Learning Stable Adaptive Explicit Differentiable Predictive Control for Unknown Linear Systems

Abstract

We present differentiable predictive control (DPC), a method for learning constrained adaptive neural control policies and dynamical models of unknown linear systems. DPC presents an approximate data-driven solution approach to the explicit Model Predictive Control (MPC) problem as a scalable alternative to computationally expensive multiparametric programming solvers. DPC is formulated as a constrained deep learning problem whose architecture is inspired by the structure of classical MPC. The optimization of the neural control policy is based on automatic differentiation of the MPC-inspired loss function through a differentiable closed-loop system model. This novel solution approach can optimize adaptive neural control policies for time-varying references while obeying state and input constraints without the prior need of an MPC controller. We show that DPC can learn to stabilize constrained neural control policies for systems with unstable dynamics. Moreover, we provide sufficient conditions for asymptotic stability of generic closed-loop system dynamics with neural feedback policies. In simulation case studies, we assess the performance of the proposed DPC method in terms of reference tracking, robustness, and computational and memory footprints compared against classical model-based and data-driven control approaches. We demonstrate that DPC scales linearly with problem size, compared to exponential scalability of classical explicit MPC based on multiparametric programming.

1 Introduction

Many real-world systems of critical interest have unknown dynamics, uncertain and dynamic operating environments, and constrained operating regimes. This presents challenges to design efficient and robust control algorithms. Advanced control design requires expertise in applied mathematics, dynamic systems theory, computational methods, modeling, optimization, and operator assisted algorithmic tuning of control parameters. These requirements increase cost and reduce their applicability only to high value systems where marginal performance improvements lead to huge economic benefits.

Data-driven dynamics modeling and control policy learning show promise to “democratize” advanced control to systems with complex and partially characterized dynamics. However, pure data-driven methods typically suffer from poor-sampling efficiency, scale poorly with problem size, and exhibit slow convergence to optimal decisions [26]. Of special concern are lack of guarantees that black-box data-driven controllers will obey operational constraints and maintain safe operation.

Model Predictive Control (MPC) methods can offer optimal performance with systematic constraints handling. However, they depend on system dynamics models, which can be difficult or costly to obtain, while their solution is based on real-time optimization which might be computationally expensive for some applications. Explicit MPC [12,2] aims to overcome the problems associated with online optimization by pre-computing the control law offline for a given set of admissible parameters. However, obtaining explicit MPC policies via multiparametric programming is computationally prohibitive, limiting its applicability to small-scale systems. Addressing scalability limitations of explicit MPC, several
Learning Stable Adaptive Explicit Differentiable Predictive Control for Unknown Linear Systems

In this paper, we propose a deep learning-based method called Differentiable Predictive Control (DPC) for optimizing constrained explicit control policies and system dynamics models directly from measurements of the system without the need for supervision from an MPC controller. We report following contributions:

1. A general method for designing closed-loop system dynamics models as differentiable computational graphs with neural network components.
2. Optimization of explicit constrained neural control policies with MPC-inspired loss function by automatic differentiation of the closed-loop system model.
3. Sufficient conditions for global asymptotic stability of the closed-loop dynamics with neural feedback policies.
4. Demonstration of DPC’s capability of learning stabilizing neural control policies for unstable systems.
5. Comparison of DPC with LQR, LQI, MPC, and reinforcement learning (RL) algorithms.
   - DPC can systematically handle constraints and is more sample efficient than evaluated RL algorithms.
   - DPC has faster execution time than implicit MPC based on online optimization.
   - DPC scales better and requires less memory than explicit MPC based on multi-parametric programming.

1.1 Related Work

Adaptive and learning-based MPC In general, Learning-based MPC (LBMPC) methods [7] are based on learning the system dynamics model from data, and can be considered generalizations of classical adaptive MPC [6]. To make LBMPC tractable, the performance and safety tasks are decoupled by using reachability analysis [5, 65]. Variations include formulation of robust MPC with state-dependent uncertainty for data-driven linear models [71], or iterative model updates for linear systems with bounded uncertainties and robust guarantees [16]. For a comprehensive review of LBMPC approaches we refer the reader to a recent review [34] and references therein. A conceptual idea of backpropagating through the learned system model parametrized via convex neural networks [4] was investigated in [21]. While authors in [48] introduced a new recurrent neural model for learning latent dynamics for MPC. However, LBMPC approaches require online solution of the corresponding optimization problem. In contrast, in the proposed DPC methodology, the neural policy is learned offline and hence represents an explicit solution of the underlying constrained optimal control problem.

Explicit Model Predictive Control For a certain class of small scale MPC problems [27, 52], the solution can be pre-computed offline using multiparametric programming (mpP) [12, 46] to obtain a so-called explicit MPC (eMPC) policy [9, 72]. The benefits of eMPC are typically faster online computations, exact bounds on worst-case execution time, and simple and verifiable policy code, which makes it a suitable approach for embedded applications. However, eMPC suffers from the well-known curse of dimensionality which severely limits its practical applicability only to small scale systems with short prediction horizons [2]. In this work, we present a scalable alternative for obtaining explicit MPC policies for linear systems by employing principles of data-driven constrained differentiable programming.

Approximate MPC Another class of methods deals with learning MPC policies from simulation data using deep learning function approximations of the control policies [39, 80, 25, 64, 74], including constraint handling capabilities [42, 82, 20, 81]. Authors in [1] represent constrained optimization problems as implicit layers in deep neural networks, leading to the introduction of neural network policy architectures designed to handle pre-defined constraints [62, 3, 18]. However, all of these approaches fall into the category of imitation learning, which requires the training data to be generated by the supervisory controller. In contrast, the presented DPC method does not require the supervisory control policy nor system model beforehand.

Constrained neural architectures There are numerous challenges associated with solving constrained deep learning problems, including guarantees on convergence to stationary points and global minima, smart learning rates, min-max optimization, non-convex regularizers, or constraint satisfaction guarantees [78]. The paper [66] showed how constraints could be handled by modifications of the Conditional Gradients (CG) algorithm. Specific neural architectures can be designed to impose a certain class of hard constraints, such as linear operator constraints [50]. Authors in [41] demonstrated that using a log-barrier method for imposing inequality constraints could lead to improved accuracy, constraint satisfaction, and training stability. Penalty methods based on regularization terms in the loss function have become a popular choice for imposing inequality constraints on the outputs of deep neural network models [61, 36]. As
pointed out by \[50\], in practice, the existing methods for incorporating hard constraints rarely outperform their soft constraint counterparts despite their weak performance guarantees.

2 Background

2.1 Model Predictive Control

Model predictive control (MPC) is an optimal control strategy that calculates the control inputs trajectories by minimizing a given objective function over a finite prediction horizon with respect to the constraints on the system dynamics. We consider a reference tracking formulation of linear MPC, given as the following constrained optimization problem:

\[
\begin{align*}
\min_{u_0,\ldots,u_{N-1}} & \quad \sum_{k=0}^{N-1} \left( ||x_k - r_k||^2_Q + ||u_k||^2_R \right) \\
\text{s.t.} & \quad x_{k+1} = Ax_k + Bu_k, \quad k \in \mathbb{N}_0^{N-1} \quad (1a) \\
& \quad g(x_k) \leq 0, \quad k \in \mathbb{N}_0^{N-1} \quad (1b) \\
& \quad h(u_k) \leq 0, \quad k \in \mathbb{N}_0^{N-1} \quad (1c) \\
& \quad x_0 = x(t), \quad (1d)
\end{align*}
\]

where \( x_k \in \mathbb{R}^{n_x} \) is the system state, \( u_k \in \mathbb{R}^{n_u} \) is the control input at time \( k \), and \( \mathbb{N}_0 \) is a set of integers. The system is subject to state \((1a)\) and input \((1d)\) constraints. The objective function \((1a)\) is the weighted squared 2-norm i.e., \( ||a||^2_Q = a^TQa \), and penalizes the distance of the states from references, while minimizing the control effort.

The constrained optimization problem \((1)\) can be re-formulated using penalty functions \([15,13]\) as the following Lagrangian form:

\[
\begin{align*}
\min_{u_0,\ldots,u_{N-1}} & \quad \sum_{k=0}^{N-1} \left( \ell(x_k, r_k, u_k) + Q_x p_x(g(x_k)) + Q_u p_u(h(u_k)) \right) \\
\text{s.t.} & \quad x_{k+1} = Ax_k + Bu_k, \quad k \in \mathbb{N}_0^{N-1} \quad (2)
\end{align*}
\]

where \( \ell(\cdot) \) defines the MPC objective function \((1a)\), while \( Q_x \) and \( Q_u \) define weights of a penalty functions \( p_x \) and \( p_u \) associated with state \((1c)\) and input \((1d)\) constraints. In section 3.1, we show how to leverage the Lagrangian reformulation \((2)\) to construct a constrained deep learning problem to obtain an approximate solution of the corresponding parametric programming problem arising in explicit MPC \([10]\) described in the following section.

2.2 Explicit MPC via Multiparametric Programming

The real-time implementation of MPC typically requires an online solution of the corresponding constrained optimization problem \((1)\). This approach corresponds to the so-called implicit MPC, whose main limiting factor is higher online computational requirements that might be prohibitive for many real-world applications with limited computing resources. Explicit MPC \([42,10,14]\) represents an alternative approach based on offline solution of the corresponding multiparametric programming problem \([31,46]\) obtained by reformulation of the original MPC problem \((1)\). A generic form of the multi-parametric programming problem is given as:

\[
\begin{align*}
\min_{U} & \quad J(U, \xi) \quad (3a) \\
\text{s.t.} & \quad c(U, \xi) \leq 0, \quad (3b) \\
& \quad \xi \in \Xi \subseteq \mathbb{R}^n \quad (3c)
\end{align*}
\]

Where \( U \) represents a vector of optimized variables, and \( \xi \) is a vector of parameters, corresponding to a set of admissible state measurements, constraint bounds, and reference signals in the original problem \((1)\). The offline solution of the parametric problem \((3)\) yields an explicit state feedback control policy \( \hat{U} = \pi(\xi) \) which for linear constraints \( c(U, \xi) \) with quadratic objective \( J(U, \xi) \) with the piecewise-affine (PWA) form:

\[
\pi(\xi) = \alpha_\xi \xi + \beta_\xi, \quad \text{if} \ \xi \in \mathcal{R}_r \quad (4)
\]
where \( \alpha_r \in \mathbb{R}^{n_u \times n_c} \) and \( \beta_r \in \mathbb{R}^{n_u} \) specify local affine laws defined over polytopic regions \( \mathcal{R}_r \) partitioning the parametric space \( \Xi \). Thus the control policy \( \Pi \) represents a pre-computed solution of the original MPC problem \( (1) \).

