avenue for interesting future work that we would like to pursue outside of the course. One particular next step we would like to take this project is to use non-Markov

environments. The acrobot has perfect information given the state of the pendulum, and so we would like to test on environments that only give partial information about the environment (similar to the towel folding examples in **?**. This will make our RNN benchmark more realistic in use case, as acrobot does not need a hidden state in order to predict the next environment step. The other step we are quite excited to try is using the NODE agent in an autoencoder-based setup. As discussed in the introduction, we want to be able to learn environment models from raw pixel data. That is computationally infeasible with the current system we have built, so we need some sort of encoder structure that can take the pixels to a small latent space. This is similar to the world models paper **?**, except we actually integrate a continuous dynamics model in the latent space. This work is quite exciting to us and we hope to continue working on interesting problems related to learning dynamical models.