Chemical Engineering Research and Design 205 (2024) 1–12

Contents lists available at ScienceDirect

Chemical Engineering Research and Design

journal homepage: www.elsevier.com/locate/cherd

Model predictive control of nonlinear processes using transfer learning-based recurrent neural networks

Mohammed S. Alhajeri [a], Yi Ming Ren [b], Feiyang Ou [b], Fahim Abdullah [b], Panagiotis D. Christofides [b,c,*]

[a] Department of Chemical Engineering, Kuwait University, Safat 13060, Kuwait
[b] Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA, 90095-1592, USA
[c] Department of Electrical and Computer Engineering, University of California, Los Angeles, CA 90095-1592, USA

ARTICLE INFO

Keywords:
Transfer learning
Masking technique
Weight sharing
Recurrent neural networks
Model predictive control
Nonlinear processes

ABSTRACT

Artificial neural networks (ANNs), one of the deep learning techniques that has sparked a lot of attention recently for its exceptional modeling capabilities of nonlinear systems, are an essential candidate for model-based control systems. In particular, recurrent neural networks (RNN) have shown remarkable capacity to forecast the dynamic behavior of nonlinear processes using time-series data. In order to regulate the rate at which chemical species are produced in intricate processes, RNNs have successfully served as the predictive model in model-based controllers. However, the necessity for massive amounts of data to capture complex processes is a drawback of typical RNN models. With the goal of efficient utilization of the available data, a different, transfer learning-based training approach for RNNs is presented in this work. The transfer learning strategy is taken into consideration to overcome the challenges stemming from lack of data for the construction of RNN models. In particular, weight-sharing RNNs are developed using *a priori* physical knowledge. Next, a comprehensive analysis of a complex chemical process on a large scale is conducted to showcase the benefits of the suggested approach across various data sizes and its effectiveness when combined with MPC in contrast to conventional RNNs.

1. Introduction

With the continuous development of effective learning algorithms, machine learning is receiving substantial interest across a variety of engineering disciplines. Modeling highly nonlinear dynamical systems is a long standing challenge since first-principles models can generally not be derived for such systems or may not achieve the desired accuracy (Yin and Kaynak, 2015; Ghadami and Epureanu, 2022). In order to represent nonlinear dynamic systems utilizing sequential or time-series data, recurrent neural networks (RNNs) have been frequently used (Wong et al., 2018; Zheng et al., 2022). RNNs have been used in a number of studies to optimize process operation with guaranteed stability and feasibility since they can capture nonlinear dynamics and can be integrated into model predictive controller (MPC) schemes (Pan and Wang, 2008; Xu et al., 2016; Zarzycki and Ławryńczuk, 2021).

Although machine learning (ML) has been found to be an effective modeling tool for nonlinear, complex manufacturing processes (e.g., Ren et al., 2022), the majority of ML modeling is carried out for individual processes using historical data obtained from the particular process of concern and the resulting model cannot then be used to describe a broader class of processes with similar configurations. In addition, the highly accurate ML models for such systems are constructed using large, clean, simulated data sets (e.g., Wu et al., 2022; Alhajeri et al., 2022), which are often unavailable in practice. Since the fidelity of ML models is highly dependent on the size and quality of the training data, the deployment of ML models in practical systems remains challenging because there often is not enough training data available (Baier et al., 2019) due to it being expensive and often impractical to install numerous sensors for measuring diverse physical quantities with a high sampling rate. Nevertheless, the development of RNN models for nonlinear process with limited data remains an unanswered question, primarily due to the high volume of training data required for machine learning modeling of highly nonlinear and complex chemical processes, which is often challenging to acquire in practical settings.

* Corresponding author at: Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA, 90095-1592, USA.
E-mail address: pdc@seas.ucla.edu (P.D. Christofides).

https://doi.org/10.1016/j.cherd.2024.03.019
Received 7 March 2024; Received in revised form 13 March 2024; Accepted 15 March 2024
Available online 20 March 2024

Transfer learning (TL) is a method that entails the transfer of knowledge from a source process abundant in data to a target process with minimal data availability. TL serves as a potent strategy for mitigating data scarcity in the modeling of chemical processes and facilitates transfer of physical knowledge to novel processes (Amabilino et al., 2020). In particular, transfer learning involves the adaptation of a pretrained model originally designed for a data-rich source process to suit a target process with limited data. This approach aims to enhance both learning speed and training performance compared to the primitive method of training a machine learning model for the target process from the ground up without utilizing any prior knowledge, effectively treating machine learning modeling as a purely black-box procedure. Several approaches for transfer learning have been developed, including instance-based, feature-based, parameter-based, and relational-based methods.

