2020 American Control Conference
Denver, CO, USA, July 1-3, 2020

Deep Learning for Control: a non-Reinforcement Learning View

Ion Matei, Raj Minhas, Maksym Zhenirovskyy, Johan de Kleer and Rahul Rai

Abstract— **Deep learning platforms have become hugely popular due to their successes in natural language processing and image processing. Our objective is to show how deep learning platforms can be used for control problems. We do not make judgments about their performance as compared to traditional control approaches. We show that the main challenge when using deep learning platforms for learning control policies for nonlinear systems is ensuring the stability of the learning algorithm that depends on the stability of the closed loop system during the learning process. We discuss two approaches for overcoming the potential instability of the optimization algorithm, and showcase them in the context of learning a stabilizing controller for an inverted pendulum.**

I. Introduction

Our main objective is to show how deep learning (DL) platforms can be used to learn control policies. We present the challenges, and propose approaches to deal with them. The proliferation of DL algorithms is enabled by their ability to deal with large amount of data, and by their success in natural language processing [7], [19] and image processing [4], [14], with direct applications in autonomous vehicles control [18]. DL algorithms are based on large scale, first order gradient-based optimization algorithms used to minimize error/loss. Many DL platforms have an automatic differentiation feature [1] that leads to the ability to accurately evaluate gradients of the loss functions. More generally, there is a growing interest in "differentiable programming" [21], a programming paradigm where programming constructs are enhanced with differential operators. In such a paradigm, the key idea is to construct a causality graph describing the control flow and data structures of the program, and the application of the differential operators by backtracking on the causality graph. The graph can be static (TensorFlow [20]) or dynamic (Pytorch [3], Autograd [12]). In this paper, we use automatic differentiation (AD) to learn control policies and use Autograd as the main tool for executing AD. The working example for illustrating the design of control policies is the stabilization of the inverted pendulum, a nonlinear system with an unstable equilibrium point. We use gradient based optimization algorithms to learn parameterized, state dependent control maps (e.g., neural networks) that stabilize the pendulum at the unstable equilibrium point. The optimization algorithms train the controller parameters based on a state- and control-dependent loss function. We also learn control inputs explicitly, by treating them as optimization variables. We show that this process is non-trivial and that the choice of initial values of the controller parameters is crucial for the success of the learning process. We propose algorithms for deriving stabilizing, sub-optimal controllers that we use as initial condition for the DL algorithms. This initial control policy derived from these sub-optimal controllers is improved by minimizing a loss function. It augments the existing DL approach for learning stabilizing controllers using reinforcement learning [16]. It is not our objective to establish which approach is more effective, but rather to introduce an alternative view.

Why would we use DL platforms to compute control policies? It is not necessary for linear systems since classical control theory can solve this problem efficiently. However, for nonlinear systems with a large number of state variables (expressed in some physics-based modeling language e.g., Modelica [5]), extracting its mathematical model and computing control policies is not trivial. In these cases, we can combine model transformation with automatic differentiation to enable computation of system linearization without the need for numerical approximations, and to provide the computational infrastructure to search for control policies regardless of the type of optimization loss function. Note that we are utilizing the automatic differentiation feature of the DL platforms rather than their ability to deal with large models.

Notations: The vector or matrix norms are denoted by $\|\cdot\|$. Indices refer to a particular norm, when needed. A (closed) neighborhood of size ε around a point z_0 is defined as $\mathcal{B}_\varepsilon(z_0) = \{z,\ \|z - z_0\| \leq \varepsilon\}$. The Jacobian of a vector valued function $f(x)$ is denoted by $\nabla f(x)$. The Jacobian becomes the gradient for a scalar valued function. For a function $f(x,y)$, $\frac{\partial f(x,y)}{\partial x}$ denotes its partial derivative with respect to x, evaluated at (x,y). The spectral radius of a matrix A is denoted by $\rho(A)$.

Paper structure: Section II describes the problem setup and the working example. Section III shows the connection between the stability of the closed loop system and the stability of the learning algorithm. Section IV describes three approaches for solving optimal control problems, and how ideas and tools from machine learning can be instantiated in control problems.

II. Problem description

We assume that a dynamical system is described by a possibly nonlinear ordinary differential equation (ODE)

Ion Matei, Raj Minhas, Maksym Zhenirovskyy and Johan de Kleer are with the Palo Alto Research Center, Inc. (PARC) (emails: imatei@parc.com, minhas@parc.com, mzhenirovskyy.parc@gmail.com, dekleer@parc.com). Rahul Rai is with University at Buffalo, SUNY (email: rahulrai@buffalo.edu).