Hence the main advantage of the explicit MPC against implicit MPC lies in the fast, optimization-free online evaluation of the PWA control policy \( \Pi \). Unfortunately, existing solutions \([22, 28, 31]\) of the parametric programming problem \( (3) \) scale geometrically with the number of constraints. Thus effectively limiting the applicability of explicit MPC only to small-scale problems with a severely limited number of decision variables and prediction horizons.

Targeting the scalability issues of explicit MPC, authors in the control community proposed approximate MPC \([80, 25, 64, 32, 82, 20, 81]\) whose solution is based on supervised learning of control policies imitating the original MPC. An interesting theoretical connection was made by authors in \([39]\) showing that every PWA control policy can be exactly represented by a deep ReLU network \([60]\). This means that the optimality of the neural control policy will depend only on the quality of training data and formulation of the learning problem. However, the remaining disadvantage of approximate MPC is its inherent dependency on the solution of the original MPC problem. The current work presents an alternative method for computing scalable explicit control policies for linear systems based on constrained deep learning which avoids need of a system model or guiding MPC controller.

## 3 Differentiable Predictive Control

We present a new perspective on the connection between parametric programming and deep learning. Leveraging this connection, we introduce a differentiable predictive control (DPC) method based on the reformulation of the explicit MPC problem as a constrained deep learning problem. DPC is based on the neural parametrization of the closed-loop dynamical system with two building blocks: i) differentiable system dynamics model, and ii) a deep learning reformulation of model predictive control policy. The conceptual illustration of the proposed DPC methodology is shown in Fig. \( 1 \) while the Algorithm \( 1 \) specifies individual steps of the proposed adaptive DPC method explained in the following sections. As a first step, the state-space model parameters are obtained from the system identification with eigenvalue constraints for stability guarantees. Next, we construct a differentiable closed-loop system model by combining the state-space model and neural control policy. The control policy is optimized via backpropagation of the MPC-inspired control loss with penalty constraints through the closed-loop system model. Finally, adaptive updates are obtained by joint policy optimization and system identification updating the parametrized closed-loop system model via online measurements of system response.

![Conceptual methodology of the proposed differentiable predictive control (DPC) of unknown linear systems.](image-url)

**Algorithm 1** Adaptive Differentiable Predictive Control.

1. Obtain linear state space model via system identification or by design. \( \triangleright \) Section \( 7.1 \)
2. Construct differentiable closed-loop system dynamics model with neural control policy. \( \triangleright \) Section \( 3.1 \)
3. Reformulate parametric programming problem \( (3) \) as constrained deep learning problem. \( \triangleright \) Section \( 3.1 \)
4. Data-driven offline optimization of constrained neural control policy by differentiating MPC loss function through the closed-loop dynamics. \( \triangleright \) Section \( 3.2 \)
5. Online deployment with adaptive system model and control policy updates. \( \triangleright \) Section \( 3.3 \)
6. Stability analysis of the learned dynamics. \( \triangleright \) Section \( 4 \)

### 3.1 Differentiable Predictive Control Problem Formulation

In this section, we show how to transform the Lagrangian MPC formulation \( (2) \) into the constrained deep learning problem representing a surrogate for the parametric programming problem \( (3) \). The proposed method called differentiable...
predictive control (DPC) is then given as the following parametric optimal control problem:

$$\begin{align}
\min_{W} & \sum_{k=0}^{N-1} (\ell(x_k, u_k, \xi_k) + Q_x p_x(g(x_k, \xi_k)) + Q_u p_u(h(u_k, \xi_k))) \\
\text{s.t.} & \quad x_{k+1} = Ax_k + Bu_k, \quad k \in \mathbb{N}^{N-1} \\
& \quad u_k = \pi_W(\xi_k) \\
& \quad \xi_k \in \Xi \subset \mathbb{R}^n
\end{align}$$ (5a)

Where $W$ are decision variables representing weights and biases of the neural control policy $\pi_W(\xi_k)$. The control parameters $W$ represent a design choice that captures all decision relevant signals of the control problem. For instance, we can define $\xi_k = \{x_k, r_k, s_k, \bar{x}_k, \bar{u}_k\}$ as a vector of state measurements, previews of the reference signals, and bounds for state and input constraints, respectively.

This reformulation (5) allows us to obtain approximate data-driven solution of the corresponding parametric programming problem (3). This is achieved by minimizing the objective function (5b) over a training dataset of control parameters $\xi_k$, synthetically sampled from a pre-defined distributions. The first term of the objective (5b) gives an performance metric, e.g. a reference tracking, while last two terms correspond to state and control action constraints penalties. One major difference between implicit MPC and DPC is that the solution of the MPC problem (1) returns an optimal sequence of the control actions $u_0, \ldots, u_{N-1}$, while solving the DPC problem (5) yields optimized parameters $W = \{H_1, \ldots H_L, b_1, \ldots b_L\}$ of the neural control policy $\pi_W : \mathbb{R}^m \rightarrow \mathbb{R}^n$ given as:

$$\begin{align}
\pi_W(\xi) &= H_L z_L + b_L \\
z_l &= \sigma(H_{l-1} z_{l-1} + b_{l-1}) \\
z_0 &= \xi
\end{align}$$ (6a)

Where $z_l$ are hidden states, $H_l$, and $b_l$ represent weights and biases of the $i$-th layer, respectively. The layer activation $\sigma : \mathbb{R}^{m_l} \rightarrow \mathbb{R}^{n_l}$ is given as the element-wise application of a univariate function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$.

### 3.1.1 Closed-loop system as neural network architecture

The core idea of DPC is based on parametrization of the differentiable closed-loop system model composed of system dynamics model (7) and neural control policy (6) as shown in Fig. 1. In a nominal case with $\xi_k = x_k$ we obtain a full state feedback formulation of the control problem (5), where the system model (5c) and control policy (5d) together define the dynamic of the closed-loop system given as:

$$x_{k+1} = Ax_k + B \pi_W(x_k)$$ (7)

To simulate $N$-step ahead closed-loop system dynamics, we use the recurrent encoding of the closed-loop system dynamics (7), this is equivalent with a single shooting formulation of MPC (27). The resulting structural equivalence of the classical implicit MPC with DPC is shown in Figure 2. Then similarly to MPC, in the closed-loop rollouts our DPC policy generates future control action trajectories over $N$-step prediction horizon given the feedback from the system dynamics model. A main advantage of having a differentiable closed-loop dynamics model is that it allows us to optimize its parameters using automatic differentiation (backpropagation through time (63)) with an MPC-inspired Lagrangian loss function (5b).

### 3.1.2 Modeling of inequality constraints in deep learning

For including the constraints into deep learning problem (5), we leverage classical penalty methods. In particular, we use ReLU units to model the violations of time-varying inequality constraints as follows:

$$\begin{align}
& x_k \leq x_k + s_k^x \equiv s_k^x = \text{ReLU}(-x_k + x_k), \\
& x_k - s_k^x \leq x_k \equiv s_k^x = \text{ReLU}(x_k - x_k).
\end{align}$$ (8a)

Where slack variables $s_k^x$, and $s_k^x$ represent the violations of the inequality constraints for inputs and states, respectively, and correspond to the penalty function terms $p_u(\cdot)$ and $p_x(\cdot)$ in the DPC objective (5b), with penalty weights $Q_x$ and $Q_u$.

### 3.2 Differentiable Predictive Control Policy Optimization

The proposed DPC method represents a novel way of learning approximate solutions of explicit MPC for linear systems based on a sampling of the parametric space and automatic differentiation of the MPC loss function (5b) through the neural closed-loop system dynamics model (7).
3.2.1 Forward pass DPC architecture

The forward pass of the DPC policy is defined in the Algorithm 2. Here, the neural control policy (6) is evaluated on line 3. The violations of the control constraints are computed on lines 4, 5, and 6 respectively. In line 7, the system model is integrated forward in time to simulate the evolution of the states. The state constraints are evaluated on lines 8, 9, and 10 respectively. And finally, the forward pass of the policy returns computed control actions, predicted states, and slacks representing constraints violations.

Algorithm 2 Forward pass of a single time step evaluation of DPC policy with closed-loop dynamics model.

1: function DPC(ξk)
2:   ξk = {xk, rk, xk, xk, uk, uk}
3:   uk = πW(ξk)
4:   s^u_k = ReLU(−uk + uk)
5:   s^u_k = ReLU(uk − uk)
6:   s^u_k = s^u_k + s^u_k
7:   x_k+1 = Ax_k + Bu_k
8:   s^x_k = ReLU(−x_k + x_k)
9:   s^x_k = ReLU(x_k − x_k)
10:  s^x_k = s^x_k + s^x_k
11:  return uk, x_k+1, s^u_k, s^x_k
12: end function

3.2.2 DPC dataset

To populate the DPC training dataset Ξ = {ξ1, . . . , ξN} we randomly sample states, reference, and constraints trajectories over an N-step horizon window. In our case, at each time step we have ξk = {xk, rk, xk, xk, uk, uk}. An alternative choice of the training data can be obtained from typical trajectories of the controlled system, e.g., from measurements or specified by a domain expert.
3.2.3 DPC loss function

The trainable parameters $W$ of the DPC are optimized w.r.t. the following weighted multi-objective loss function:

$$L_{\text{MSE}}(\Xi|\text{DPC}(\xi_k)) = \frac{1}{N} \sum_{k=0}^{N-1} (Q_r ||r_{k+1} - x_{k+1}||_2^2 + Q_u ||u_k||_2^2 + \lambda ||s_k^x||_2^2 + \mu ||s_k^u||_2^2) + \frac{1}{N} \sum_{k=0}^{N-1} Q_r ||r_{k+1} - x_{k+1}||_2^2 + Q_u ||u_k||_2^2 + \lambda ||s_k^x||_2^2 + \mu ||s_k^u||_2^2).$$

(9)

Where, the first term stands for the reference tracking loss, the second term penalizes the square of the control actions, while the last two terms penalize the constraints violations. The relative importance of the individual terms is balanced by weight factors $Q_r$, $Q_u$, $\lambda$, and $\mu$.

