2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Deep learning-based embedded mixed-integer model predictive control

Benjamin Karg and Sergio Lucia

Abstract— **We suggest that using deep learning networks to learn model predictive controllers is a powerful alternative to online optimization, especially when the underlying problems are complex, as in the case of mixed-integer quadratic programs.**

The use of deep learning has two important advantages compared to classical shallow networks regarding its embedded implementation. A better function approximation can be achieved with the same number of neurons and less weights are necessary due to the use of more, but smaller, layers. This reduces significantly the memory footprint of the necessary code for its embedded implementation. As with shallow networks, deep neural networks are extremely easy to implement and to deploy on embedded platforms.

The potential of the approach is illustrated with simulation results of an energy management system in a smart building, including the implementation of the proposed controller using a low-cost microcontroller.

I. Introduction

Due to the advances in optimization algorithms, hardware platforms and tailored implementations, the interest in embedded optimization has grown significantly during the last decade. A widespread application of embedded optimization is model predictive control, where optimization problems are sequentially solved online to achieve a control goal subject to constraints.

Different variations of the Nesterov's fast gradient method (see e.g. [1], [2]) and of the alternating directions method of multipliers (ADMM) [3] have been very successful for embedded optimization and model predictive control. Some of these algorithms are readily available via software tools that automatically generate c-code that can be used on embedded platforms in a simple form [4], [5], [6], [7].

While relevant case studies can be solved with these tools, many real-world applications require the solution of more complex optimization problems. In this paper, we focus on the solution of mixed-integer quadratic programs (MIQPs). Recent works have proposed embedded solutions for MIQP problems [8], [9]. However, it is still challenging to achieve fast reliable local solutions of moderate-size MIQPs on low-cost embedded hardware, and very difficult to obtain global solutions in real-time.

The main motivation of our work is to achieve an approximate global solution for moderate-size MIQPs with a very simple and fast embedded implementation with extremely low memory footprint that enables its deployment on low-cost microcontrollers.

B. Karg and S. Lucia are with the Laboratory of Internet of Things for Smart Buildings, Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany and Einstein Center Digital Future. (e-mail: sergio.lucia@tu-berlin.de)

We propose to use deep neural networks (DNN) [10] for two main reasons. First, to avoid solving online the MIQP that results from the formulation of a model predictive controller with integer decisions, which is a non-convex optimization problem. Second, to learn the MPC controller with integer decisions based on large amounts of data that can be generated offline using available solvers, which obtain global solutions. Thus, no real-time solution of the optimization problem is required, as also done in explicit MPC [11] which renders possible the control of systems with very small time constants. In contrast to explicit MPC, we obtain an analytic representation of the controller once the network has been trained. This network can be used to provide the optimal inputs using a low-power microcontroller with small memory footprint and with a very simple implementation or using recently developed integrated circuits tailored for efficient and low-power neural network evaluations.

The use of deep neural networks (networks with several hidden layers) has several advantages over traditional shallow networks with only one hidden layer: deep networks can better represent some functions with the same amount of neurons [12] and deep networks need to store less weights to be represented for a fixed number of neurons.

We consider the case-study of the energy management system of a smart building that includes temperature management of a building using a heating, ventilation and air conditioning (HVAC) system, a battery, and energy generation through solar panels that are connected to the grid.

II. Deep learning-based MPC

The central idea of this paper is to obtain an approximation of the parametric mixed-integer quadratic pogram that is obtained when formulating an MPC problem with integer decisions, as it is also done in explicit MPC [11], [13].

We propose to perform this aproximation using machine learning. Few works have explored this possibility, see e.g. [14] which uses support vector machines to learn a piecewise affine control law. To our knowledge, no previous works have considered before the use of deep learning to approximate a mixed-integer MPC problem and implement it on embedded platforms.

Deep learning has gained popularity in recent years, mainly for the sucessful solution of difficult problems that remained unsolvable for artifial intelligence applications (see [10]). Recently, also some theoretical background supports the experimental observations which conclude that deep neural networks, with many hidden layers, greatly outperform shallow networks with only one hidden layer (see e.g. [12]).