This material is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) Award HR00111890037 Physics of AI (PAI) Program.

given by

$$\dot{z} = f(z,u;w), \tag{1}$$

where z denotes the state vector, u represents the input signal, and w is a set of system parameters. Without loss of generality, we assume that $z = 0$ is a, possibly unstable, equilibrium point. Our objective is to learn control inputs that stabilize the state at the equilibrium point. We consider state-dependent, parametric (e..g, $u = g(z;\beta)$ where β are the controller parameters) as well as non-parametric control schemes (e.g., control inputs are optimization variables). In addition, we would like to optimize some state-dependent loss function $\mathcal{L}(z_0,u) = \int_0^T l(z,u)dt$, for some time horizon T and some initial condition z_0. Control theory offers many options to approach this problem, including methods based on linearization, Lyapunov functions [8] or model predictive control approaches [6]. In this paper, we explore how DL models and platform features can be used to learn control policies. We borrow principles such as transfer learning to ensure the stability of the learning process and reduce the time to compute an optimal solution. More formally, the goal is to solve an optimization problem of the form $\min_u \mathcal{L}(z_0,u)$. We can further impose additional conditions on u such as form (e.g., $u = g(z;\beta)$) or magnitude limitations (e.g., u belongs to some bounded set). In control theory, typical loss functions are quadratic in the state and the control input: $l(z,u) = \frac{1}{2}(z^T Qz + u^T Ru)$ for some semi-positive matrices Q and R. If $f(z,u;w)$ is linear in the state and input, we recover the well known linear quadratic regulator (LQR) optimal control problem [11].

For our working example, we are using the inverted pendulum model [2], whose dynamics are given by

$$\begin{aligned}
\dot{x} &= v,\\
\dot{v} &= \frac{(J+ml^2)(bv - F - ml\omega^2\sin\theta) - m^2l^2g\sin\theta\cos\theta}{(ml\cos\theta)^2 - (m+M)(J+ml^2)},\\
\dot{\theta} &= \omega,\\
\dot{\omega} &= \frac{ml(F\cos\theta + ml\omega^2\sin\theta\cos\theta - bv\cos\theta + (m+M)g\sin\theta)}{(ml\cos\theta)^2 - (m+M)(J+ml^2)},
\end{aligned}$$

where x and v are the cart's position and velocity, respectively, while θ and ω are the pole's angle and angular velocity, respectively. The symbol F denotes the force acting on the cart and plays the role of the input signal. The state vector is given by $z^T = [x,v,\theta,\omega]$. The remaining symbols represent parameters of the system and their values are listed in Table I. Note that in its original form, the

TABLE I

INVERTED PENDULUM PARAMETERS

M	m	l	g	J	b
0.5	0.2	0.3	9.81	0.006	0.1

dynamics of the inverted pendulum system are represented as a differential algebraic equation (DAE). We explicitly solved for the accelerations to generate the ODE form. It is not always possible to transform DAEs into ODEs. In such cases DL platforms (e.g., Pytorch or TensorFlow) cannot be used since they do not support DAE solvers. In that case, we need to use platforms that support DAE solvers enhanced with sensitivity analysis capabilities such as DAETools [13], a Python package.

III. Learning dynamics

DL platforms require training data and first order gradient-based algorithms to learn model parameters. When learning control policies, we start with a set of initial conditions of the system ODE, and the training data is generated online through model simulation, i.e., by solving an ODE for each iteration of the learning algorithm. For brevity, we use a continuous time version of the learning algorithm that can be readily discretized to obtain the familiar gradient descent algorithm. To minimize the chain rule application, when considering a state-dependent parameterized map, we assume that $g(z;\beta)$ has already been integrated in the loss function l, and vector field f, so that they only depend on z and β. The continuous dynamics of the gradient descent algorithm are given by $\dot{\beta} = \nabla_\beta \mathcal{L}(\beta),\ \beta(0) = \beta_0$. The derivative of the loss function can be explicitly written as

$$\nabla_\beta \mathcal{L}(\beta)^T = \int_0^T \left(\frac{\partial l}{\partial z}\frac{\partial z}{\partial \beta} + \frac{\partial l}{\partial \beta}\right)dt.$$

For notational brevity let $z_\beta = \frac{\partial z}{\partial \beta}$, which represents the sensitivity of the state vector with respect to the controller parameters. The sensitivity z_β has its own dynamics, and is given by:

$$\dot{z}_\beta = \frac{\partial f(z)}{\partial z} z_\beta + \frac{\partial f(z)}{\partial \beta},\ z_{\beta_0} = 0. \tag{2}$$

A first observation is that (2) is identical to the linearized system dynamics around the equilibrium point, up to the initial conditions. More importantly, such linearization enables the stability analysis of the closed loop system around the equilibrium point. In particular, as is well established in control theory [8], the spectral properties of $\frac{\partial f(0)}{\partial z}$ determine the behavior of the system near the equilibrium point. This result states if he real parts of eigenvalues of the matrix $\frac{\partial f(0)}{\partial z}$ are negative, there exists an $\varepsilon > 0$ depended ball $\mathcal{B}_\varepsilon = \{z, \|z\| \leq \varepsilon\}$ such that $\lim_{t\to\infty} \|z(t)\| = 0,\ \forall z_0 \in \mathcal{B}_\varepsilon$. The result gives us the means to test, at least locally, the stabilizing properties of control maps.

