Generating Control Policies for Autonomous Vehicles Using Neural ODEs

Houston Lucas
University of Nevada
Reno, NV 89557
houstonlucas@nevada.unr.edu

Richard Kelley
University of Nevada
Reno, NV 89557
rkelley@unr.edu

Abstract

The problem of robot control often requires solving a system of ordinary differential equations (ODEs). Traditionally this has been accomplished by using iterative ODE solvers. These solvers start with an initial guess, which is iteratively improved to converge to a correct solution. However, traditional solvers can be slow and do not combine well with other systems since they are not differentiable. In response, some researchers have proposed using neural networks in an end-to-end system that directly maps perceptual inputs to control actions. Because of their differentiablity, end-to-end approaches can be composed with other modules more readily than traditional ODE solvers. However the end-to-end approach no longer carries the guarantee that the solution obeys the required dynamics. We propose a framework for using Neural ODE to combine the flexibility of the end-to-end approach with the guarantees of traditional solvers. In our approach a neural network is used to provide the initial guess to a differentiable ODE solver. The ODE solver then yields a solution trajectory. We use this trajectory to improve the guesses of the neural network. This framework allows the neural network to learn initial guesses that are close to the correct solution, improving overall system performance while ensuring that dynamics constraints are always satisfied. We demonstrate the utility of this framework in the case of robot control, where we use it to solve a family of boundary value problems that are essential for steering an autonomous vehicle to a goal state.

1 Introduction

For robots to have a positive impact on society, they must be able to safely navigate in complex changing environments. Because robots are physical systems, their motion is ultimately governed by sets of differential equations, so that safe navigation reduces to solving a system of (typically) ordinary differential equations subject to some boundary value constraints (known as *boundary value problems* or BVPs). Some BVPs can be very slow to solve and as such it has been suggested to avoid solving them where possible(1, 14.3.3). This however limits the scope of applicable algorithms for systems where BVP solutions are slow.

The traditional approach to building a robot navigation system is to write down a parametric form for the robot's dynamics, estimate the parameters in a system identification process, and then solve the equations (offline in advance or online as the robot moves) to determine appropriate controls (2). This process is time-consuming at every step, requiring substantial expertise to produce a system that functions in realistic settings. The resulting systems are also difficult to integrate with other subsystems such as perception and low-level control. In light of these challenges, some researchers have proposed replacing the entire robot control pipeline with an end-to-end approach that maps perceptual inputs to controls using large neural networks (3). This approach has shown promise in small demonstrations (4), but has yet to be applied in demanding safety-critical settings, in large part because end-to-end approaches have no guarantees of correctness or safety: if an end-to-end model successfully drives a car down a road, that provides little confidence that the system can operate successfully in any other conditions.

In this work we offer two contributions: first we describe how to extend the Neural ODE framework to create a fully differentiable solver for boundary value problems, and second we show that using a neural network to initialize our BVP solver leads to significant performance improvements along with a guarantee of safe operation regardless of the neural network's outputs.

2 Initial & Boundary Value Problems

We are interested in controlling robots by solving differential equations where we are given both a starting point (typically the robot's current position) and an endpoint (some goal to which we want to move). Before we show how neural networks can be used to safely solve this problem, we review the relevant numerical mathematics of initial value problems and boundary value problems, pointing out connections to machine learning along the way.