TABLE I: Number of weights (including the bias terms) for shallow and deep networks with the same number of hidden neurons and the same number of inputs and outputs.

n_{inp}	Neurons per hidden layer	n_{out}	n_{weights}
80	70	5	6025
80	10-10-10-10-10-10-10	5	1525

In an artificial neural network, each element or neuron i performs the following operation:

$$\tilde{z}^{(i)} = \sigma^{(i)}(W^{(i)} z^{(i)} + b^{(i)}),$$

where $\tilde{z}^{(i)}$ is the output of the neuron, $z^{(i)}$ is the input $\sigma^{(i)}$ is the activation function, $W^{(i)}$ is the kernel and $b^{(i)}$ is the bias.

If dense networks are considered, where all layers are fully connected, the amount of weights that have to be stored to completely describe a neural network can be computed as a function of the number of inputs n_{inp} to the network, the number of outputs n_{out} and the amount of neurons in each hidden layer.

The number of elements in the kernels $W^{(i)}$ scales quadratically with the size of the hidden layers. This is a significant advantage of deep neural networks for embedded applications, since much less memory is necessary to completely represent a network with the same number of neurons when compared to shallow networks. Table I illustrates this fact with a comparative example.

One of the most common choices for the activation function σ is the hyperbolic tangent function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

although recent works suggest that the use of a rectifier unit (*ReLU*) helps in deep learning to avoid the problem of vanishing gradients that often appears when using the hyperbolic tangent function leading to a slow training. The rectifier activation function

$$\textit{ReLU}(x) = \max\{0, x\}$$

is very simple and we choose it in this work because its simplicity leads to an accelerated embedded implementation and lower memory consumption during runtime, as it will be shown in Section IV.

We consider linear dynamic systems of the form

$$x_{k+1} = Ax_k + Bu_k + Ed_k, \tag{1}$$

where x_k, u_k and d_k denote the states, the control input and the disturbances at time step k where some of the control inputs u_k are possibly integer variables.

A general MPC problem with integer decisions can be written as:

$$\begin{aligned} \underset{(x,u)}{\text{minimize}} \quad & \sum_{k=0}^{N-1} J(x_k, u_k, d_k) && \text{(2a)} \\ \text{subject to} \quad & \text{model in (1)} && \text{(2b)} \\ & lb \leq M_x x_k + M_u u_k + M_d d_k \leq ub, && \text{(2c)} \end{aligned}$$

where $J(\cdot)$ is a quadratic cost function, N is the prediction horizon and (2c) describes mixed constraints for states, inputs and disturbances.

We consider that predictions of the disturbances d_k are available at each sampling time and therefore the MPC problem can be formulated as a multi-parametric MIQP in which the current state x_0 and the predictions of the disturbances $d^T = \left[d_0^T, d_1^T, \ldots, d_{N-1}^T\right]$ are the parameters.

To train a deep neural network, we consider as input data the parameters of the MIQP and as output data the first element of the optimal solution (u_0^*).

Instead of sampling the full state space for the parametric program, we sample the space of global optimal solutions, including different initial conditions and disturbance scenarios. That is, we choose random initial conditions and random - but reasonable - disturbance trajectories and solve an MPC problem for several steps.

The solution obtained at each step is stored as training data and the procedure is repeated for different initial conditions and predicted disturbances. In this way, it is possible to obtain enough training data to achieve good performance even for systems with many states and disturbances. Once the deep neural network is successfully trained, its application for an embedded controller requires only the forward evaluation of the network which includes only matrix-vector multiplications, sums and the application of the *ReLU* function leading to a very efficient embedded implementation.

III. Energy management system

We present an energy management system for a single-room building with an HVAC and a battery system.

The building model is based on the one presented in [15]. We extend this building with an energy system that consists of a solar panel for energy generation, a battery to store energy locally and a connection to the grid (see Fig. 1). It is possible to buy or sell electricity via the grid. However, we assume that legal requirements prevent to sell electricity that is stored in the battery. Thus it is only allowed to sell electricity currently generated through the solar panels.

