Industrial, large-scale model predictive control with deep neural networks

James B. Rawlings and Pratyush Kumar

Department of Chemical Engineering

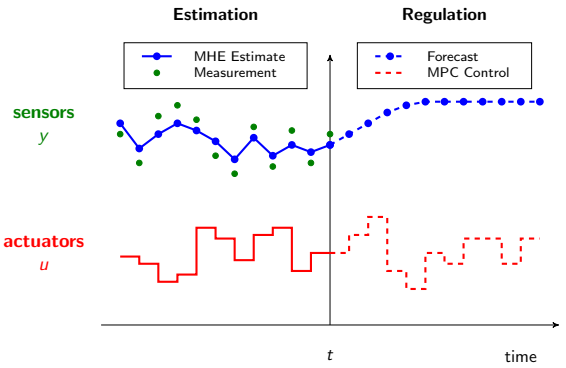
Intersections of control, learning, and optimization, 2020

UCLA

February 27, 2020

Outline

Introduction to model predictive control (MPC)



$$\min_{u(t)} \int_0^T \|y_{sp} - g(x,u)\|_Q^2 + \|u_{sp} - u\|_R^2 dt$$

$$\begin{aligned} \dot{x} &= f(x,u) \\ x(0) &= x_0 \quad \text{(given)} \\ y &= g(x,u) \end{aligned}$$

How can machine learning contribute to process operations and control

Three different approaches—

Reinforcement Learning (RL)

- Bypass both the plant model and the online optimization. Find the control law directly from the data using RL techniques.
- **Motivation:** Eliminates the need of plant modeling and the online optimization.

Approximate the MPC control law

- Replace the online optimization in MPC by a neural network (NN) that approximates the MPC control law.
- **Motivation:** Simplifies the controller implementation, and enlarges the class of applications accessible by MPC.

Build the plant model from data

- Construct accurate nonlinear predictive models from the data for subsequent use in online optimization based MPC framework.
- **Motivation:** Improves the closed-loop performance of MPC controllers.

Reinforcement Learning

Background

- Originally developed in the 1990s for control in finite Markov decision processes (MDPs) [1] – systems with finite state and action spaces. The control problem is: choose actions which maximize a future sum of rewards.
- **Successes in computer games:** RL algorithms combined with deep learning have been demonstrated to achieve breakthroughs in decision-making environments such as computer games, e.g., Go[2] and Atari Games [3].

Model-free RL algorithms

- **Q-learning:**[4] Parameterize a Q-function and estimate the parameters from data. Iterate over control laws using the estimated Q-function.
- **Policy optimization:**[5] Directly parameterize the control law. Find the parameters in the control law from the data.

[1] Watkins and Dayan (1992); Williams (1992)
[2] Silver et al. (2016)
[3] Mnih et al. (2015)
[4] Bradtke et al. (1994)
[5] Fazel et al. (2018)

Linear quadratic regulator (LQR)

Problem setup

Linear System: $x^+ = Ax + Bu$

Control Objective: $V(x(0), \mathbf{u}) = \frac{1}{2}\sum_{k=0}^{\infty}\left(x(k)^T Q x(k) + u(k)^T R u(k)\right)$

Optimal control law: $u = Kx, \quad K = -(B'\Pi B + R)^{-1}B'\Pi A$

System Identification

- Estimate A and B from the data. Compute the gain using the standard analytical solution.

Q-learning for LQR

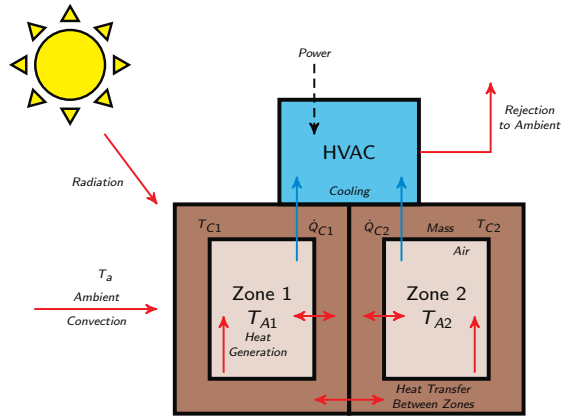
- Define the Q-function as:

