Approximation of Nonlinear Model Predictive Control Using Mixture Density Networks

Morimasa Okamoto [a], Jiayang Ren [a], Qiangqiang Mao [a] and Yankai Cao [a,1]

[a] Department of Chemical and Biological Engineering, The University of British Columbia, Vancouver, BC, Canada

Abstract

Model Predictive Control (MPC) is an advanced control method broadly applied to chemical processes. However, the prohibitive online computation time limits its application to nonlinear systems. Although the approximation of the MPC control law via deep neural networks (DNNs) has been studied in these recent years, this approach cannot be applied to nonlinear systems if the optimal control problems have multiple optima. When the MPC control law follows one-to-many mappings, it cannot be effectively approximated via DNNs, which provide one-to-one mappings. In this paper, we propose a mixture density network(MDN)-based approximation method for nonlinear MPC. MDNs approximate the MPC control law through conditional probabilities by mixing several estimated Gaussians and then generate several control inputs with the highest probabilities, which means that the network can realize the one-to-many mappings. We also investigate a case study of a nonlinear benchmark process, which demonstrates that our proposed scheme exhibits better control performance than the DNN-based approximation method.

Keywords

Model Predictive Control, Mixture Density Networks, Nonlinear systems.

Introduction

Model Predictive Control (MPC) is an advanced control method applied in various industrial processes, such as chemical plants and power systems (Qin and Badgwell, 2003; Xi et al., 2013). Compared with traditional control methods such as PID controllers, MPC typically provides better performance while satisfying a set of constraints. It is realized by optimizing a finite time horizon control problem and implementing the first control input. However, the prohibitive online computation time limits its real-world applications, especially for nonlinear systems. To achieve fast online control, explicit MPC is proposed by pre-computing the explicit optimal control law using parametric programming techniques (Tøndel et al., 2003). In this way, online computations are reduced to a simple function evaluation. However, offline computing scales poorly with the number of constraints and prediction horizon. Moreover, the application of this approach to nonlinear systems is not straightforward.

In recent years, machine-learning and deep-learning methods have proliferated in many fields, such as image recognition (Krizhevsky et al., 2012) and system identification (Wu et al., 2019). Inspired by these advances, realizing fast online control while maintaining scalability in offline computing by approximating the complex MPC control law via deep neural networks (DNNs) has attracted many focuses. Karg and Lucia (2020) showed that a neural network with rectifier units could exactly approximate the MPC control law of linear time-invariant systems. They further demonstrated that DNNs with several hidden layers could robustly approximate the nonlinear MPC control law (Lucia and Karg, 2018). Considering the time series nature of the process states, Kumar et al. (2018) proposed utilizing Long Short-Term Memory (LSTM) networks to approximate the MPC control law more accurately.

While these DNN-based approximation methods can significantly accelerate online control with acceptable offline computing costs, they cannot address set-valued optimal inputs for nonlinear systems (Cao and Gopaluni, 2020; Li et al., 2022). It is possible that a system with complex nonlinear dynamics has multiple global optima. Therefore, the calculated optimal MPC control laws are usually one-to-many mappings, one state corresponding to several optimal control inputs (which is also called set-valued optimal inputs). Additionally, since the lack of efficient global optimal solvers for the nonlinear MPC problems, when generating the training datasets, we often rely on the local optimal solvers such as Ipopt (Wächter and Biegler, 2006). In that case, the solver computes one of the multiple local optima; thus, the computed MPC control laws also follow one-to-many mappings. The aforementioned DNN-based approximation methods can only approximate one-to-one mappings and are not suitable for one-to-many mappings (as demonstrated in Example 1).

To address the potential set-valued optimal inputs, we propose a mixture density network(MDN)-based approximation

[1] Corresponding author: Yankai Cao (E-mail: yankai.cao@ubc.ca).

method (MDN-based MPC) for nonlinear MPC. MDNs with several Gaussian components in the output layer can predict the conditional probabilities of possible control inputs by mixing multiple estimated Gaussians, then generate several control inputs with the highest probabilities (Bishop, 1994). Therefore, the network can implement the one-to-many mappings and approximate the nonlinear MPC control law with set-valued optimal inputs.

This paper is organized as follows. Section 2 introduces the DNN-based MPC that has already been proposed and studied. Section 3 describes the proposed MDN-based MPC. We investigate a case study of a nonlinear benchmark process in section 4, which demonstrates that our proposed scheme exhibits better control performance than conventional DNN-based MPC. Finally, section 5 concludes this paper.

