Approximate MPC based on machine learning and probabilistic verification

Sergio Lucia

Technische Universität Berlin
Einstein Center Digital Future
www.iot.tu-berlin.de
Motivation

Solving NMPC problems in real time is still challenging:
- For very fast systems
- On low-cost embedded hardware

Even more challenging in the case of robust NMPC

**Goal:** Development of an approach that simultaneously
- Obtains approximate optimal robust solutions
- Has small memory footprint
- Can be rapidly evaluated on an embedded device
Explicit MPC in the linear case

The MPC control law for LTI systems is a **piecewise-affine function**

- Depends only on the current state (and possibly parameters)
- Can be offline precomputed and stored

\[
K(x_{\text{init}}) = \begin{cases} 
K_1 x_{\text{init}} + g_1 & \text{if } x_{\text{init}} \in \mathcal{R}_1, \\
& \vdots \\
K_r x_{\text{init}} + g_r & \text{if } x_{\text{init}} \in \mathcal{R}_r,
\end{cases}
\]

Each region is described by a polyhedron:

\[
\mathcal{R}_i = \{ x \in \mathbb{R}^{n_x} \mid Z_i x \leq z_i \} \quad \forall i = 1, \ldots, n_r.
\]

[A. Bemporad, M Morari, V. Dua, E.N. Pistikopoulos, 2002]
Reducing complexity of explicit MPC

• Optimal representations:
  • Merging of regions with same feedback law
  • Lattice representation

• Suboptimal approximations:
  • Trade-off between complexity reduction and performance
  • Simplicial partitions
  • Neural networks

[T. A. Johannsen, A. Bemporad, F. Borrelli, C. Jones, M. Morari, M. Kvanisca and others]
Using neural networks to approximate MPC

It is an old idea: already done in 1995 for nonlinear MPC

• What is new?

Common practice until recent successes in deep learning was:

• Because of universal approximation theorem: use only 1 layer

What are the possible advantages of deep learning for approximating complex MPC laws?

Deep neural networks (DNN)

Neural network with \( L \) hidden layers and \( M \) neurons per layer

\[
N(x; \theta, M, L) = f_{L+1} \circ g_L \circ f_L \circ \cdots \circ g_1 \circ f_1(x)
\]

- Affine transformation \( \theta_l = \{W_l, b_l\} \)
  \[
f_l(x_{l-1}) = W_l x_{l-1} + b_l
\]
- Activation function
  - \( \text{tanh}: g_l(f_l) = \tanh(f_l) = \frac{e^{f_l} - e^{-f_l}}{e^{f_l} - e^{-f_l}} \)
  - \( \text{ReLU}: g_l(f_l) = \max(0, f_l) \)
Why deep (and not shallow)?

Number of linear regions represented by a ReLU network of depth $L$ and width $M$

$$n_r = \left( \prod_{l=1}^{L-1} \left[ \frac{M}{n_x} \right]^{n_x} \right) \sum_{j=0}^{n_x} \binom{L}{j}$$

- **Exponential** growth of regions w.r.t. depth
- Greater expressiveness with same amount of weights

[Montufar et al., 2014]
Proposed approach

1: Generate training samples by solving many MPC problems

2: Offline training of the deep neural network

\[ (x_0, u_0^*) \]

3: High performance implementation on low-cost embedded hardware
Increasingly popular

In many cases including strategies to have some guarantees:

- Chen et al., ACC 2018 (Projection at the output to achieve guarantees)
- Hertneck et al., IEEE Control System Letters 2018 (Hoeffdings inequality)
- Zhang, Bujarbaruah and Borrelli, ACC 2019 (Statistical validation)
- Drgona et al., Applied Energy 2018 (application on building control)
- Karg and Lucia, ECC 2018, NMPC 2018, ECC 2019 (applications, validation)
It works well in practice