Given the high data requirements for neural network (NN) models, transfer learning has been growing in popularity and applicability in many domains where NN models are used, within and beyond the computer science domain. In process systems engineering, TL has been used in a number of various areas, including the development of efficient meta-models to assist high-fidelity computational fluid dynamics simulations (Chuang et al., 2018) and non-invasive monitoring of opaque multiphase flow systems (Lindner et al., 2022). Possibly the widest application of TL in process systems is in fault detection, prognostics, and diagnostics, where TL has been used for improving detection of unknown faults (Wang et al., 2022), for multimode chemical processes where the number of data points available for faulty process operation may vary greatly between the modes or steady-state operating points (Wu and Zhao, 2020; Zhou et al., 2023), and to build models based on simulation data and use a smaller data set from a physical process to improve model performance (Li et al., 2020). In terms of process modeling and optimization, the field of TL is still in its infancy. Laptev et al. (2018) have shown that the incorporation of transfer learning dramatically improves time-series forecasting accuracy in most cases, especially under small to medium training data size conditions. In healthcare applications, Gupta et al. (2020) proposed the usage of transfer learning for time-series classification tasks by adapting a pre-trained RNN on a diverse set of tasks to predict new related tasks. In environmental science applications, Fong et al. (2020) have demonstrated that applying transfer learning on LSTM RNNs for the prediction of air pollutant concentrations leads to not only a higher level of prediction accuracy but also reduced training time. TL has also been applied to find potential design changes to improve the performance of fuel cells (Briceno-Mena et al., 2023). To build predictive models for unseen bioprocesses, Rogers et al. (2022) investigated two different case studies and various network topologies for TL. While the results were promising, the network topology greatly impacted the results, and it is generally not clear for process systems how the topology must be altered, if required, for efficient transfer learning. Xiao and Wu (2023) proposed a physics-based TL methodology to obtain an ML model for a larger chemical process network based on a limited number of subsystems from the network.

An accurate process model is an essential initial step for advanced process control employing model-based controllers, such as model predictive controllers (MPC). Model accuracy remains the primary factor in helping MPC cope with various systems, even with the availability of several tuning parameters. Owing to the aforementioned modeling difficulties, transfer learning offers a chance to quickly construct new NN models for process systems by leveraging old models and scarce new data sets; these models can then be used in MPC if they are judged accurate enough. The development of TL models for MPC has been investigated in Xiao et al. (2023), where generalization error bounds as well as a blueprint for the construction of TL models based on the model capacity and discrepancy between source and target domains were derived.

Motivated by the above considerations, in this work, we propose a transfer learning based RNN architecture, specifically weight-sharing RNN model to incorporate process physical knowledge into RNN modeling and training. Subsequently, the proposed weight-sharing RNN model is incorporated in the design of an MPC to provide predictions of future states for the optimization problem to optimize process performance in terms of closed-loop stability and setpoint tracking. Finally, the RNN-MPC is applied to a chemical process example to demonstrate its improved closed-loop performances in terms of faster convergence to the steady-state under RNN-MPC and enhanced dynamical behavior in terms of lower cost, and faster and smoother responses under RNN based MPC than the controllers using a conventional RNN (C-RNN) model.

2. Preliminaries

2.1. Notation

The notation $|\cdot|$ is used to represent the Euclidean norm, while x^T denotes the transpose of x. A function $f(\cdot)$ is of class C^1 if it is continuously differentiable in its domain. Finally, for variable notation in this paper, x is used to denote two objects depending on the context. In the context of control, x denotes the states within the system. In the context of machine learning, x denotes the input to the model. This is to keep the notation consistent with other work in the fields of both control systems and machine learning.

2.2. Class of systems

In this work we consider a generalized nonlinear multi-input multi-output (MIMO) system with subsystems, where each subsystem depends on its own inputs and the outputs of the previous subsystems. A nonlinear state-space representation is used for each subsystem, with the dynamics of the n subsystems expressed as:

$$\dot{x}_1 = f_1(x_1, u_1, x_0), \quad x_1(t_0) = x_{1,0} \tag{1a}$$

$$\dot{x}_2 = f_2(x_2, u_2, x_1), \quad x_2(t_0) = x_{2,0} \tag{1b}$$

$$\vdots$$

$$\dot{x}_n = f_n(x_n, u_n, x_{n-1}), \quad x_n(t_0) = x_{n,0} \tag{1c}$$

where f_i represents the vector of nonlinear differential equations describing the evolution of the state variables for subsystem i, and t denotes time. The state vector $x_i(t)$ represents the state of subsystem i, and $\dot{x}_i(t)$ is its time derivative. The input vector $u_i(t)$ consists of external inputs to subsystem i, and $x_{i-1}(t)$ represents the outputs of the preceding subsystems that serve as inputs to subsystem i. To model the dependencies, the input vector $u_i(t)$ for subsystem i will also include its own inputs:

$$u_i(t) = \begin{bmatrix} u_{i,1}(t) \\ u_{i,2}(t) \\ \vdots \\ u_{i,m_i}(t) \end{bmatrix}$$

where $u_{i,j}(t)$ is the j^{th} input to subsystem i. The overall model for the entire nonlinear MIMO system, considering the dependencies between subsystems, can be obtained by connecting the equations of each subsystem. Specifically, the outputs of each subsystem become part of the inputs for the subsequent one. Hence, the model can be represented in vector form as:

$$\dot{x} = F(x, u) \tag{2}$$

where $x = [x_1, \ldots, x_n]^T$ and $u = [u_1, \ldots, u_m]^T$. This generalized model captures the nonlinear interactions and dependencies between subsystems in a MIMO system, allowing for flexibility in representing a wide range of nonlinear dynamic systems. The functions f_i would be tailored to the specific dynamics of each subsystem in the context of the overall interconnected system. $F = [f_1, \ldots, f_n]^T$ is assumed to be sufficiently smooth vector functions in its domain.

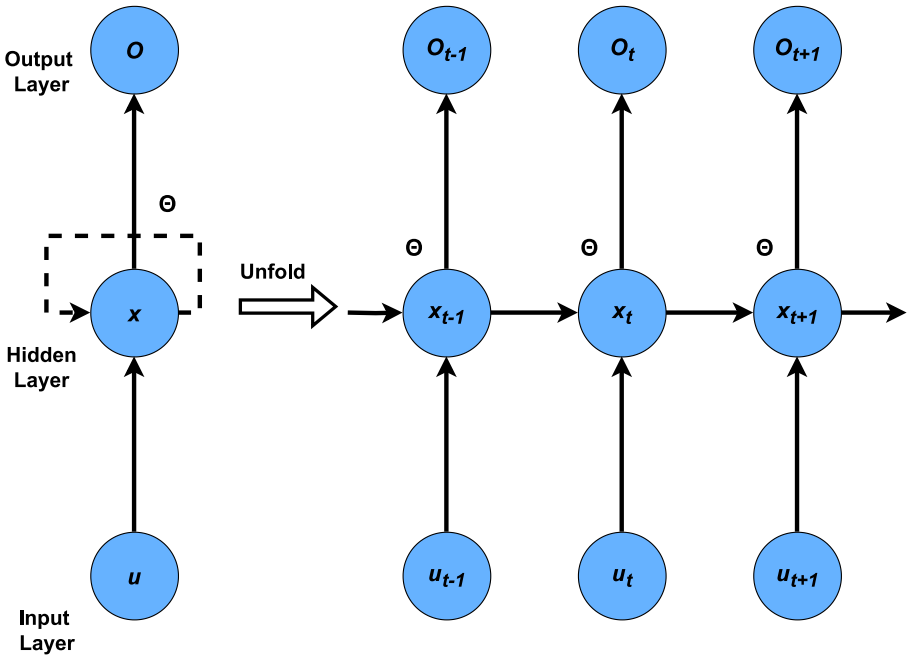

Fig. 1. A schematic of a recurrent neural network.

2.3. Stabilizability assumption

With respect to stability, it is assumed that there exists a stabilizing feedback control law of the type $u = \Phi(x) \in U$, whose goal is to guarantee exponential stability of the origin of the nominal system of Eq. (2) in closed loop. Specifically, there exists a control Lyapunov function of class C^1, represented as $V(x)$, such that, for all x inside an open neighborhood D around the origin, the following inequalities hold:

$$c_1|x|^2 \leq V(x) \leq c_2|x|^2, \tag{3a}$$

$$\frac{\partial V(x)}{\partial x} F(x, \Phi(x)) \leq -c_3|x|^2, \tag{3b}$$

$$\left|\frac{\partial V(x)}{\partial x}\right| \leq c_4|x| \tag{3c}$$

where c_1, c_2, c_3, c_4 are all positive real numbers. The controller $\Phi(x)$ can be constructed in the form of the universal Sontag control law proposed in Lin and Sontag (1991). First, we can designate a region associated with the stabilizing controller $u = \Phi(x) \in U$ where the time-derivative of V occurs to be negative (i.e., Eq. (3b) holds), according to Wu et al. (2019). Then, given $\rho > 0$ and $\Omega_\rho = \{x \in \mathbb{R}^n \,|V(x) \leq \rho\}$, the closed-loop stability region Ω_ρ is found to be a level set of $V(x)$ within the area D.