The states that describe the system depicted in Fig. 1 are $x^T = [T_{\text{r}}, T_{\text{w}}, T_{\text{e}}, E_{\text{bat}}]$, where T_{r} is the room temperature, T_{w} is the temperature of internal walls, T_{e} the temperature of extenal walls and E_{bat} is the energy stored in the battery. The available control inputs are $u^T = [P_{\text{hvac}}, P_{\text{bat}}, P_{\text{grid}}, \alpha]$, where P_{hvac} is the power needed for the HVAC, P_{bat} the power delivered by or charged to the battery, P_{grid} the power sold or bought from the grid and α is a binary decision variable describing if the system is selling or buying energy. If $P_{\text{grid}} < 0$ energy is sold to the grid which is indicated by $\alpha = 1$. If $P_{\text{grid}} \geq 0$ energy is bought which is indicated

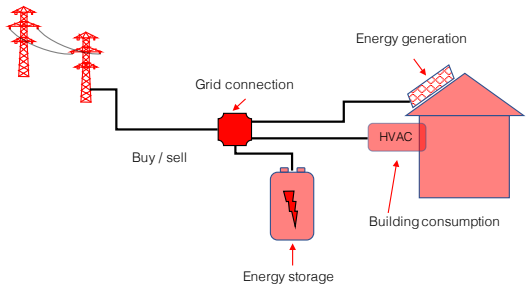

Fig. 1: A simplified energy management system.

by $\alpha = 0$. For $P_{\text{hvac}} < 0$ the building is cooled whereas for $P_{\text{hvac}} \geq 0$ the building is heated. If $P_{\text{bat}} \geq 0$ energy stored in the battery is used to power the HVAC, if $P_{\text{bat}} < 0$ the battery is being charged.

The external disturbances acting on the building are $d = [d_{\text{T}}, d_{\text{sr}}, d_{\text{int}}]$, where d_{T} is the external temperature, d_{sr} is the solar radiation and d_{int} are the internal gains.

The system can be described by (1), where

$$A = \begin{bmatrix} 0..8511 & 0.0541 & 0.0707 & 0 \\ 0.1293 & 0.8635 & 0.0055 & 0 \\ 0.0989 & 0.0003 & 0.0002 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.0035 & 0 & 0 & 0 \\ 0.0003 & 0 & 0 & 0 \\ 0.0002 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 \end{bmatrix},$$

$$E = 10^{-3} \begin{bmatrix} 22.217 & 1.7912 & 42.212 \\ 1.5376 & 0.6944 & 2.9214 \\ 103.18 & 0.1032 & 196.04 \\ 0 & 0 & 0 \end{bmatrix}.$$

The goal of the energy management system is to fulfill several constraints that are described in the following. The state constraints are given by

$$20.0 \leq T_{\text{r}} \leq 23.0^{\circ}\text{C}, \tag{3}$$

which gives a band of comfortable room temperatures and

$$0.0\% \leq SoC \leq 100.0\%, \tag{4}$$

where SoC is the state of charge of the battery.

The following mixed constraints ensure fulfilling the legal requirements and the energy balances. Regardless of weather conditions, the power required by the HVAC for cooling or heating should be provided by the energy stored in the battery, the power currently generated by the solar panels P_{PV} and the power bought from the grid:

$$|P_{\text{hvac}}| \leq P_{\text{bat}} + P_{\text{grid}} + P_{\text{PV}}, \tag{5}$$

where $P_{\text{PV}} = 0.5 \frac{1}{m^2} \cdot d_{\text{sr}}$, meaning that we assume that 50% of the solar radiation can be converted to electrical energy.

Furthermore, we assume that legal requirements enforce that only solar energy can be sold to the energy grid (and not stored in the battery). This can be modeled by the following constraint

$$P_{\text{grid}} \leq \alpha P_{\text{PV}}. \tag{6}$$

Finally, it is necessary to enforce the grid power to be negative if energy is sold. This can be modeled with the following constraint:

$$P_{\text{grid}} \leq (\alpha - 1) P_{\text{grid}}^{\text{ub}}. \tag{7}$$

The *economic* control goal is to maximize the energy sold to the grid. Additionally, we assume that it is desired to have a desired energy level in battery to ensure flexibility, which is modeled by a small tracking term. The resulting MIQP that should be solved at each sampling time of the MPC can be written as:

$$\begin{aligned}
\underset{(x,u)}{\text{minimize}} \quad & \sum_{k=0}^{N-1} (P_{\text{grid}}^k + \gamma (E_{\text{bat}}^k - E_{\text{bat}}^{\text{ref}})^2) && \text{(8a)} \\
\text{subject to} \quad & x_{\text{lb}} \leq x_k \leq x_{\text{ub}}, && \text{(8b)} \\
& u_{\text{lb}} \leq u_k \leq u_{\text{ub}}, && \text{(8c)} \\
& \alpha \in \{0, 1\}, && \text{(8d)} \\
& m_{\text{lb}} \leq D u_k + G d_k \leq m_{\text{ub}}, && \text{(8e)} \\
& x_{k+1} = A x_k + B u_k + E d_k, && \text{(8f)}
\end{aligned}$$