$$Q_K(x,u) = x'Qx + u'Ru + (x^+)'\Pi x^+$$

$$Q_K(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}' \begin{bmatrix} Q + A'\Pi A & A'\Pi B \\ B'\Pi A & R + B'\Pi B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} = \underbrace{\begin{bmatrix} x \\ u \end{bmatrix}'}_{z'} \underbrace{\begin{bmatrix} S_{xx} & S_{xu} \\ S_{ux} & S_{uu} \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} x \\ u \end{bmatrix}}_{z}$$

- Estimate S from samples and update the gain as: $K \leftarrow -S_{uu}^{-1}S_{ux}$

Heating, ventilation, and air-conditioning (HVAC) example

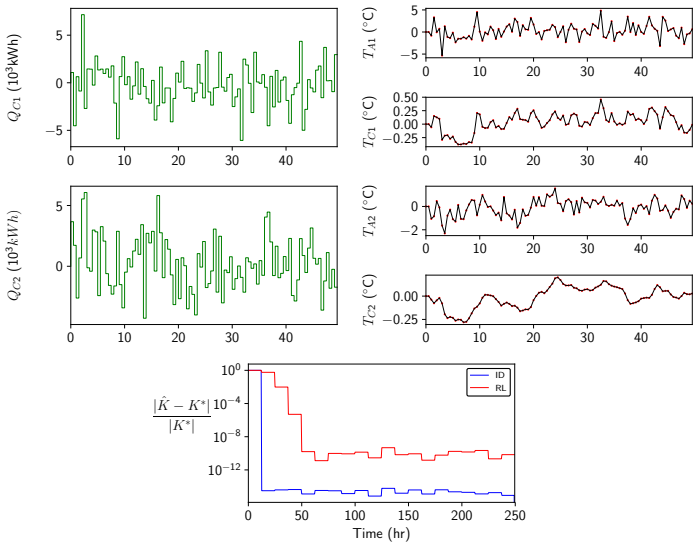


$$C_i\frac{dT_i}{dt} = -H_i(T_i - T_a) - \sum_{j\neq i}\beta_{ij}(T_i - T_j) - \dot{Q}_{ci} + \dot{Q}_{other,i}$$

$$T_i \in \{T_{A1}, T_{C1}, T_{A2}, T_{C2}\}$$

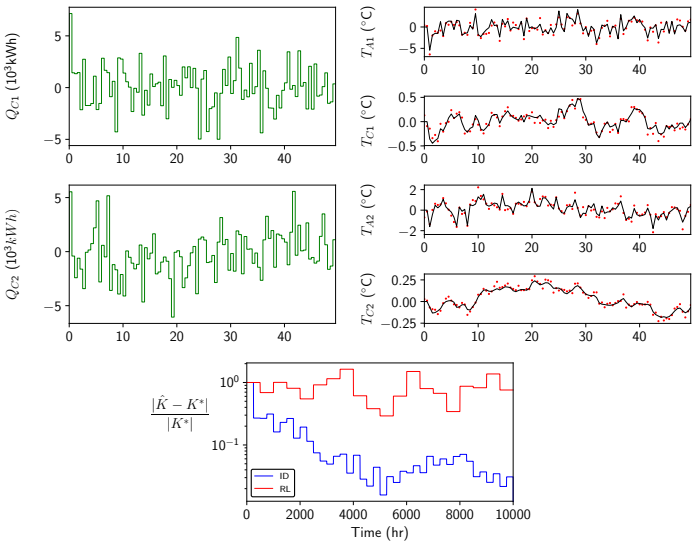
Linear System: $x^+ = Ax + Bu$. 4 states (T_i) and 2 control inputs (Q_{ci}).
Objective: Compare the ID and RL approaches to find the optimal gain K.

RL vs ID — No measurement noise



Both the methods can compute the optimal gain from noise-free measurements.