3.2.4 DPC policy optimization

Now combining individual components together, the DPC policy optimization algorithm is summarized in Algorithm 3. The presented procedure represents a generic approach for approximate data-driven solution of parametric optimal control problem generating optimized constrained control policies.

Algorithm 3 DPC policy optimization.

1: input training dataset $\Xi = \{\xi_1, \ldots, \xi_N\}$
2: input learned or known system dynamics model $f_{\text{SM}}$
3: input DPC policy architecture as given in Algorithm 2
4: input optimizer $O$
5: train $\pi_W$ to optimize DPC loss (9) using optimizer $O$
6: return optimized policy $\pi_W$

3.3 Adaptive DPC Policy Optimization

The objective of the adaptive DPC approach is to enable online updates of the learned system dynamics model and constrained control policy.

3.3.1 Forward pass adaptive DPC architecture

The modified forward pass architecture is defined by Algorithm 4, where the the control policy (line 4) is learned together with system model dynamics (line 9). Training of both the control policy and the dynamics model now requires control dataset $\Xi$ alongside the system identification $\Xi^{\text{ID}} = \{X^{\text{ID}}, U^{\text{ID}}\}$ dataset. Examples of state trajectories $X^{\text{ID}}$, as defined in section 7.1, are used as targets for system identification process. On the other hand, randomly sampled states $X$ are used for simulations of the internal model on line 8 to guide the control policy update. Analogously, $U^{\text{ID}}$, and $U$ are control input trajectories associated with the system model and policy learning, respectively. To promote learning smooth state trajectories, line 10 defines the state residual to be minimized during the training. The constraints penalties in Algorithm 4 are used analogously to Algorithm 2.

3.3.2 Adaptive DPC loss function

The DPC policy learning loss function (9) is extended with a system identification term penalizing model deviations from measured states with weight $Q_{ID}$, and with smooth state trajectory penalty term weighted by $Q_{\Delta x}$ as follows:

$$L_{\text{MSE}}(\Xi, \Xi^{\text{ID}}|\text{aDPC}(\xi_k, \xi_k^{\text{ID}})) =$$

$$\frac{1}{N} \sum_{k=0}^{N-1} (Q_r ||r_{k+1} - x_{k+1}||_2^2 + Q_u ||u_k||_2^2 + \lambda ||s_k^x||_2^2 + \mu ||s_k^u||_2^2 + Q_{ID} ||x_k^{\text{ID}} - x_k||_2^2 + Q_{\Delta x} ||\Delta x_k||_2^2).$$

(10)

3.3.3 Adaptive DPC policy optimization

To jointly optimize the model and policy parameters $A, B, W$, the gradient updates are performed using the aggregate loss (10) associated with the parallel system identification and control policy trajectories. The modified adaptive DPC policy optimization algorithm is then given in Algorithm 5.
An alternative which can be used in conjunction with terminal constraints is the use of terminal penalties $F$. Adaptive DPC policy optimization.

Algorithm 4 Forward pass of a single time step evaluation of adaptive DPC policy with system model updates.

1: function $\Lambda_{\text{DPC}}(\xi_k, \xi_{\text{ID}}^k)$
2: $\xi_k = \{x_k, r_k, \overline{x}_k, \underline{x}_k, u_k, \underline{u}_k\}$
3: $\xi_{\text{ID}}^k = \{x_{\text{ID}}^k, u_{\text{ID}}^k\}$
4: $u_k = \pi_W(\xi_k)$
5: $s_k^a = \text{ReLU}(-u_k + \underline{u}_k)$
6: $s_k^b = \text{ReLU}(u_k - \overline{u}_k)$
7: $s_k^v = s_k^a + s_k^b$
8: $x_{k+1} = Ax_k + Bu_k$
9: $\tilde{x}_{\text{ID}}^k = \text{ReLU}(Ax_k + Bu_{\text{ID}}^k)$
10: $\Delta x_{\text{ID}}^k = \tilde{x}_{\text{ID}}^{k+1} - x_{\text{ID}}^k$
11: $s_k^a = \text{ReLU}(-\tilde{x}_{k+1} + \overline{x}_k)$
12: $s_k^b = \text{ReLU}(\tilde{x}_{k+1} - \underline{x}_k)$
13: $s_k^v = s_k^a + s_k^b$
14: return $u_k, \tilde{x}_{k+1}, s_k^v, s_k^b, \tilde{x}_{\text{ID}}^k, \Delta x_{\text{ID}}^k$
15: end function

Algorithm 5 Adaptive DPC policy optimization.

1: input training datasets for control $\Xi$
2: input training datasets for system identification $\Xi_{\text{ID}}$
3: input adaptive DPC architecture as given in Algorithm 4
4: input optimizer $O$
5: train system dynamics model $f_{\text{SR}}$ and policy $\pi_W$ to optimize adaptive DPC loss (10) using optimizer $O$
6: return trained model $f_{\text{SR}}$ and policy $\pi_W$

Remark. To avoid catastrophic forgetting during online learning using a small data samples, we can update only a small subset of the policy and model parameters while fixing the rest. Alternatively, we can keep the nominal model and policy fixed, and learn only additive error terms parametrized by smaller neural networks. In this way the policy and the system models can adapt to new data while preserving most of the learned dynamics from the offline learning phase.

3.4 Stability of Differentiable Predictive Control

Closed-loop system stability guarantees is a premiere feature of MPC. In this section, we elaborate on the stability of the DPC method by leveraging the structural equivalence with MPC, this allows us to directly apply stability analysis and guarantees developed for constrained MPC in the context of DPC. Most of the stability analysis methods of MPC are based on enforcing MPC’s value function to be a Lyapunov function [57]. For further details on the Lyapunov stability of MPC see e.g. [57] and references therein. The second class of methods deals with enforcing contraction constraints on normed state variables [11]. We now show how to use these stability principles of MPC to guarantee the stability of DPC via modifying the problem formulation [5] with the loss function [9].

3.4.1 Terminal Constraints

This method is based on including additional terminal constraints $x_N \in \mathcal{X}_f$ to the DPC problem [5]. Stability guarantees [57] exist for cases with terminal equality $\mathcal{X}_f = \{0\}$ or inequality given as a set $\mathcal{X}_f \in \mathbb{R}^n$ containing the origin. In DPC we treat terminal constraints as any other constraints and enforce them via penalty methods [8].

3.4.2 Terminal Penalties

An alternative which can be used in conjunction with terminal constraints is the use of terminal penalties $F(x_N)$ as extra terms in the DPC loss function [9]. Authors in the MPC literature have long before proposed the use of quadratic terminal penalties $F(x_N) = \frac{1}{2}x_N P_f x_N$, where $P_f$ can be obtained e.g. by solving the Riccati equation, or as a value function of the stabilizing controller [57]. More recent approaches enforce $F(x_N)$ to be a control Lyapunov function [38] [77] [84].
3.4.3 Contraction Constraints

This idea is based on enforcing the contraction of state variables w.r.t. some norm \([11]\):

\[
||x_{k+1}||_p \leq \alpha ||x_k||_p, \quad \alpha < 1
\]  

(11)

Now it is straightforward to see that the contraction is enforced by employing regularization and design methods introduced in the following section 4.3 based on the Theorem 2. Alternatively we can penalize the constraint (11) directly in the DPC loss function (9) via penalty methods.

3.4.4 Learning Lyapunov Functions

In addition to the methods inspired by MPC literature, differentiability of DPC allows us to employ newly proposed data-driven methods based on learning Lyapunov function candidates for stable dynamics models and control policies as part of the neural architecture [45, 22].

4 Stability of Neural Feedback Systems

In this section we provide sufficient asymptotic stability conditions for generic closed-loop systems with a linear system model controlled by a full state feedback neural policy (7). We sketch the proof in the following steps of Algorithm 6.

Algorithm 6 Sketch of the proof of global asymptotic stability conditions of neural feedback system in the form (7).

1: Show equivalence of (7) with linear parameter varying systems (LPV) via Lemma 1. \(\triangleright\) Section 4.1
2: Revise contraction conditions of LPV systems as guarantees of global asymptotic stability via Lemma 2. \(\triangleright\) Section 4.2
3: In Theorem 2 leverage the equivalence of (7) with LPV to prove the sufficiency of contractivity of all linear regions to guarantee the global asymptotic stability of the neural feedback system (7). \(\triangleright\) Section 4.2

From these results, we propose practically oriented regularization methods for enforcing stability of neural policies trained with data-driven methods such as DPC Algorithm 3.

4.1 Neural Closed-loop Dynamics as LPV System

The closed-loop dynamics model (7) with full state feedback policy can be equivalently reformulated as follows:

\[
x_{k+1} = (A + BK_{x_k})x_k + Bb_{x_k}
\]  

(12)

Here \(K_{x}, x_{k} + b_{x_{k}}\) represents an exact point-wise linearization of the neural policy \(\pi_{W}\) based on the following Lemma 1. Any deep feedforward neural network \(\pi_{W} : \mathbb{R}^m \rightarrow \mathbb{R}^n\) defined as (6) with arbitrary activation function \(\sigma\) and zero bias, i.e. \(b_i = 0, \forall i \in \mathbb{N}_1^L\), satisfies the following:

\[
\pi_{W}(x) = K_{x}x
\]  

(13)

In the case of non-zero bias in (6) the following holds:

\[
\pi_{W}(x) = K_{x}x + b_{x}
\]  

(14)

Where \(K_{x} : \mathbb{R}^m \rightarrow \mathbb{R}^n\) is a parameter varying matrix, and \(b_{x} \in \mathbb{R}^n\) is a varying vector parametrized by \(x\), respectively.