where N is the prediction horizon, $\gamma = 0.5$ is a penalty parameter, and x_{lb}, x_{ub}, u_{lb}, u_{ub}, m_{lb} and m_{ub} are the lower and upper bounds of the states, inputs and mixed constraints. The mixed constraints explained above can be defined by the matrices:

$$D = \begin{bmatrix} -1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & -P_{\text{PV}}^k \\ 0 & 0 & 1 & P_{\text{grid}}^{\text{ub}} \end{bmatrix}, G = \begin{bmatrix} 0 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

By inserting the equality constraints that describe system equations into the constraints and the cost function, the optimization problem can be reformulated as a function of the current initial state x_0 and and the disturbance predictions $d^T = \left[d_0^T, d_1^T, \ldots, d_{N-1}^T\right]$ in condensed form as

$$\begin{aligned}
\underset{u}{\text{minimize}} \quad & u^T H u + q^T(x_0, d) u, && \text{(9a)} \\
\text{subject to} \quad & z_{\text{lb}}(x_0, d) \leq M(d) u \leq z_{\text{ub}}(x_0, d), && \text{(9b)}
\end{aligned}$$

where $u^T = \left[u_0^T, u_1^T, \ldots, u_{N-1}^T\right]$ is the input trajectory, H is the Hessian of the problem, $M(d)$ is the constraint matrix and $z_{\text{lb}}(x_0, d)$ and $z_{\text{ub}}(x_0, d)$ are the lower and the upper bounds. The problem formulation has to be updated with the new initial state x_0 and disturbance predictions d^T at every iteration step. The derivation of the condensed formulation has been presented in several publications as in [16] and is omitted here for brevity.

IV. Results

A. Training of the deep learning-based MPC

To train the desired deep learning-based MPC, we use the data obtained from 500 different MPC runs, each one including 24 hours which means 48 steps (sampling time of MPC is 0.5 hours) resulting in 24000 data pairs.

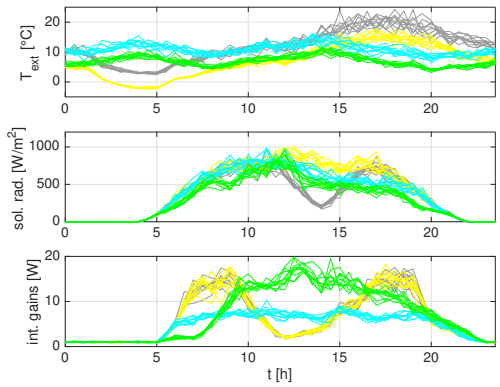

Fig. 2: Example of 80 disturbance trajectories used for training. The different colors represent noise variations of typical trajectories of weather data.

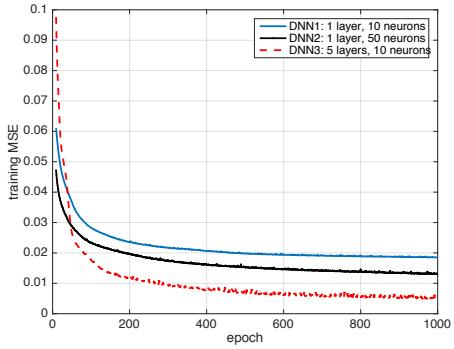

Fig. 3: Training error for different network architectures.

We use the initial condition of each optimization problem and the available weather predictions at each step as input data for the neural network. The first control input of the obtained optimal trajectory, that is, the input that is applied in the MPC implementation, is taken as output for the training of the neural network. For the offline solution of the MIQP (9) we use the solver SCIP [17] which implements an efficient branch-and-bound algorithm. All the optimization problems are solved to global optimality.

To generate the training data, uniformly distributed initial conditions and random noisy disturbance trajectories depicted in Fig. 2 for 80 different days are generated. The room temperature was varied between $T_\mathrm{r} = 20$ and 23°C, the wall temperatures between 15 and 25°C and the battery SoC between 10% and 90%. The trajectories of the disturbances (external temperature, solar radiation, internal gains) follow four different typical trajectories as it can be seen in Fig.2.