A. But where to start?

The main message here is that the stability of the sensitivity vector z_β is directly related to the (local) stability of the closed loop system. More importantly, an unstable sensitivity vector will induce instability in the gradient of the loss function. As a result, unless the gradient descent is able to update the parameters β fast enough such that the closed loop system becomes stable, the gradients of the loss function will grow unbounded leading to the instability of the parameter learning algorithm itself. Since gradient based algorithms are first order algorithms, they typically have a slow convergence rate. Hence, unless the initial controller parameters are such that they induce a stable closed loop system, the learning algorithm will very

likely fail. We performed an experiment with respect to the stability properties of the initial controller parameters. We considered a linear map for the controller since it has only four parameters. We drew 10^5 controller parameters from the interval $[-10, 10]$. Out of 10^5 parameter choices, only 18 controllers were stabilizing near the equilibrium point. This suggests an extremely high probability (0.99982 in this case) to start with an unstable controller.The situation becomes even worse when considering more complex controller maps, e.g., neural networks, whose number of parameters is considerably larger. *Therefore, the main challenge with using DL platforms for control design is ensuring the stability of the learning algorithm*. In what follows, we propose two such strategies that have a transfer learning interpretation. The first strategy (Sections IV-A and IV-B) is based on first learning an initial controller that is guaranteed to stabilize the close loop, at least near the equilibrium point. The second strategy (Section IV-C) is based on a sequence of learning problems where we start with a short enough time horizon to avoid gradient blow-up, and then keep increasing the time horizon until we converge to a stable controller.

B. On the stability of the learning process

We now formally show why stability of the close loop system matters for learning a controller. Consider the linear dynamics $z(t+1) = Az(t) + bu(t)$ with $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ and a linear controller parameterization $u(t) = \beta^T z(t)$. Our goal is to find a stabilizing controller that minimizes the quadratic loss function $\mathcal{L}(\beta) = \frac{1}{2}\sum_{t=1}^{N} z(t)^T z(t)$. We use a first order, gradient based optimization algorithm given by $\beta_{k+1} = \beta_k - \alpha_k \nabla\mathcal{L}(\beta_k)$, where α_k is the iteration step-size. The next result shows that provided we start with a stabilizing controller, the gradient descent algorithm generates a sequence of stabilizing controllers. We assume that (A, b) is controllable so that stabilizing controllers exists for any initial conditions.

Proposition 3.1: Let $u(t) = \beta_0^T z(t)$ be a stabilizing controller, where β_0 is the initial value for the vector of optimization variables β. Then there exits a sequence of step-sizes $\{\alpha_k\}_{k\geq 0}$ such that the sequence of intermediate controllers $\{\beta_k\}_{k\geq 0}$ generated by the gradient descent algorithm are stabilizing.

Proof: The gradient of the loss function can be explicitly written as

$$\nabla\mathcal{L}(\beta)^T = \frac{1}{N}\sum_{t=1}^{N} z(t)^T z_\beta(t),$$

where $z(t) = (A + b\beta^T)^t z(0)$ and $z_\beta(t+1) = (A + b\beta^T)z_\beta(t) + bz(t)^T$. Recalling that $z_\beta(0) = 0$, we have that

$$\begin{aligned} z_\beta(t) &= \sum_{i=0}^{t-1} (A + b\beta^T)^i b z(t-1-i)^T \\ &= \sum_{i=0}^{t-1} (A + b\beta^T)^i b z(0)^T \left[(A + b\beta^T)^T\right]^{t-1-i}. \end{aligned}$$

Since $u(t) = \beta_k^T z(t)$ is a stabilizing controller, we have that there exits $\gamma > 0$ and $0 < \lambda(\beta) < 1$ so that $\|(A + b\beta^T)^i\| \leq \gamma\lambda^i$. In turn, this imply that $\|z(t)\| \leq \gamma_1\lambda^t$ for some positive γ_1. In addition, we have that $\|z_\beta(t)\| \leq \gamma\|bz(0)^T\| \sum_{i=0}^{t-1} \lambda^i\lambda^{t-1-i} = \gamma_2 t\lambda^{t-1}$, for some positive γ_2. Next we obtain an upper bound for the loss function gradient. Namely, we have that

$$\|\nabla\mathcal{L}(\beta)\| \leq \gamma_3 \sum_{t=1}^{N} t\lambda^{2t-1} = \gamma_3 \frac{\lambda^{2N+1}(N\lambda^2 - N - 1) + \lambda}{(\lambda^2 - 1)^2},$$

for some positive scalar γ_3. The scalars γ_1, γ_2 and γ_3 are positive functions of $z(0)$, b and γ. This shows that if the controller at learning step k is stabilizing, the gradient is bounded.