Proof. To prove the Lemma 1 we start with the observation that activation functions at \(l\)-th layer \(\sigma : \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_l}\) can be arranged in a square matrix form as follows:

\[
\sigma(z) = \begin{bmatrix}
\frac{\sigma(z_1)}{z_1} \\
\vdots \\
\frac{\sigma(z_n)}{z_n}
\end{bmatrix}
= \begin{bmatrix}
\frac{\sigma(z_1)}{z_1} & \cdots & \frac{\sigma(z_n)}{z_n}
\end{bmatrix}
\end{bmatrix}
\]  

(15)

Where \(z\) represents vector of hidden states, and diagonal parameter varying matrix \(\Lambda_{z}\) represents the operation of activation function \(\sigma\) on \(z\).
Now for the case without bias, the reformulation (15) allows us to equivalently write the layer dynamics $z_{l+1} = \sigma(H_lz_l)$ of a bias-free deep neural network (6) as:

$$z_{l+1} = \Lambda z H_l z_l \tag{16}$$

By composition of layers (16) we obtain:

$$K_{x}x = H_L \Lambda x_L H_{L-1} \ldots \Lambda x_1 H_0 x$$

Where $K_{x}x$ now represents an exact point-wise linearization of deep neural network $\pi_W(x)$. Similarly for the case with bias, the layer dynamics $z_{l+1} = \sigma(H_lz_l + b_l)$ of the network (6) can be cast as:

$$z_{l+1} = \Lambda z H_l z_l + \Lambda z b_l \tag{18}$$

Now form (14) is obtained by straightforward composition of (18).

Notice that $\Lambda z$ is undefined at $z = 0$. We address this by considering the following assumption:

**Assumption 1.** The activation function $\sigma(z)$ is Lipschitz continuous, i.e. $\sigma(0) = 0$, and for the sake of analysis, for the point $z = 0$ we define $\sigma(0) = 0$.

**Remark.** Conveniently, many activation functions used in practice are Lipschitz continuous, examples include ReLU, LeakyReLU, PReLU, ELU, or tanh, and thus can be used in conjunction with the Assumption 1.

The reformulation of a deep neural network $\pi_W(x)$ into parameter varying linear $K_{x}x$ (17) now allows us to apply stability analysis methods developed for discrete-time parameter varying linear systems [49].

### 4.2 Stability of Neural Closed-loop Dynamics

Before we start with the stability analysis of (12) we pose following assumptions and definitions.

**Assumption 2.** The linear system dynamics model $x_{k+1} = Ax_k + Bu_k$ is controllable.

**Assumption 3.** We want to control the closed-loop system towards stable equilibrium $x_{tf}$ in finite time $t_f$.

The extension to the reference tracking problem can be achieved by straightforward coordinate transformation.

**Definition 4.1.** Induced operator norm of $A \in \mathbb{R}^{n \times m}$ is given as:

$$||A||_p = \max_{x \neq 0} \frac{||Ax||_p}{||x||_p} = \max_{||x||_p = 1} ||Ax||_p, \forall x \in \mathcal{X},$$

where $\mathcal{X}$ is a normed vector space, and $|| \cdot ||_p : \mathbb{R}^n \rightarrow \mathbb{R}$ represents the vector $p$-norm inducing the operator norm $||A||_p : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}$.

**Definition 4.2.** Induced $p$-norm $|| \cdot ||_p : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}$ is called submultiplicative if it satisfies [55]:

$$||AB||_p \leq ||A||_p ||B||_p.$$  \(20\)

**Theorem 1.** [75] Let us have a vector norm $|| \cdot ||_p : \mathbb{R}^n \rightarrow \mathbb{R}$ defined for all $n$ with corresponding induced operator norm defined as (19), then the submultiplicativity of the operator norm (20) is satisfied for any $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$.

Now we exploit the fact, that using Lemma 1 the closed-loop system dynamics (12) without bias can be represented as a linear parameter varying (LPV) system.

**Definition 4.3.** Discrete-time linear parameter varying (LPV) system is given as:

$$x_{k+1} = \Phi_k x_k \tag{21}$$

Now assuming the system dynamics matrix $\Phi_k = (A + BK_{x_k})$, the form (21) allows us to leverage known stability conditions for the LPV systems given in Lemma 2.

**Lemma 2.** [8] Let $\Phi_k \in \Phi^l$. Then the LPV system (21) is globally asymptotically stable if and only if there exists a finite number $M$ such that:

$$|| \prod_{k=1}^{M} \Phi_k ||_p < 1, \forall \Phi_k \in \Phi^l, k \in \mathbb{N}_1^M.$$

Where the interval matrix $\Phi^l$ is given as follows [8]:

$$\Phi^l = \{A^0 \in \mathbb{R}^{n \times n} | A^0 = A_0 + \sum_{i=1}^{p} \lambda_i A_i, \lambda_i = [\Delta_i, \bar{\Delta}_i], i \in \mathbb{N}_1^p, A_0, A_i \in \mathbb{R}^{n \times n} \} \tag{23}$$

...
For the proof and the verification algorithm see [8]. Now we formulate the main result of this section.

*Theorem 2.* A closed-loop system (7) with linear system dynamics and full state feedback policy parametrized by neural network \( \pi_W(x) \) [6] with piece-wise linear activation functions (e.g. ReLU) is globally stable if:

\[
\|A + BK_{x_k \in \mathcal{R}_i}\|_p < 1, \forall i \in \mathbb{N}_1^{n_R}
\]  

(24)

where \( n_R \) represents the total number of distinct linear regions \( \mathcal{R}_i \) of a deep neural network [6].

**Proof.** The proof is based on Lemma 2 and on the LPV form (21) of the closed-loop system (12) without bias. First we need to show the equivalence of the LPV system (21) with non-zero equilibrium holds also for affine maps, thus sufficiently satisfying the contraction condition to hold it will require \( \Phi \). Now we formulate the main result of this section.

\[
\Phi^{NN} = \{A^o \in \mathbb{R}^{n \times n}|A^o = A + \sum_{i}^{n_R} \lambda_i BK_{x_k \in \mathcal{R}_i}, \lambda_i \in \{0, 1\}, i \in \mathbb{N}_1^{n_R}, A, BK_{x_k \in \mathcal{R}_i} \in \mathbb{R}^{n \times n}\}
\]  

(25)

Where \( \Phi^{NN} \subset \Phi^f \). Thus we demonstrated that the closed-loop system with neural feedback policy (12) is a set with finite number of elements. This is true if the neural network policy \( \Phi^{NN} \) is a set with finite number of distinct linear regions \( \mathcal{R}_i \) with unique linear maps \( K_{x_k \in \mathcal{R}_i} \), which is known to be true for networks with piece-wise linear activation functions (e.g., ReLU) [29, 59].

The next part of the proof is to guarantee that there exist a finite number \( M \) which satisfies the Lemma 2. For this condition to hold it will require \( \Phi^f \) to be a set with finite number of elements. This is true if the neural network policy [9] has a finite number of distinct linear regions \( \mathcal{R}_i \) with unique linear maps \( K_{x_k \in \mathcal{R}_i} \), to be equal to the state transition matrix of the open-loop system \( A \). The proof is based on Lemma 2 and on the LPV form (21) of the closed-loop system (12) without bias. The contraction condition to hold it will require \( \Phi^f \) to be a set with finite number of elements. This is true if the neural network policy [9] has a finite number of distinct linear regions \( \mathcal{R}_i \) with unique linear maps \( K_{x_k \in \mathcal{R}_i} \), which is known to be true for networks with piece-wise linear activation functions (e.g., ReLU) [29, 59].

The final step of the proof is to show that arbitrary sequence of the linear regions is a contractive mapping satisfying Lemma 2. Here we leverage the submultiplicativity of induced \( p \)-norms (20) imposed on linear regions of the closed-loop system (12) given as:

\[
\|\Phi_1 \cdots \Phi_n\|_p \leq \|\Phi_1\|_p \cdots \|\Phi_n\|_p < 1
\]  

(26)

Now it is straightforward to see the sufficiency of \( \|\Phi_n\|_p < 1, \forall i \in \mathbb{N}_1^{n_R} \) to satisfy \( \|\Phi_1 \cdots \Phi_n\|_p < 1 \) in Lemma 2. This corresponds to having each linear region of the closed-loop system (12) be a contractive mapping \( \|\Phi_i\|_p < 1 \), yielding:

\[
\|(A + BK_{x_k})\|_p < 1, \forall x_k \in R_i, i \in \mathbb{N}_1^{n_R}
\]  

(27)

The contraction condition (27) with non-zero equilibrium holds also for affine maps, thus sufficiently satisfying the case with neural control policy with non-zero bias terms [14]. For the proof of the contraction condition for affine maps see [44].

4.3 **Enforcing Stability of Neural Closed-loop Dynamics**

Based on Theorem 2 this section presents three practical approaches for imposing stability of neural feedback policies obtained via learning-based methods such as DPC Algorithm 3.

The most straightforward method is to include the penalty on the condition (24) as an extra term in the loss function:

\[
\mathcal{L}_{\text{stable}} = \frac{1}{N} \sum_{k=0}^{N-1} \max \left(1, \|A + BK_{x_k}\|_p\right)
\]  

(28)

Second approach leverages subadditivity of induced \( p \)-norms imposing the upper bound on the stability condition (24) as:

\[
\|A + BK_{x_k}\|_p \leq \|A\|_p + \|BK_{x_k}\|_p < 1,
\]  

(29)

By separating autonomous dynamics from the feedback, this approach allows us to leverage methods for constraining the eigenvalues of neural networks by design without the need for extra regularization terms. This could be achieved for instance by using singular value decomposed (SVD) weights [79], Perron-Frobenius weights [73], or Gerschoring disc
Where all constraints are implemented using penalties (8) with weights.

As a benchmark we synthesize an explicit MPC policy solved with classical multi-parametric programming solver.

Would guarantee closed-loop stability only for systems with already stable system dynamics matrix \( A \).

This section presents experimental results and analysis for two case studies demonstrating the capabilities of DPC.

The advantage of this approach is that we could leverage the design methods for eigenvalue constraints of deep neural nets. However this comes with the cost of loosing the guarantees on enforcing the closed-loop stability due to the lower bound condition. To alleviate this disadvantage, we could introduce a safety margins tightening the lower bound in (31), or simply enforcing \( \| BK \|_X > \| A \|_p \).

Finally, we can use the condition of Theorem 2 as a practical stability assessment of closed-loop systems with neural policies (12) by sampling the state space after training.

\[
\begin{align*}
\| (A + BK) \|_p &< 1, \quad \forall \{ x_k \in \mathcal{X} \} \\
\| (A - BK) \|_p &< 1, \quad \forall \{ x_k \in \mathcal{X} \}
\end{align*}
\]

Where \( \mathcal{X} \) is a set of all admissible values of \( x_k \), while \( \mathcal{X} \) is a set of \( m \) samples from \( \mathcal{X} \). Then with sufficiently large \( m \) the condition (32) represents a reasonable proxy for the stability condition. More sophisticated sampling approaches with guarantees on coverage of distinct linear regions of the neural feedback system (12) could be inspired by works [59, 69, 29] on evaluating the number of linear regions of the neural networks. We leave this line of research for future work.

