Optimization of Model Predictive Control Using Neural Network and iLQR

Chinmaya Kausik, Dao Nguyen, Congyan Song, Elham Islam, Pranjal Sharma, Yurong Chen
University of Michigan

Introduction

Finding the best control for a dynamical system is a critical problem in many fields and its solutions are often bottlenecked by computational requirements.

In particular, MPC is a very popular method for real-life process control in chemical, aerospace and biomedical applications.

Deep learning algorithms have shown effectiveness in the approximation of high-dimensional systems. It has been effective in reinforcement learning, which frequently overlaps with optimal control theory in the type and complexity of problems being approached.

Motivation

Solving high dimensional closed loop problems, as discussed in [**3**], is a difficult task and has been a recurring hurdle for many decades.

The success of machine learning (ML) in novel applications such as imagine recognition, natural language processing and decision making has raised new hope. Ultimately, these ML algorithms succeed by addressing the underlying objectives of effectively approximating functions and making inferences between inputs, states and outputs.

Further, these methods can be applied to higher dimensional problems with more ease than the typical numerical and computational methods that have existed for much longer. It only seems natural then to turn to ML, particularly deep learning methods to improve the efficiency of solving closed loop problems.

What is MPC?

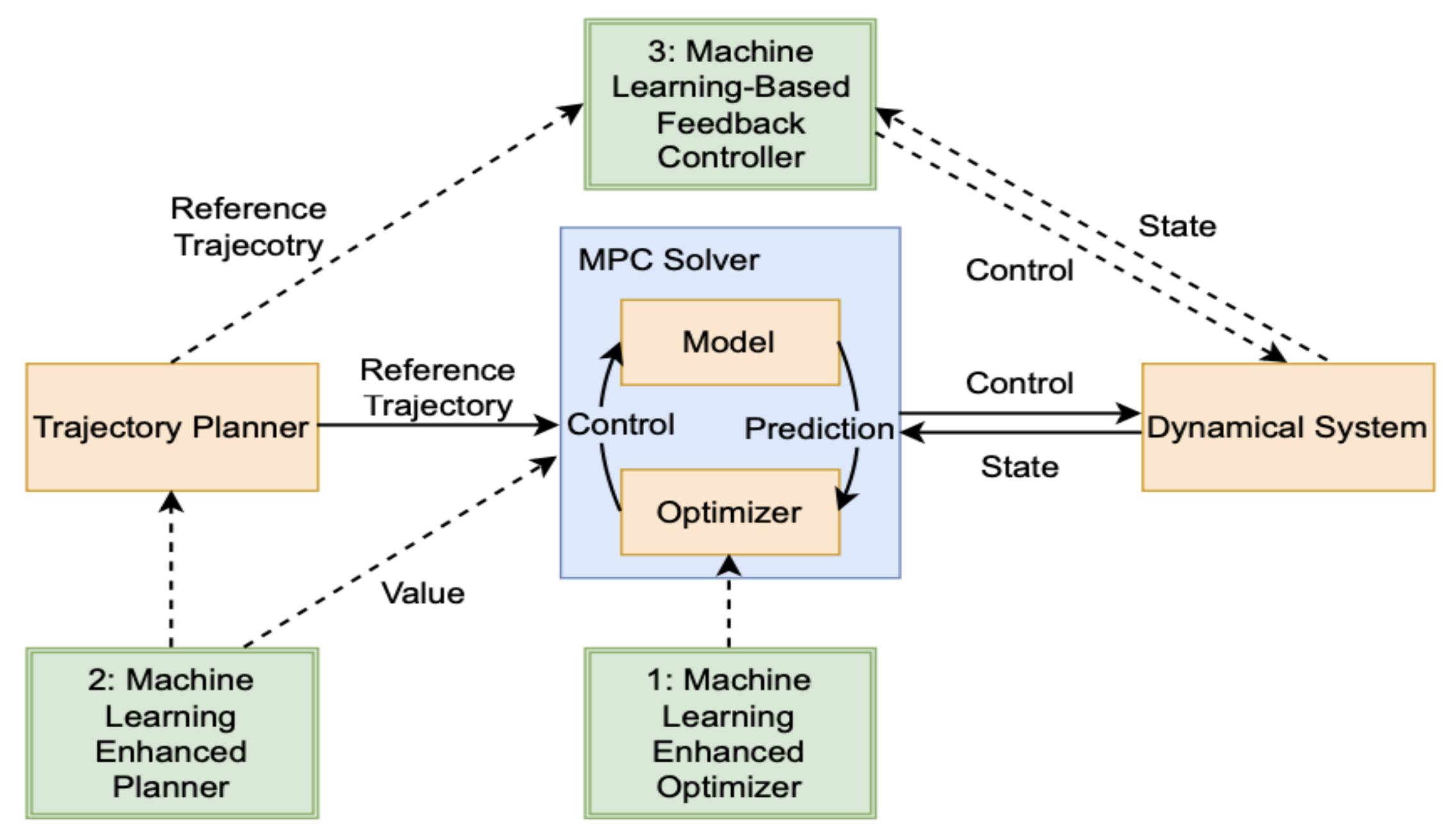
In MPC, we are interested in the following optimal control problem:

$$\min_{\{x_k,u_k\}_{k=0}^{T}} \Sigma_{k=0}^{T-1} L_k(x_k,u_k) + M(x_T)$$

This is called the global optimal control problem. Here, we usually deal with a large time horizon T. The idea of MPC is to split this T into many small time horizons p. Then we only need to find the optimal control for the system inside each such p and approximate the cost for the rest of the time.

Even after dividing the global optimal control problem into smaller problems, MPC can be computationally costly. As a result, we are trying to minimize the costs.

As described by the figure below from [**1**], there are 3 main ways of enhancing MPC using machine learning. We are focusing on enhancing MPC using a combination of methods 3 and 2. We implement these methods using a Neural Network and iLQR.



System Formulation

We attempted to apply to our methods to the quintessential inverted pendulum system. We define the state and control vectors x and u as so:

$$\mathbf{x} = [\theta \ \dot{\theta}]$$

$$\mathbf{u} = [\tau]$$

such that $\tau \in [-1,1]$ and $\theta \in [0, 2\pi]$.

Essentially, the state of the system comprises of the angle of the the pendulum about a pivot at the base and its time derivative. We define our control input as the torque applied at the base of the pendulum (i.e. a motor response).

The goal is to keep the pendulum the upright. Perfectly balanced suggests $\dot{\theta} = 0$ and we define our coordinate system for upright to be $\theta = 0$

$$\mathbf{x}_{goal} = [0 \ 0]$$

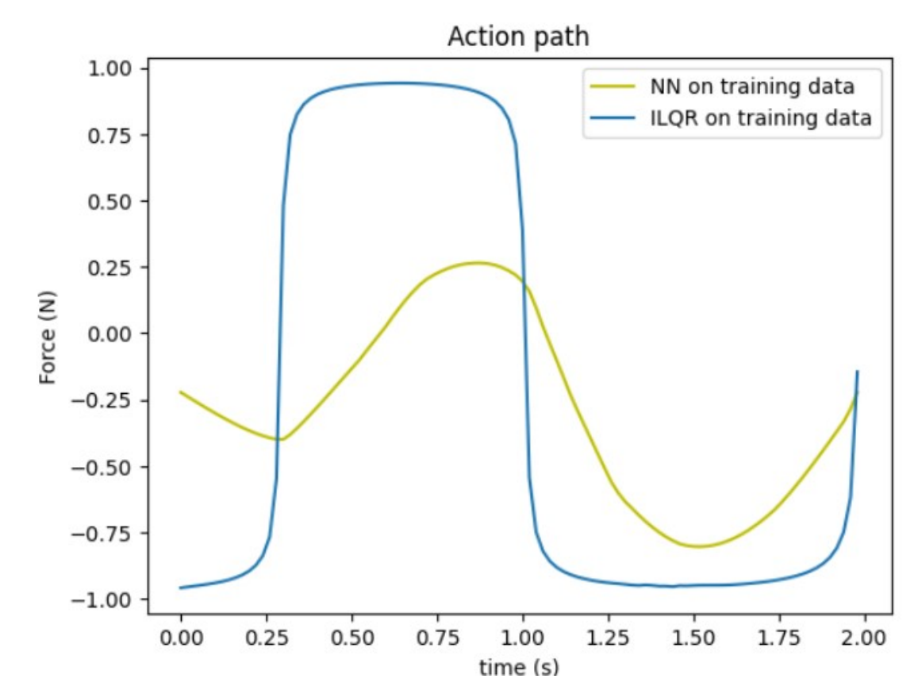
Developing and Implementing Neural Network: Outlined Approach

