Approximating Explicit Model Predictive Control Using Constrained Neural Networks

Steven Chen[1], Kelsey Saulnier[1], Nikolay Atanasov[2],
Daniel D. Lee[1], Vijay Kumar[1], George J. Pappas[1], and Manfred Morari[1]

Abstract— **This paper presents a method to compute an approximate explicit model predictive control (MPC) law using neural networks. The optimal MPC control law for constrained linear quadratic regulator (LQR) systems is piecewise affine on polytopes. However, computing this optimal control law becomes computationally intractable for large problems, and motivates the application of reinforcement learning techniques using neural networks with rectified linear units. We introduce a modified reinforcement learning policy gradient algorithm that utilizes knowledge of the system model to efficiently train the neural network. We guarantee that the network generates feasible control inputs by projecting onto polytope regions derived from the maximal control invariant set of the system. Finally, we present numerical examples that demonstrate the characteristics and performance of our algorithm.**

I. Introduction

Model predictive control (MPC) is a dynamic optimization technique that has seen widespread use in industrial process applications such as oil refineries and chemical plants [1]. Recently, MPC has found mainstream use in robotics for the control of quadrotors [2], [3], autonomous vehicles [4], and humanoid robots [5] due to its versatility, robustness, and safety guarantees. The transition from the process industry to robotics brings an additional challenge since the available computation time is reduced from hours to milliseconds.

One way to manage the computational load is to pre-compute the optimal control law $\mathbf{u}^* = \boldsymbol{\mu}^*(\mathbf{x})$ offline as a function of all feasible states $\mathbf{x}$. The resulting control law for a linear system with quadratic cost is known to be piecewise affine (PWA) on polytopes. If $\boldsymbol{\mu}^*(\mathbf{x})$ is pre-computed offline, the online optimization problem is reduced to determining the polytopic region the system state is in, and applying the pre-computed affine control. This method is called *explicit* MPC, in contrast to *implicit* MPC, which solves the optimization problem online at the current state $\mathbf{x}$ at each time step as needed. The drawback of explicit MPC approaches, however, is that the computational complexity, measured by the number of polytopic regions, grows quickly with the number of constraints. As a result, computing the optimal explicit control law becomes computationally intractable for large systems. In addition, even if this optimal control law can be computed, the process of determining which region contains the system state can require too much processing power or memory storage for real-time evaluation online [6].

Fast MPC methods for implicit MPC focus on speeding up the online optimization process. Wang et al. [7] exploit the specific problem structure of MPC to decrease the time complexity of solving the resulting quadratic program (QP). Richter et al. [8] use the fast gradient method and provide a practical upper bound on the number of iterations needed for a specified accuracy.

One approach to address the computational limitations of optimal explicit MPC control laws is to identify a sub-optimal polytope partition of the state space, and construct a control law or cost function based on those polytope regions. Johansen et al. [9] partition the state space into orthogonal hypercubes organized in a hierarchical data structure, and the tree-structure of the resulting controller allows for real-time computational complexity that is logarithmic with respect to the number of hypercube regions. Summers et al. [10] similarly use a hierarchical sparse grid structure to approximate the value function and control law. Jones et al. [11] use a double-description method to build piecewise affine (PWA) approximations of the value function, and use barycentric functions on the polytopic regions of the approximate value function to compute an approximate control law. Our approach differs from these methods because we do not explicitly construct the polytope regions.

Bemporad et al. [12] use canonical PWA basis functions to construct PWA Simplicial (PWAS) approximations of the optimal control law, obtaining the corresponding weights of these basis functions by solving a convex optimization problem. This approach is similar to our method in that both can be viewed as function approximation, but differs in that they design their basis functions, or features, while our neural network instead learns features during training.

An alternative approach assumes that the explicit optimal control law is available, and modifies it to obtain a simpler control law with fewer polytopic regions. Kvasnica et al. [6], [13] reduce the complexity of the PWA control law by eliminating regions which have attained saturated values, thus resulting in a simpler, but still optimal control law. Takacs et al. [14] find a sub-optimal PWA control law by first obtaining polytope regions by solving the explicit MPC optimization problem with a reduced time horizon. They then locally minimize the integrated squared error with respect to the optimal control law within each of these regions. These

[1]S. Chen, K. Saulnier, D. Lee, V. Kumar, G. Pappas, and M. Morari are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA `chenste@seas.upenn.edu`

[2]N. Atanasov is with the Department of Electrical and Computer Engineering, UC San Diego, La Jolla, CA 92093, USA `natanasov@ucsd.edu`

This work is supported in part by ARO grant W911NF-13-1-0350 and DARPA grant HR001151626/HR0011516850.

methods differ from our approach since we do not assume that the optimal explicit control law is available.

Parisini et al. [15] use a neural network to approximate the control law, but differs from our approach in that they consider nonlinear systems, do not consider constraints, and train their network using supervised learning. Other applications of neural networks in MPC focus on approximating nonlinear system models [16]–[18]. The use of function approximators in control problems, called *Approximate Dynamic Programming (ADP)*, has many connections to reinforcement learning (RL) [19]. The application of RL to linear quadratic regulator (LQR) and MPC problems has been previously explored [20]–[22], but the motivation in those cases is to handle dynamics models of known form with unknown parameters. The recent success of deep RL demonstrates the ability of neural networks to model extremely large-scale RL problems [23]–[25]. However, most of these approaches suffer from high sample complexity and do not guarantee feasible control inputs.

The goal of this work is to develop methods for incorporating prior knowledge about constrained LQR, such as the piecewise affine structure of the optimal control law and the maximal control invariant sets, into deep RL techniques, in order to train a neural network that approximates the explicit MPC control law. The contributions of the paper are:

- an architecture for a PWA neural network that is guaranteed to generate feasible control inputs, accomplished by projecting onto convex constraint regions derived from the maximal control invariant set of the system, and
- a reinforcement learning algorithm that improves the efficiency and performance of policy gradient methods by incorporating the known system model.

II. Problem Statement

Consider a discrete-time linear time-invariant system

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \quad \mathbf{x}_k \in \mathbb{R}^n,\ \mathbf{u}_k \in \mathbb{R}^m.$$

Our goal is to compute an infinite sequence of control inputs $\mathbf{u}_{0:\infty}$ to regulate the system to a desired state subject to a set of constraints. It is assumed that the pair $(\mathbf{A}, \mathbf{B})$ is stabilizable. This problem, known as the constrained infinite-horizon LQR, is:

$$\begin{aligned} \min_{\mathbf{u}_{0:\infty}} \quad & V_\infty(\mathbf{x}_0) = \sum_{k=0}^{\infty} \left(\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \\ & \mathbf{x}_k \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U}, \end{aligned} \tag{1}$$

where $\mathbf{M} \in \mathcal{S}^n_{\succeq 0}$ and $\mathbf{R} \in \mathcal{S}^m_{\succ 0}$ are chosen to define the desired optimal behavior for the system, and $\mathcal{X}$, $\mathcal{U}$ are polyhedra which contain the origin in their interior. Rather than computing a sequence of control inputs $\mathbf{u}_{0:\infty}$ for a given state $\mathbf{x}_0$, our goal is to compute a control function $\boldsymbol{\mu}^*(\mathbf{x}_k)$ that specifies the optimal control input for an arbitrary state $\mathbf{x}_k$ at time k. This is possible since the system is time-invariant and Problem (1) is infinite-horizon, and hence the optimal control policy is stationary.