5 Numerical Case Studies

This section presents experimental results and analysis for two case studies demonstrating the capabilities of DPC to learn constrained control policies: i) stabilizing an unstable linear system, and ii) controlling an unknown linear system subject to constraints and uncertainties given observations of its input output dynamics. The open-source code implementation associated with the experiments presented in this section is available at [https://github.com/pnnl/deps_arXiv2020](https://github.com/pnnl/deps_arXiv2020).

5.1 Example 1: Learning Stabilizing Neural Control Policy

Here we demonstrate the capabilities of DPC to learn a stabilizing neural feedback policy for an unstable system with known dynamics. We control the unstable double integrator:

\[
x_{k+1} = \begin{bmatrix} 1.2 & 1.0 \\ 0.0 & 1.0 \end{bmatrix} x_k + \begin{bmatrix} 1.0 \\ 0.5 \end{bmatrix} u_k
\]

Subject to state and input constraints given as \( x_k \in \mathcal{X} := \{ x \mid -10 \leq x^1 \leq 10, \; i \in \mathbb{Z}_0 \} \), and \( u_k \in \mathcal{U} := \{ u \mid -1 \leq u \leq 1 \} \), respectively. For control we consider the following objective.

\[
\mathcal{L}_{\text{MPC}} = \sum_{k=0}^{N-1} \left( \| x_k \|_Q^{2} + \| u_k \|_Q^{2} \right)
\]

As a benchmark we synthesize an explicit MPC policy solved with classical multi-parametric programming solver using MPT3 toolbox [31] with prediction horizon of \( N = 10 \), and weights \( Q_x = 1 \), \( Q_u = 1 \), which surface is shown on the left in Fig. 5.

In the case of DPC, we learn a full state feedback neural policy \( u_k = \pi_W(x_k) \) in the closed-loop system (1) via Algorithm 3, which surface is shown on the right in Fig. 5. A side by side comparison in Fig. 5 shows almost identical control surfaces of compared policies. For the DPC policy training we used the MPC loss function (34) subject to state \( x_k \in \mathcal{X} \), input \( u_k \in \mathcal{U} \), as well as terminal set constraints \( x_N \in \mathcal{X}_f := \{ x_k \mid -0.1 \leq x^1_k \leq 0.1, \; i \in \mathbb{Z}_0 \} \). All constraints are implemented using penalties (8) with weights \( Q_X = 10 \), \( Q_U = 20 \), \( Q_{X_f} = 1 \). For the main...
5.2 Example 2: Control of Uncertain Linear System

In this example we consider the following uncertain parameter varying linear system:

\[ x_{k+1} = A(v_k)x_k + Bu_k + Ed_k + w_k, \]  

(35)
The model (35) in our case study represents thermal dynamics of a building [24]. The states $x_k \in \mathbb{R}^4$ represent temperatures, control input $u \in \mathbb{R}$ represents heat flow, and $d_k \in \mathbb{R}^3$ represent measured disturbances, respectively. We run the simulation of the ground truth model (35) for a period of one month, with 5 minute sampling intervals giving a total of 288 samples per day and 2016 samples per week. The training, validation, and test sets are the 2nd, 3rd, and 4th weeks of simulation. To assess the robustness of investigated control policies we consider unknown parametric $v_k$ and additive $w_k$ uncertainties, where $v_k \in \mathcal{N}(\mu_v, \sigma_v)$, $w_k \in \mathcal{N}(\mu_w, \sigma_w)$ with $\mu_v = 0$, $\sigma_v = 0.01$, $\mu_w = 0$, and $\sigma_w = 0.1$.

The disturbance trajectories $D$ are visualized at the bottom of Figure 8. The reference trajectory $R$ for closed-loop control task is generated as a static sine wave in range $18^\circ C$ to $22^\circ C$ and period of one day as shown at the top of Figure 8. However, for learning the DPC policy (9), the trajectory $R$ is uniformly sampled in the realistic operative range of $15^\circ C$ to $25^\circ C$ to increase the generalization. The simulated state trajectories, $X$, are uniformly sampled state distribution in realistic range $0^\circ C$ to $25^\circ C$ to robustify the learned control policy. The control inputs trajectory $U$ is generated by the control policies.

In general, the true parameters $A$, $B$, and $E$ of the model (35) are unknown. However, we assume that the transition matrix $\tilde{A}$ has stable eigenvalues, which is motivated by the dissipativeness of the thermal dynamics models. To enforce this property, we use the eigenvalue constraints given by (41) in the Appendix 7.1. Further we impose penalty constraints on the state trajectories, which are bounded to remain within a physically meaningful range.

### 5.2.1 Constrained Control with Known System Model

We compare the proposed DPC policy against two classical control methods (LQR, LQI), three formulation variants of MPC (nominal, robust, and stochastic), and three deep RL algorithms (PPO2, A2C, ACKTR), respectively. The details on control design of each method are elaborated in the Appendix 7.3. In this scenario, all of the model-based methods, including DPC, are designed with known nominal matrices of the system (35).

We evaluate each investigated method on reference tracking MSE: $\frac{1}{T} \sum_{k=1}^{T} || r_k - \tilde{x}_k ||_2^2$, energy minimization given as mean absolute (MA) value of the control signal: $\frac{1}{T} \sum_{k=1}^{T} |u_k|$, and state constraints violations evaluated as MA value of the slack variables: $\frac{1}{T} \sum_{k=1}^{T} |s_k^2|$. Table 1 shows the corresponding closed-loop control performance of the investigated methods evaluated with various degrees of uncertainties $v_k$ and $w_k$ acting on the simulation model. Figure 6 and Figure 7 compare visually the control performance of nominal simulations without uncertainties (second column of Table 1), and simulations with both parametric and additive uncertainties (fifth column of Table 1), respectively.

![Performance Comparison Nominal Simulations](image)

Figure 6: Control performance comparison in simulations without uncertainties.

The results from Table 1 demonstrate robust performance of the proposed DPC in the reference tracking and constraints satisfaction against all compared methods while paying a cost in terms of increased energy use. The zero values of the constraints violations (MA con.) demonstrate that the learned DPC policy is capable of 100% constraints satisfaction also in the presence of parametric and additive uncertainties, while simultaneously optimizing the reference tracking objective. These results indicate that the proposed state-space sampling-based method with differentiable closed-loop system model and penalty constraints is a competitive alternative to constraints tightening-based robust MPC and stochastic MPC based on a sampling of additive uncertainties. An interesting observation is that while using the same objective weights as all MPC variants, the proposed DPC method favors the constraints satisfaction and tracking performance before the energy use minimization. This can be explained by the state space and reference trajectory sampling stimulating the policy to learn robust behavior across a variety of initial conditions and reference signals.
Table 1: Control performance on reference tracking, energy use, and constraints violations evaluated on closed-loop simulations with various degrees of uncertainties $v_k$ and $w_k$. Evaluated controllers were learned/designed with the ground truth model.

| Test set | No unc. | $w_k$ | $v_k$ | $w_k$ & $v_k$ |
|----------|---------|-------|-------|--------------|
| **Proposed DPC** | | | | |
| MSE ref. | 1.244 | 1.470 | 1.991 | 2.355 |
| MA ene. | 1111 | 1092 | 1177 | 1139 |
| MA con. | **0.000** | **0.000** | **0.000** | **0.000** |
| **Nominal MPC** | | | | |
| MSE ref. | 1.398 | 1.785 | 3.343 | 3.711 |
| MA ene. | **897** | **847** | 917 | 866 |
| MA con. | **0.000** | **0.000** | 1.002 | 1.106 |
| **Robust MPC** | | | | |
| MSE ref. | 1.405 | 1.640 | 2.727 | 2.839 |
| MA ene. | 899 | 892 | **910** | **836** |
| MA con. | **0.000** | **0.000** | 0.007 | 0.066 |
| **Stochastic MPC** | | | | |
| MSE ref. | 1.398 | **1.422** | 2.238 | 3.579 |
| MA ene. | **897** | 995 | 977 | 856 |
| MA con. | **0.000** | **0.000** | 0.476 | 0.572 |
| **LQR** | | | | |
| MSE ref. | 1.954 | 2.544 | 3.399 | 4.957 |
| MA ene. | 899 | 861 | 920 | 676 |
| MA con. | 1.893 | 3.491 | 5.346 | 5.059 |
| **LQI** | | | | |
| MSE ref. | 16.978 | 19.032 | 24.717 | 27.885 |
| MA ene. | 531 | 530 | 528 | 526 |
| MA con. | 2.525 | 2.718 | 3.230 | 3.546 |
| **PPO2** | | | | |
| MSE ref. | 10.682 | 10.371 | 12.559 | 14.682 |
| MA ene. | 732 | 729 | 736 | 731 |
| MA con. | 1.608 | 1.536 | 1.690 | 1.883 |
| **ACKTR** | | | | |
| MSE ref. | 9.556 | 10.264 | 11.412 | 15.473 |
| MA ene. | 1496 | 1524 | 1513 | 1510 |
| MA con. | 0.557 | 0.487 | 0.585 | 0.768 |

As expected, the performance of the LQR and LQI on the given control task is unsatisfactory, as these classical control approaches are not designed to handle control problems with inequality constraints. The investigated deep RL algorithms are able to outperform classical control methods in terms of constraints satisfaction due to the rewards augmented with constraints penalties. However, all deep RL algorithms perform significantly worse in terms of reference tracking and constraints satisfaction compared to the proposed DPC policy and three variants of MPC. Given greater opportunity to explore the state space beyond the 30,000 provided training episodes, the reinforcement learning algorithms could potentially perform better on the provided metrics as deep RL algorithms typically require a large number of samples to converge to a near-optimal solution. However, provided the same number of gradient updates, DPC policy optimization demonstrates significantly better sampling efficiency against investigated RL methods.
Learning Stable Adaptive Explicit Differentiable Predictive Control for Unknown Linear Systems

A PREPRINT

Figure 7: Control performance comparison in simulations with parametric $v_k$ and additive $w_k$ uncertainties.

