Deep Neural Network Approximation of Nonlinear Model Predictive Control [⋆]

Yankai Cao [*] **R. Bhushan Gopaluni** [*]

[*] *Department of Chemical and Biological Engineering, Vancouver, BC, Canada (e-mail:yankai.cao@ubc.ca;bhushan.gopaluni@ubc.ca.)*

Abstract: This paper focuses on developing effective computational methods to enable the real-time application of model predictive control (MPC) for nonlinear systems. To achieve this goal, we follow the idea of approximating the MPC control law with a Deep Neural Network (DNN). To train the deep neural network offline, we propose a new "optimize and train" method that combines the steps of data generation and neural network training into a single high-dimensional stochastic optimization problem. This approach directly optimizes the closed loop performance of the DNN controller over a finite horizon for a number of initial states. The large-scale optimization problem can be solved efficiently using parallel computing techniques. The benefits of this approach over the conventional "optimize then train" protocol is illustrated through numerical results.

Keywords: Model Predictive Control, Stochastic Optimization, Deep Neural Networks, Nonlinear Systems

1. INTRODUCTION

Optimal real-time operation of a variety of industrial systems (e.g., power networks, buildings, batteries, wind turbines) requires control of nonlinear systems that are typically subjected to a number of uncertain factors (e.g., markets, weather, demands, and equipment failures). Conventional single-loop low level controllers (e.g., PID controllers) are easy to be deployed in real-time operations, however, their performance is often suboptimal. Advanced control algorithms, such as model predictive control (MPC), provide a general framework for controlling nonlinear systems under uncertainty. However, the high computational latency of such approaches and the lack of sufficiently robust nonlinear programming (NLP) solvers, limits the scope of their applications (especially for systems with fast dynamics).

One way to enable the real-time application of MPC is based on the idea of *explicit MPC* (Bemporad et al., 2002b,a; TøNdel et al., 2003). Explicit MPC computes the optimal control law offline as a function of all possible states in different regions using multi-parametric optimization. The online computational cost is limited to determining the region to which the current state of the system belongs and then apply a pre-determined control law. However, the offline computational cost grows exponentially with the number of constraints and the size of prediction/control horizons. Therefore, explicit MPC is often computationally intractable for large systems. We can mitigate this issue by eliminating redundant regions of states (Geyer et al., 2008; Kvasnica et al., 2013) or by identifying suboptimal partitions of regions (Johansen and Grancharova, 2003; Summers et al., 2011).

An alternative to explicit MPC is to approximate the explicit control law with a neural network (Parisini and Zoppoli, 1995; Csekő et al., 2015; Chen et al., 2018b; Kumar et al., 2018; Spielberg et al., 2019). The hypothesis of this approach is that a deep learning neural network can sufficiently approximate the nonlinear behavior of MPC, but at an online computational cost that is significantly better than that of a full-fledged MPC. This is based on the observation that neural networks with a sufficient number of neurons and layers can approximate any nonlinear mapping with arbitrary accuracy (Hornik et al., 1990). For linear systems with quadratic cost function, the optimal control law is piecewise affine on polytopes. A neural network with rectified linear units as activation functions can exactly represent piecewise affine optimal control laws. Karg and Lucia (2018) provides a conservative upper bound on the width and depth of a neural network required to approximate the explicit MPC.

An essential aspect of the above-mentioned approaches is the training of large dimensional neural networks. Many researchers follow the "optimize then train" protocol. The idea in these algorithms is to simulate the optimal controller, generate corresponding state and optimal control action data pairs and then use these data to train a large dimensional neural network via supervised learning to approximate the original optimal controller. This conventional "optimize then train" protocol is not guaranteed to work on nonlinear systems. For nonlinear systems, it is possible that multiple optimal control actions exit for the same initial state. Even if the uniqueness of optimal control actions is guaranteed, the optimal control problem might have multiple local optimal control actions, which are typically identified using local solvers due to the prohibitive computational cost of global solvers. The non-uniqueness of the state-action data pairs makes it

[⋆] yankai.cao@ubc.ca;bhushan.gopaluni@ubc.ca challenging to learn the underlying nonlinear mapping from states to optimal control actions, as illustrated in an example in Section 2. Moreover, for general nonlinear systems, no neural network structure with finite neurons can guarantee an exact representation of the MPC control laws. The training error of the neural network can lead to sub-optimal control actions, and the resulting errors could potentially accumulate over time. Thus, eventually making the neural network deviate significantly from the optimal controller. Some researchers trained the neural network with reinforcement learning (Bradtke, 1993; Vamvoudakis, 2017; Chen et al., 2018a). A limitation of these approaches is that they cannot explicitly tackle constraints.