III. Preliminaries

A. Explicit Model Predictive Control

In most cases, the constrained infinite-horizon formulation is not feasible to solve [26], and a Receding Horizon Control (RHC) or MPC technique is used. These methods simplify Problem (1) by restricting the optimization to a finite horizon, N, and introducing an appropriate terminal cost $\mathbf{x}_N^T\mathbf{F}\mathbf{x}_N$ and terminal state constraints $\mathbf{x}_N \in \mathcal{X}_f$ as follows:

$$\begin{aligned} \min_{\mathbf{u}_{0:(N-1)}} \quad & V(\mathbf{x}_0) = \mathbf{x}_N^T\mathbf{F}\mathbf{x}_N + \sum_{k=0}^{N-1} \left(\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \\ & \mathbf{x}_k \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U}, \\ & \mathbf{x}_N \in \mathcal{X}_f. \end{aligned} \tag{2}$$

The terminal cost, $\mathbf{x}_N^T\mathbf{F}\mathbf{x}_N$, is chosen to bound the cost for the remaining time (N, ∞). A common choice for $\mathbf{F}$ is the solution to the Algebraic Riccati Equation:

$$\mathbf{F} = \mathbf{A}^T\mathbf{F}\mathbf{A} + \mathbf{M} - \mathbf{A}^T\mathbf{F}\mathbf{B}(\mathbf{B}^T\mathbf{F}\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^T\mathbf{F}\mathbf{A} \tag{3}$$

which corresponds to the optimal cost-to-go after N time steps for the unconstrained infinite-horizon LQR problem. The optimal control law for the corresponding unconstrained infinite-horizon LQR problem is $\mathbf{u}_k = -\mathbf{K}\mathbf{x}_k$ where the LQR gain matrix is:

$$\mathbf{K} = (\mathbf{R} + \mathbf{B}^T\mathbf{F}\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{F}\mathbf{A}). \tag{4}$$

There exists an $N_* < \infty$ such that, for all $N > N_*$, the system reaches the unconstrained region around the origin within the time horizon [27], [28]. In this case, the cost-to-go, $\mathbf{x}_N^T\mathbf{F}\mathbf{x}_N$, is exact and Problems (1) and (2) are equivalent.

Problems (1) and (2) may be reformulated as learning problems by introducing a family of functions $\mathbf{u}_k = \boldsymbol{\mu}^{\boldsymbol{\theta}}(\mathbf{x}_k)$, parameterized by $\boldsymbol{\theta}$, and minimizing the cost function:

$$V^{\boldsymbol{\theta}}(\mathbf{x}_0) = \mathbf{x}_N^T\mathbf{F}\mathbf{x}_N + \sum_{k=0}^{N-1}(\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + \boldsymbol{\mu}^{\boldsymbol{\theta}}(\mathbf{x}_k)^T\mathbf{R}\boldsymbol{\mu}^{\boldsymbol{\theta}}(\mathbf{x}_k)) \tag{5}$$

with respect to $\boldsymbol{\theta}$. In this case, $V^{\boldsymbol{\theta}}(\mathbf{x}_0)$ is called the *policy conditional* value function and represents the cost incurred by starting at $\mathbf{x}_0$ and following control law $\boldsymbol{\mu}^{\boldsymbol{\theta}}$.

Similarly, we define a *policy conditional* Q-function:

$$\begin{aligned} Q^{\boldsymbol{\theta}}(\mathbf{x}_0, \mathbf{u}_0) = \quad & \mathbf{x}_0^T\mathbf{M}\mathbf{x}_0 + \mathbf{u}_0^T\mathbf{R}\mathbf{u}_0 + \mathbf{x}_N^T\mathbf{F}\mathbf{x}_N + \\ & \sum_{k=1}^{N-1}\left(\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + \boldsymbol{\mu}^{\boldsymbol{\theta}}(\mathbf{x}_k)^T\mathbf{R}\boldsymbol{\mu}^{\boldsymbol{\theta}}(\mathbf{x}_k)\right) \end{aligned} \tag{6}$$

The traditional approach to obtain an explicit feedback law $\boldsymbol{\mu}^*$ for Problem (1) is to formulate it as Problem (2) and solve it via multi-parametric Quadratic-Programming (mp-QP) techniques [26, Chs. 6, 11]. As a result, the optimal control law $\boldsymbol{\mu}^*(\mathbf{x})$ is *continuous and piecewise affine on polyhedra* and the optimal value function $V^*(\mathbf{x})$ is *continuous, piecewise quadratic on the same polyhedra, and convex in the state* $\mathbf{x}$.

B. Overview of Our Approach

The challenge with computing the explicit MPC control law is that the number of polytopic regions that determine the optimal control law may grow exponentially with the problem size. Specifically, the number of critical regions is upper-bounded by the number 2^q of possible combinations of active constraints, where q is the number of constraints [29]. Increasing the state and control dimensionality, or increasing the time horizon, all increase q. The actual number of regions in the optimal control law is usually significantly less than the exponential upper bound, and most prior work on constructing approximate explicit MPC control laws focus on construction or refinement of polytopic regions.

Rather than focusing on the construction of these polytopic regions, we instead use function approximation and reinforcement learning techniques to directly learn an approximate explicit control law. We first specify the architecture of a deep neural network with rectified linear units (DNN ReLU) that is guaranteed to respect constraint satisfaction of our MPC problem. Once this architecture has been specified, rather than minimizing some error function with respect to the optimal control law, the algorithm then minimizes the infinite-horizon value function V_∞^θ over the parameters of the neural network. The two key components of our approach are: designing a neural network architecture that respects constraint satisfaction (Sec. IV) and designing an algorithm to find the optimal parameters θ^* (Sec. V).

IV. Neural Network Architecture

We first introduce and motivate using a DNN ReLU to approximate the control law and provide guidance on choosing an appropriate architecture for these neural networks. We then introduce *Dykstra's projection* algorithm, which we use to guarantee that the network will not generate control inputs which lead to constraint violations.