Deep Neural Network-based Model Predictive Control (DNN-based MPC)

MPC is a method that determines control inputs by solving optimization problems repeatedly at each sampling time (Rawlings et al., 2020). Given the initial state x_0 and the prediction horizon N, the optimal control problem $\mathbb{P}_N(x_0)$ can be formulated as following

$$\begin{aligned} \min_{u(k)} \quad & \sum_{k=0}^{N-1} l(x(k),u(k)) + V_f(x(N)) \\ \text{s. t.} \quad & x(k+1) = f(x(k),u(k)),\ \ x(0) = x_0 \\ & x(k) \in \mathbb{X},\ \ x(N) \in \mathbb{X}_f,\ \ u(k) \in \mathbb{U},\ \ \forall k \in \mathbb{T} \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ is the state vector of the dimension n, $u \in \mathbb{R}^m$ is the control input vector of the dimension m, $l(\cdot)$ is the stage cost function, $V_f(\cdot)$ is the terminal penalty function, k is the sampling time, $f(\cdot)$ is the function representing process dynamics, $\mathbb{X}$ is the state constraint set, $\mathbb{X}_f$ is the terminal state constraint set, $\mathbb{U}$ is the input constraint set, and $\mathbb{T} := \{0, 1, \cdots, N-1\}$ is the prediction step set.

We denote the optimal control inputs for $\mathbb{P}_N(x_0)$ as $u^\circ(x_0) = (u^\circ(0;x_0), u^\circ(1;x_0), \cdots, u^\circ(N-1;x_0))$. Based on the receding horizon scheme, MPC applies the first step input $u^\circ(0;x_0)$ of the optimal control input vector. Therefore, the MPC control law $\kappa_N(\cdot)$ is a mapping from the initial state to the control input, and is defined as $\kappa_N(x_0) := u^\circ(0;x_0)$.

DNN-based MPC approximates these mappings in MPC control law via DNNs. Here, DNNs are neural networks with multiple hidden neurons and layers, which are formulated as

$$h^{(l)} = g^{(l)}(W^{(l)} h^{(l-1)} + b^{(l)}) \tag{2}$$

where l is the layer number, $g^{(l)}$ is the activation function for the layer l, $W^{(l)}$ is the weight of the layer l, $h^{(l)}$ is the hidden variables of the layer l, and $b^{(l)}$ is the bias of layer l (Goodfellow et al., 2016). $h^{(0)}$ corresponds to the system states of the training samples. By learning the parameter set composed of $W^{(l)}$ and $b^{(l)}$, DNNs can represent complex relationships between the system states and the control inputs as nonlinear approximation functions. Therefore, DNNs can approximate the MPC control law $\kappa_N(\cdot)$ by the pairs of the initial states x_0 and the optimal control inputs $u^\circ(0;x_0)$. In this way, DNN-based MPC can dramatically reduce online computation time because we can obtain the control inputs predicted directly by pre-trained DNNs.

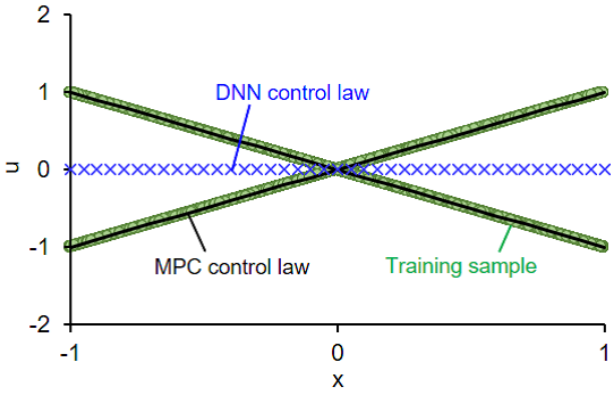


Figure 1: Comparison of MPC and DNN control laws for Example 1.

One drawback of the DNN-based MPC is that it can only approximate one-to-one mappings between the system states and the control inputs. When multiple optimal control inputs exist for some system states (called set-valued MPC control law), it will fail to give an accurate approximation. The following example shows a simple nonlinear optimal control problem introduced by Cao and Gopaluni (2020) and Li et al. (2022), in which a set-valued MPC control law is approximated.