We propose a new "optimize and train" method that combines the steps of data generation and neural network training into one single stochastic optimization problem. This approach directly optimizes the closed loop performance of the DNN controller over a number of possible initial states. The critical challenge arising in this approach is solving a large-scale nonlinear stochastic optimization problem. However, tremendous progress has been made recently in the field of stochastic optimization that allows us to design and implement efficient local and global algorithms. For instance, recent work by Cao et al. (2018) showed that the nonlinear stochastic optimization problem, arising from designing control systems for wind turbines, has approximately 7.5 million variables, and it can be solved in less than 1.3 hours using parallel solvers.

The paper is organized as follows: Section 2 introduces basic nomenclature and the conventional approach to train a DNN controller. Section 3 introduces a new "optimize and train" method that combines the steps of data generation and neural network training into one single optimization problem. Section 4 discusses the computational aspects of the new approach. Section 5 illustrates the numerical performance of the proposed algorithm on an example. The paper closes with final remarks and directions of future work in Section 6.

2. "OPTIMIZE THEN TRAIN" METHOD

MPC is an advanced control strategy that repeatedly solves the following optimal control problems $\mathbb{P}_N(x_0)$ at each sample time step with the updated initial states x_0 and prediction horizon N

$$\min_{x(k),u(k)} \quad \sum_{k\in\mathcal{T}} l(x(k),u(k)) + V_f(x(N)) \tag{1a}$$

$$\text{s.t.} \quad x(k+1) = f(x(k),u(k)) \tag{1b}$$

$$x(0) = x_0 \tag{1c}$$

$$x(k)\in\mathbb{X}, x(N)\in\mathbb{X}_f, u(k)\in\mathbb{U} \tag{1d}$$

$$\forall k\in\mathcal{T} \tag{1e}$$

where $\mathcal{T} := [0,\ldots,N-1]$ is time step set, $x\in\mathbb{R}^{n_x}$ are the state variables, $u\in\mathbb{R}^{n_u}$ are the controls, $f(.)$ represents the nonlinear process dynamics, $l(.)$ is the stage cost function, V_f is the terminal cost function, $\mathbb{X}$ and $\mathbb{X}_f$ denote diverse state constraints, $\mathbb{U}$ denotes input constraints. The optimal control actions of this optimization problem are denoted as $u(x_0) = (u(0;x_0), u(1;x_0),\ldots,u(N-1;x_0))$. Due to the receding horizon nature of MPC, only the control action of the first step $u(0;x_0)$ is applied. Therefore, the MPC control law is defined by $\kappa_N(x_0) = u(0;x_0)$.

Although it is often computationally intractable to extract an explicit MPC control law $\kappa_N(.)$, evaluation of $\kappa_N(x_0)$ for a specific value of the x_0 is computationally inexpensive. The central idea of "optimize then train" method is to first generate a number of initial state scenarios $x_{0,s}$, where $s\in\mathcal{S} := [1,\ldots,S]$ is a scenario set and S is the number of scenarios. We then obtain $\kappa_N(x_{0,s}) = u(0;x_{0,s})$ by solving the corresponding optimal control problem $\mathbb{P}_N(x_{0,s})$. These generated data pairs $(x_{0,s},\kappa_N(x_{0,s}))$ are used to train a neural network that approximates the MPC control law $\kappa_N(.)$ via supervised learning. One challenge with this approach is that, for nonlinear systems, multiple optimal control actions might exist for the same initial state values. In this case, the MPC control law $\kappa_N(.)$ is a set-valued function, and the MPC controller randomly selects one element from the set. However, approximating set-valued functions using samples is challenging. The following example illustrates the problem of "optimize then train" Method caused by the nonuniqueness in the data pairs. It is also worth mentioning that even if the uniqueness of optimal control actions is guaranteed, the optimal control problem might have multiple local optimal control actions, and we typically use local solvers because of the prohibitive computational cost of global solvers.