RL vs ID — With measurement noise



The Q-learning algorithm does not find the optimal gain from noisy measurements.

Why is RL so sensitive to noise despite using many samples?

$$Q_\pi(x,u) = \ell(x,u) + x^+\Pi x^+ = \ell(x,u) + Q_\pi(x^+, Kx^+)$$
$$Q_\pi(x,u) - Q_\pi(x^+, Kx^+) = \ell(x,u)$$

Substituting the Q-function gives

$$\begin{bmatrix} x \\ u \end{bmatrix}' S \begin{bmatrix} x \\ u \end{bmatrix} - \begin{bmatrix} x^+ \\ Kx^+ \end{bmatrix}' S \begin{bmatrix} x^+ \\ Kx^+ \end{bmatrix} = x'Qx + u'Ru$$

Taking the vec of both sides gives

$$\left\{ \begin{bmatrix} x \\ u \end{bmatrix}' \otimes \begin{bmatrix} x \\ u \end{bmatrix}' - \begin{bmatrix} x^+ \\ Kx^+ \end{bmatrix}' \otimes \begin{bmatrix} x^+ \\ Kx^+ \end{bmatrix}' \right\} \operatorname{vec} S = x'Qx + u'Ru$$

Use this {row vector} (data) times vector (unknown) equals scalar (data) at every sample to build a least-squares problem with s samples

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_s \end{bmatrix} \begin{bmatrix} \theta \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{bmatrix} \qquad \theta = \operatorname{vec} S$$

$$A\theta = b$$
$$\widehat{\theta} = (A'A)^{-1}A'b$$

So what's wrong with the least-squares estimation scheme?

Let us count the ways . . .

1. Row vector a_k and scalar b_k are complicated nonlinear (quadratic) functions of measurements (x_k, u_k, x_{k+1}), $k = 1, \ldots, s$.
2. The samples a_k and b_k are not independent. Note $x_{k+1} = Ax_k + Bu_k$.
3. There are errors in *both* A and b in the $A\theta = b$ least-squares problem.
4. For the least-squares estimate to be statistically correct (minimum variance, unbiased), we require *no error* in A, and *independent errors* in b.
5. Since A's errors are as large as b's errors, an error-in-variables rather than a least-squares approach is suggested.

Error-in-variables estimation

Nonlinear model $y = f(x;\theta)$. Error in measurement of both x and y.

Least squares (LS)

$$\min_\theta \sum_{k=1}^{s} \|y_k - f(x_k;\theta)\|^2$$

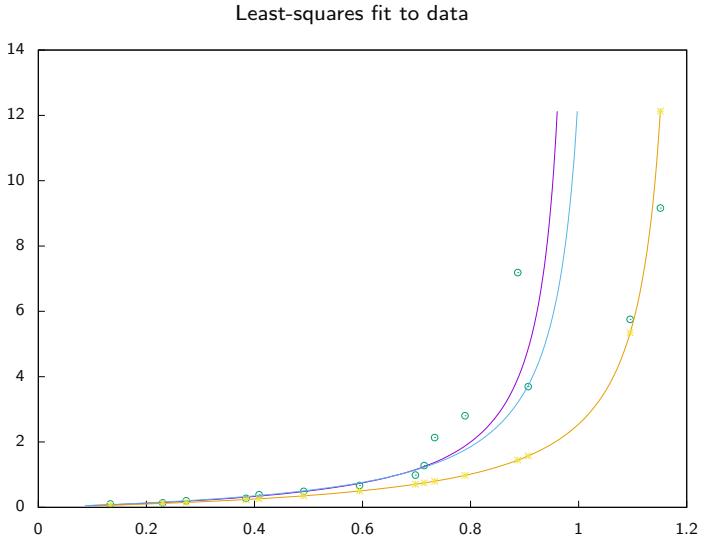
Error in variables (EIV)

$$\min_{\theta,\hat{x}_k,\hat{y}_k} \sum_{k=1}^{s} \frac{\|y_k - \hat{y}_k\|^2}{\sigma_y^2} + \frac{\|x_k - \hat{x}_k\|^2}{\sigma_x^2}$$

subject to: $\hat{y}_k = f(\hat{x}_k;\theta)$

- Taking limit $\sigma_x \to 0$ in EIV implies $\hat{x}_k \to x_k \implies \hat{y}_k \to f(x_k;\theta)$ recovering LS approach.
- The nonlinear program in EIV $(\theta, \hat{x}_k, \hat{y}_k)$ is larger than LS (θ).
- So when is EIV worth it?
- Hint: *usually not* worth it given a *linear* relationship between x and y, $y_k = mx_k + b$.