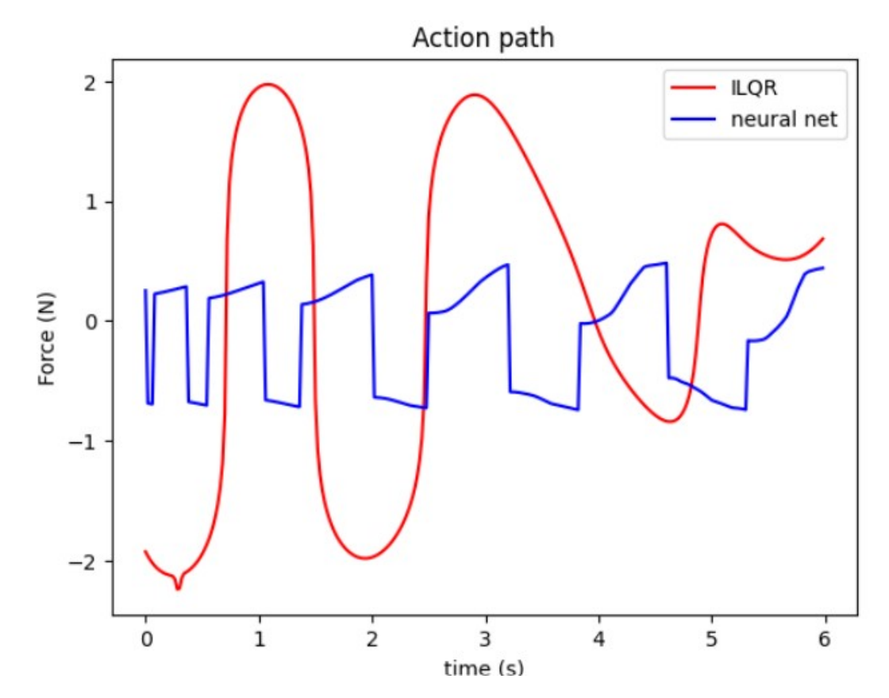
1. <u>**Generate training data:**</u>
   (a) **Discretize state space:** To get data that is representative of the state space but also computationally tractable, we create a mesh for the intervals $\theta \in [0, 2\pi]$ and $\dot{\theta} \in [-2\pi, 2\pi]$ parameterized over 20 values in each interval.
   (b) **Generate optimal controls using the iLQR Algorithm:** Using our mesh developed from (a), we run each state as an initial condition to the iLQR algorithm. From each run of the algorithm, we will obtain a list of states $\mathbf{x}_{\theta,\dot{\theta}}$ and corresponding optimal controls $\mathbf{u}_{\theta,\dot{\theta}}$ that we will then merge to form a larger data set.
   (c) **Compile results from iLQR:** From performing (a), (b) over a length of N steps of size $t_{step} = 0.02$, we developed a set of training data consisting of 3000 sets of states with respective optimal control inputs.
2. <u>**Construct and Train Neural Network:**</u>
   (a) **Neural net specifications:** Using **PyTorch**, we constructed a Neural Network with 1 input layer, 2 hidden layers, and an output layer. The activation function for the hidden variables were rectified linear units (ReLUs) as they are generally the recommended function of choice, as used in [**4**]. We used a batch size of 100 and learning rate of 0.01.
   (b) **Train Neural Network:** We trained this neural network without any GPU acceleration. Passing an input $\mathbf{x}_{\theta,\dot{\theta}}$ the neural network generates an output control $\mathbf{u}_{mm}$. We defined our loss as the mean squared error between $\mathbf{u}_{mm}$ and predicted output control $\mathbf{u}_{\theta,\dot{\theta}}$ we generated from the ILQR. We iterated through 1000 epochs, (i.e. we passed the entire training set forward and backward through the NN 1000 times).
3. <u>**Benchmark Neural Network**</u>: We evaluated performance through 2 methods:
   (a) Compare iLQR and NN control outputs when given the same initial conditions.
   (b) **Initialize iLQR with NN:** so far we have initialized ILQR through randomly sampling from a uniform distributed interval $[-1, 1]$. We see if using the NN output as an initial control policy allows for faster convergence for iLQR.

   The results of this section are discussed in the following section.