Example 1. Approximating Set-Valued MPC Control Law. Consider an illustrative example of the following form:

$$\begin{aligned}\min_{x(k),u(k)} \quad & \sum_{k=0}^{1} x(k)^2 \\ \text{s.t.} \quad & x(1) = x(0)^2 - u(0)^2 \\ & x(0) = x_0\end{aligned} \tag{2}$$

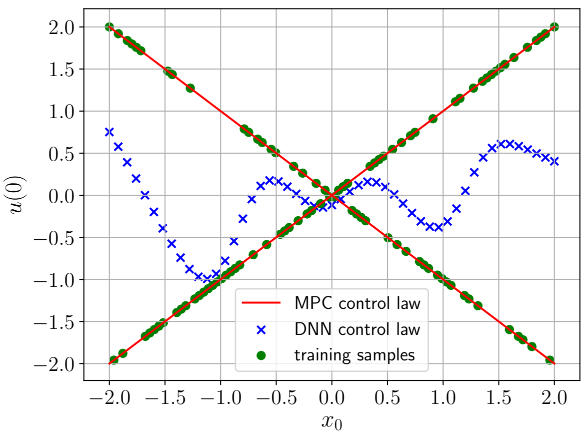

Fig. 1. DNN control law trained with "optimize then train" method

Example 1 shows a case study that is so simple that MPC can drive the state to the origin in one step. In this case, terminal cost, terminal constraints, or a long prediction horizon is not necessary. The MPC control law can be computed analytically $\kappa_N(x_0) = \pm x_0$. We generated 100 samples by solving the corresponding optimal control problems. The NLP solver chose either positive or negative optimal control action depending on the initial guess. Figure 1 shows the control law trained with the "optimize then train" method. The neural network control law has one input x_0, one output $u(0)$, and one hidden layer with 20 neurons. It is obvious that the trained control law, shown in Figure 1, is significantly different from the MPC control law. In fact, the DNN control law drives the states to infinity very quickly. We highlight that this problem can not be mitigated by increasing the number of samples or choosing a different neural network architecture.

3. "OPTIMIZE AND TRAIN" METHOD

We propose to train the DNN controller to approximate the MPC control law by solving the following optimization problem:

$$\min_{\pi, x_s(k), u_s(k)} \quad \sum_{s\in\mathcal{S}}\sum_{k\in\mathcal{T}} l(x_s(k),u_s(k)) + V_f(x_s(N)) \tag{3a}$$
$$\text{s.t.} \quad x_s(k+1) = f(x_s(k),u_s(k)) \tag{3b}$$
$$u_s(k) = \mu(\pi, x_s(k)) \tag{3c}$$
$$x_s(0) = x_{0,s} \tag{3d}$$
$$x_s(k) \in \mathbb{X}, x_s(N) \in \mathbb{X}_f, u_s(k) \in \mathbb{U} \tag{3e}$$
$$\forall s \in \mathcal{S}, \forall k \in \mathcal{T} \tag{3f}$$

where $x_{0,s}$ is a possible realization of the initial states x_0 with $s \in \mathcal{S}$, $\mu(.)$ represents the neural network controller, and π denotes the controller settings (parameters of the neural network). This formulation optimizes the closed-loop performance of the DNN control law over the prediction horizon N for a set of initial state scenarios. We highlight that by removing the DNN control policy 3c, one obtains a set of optimal control problems $\mathbb{P}_N(x_{0,s})$ for each initial state scenario. Consequently, one can think of this approach as a restricted (conservative) form of MPC. The level of conservativeness can be relaxed by increasing the number of layers, the number of neurons per layer, and the number of scenarios.