2.4. Recurrent neural network (RNN)

A recurrent neural network (RNN) is a type of machine learning models that is well-suited for approximating dynamical systems due to its inherent ability to capture temporal dependencies and sequential patterns within data sequences based on its architecture allowing for previous information to influence current and future outputs. Specifically, the RNN is able to retain the memory of the prior states because to the recursive activity in the hidden layer neurons, which helps with time-series dataset approximation. Using process operational data, the RNN model employed in this work for modeling the nonlinear system of Eq. (2) can be expressed as follows:

$$\dot{\bar{x}} = F_{nn}(\bar{x}, u) := A\bar{x} + \Theta^T y \tag{4}$$

where the manipulated input vector is $u = [u_1, \ldots, u_m]$, and the state vector of the RNN is represented by $\bar{x} = [\bar{x}_1, \ldots, \bar{x}_n]$. The vector y is expressed as $[y_1, \ldots, y_n, y_n + 1, \ldots, y_{n+m}]$ as it is a vector of both $\bar{x}$ and u. In $\mathbf{R}^{n+m}$, $y = [\sigma(\bar{x}_1), \ldots, \sigma(\bar{x}_n), u_1, \ldots, u_m]$. The nonlinear activation function (e.g., a hyperbolic tangent function) utilized in the hidden layers is denoted by the notation $\sigma(\cdot)$. $A = \text{diag}[-a_1, \ldots, -a_n]$ is a diagonal and negative coefficient matrix, where $a_i > 0$ such that each state $\bar{x}$ is stable in the terms of bounded-input bounded-state stability. $\Theta = [\theta_1, \ldots, \theta_n] \in \mathbf{R}^{(n+m)\times n}$ represents a matrix containing associated weights (i.e., θ) to be optimized throughout the neural network training process. Each vector $\theta_i = b_i[w_{i1}, \ldots, w_{i(n+m)}]$ is an element of Θ where b_i is a constant, and w_{ij} denotes the weight on the linkage that connects the j^{th} input to the i^{th} neuron where $j = 1, \ldots, (n+m)$ and $i = 1, \ldots, n$ (see Fig. 1).

Subsequently, the RNN is trained following a standard learning procedure as discussed in Wu et al. (2019) and Alhajeri et al. (2022). The datasets for training, validation and testing are generated from extensive open-loop simulations of the process model under sufficient variation of initial conditions and control actions. In particular, the continuous-time system of Eq. (2) is numerically integrated using the explicit Euler method with an appropriately small integration time step h_c, and the control actions u are implemented in a sample-and-hold fashion, i.e., $u(t) = u(t_k)$, $\forall t \in [t_k, t_{k+1})$, where $t_{k+1} := t_k + \Delta$, and Δ denotes sampling period. RNNs are known for their ability to capture nonlinear dynamic behavior from time-series data (Park and Sandberg, 1991; Chen and Chen, 1995). Furthermore, RNNs can be trained using all or some of the integration step data points (i.e., data at each integration time step h_c) within each sampling period to capture the state evolution. Finally, the RNN model is required to achieve a sufficiently small modeling error ν (i.e., $|\nu| = |F(x,u) - F_{nn}(\bar{x},u)| \leq \gamma|x| \leq \nu_{\min}$, where $\gamma, \nu_{\min} > 0$) during the model training process, to ensure that it can accurately represent process dynamics within the considered operating region.

Remark 1. The modeling error, denoted as ν, varies across different inputs and states, and it is not a constant. However, constraining the operation to the stability region Ω_ρ ensures that both inputs and states remain bounded. Consequently, training the RNN model using datasets generated within Ω_ρ allows the modeling error to be upper bounded by a sufficiently small positive value $\nu_{\min}$ for all states within the stability region (Golowich et al., 2018; Wu et al., 2022; Alhajeri et al., 2023).