Next we show that there exists α_k such that the controller $u(t) = \beta_{k+1}^T z(t)$ is stabilizing. Let $c_k = \nabla\mathcal{L}(\beta_k)$ be the gradient vector at iteration k. Then the stability of the closed loop system is determined by the transition matrix $A + b\beta_{k+1}^T = A + b\beta_k^T + \alpha_k bc_k^T = A + b\beta_k^T + M_k$. Since the gradient of the loss function is upper bounded, there exists a positive constant η_k so that $\|M_k\| \leq \eta_k$. In addition, there exists also $\gamma_k > 0$ and $0 < \lambda_k < 1$, such that $\|(A + b\beta_k^T)^t\| \leq \gamma_k\lambda_k^t$, for all $t \geq 0$. Using a perturbation theory argument, similar to the one in Theorem 24.7, page 455 [15], we can show that the state vector when using $u(t) = \beta_{k+1}^T z(t)$ satisfies $\|z(t)\| \leq \gamma_k(\lambda_k + \gamma_k\eta_k)^t\|z(0)\|$. But since $0 < \lambda_k < 1$, we can choose α_k small enough such that η_k becomes small enough for the inequality $\lambda_k + \gamma_k\eta_k < 1$ to be satisfied. This, in turn, ensures that the control scheme $u(t) = \beta_{k+1}^T z(t)$ is stabilizing, which concludes the proof. ■

Remark 3.1: The theoretical bound on the step-size α_k that ensures closed loop stability is $\alpha_k < \frac{1-\lambda_k}{\gamma_k\|bc_k^T\|}$. This bound should be compared with the bound that ensures the convergence of the gradient descent algorithm and the minimum between the two bounds should be chosen.

Remark 3.2: The previous result acts as a surrogate for the nonlinear case and an arbitrary loss function. It shows that the choice of the learning rate α_k is the result of a trade-off between the stability of the close loop system and the convergence speed of the optimization algorithm. When we start with an unstable controller, the rate at which the controller converges to a stabilizing one may not be fast enough to overcome the increase in the gradients magnitude due to the instability of the closed loop system. This instability becomes even more damaging for larger time horizons.

IV. Learning algorithms for control

In this section we present three approaches for learning a stabilizing controller. All approaches borrow ideas from transfer learning [17] methods used in deep learning. In the first approach, we learn a parameterized map for a stabilizing controller that minimizes a quadratic loss function. We choose the initial values of the map parameters by separately training a controller that stabilizes the linear approximation of the nonlinear dynamics around the equilibrium point. In the second approach, we again use a stabilizing controller learned for the linearized dynamics, but this time we use the non-parameterized control inputs as initial conditions to solve a finite horizon optimal control problem that explicitly generates optimal control inputs. In the third approach, we

again learn a parameterized, state-dependent control map by solving a sequence of optimal control problems whose time horizons are sequentially increased. For all three approaches, we used Autograd [12] to compute the gradients of the loss function.

A. *Learning a state-dependent control parameterization with a stabilizing initial controller*

Learning a stabilizing initial controller is based on manipulating the spectral properties of the Jacobian of $f(z;\beta)$ at the equilibrium point such that we ensure the existence of an attractor around the equilibrium point. Recall that one of the approaches for finding a stabilizing controller is to locally linearize in terms of the state z and control input u in order to obtain a linear ODE $\dot{z} = Az + Bu$. Under a controllability assumption that ensures the existence of a stabilizing linear controller, a simple pole placement approach will find a linear control $u = Kz$. In our case, *we will not impose a linear constraint on the controller* since our objective to transfer this initial controller to a nonlinear map.

Control theory connects the local stability properties of the closed loop system to the spectral properties of the Jacobian $\frac{\partial f(0;\beta)}{\partial z}$. A key feature of DL platforms is the ability to automatically differentiate loss functions. By using this feature, we generate a function that can be repeatedly evaluated at any state and control parameters pair (z,β). More importantly, it can be applied to high dimensional vector fields $f(z;\beta)$. It is more convenient to look at the time-discrete version of $A(\beta) = \frac{\partial f(0;\beta)}{\partial z}$. For a small enough time step h, the discrete version of $A(\beta)$ is given by $A_d(\beta) = e^{A(\beta)h} = I + \frac{1}{1!}hA(\beta) + \frac{1}{2!}h^2A(\beta)^2 + \ldots$. Since the matrix multiplication operations are efficiently executed on DL platforms, we can keep enough terms of the Taylor expansion to ensure a good approximation of the matrix exponential. The stability of the closed loop system can now be expressed in terms of the spectral radius $\rho(A_d;\beta)$, which is the largest of the absolute values of the eigenvalues of $A_d(\beta)$. The closed loop system is stable if and only if $\rho(A_d;\beta) < 1$. A useful inequality for our purposes is: $\rho(A_d;\beta)^k \leq \|A_d(\beta)^k\|$ for all $k \geq 1$ and for all consistent norms. Hence it is sufficient to impose that $\|A_d(\beta)^k\|$ converges to zero at an exponential rate to ensure the closed loop stability. To learn the controller parameters β, we solve the optimization problem