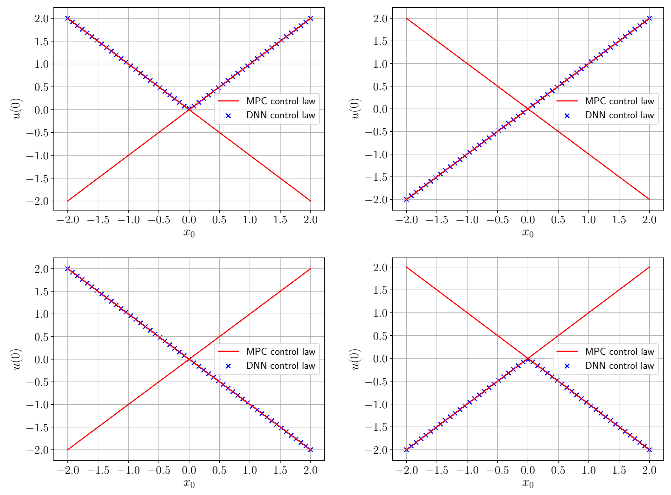

Fig. 2. DNN control law trained with "optimize and train" method

Instead of first optimizing control actions and then learning the control laws from data pairs between initial states and control actions as the conventional "optimize then train" approach does, the new "optimize and train" approach directly optimizes the control laws by solving only one large-scale optimization problem. Directly optimizing the control law avoids the difficulty caused by the non-uniqueness of optimal control actions. Even if the MPC control policy is set-valued, this approach learns a function approximating a subset of $\kappa_N(.)$ (DNN provides a one-to-one mapping). Figure 2 illustrates the DNN controller laws learned using this approach for Example 1. The number of scenarios and neural network structure are kept the same as those used with the "optimize then train" approach. Problem 3 has multiple optimal solutions for this example. The NLP solver runs with multiple random initial guesses. We obtained four different DNN control laws. Each control law is a good approximation to a subset of $\kappa_N(.)$ and provides closed-loop performance very close to the MPC controller.

Another feature of our approach is that it directly optimizes the closed loop performance over the following N steps since the control policies of the following steps are explicitly handled in the formulation. Because the control actions are trained at every step within the prediction horizon as a function of different state variables, we expect that this approach would require significantly fewer samples for training.

The last feature of this approach is that all constraints are guaranteed to be satisfied for all initial state scenarios considered in the optimization. We call these initial state scenarios used in the optimization, training scenarios. We highlight that this guarantee does not hold for the "optimize then train" method because the training error of the DNN can lead to sub-optimal or even infeasible control actions, even for training scenarios.

It is clear that the optimization formulation does not account for all possible initial states, instead only a small subset are considered. How can we ensure constraints satisfaction for all possible initial states? If the input constraints are defined by a box set $\{u|\underline{u} \leq u(k) \leq \bar{u}\}$, their satisfaction can be guaranteed by appropriate design of the activation functions in the last layer of the DNN. In addition, after obtaining the controller law $\mu(\pi, .)$ from the optimization, we can check if all constraints are satisfied using this control law for all initial states $x(0) \in \mathbb{X}_0$ by solving the following optimization problem:

$$C_v(\pi) = \max_{x(0)\in\mathbb{X}_0, x(k), u(k)} \|[g(x)]_+\| \tag{4a}$$
$$\text{s.t.} \quad x(k+1) = f(x(k),u(k)) \tag{4b}$$
$$u(k) = \mu(\pi, x(k)) \tag{4c}$$
$$\forall k \in \mathcal{T} \tag{4d}$$

where $[z]+ := \max\{z, 0\}$ and we assume that the state constraints $x(k) \in \mathbb{X}, x(N) \in \mathbb{X}_f$ have the general form $g(x) \leq 0$. This formulation finds the initial states that maximize the constraint violations resulting from applying control law $\mu(\pi, .)$. If $C_v(\pi) = 0$, then it means that the control law $\mu(\pi, .)$ will not cause any constraint violation for any initial state. In other words, the control law can drive any state $x_0 \in \mathbb{X}_0$ to $\mathbb{X}_f$ in N steps. Assume that we can obtain a local control law $\mu_f(x_0) = Kx_0$ that can stabilize any states in $\mathbb{X}_f$, then it is obvious that the DNN control law is stable. If $C_v(\pi) > 0$, it implies that the control law $\mu(\pi, .)$ will cause constraint violations, and the above formulation provides the initial state/scenario leading to the largest constraint violations. We can add this scenario to scenario set $\mathcal{S}$ and resolve the optimization formulation 3. Another way to improve the constraints satisfaction is to choose tighter state constraints in the optimization (e.g., solving formulation 3 with $g(x) \leq -\epsilon$), where ϵ is a positive margin.