Nonlinear example: $y = \frac{1}{K}\left(\frac{c_m}{x} - 1\right)^{-1}$ (c_m, K surface kinetic parameters)



But EIV is intractable for Q-learning

We had this {row vector} (data) times vector (unknown) equals scalar (data) at every sample

$$\left\{\begin{bmatrix} x_k \\ u_k \end{bmatrix}' \otimes \begin{bmatrix} x_k \\ u_k \end{bmatrix} - \begin{bmatrix} x_{k+1} \\ Kx_{k+1} \end{bmatrix}' \otimes \begin{bmatrix} x_{k+1} \\ Kx_{k+1} \end{bmatrix}\right\} \operatorname{vec} S = x_k' Q x_k + u_k' R u_k$$

Call this a function

$$f(x_k, x_{k+1}, u_k; \theta) = 0 \qquad \theta = \operatorname{vec} S$$

EIV problem

$$\min_{\hat{x}_k, \hat{u}_k, \theta} \sum_{k=1}^{s} \|x_k - \hat{x}_k\|^2_{P_x^{-1}} + \|u_k - \hat{u}_k\|^2_{P_u^{-1}}$$

subject to: $f(\hat{x}_k, \hat{x}_{k+1}, \hat{u}_k; \theta) = 0$

A nonlinear program with $s(n+m) + (n+m)^2/2$ decision variables!

Approximating the MPC feedback control law

- The deployment of model predictive control using linear models requires solutions to convex quadratic programs (QPs) in real-time.
- The feedback law $u(x)$ is parameterized by the current state of the dynamic model $x^+ = Ax + Bu$.
- Calculating and storing this control law suffers from the curse of dimensionality; it has not been possible to solve for industrial process control applications. (Hence MPC.)
- We will discuss the design and offline training of neural network approximations of the feedback law, so that only feedforward evaluations of the neural network are required online in place of solving QPs.
- Motivation—Once trained offline, neural networks can provide control inputs faster online, enabling the deployment of linear MPC on large-scale problems not accessible with real-time QP solutions.

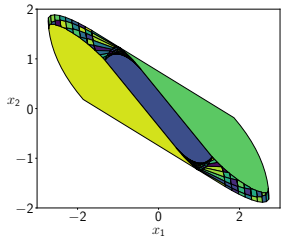
The MPC Control Law

Explicit Model Predictive Control

- For linear systems, linear constraints, quadratic stage cost, and control to the origin, the MPC control law is a piecewise affine (PWA) function of the system state over polytopic regions.[6]
- The number of the polytopic regions in the MPC control law grows rapidly with the increase in the system dimension, and the control horizon length.

Approximating the Control Law

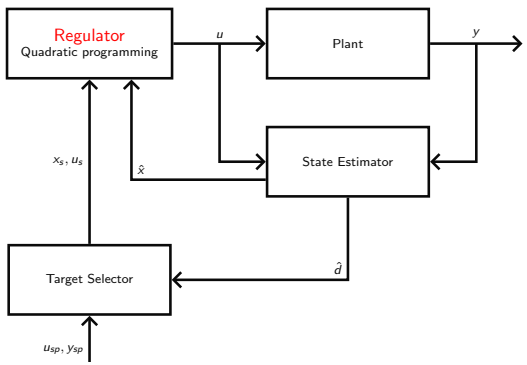
- Merge the regions.[7]
- Use deep neural networks.[8]