Table 2: Closed-loop control performance of adaptive DPC on model prediction, reference tracking, energy use, and constraints violations of simultaneous model and policy learning evaluated on closed-loop simulations with various degrees of uncertainties $v_k$ and $w_k$.

| Test set       | No unc. | $w_k$  | $v_k$  | $w_k \& v_k$ |
|----------------|---------|--------|--------|--------------|
| MSE mod.       | 0.344   | 0.485  | 1.325  | 1.781        |
| MSE ref.       | 0.952   | 1.126  | 2.263  | 2.932        |
| MA ene.        | 1102    | 1139   | 1134   | 1109         |
| MA con.        | 0.000   | 0.002  | 0.014  | 0.053        |

5.2.2 Constrained Adaptive Control with Unknown System Model

In this scenario, the system dynamics matrices of the controlled system (35) are considered unknown and must be learned by DPC from data. For the simultaneous system identification and constrained control learning task we train the parametrized closed-loop dynamics model as defined in algorithm 4. To demonstrate the adaptive capabilities we apply online updates of DPC policy by optimizing the objective (10) over 10 epochs at each time step during the test set.

Table 2 shows the closed-loop control performance of the adaptive DPC policy evaluated on the modeling MSE (MSE mod.), reference tracking MSE (MSE ref.), energy use (MA ene.), and constraints violation (MA ene., MA con.). Interestingly, the performance in terms of the constraint satisfaction is comparable to robust MPC designed with the ground truth model, while being significantly less conservative in terms of reference tracking. We demonstrate that due to large weight on the constraints violation penalties the learned policy deems state and input constraints satisfaction a superior task. Hence, the policy may sacrifice reference tracking performance in situations when constraints could be compromised.

Figure 8 shows the closed-loop simulations for the training, validation and test sets with simulation model affected by additive uncertainty $w_k$. The upper plot shows the trajectories of the controlled (orange) and learned predicted (green) state. The middle plot shows control’s policy actions, while the bottom plot shows the influence of measured disturbances. Error bars represent the influence of various realizations of uncertainties. These plots demonstrate that adaptive DPC policy can consistently predict the trajectory and track the reference of a controlled state while satisfying the state and input constraints most of the time. Online learning was active only during the test set and as a result decreased the difference between predicted (green) and controlled (orange) state, as well as the variance of the controlled (orange) state. As a consequence, the constraints handling was also improved with online updates.

5.3 Computational Aspects and Scalability

From the computational and memory standpoint we compare the proposed method with implicit and explicit solutions of the MPC problems. The implicit problem is solved online via quadratic programming (QP) using Matlab’s Quadprog solver. While, the explicit problem is solved offline via multiparametric quadratic programming (mpQP) using MPT3 toolbox [31]. The online computation time and memory footprints of the investigated control policies were evaluated on a laptop with 2.60 GHz Intel(R) i7-8850H CPU and 16 GB RAM on a 64-bit operating system. The assessment of the memory requirements of the implicit MPC is not straightforward, as it depends on the memory footprint of the chosen
Figure 8: Closed-loop control simulation trajectories of simultaneous system identification and adaptive DPC policy learning.

Table 3: Scalability with increasing prediction horizon $N$. Comparison of mean and worst case online computational time per sample, and memory footprint of the proposed DPC policy against implicit (iMPC) and explicit (eMPC) solutions.

| $N$  | 1   | 2   | 4   | 6   | 8   |
|------|-----|-----|-----|-----|-----|
|      | mean CPU time [10^{-3} s] |     |     |     |     |
| DPC  | 0.37 | 0.60 | 1.03 | 1.50 | 1.92 |
| eMPC | 0.15 | 0.45 | 4.04 | 9.48 | 9.76$10^{-3}$ |
| iMPC | 3.00 | 7.86 | 10.52| 10.31| 8.53 |
|      | max CPU time [10^{-3} s] |     |     |     |     |
| DPC  | 9.94 | 15.93| 10.06| 11.73| 10.19 |
| eMPC | 6.00 | 10.00| 23.00| 82.00| 97.49$10^{-3}$ |
| iMPC | 53.00| 40.00| 68.00| 73.00| 92.00 |
|      | memory footprint [kB] |     |     |     |     |
| DPC  | 9.18 | 8.96 | 11.60| 15.80| 21.60 |
| eMPC | 48   | 1051 | 7523 | 35176| 459$10^3$ |
| iMPC | ~150 | ~150 | ~150 | ~150 | ~150 |

In general, the complexity of the QP problem depends on the number of constraints, which scale exponentially with the prediction horizon steps $N$, the number of states, and decision variables. The Table 3 demonstrates the scalability analysis of the evaluation time, and memory footprint as a function of increasing prediction horizon $N$. We train two layer DPC policy, with increasing number of hidden layers $n_{\text{hidden}} = 10 \times N$. Both, DPC and iMPC policies share the same objectives, however, due to the exponential complexity growth in the case of eMPC we had to implement the move blocking strategy limiting the length of the control horizon for prediction horizons larger than 4 as follows: if $N = 4$ then $N_c = 2$, if $N = 6$ then $N_c = 1$, while the problem with $N = 8$ is practically intractable even with $N_c = 1$. The results show that the learned DPC policy is on average 5 to 10 times faster than iMPC policy. In comparison with eMPC, the DPC has significantly smaller memory footprint across all scales. Both, CPU time as well as memory footprint of the DPC policies scale linearly with the problem complexity, while eMPC solutions scale exponentially and are practically infeasible for larger scale problems. These findings are supported by the analytical analysis showing that DNNs with ReLU layers are more memory efficient than lookup tables for parametrizing the PWA functions representing eMPC policies [39]. Moreover, the online complexity of DPC depends entirely on the choice of the policy parametrizations, i.e., number of layers, and hidden neurons. Therefore it can represent a tunable trade-off between the performance and guaranteed complexity, a design trait desirable for embedded applications with limited computational resources.

Footnotes:

2The memory footprint estimate of implicit MPC in Table 3 is based on the standalone Python implementation of the open-source Quadprog solver: https://pypi.org/project/quadprog/
6 Limitations and Future Work

The proposed paper presents a proof of concept with stability and performance analysis of the proposed DPC methodology. The main limitation of this study is the demonstration on small scale fully observable linear systems. The performance on a large-scale, partially observable, and non-linear systems will be addressed in the future work. Similarly to MPC, the DPC policy used in this paper is a full state feedback controller. Therefore a state estimator is necessary for practical applications with partially observable systems. Fortunately, the learned model can be used for the design of the Kalman filter or Moving horizon estimator (MHE). An alternative approach is to learn an autoregressive state-space model based on the output measurements \[70, 23\] as part of the DPC policy.

7 Conclusions

In this work we present a novel data-driven control method systematically combining model-based and learning-based principles. We provide a new perspective on the use of data-driven differentiable programming for obtaining solutions of the parametric programming problems arising in explicit model predictive control (MPC). The proposed method called differentiable predictive control (DPC) is capable of learning neural control policies subject to state and input constraints imposed on the system dynamics models of the controlled process. The system model, as well as the policy, are learned end-to-end without the need for an expert policy to imitate.

We provide sufficient conditions for the global asymptotic stability of the closed-loop systems with linear system dynamics and neural control policies and propose practical stability regularizations for learning neural policies. Furthermore, we discuss the stability of DPC through the perspective of the stability analysis of MPC and provide motivating examples on learning stabilizing neural control policies for systems with unstable dynamics. We compare the control performance, data efficiency, and scalability of the proposed DPC method against different MPC formulations, classical unconstrained controllers, and purely data-driven deep reinforcement learning algorithms.

Based on the presented results and user-friendly end-to-end training, the proposed methodology has the potential to be deployed in the application domains beyond the computational reach of the traditional MPC solution methods while keeping performance and stability guarantees which are typically lacking in purely data-driven approaches. Moreover, we believe that due to the appealing theoretical connections with MPC, the proposed data-driven methodology provides new research opportunities on systematic combination of modern data-driven methods and tools with mature control theoretic concepts.

Acknowledgements

This research was supported by the Mathematics for Artificial Reasoning in Science (MARS), and Data Model Convergence (DMC) initiatives via the Laboratory Directed Research and Development (LDRD) investments at Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL0-1830.

References

[1] Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, Z. Differentiable convex optimization layers. ArXiv abs/1910.12430 (2019).
[2] Alessio, A., and Bemporad, A. A Survey on Explicit Model Predictive Control, in Nonlinear Model Predictive Control: Towards New Challenging Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 345–369.
[3] Amos, B., Rodriguez, I. D. J., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable MPC for end-to-end planning and control. CoRR abs/1810.13400 (2018).
[4] Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. CoRR abs/1609.07152 (2016).
[5] Asarin, E., Bournez, O., Dang, T., and Maler, O. Approximate reachability analysis of piecewise-linear dynamical systems. In Hybrid Systems: Computation and Control (Berlin, Heidelberg, 2000), N. Lynch and B. H. Krogh, Eds., Springer Berlin Heidelberg, pp. 20–31.
[6] Aswani, A., Bouffard, P., Zhang, X., and Tomlin, C. Practical comparison of optimization algorithms for learning-based mpc with linear models, 2014.
[7] ASWANI, A., GONZALEZ, H., SAISTRY, S. S., AND TOMLIN, C. Provably safe and robust learning-based model predictive control. *Automatica* 49, 5 (2013), 1216 – 1226.

[8] BAUER, P., PREMARATNE, K., AND DURAN, J. A necessary and sufficient condition for robust asymptotic stability of time-variant discrete systems. *IEEE Transactions on Automatic Control* 38, 9 (1993), 1427–1430.

[9] BEMPORAD, A., BORRELLI, F., AND MORARI, M. Explicit solution of LP-based model predictive control. In *CDC* (Sydney, Australia, Dec. 2000).

[10] BEMPORAD, A., M. M., DUA, V., AND PISTIKOPOULOS, E. N. The explicit linear quadratic regulator for constrained systems. *Automatica* 38, 1 (2002), 3 – 20.

[11] BEMPORAD, A., AND MORARI, M. Robust model predictive control: A survey. In *Robustness in identification and control* (London, 1999), A. Garulli and A. Testi, Eds., Springer London, pp. 207–226.

[12] BEMPORAD, A., MORARI, M., DUA, V., AND PISTIKOPOULOS, E. N. The explicit solution of model predictive control via multiparametric quadratic programming. In *Proceedings of the 2000 American Control Conference, Danvers, MA, USA* (2000), vol. 2, pp. 872–876.

[13] BEN-TAL, A., AND NEMIROVSKI, A. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS/SIAM Series on Optimization. SIAM, 2001.

[14] BORRELLI, F., BEMPORAD, A., AND MORARI, M. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[15] BOYD, S., AND VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, 2004.