Example 1: Approximating set-valued MPC control law of the optimal control problem.

$$\begin{aligned} \min_{u(k)} \quad & \sum_{k=0}^{1} x(k)^2 \\ \text{s. t.} \quad & x(1) = x(0)^2 - u(0)^2,\ \ x(0) = x_0 \end{aligned} \tag{3}$$

The prediction horizon of this problem is one step. We do not consider the state and input constraint. We generated 10000 training samples by solving the above optimal control problem for 10000 randomly generated initial system states x_0. We also generated 10000 test samples in the same way. Since this problem obviously has a set-valued MPC control law $\kappa_N(x_0) := \pm x_0$, two optimal inputs for one initial state x_0, the optimization solver selects any one of two optimal control inputs based on the randomly chosen initial guess.

Figure 1 illustrates the MPC control law and the approximated DNN control law with 10 hidden neurons, 1 hidden layer and *tanh* activation function. From the figure, it seems that the DNN control law is approximately the average of the multiple inputs in the training samples and is close to $\hat{u}(x_0) = 0$, where $\hat{u}$ is the control input predicted by the DNN. It is clear that the trained DNN control law deviates far from the original MPC control law. That is because we try to fit the one-to-many mapping via the DNN, which can only provide the one-to-one mapping.

Table 1 shows the training and test errors of the trained DNN controller with increased hidden layers. To consider the set-valued results, we calculate the mean squared errors

Table 1: Training and test errors of DNN controllers for Example 1.

Hidden layers (10 neurons)	Training error	Test error
1	0.3311	0.3311
2	0.3277	0.3277
3	0.3294	0.3294

(MSE) in the following way:

$$MSE = \frac{1}{S}\sum_{s\in\mathbb{S}} min\left((\hat{u}_s - u_s^1)^2, \cdots, (\hat{u}_s - u_s^p)^2\right) \tag{4}$$

where s is the index of training samples, $\mathbb{S} := \{1, 2, \cdots, S\}$ is the set, S is the total number of training samples, and $\hat{u}_s$ is the predicted control input from DNN for state x_s. Each state x_s has multiple optimal MPC control inputs u_s^p, where p is the number of optimal inputs ($p = 2$ for this example). Table 1 highlights that the discrepancy between the MPC controller and the DNN approximation cannot be alleviated by tuning the DNN structures.

Mixture Density Network-based Model Predictive Control (MDN-based MPC)

MDNs are neural networks with several Gaussian components in the output layer (Bishop, 1994). Figure 2a is a MDN with 2 Gaussian components for the one-dimensional target variable. As Figure 2b shows, the MDN can calculate the conditional probability of the target variable by superposing these Gaussians. The function for computing conditional probabilities is

$$p(u|x) = \sum_{i=1}^{K} \alpha_i(x) \cdot \phi_i(u|x) \tag{5}$$

where i is the Gaussian index, K is the total number of Gaussians, x is the system state, u is the control input, and $\alpha_i(x)$ is the contribution of Gaussian i for x, with $\sum_{i=1}^{K} \alpha_i(x) = 1$. $\phi_i(u|x)$ is the function computing the conditional probability density of the control input u, given x.

Assuming the control input is a multivariate vector, the function $\phi_i(u|x)$ is the probability density of multivariate Gaussian distribution:

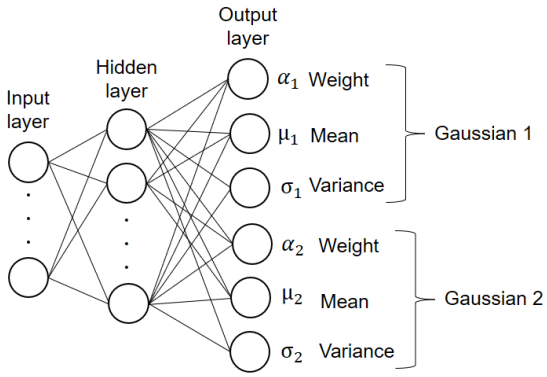
$$\begin{aligned}\phi_i(u|x) \quad = \quad & \frac{\sqrt{|\Sigma_i(x)^{-1}|}}{\sqrt{(2\pi)^m}} \\ & \cdot \exp\left(-\frac{1}{2}\cdot(u-\mu_i(x))^T \cdot \Sigma_i(x)^{-1} \cdot (u-\mu_i(x))\right)\end{aligned} \tag{6}$$

where $\Sigma_i(x)$ is the covariance matrix of Gaussian i for x.