4. COMPUTATIONAL METHODS

The key challenge arising from the "optimize and train" method is the solution of a large-scale nonlinear optimization problem 3. By noticing that the only variables linking different scenarios are the parameters of the DNN, problem 3 can be cast as a structured NLP of the form:

$$\min_{\pi, y_s} \sum_{s \in \mathcal{S}} g_s(\pi, y_s) \tag{5a}$$

$$\text{s.t. } c_s(\pi, y_s) = 0, \quad s \in \mathcal{S} \tag{5b}$$

$$y_s \geq 0, \quad s \in \mathcal{S} \tag{5c}$$

Here, π is the controller settings and y_s represents scenario variables (associated with the states $x_s(k)$, control variables $u_s(k)$, and auxiliary variables). General inequality constraints can be transformed to this form by introducing auxiliary variables. Problem 5 has the same structure as the stochastic optimization problems, in which π is called first-stage variables and y_s is called second stage variables. Therefore the structure of problem 5 can be exploited by our existing solvers for two-stage stochastic optimization problems.

If we solve the problem 5 following interior point methods, the search steps are computed by solving linear systems. A key observation is that the linear systems derived using interior point methods have the block-bordered-diagonal form (Kang et al., 2014; Zavala et al., 2008; Chiang et al., 2014):

$$\left[\begin{array}{c|cccc} K_\pi & B_1^T & B_2^T & \dots & B_S^T \\ \hline B_1 & K_1 & & & \\ B_2 & & K_2 & & \\ \vdots & & & \ddots & \\ B_S & & & & K_S \end{array}\right] \begin{bmatrix} \Delta\pi \\ \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_S \end{bmatrix} = - \begin{bmatrix} r_\pi \\ r_1 \\ r_2 \\ \vdots \\ r_S \end{bmatrix}, \tag{6}$$

where $\Delta w_s = (\Delta y_s, \Delta\lambda_s)$ are the Newton steps,

$$K_\pi = W_\pi, \tag{7a}$$

$$K_s = \begin{bmatrix} W_s & J_s^T \\ J_s & \end{bmatrix}, \tag{7b}$$

$$B_s = \begin{bmatrix} Q_s \\ T_s \end{bmatrix}, \tag{7c}$$

$J_s = \nabla_{y_s} c_s(\pi, y_s)$, $T_s = \nabla_\pi c_s(\pi, y_s)$, $W_\pi = \nabla_{\pi,\pi}\mathcal{L}$, $W_s = \nabla_{y_s,y_s}\mathcal{L} + \text{diag}(y_s)^{-1}\text{diag}(\lambda_s)$, $Q_s = \nabla_{\pi,y_s}\mathcal{L}$, $r_\pi = \nabla_\pi\mathcal{L}$, $r_s = \nabla_{w_s}\mathcal{L}$, $\mathcal{L}(\cdot)$ is the Lagrange function of problem 5, and λ_s are the multipliers of y_s.

Assuming that all K_s are of full rank, we can show with the Schur complement method that the solution of the Equation (6) is equivalent to that of the following system:

$$\left(K_\pi - \sum_{s \in \mathcal{S}} B_s^T K_s^{-1} B_s\right) \Delta\pi = -r_\pi + \sum_{s \in \mathcal{S}} B_s^T K_s^{-1} r_s \tag{8a}$$

$$K_s \Delta w_s = -r_s - B_s \Delta\pi, \; s \in \mathcal{S}. \tag{8b}$$

Here, $Z := K_\pi - \sum_{s \in \Omega} B_s^T K_s^{-1} B_s$ is the Schur complement matrix which has the same dimension as that of π. If some K_s are not full rank, then diagonal modification is applied. The idea of the interior point method coupled with Schur complement method has been implemented in several packages including `PIPS-NLP` Chiang et al. (2014). If the dimension of π is small, then the solution of equation (8a) is computationally cheap and the solution of equation (8b) can be performed in parallel. However, the scalability of Schur complement decomposition is significantly hindered by the dimension of π (the solution of equation (8a) induces dense linear algebra). One approach to circumvent the bottlenecks of the Schur complement decomposition is based on a clustering-based preconditioning technique that adaptively aggregates (clusters) scenarios (Cao et al., 2016). The preconditioner is designed in such a way that it fully avoids dense linear algebra operations and can thus tackle problems that Schur decomposition is unable to. This strategy has shown significant improvement in solution time when the number of coupling variables is large.