The prediction horizon is $N = 24$ steps which for a sampling time of 0.5 hours leads to a total horizon of 12 hours. The total number of inputs of the network is 76, 4 for the initial states and 72 for the trajectories along the prediction horizon of the three disturbances. The outputs of the network are the first step of the optimal solution, which corresponds to 4 outputs. To obtain the training data, it is assumed that the predictions of the disturbances are exact. The first 90 % of the obtained data has been used for training and the remaining 10 % is used as evaluation set. For the design of the DNN we use Tensorflow [18] and Keras [19] with the default parameters and the training is performed using the stochastic first-order gradient method Adam [20].

We use a rectifier function as activation function except for the output layer where a linear function is used.

Fig. 3 shows the training error for different structures of the network. The evaluation error is very similar, suggesting that no overfitting occurs and is omitted here for brevity. DNN1 is a shallow network with only 1 hidden layer and 10 neurons, DNN2 is a shallow network with 1 hidden layer and 50 neurons and DNN3 is a deep network with 5 hidden layers and 10 neurons per layer. Although DNN2 and DNN3 have the same total number of neurons, the deep network DNN3 achieves a significantly smaller training error. In addition, the number of weights needed to represent DNN3 (894) is smaller than to represent DNN2 (4054) leading to a smaller memory footprint.

For these reasons, DNN3 is chosen as an alternative to online mixed-integer MPC. The performance and robustness of this controller is evaluated in the next subsection.

B. Performance evaluation of deep learning-based MPC

We present here the results obtained when solving the MIQP presented in (9) in a receding horizon fashion. Fig. 4 and Fig. 5 show the results for 24 hours using an MPC controller that solves a MIQP online using SCIP and the proposed DNN controller which only evaluates the deep neural network previously trained. The same initial conditions and disturbance trajectories are used for both solutions. The resulting optimization problems solved by SCIP have 288 constraints, 72 continuous optimization variables and 24 integer optimization variables.

As it can be seen, the proposed deep learning-based MPC controller achieves a very similar performance compared to the MPC that solves MIQPs to global optimality at each sampling time. The trajectories for the binary variable α are identical meaning the DNN makes the optimal decision when to buy energy from the grid and when to sell energy to the grid. The main advantage of the DNN is the very simple implementation and fast online computation while SCIP cannot not be easily used for embedded platforms. Due to the approximation error of the DNN, small constraint violations occur for the deep learning-based controller.

Table II shows the average accumulated cost as well as the average constraint violations obtained when applying both controllers for 20 different runs with random initial conditions and disturbance realizations with perfect predictions. A very similar cost is obtained and only minor violations occur for the proposed deep learning-based MPC.

To evaluate the robustness of the proposed controller, we present 50 different MPC runs in Fig. 7 and Fig. 8 for the two algorithms with different initial conditions and different realizations of the disturbances in their predictions. In this

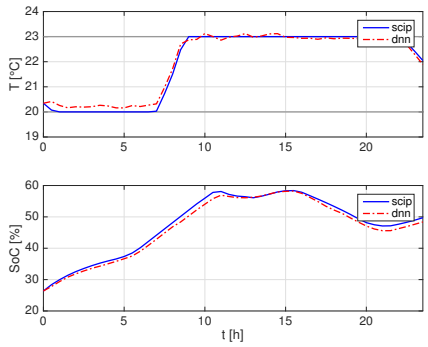

Fig. 4: State trajectory for disturbance predictions without noise. SCIP fulfills the constraints for all times while DNN shows small violations.

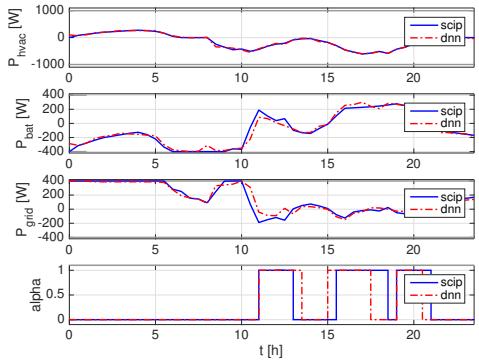

Fig. 5: Input trajectories for disturbance predictions without noise.

case, we use disturbance scenarios that are qualitatively and quantitatively different than those considered in the training data set, as it can be seen in Fig. 6. As input for the controllers (both for the online optimization with SCIP and for the DNN) the predictions are corrupted with 5% uniform noise to include errors in the predictions.