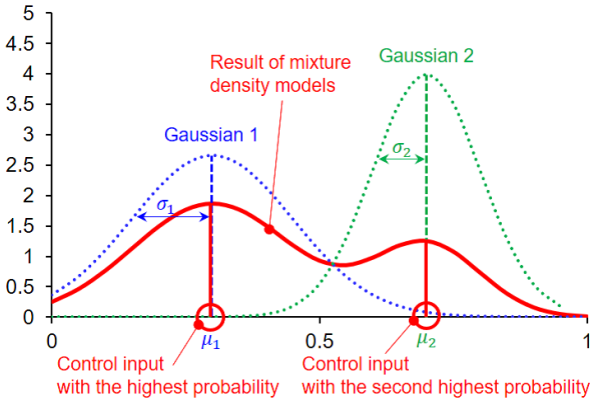
The loss function for training MDNs is the negative logarithm of the likelihood:

$$E = \sum_{s\in\mathbb{S}} \left[-\ln\left(\sum_{i=1}^{K} \alpha_i(x_s)\phi_i(u_s|x_s)\right)\right] \tag{7}$$

The parameter set minimizing this loss function is the optimal MDNs parameter.



(a) Mixture density network with 2 Gaussians



(b) Conditional probability density of mixture model

Figure 2: Example of mixture density networks.

The inverse of the covariance matrix $\Sigma_i(x)^{-1}$ needs to be positive-definite. We can ensure this by Cholesky decomposition: $\Sigma_i(x)^{-1} = U_i(x)^T \cdot U_i(x)$ where $U_i(x)$ is the upper triangular matrix. Following this idea, the output layer includes not the elements of $\Sigma_i(x)^{-1}$ but the elements of $U_i(x)$ (Kruse, 2020).

One feature of the MDN architecture is that the activation functions of the output layer are different for each output neuron. Firstly, the activation function for the output neurons corresponding to the contribution α_i is set to be the softmax function because the sum of contributions is 1. Then, the identity function is often chosen to be the activation function for the output neurons corresponding to the mean μ_i. Finally, for the covariance matrix, if the values of its elements are too small, then the probabilities of the target vectors may be almost zero. In this case, since the loss function Eq. (7) includes the logarithm of the conditional probability, MDNs cannot be trained because the loss becomes negative infinity. Therefore, we introduce a tailored *sigmoid* function with a variable range transformation as the activation function $g_{\sigma_{ii}}^{(L)}$ for the diagonal elements of the covariance matrix:

$$g_{\sigma_{ii}}(z) = \underline{z} + \frac{\bar{z} - \underline{z}}{1 + \exp(-z)} \tag{8}$$

where $\underline{z}$ is the lower limit and $\bar{z}$ is the upper limit. This function project z to the range between $\underline{z}$ and $\bar{z}$. Also, for the non-diagonal elements of the covariance matrix, the activation function is often *tanh*.

Table 2: Training and test errors of MDN controllers for Example 1.

Number of Gaussians	Training error	Test error
1	0.3074	0.3031
2	0.0013	0.0013

The MDN control law is the control input with the highest conditional probability: $\kappa_N(x_0) := \arg\max_{u\in\mathbb{Q}} p(u|x)$, where $\mathbb{Q}$ is the possible control input set. To obtain the control input using the MDN control law, we can directly apply optimization methods to find the control inputs by maximizing $p(u|x)$. Alternatively, we can generate a control input table for each state in advance and use the table to decide the control inputs in the online situation. However, these methods are time-consuming when possible system states and control input dimensions are numerous. To conquer this problem, we find that the control inputs with the highest probabilities are generally among the mean value set of all the Gaussians. Therefore, we can compute the probabilities of mean values of the Gaussians based on Eq. (5), then obtain the control inputs by selecting the mean value with the maximum probability.

We demonstrated that MDNs could approximate the MPC control law with set-valued inputs using the same training samples in Example 1. Specifically, the MDNs have one hidden layer with 30 hidden neurons. The activation function, except for the output layer, is *tanh*. Besides, we deployed MDNs with 1 and 2 Gaussians to check the advantage of multiple Gaussians. Table 2 illustrates the training and test errors of the MDNs. In these experiments, MDNs with 1 Gaussian have similar errors to DNNs in Table 1 because 1 Gaussian can only achieve one-to-one mapping as DNN does. On the contrary, MDNs with 2 Gaussians have significantly smaller errors. That is because 2 Gaussians can approximate a one-to-two mapping as shown in Figure 3. The MDN with 2 Gaussians can approximate the MPC control law with a shape of "×". The first Gaussian is in charge of estimating "＼" and the second Gaussian is responsible for estimating "／" so that the overall network can represent the one-to-two mapping. These results imply that MDNs with the same number of Gaussians as the number of the multiple control inputs can ideally represent the set-valued MPC control law.