Problem 5 can be implemented using Julia-based modeling package `Plasmo.jl`(Jalving et al., 2019). `Plasmo.jl` facilitates the construction and analysis of structured optimization models. To achieve this goal, it leverages a hierarchical graph abstraction wherein nodes and edges can be associated with individual optimization models that are linked together. Given a graph structure with models and connections, `Plasmo.jl` can produce either a pure (flattened) optimization model to be solved using off-the-shelf optimization solvers such as `IPOPT` (Wächter and Biegler, 2006), or it can communicate graph structures to parallel solvers such as `PIPS-NLP` and thus enable decomposition.

```
#call libraries
using Plasmo,JuMP, Ipopt
# create two-stage model
graph = ModelGraph()
# define first-stage variables in parent node
master = Model()
add_node!(graph,master)
@variable(master,pi)

# create array of scenario models
scenm=Array(JuMP.Model,S)
for j in 1:S
    # get scenario model and append to parent node
    scenm[j] = get_scenario_model(j)
    add_node!(graph,scenm[j])
   # link children to parent variables
    @linkconstraint(graph, master[:pi] == scenm[j][:pi])
end

# solve two-stage program with PIPS-NLP
solver = PipsSolver()
# alternatively, solve two-stage program as a general NLP with IPOPT
solver = IpoptSolver()
solve(graph,solver)
```

Fig. 3. Snippet of an implementation of problem 5 in `Plasmo.jl`

The code snippet shown in Figure 4 illustrates how to implement problem (5) in `Plasmo.jl`. As can be seen, the individual scenario models are created and appended to the parent node to create a two-level graph structure. The structure is directly communicated to `PIPS-NLP` and thus the solver can execute the parallel Schur decomposition approach previously discussed. Note also that, under this modeling framework, the user does not need to have any knowledge of parallel computing. From the snippet, we also note that `Plasmo.jl` can also create a general (unstructured) NLP to be solved by off-the-shelf solvers like `IPOPT`.

5. NUMERICAL CASE STUDY

The case study we consider is a quadtank problem with the following dynamics (Raff et al., 2006):

$$\frac{dz_1}{dt} = -\frac{a_1}{A_1}\sqrt{2g(z_1 + x_{1s})} \tag{9a}$$

$$+ \frac{a_3}{A_1}\sqrt{2g(z_3 + x_{3s})} + \frac{\gamma_1}{A_1}(v_1 + u_{1s}) \tag{9b}$$

$$\frac{dz_2}{dt} = -\frac{a_2}{A_2}\sqrt{2g(z_2 + x_{2s})} \tag{9c}$$

$$+ \frac{a_4}{A_2}\sqrt{2g(z_4 + x_{4s})} + \frac{\gamma_2}{A_2}(v_2 + u_{2s}) \tag{9d}$$

$$\frac{dz_3}{dt} = -\frac{a_3}{A_3}\sqrt{2g(z_3 + x_{3s})} + \frac{(1-\gamma_2)}{A_3}(v_2 + u_{2s}) \tag{9e}$$

$$\frac{dz_4}{dt} = -\frac{a_4}{A_4}\sqrt{2g(z_4 + x_{4s})} + \frac{(1-\gamma_1)}{A_4}(v_1 + u_{1s}) \tag{9f}$$

where z_i is deviation of water level in tank i from the setpoint $x_s = [14cm\ 14cm\ 14cm\ 21.3cm]^T$, v_i is the deviation of the flow rate of pump i from the steady state value $u_s = [43.4ml/s\ 35.4ml/s]^T$, a_i and A_i are tank parameters, while γ_i are valve parameters. The objective is to control the water levels x_1 and x_2 around the setpoints. The stage cost is defined as below:

$$l = z_1^2 + z_2^2 + 0.01(v_1^2 + v_2^2). \tag{10}$$

The controller also needs to satisfy the input and state constraints at each stage

$$v_{min} \le v \le v_{max} \tag{11a}$$

$$z_{min} \le z \le z_{max} \tag{11b}$$

with $v_{min} = [-43.4ml/s\ \ -35.4ml/s]^T$, $v_{max} = [16.6ml/s\ 24.6ml/s]^T$, $z^{min} = [-6.5cm\ \ -6.5cm\ \ -10.7cm\ -16.8cm]^T$ and $z^{max} = [14cm\ 14cm\ 13.8cm\ 6.7cm]^T$.