A. Deep Neural Network with Rectified Linear Units

A neural network is a parameteric function approximator, $\mathbf{g}(\mathbf{x};\boldsymbol{\theta})$, which can be used to approximate the optimal control law $\boldsymbol{\mu}^*(\mathbf{x}) \approx \mathbf{g}(\mathbf{x};\boldsymbol{\theta})$ by an appropriate choice of parameters $\boldsymbol{\theta}$. A deep neural network with L layers represents $\mathbf{g}(\mathbf{x};\boldsymbol{\theta})$ as a composition of L affine functions $\boldsymbol{\lambda}_j(\mathbf{x}) := \mathbf{W}_j\mathbf{x} + \mathbf{b}_j$, each except the last one followed by a nonlinear activation function $\mathbf{h}$, so that (see Fig. 1):

$$\mathbf{g}(\mathbf{x};\boldsymbol{\theta}) = \boldsymbol{\lambda}_L \circ \mathbf{h} \circ \boldsymbol{\lambda}_{L-1} \circ \cdots \circ \mathbf{h} \circ \boldsymbol{\lambda}_1(\mathbf{x}),$$

where $\boldsymbol{\theta} := \{\mathbf{W}_{1:L}, \mathbf{b}_{1:L}\}$ are the affine function parameters to be optimized, and $\mathbf{h}$ is a fixed (not optimized) function, typically chosen as a sigmoid, hypertangent, or rectified linear unit (ReLU) function (see [31, Ch.6] for details).

The optimal control law $\boldsymbol{\mu}^*(\mathbf{x})$ of the constrained LQR problem is a PWA function on polytopes. As a result, the ReLU activation function, $\mathbf{h}(\mathbf{x}) := \max\{0, \mathbf{x}\}$ (element-wise), is of particular interest because a DNN ReLU is a composition of PWA functions on polytopes, and as a result is also a PWA function on polytopes [32]. In addition, a DNN ReLU with n_0 inputs, n_L outputs, and $L-1$ hidden

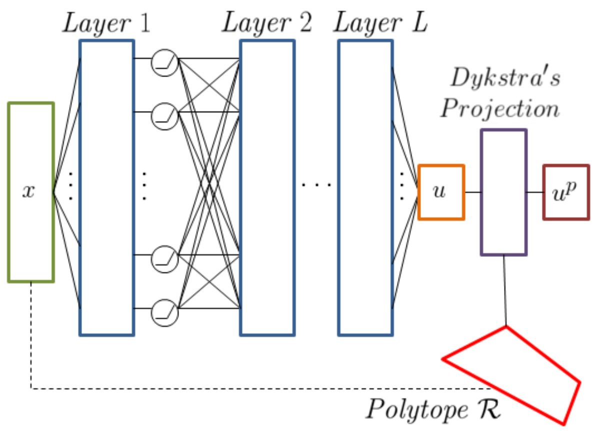

Fig. 1: The control law neural network is used to approximate the optimal control input $\mathbf{u}^*$ with $\mathbf{u}$ for a given state $\mathbf{x}$. This potentially infeasible $\mathbf{u}$ is projected onto the polytope $\mathcal{R}$ defined by the constraints of (9) to output the feasible control input $\mathbf{u}^p$.

layers of width $n \geq n_0$ can represent PWA functions with $\Omega\left((\frac{n}{n_0})^{(L-2)n_0} n^{n_0}\right)$ affine regions [34, Thm. 4, Cor. 5]. The exact formula is

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0}\right) \sum_{j=0}^{n_0} \binom{n_L}{j}. \tag{7}$$

DNN ReLU are universal approximators [33], [36], and in current deep learning practice, ReLU activations have become the de-facto standard activation function [36]. These observations make a DNN ReLU attractive for synthesizing an approximate explicit MPC controller because 1) the number of polytopic regions computed by the deep ReLU models grows exponentially with the number of layers L and polynomially with the number of weights per layer n and 2) for a given architecture, we can calculate a lower bound on the maximal number of polytopic regions that the architecture can compute. Thus we can first design an architecture based on considerations such as online computation speed or memory requirements, and then use Eqn. (7) to estimate the number of regions our chosen DNN ReLU can compute.

B. Dykstra's Projection Algorithm

One difficulty in using a DNN ReLU in the constrained LQR problem is guaranteeing constraint satisfaction, i.e. that $\mathbf{x}_{0:\infty} \in \mathcal{X}$, $\mathbf{u}_{0:\infty} \in \mathcal{U}$. Prior work has restricted $\mathbf{u}_k \in \mathcal{U}$ by manipulating the gradient $\nabla_{\boldsymbol{\theta}}\mathbf{g}$ (zeroing, squashing, or inverting) as the output $\mathbf{u}_k$ nears constraint violation [37]. It is, however, unclear how these approaches affect the learning performance or how they could be used to also enforce the remaining trajectory constraints $\mathbf{x}_{k+1:\infty} \in \mathcal{X}$.

The feasible set $\mathcal{X}_\infty$ is the set of all states $\mathbf{x}_0$ where Problem (1) is feasible and $V_\infty^* < +\infty$. We use Dykstra's projection algorithm [30] to guarantee that $\forall\ \mathbf{x}_0 \in \mathcal{X}_\infty$, recursively calling our neural network control law $\mathbf{g}(\mathbf{x_k};\boldsymbol{\theta})$ at each state $\mathbf{x}_k$ will generate feasible state and control input trajectories $\mathbf{x}_{1:\infty} \in \mathcal{X}$ and $\mathbf{u}_{0:\infty} \in \mathcal{U}$. Recall that the constraint regions $\mathcal{X}$ and $\mathcal{U}$ are polytopes. For a given state $\mathbf{x}_k$, the neural network outputs a single, potentially infeasible, control input $\mathbf{u}_k$ which must be projected in a way that ensures that the subsequent state and control trajectories remain feasible. To obtain such a guarantee, we compute the *maximal control invariant set* $\mathcal{C}_\infty \subseteq \mathcal{X}$:

$$\begin{aligned}\mathcal{C}_\infty := \{\mathbf{x}_k \in \mathcal{X} \mid \exists \{\mathbf{u}_t\}_{t=k}^{\infty} \text{ s.t. } \mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t,\\ \mathbf{u}_t \in \mathcal{U}, \mathbf{x}_t \in \mathcal{X}, \forall t \in \{k, k+1, \ldots\}\}.\end{aligned}$$

Hence, for any state $\mathbf{x}_k \in \mathcal{C}_\infty$, we project the network output $\mathbf{u}_k$ onto a polytope

$$\mathcal{R}(\mathbf{x}_k) = \{\mathbf{u} | \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u} \in \mathcal{C}_\infty, \mathbf{u} \in \mathcal{U}\}. \tag{8}$$

Starting at $\mathbf{x}_k$, by recursively following our network control law and performing a similar projection at all resulting states $\mathbf{x}_{k+1:\infty}$, we will guarantee that $\mathbf{x}_{k+1:\infty} \in \mathcal{X}$ and $\mathbf{u}_{k:\infty} \in \mathcal{U}$. Notice that since $\mathcal{X}_\infty \subseteq \mathcal{C}_\infty$ [26], projecting $\mathbf{u}_k$ onto $\mathcal{R}(\mathbf{x}_k)$ does not eliminate any feasible solutions.

There are standard algorithms to compute $\mathcal{C}_\infty$, but in general it is difficult and is not guaranteed to terminate in finite time [26]. In this work we make the assumption that $\mathcal{C}_\infty$ is computable, but this aspect is an interesting direction for future work. One promising alternative is to approximate the set, and there has already been some work in this area [39]. A simpler alternative is to relax this guarantee by projecting $\mathbf{u}_k$ onto a set associated with a smaller control invariant set, or onto a user-defined safe region.