In summary, MDN-based MPC has three steps: 1) training sample generation: generate several pairs of system states and optimal control inputs as training samples by solving MPC optimal problems for multiple initial states. 2) offline training: train MDNs offline and acquire the MDN control law approximating the MPC control law. 3) online control: repeatedly utilize the pre-trained MDNs to decide the control inputs for the current states.

Numerical Case Study

Problem Description

In the case study, we consider a nonlinear system based on a benchmark problem introduced by Morari and Maeder (2012). The plant model is

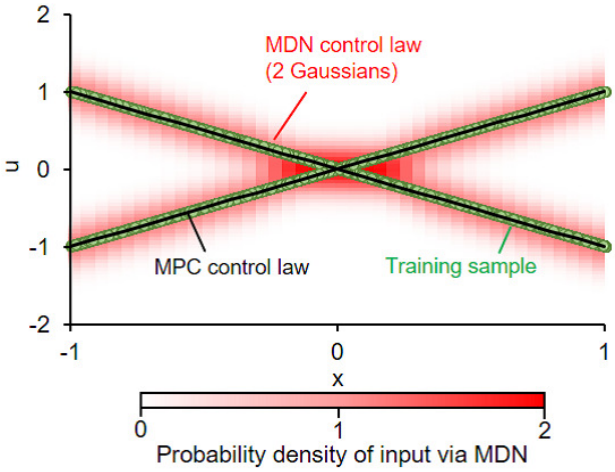


Figure 3: Comparison of MPC and MDN control laws for Example 1.

$$x_1(k+1) = 1.05x_1(k) - 0.25x_1(k)x_2(k) + x_2(k) \tag{9}$$

$$x_2(k+1) = 0.7x_2(k) + u(k) \tag{10}$$

where x_1 and x_2 are elements of the system state and u is the control input which has a constraint $-10 \le u \le 10$. We change the coefficient in the first term of Eq. (9) from 0.95 to 1.05. Since this system converges from any states to $(x_1,\ x_2) = (0,\ 0)$, we set the terminal cost $V_f(x(N))$ to 0. The stage cost is

$$l(x(k), u(k)) = x_1(k)^2 + x_2(k)^2. \tag{11}$$

The prediction horizon N is 20.

Training and Evaluation

Data Generation: To check the control abilities of DNN and MDN-based MPC from various system states, we divide the state range between -10 and 10 into 321 sub-ranges, such as $-10, -9.9375, \cdots, 10$. Then, we solve the corresponding optimal control problem for all 103041 initial states and store all pairs of system states and control inputs as the dataset. Since this problem has lots of local optima for one state, the optimization solver will generate different control inputs for one system state if we solve the optimal problem multi-times with random initial guesses. Therefore, we store up to three best inputs with a difference of more than one. Consequently, the number of the stored pairs is 158225, which has 10163 states with three control inputs, 34858 states with two control inputs, and 58020 states with one control input. To compare the difference in performance between DNNs and MDNs, we divided the stored pairs into a training dataset (40%), a validation dataset (30%), and a test dataset (30%).

Training Setting: The settings for training DNNs and MDNs are shown in Table 3. We evaluated different settings for both DNNs and MDNs on the validation dataset and then report the best settings. We limit σ_{ii} to the range between 0.3 and 0.9 by adjusting $\underline{z}$ and $\bar{z}$. σ_{ij} does not exist because the dimension of u is one.

Table 3: Training setting for DNNs and MDNs.

	DNNs	MDNs
Loss function	MSE	(7)
Activation (output)	Linear	Mixed
Activation (others)	tanh	tanh
Optimizer	Adam	Adam
Learning rate	0.01	0.01
Epoch	200	200
Mini-batch	512	512
Hidden neuron	20	20
Hidden layer	2	2
Number of Gaussians	—	1, 2, 3

Evaluation: We evaluate the performance from two perspectives. The first one is to evaluate the deviation of the predicted control input from the optimal MPC control input. We use the tailored MSE in Eq. (4) to compute the training and test errors. The second one is to evaluate the overall closed-loop performance of the controller. Specifically, different controllers are simulated in the closed-loop system over 20 time steps, starting from every state in test sets. Then we calculate the average values of the cost function. The average cost of ideal MPC is 256.3. Here, because of the input constraints, the predicted inputs from DNNs and MDNs are all projected to be between -10 and 10.