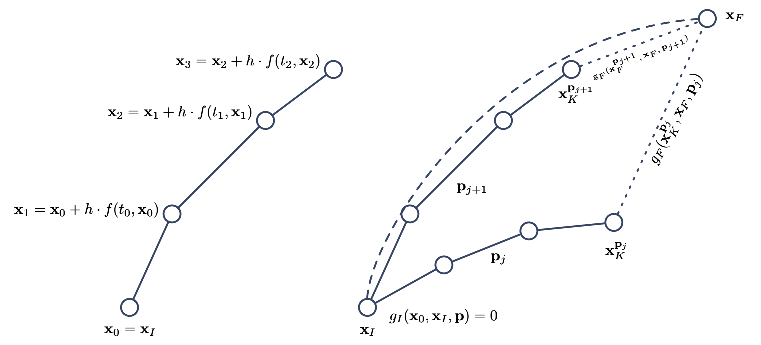
$\mathbf{x}_3 = \mathbf{x}_2 + h \cdot f(t_2, \mathbf{x}_2)$
$\mathbf{x}_2 = \mathbf{x}_1 + h \cdot f(t_1, \mathbf{x}_1)$
$\mathbf{x}_1 = \mathbf{x}_0 + h \cdot f(t_0, \mathbf{x}_0)$
$\mathbf{x}_0 = \mathbf{x}_I$
$\mathbf{x}_F$
$g_F(\mathbf{x}_F^{\mathbf{p}_{j+1}}, \mathbf{x}_F, \mathbf{p}_{j+1})$
$\mathbf{x}_K^{\mathbf{p}_{j+1}}$
$g_F(\mathbf{x}_K^{\mathbf{p}_j}, \mathbf{x}_F, \mathbf{p}_j)$
$\mathbf{p}_{j+1}$
$\mathbf{p}_j$
$\mathbf{x}_K^{\mathbf{p}_j}$
$\mathbf{x}_I$
$g_I(\mathbf{x}_0, \mathbf{x}_I, \mathbf{p}) = 0$

Figure 1: (left) Euler's method for solving an initial value problem. (right) Shooting method for solving a boundary value problem. The iterative solver updates $\mathbf{p}_j$ to $\mathbf{p}_{j+1}$ to reduce the constraint violation g_F.

To begin, suppose we have a physical system characterized by a *state vector* $\mathbf{x} \in \mathbb{R}^n$. We will write the derivative of the state with respect to time as $\dot{\mathbf{x}}$, and we will fix an interval of time $[t_0, t_F] \subseteq \mathbb{R}$. With this notation, the *initial value problem* (IVP) is to find a function $\mathbf{x} : \mathbb{R} \to \mathbb{R}^m$ satisfy the equations $\dot{\mathbf{x}} = f(t, \mathbf{x})$ and $\mathbf{x}(t_0) = \mathbf{x}_I$ for some given initial value $\mathbf{x}_I$. The problem is, starting from some externally given value $\mathbf{x}_I$, to find a *trajectory* $\mathbf{x}(t)$ that is consistent with the dynamics that are specified by the function f, which may be quite complicated. In practice such problems are solved numerically. A representative numerical method is *Euler's method*, which starts from $\mathbf{x}_0$ and repeatedly extends the solution forward in time by a small increment $h \in \mathbb{R}$ up to some final iteration K using the rule

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot f(t_k, \mathbf{x}_k). \tag{1}$$

The sequence of points $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K\}$ is then returned as the solution trajectory. Although simple to describe, implementations of Euler's method with finite-precision arithmetic suffer from accuracy and stability problems; generalizations of the method use additional function evaluations to mitigate these problems, but ultimately share the same iterative structure. Since all of the operations involved in these methods are differentiable, iterative ODE solvers can be implemented in an automatic differentiation framework such as PyTorch (5). This allows us to compute the gradient of some loss function with respect to the inputs of the ODE solver. One of the significant innovations of the Neural ODE approach of Chen et. al. (6) is to show this can be done with constant memory cost.

Solving IVPs is a critical intermediate step in solving *boundary value problems* (BVP), which is our primary goal and where our approach inherits its efficiency and safety guarantees. To specify a BVP, suppose we are given not just an initial state for our system, but also a final state $\mathbf{x}(t_F) = \mathbf{x}_F$. In addition to the state of the system, BVPs also assume the existence of a vector of *parameters* $\mathbf{p} \in \mathbb{R}^m$. These parameters can be adjusted to change the shape of the solution trajectory, and are used in numerical procedures to solve the BVP. In the presence of parameters, we write the differential equation as $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{p})$ and the constraint equations as $g_I(\mathbf{x}(t_I), \mathbf{x}_I, \mathbf{p}) = 0$ and $g_F(\mathbf{x}(t_F), \mathbf{x}_F, \mathbf{p}) = 0$ for functions g_I and g_F.