$$\min_{\beta} \ \max\{0, \|A_d(\beta)^k\| - \lambda^k\}, \tag{3}$$

for some large enough $k \geq 1$, and some positive real scalar $\lambda < 1$. Note that if we start with a large k and the initial values of β are such that $\rho(A_d;\beta) > 1$, computing the matrix powers will quickly lead to a numerical overflow. To avoid this situation, we solve a sequence of the optimization problems of type (3), where we can start with a small k, and continually increase k until no further improvement is observed. Note that it is not necessary to use the "max" operator in the cost function - its primary purpose is to stop the optimization when a feasible solution is reached. The process for learning stabilizing initial controller parameters is summarized in Algorithm 1. We applied Algorithm 1 for the inverted pendulum example where we modeled the controller map as a neural network with one hidden layer of size 20. We considered the bias vector to be zero. Hence we have a total of 104 parameters. The initial values of the entries of β were drawn uniformly from the interval $[0, 0.1]$, and the time step size was chosen as $h = 0.01$. The discrete matrix $A_d(\beta)$ was computed using 5 terms in the Taylor expansion. We solved (3) for $k \in \{1, 2, \ldots, 11\}$ using the Adam [9] algorithm, where each problem was allotted 2000 iterations and a stepsize of 0.001. The imposed decay rate was chosen as $\lambda = 0.999$. The results produced by Algorithm 1 are shown in Figure 1. We note that for $k \geq 8$ the parameterized control map is stabilizing. We also note an increase in the loss at the first iteration. This is expected, especially for matrices with complex eigenvalues. In fact, there exists some positive scalar c such that $\|A_d(\beta)^k\| \leq c\lambda^k$. We neglected this scalar since for large enough k, its effect can be neglected.

Algorithm 1 Algorithm for learning stabilizing initial control parameters based on Jacobian spectral properties

1: **Input data**: vector field map $f(z;\beta)$, discretization step h, set of increasing indices $K = \{k_1, k_2, \ldots, k_N\}$, initial values for the controller parameters β
2: Use auto-differentiation to generate the Jacobian map $\frac{\partial f(z;\beta)}{\partial z}$
3: Compute the discrete matrix $A_d(\beta;h) = I + \frac{1}{1!}hA(\beta) + \frac{1}{2!}h^2A(\beta)^2 + \ldots$
4: **for** k in K **do**
5: $\quad \beta \longleftarrow \arg\min_{\beta}\{0, \|A_d(\beta)^k\| - \lambda^k\}$
6: **Return** β

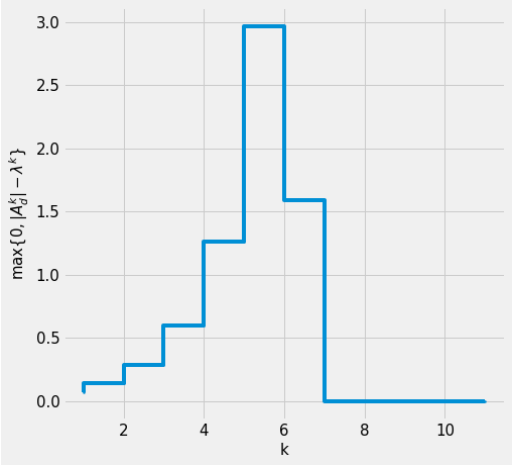

Fig. 1. Loss function $\max\{0, \|A_d(\beta)^k\| - \lambda^k\}$ evolution as a function of the k. Each optimization problem was limited to 2000 iteration with a step-size 10^{-3}.

The behavior of the state vector under the suboptimal stabilizing controller learned using Algorithm 1 is shown in Figure 2, where the initial conditions are $x_0 = 0.5$, $v_0 = 0$, $\theta_0 = 0.2$ and $\omega_0 = 0$. We can easily check that the eigenvalues of $\frac{\partial f(0;\beta)}{\partial z}$ are complex, with negative real parts.

We used the controller map learned using Algorithm 1 as the initial value for learning the control map parameters that minimize a quadratic loss function of the form

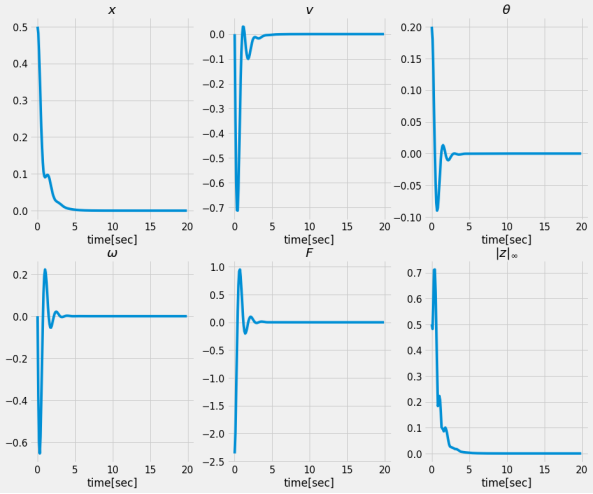

Fig. 2. Behavior of the state entries for 20 sec with a controller learned using Algorithm 1.