As it can be seen in Fig. 7 and Fig. 8, similar results are obtained for both controllers, demonstrating the robustness and graceful degradation of the approach to unseen scenarios as well as to wrong predictions and initial conditions. In a realistic scenario where predictions are not perfect, the differences between an exact MPC solution and the proposed deep learning-based controller are small, since both lead to small constraint violations due to plant model mismatch.

TABLE II: Average integrated cost and average violation of the temperature constraint (in °C/h) for exact MPC and deep learning-based MPC over 50 different simulations.

Controller	Average integrated cost	Average violations
SCIP	1.469e3	0
DNN	1.443e3	1.46e-2

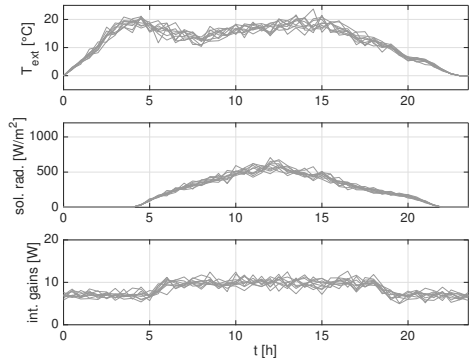

Fig. 6: Disturbances trajectories used for the robustness evaluation of the controllers, which are qualitatively and quantitatively different than those used for training. The disturbances shown here are the ones used to simulate the real system. The controllers receive as input a noisy version of these disturbances to simulate the error in weather and occupancy predictions.

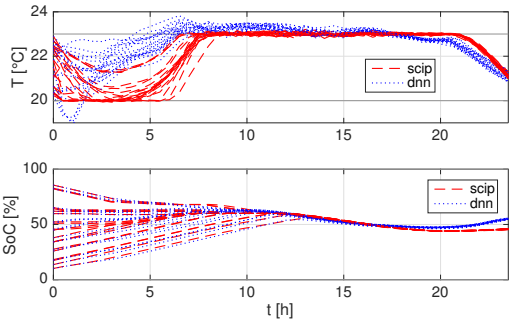

Fig. 7: State trajectory for disturbance predictions with noise. Both SCIP and DNN lead to small constraint violations. DNN has a graceful degradation when used with qualitatively and quantitatively different disturbance trajectories.

C. Embedded implementation

The main advantage of the proposed approach is that its online implementation is extremely simple and very fast, since the only necessary computations are those required to forward evaluate the trained neural network.

The workflow of the proposed approach requires the generation of the data by solving (offline) many MPC problems and then training a neural network. This network is used as the input for the tool *EdgeAI* that automatically generates library-free c-code that can be directly deployed on a microcontroller. We made the tool *EdgeAI* available on *https://github.com/edgeAI* and including an example of the c-code generation for a neural network.

The deep learning-based MPC controller presented in the previous subsection has been implemented on a low-cost 32-bit microcontroller (ARM Cortex M3) with 96 kB of RAM, 512 kB of Flash and a frequency of 89 MHz.

The necessary computation time for each sampling time

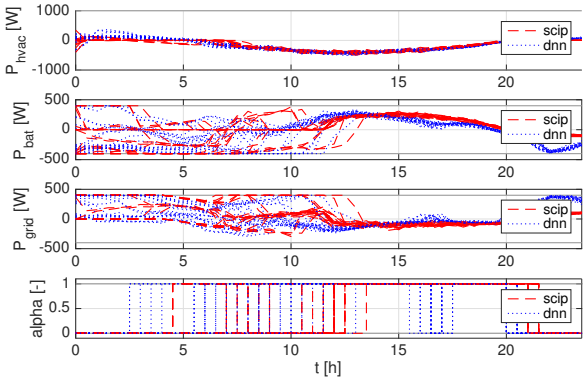

Fig. 8: Input trajectories for disturbance predictions with noise.

for a network with 5 hidden layers and 10 neurons per layer is 2.9 ms and the memory footprint of the compiled code on the microcontroller is only 35 kB. That is, using a low-cost microcontroller it is possible to obtain an almost globally optimal solution of an MIQP in a very small time with a very simple implementation.