Since $\mathcal{C}_\infty$ and $\mathcal{U}$ are both polytopes, we can use their $\mathcal{H}$-representations to express the constraints as an intersection of halfspaces:

$$\begin{aligned}\mathcal{C}_\infty &= \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{C}_c\mathbf{x} \le \mathbf{d}_c\}\\ \mathcal{U} &= \{\mathbf{u} \in \mathbb{R}^m \mid \mathbf{C}_u\mathbf{u} \le \mathbf{d}_u\}.\end{aligned}$$

Thus, given $\mathbf{x}_k \in \mathcal{C}_\infty$ and the potentially infeasible neural network output $\mathbf{u}_k = \mathbf{g}(\mathbf{x}_k; \boldsymbol{\theta})$, we can compute a projected control input $\mathbf{u}_k^p \in \mathcal{R}(\mathbf{x}_k)$ which ensures that $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k^p \in \mathcal{C}_\infty$ by solving the following quadratic program:

$$\begin{aligned}\underset{\mathbf{u}_k^p}{\arg\min} \quad & \|\mathbf{u}_k - \mathbf{u}_k^p\|_2^2\\ \text{s.t.} \quad & \mathbf{C}_c\mathbf{B}\mathbf{u}_k^p \le \mathbf{d}_c - \mathbf{C}_c\mathbf{A}\mathbf{x}_k\\ & \mathbf{C}_u\mathbf{u}_k^p \le \mathbf{d}_u\end{aligned} \tag{9}$$

where the constraints are the $\mathcal{H}$-representation of $\mathcal{R}(\mathbf{x}_k)$, and the optimal solution is the orthogonal projection of $\mathbf{u}_k$ onto $\mathcal{R}(\mathbf{x}_k)$. Notice that once $\mathcal{C}_\infty$ is calculated offline, $\mathcal{R}(\mathbf{x}_k)$ can easily be computed online through matrix multiplication operations. While (9) can be solved via standard quadratic programming methods, we choose to use Dykstra's projection algorithm because the projection onto the individual halfspace constraints has a closed form solution, and Dykstra's is *guaranteed* to converge to the orthogonal projection since $\mathcal{R}(\mathbf{x}_k)$ is the intersection of closed convex sets [30].

Let $\mathbf{u}^p = \mathcal{P}^{\mathcal{R}}(\mathbf{u})$ be the orthogonal projection of $\mathbf{u}$ onto a polytope $\mathcal{R}$ corresponding to the intersection of r halfspace constraints, $\mathbf{c}_i^T\mathbf{u} \le d_i$, for $i = 1, \ldots, r$ in (9) and let $\mathcal{P}_i^{\mathcal{R}}$ denote the projection onto the i-th halfspace:

$$\mathcal{P}_i^{\mathcal{R}}(\mathbf{u}) := \begin{cases}\mathbf{u} + (d_i - \mathbf{c}_i^T\mathbf{u})\mathbf{c}_i/||\mathbf{c}_i||_2^2 & \text{if } \mathbf{c}_i^T\mathbf{u} > d_i\\ \mathbf{u} & \text{if } \mathbf{c}_i^T\mathbf{u} \le d_i.\end{cases} \tag{10}$$

Dykstra's projection algorithm [30] generates a sequence of variables $\mathbf{u}^{(i,j)}$ and $\mathbf{I}^{(i,j)}$ for $i = 1 \ldots r$ and $j \in \mathbb{N}$. For convenience, let $\mathbf{u}^{(0,j)} = \mathbf{u}^{(r,j-1)}$. The algorithm initializes with $\mathbf{u}^{(0,0)} := \mathbf{u}$ and $\mathbf{I}^{(0,0)} = \mathbf{0}$. It then iterates as follows:

$$\begin{aligned}\mathbf{u}^{(i,j)} &= \mathcal{P}_i^{\mathcal{R}}\Big(\mathbf{u}^{(i-1,j)} - \mathbf{I}^{(i,j-1)}\Big)\\ \mathbf{I}^{(i,j)} &= \mathbf{u}^{(i,j)} - \Big(\mathbf{u}^{(i-1,j)} - \mathbf{I}^{(i,j-1)}\Big).\end{aligned} \tag{11}$$

The variable $\mathbf{u}^{(i,j)}$ converges to $\mathcal{P}^{\mathcal{R}}(\mathbf{u})$ as $j \to \infty$, terminating once all constraints are satisfied. Thus given a state $\mathbf{x}_k$, neural network $\mathbf{g}(\mathbf{x}_k; \boldsymbol{\theta})$, and polytope $\mathcal{R}(\mathbf{x}_k)$, our approximate control law will output $\mathbf{u}_k^p = \mathbf{f}(\mathbf{x}_k; \boldsymbol{\theta}) = \mathcal{P}^{\mathcal{R}(\mathbf{x}_k)}(\mathbf{g}(\mathbf{x}_k; \boldsymbol{\theta}))$.

V. Training the Neural Network

This section discusses optimization over the parameters $\boldsymbol{\theta}$ of the neural network architecture with Dykstra's projection $\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})$ developed in Sec. IV in order to approximate the optimal control law $\boldsymbol{\mu}^*(\mathbf{x})$. One possible approach is to use a supervised learning method, which entails sampling a finite set $\{\mathbf{x}^{(i)}\}_{i=1}^{N_B} \in \mathcal{C}_\infty$ of size N_B, using implicit MPC to solve for the corresponding optimal control inputs $\{\mathbf{u}^{*(i)}\}_{i=1}^{N_B}$, and then minimizing $\frac{1}{N_B}\sum_{i=1}^{N_B}(\mathbf{u}^{*(i)} - \mathbf{f}(\mathbf{x}^{(i)}; \boldsymbol{\theta}))^2$ with respect to parameters $\boldsymbol{\theta}$. While this approach might work well for linear systems, generalizing it to nonlinear systems would be challenging as it would require solving implicit MPC for a nonlinear system.

Instead, we propose a reinforcement learning approach, specifically a policy gradient method, that iteratively updates the parameters $\boldsymbol{\theta}$ in the direction of the gradient $\nabla_{\boldsymbol{\theta}} V_\infty^{\boldsymbol{\theta}}(\mathbf{x})$ of the cost function, and extends naturally to nonlinear systems. For our application, policy gradient methods are more suitable than value-based methods because policy gradient methods will use the PWA DNN ReLU to approximate the PWA control law, rather than the piecewise quadratic (PWQ) value function. In addition, while it has been difficult to provide convergence assurances for value-based algorithms that rely on function approximation [19], policy gradients using function approximation yield unbiased estimates of the gradient with respect to the parameters $\boldsymbol{\theta}$, and hence are guaranteed to converge to locally optimal solutions [40].