[6] Bemporad et al. (2002); Seron et al. (2000); Alessio and Bemporad (2009); Borrelli et al. (2017)
[7] Bemporad and Filippi (2003)
[8] Karg and Lucia (2018); Chen et al. (2018); Zhang et al. (2019); Chen et al. (2019); Karg and Lucia (2018)

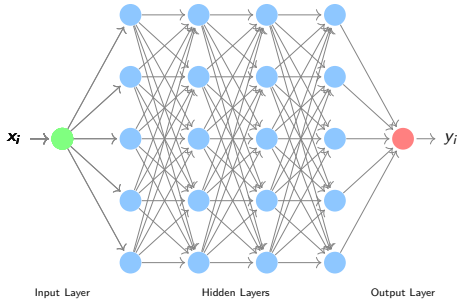
Offset-free model predictive control

- Augment the system model with a disturbance model for offset-free control.[9]
- Three components: state estimator, steady-state target selector, regulator.
- The regulator quadratic program is parameterized by $(\hat{x},\ x_s,\ u_s)$, and the optimal control input (u) is a piecewise affine function of these parameters.[10]
- Design a neural network that takes $(\hat{x},\ x_s,\ u_s)$ as its input and provides the current control (u) as its output.



[9] Pannocchia and Rawlings (2003)
[10] Bemporad et al. (2002)

Neural networks



- Neural networks represent a nonlinear function map from their input to the output.[11]

$$y_i = W_{h+1} \cdot \cdot g(W_2 g(W_1 x_i + b_1) + b_2) \cdot \cdot + b_{h+1}$$

- Compute the weights by minimizing a prediction error: $J = \frac{1}{m} \sum_{i=0}^{i=m} (\hat{y}_i(x_i) - y_i)^2$
- Use an optimization algorithm such as stochastic gradient descent (SGD) with software such as tensorflow.

[11] LeCun et al. (2015)

Structured neural network

Motivation

- The neural network should be structured such that at steady state, it maintains the plant at the appropriate steady-state target pair, i.e, $u_s = \kappa_{NN}(\hat{x} = x_s, x_s, u_s)$ for all (x_s, u_s).

Parameterization

$$u = u_s + \begin{bmatrix} W_{h+1} & -W_{h+1} \end{bmatrix} \cdot \cdot g\left(\begin{bmatrix} W_2 & 0 \\ 0 & W_2 \end{bmatrix} g\left(\begin{bmatrix} W_1 & 0 \\ 0 & W_1 \end{bmatrix} \begin{bmatrix} x \\ x_s \\ u_s \\ x_s \\ x_s \\ u_s \end{bmatrix} + \begin{bmatrix} b_1 \\ b_1 \end{bmatrix} \right) + \begin{bmatrix} b_2 \\ b_2 \end{bmatrix} \right)$$

- This structure of the neural network ensures that if the constraints are not active at steady state, there is no offset in the controlled plant outputs.

Controller design

Offline data generation

- Determine a range of typical anticipated set-point changes and disturbances to be encountered during plant operation.
- Simulate the plant by solving the model over a selection of setpoint and disturbance trajectories in this range.
- Gather the data set $(\hat{x},\ x_s,\ u_s,\ u)$ by solving quadratic programs (QPs) for all the transient states generated by the simulation.

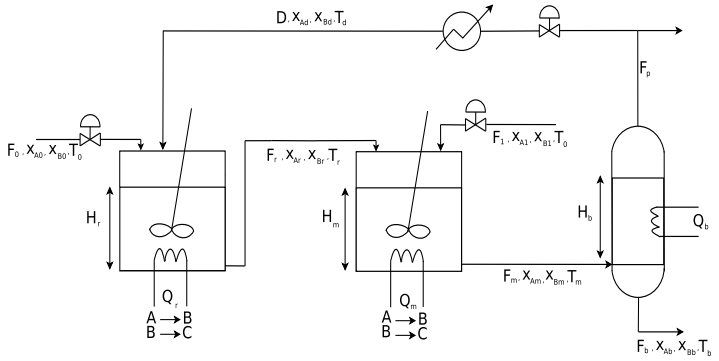
Neural network training

- Choose an architecture for the structured neural network.
- Train the structured network using the samples $(\hat{x},\ x_s,\ u_s)$ as the input and u as the output of the network.