Simulation Result

Since each state may have more than one local optimal input, we prepare the dataset in two ways.

In the first case, for each state, we randomly select one control input from up to 3 best local optimal inputs. This is to simulate the case when we use local solvers to solve optimal control problems only once for each state. Figure 4 shows the MPC control law, and the DNN approximation, and the MDN approximation, when x_1 is fixed to 0. As we expected, the training and test errors of the MDNs reduce as the number of Gaussians increases. In addition, while the DNN can approximate the MPC control law when a state has a unique control input ($x_2 \in [-5,4]$), the DNN cannot represent the MPC control law when multiple control inputs exist for one state ($x_2 \in [4,10]$), despite lots of training samples. The same as the phenomenon we observed in Figure 1 of the Example 1, the predicted input via the DNN is closely the average of multiple inputs. In contrast, the MDN with 3 Gaussians can capture the MPC control law and estimate one of the multiple control inputs. As shown in Table 4, the training errors, test errors, and the average cost of the MDN with 3 Gaussians are all smaller than the DNN. This result shows that DNN-based MPC does not work well if the solver cannot compute a unique optimum because of multiple global or local optima.

In the second case, for each state, we select the control input with the smallest cost from multiple local optimal inputs. This is to simulate the case when we use multi-start to obtain a unique close-to-global-optimal solution. The computational cost of generating data using this approach in practice can be quite high for complicated systems. Figure 5 illustrates the MPC control law and control laws of the DNN and the MDN trained in this way, when x_1 is fixed to 0. Overall, both the DNN and the MDN with 3 Gaussians can approximate the MPC control law relatively accurately. Figure 5b inspects the performance when $x_2 \in [2,6]$ in detail, where the MPC control law is discontinuous over three segments. We observe that the MDN can approximate the discontinuity quite accurately by combining 3 Gaussians while the DNN tends to have errors at the point where the control law changes significantly. Additionally, as shown in Table 5, MDN-based MPC with several Gaussians exhibits better control performance than DNN-based MPC. These results imply that MDN-based MPC with adequate Gaussians outperforms DNN-based MPC unless MPC control laws are regularly lined up.

Table 4: Training and test errors of DNN and MDN trained by the dataset with one randomly selected local optimal input for each state.

Model	DNN	MDN		
Gaussian	–	1	2	3
Training error	2.819	2.999	0.536	0.303
Test error	2.810	2.974	0.519	0.343
Average cost	426.5	480.4	265.0	262.3

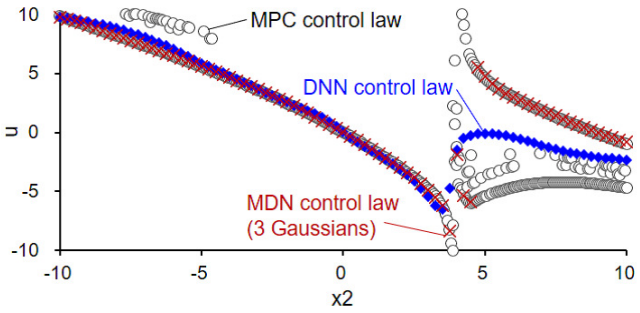


Figure 4: Comparison of MPC and DNN and MDN control law trained by the dataset with one randomly selected local optimal input for each state ($x_1 = 0$).

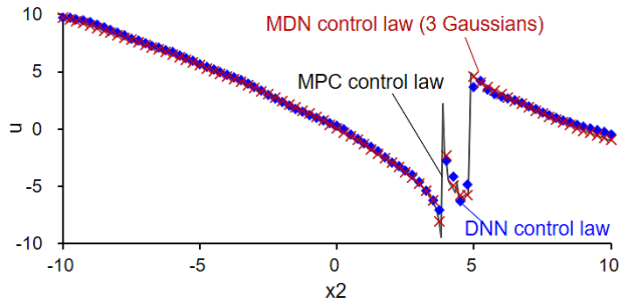
Finally, in terms of online computation time, which is critical for online control, one step computation time of MDN-based MPC with 3 Gaussians is 0.00265 (sec) in Python environment. Compared with the other methods in the same condition, it was slower than the DNN-based MPC, 0.00083 (sec), but faster than the ideal MPC, 0.05513 (sec). The runtime gap between MDN-based MPC and ideal MPC will further expand as the prediction horizon N of the optimization problem increases.