[Lucia, Andersson, Brandt, Diehl and Engell. JPC 2014]
An industrial polymerization reactor

\[ \dot{m}_W = \dot{m}_{W,F} \]
\[ \dot{m}_A = \dot{m}_{A,F} - k_{R1}m_{A,R} - \frac{p_1k_{R2}m_{AWT}m_A}{m_{ges}} \]
\[ \dot{m}_P = k_{R1}m_{A,R} + \frac{p_1k_{R2}m_{AWT}m_A}{m_{ges}} \]
\[ \dot{T}_R = \frac{1}{c_{p,R}m_{ges}} \left[ \dot{m}_{F}c_{p,F}(T_F - T_R) + \Delta H_R k_{R1}m_{A,R} - k_A(T_R - T_S) - \dot{m}_{AWT}c_{p,R}(T_R - T_{EK}) \right] \]
\[ \dot{T}_S = \frac{1}{c_{p,S}m_S}[k_A(T_R - T_S) - k_A(T_S - T_M)] \]
\[ \dot{T}_M = \frac{1}{c_{p,W}m_{M,KW}} \left[ \dot{m}_{M,KW}c_{p,W}(T_{M}^N - T_M) + k_A(T_S - T_M) \right] \]
\[ \dot{T}_{EK} = \frac{1}{c_{p,R}m_{AWT}} \left[ \dot{m}_{AWT}c_{p,W}(T_R - T_{EK}) - \alpha(T_{EK} - T_{AWT}) + \frac{p_1k_{R2}m_{A}m_{AWT}\Delta H_R}{m_{ges}} \right] \]
\[ \dot{T}_{AWT} = \frac{1}{c_{p,W}m_{AWT,KW}} \left[ \dot{m}_{AWT,KW}c_{p,W}(T_{AWT}^N - T_{AWT}) - \alpha(T_{AWT} - T_{EK}) \right] \]

8 differential states
3 control inputs
2 uncertain parameters
Simulation results for multi-stage NMPC

Simple scenario tree
- Extreme values of the uncertainty
- Branch the tree only one stage
- Economic cost function

Simulations for different values of $k$ and $\Delta H$ (±30%)
Proposed approach

1: Generate training samples by solving many MPC problems

\[ \mathbf{x}_0, \mathbf{u}^* \]

2: Offline training of the deep neural network

\[ \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \]

3: High performance implementation on low-cost embedded hardware
Performance of deep-learning based ms-NMPC

**Exact vs. deep vs. shallow multi-stage NMPC**

Deep-learning based multi-stage NMPC
Performance of deep-learning based ms-NMPC

**Exact vs. deep vs. shallow** multi-stage NMPC

---

Average performance over random 100 batches

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Batch time [h]</th>
<th>Cons. Viol. [°C/h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>1.6459</td>
<td>0.0058</td>
</tr>
<tr>
<td>Shallow</td>
<td>1.7328</td>
<td>0.3087</td>
</tr>
<tr>
<td>Deep</td>
<td>1.6297</td>
<td>0.0549</td>
</tr>
</tbody>
</table>
Two main advantages

Enable low-cost emb. implementation
- 32 bit ARM Cortex M0+
- 48 MHz with 32 kB RAM

Approx. robust NMPC:
- Memory footprint: 27 kB
- Evaluation time on a uC: 37 ms
- Trivial code-generation (uC, FPGA)
Two main advantages

Enable low-cost emb. implementation
- 32 bit ARM Cortex M0+
- 48 MHz with 32 kB RAM

Approx. robust NMPC:
- Memory footprint: 27 kB
- Evaluation time on a uC: 37 ms
- Trivial code-generation (uC, FPGA)

Enable large(r)-scale systems
Problem with 5 uncertainties
- 243 scenarios
- ~115,000 variables and constraints
Mixed-integer case: Energy management system

- You can learn a **global optimal** solution

$$A = \begin{pmatrix}
0.8511 & 0.0541 & 0.0707 & 0 \\
0.1293 & 0.8635 & 0.0055 & 0 \\
0.0989 & 0.0003 & 0.0002 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}, \quad B = \begin{pmatrix}
0.0035 \\
0.0003 \\
0.0002 \\
-5
\end{pmatrix}.$$  