The dynamic model was discretized using an explicit Euler discretization scheme with a prediction horizon of 20 steps and a sampling time of 3 seconds. To train the DNN, we generated 81 initial state scenarios with each state variable discretized by 3 points ($z_i(t=0)$ can take any value from $\{max, 0, min\}$). The cumulative stage cost over 20 steps is used to evaluate the performance of the controller. By using the ideal NMPC, we need to solve 1620 optimization problems. These 1620 data pairs are used to train the DNN in the "train then optimize" approach, while the "train and optimize" approach only uses the 81 initial state scenarios to train the DNN. To test the performance of the controllers, we generated 256 initial state scenarios with each state variable discretized by 4 points. There is some overlap between the training scenarios and test scenarios to highlight the importance of these scenarios with extreme initial states. All optimization problems are modelled using `JuMP` and solved using `IPOPT`. The training of DNN in the "train and optimize" method is performed via the `Julia` Package `Flux` using the adam method with 5000 epochs. Our implementation runs on a PC with Intel i7 CPU running at 2.2 GHz.

Table 1 compares the performance of different controllers. Both "train then optimize" and "train and optimize" methods consider the same DNN structure, which has 4 neurons in the input layer, 10 neurons in the first hidden layer, 2 neurons in the second hidden layer, and 2 neurons in the last layer. Both hidden layers use the activation function tanh. To guarantee the satisfaction of input constraints, the last layer projects values in the range of $[-1, 1]$ to $[v_{min}, v_{max}]$. As expected, the ideal NMPC provides the best performance. The accumulative cost of the DNN controller trained by the "train and optimize" method is very close to that of ideal NMPC. The cost increase caused by switching from ideal NMPC to DNN controller trained by the "train and optimize" method is only 0.42 (in terms of test scenarios), while the cost increase caused by switching to "train then optimize" method is 4.3. Besides that, the constraint violations caused by the "train and optimize" is only 5.8% of the constraint violations caused by the "train then optimize" method. We highlight that the "train and optimize" method can guarantee constraint satisfaction at least for training scenarios, while the "train then optimize" method cannot.

Table 1. Performance of different controllers

	training		testing	
	cost	Cons. Viol.	cost	Cons. Viol.
ideal	582.18	0	488.53	0
"train then optimize"	582.64	1.85	492.83	1.85
"train and optimize"	582.29	0	488.95	0.109

Table 2 compares the averaged online and offline computational time of different controllers. The online computational time can be reduced from 16 ms to 0.04 ms by switching from the ideal NMPC to DNN controller, resulting in a speedup of 400 times. For large scale systems, we can expect even better speedup. The huge reduction in the online computational time is at the cost of offline computational time to train the DNN. Table 1 shows that the time to train the DNN controller is reasonable for this problem. We can further reduce the offline computational time by training the DNN on GPU for "train then optimize" method, and on distributed memory HPC for "train and optimize" methods.

Table 2. Comparison of averaged online and offline computational time

	online (s)	offline (s)
ideal	0.016	-
"train then optimize"	4e-5	2134
"train and optimize"	4e-5	1194

Table 3. Performance of "train then optimize" method with different DNN layers

# of layers	training Cons. cost	training Viol. [cm]	test Cons. cost	test Cons. Viol. [cm]
2	582.64	492.83	1.8502	1.8502
4	584.40	490.26	0.599	1.008
6	583.74	491.06	1.12	1.18
8	586.26	492.23	0.259	1.240
10	582.87	489.71	0.636	0.746
12	586.08	492.84	0.513	1.250

One might argue that the inferior performance of the "train then optimize" method might be caused by the choice of a simple DNN. Table 3 compares the performance of the "train then optimize" method with different DNN layers. The number of hidden layers ranges from 2 to 12. The last hidden layer has 2 neurons, while the remaining layers all have 10 neurons per layer. This table shows that no matter how many layers we chose, the performance of the "train then optimize" method is always inferior to the performance of the "train and optimize" method in terms of both cumulative cost and constraint violations.