Conclusions

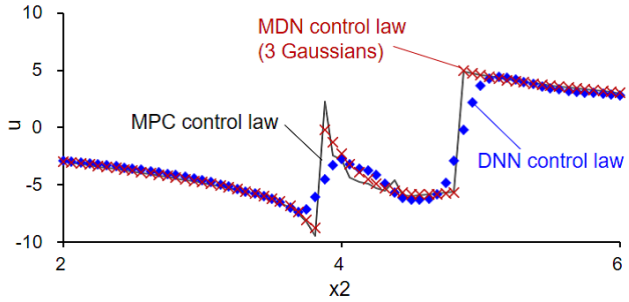
MPC is a control strategy widely applied to a variety of industrial processes. It is challenging for DNNs to approximate the MPC control law of nonlinear systems with set-valued optimal inputs. We propose the approximation of the MPC control law via MDNs to enable approximating the one-to-many mappings. In the case study of the nonlinear benchmark process, MDN-based MPC exhibits high control performance for general nonlinear processes with multiple

Table 5: Training and test errors of DNN and MDN trained by the dataset including one input with the smallest cost for each state.

Model	DNN	MDN		
Gaussian	–	1	2	3
Training error	0.463	0.464	0.190	0.145
Test error	0.453	0.455	0.177	0.161
Average cost	275.6	276.3	263.1	265.4



(a) Overall control law



(b) Detailed control law

Figure 5: Comparison of MPC and DNN and MDN control law trained by the dataset including one input with the smallest cost for each state ($x_1 = 0$).

global or local optima.

Although MDNs with several Gaussians are highly expressive and can represent complex MPC control laws, our proposed method has its limitations. One limitation is that the training of MDNs can be computationally expensive. Although this issue is not fatal for the case study because of its simplicity, it will be very expensive if the dimension of the control inputs is numerous. That is because neurons in the output layer increase as the dimension of the control input increases, which means that the networks have numerous parameters to be learned. In the future, we need to investigate efficient approaches to train MDNs with a large network.

Acknowledgment

Y. C. acknowledges funding from the discovery program of the Natural Science and Engineering Research Council of Canada under grant RGPIN-2019-05499. The authors also gratefully acknowledge the computing resources provided by SciNet (www.scinethpc.ca) and Digital Research Alliance of Canada (www.alliancecan.ca).

References

Bishop, C. M. (1994). Mixture density networks. *Technical Report NCRG/94/004, Neural Computing Research Group, Aston University*.

Cao, Y. and R. B. Gopaluni (2020). Deep neural network approximation of nonlinear model predictive control. *IFAC-PapersOnLine 53(2)*, 11319–11324.

Goodfellow, I., Y. Bengio, and A. Courville (2016). Deep learning. *MIT Press*.

Karg, B. and S. Lucia (2020). Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics 50(9)*, 3866–3878.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*.

Kruse, J. (2020). Training mixture density networks with full covariance matrices. *Visual Learning Lab, Heidelberg University*.

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

Li, Y., K. Hua, and Y. Cao (2022). Using stochastic programming to train neural network approximation of nonlinear mpc laws. *Automatica 146*, 110665.

Lucia, S. and B. Karg (2018). A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine 51(20)*, 511–516.

Morari, M. and U. Maeder (2012). Nonlinear offset-free model predictive control. *Automatica 48(9)*, 2059–2067.

Qin, S. J. and T. A. Badgwell (2003). A survey of industrial model predictive control technology. *Control Engineering Practice 11(7)*, 733–764.

Rawlings, J. B., D. Q. Mayne, and M. M. Diehl (2020). Model predictive control: Theory, computation, and design 2nd edition. *Nob Hill Publishing 3rd printing*.

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

Wu, Z., A. Tran, D. Rincon, and P. D. Christofides (2019). Machine learning-based predictive control of nonlinear processes. part i: Theory. *AIChE Journal 65(11)*.

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

Xi, Y. G., D. W. Li, and S. Lin (2013). Model predictive control — status and challenges. *Acta Automatica Sinica 39(3)*, 222–236.