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

\[
\min_{(x,u)} \sum_{k=0}^{N-1} \left( P_{\text{grid}}^k + \gamma (E_{\text{bat}}^k - E_{\text{bat}}^{\text{ref}})^2 \right)
\]

subject to

\[
x_{k+1} = Ax_k + Bu_k + Ed_k,
\]

\[
x_{lb} \leq x_k \leq x_{ub},
\]

\[
u_{lb} \leq u_k \leq u_{ub},
\]

\[
\alpha \in \{0, 1\},
\]

\[
m_{lb} \leq Du_k + Gd_k \leq m_{ub}.
\]

\[
x = [T_r, T_{w,i}, T_{w,e}, E_{\text{bat}}]^T
\]

\[
u = [P_{\text{grid}}, P_{\text{hvac}}, P_{\text{bat}}, \alpha]^T
\]

\[
d = [d_{\text{amb}}, d_{\text{sol}}, d_{\text{int}}]^T
\]
Mixed-integer case

<table>
<thead>
<tr>
<th>Controller</th>
<th>Average integrated cost</th>
<th>Average violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIP</td>
<td>1.469e3</td>
<td>0</td>
</tr>
<tr>
<td>DNN</td>
<td>1.463e3</td>
<td>1.46e-2</td>
</tr>
</tbody>
</table>

[Karg and Lucia, ECC 2018]
Robust NMPC in 1 microsecond

**Induction heating** is currently used in many industrial and domestic applications

Control switching frequency and duty cycle. Satisfy constraints under uncertainty
Hardware-in-the-loop implementation

Advanced approximate optimization-based control in 1 μs (on an FPGA).
Easy to optimize FPGA implementation
Moving horizon estimation for sensor fusion

Fusing visual information and inertial sensors

- Common problem in autonomous driving, robotics
- Usually many assumptions to simplify online optimization (or EKF)

Fiedler et al., ECC 2020
Wait a minute... guarantees?

Compute the maximum approximation error

\[ d = \max_{x_0} |\pi_{\text{NN}}(x_0) - \pi_{\text{MPC}}(x_0)| \]

Design a controller that is robust against \( d \) and iterate
Wait a minute... guarantees?

Compute the maximum approximation error

\[ d = \max_{x_0} |\pi_{\text{NN}}(x_0) - \pi_{\text{MPC}}(x_0)| \]

Design a controller that is robust against \( d \) and iterate

- Computing the maximum is often not possible
  - Probabilistic Validation
Wait a minute... guarantees?

Compute the maximum approximation error

\[
d = \max_{x_0} |\pi_{\text{NN}}(x_0) - \pi_{\text{MPC}}(x_0)|
\]

Design a controller that is robust against \(d\) and iterate

• Computing the maximum is often not possible

• **Probabilistic Validation**
  – Hertneck et al., IEEE CSL 2018:
    – Based on Hoeffdings inequality and indicator (binary functions)
    – Based on Hoeffdings inequality and temporal logic with finite-time simulations
  – Zhang et al., ACC 2019:
    – Based on prob. validation results (Tempo, Bai, Dabbene, 1997) to achieve primal and dual guarantees
Probabilistic validation with performance indicators

A general (not necessarily binary) finite-time performance indicator

\[ \phi(w; N_{\text{sim}}, \kappa) = \phi(x(0), \hat{x}(0), \kappa(\hat{x}(0)), d(0), x(1), \kappa(\hat{x}(1)), d(1), \ldots, x(N_{\text{sim}})). \]

Given a controller \( \kappa \), a final simulation step \( N_{\text{sim}} \) and \( N \) i.i.d samples

\[ w^{(j)} = \{ x^{(j)}(0), \hat{x}^{(j)}(0), d^{(j)}(0), \ldots, d^{(j)}(N_{\text{sim}}) \}, \quad j = 1, \ldots, N, \]
Probabilistic validation with performance indicators

A general (not necessarily binary) finite-time performance indicator

$$\phi(w; N_{\text{sim}}, \kappa) = \phi(x(0), \hat{x}(0), \kappa(\hat{x}(0)), d(0), x(1), \kappa(\hat{x}(1)), d(1), \ldots, x(N_{\text{sim}})).$$

Given a controller $\kappa$, a final simulation step $N_{\text{sim}}$ and $N$ i.i.d samples

$$w^{(j)} = \{x^{(j)}(0), \hat{x}^{(j)}(0), d^{(j)}(0), \ldots, d^{(j)}(N_{\text{sim}})\}, \ j = 1, \ldots, N,$$

With probability no smaller than $\delta$

$$\text{Prob}\{\phi_i(w) > \psi_N^\phi(r)\} \leq \epsilon, \ i = 1, \ldots, M,$$

$\psi_N^\phi(r)$ is the maximum value of simulated $\phi_i(w)$ among all $N$, after removing the largest $r$ elements