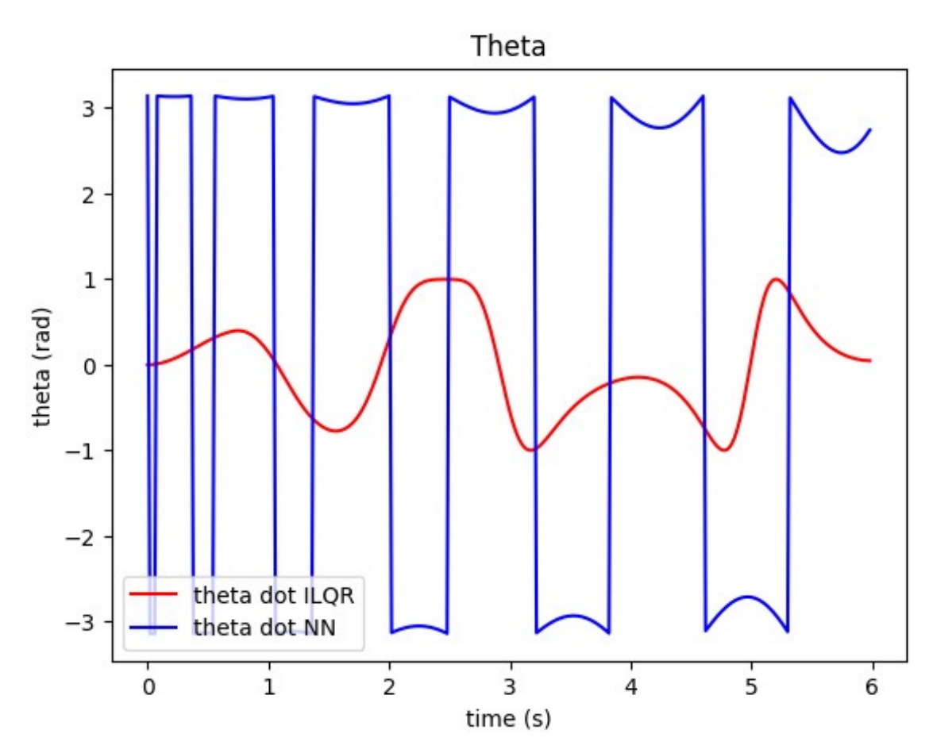
Results

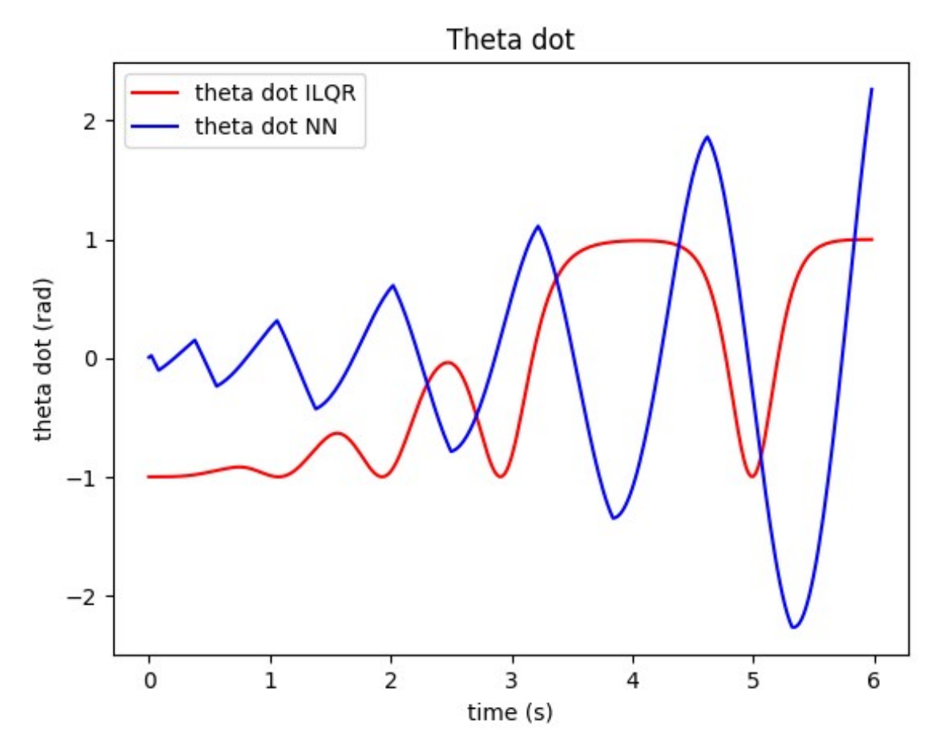
1. **Comparing ILQR and NN:**
   - Comparing against training data **(left)** and new test data **(right)**
   - We see that the NN output controls very generally coincide with iLQR output with training data while it does not align well with test data.
   - This can be possibly due to oversights such as over training with the test data or not a sufficiently varied training data set.





   - We can compare the states realized by both the models (θ on the left and $\dot{\theta}$ on the right
   - While they both are very different between models, we can see some slight similarities in $\dot{\theta}$ while it seems that θ oscillates very sharply and with little to no correspondence to the iLQR solution.





2. **Initializing iLQR with NN:** Using the NN controls as an initialize also results in the iLQR algorithm taking much longer to converge than with a randomly sampled initialization. This may be a consequence of the the vast differences in behaviour of the 2 models as shown earlier.

Next Steps

1. Investigate what is required for a sufficient training data set for NN to function feasibly as a controller. Also, determine optimal hyper parameters for NN.
2. With a fine tuned and designed controller, apply as a solver for the local optimal control problems in MPC and compare against other algorithms used with MPC.

References

[1] Weinan E. Empowering Optimal Control with Machine Learning: A Perspective from Model Predictive Control *IFAC-PapersOnLine 55.30 (2022) 121-126*. 2022.

[2] Travis DeWolf. *The Iterative Linear Quadratic Regulator Algorithm*, 2016

[3] Richard E. Bellman. Dynamic Programming. *Princeton University Press,* 1957.

[4] Jiequn Han. Weinan E. Deep Learning Approximations for Stochastic Control Problems *arXiv* 2016.

Thank you to our mentors, Dao Nguyen and Chinmaya Kausik.