[16] BUJARBARUHA, M., ZHANG, X., ROSOLIA, U., AND BORRELLI, F. Adaptive mpc for iterative tasks. 2018 *IEEE Conference on Decision and Control (CDC)* (2018), 6322–6327.

[17] CAGIENARD, R., GRIEDER, P., KERRIGAN, E., AND MORARI, M. Move Blocking Strategies in Receding Horizon Control. *Journal of Process Control* 17, 6 (July 2007), 563–570.

[18] CHEN, B., CAI, Z., AND BERGÉS, M. Gnu-rl: A precocial reinforcement learning solution for building hvac control using a differentiable mpc policy. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (New York, NY, USA, 2019), BuildSys ’19, Association for Computing Machinery, p. 316–325.

[19] CHEN, S., SAULNIER, K., ATANASOV, N., LEE, D. D., KUMAR, V., PAPPAS, G. J., AND MORARI, M. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)* (June 2018), 1520–1527.

[20] CHEN, S. W., WANG, T., ATANASOV, N., KUMAR, V., AND MORARI, M. Large scale model predictive control with neural networks and primal active sets, 2019.

[21] CHEN, Y., SHI, Y., AND ZHANG, B. Optimal control via neural networks: A convex approach, 2018.

[22] DONTI, P. L., RODERICK, M., FAZLYAB, M., AND KOLTER, J. Z. Enforcing robust control guarantees within neural network policies. *CoRR abs/2011.08105* (2020).

[23] DROGONA, J., KIS, K., TUOR, A., VRAIBIE, D., AND KLAUČO, M. Differentiable Predictive Control: An MPC Alternative for Unknown Nonlinear Systems using Constrained Deep Learning.

[24] DROGONA, J., KVASNICA, M., KLAUČO, M., AND FIKAR, M. Explicit Stochastic MPC Approach to Building Temperature Control. In *IEEE Conference on Decision and Control* (Florence, Italy, 2013), pp. 6440–6445.

[25] DROGONA, J., PICARD, D., KVASNICA, M., AND HELSEN, L. Approximate model predictive building control via machine learning. *Applied Energy* 218 (2018), 199 – 216.

[26] DULAC-ARNOLD, G., MANKOWITZ, D. J., AND HESTER, T. Challenges of real-world reinforcement learning. *CoRR abs/1904.12901* (2019).

[27] GRÖTSCHEL, M., KRUMKE, S. O., AND RAMBAU, J., Eds. *Introduction to Model Based Optimization of Chemical Processes on Moving Horizons*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 295–339.

[28] GUPTA, A., BHARTIYA, S., AND NATARAJ, P. A novel approach to multiparametric quadratic programming. *Automatica* 47, 9 (2011), 2112–2117.

[29] HANIN, B., AND ROLNICK, D. Complexity of linear regions in deep networks. K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 2596–2604.

[30] HENDRIKS, J., JIDLING, C., WILLS, A., AND SCHÖN, T. Linearly constrained neural networks, 2020.

[31] HERCEG, M., KVASNICA, M., JONES, C., AND MORARI, M. Multi-parametric toolbox 3.0. In *2013 European Control Conference, Zurich, Switzerland* (2013), pp. 502–510.
[32] HERTNECK, M., KÖHLER, J., TRIMPE, S., AND ALLGÖWER, F. Learning an approximate model predictive controller with guarantees. CoRR abs/1806.04167 (2018).

[33] HERTNECK, M., KÖHLER, J., TRIMPE, S., AND ALLGÖWER, F. Learning an approximate model predictive controller with guarantees. IEEE Control Systems Letters 2, 3 (2018), 543–548.

[34] HEWING, L., WABERSICH, K. P., MENNER, M., AND ZEILINGER, M. N. Learning-based model predictive control: Toward safe learning in control. Annual Review of Control, Robotics, and Autonomous Systems 3, 1 (2020), null.

[35] HILL, A., RAFFIN, A., ERNESTUS, M., GLEAVE, A., KANERVISTO, A., TRAORE, R., DHARIWAL, P., HESSE, C., KLIMOV, O., NICHL, A., PLAPPERT, M., RADFORD, A., SCHULMAN, J., SIDOR, S., AND WU, Y. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[36] JIA, Z., HUANG, X., CHANG, E. I., AND XU, Y. Constrained deep weak supervision for histopathology image segmentation. IEEE Transactions on Medical Imaging 36, 11 (2017), 2376–2388.

[37] JOHN, V., BOYALI, A., AND MITA, S. Gabor filter and gershgorin disk-based convolutional filter constraining for image classification. Int. J. Mach. Learn. Comput 7, 4 (2017), 55–60.

[38] KANG, K., AND BITMEOAD, R. R. Model Predictive Control with Control Lyapunov Function Support. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 79–87.

[39] KARG, B., AND LUCIA, S. Efficient representation and approximation of model predictive control laws via deep learning. IEEE Transactions on Cybernetics 50, 9 (2020), 3866–3878.

[40] KARG, B., AND LUCIA, S. Approximate moving horizon estimation and robust nonlinear model predictive control via deep learning. Computers & Chemical Engineering 148 (2021), 107266.

[41] KERVADEC, H., DOLZ, J., YUAN, J., DESROSIES, C., GRANGER, E., AND BEN AYED, I. Constrained deep networks: Lagrangian optimization via log-barrier extensions, 05 2020.

[42] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).

[43] KNILL, O. Linear Algebra with Probability. Harvard College Course Math 19b, 2011.

[44] KOCI, L., AND MATEJI, M. Contractive affine transformations of complex plane and applications. Facta Universitatis Series Mathematics and Informatics 21 (01 2006), 65–75.

[45] KOLTER, J. Z., AND MANEK, G. Learning stable deep dynamics models. In Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alch-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 11126–11134.

[46] KVASNICA, M. Real-Time Model Predictive Control via Multi-Parametric Programming: Theory and Tools. VDM Verlag, Saarbrücken, Jan. 2009.

[47] LECHNER, M., HASANI, R., RUS, D., AND GROSU, R. Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In 2020 International Conference on Robotics and Automation (ICRA). IEEE (2020).

[48] LENZ, I., KNEPPER, R. A., AND SAXENA, A. Deepmpc: Learning deep latent features for model predictive control. In Robotics: Science and Systems (2015).

[49] LIN, H., AND ANTSAKLIS, P. J. Stability and stabilizability of switched linear systems: A survey of recent results. IEEE Transactions on Automatic Control 54, 2 (2009), 308–322.

[50] LÖFBERG, J. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In Proc. of the CACSD Conference (Taipei, Taiwan, 2004). Available from http://users.isy.liu.se/johanl/yalmip/.

[51] LÖFBERG, J. Minimax approaches to robust model predictive control, 2003.

[52] MACIEJOWSKI, J. M. Predictive Control with Constraints. Prentice Hall, 2002.

[53] MADDALENA, E. T., DA S. MORAES, C. G., WALTRICH, G., AND JONES, C. N. A neural network architecture to learn explicit mpc controllers from data, 2019.

[54] MAHMOOD, M., AND MHASKAR, P. Constrained control lyapunov function based model predictive control design. International Journal of Robust and Nonlinear Control 24, 2 (2014), 374–388.

[55] MALEK-SHAHMIKRZADI, M. A characterization of certain classes of matrix norms. Linear and Multilinear Algebra 13, 2 (1983), 97–99.

[56] MÁRQUEZ-NEILA, P., SALZMANN, M., AND FUA, P. Imposing hard constraints on deep networks: Promises and limitations. CoRR abs/1706.02025 (2017).
[57] Mayne, D., Rawlings, J., Rao, C., and Scokaert, P. Constrained model predictive control: Stability and optimality. *Automatica* 36, 6 (2000), 789–814.

[58] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *CoRR* abs/1602.01783 (2016).

[59] Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2014), NIPS’14, MIT Press, p. 2924–2932.

[60] Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks, 2014.

[61] Pathak, D., Krähenbühl, P., and Darrell, T. Constrained convolutional neural networks for weakly supervised segmentation. *CoRR* abs/1506.03648 (2015).

[62] Pereira, M., Fan, D. D., An, G. N., and Theodorou, E. A. Mpc-inspired neural network policies for sequential decision making. *CoRR* abs/1802.05803 (2018).

[63] Puskorius, G., and Feldkamp, L. Truncated backpropagation through time and Kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)* (1994), vol. 4, IEEE, pp. 2488–2493.

[64] Raha, A., Chakrabarty, A., Raghunathan, V., and Buzzard, G. T. Embedding approximate nonlinear model predictive control at ultrahigh speed and extremely low power. *IEEE Trans. Control. Syst. Technol.* 28, 3 (2020), 1092–1099.

[65] Rakovic, S. V., Kerrigan, E. C., Mayne, D. Q., and Lygeros, J. Reachability analysis of discrete-time systems with disturbances. *IEEE Transactions on Automatic Control* 51, 4 (April 2006), 546–561.

[66] Ravi, S. N., Dinh, T., Lokhande, V. S. R., and Singh, V. Constrained deep learning using conditional gradient and applications in computer vision. *CoRR* abs/1803.06453 (2018).

[67] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR* abs/1707.06347 (2017).

[68] Schuurmans, J., and Rossiter, J. A. Robust predictive control using tight sets of predicted states. *Control Theory and Applications* 147 (2000), 13–18.

[69] Serra, T., Tiandraatmadja, C., and Ramalingam, S. Bounding and counting linear regions of deep neural networks. *CoRR* abs/1711.02114 (2017).

[70] Skomski, E., Vasisht, S., Wight, C., Tuor, A., Drgona, J., and Vrabie, D. Constrained block nonlinear neural dynamical models.

[71] Soloperto, R., Müller, M. A., Trimpe, S., and Allgöwer, F. Learning-based robust model predictive control with state-dependent uncertainty. *IFAC-PapersOnLine* 51, 20 (2018), 442 – 447. 6th IFAC Conference on Nonlinear Model Predictive Control 2018.

[72] Tøndel, P., Johansen, T. A., and Bemporad, A. An Algorithm for Multi-Parametric Quadratic Programming and Explicit MPC Solutions. *aut* (Nov. 2001). Preprint submitted.

[73] Tuor, A., Drgona, J., and Vrabie, D. Constrained neural ordinary differential equations with stability guarantees. *arXiv preprint arXiv:2004.10883* (2020).