Online implementation

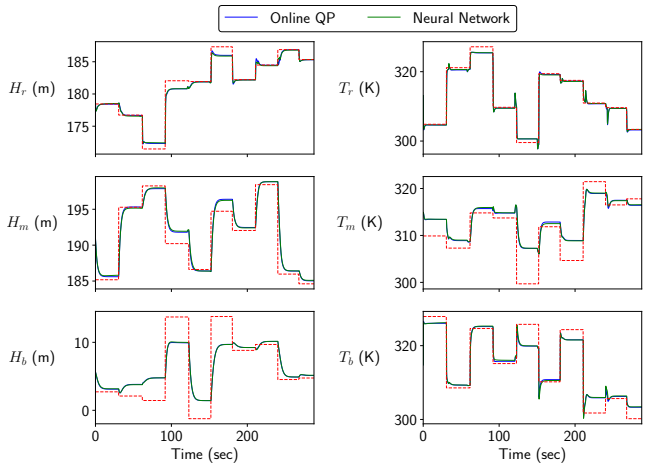
- Use a Kalman filter for state estimation (fast), a target selector for computing the steady state target pair (fast), and the trained neural network as the regulator.

Example #1 – CSTRs in series with a flash separator



- Nonlinear plant – 12 states, 6 manipulated inputs (the valve positions and the heat duties to the reactors), 6 controlled outputs (heights and temperatures).
- Plant disturbances are the feed stream temperature and the feed stream compositions.

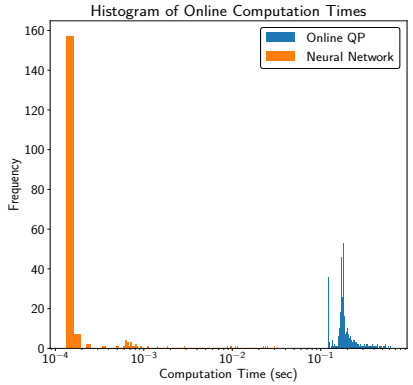
Online QP vs the NN controller – Closed-loop performance



- Structured neural network with 36 inputs, two hidden layers with 128 nodes in each layer, and 6 outputs. The network was trained using 24,000 samples (2.5 hours).
- Input constraints are active in places with nonzero offset.

Optimal MPC vs the NN Controller – Problem statistics

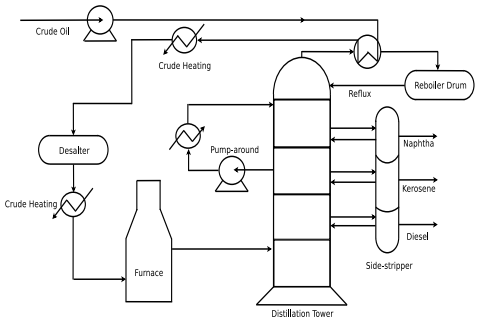
$$\Lambda = \frac{1}{N_{sim}} \sum_{t=1}^{N_{sim}} \left(|x(t) - x_s(t)|_Q^2 + |u(t) - u_s(t)|_R^2 + |\Delta u(t)|_S^2 \right)$$



- Average online computation times – 0.2s (Online QP[12]), 0.8ms (Neural Network). The speed-up factor achieved by the neural network is 250.

[12] The QP solver used is CVXOPT: Vandenberghe (2010)

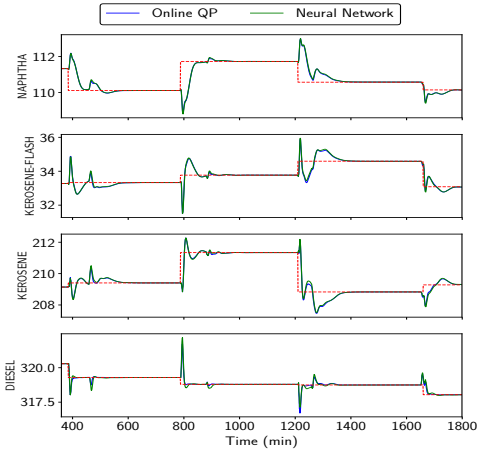
Example #2 – Large-scale crude distillation unit