6. CONCLUSION

Model Predictive Controllers (MPC) are widely used in the industry on a wide range of processes including those with nonlinear and stochastic characteristics. However, the online implementation of nonlinear MPC is computationally rather challenging due to the complexity of the underlying optimization problem. We proposed a novel "All-in-One" approach to approximate a nonlinear MPC with constraints using a Deep Neural Network. This approach directly optimizes the closed loop performance of the DNN controller over a finite horizon for a number of initial states using parallel computing techniques. This approach works well even if the MPC control policy is set-valued, and it provides performance that is close to that of the original nonlinear MPC.

REFERENCES

Bemporad, A., Borrelli, F., Morari, M., et al. (2002a). Model predictive control based on linear programming —the explicit solution. *IEEE Transactions on Automatic Control*, 47(12), 1974–1985.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E.N. (2002b). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3–20.

Bradtke, S.J. (1993). Reinforcement learning applied to linear quadratic regulation. In *Advances in neural information processing systems*, 295–302.

Cao, Y., Laird, C.D., and Zavala, V.M. (2016). Clustering-Based Preconditioning for Stochastic Programs. *Computational Optimization and Applications*, 64, 379–406.

Cao, Y., Zavala, V.M., and D'Amato, F. (2018). Using stochastic programming and statistical extrapolation to mitigate long-term extreme loads in wind turbines. *Applied Energy*, 230, 1230–1241.

Chen, S., Saulnier, K., Atanasov, N., Lee, D.D., Kumar, V., Pappas, G.J., and Morari, M. (2018a). Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, 1520–1527. IEEE.

Chen, Y., Shi, Y., and Zhang, B. (2018b). Optimal control via neural networks: A convex approach. *arXiv preprint arXiv:1805.11835*.

Chiang, N., Petra, C.G., and Zavala, V.M. (2014). Structured nonconvex optimization of large-scale energy systems using pips-nlp. In *Power Systems Computation Conference (PSCC), 2014*, 1–7. IEEE.

Csekő, L.H., Kvasnica, M., and Lantos, B. (2015). Explicit mpc-based rbf neural network controller design with discrete-time actual kalman filter for semiactive suspension. *IEEE Transactions on Control Systems Technology*, 23(5), 1736–1753.

Geyer, T., Torrisi, F.D., and Morari, M. (2008). Optimal complexity reduction of polyhedral piecewise affine systems. *Automatica*, 44(7), 1728–1740.

Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5), 551–560.

Jalving, J., Cao, Y., and Zavala, V.M. (2019). Graph-based modeling and simulation of complex systems. *Computers & Chemical Engineering*, 125, 134–154.

Johansen, T.A. and Grancharova, A. (2003). Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Transactions on Automatic Control*, 48(5), 810–815.

Kang, J., Cao, Y., Word, D.P., and Laird, C.D. (2014). An interior-point method for efficient solution of block-structured nlp problems using an implicit schur-complement decomposition. *Computers & Chemical Engineering*, 71, 563–573.

Karg, B. and Lucia, S. (2018). Efficient representation and approximation of model predictive control laws via deep learning. *arXiv preprint arXiv:1806.10644*.

Kumar, S.S.P., Tulsyan, A., Gopaluni, B., and Loewen, P. (2018). A deep learning architecture for predictive control. *IFAC-PapersOnLine*, 51(18), 512–517.

Kvasnica, M., Hledík, J., Rauová, I., and Fikar, M. (2013). Complexity reduction of explicit model predictive control via separation. *Automatica*, 49(6), 1776–1781.

Parisini, T. and Zoppoli, R. (1995). A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10), 1443–1451.

Raff, T., Huber, S., Nagy, Z.K., and Allgower, F. (2006). Nonlinear model predictive control of a four tank system: An experimental stability study. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, 237–242. IEEE.

Spielberg, S., Tulsyan, A., Lawrence, N., Loewen, P., and Gopaluni, R. (2019). Towards self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal*.

Summers, S., Jones, C.N., Lygeros, J., and Morari, M. (2011). A multiresolution approximation method for fast explicit model predictive control. *IEEE Transactions on Automatic Control*, 56(11), 2530–2541.

TøNdel, P., Johansen, T.A., and Bemporad, A. (2003). An algorithm for multi-parametric quadratic programming and explicit mpc solutions. *Automatica*, 39(3), 489–497.

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

Wächter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25–57.

Zavala, V.M., Laird, C.D., and Biegler, L.T. (2008). Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chemical Engineering Science*, 63(19), 4834–4845.