We propose an algorithm similar to REINFORCE [41] to solve Problem (1) and exploit the reduction to the MPC Problem (2) to estimate the infinite-horizon value function efficiently. To derive the algorithm, we define a stochastic control law, which samples control inputs $\mathbf{u}_k$ from a multivariate Gaussian probability density function (pdf) $\phi(\cdot; \mathbf{f}(\mathbf{x}_k; \boldsymbol{\theta}), \boldsymbol{\Sigma})$, centered at the DNN ReLU output $\mathbf{f}(\mathbf{x}_k; \boldsymbol{\theta})$ with diagonal covariance $\boldsymbol{\Sigma}$. The covariance $\boldsymbol{\Sigma}$ is annealed to $\mathbf{0}$ by the end of training to return to the deterministic control law, and plays a similar role as the exploration parameter ϵ in ϵ-greedy annealing for Q-learning [23].

For a stochastic control law, the infinite-horizon value function should be redefined as follows:

$$V_\infty^{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{\tau \sim p(\cdot|\boldsymbol{\theta})}\left[\sum_{k=0}^{\infty}(\mathbf{x}_k^T\mathbf{M}\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k)\middle|\mathbf{x}_0 = \mathbf{x}\right]$$

where $\tau := (\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots)$ is a state-control trajectory with pdf $p(\cdot|\theta)$ defined by the stationary distribution of the states encountered under the stochastic control law $\phi(\cdot; \mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), \boldsymbol{\Sigma})$. Define the Q-functions $Q^{\theta}_{\infty}(\mathbf{x}, \mathbf{u})$ similarly. We rely on the following result to obtain the gradient of the value function.

Theorem (Policy Gradient [40])**.** *The gradient of the infinite-horizon value function $V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x})$ with respect to the policy parameters $\boldsymbol{\theta}$ is*

$$\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}) = \mathbb{E}_{\mathbf{u}}\big[Q^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u}) \nabla_{\theta} \log \phi(\mathbf{u}; \mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), \boldsymbol{\Sigma})\big]$$

where the expectation is with respect to $\mathbf{u} \sim \mathcal{N}(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), \boldsymbol{\Sigma})$.

Due to the policy gradient theorem, we can use stochastic gradient descent to update the neural network parameters:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t Q^{\boldsymbol{\theta}_t}_{\infty}(\mathbf{x}_t, \mathbf{u}_t) \nabla_{\boldsymbol{\theta}_t} \log \phi(\mathbf{u}_t; \mathbf{f}(\mathbf{x}_t; \boldsymbol{\theta}_t), \boldsymbol{\Sigma}) \quad (12)$$

where $\mathbf{x}_t \sim \mathbf{U}(\mathcal{C}_{\infty})$ is uniformly sampled within the maximal control invariant set $\mathcal{C}_{\infty}$, $\mathbf{u}_t \sim \mathcal{N}(\mathbf{f}(\mathbf{x}_t; \boldsymbol{\theta}_t), \boldsymbol{\Sigma})$, and α_t is a non-negative learning rate.

The policy gradient theorem can be generalized to include a comparison of the state-action value $Q^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u})$ to an arbitrary baseline function $b(\mathbf{x})$. The baseline can be any function, even a random variable, as long as it does not vary with $\mathbf{u}$, since the expectation above remains unchanged [42]:

$$\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}) = \mathbb{E}_{\mathbf{u}}\big[\big(Q^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u}) - b(\mathbf{x})\big) \nabla_{\theta} \log \phi(\mathbf{u}; \mathbf{f}(\mathbf{x}; \boldsymbol{\theta}), \boldsymbol{\Sigma})\big].$$

The reason for introducing the baseline is that it can have a significant effect on the *variance* of the stochastic update rule in (12). A near-optimal reduction in variance can be achieved [43] by picking $b(\mathbf{x}) = V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x})$ because the difference $A^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u}) := Q^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u}) - V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x})$, known as the *advantage function*, measures the expected contributed value of individual control inputs $\mathbf{u}$ at state $\mathbf{x}$, relative to the average expected value $V^{\boldsymbol{\theta}}_{\infty}(\mathbf{x})$. Using this variance reduction strategy and simplifying the expression in (12), we arrive at the final update rule for the neural network parameters:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t A^{\boldsymbol{\theta}_t}_{\infty}(\mathbf{x}_t, \mathbf{u}_t) [\nabla_{\boldsymbol{\theta}_t} \mathbf{f}(\mathbf{x}_t; \boldsymbol{\theta}_t)] \Sigma^{-1} (\mathbf{u}_t - \mathbf{f}(\mathbf{x}_t; \boldsymbol{\theta}_t)) \quad (13)$$

where as before $\mathbf{x}_t \sim \mathbf{U}(\mathcal{C}_{\infty})$ and $\mathbf{u}_t \sim \mathcal{N}(\mathbf{f}(\mathbf{x}_t; \boldsymbol{\theta}_t), \boldsymbol{\Sigma})$.

A key idea in our approach is to estimate the infinite-horizon advantage function $A^{\boldsymbol{\theta}}_{\infty}(\mathbf{x}, \mathbf{u})$ via the terminal cost $\mathbf{x}_N^T \mathbf{F} \mathbf{x}_N$ used by MPC. More precisely, given a state $\mathbf{x}_t \in \mathcal{C}_{\infty}$, we sample two state-control trajectories $\tau^q := \{(\mathbf{x}^q_{k+1}, \mathbf{u}^q_k)\}_{k=t}^{N-1}$ and $\tau^v := \{(\mathbf{x}^v_{k+1}, \mathbf{u}^v_k)\}_{k=t}^{N-1}$ of length N, by following the stochastic control law $\phi(\cdot; \mathbf{f}(\mathbf{x}_k; \boldsymbol{\theta}_t), \boldsymbol{\Sigma})$ and the system dynamics $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$ and ensuring that $\mathbf{u}^q_k$ and $\mathbf{u}^v_k$ are projected via (9), to estimate the advantage:

$$\begin{aligned} A^{\boldsymbol{\theta}_t}_{\infty}(\mathbf{x}_t, \mathbf{u}^q_t) \approx{} & [\mathbf{u}^q_t]^T \mathbf{R} \mathbf{u}^q_t - [\mathbf{u}^v_t]^T \mathbf{R} \mathbf{u}^v_t + [\mathbf{x}^q_N]^T \mathbf{F} \mathbf{x}^q_N - [\mathbf{x}^v_N]^T \mathbf{F} \mathbf{x}^v_N \\ & + \sum_{k=t+1}^{N-1} [\mathbf{u}^q_k]^T \mathbf{R} \mathbf{u}^q_k - [\mathbf{u}^v_k]^T \mathbf{R} \mathbf{u}^v_k + [\mathbf{x}^q_k]^T \mathbf{M} \mathbf{x}^q_k - [\mathbf{x}^v_k]^T \mathbf{M} \mathbf{x}^v_K \end{aligned} \quad (14)$$

Exploiting the terminal cost $\mathbf{F}$ and the system dynamics to compute the advantage function significantly increases training efficiency. Moreover, our policy gradient algorithm (13) *can be applied to a nonlinear system directly* (as long as $\mathcal{C}_{\infty}$, or an appropriate subset, can be calculated), as it only depends on the assumptions that the system model and cost function are known, and not on their specific representational forms.