There are several approaches to solving BVPs; we focus on the class of *shooting methods*. In this approach, the initial conditions are fixed, an initial parameter vector $\mathbf{p}_0$ is chosen, and a numerical procedure is used to solve the IVP $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{p}_0)$, $\mathbf{x}(t_0) = \mathbf{x}_0$. The solution is a sequence $\{\mathbf{x}_0^{\mathbf{p}_0}, \ldots, \mathbf{x}_K^{\mathbf{p}_0}\}$. For an arbitrarily chosen parameter vector $\mathbf{p}$ the final value constraint $g_F(\mathbf{x}_K^{\mathbf{p}}, \mathbf{x}_F, \mathbf{p}) = 0$ will generally be violated. A shooting method for solving the BVP consists of using a root-finding algorithm (7) to choose a sequence of parameter vectors $\{\mathbf{p}_j\}_{j=0}^{\infty}$ so that

$$\lim_{j \to \infty} g_F(\mathbf{x}_K^{\mathbf{p}_j}, \mathbf{x}_F, \mathbf{p}_j) = 0. \tag{2}$$

That this sequence converges for a broad class of problems is a consequence of standard existence results in the theory of BVPs (8). In general, the rate of convergence of this process depends very strongly on the choice of $\mathbf{p}_0$. Our main contribution is to show that we can use a neural network to learn how to choose initial parameter vectors that lead to rapid convergence in practice.

3 Neural BVP

In the Neural ODE framework described by Chen et. al. (6), we work with the ordinary differential equation $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), t, \theta)$ where $t \in [t_0, t_1] \subset \mathbb{R}$ is the scalar input, $\mathbf{x}(t)$ is the unknown function we wish to find, and θ is a vector of parameters to the function f. The Neural ODE approach enables the computation of the gradient of a scalar-valued loss function L with respect to the inputs of an ODE solver.

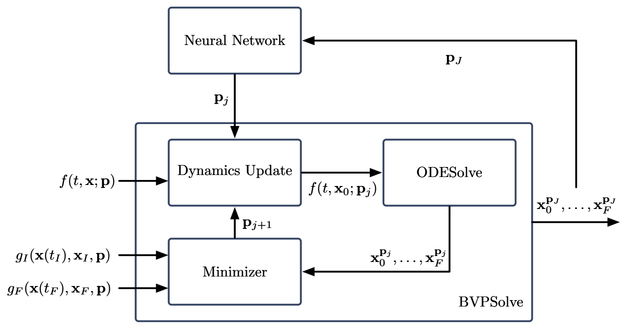
Neural Network
$\mathbf{p}_J$
$\mathbf{p}_j$
$f(t, \mathbf{x}; \mathbf{p})$
Dynamics Update
$f(t, \mathbf{x}_0; \mathbf{p}_j)$
ODESolve
$\mathbf{x}_0^{\mathbf{p}_J}, \ldots, \mathbf{x}_F^{\mathbf{p}_J}$
$\mathbf{p}_{j+1}$
$g_I(\mathbf{x}(t_I), \mathbf{x}_I, \mathbf{p})$
$g_F(\mathbf{x}(t_F), \mathbf{x}_F, \mathbf{p})$
Minimizer
$\mathbf{x}_0^{\mathbf{p}_j}, \ldots, \mathbf{x}_F^{\mathbf{p}_j}$
BVPSolve

Figure 2: Solver architecture.

We propose to use this approach as the inner loop in a differentiable BVP solver (see Figure 2). Our approach is encapsulated in a function, `BVPSolve`, which takes as input the parametrized dynamics function $f(t, \mathbf{x}; \mathbf{p})$, the constraint functions g_I and g_F, and an initial guess $\mathbf{p}_0$ for the parameter vector. Given parameter vector $\mathbf{p}_j$, The dynamics update module determines the dynamics function $f(t, \mathbf{x}; \mathbf{p}_j)$ to pass into ODESolve. That function produces a candidate solution $\mathbf{x}_0^{\mathbf{p}_j}, \ldots, \mathbf{x}_F^{\mathbf{p}_j}$. If that solution satisfies the boundary conditions given by g_I and g_F, the trajectory is returned. Otherwise, the minimizer module computes an updated parameter vector $\mathbf{p}_{j+1}$ that is used in the next solver iteration.