Provided that:

$$N \geq \frac{1}{\epsilon} \left( r - 1 + \ln \frac{M}{\delta} + \sqrt{2(r - 1) \ln \frac{M}{\delta}} \right).$$
Differences with previous works

• Validation on general closed-loop performance guarantees
  • Not only binary functions
  • Not only error in the controller. Validation includes e.g. estimation errors

• Discard the $r$ largest values to facilitate successful validations

• Simultaneous design of several controllers (finite families)

More details in:
Some further results

Probabilistically safe, embedded robust output-feedback NMPC

\[
\dot{\theta}_{\text{kite}} = \frac{v_a}{L_T} (\cos \psi_{\text{kite}} - \frac{\tan \theta_{\text{kite}}}{E}),
\]

\[
\dot{\phi}_{\text{kite}} = -\frac{v_a}{L_T \sin \theta_{\text{kite}}} \sin \psi_{\text{kite}},
\]

\[
\dot{\psi}_{\text{kite}} = \frac{v_a}{L_T} \ddot{u} + \dot{\phi}_{\text{kite}} \cos \theta_{\text{kite}},
\]

Objective is to maximize thrust
Two states can be measured, EKF to estimate
Uncertain aerodynamic coefficients and wind parameters

Erhard and Strauch, 2012
Results

Embedded real-time implementation on an ARM-Cortex M3
• 96 kB memory footprint, 32 ms running time for DNN and 28 ms for EKF
Summary

1. Efficient approximation of the MPC control law using deep learning
   - Enables simple embedded implementation with very low memory footprint
   - Enables real-time robust NMPC of large complex systems

2. Statistical verification can be used to achieve guarantees

3. Recently good results for many different applications
Some material for discussion

• What is better:
  • First approximate then solve (usual path)
  • First solve (as complex as you can) then approximate

• What is more rigorous:
  • A priori guarantees (assumes knowledge of reality, including unc. description)
  • Probabilistic validation (assumes a reality simulator exists)
  • Many approaches use safety sets / backup controllers:
    • Nonlinear optimization running online -> one probably needs safety checks anyway…

• Hierarchy: learn a new controller when *something* changes

• Are finite-time guarantees acceptable? (even t is large?)
Open Invited Track at IFAC WC 2020 in Berlin

• Together with Ali Mesbah (UC Berkeley)
• Open Invited Track on „Machine Learning and MPC“
• Use submission code a1d55

• Deadline just extended to November 18th
Graceful performance degradation

Larger set of random initial conditions and uncertain parameters (±40%)

Exact multi-stage NMPC

Deep-learning based multi-stage NMPC

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Batch time [h]</th>
<th>Cons. Viol. [°C/h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>1.7882</td>
<td>0.1007</td>
</tr>
<tr>
<td>Deep</td>
<td>1.7800</td>
<td>0.1502</td>
</tr>
</tbody>
</table>
Training

Samples generated solving multi-stage NMPC (CasADi + IPOPT)
100 batches of data (with random initial cond. and uncertain param.)
  - 50 s sampling time
  - Total of 21050 samples
• Training with Keras / Tensorflow

\[
\min_{W_i, b_l} \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \left( \hat{u}(x_{tr,i}) - u^*(x_{tr,i}) \right)^2
\]

Output neural network Output multi-stage NMPC

<table>
<thead>
<tr>
<th>$n_l$</th>
<th>$n_n$</th>
<th>$n_{tot}$</th>
<th>$n_w$</th>
<th>$MSE_{train}$</th>
<th>$MSE_{test}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90</td>
<td>90</td>
<td>1263</td>
<td>0.0043</td>
<td>0.0046</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>90</td>
<td>2703</td>
<td>0.0024</td>
<td>0.0024</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>90</td>
<td>1413</td>
<td>0.0015</td>
<td>0.0014</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>90</td>
<td>1023</td>
<td>0.0014</td>
<td>0.0014</td>
</tr>
</tbody>
</table>
Summary

• Scheme to approximate complex model predictive controllers
• Efficient approximation of the MPC control law using deep learning
• Two main advantages
  • Enable embedded implementation with very low memory footprint
  • Enable real-time robust NMPC of large complex systems

• Some kind of safety net is necessary to have guarantees
  • (don't we always need this in reality, at least for the complex nonlinear case?)
• Other problems: adaptation and RL, optimal training, optimal structure