We make two additional modifications which we observed improved empirical performance. First, we only use $(\mathbf{x}, \mathbf{u})$ pairs with positive advantage. Updating using positive reinforcement and ignoring negative reinforcement is analogous to other techniques such as stochastic hill-climbing [44], *Continuous Actor Critic Learning Automaton (CACLA)* [45], and the linear reward-inaction update [46]. This sampling can easily be performed using rejection sampling.

Second, we integrate policy gradients into a curriculum learning framework. Curriculum learning is based on numerical continuation methods for optimizing complex non-convex functions [47], which define a family of cost functions V_η for $\eta = 0 \ldots N$ where V_0 can be optimized easily and V_N is the actual performance criteria to minimize. Applying a continuation method involves tracking the minimizing solution of increasingly non-smooth cost functions as η goes from 0 to N, and has been empirically shown to yield better solutions, especially in optimizing DNNs [48]. In our application, V_0 corresponds to the value function of the unconstrained LQR solution projected onto the polytope via (9). V_N corresponds to the value function of the optimal N-horizon control law in Problem (2). The entire curriculum policy gradient for constrained LQR is presented in Alg. 1.

Algorithm 1 Constrained LQR Curriculum Policy Gradient

```
 1: procedure POLICY GRADIENT ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{M}$, $\mathbf{R}$, $\mathbf{F}$, $\mathbf{K}$, $N$)
 2:   Randomly initialize $\theta$
 3:   Compute $\mathcal{C}_{\infty}$
 4:   Set learning rate $\alpha$, batch size $N_B$, sample size $N_S$
 5:   $t \leftarrow 0$
 6:   repeat                                  ▷ Initial Fit to Projected LQR
 7:     Sample batch $N_B$ of $\mathbf{x}^{(i)} \sim \mathbf{U}(\mathcal{C}_{\infty})$
 8:     For each $\mathbf{x}^{(i)}$, compute polytope $\mathcal{R}(\mathbf{x}^{(i)})$
 9:     $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_t \nabla_{\boldsymbol{\theta}_t} \left[\frac{1}{N_B} \sum_{i=1}^{N_B} ||\mathcal{P}^{\mathcal{R}(\mathbf{x}^{(i)})}(-\mathbf{K}\mathbf{x}^{(i)}) - f(\mathbf{x}^{(i)}; \boldsymbol{\theta}_t)||_2^2\right]$
10:     $t \leftarrow t + 1$
11:   until convergence
12:   for $\eta = 1 \ldots N$ do                     ▷ Curriculum
13:     repeat
14:       repeat                              ▷ Rejection Sampling
15:         $\mathcal{D} \leftarrow \{\}$
16:         Sample batch $N_S$ of $(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}) \sim [\mathbf{U}(\mathcal{C}_{\infty}) \times \mathcal{N}\big(\mathbf{f}(\mathbf{x}^{(i)}; \boldsymbol{\theta}_t), \boldsymbol{\Sigma}\big)]$
17:         For each $\mathbf{x}^{(i)}$, compute polytope $\mathcal{R}(\mathbf{x}^{(i)})$
18:         Compute $A^{\boldsymbol{\theta}_t}_{\infty}(\mathbf{x}, \mathbf{u})$ according to (14)
19:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}) | A^{\boldsymbol{\theta}_t}_{\infty}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)}) > 0\}$
20:       until $|\mathcal{D}| \geq N_B$
21:       Apply update rule (13)
22:       $t \leftarrow t + 1$
23:     until convergence
      return $\boldsymbol{\theta}_t$
```

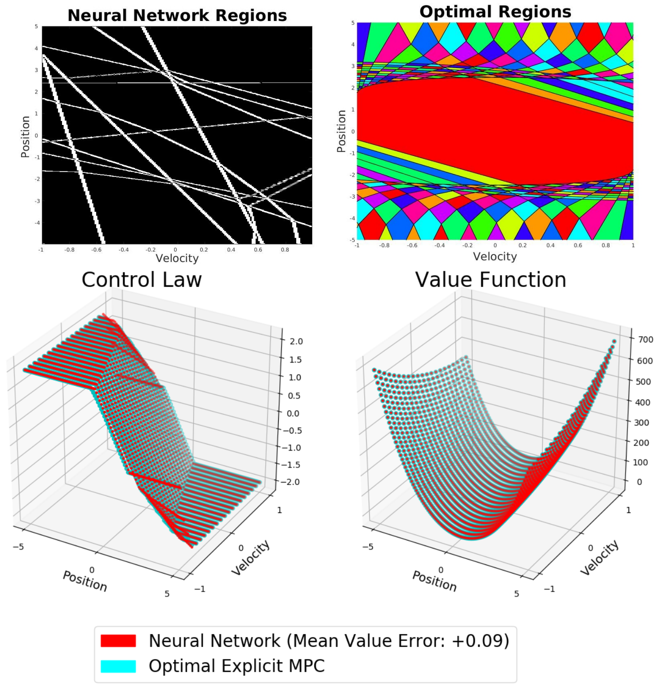

Fig. 2: **Top Left**: Neural Network Regions; **Top Right**: Optimal Explicit MPC Regions; The general shape of the explicit MPC regions can be seen in the neural network solution **Bottom Left**: Control Law Comparison; **Bottom Right**: Value Function Comparison; Our network is able to closely approximate the optimal control law.

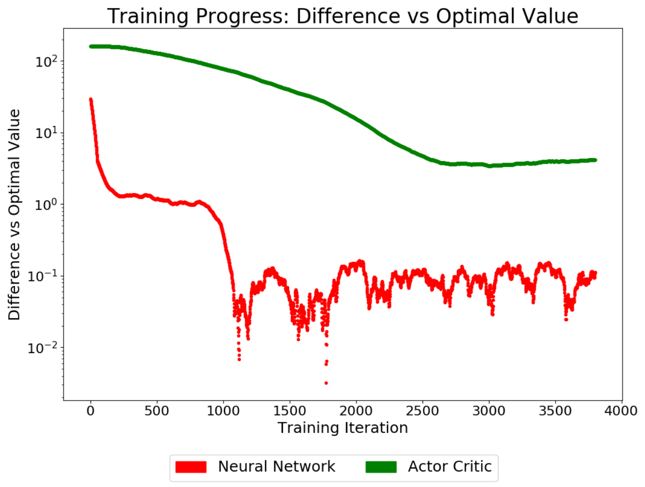

Fig. 3: **Neural Network**: Our approach; **Actor Critic** Same control law network architecture trained using actor critic algorithm. Our method plateaus after 1000 iterations, while the actor critic method plateaus after 3000 iterations. The resulting control law learned by our method has better performance as measured by the difference in value compared to the optimal value. The drops in this error around training iterations 200 and 1000 in the neural network method is due to the curriculum schedule.

VI. Numerical Examples

The examples are intended as a proof of concept and to check the quality of the approximation algorithm. We did not make any effort to optimize the offline and online computation speeds.