If we use instead of *ReLU* classical activation functions such as $\tanh$, the computation time is significantly longer (7.3 ms) and the code is larger (37.3 kB) because it is necessary to use advanced math libraries.

These promising results show that deep neural networks with *ReLU* activation functions are powerful -but simple- alternatives to the online solution of non-convex optimization problems for model predictive control applications.

V. Conclusion

We have shown that deep learning-based MPC is a powerful alternative to online optimization for model predictive control, especially when the underlying problems are complex such as mixed-integer quadratic programs. The main advantages of deep learning are its reduced memory consumption compared to shallow networks and its better representation capabilities. Simulation results for an energy management system in a smart building show that a very similar performance can be achieved with the proposed controller compared to an exact solution of a non-convex optimization problem, but the proposed solution can be very easily and efficiently implemented in embedded hardware with a very low memory footprint. In other applications such as power electronics, where the execution time is crucial and an exact solution is therefore infeasible, the presented method enables online control due to its short computation time.

Future work includes the use of sparse networks to further reduce the memory footprint of the approach, as well as the explicit consideration of the uncertainty using techniques of robust model predictive control [21], [22] and a rigorous analysis of the approximation and representation capabilities of deep networks for explicit MPC.

References

[1] P. Giselsson, M. D. Doan, T. Keviczky, B. D. Schutter, and A. Rantzer, "Accelerated gradient methods and dual decomposition in distributed model predictive control," *Automatica*, vol. 49, no. 3, pp. 829 – 833, 2013.

[2] M. Kögel and R. Findeisen, "A fast gradient method for embedded linear predictive control," in *Proceedings of the 18th IFAC World Congress*, 2011, pp. 1362–1367.

[3] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[4] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[5] P. Zometa, M. Kögel, and R. Findeisen, "muAO-MPC: A free code generation tool for embedded real-time linear model predictive control," in *Proc. of the American Control Conference*, June 2013, pp. 5320–5325.

[6] S. Lucia, M. Kögel, P. Zometa, D. E. Quevedo, and R. Findeisen, "Predictive control, embedded cyberphysical systems and systems of systems – A perspective," *Annual Reviews in Control*, vol. 41, pp. 193–207, 2016.

[7] S. Lucia, D. Navarro, O. Lucia, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control using high level synthesis tools," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2018.

[8] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," in *American Control Conference*, July 2016, pp. 5619–5625.

[9] V. V. Naik and A. Bemporad, "Embedded Mixed-Integer Quadratic Optimization using Accelerated Dual Gradient Projection," in *Prof. of the 20th IFAC World Congress*, 2017, pp. 11 210–11 215.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, p. 484, jan 2016.

[11] 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.

[12] I. Safran and O. Shamir, "Depth-width tradeoffs in approximating natural functions with neural networks," in *ICML*, 2017.

[13] A. Alessio and A. Bemporad, "A Survey on Explicit Model Predictive Control," *Nonlinear Model Predictive Control*, vol. 384, pp. 345–369, 2009.

[14] L. Csko, M. Kvasnica, and B. Lantos, "Explicit MPC-Based RBF Neural Network Controller Design With Discrete-Time Actual Kalman Filter for Semiactive Suspension," vol. 23, no. 5, pp. 1736–1753, 2015.

[15] F. Oldewurtel, C. Jones, and M. Morari, "A tractable approximation of chance constrained stochastic MPC based on affine disturbance feedback," in *Proc. of the 47th IEEE Conference on Decision and Control*, Dec 2008, pp. 4731–4736.

[16] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[17] S. J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R. L. Gottwald, G. Hendel, T. Koch, M. E. Lübbecke, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, D. Weninger, J. T. Witt, and J. Witzig, "The scip optimization suite 4.0," ZIB, Tech. Rep., 2017.

[18] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[19] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[21] S. Lucia, T. Finkler, and S. Engell, "Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty," *Journal of Process Control*, vol. 23, pp. 1306–1319, 2013.

[22] S. Lucia, P. Zometa, M. Kögel, and R. Findeisen, "Efficient stochastic model predictive control based on polynomial chaos expansions for embedded applications," in *Proc. of the 54th IEEE Conference on Decision and Control*, 2015, pp. 3006–3012.