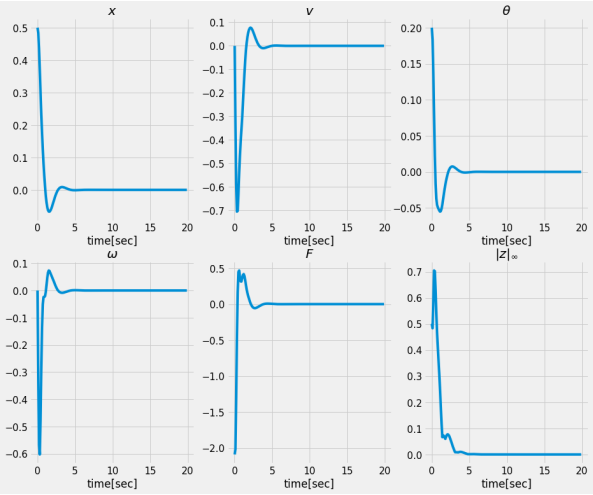

Fig. 3. Behavior of the state entries for 20 sec with a controller learned by minimizing the quadratic loss function $\frac{1}{N}\sum_{i=0}^{N} q\|z(t_i)\|^2 + r\|u(t_i)\|^2$ with $q = 1$ and $r = 2$.

$\frac{1}{N}\sum_{i=0}^{N} q\|z(t_i)\|^2 + r\|u(t_i)\|^2$, where $u(t) = \psi(z;\beta)$ and has the same structure as the controller learned using Algorithm 1. We executed 100 iteration of the Adam gradient-based algorithm, for $q = 1$ and $r = 2$, with a stepsize $\alpha = 0.001$, where the gradients of the loss function were computed using `Autograd`. The results are shown in Figure 3.

B. Learning finite horizon optimal control inputs

Model predictive control (MPC) [6] is a modern control approach that generates control inputs that optimize the present while considering the future as well. It is an effective approach to fight model uncertainties and external disturbances, but it has a high computational cost. At each time instance a finite horizon, optimal control problem must be solved. The feasibility of the MPC approach to feedback control depends on the size of the problem (i.e., the number of optimization variables), the number of constraints, the computational power of the platform implementing the MPC method, and the system time constants. One of the factors that affects how fast we converge to a (local) minima is the initial value of the optimization algorithms. Automatic differentiation can again be used to compute the gradient and the Hessian (if needed) of the loss function. We tested the effect of a "good" initial condition on the optimization algorithm solving one step of the MPC approach. We used the same loss function as in the previous section. However, here the control policy is no longer explicitly parameterized as a function of the state - the control inputs themselves are the optimization variables. Due to its ability to accommodate constraints, we use the Sequential Least SQuares Programming (SLSQP) optimization algorithm [10]. This time we constrained the control input (e.g., $\|u^2\| \leq 9$), and we used Autograd to compute the loss function gradients. These gradients were used as inputs for the SLSQP algorithm provided by the `SciPy` Python library. When training DL models, bound constraints are typically imposed through clipping. Here, they are explicitly considered. We tested the optimization algorithm under three scenarios: (a) initial control inputs generated using Algorithm I, (b) random initial conditions ($u \in [-2, 2]$), and (c) zero initial conditions. The optimization algorithm statistics (optimal value, number of iterations and number of function evaluations) for one finite horizon problem are shown in Table II. It is clear that using a stabilizing initial control decreases the convergence time. In MPC, we have to solve a sequence of such problems. We

TABLE II

RESULTS OF THE SLSQP ALGORITHM UNDER DIFFERENT INITIALIZATION SCENARIOS

Scenario	Optimal value	Iterations	Function eval.
Stabilizing initial conditions	0.1739	143	168
Random initial conditions	0.1737	296	316
Zero initial conditions	0.1744	292	316

may be tempted to use the solution of the previous step as initial condition for the next MPC step. If the time horizon is not long enough, we typically cannot guarantee that the control input stabilizes the feedback loop. As a consequence, the learning algorithm itself can become unstable leading to learning failure. To reinforce this idea, Figures 4 and 5 show the behavior of the inverted pendulum when using control inputs computed by minimizing $\frac{1}{N}\sum_{i=0}^{N} q\|z(t_i)\|^2 + r\|u(t_i)\|^2$ ($r = 1$, $q = 2$) over a 6 sec time horizon. It is clear from Figure 5 that the control input computed using zero initial condition is not stabilizing the pendulum. We can of course increase the time horizon to make sure the optimal control is stabilizing. This can have a highly damaging effect on the learning process though: if the initial control is not stable, gradients tend to explode, leading again to learning failure.

Since we have a better control of the parameters of the algorithm we have repeated the learning process using Adam algorithm. We empirically noticed that long time horizons tend to induce gradient explosions. This phenomenon can

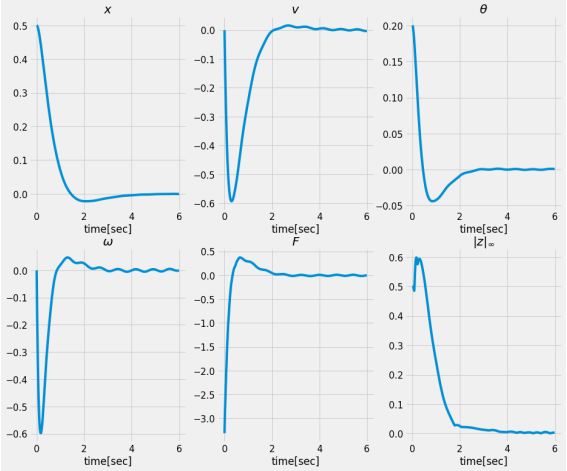

Fig. 4. One step of the MPC problem: stabilizing initial conditions.