A. Double Integrator

Consider a 2-D double integrator system:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \epsilon\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \frac{\epsilon^2}{2}\mathbf{I} \\ \epsilon\mathbf{I} \end{bmatrix}$$

where $\mathbf{I} = 1$, $\mathbf{x} \in \mathbb{R}^2, \mathbf{u} \in \mathbb{R}$, and $\epsilon = 0.1$ is a time discretization parameter. We are interested in stabilizing the system by solving the MPC problem in (2) with cost terms $\mathbf{R} = 1$, $\mathbf{M} = \mathbf{I}_2$, horizon $N = 15$, position constraints $|x_k^{(1)}| \leq 6$, velocity constraints $|x_k^{(2)}| \leq 1,\ k = 0, \dots, N$ and input constraint $|u_k| \leq 2,\ k = 1, \dots, N-1$.

Using prior knowledge that the optimal control law has 439 regions, we construct a neural network with 2 hidden layers of width 8. Since the input size $n_0 = 2$, according to Eqn. (7), the lower bound on the maximal number of polytope regions this neural network can compute is 576.

Fig. 2 compares the proposed policy gradient method to the optimal explicit MPC solution in terms of the computed control law, value function, and the polytopic regions that define the piecewise affine control law structure. Even though the network is small, with only 16 nodes, it is able to closely approximate the optimal solution. The plot of the regions defining the neural network control law is illustrated in Fig. 2 and indicates that the neural network ignores saturated regions at the top and bottom of the optimal regions plot. These saturated regions correspond to regions which add additional complexity to the control law, but have no effect on the performance. The extra lines in the neural network regions plot, such as the ones through the large center region, could cause approximation error, but the effects are close to negligible as shown by the close match between the optimal and approximate control laws and corresponding value functions. The regions of the neural network can be visualized by sampling on a dense grid, and plotting where changes in the gradient of the neural network with respect to the inputs occur.

Since one of our contributions is in improving the policy gradient algorithm by computing the finite horizon advantage in (14), we quantify the impact of this change by comparing our algorithm against an actor-critic algorithm based on *A3C* [49]. The A3C algorithm approximates the advantage function rather than computing it exactly based on the system model. The same neural network architecture is used to represent the control law in the actor-critic method, while a second neural network is used to approximate the value function. The value network has 3 hidden layers of width 64 and is trained via standard techniques of minimizing a temporal difference error.

Fig. 3 compares the control values generated by each method during training with the optimal value obtained from the MPC controller. Our method learns significantly faster and results in a better control law (mean value difference of $+0.09$) in 1000 iterations, while the actor-critic method converges to a control law with worse performance ($+5.35$) after 3000 iterations. These results indicate that computing a finite-horizon advantage based on the system model impacts both the training efficiency and the quality of the resulting neural network controller.

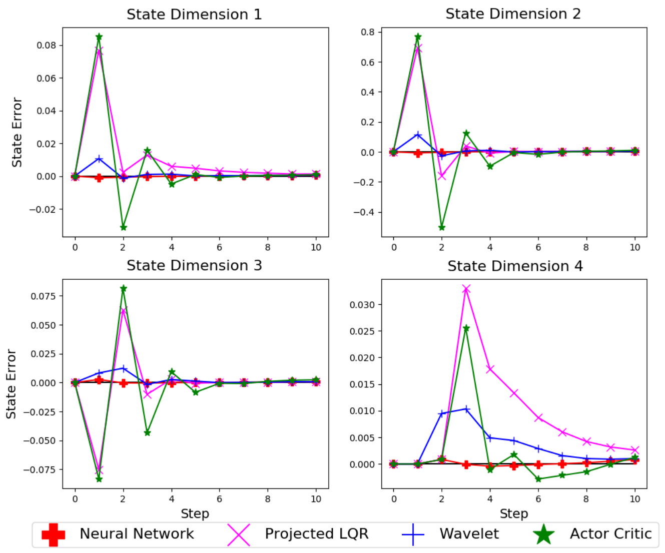

Fig. 4: **Example Trajectory**: These plots depict the state error for each method compared to optimal implicit MPC on a sample trajectory. The corresponding value errors are Neural Network (0.11), Projected LQR (1.82), Wavelet (1.16), and Actor Critic (2.48).

B. 4-Dimensional System

We consider a second numerical example that compares our method to other approximation methods for explicit MPC. Consider the following system with 4 state dimensions and 2 action dimensions (from [11]):

$$\mathbf{A} = \begin{bmatrix} 0.7 & -0.1 & 0.0 & 0.0 \\ 0.2 & -0.5 & 0.1 & 0.0 \\ 0.0 & 0.1 & 0.1 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.5 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 1.0 \\ 0.1 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

with constraints given by the inequalities:

$$|\mathbf{x_k}| \leq \begin{bmatrix} 6.0 \\ 6.0 \\ 1.0 \\ 0.5 \end{bmatrix}, \quad |\mathbf{u_k}| \leq \begin{bmatrix} 5.0 \\ 5.0 \end{bmatrix},$$

costs parameters $\mathbf{R} = \mathbf{I}$, $\mathbf{M} = \mathbf{I}$, and time horizon $N = 10$. We compare our neural network method with: (1) Projected LQR; (2) Wavelet; and (3) Actor Critic.

The Projected LQR control law uses Dykstra's projection algorithm to project the unconstrained LQR control law onto the polytope defined by the constraints in (9). Our initial curriculum learning step fits the neural network to projected LQR, so improvement beyond this control law is due to the policy gradient optimization. The wavelet method [10] splits the feasible region into a grid of hypercubes, and stores the value and control law at each vertex. However, some of the vertices of the hypercube may lie outside the feasible region and have no solution, and as a result the approximate control law may not feasible [10, Lemma 3]. Thus, it is necessary to formulate the MPC problem using soft-constraints when using the wavelet approach, and the resulting control law may violate the constraints. Our third benchmark is the actor-critic method introduced in Sec.VI-A.

Error	Avg %	Max %	SD %	Fail %
Neural Network	**1.7**	**10.5**	**1.4**	**0**
Projected LQR	2.4	55.9	5.6	0
Wavelet	15.7	157.9	25.6	8.3
Actor Critic	8.4	89.8	9.5	0

TABLE I: **Value Error Metrics**: These statistics are conditional on the approximate control law computing a feasible sequence of control inputs. Due to Dykstra's projection algorithm, the Neural Network, Projected LQR, and Actor Critic are guaranteed to generate feasible control inputs. However, due to its soft-constrained formulation, the Wavelet method failed to find a feasible trajectory for 83 out of the 1000 sampled states.