- Size of the problem: 252 states, 32 manipulated inputs, 90 measured outputs, 4 controlled outputs [13].
- Manipulated control inputs – Valve positions throughout the plant.
- Controlled outputs – Product quality variables for the side-cuts kerosene, naphtha, and diesel.
- Plant disturbances – Crude feed composition, steam header pressure, and fuel gas quality.

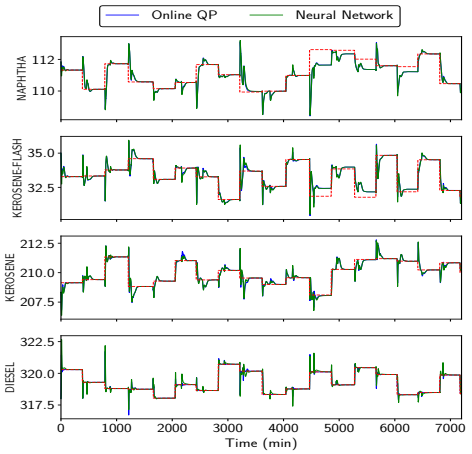
[13] Pannocchia et al. (2007)

Online QP vs the NN controller – Closed-loop performance



- Structured neural network with 568 inputs, two hidden layers with 2048 nodes in each layer, and 32 outputs. The network was trained using 150,000 samples (46 hours).

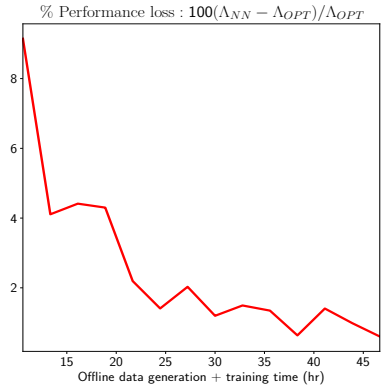
Online QP vs the NN controller – Closed-loop performance

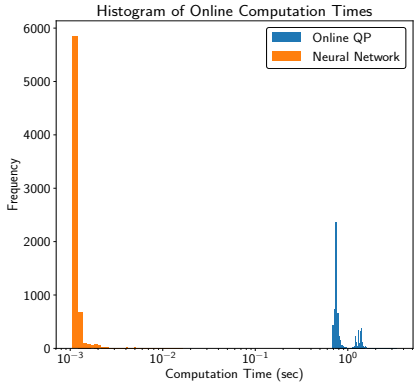


- Structured neural network with 568 inputs, two hidden layers with 2048 nodes in each layer, and 32 outputs. The network was trained using 150,000 samples (46 hours).
- Input constraints are active in places with nonzero offset.

Optimal MPC vs the NN Controller – Problem statistics

$$\Lambda = \frac{1}{N_{sim}} \sum_{t=1}^{N_{sim}} \left(|x(t) - x_s(t)|_Q^2 + |u(t) - u_s(t)|_R^2 + |\Delta u(t)|_S^2 \right)$$





- Average online computation times – 1s (Online QP [14]), 1.4ms (Neural Network). The speed-up factor achieved by the neural network is 720.

[14] The QP solver used is CVXOPT: Vandenberghe (2010)

When is this technology advantageous

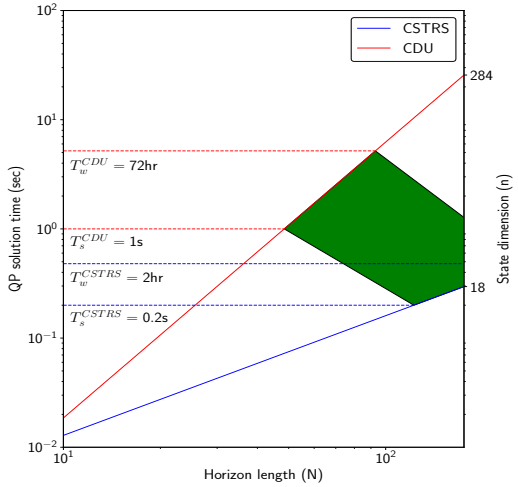
- T_s : Desired controller execution sampling rate