[74] Ushida, D., and Konaka, E. Model predictive control implementation on neural networks using denoising autoencoder. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (Oct 2016), pp. 000149–000154.

[75] Van de Geijn, R., and Myers, M. *Advanced Linear Algebra: Foundations to Frontiers*. Creative Commons NonCommercial (CC BY-NC), 2020.

[76] Wu, Y., Mansimov, E., Liao, S., Grosse, R. B., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR* abs/1708.05144 (2017).

[77] Wu, Z., Albalawi, F., Zhang, Z., Zhang, J., Durand, H., and Christofides, P. D. Control lyapunov-barrier function-based model predictive control of nonlinear systems. *Automatica* 109 (2019), 108508.

[78] Yang, T. Advancing non-convex and constrained learning: Challenges and opportunities. *AI Matters* 5, 3 (Dec. 2019), 29–39.

[79] Zhang, J., Lei, Q., and Dhillon, I. S. Stabilizing gradients for deep neural networks via efficient SVD parameterization. *CoRR* abs/1803.09257 (2018).
Learning Stable Adaptive Explicit Differentiable Predictive Control for Unknown Linear Systems

Appendix

7.1 Stable System Identification

From a systems perspective, we can think of the layers of a recurrent neural network (RNN) as a following discrete-time dynamical system:

$$x_{k+1} = f(x_k, u_k)$$  \(36\)

Using the notation from classical control, \(x_k\) corresponds to the hidden state of the RNN at time step \(k\), whereas \(u_k \in \mathbb{R}^n_u\) is considered the input to the network at time \(k\). In this work we consider three state transition functions, \(f_{\text{LIN}}\), \(f_{\text{RNN}}\), and \(f_{\text{SM}}\), standing for a simple unconstrained linear model, a standard recurrent neural network with Rectified Linear Unit (ReLU) activation, and proposed linear state space model with eigenvalue constraints, respectively. Where \(f_{\text{LIN}}\) has a form:

$$f_{\text{LIN}}(x, u) = Ax + Bu$$  \(37\)

A standard RNN layer \(f_{\text{RNN}}\) is defined as:

$$f_{\text{RNN}}(x, u) = \text{ReLU}(Ax + Bu)$$  \(38\)

where \(A\) and \(B\) correspond to the hidden and input weights respectively.

7.1.1 Enforcing Stability of Learned Dynamics

A stable linear transition \(f_{\text{SM}}\) is obtained by imposing constraints on the \(A\) matrix of the model \(37\).

**Definition 7.1.** Asymptotic Stability: A discrete-time system is said to be asymptotically stable if its bounded initial condition \(x_0\) converges to its equilibrium point \(\bar{x}\):

$$||x_0 - \bar{x}|| < \epsilon \implies \lim_{k \to \infty} ||x_k|| = \bar{x}$$  \(39\)

For discrete-time linear system \(37\) holds that all eigenvalues \(\lambda\) of the state transition matrix \(A\) lie within unit circle.

By the Perron–Frobenius theorem, the dominant eigenvalue (spectral radius) of a non-negative square matrix \(A\) is bounded by the minimum and maximum of its row-wise sums. That is, there exists a real eigenvalue of \(A\), \(\rho > 0\) s.t. for any other eigenvalue, \(\lambda\), we have \(|\lambda| \leq \rho\), and \(\min_i \sum_j A_{ij} \leq \rho \leq \max_i \sum_j A_{ij}\). Hence, one can enforce bounds on dominant eigenvalue by constraining the row-wise sums of \(A\)'s elements. Let \(\sigma\) be the elementwise standard logistic sigmoid function, \(\epsilon \in [0, 1)\), and \(\mathbb{1}\) a matrix of 1’s. Our parametrization, bounding the dominant eigenvalue of the transition matrix \(A\) to \(\rho \in [1 - \epsilon, 1]\), is formulated as:

$$M = \mathbb{1} - \epsilon \sigma(M')$$  \(40\)

$$A_{i,j} = \frac{\exp(A'_{i,j})M_{i,j}}{\sum_{k=1}^{n_x} \exp(A'_{i,k})}$$  \(41\)

\(M\) is a matrix modeling damping defined as a function of parameter matrix \(M' \in \mathbb{R}^{n_x \times n_x}\). \(M\)'s elementwise multiplication with the softmax regularized rows of the \(A'\) matrix gives the state transition matrix \(A\). By constraining the magnitudes of the dominant eigenvalue to be less or equal to one, the stability of the learned dynamics of the discrete-time linear system \(37\) is guaranteed as given by definition 7.1. For future reference we denote the linear system with active eigenvalue constraints \(41\) as \(f_{\text{SM}}\).
Table 4: Test set $N$-step MSE.

| $N$   | 8   | 16   | 32   | 64   | 128  | 256  |
|-------|-----|------|------|------|------|------|
| LIN   | 0.009 | 0.288 | 0.349 | 0.228 | 0.508 | 0.800 |
| SM    | 0.130  | 0.276  | 0.432  | 0.498  | 0.446  | 0.504  |
| RNN   | 0.106  | 0.044  | 0.299  | 0.481  | 0.411  | 0.800  |

Table 5: Test set open-loop MSE.

| $N$   | 8   | 16   | 32   | 64   | 128  | 256  |
|-------|-----|------|------|------|------|------|
| LIN   | 1.685  | 0.472 | 1.134 | 1.023 | 0.350 | 1.071 |
| SM    | 5.374  | 1.331 | 0.938 | 0.469 | 0.446 | 0.313 |
| RNN   | 2.585  | 5.634 | 1.462 | 1.437 | 0.884 | 1.071 |

7.1.2 Model optimization

For the given recurrent models we optimize the corresponding $A$, $B$, (and $M'$ for stable model $f_{SM}$) parameter matrices by an objective function which serves as a reasonable proxy for our goal to reproduce the response of the original system. We use the average Mean Squared Error (MSE) for the model response over an $N$-step prediction horizon compared to observed system trajectories. For each $N$ time steps we have sequences of vector-valued measurements, $X^{ID} = \{x^{ID}_0, ... x^{ID}_N\}$, $U^{ID} = \{u^{ID}_0, ... u^{ID}_N\}$. The models are optimized using gradient descent on the following loss function:

$$L_{MSE}(X, X^{ID} | A, B) = \frac{1}{N} \sum_{k=1}^{N} \|x_{k,i} - x^{ID}_{k,i}\|_2^2$$ (42)

7.2 System Identification Task

Table 4 shows the $N$-step MSE on the test set for each $N$-step prediction horizon training objective. The bold entries indicate the best performance for a given prediction horizon. While Table 5 shows the open-loop MSE on the test set for models trained with each $N$-step prediction horizon. For all prediction horizons, either the stable or simple linear model perform better on open-loop MSE, with the stable model achieving the best overall open-loop MSE when trained on the longest prediction horizon. In fact, for the stable model we see an opposite trend from the $N$-step MSE, in that as $N$ grows larger, models tend to do better at aligning the open-loop simulation with the reference trajectory. The absolute and percentage of reduction in open-loop MSE from the contending LIN model (0.35 MSE) to the best performing stable model (0.313 MSE) are 0.037 and 10% respectively.

Figure 9 shows the concatenated open-loop simulations from the training, validation and test sets, with the solid blue line indicating the true system trajectory. All models closely trace the path of the reference $x_4$ given only the initial state of the system and external inputs.

7.3 Control Method Design and Tuning

7.3.1 Differentiable predictive control

For continuous control task we train the proposed DPC policy as defined in algorithm 3 by optimizing the loss function (9) with one-step ahead prediction horizon $N = 1$. The choice of the control policy $u_k = \pi_W(\xi)$ depends on the problem complexity. For the purposes of this paper we used a two-layer neural network $\pi_W(\xi): \mathbb{R}^{n_{\xi}} \rightarrow \mathbb{R}^{N_u}$ with ReLU activations, and policy features $\xi = \{x_k, r_k, x_k, x_k, u_k, u_k\}$. The number of hidden units in each layer given by $n_{\text{hidden}} = 10 \times N$. The policy is trained with Adam optimizer [42] and learning rate of $0.001$ on $30,000$ epochs. The policy begins training with randomly initialized weights. The different one week periods of 2016 samples are used as training, validation, and test sets, respectively. The weights of the loss function (9) are tuned using the physical insight about the system dynamics with the following values: $Q_r = 2e1, Q_u = 1e-6, \mu = 5e-7, \lambda = 5e1$. The results are reported on the test set.
7.3.2 Classical control methods

We compare the performance of the trained DPC policy with classical model-based linear control methods, namely linear quadratic regulator (LQR), and Linear quadratic integral controller (LQI) implemented in Matlab environment using commands `dlqr` and `lqi`, respectively. The weights of LQI controller $Q_r$, and $Q_u$ are chosen to be identical to the DPC policy, while tuning of the LQR weights was necessary to obtain satisfactory performance.

7.3.3 Model predictive control

We evaluate the performance of three different formulations of model predictive control (MPC) problem. The nominal MPC is obtained by the solution of the constrained optimization as defined by (1) with softened input and state constraints. Robust MPC (RMPC) represents the worst-case scenario approach via constraints tightening techniques [68, 51]. On the other hand, stochastic MPC (SMPC) is based on probabilistic constraints approximated by sampling from the known distribution of the unknown disturbances. The implementation of RMPC and SMPC is based on the formulations presented in [24]. For a fair comparison, all MPC problems are designed with identical setup of the hyperparameters ($N, Q_r, Q_u, \lambda, \mu$) as in the case of trained DPC policy. The MPC problems are implemented in the Matlab environment using the Yalmip optimization toolbox [50].

7.3.4 Deep reinforcement learning

Additionally, we assess the performance of three deep reinforcement learning (RL) algorithms suitable for dealing with continuous state and action spaces obtained from the Stable Baselines implementation [35]. In particular, we investigate an asynchronous, deterministic variant of Advantage Actor Critic (A2C) [58], Actor Critic using Kronecker-Factored Trust Region (ACKTR) [76], and Proximal Policy Optimization algorithm [67]. We design the reward to be equal to the negative value of the MPC loss (9) using the same values of the weights $Q_r = 2e1, Q_u = 1e - 6, \mu = 5e - 7, \lambda = 5e1$. For a fair comparison, we train randomly initialized RL policies on 30,000 episodes over the same number of time steps as DPC with a learning rate of 0.01. We use the ground truth system dynamics model as the training environment.