Unlike an end-to-end approach that simply produces an output trajectory, this approach ensures that the output of the neural network is corrected by the iterative solver, providing a guarantee that the system can be used for safety-critical applications such as steering an autonomous vehicle (see Section 4).

4 Evaluation

As mentioned in Section 1, one use of boundary value problems is to control an autonomous system. We use this problem to demonstrate that well selected values of $\mathbf{p}_0$ result in fewer iterations to convergence.

We assume our vehicle moves in a plane, so that its state vector is given by a position (x, y) and a heading θ. The vehicle is further parametrized by the wheelbase $L = 2.33m$ and has a constant speed $v = 10m/s$. The steering angle is set by a function $\delta(t, \mathbf{p})$, where $\mathbf{p}$ contains the parameters of a spline. The vehicle moves with dynamics

$$\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{p}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v\tan\delta(t,\mathbf{p})}{L} \end{bmatrix} \tag{3}$$

For our test problem the vehicle starts in the state $\mathbf{x}(t_I) = (0, 0, 0)$. The objective is to finish at time $t_F = 1s$ with a lateral displacement of d meters to the right in the state $\mathbf{x}(t_F) = (\alpha, d, 0)$ where α is unspecified.

We solve this problem with two approaches. In the first approach we produce $\mathbf{p}_0$ by sampling the components near zero, which results in a steering angle near zero. In the second approach we have a neural network producing the value for $\mathbf{p}_0$ from the target displacement d. We then start the Boundary Value Problem solver with this initial guess and record how many iterations it takes to converge. For each approach we uniformly sampled 1000 values for d in the range [-0.7, 0.7] and recorded the number of iterations to convergence. The average number of iterations to convergence was 501.35 iterations for the first approach and 14.45 iterations for the second approach. On average the neural network approach converged 34 times faster than the traditional approach. A plot of d vs. number of iterations to convergence can be seen in Figure 3.

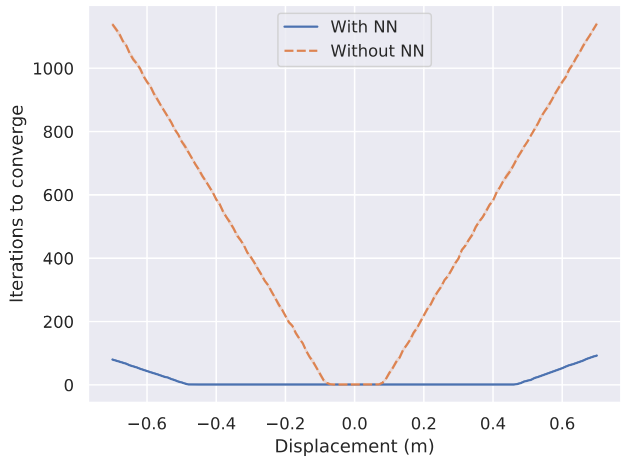

Figure 3: Displacement vs. Iterations to convergence. The low number of iterations to converge near $d = 0$ for the non-neural network approach is explained by the fact that $\mathbf{p}_0$ sampled yields a steering value of zero which drives a straight line giving a lateral deviation $d = 0$.

5 Conclusion and Future Work

We have shown that a system that combines a neural network with a differentiable iterative solver for boundary value ordinary differential equations inherits the flexibility and speed of the neural network and the safety guarantees of the iterative solver. By using the neural network to produce an initial guess rather than final controls, we can guarantee that the combined system will be applicable wherever we would have used the iterative solver alone. Moreover, we have shown that the trained hybrid system runs more than an order of magnitude faster than the iterative system alone, expanding the range of applications of the system to more demanding control problems. We are investigating such applications, as well as methods to guarantee graceful degradation of the system in the presence of adversarial inputs. In the future, we expect that our proposed combination of a neural network and an iterative solver to find use in many more domains where safe and fast operation is essential.

References

[1] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[2] Shilpa Gulati, Chetan Jhurani, and Benjamin Kuipers. A nonlinear constrained optimization framework for comfortable and customizable motion planning of nonholonomic mobile robots - part I. *CoRR*, abs/1305.5024, 2013.

[3] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[6] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018.

[7] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[8] U. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Society for Industrial and Applied Mathematics, 1995.