  $QP(n, N)$: QP solution time

  N_s : Number of training samples required

  T_w : Offline waiting time

- The size of accessible problems are: $T_s \leq QP(n, N) \leq \frac{T_w}{N_s}$
- We assume that both the offline training and the online deployment is being done on a 2.7GHz CPU with CVXOPT [15] as the QP solver.



[15] Vandenberghe (2010)

Deployment opportunities

Scenario 1

- Offline training and online deployment on the same machine.
- The approach is tractable and implementable for systems evolving at timescales of milliseconds/seconds.
- The class of control applications include fast low-level regulatory control.

Scenario 2

- Offline training—powerful CPU, online deployment – cheaper CPU.
- Access to powerful/parallel CPUs for offline training enables the deployment of neural network controllers on a larger class of problems.
- Multiple large-scale processes can be managed using the same machine with re-training required when model/MPC tuning parameters are updated.

Conclusions

Summary

- Q-learning of the optimal control struggled to converge with noisy data compared to the model-based approach.
- Deep neural networks can be used to learn the MPC control law even for large-scale industrial models.
- The **structure of the neural network** and sampling the state-space **only for the plant operational scenarios** are the key features to **avoid the curse of dimensionality**.
- Neural networks can provide a faster and reliable execution of MPC with equivalent closed-loop performance as the optimal MPC controller.

Future research direction

- Investigate what is achievable with neural networks to learn the control law for economic and nonlinear MPC problems.
- Develop systematic frameworks to use neural networks for building dynamic models (grey-box and data-driven).

Acknowledgments

The authors would like to acknowledge helpful discussions with Steve Wright and Ben Recht. Funding and sample data were provided by Johnson Controls, Inc. and the National Science Foundation.

Optimizations were performed using Gurobi via GNU Octave and Python.

References I

A. Alessio and A. Bemporad. A survey on explicit model predictive control. In L. Magni, D. Raimondo, and F. Allgöwer, editors, *Nonlinear Model Predictive Control - Towards New Challenging Applications*, pages 345–369. Springer Berlin / Heidelberg, 2009.

A. Bemporad and C. Filippi. Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming. *IEEE Trans. Auto. Cont.*, 117(1):9–38, 2003.

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

F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for linear and hybrid systems*. Cambridge University Press, University Printing House, Cambridge CB2 8BS, United Kingdom, 2017.

S. J. Bradtke, B. E. Ydstie, and A. G. Barto. Adaptive linear quadratic control using policy iteration. In *American Control Conference, 1994*, volume 3, pages 3475–3479. IEEE, 1994.

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

S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari. Large scale model predictive control with neural networks and primal active sets. *arXiv preprint arXiv:1910.10835*, 2019.

M. Fazel, R. Ge, S. M. Kakade, and M. Mesbahi. Global convergence of policy gradient methods for the linear quadratic regulator. *arXiv preprint arXiv:1801.05039*, 2018.

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

References II

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

G. Pannocchia and J. B. Rawlings. Disturbance models for offset-free MPC control. *AIChE J.*, 49 (2):426–437, 2003.

G. Pannocchia, J. B. Rawlings, and S. J. Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43:852–860, 2007.

M. M. Seron, J. A. De Doná, and G. C. Goodwin. Global analytical model predictive control with input constraints. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pages 154–159, Sydney, Australia, December 2000.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

L. Vandenberghe. The cvxopt linear and quadratic cone program solvers. *Online: http://cvxopt. org/documentation/coneprog. pdf*, 2010.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

X. Zhang, M. Bujarbaruah, and F. Borrelli. Near-optimal rapid mpc using neural networks: A primal-dual policy learning framework. *arXiv preprint arXiv:1912.04744*, 2019.