3. Leveraging prior knowledge

3.1. Transfer learning: Use of structure and weights

Transfer learning is a technique used in deep learning to take advantage of the knowledge learned in trained reference models to train similar models. According to Bozinovski (2020), the basis of this method roots back to a report published in the 70 s, which was then researched and developed further for numerous applications afterwards (Tan et al., 2018).

In neural network training, weight initialization is a significant factor in determining a neural network's performance, instigating a breadth of literature in this domain in recent years (Narkhede et al., 2022). Initializing the weights and bias of a new model with the trained weights and bias of a reference model can be one effective way to reuse the prior knowledge (Yosinski et al., 2014). This method requires the input dimension, output dimension and network structure of the new model to be exactly the same as the reference model. The real-world application of the aforementioned weight initialization method in the area of chemical processes can take a variety of forms, such as the case when the catalyst in a chemical reactor is changed, either due to reduced activity or replacement with a new type of catalyst entirely. As only the activation energy of reaction changes, with no other change in state variables and controlled variables, the input and output dimensions of the network remain exactly the same as the model trained with the data generated with the old catalyst in use. Nevertheless, when the number of state variables and/or controlled variables change in the modeling process, extraction and modifications of the weights of the pre-trained model is required to transfer the knowledge of the reference model to the other model. One approach to solve this problem is to create a new model by adding fully connected layer before the input layer and after the output layer of reference model to adapt the change of dimension of input and output dimensions. To preserve the prior knowledge saved in reference model, the middle reference model part of the new model is set to be non-trainable, only train the two fully connected layers. After that, the middle reference model part is reset to trainable to fine tuning the result model. Since preserving prior knowledge is still essential in the fine tuning step, the learning rate should be set to an extremely low value to prevent losing of knowledge. While a low learning rate may generally decelerate the model training process, the re-training of the model for subsequent subsystems occurs with significantly smaller data sets. As a result, each epoch takes a significantly shorter time, and the overall transfer learning model training occurs within a reasonable time even with the low value of the learning rate. The commonly used value is 10^{-5} according to Keras Tutorial Documents (Gulli and Pal, 2017).

3.2. Weight-sharing RNN: Efficient use of data and training method

In transfer learning, neural network trained from a large dataset (ex: MNIST) is used to fit a smaller (more specific) dataset by freezing the early layer weights and only fine-tuning the latter layer weights. In the proposed weight-sharing RNN (WS-RNN), the weights are not frozen and are optimized normally during backpropagation. Rather than reusing model structure and parameters, the WS-RNN changes the model structure to more efficiently use the existing datasets. Specifically, the WS-RNN excels at modeling a system with multiple similar subsystems compared to traditional RNNs. The main motivation behind WS-RNN is that we hope to leverage all subsystems' known similarities to improve the learning of complex dynamics. Instead of training separate neural network models for each subsystem, a generalized neural network model is trained for the overall system. In the proposed WS-RNN, we change the input and output layers of the neural network to leverage multiple subsystem states (x_1, x_2, …, x_n) and inputs (u_1, u_2, …, u_n) so that it can predict future subsystem states (x_1^+, x_2^+, …, x_n^+).

This formulation is similar to the approach of using a regular RNN that models the system as a whole as both neural networks will take all states, $X = [x_1, \ldots, x_n]$, and manipulated inputs, $U = [u_1, \ldots, u_m]$, as model inputs and the future system states, $\bar{X} = (x_1^+, x_2^+, \ldots, x_n^+)$, as outputs at $t = t_k + \Delta$. A problem associated with this type of input–output formulation is that, in many cases, there might not be enough data and, in certain scenarios, not all subsystems will have the same amount of data. When there is an insufficiency or imbalance of data between subsystems, this type of neural network model can only use data up to the subsystem with the least amount of data. In addition, each subsystem's data also must be from the same experiment and recorded at the same time as all other subsystems to ensure the data sampling is consistent, otherwise causing further data usage limitations. As a result, it is critical to address this issue by establishing input independence across the subsystems.