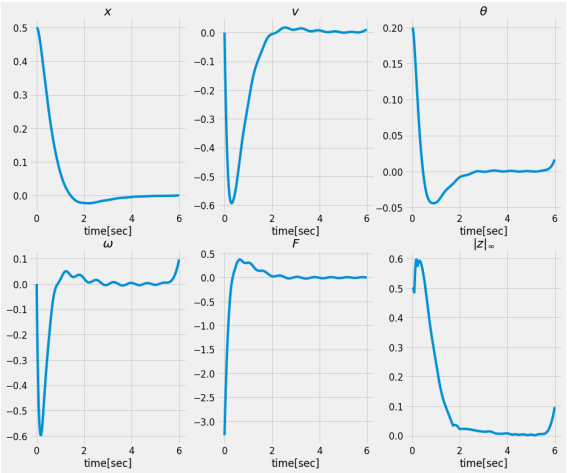

Fig. 5. One step of the MPC problem: zero initial conditions.

be explained as follows: during the search process, control inputs are generated that do not stabilize the system. As a result, for longer time horizons the instability is amplified. Consequently, the loss function gradients become unstable as well. We can control the gradient explosion to some extent by decreasing the learning rate, but we pay a price in terms of convergence speed.

C. Iterative learning based approach

We emphasized in the previous section that longer time horizon and unstable initial control inputs can lead to learning failure. Here we explore a "small steps" approach, where we solve a sequence of optimal control problems, where in the beginning we use small time horizons that are gradually increased. This approach does not require first computing a stabilizing control policy. We use a parameterized, state-dependent control map $u = \psi(z;\beta)$ - the objective is to learn the parameters of the map such that a quadratic loss function is minimized. We consider a set on initial states and our objective is to "encourage" the state vector to be close to zero and remain there after some time T_{ss}. As such, the loss function is now given by $\mathcal{L}(\beta;T) = \sum_{i=1}^{N} \mathcal{L}_i(\beta;z_0^{[i]})$, where $\mathcal{L}_i(\beta;z_0^{[i]}) = \sum_{j,T_{ss}\leq t_j\leq T} \|z(t_j)\|^2$, where index i refers to a particular initial state. To avoid the blow-up of gradients during the learning process, we use an iterative approach where we solve a sequence of optimization problems. Each such optimization problem is solved for a time horizon chosen from the set $\mathcal{T} = \{T_l\}$ with $T_{l+1} > T_l$. For time horizons $T_l < T_{ss}$, the loss function is defined as $\mathcal{L}_i(\beta;z_0^{[i]}) = \|z(T_l)\|^2$, which basically encourages the state vector at the end of the horizon to get closer to zero. In addition to the loss function, we add a regularization function that ensures stabilization happens at the zero equilibrium point. In particular, regularization function takes the form $\|f(0;\beta)\|^2$. The sequential learning method is summarized in Algorithm 2. Figures 6 and 7 show the learning results for two time horizons: 1 sec and 10 sec, respectively. The time at which we would like the system to be in steady state was chosen as $T_{ss} = 4$ sec. We chose 50 initial conditions with positions and angles selected randomly from the intervals $[-1,1]$ and [-0.5, 0.5] respectively. We used Autograd and Adam optimization algorithm to solve the optimization problems. The sequence of time horizons used was $\mathcal{T} = \{0.4, 0.6, 0.8, 1, 1.4, 2, 5, 6, 10\}$ sec. The intuition behind the reason such an approach works is that short time horizons control gradient magnitudes even for unstable feedback loops. In addition, starting from a pre-trained control policy at the previous time horizon, ensures faster convergence times since for small differences in two consecutive time horizons, the differences in the control policies (as defined by the controller parameters) are not significant.

Algorithm 2 Algorithm for learning stabilizing initial control parameters based on increasing time horizons

1: **Input data**: time horizon set $\mathcal{T} = \{T_0, T_1, \ldots, T_L\}$, initial values for the controller parameters β, set of initial conditions, regularization weight μ $\{z_0^{[0]}, \ldots, z_0^{[N]}\}$
2: **for** $T \in \mathcal{T}$ **do**
3: $\quad \beta \leftarrow \arg\min \mathcal{L}(\beta;T) + \mu\|f(0;\beta)\|^2$
4: **Return** β

V. Conclusions

We demonstrated that deep learning platforms can be used for training stabilizing controllers. We discussed the connection between the stability of the closed loop system and the stability of the learning algorithm. We demonstrated that starting with a stable initial controller, we are guaranteed the existence of gradient descent stepsizes that ensure the stability of the closed loop system during the learning process. We introduced three strategies to overcome learning instability. In the first approach, we learned an optimal nonlinear controller by using a sub-optimal stabilizing controller as the initial value. The sub-optimal control policy was learned by constraining the real part of the eigenvalues of the closed loop system Jacobian at the equilibrium to be negative. In the second approach, we used an MPC strategy and learned non-parameterized control inputs that minimize a quadratic cost,