Our neural network controller has 3 hidden layers with 16 nodes in each layer. The wavelet method controller has 3256 hierarchical details. The value network in the actor critic method has 3 hidden layers with 64 nodes. The computation of the optimal explicit MPC controller for a system of this size is already computationally burdensome, so instead we randomly select 1000 states from $\mathcal{C}_\infty$, and solve for the optimal controller using implicit MPC. Fig. 4 shows an example trajectory of each method compared to optimal implicit MPC, and Table I shows the error statistics. Our neural network approach performs the best, with an average value error of 1.7% from the optimal value, and maximum error of 10.5%. It significantly improves over projected LQR, especially at initial states for which the optimal value is high, indicating that the policy gradient approach is effective in training the network. Due to the soft constraint formulation, the wavelet method fails to output a feasible solution for 83 out of the 1000 examples. The actor-critic method converges to a control law that has similar value to the projected LQR control law, but does not improve beyond that.

VII. Conclusion

This paper presented a deep reinforcement learning approach for synthesizing an approximate explicit MPC control law. We extended the DNN ReLU network structure by using Dykstra's projection algorithm to guarantee constraint satisfaction. In addition, we proposed a policy gradient algorithm that explicitly takes the known system model into account when calculating the advantage to determine the gradient direction. We showed in simulation that this increases the training efficiency and quality of the resulting control law. Finally, we incorporated the idea of curriculum learning using the closed form unconstrained LQR solution to initialize the network and incrementally train on increasing time horizons. Empirically, this not only decreases training time but also increases accuracy by helping avoid local minima which exist in the parameter space.

There are a few limitations with our current work. First, we do not provide measures of the sub-optimality of the resulting control law synthesized by our method. In addition, there are no guarantees or considerations about stability. These issues have previously been noted as drawbacks of function approximation methods [12]. Future work will focus on addressing these issues, as well as extending our algorithm to more complicated scenarios such as higher dimensional or non-linear systems.

References

[1] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[2] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 3235–3240, 2015.

[3] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 279–284.

[4] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," in *Robotics Research*. Springer, 2018, pp. 325–341.

[5] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *IEEE Int. Conf. on Humanoid Robots*, 2013, pp. 292–299.

[6] M. Kvasnica and M. Fikar, "Clipping-based complexity reduction in explicit mpc," *IEEE Trans. on Auto. Control*, vol. 57, no. 7, pp. 1878–1883, 2012.

[7] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, 2010.

[8] S. Richter, C. N. Jones, and M. Morari, "Real-time input-constrained mpc using fast gradient methods," in *IEEE Conf. on Decision and Control (CDC)*, 2009, pp. 7387–7393.

[9] T. A. Johansen and A. Grancharova, "Approximate explicit constrained linear model predictive control via orthogonal search tree," *IEEE Trans. on Auto. Control*, vol. 48, no. 5, pp. 810–815, 2003.

[10] S. Summers, C. N. Jones, J. Lygeros, and M. Morari, "A multiresolution approximation method for fast explicit model predictive control," *IEEE Trans. on Auto. Control*, vol. 56, no. 11, pp. 2530–2541, 2011.

[11] C. N. Jones and M. Morari, "Polytopic approximation of explicit model predictive controllers," *IEEE Trans. on Auto. Control*, vol. 55, no. 11, pp. 2542–2553, 2010.

[12] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Trans. on Auto. Control*, vol. 56, no. 12, pp. 2883–2897, 2011.

[13] M. Kvasnica, J. Hledík, I. Rauová, and M. Fikar, "Complexity reduction of explicit model predictive control via separation," *Automatica*, vol. 49, no. 6, pp. 1776–1781, 2013.

[14] B. Takács, J. Holaza, M. Kvasnica, and S. Di Cairano, "Nearly-optimal simple explicit mpc regulators with recursive feasibility guarantees," in *IEEE Conf. on Decision and Control (CDC)*, 2013, pp. 7089–7094.

[15] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, pp. 1443–1451, Oct. 1995.

[16] M. Lawryńczuk, "Accuracy and computational efficiency of suboptimal nonlinear predictive control based on neural models," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2202–2215, Mar. 2011.

[17] M. Lazar and O. Pastravanu, "A neural predictive controller for non-linear systems," *Mathematics and Computers in Simulation*, vol. 60, no. 3, pp. 315–324, 2002.

[18] G. Liu and V. Kadirkamanathan, "Predictive control for non-linear systems using neural networks," *Int. Journal of Control*, vol. 71, no. 6, pp. 1119–1132, 1998.

[19] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Athena Scientific, 1996.

[20] S. J. Bradtke, "Reinforcement learning applied to linear quadratic regulation," in *Advances in neural information processing systems*, 1993, pp. 295–302.

[21] K. G. Vamvoudakis, "Q-learning for continuous-time linear systems: A model-free infinite horizon optimal control approach," *Systems & Control Letters*, vol. 100, pp. 14–20, 2017.

[22] J. E. Morinelly and B. E. Ydstie, "Dual mpc with reinforcement learning," *IFAC-PapersOnLine*, vol. 49, no. 7, pp. 266–271, 2016.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. of the 32nd Int. Conf. on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[25] S. Levine and V. Koltun, "Guided policy search," in *Int. Conf. on Machine Learning (ICML)*, 2013, pp. 1–9.

[26] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[27] D. Chmielewski and V. Manousiouthakis, "On constrained infinite-time linear quadratic optimal control," *Systems & Control Letters*, vol. 29, no. 3, pp. 121 – 129, 1996.

[28] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[29] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear model predictive control*. Springer, 2009, pp. 345–369.

[30] N. Gaffke and R. Mathar, "A cyclic projection algorithm via duality," *Metrika*, vol. 36, no. 1, pp. 29–54, 1989.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[32] R. Pascanu, G. Montufar, and Y. Bengio, "On the number of response regions of deep feed forward networks with piece-wise linear activations," *arXiv preprint arXiv:1312.6098*, 2013.

[33] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," *arXiv preprint arXiv:1611.01491*, 2016.

[34] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[35] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," *arXiv preprint arXiv:1703.00443*, 2017.

[36] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233–268, 2017.

[37] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *Proc. of the International Conference on Learning Representations (ICLR)*, May 2016.

[38] J. Dattorro, *Convex optimization & Euclidean distance geometry*. Meboo Publishing, 2010.

[39] F. Mirko and A. Mazen, "Computing control invariant sets is easy," *arXiv preprint arXiv:1708.04797*, 2017.

[40] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Info. Proc. Sys. (NIPS)*, 2000, pp. 1057–1063.

[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

[43] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *CoRR*, vol. abs/1506.02438, 2015. [Online]. Available: http://arxiv.org/abs/1506.02438

[44] S. Russell and P. Norvig, *Artificial Intelligence: A modern approach*. Prentice-Hall, 1995.

[45] H. Van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning*. Springer, 2012, pp. 207–251.

[46] K. S. Narendra and M. A. L. Thathachar, "Learning automata - a survey," *IEEE Trans. Systems, Man., Cybernetics*, pp. 323–334, 1974.

[47] E. Allgower and K. Georg, "Simplicial and continuation methods for approximating fixed points and solutions to systems of equations," *SIAM Review*, vol. 22, no. 1, pp. 28–85, 1980.

[48] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. on Machine Learning*. ACM, 2009, pp. 41–48.

[49] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *ArXiv e-print:1602.01783*, 2016.