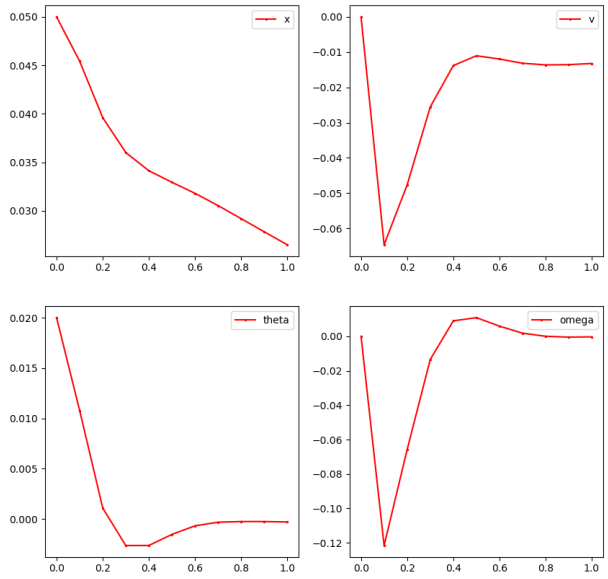

Fig. 6. Solution of the optimal control problem for a 1 sec time horizon.

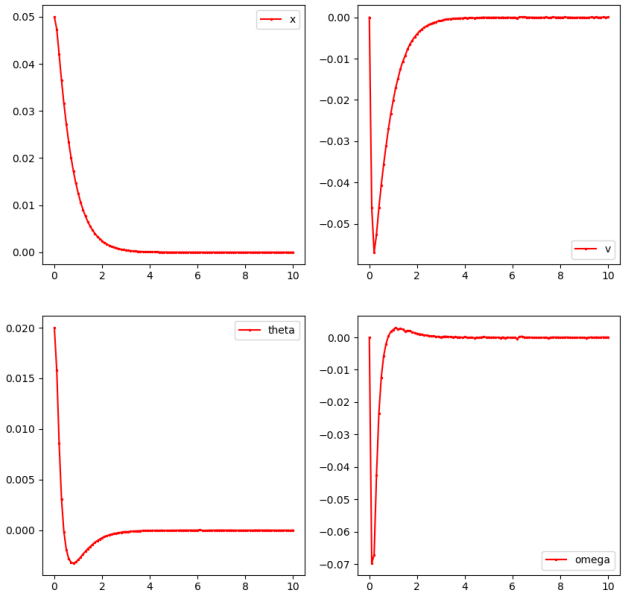

Fig. 7. Solution for the optimal control problem for a 10 sec time horizon.

starting once again from a sub-optimal stabilizing controller. In the third approach, we applied a "small steps" strategy, where we solved a sequence of quadratic optimal control problems with increasing time horizons, preventing gradient explosion. For all three approaches, we used automatic differentiation for computing and evaluating Jacobians and loss function gradients. Automatic differentiation is a useful tool for learning control policies. It can be directly applied on system models to generate linear approximations and compute stabilizing controllers. First order gradient algorithms (e.g., Adam) applied to large scale problems, together with automatic differentiation that computes loss function gradients, enable control policy design for nonlinear system (although only local solutions can be guaranteed without additional assumptions, like loss function convexity). Our experiments showed that offline design of control policies based on parameterized, state-dependent maps definitely benefits from the computational tools offered by DL platforms. This is due, in part, to the integration of ODE/DAE solvers. More research is needed to asses the feasibility of using DL platforms for real-time control design (e.g., MPC) in order to investigate the stability of the learning algorithms and the practical challenges of transferring learning models to physical platforms that implement the control policies.

References

[1] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.

[2] MathWorks Corporations. Inverted pendulum: System modeling. `http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling`.

[3] A. Paszke et al. Automatic differentiation in pytorch. 2017.

[4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[5] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. IEEE Press. Wiley, Hoboken, NJ, 2 edition, 2015.

[6] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice - A survey. *Automatica*, 25(3):335 – 348, 1989.

[7] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[8] H. K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[10] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.

[11] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1972.

[12] D. Maclaurin, D. Duvenaud, and R. P. Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.

[13] D. D. Nikolić. DAE Tools: equation-based object-oriented modelling, simulation and optimisation software. *PeerJ Computer Science*, 2:e54, April 2016.

[14] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 91–99, Cambridge, MA, USA, 2015. MIT Press.

[15] W. J. Rugh. *Linear system theory.* Prentice Hall,, Upper Saddle River, N.J., 2nd ed. edition, 1996.

[16] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

[17] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning. *CoRR*, abs/1808.01974, 2018.

[18] A. Fridman *et. al*. MIT autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *CoRR*, abs/1711.06976, 2017.

[19] K. M. Hermann *et al.* Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1693–1701, Cambridge, MA, USA, 2015. MIT Press.

[20] M. Abadi *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[21] F. Wang, X. Wu, G. M. Essertel, J. M. Decker, and T. Rompf. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *CoRR*, abs/1803.10228, 2018.