In the proposed weight-sharing RNN, which will be referred to as TL-RNN, we attempt to alleviate these problems through the use of a masking technique by hiding certain inputs and outputs for different subsystems, and thus creating independence. In other words, while the input and output shapes for the TL-RNN are constant, during training and inference, each subsystem is trained and evaluated using independent datasets with masks. Specifically, the inputs and outputs of the TL-RNN will be masked with out-of-bound values (i.e., generally -1 if using data scaled from 0 to 1, or $-\infty$ otherwise). Correspondingly, the loss function also needs to be altered appropriately so that states assigned with the masked values are skipped during the loss function calculation as shown in Eq. (5) below:

$$F_{loss}(\bar{x}_t, \bar{x}_p) = \begin{cases} \bar{x}_t^2 - \bar{x}_p^2 & \bar{x}_t \neq \text{mask\_val} \\ 0 & \bar{x}_t = \text{mask\_val} \end{cases} \tag{5}$$

where F_{loss} is the loss function for each sample, while $\bar{x}_t$ and $\bar{x}_p$ are the true and the predicted output, respectively. The loss function is changed in a way so that masked states do not affect the gradient optimization during training. As the contribution of the masked data points to the loss function or accuracy metric is zeroed, the masking value itself does not have any impact on the training or accuracy metrics. The mask value is simply chosen in a manner that allows the masked data points to be easily filtered out of the training data set based on the valid range of the variable(s). The input layer is where all inputs will be fed into the model. In particular, the masked inputs will also be assigned values that are outside of the range; as a result, the weights associated with the masked inputs will be forced to be extremely small to maintain validity of the output of the layer for forward propagation, causing the training to focus on the unmasked inputs. These changes to the input, output, and loss function will allow the TL-RNN to recognize which subsystem it is predicting through the mask placements.

While each subsystem is trained using independent datasets, the training of the weights themselves is not independent as all subsystems share the same weights within the TL-RNN. As the TL-RNN weights are optimized using one subsystem's data, these weights will also affect the prediction of all other subsystems. An advantage of dynamically sharing weights is that the shared components between subsystems can be learned easier because the initial model, trained with a large dataset, can already capture these dynamics and will not need significant retraining even when the model is generalized to the larger system. Furthermore, all subsystems' data contribute to the training of this shared aspect and will only improve the model further when newer, smaller datasets are added. For example, if two reactors share the same underlying reaction mechanism despite having different hardware specifications (size, input flow rates), the TL-RNN can learn this complex reaction through the data from both reactors. Conversely, if the two subsystems differ significantly in their underlying mechanisms, then this weight sharing property of the TL-RNN will result in performance deterioration rather than improvement.

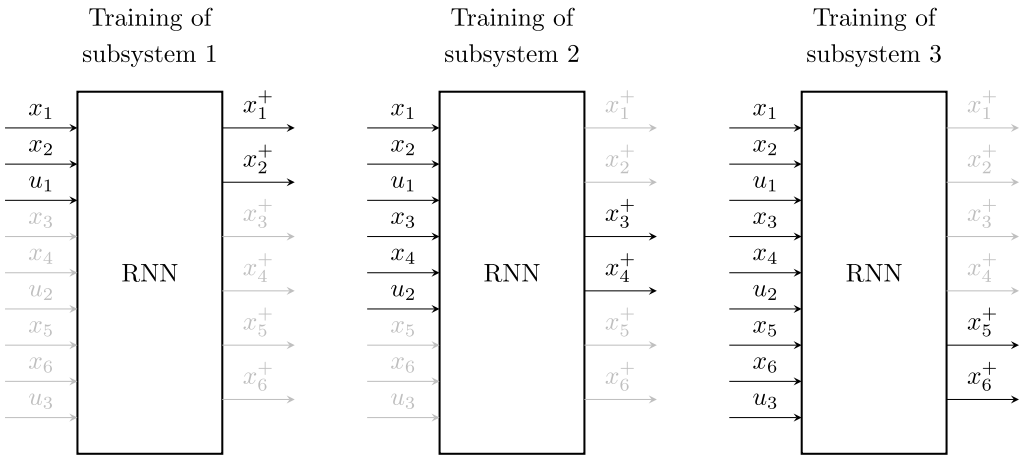

Fig. 2. Masking procedure of WS-RNN training data set for a system comprising of $n = 3$ subsystems with 3 variables (2 states and 1 manipulated input) per subsystem. Variables in gray are masked.

Remark 2. It is important to note that, as the number of subsystems n increases, the number of layers and number of neurons within each recurrent unit also need to increase to compensate for the increased system complexity. While theoretically, as n becomes very large, the WS-RNN will not suffer deteriorating model performance, in practice, longer training times and memory consumption requirements must be considered before increasing n.

The WS-RNN training procedure for a system of n subsystems consists of n training phases. In the i^{th} subsystem's training phase, the inputs of the downstream subsystems, i.e., $(i+1)^{\text{th}}$, $(i+2)^{\text{th}}$, …, n^{th} are masked. Additionally, the outputs of all other subsystems, i.e., 1^{st}, 2^{nd}, …, $(i-1)^{\text{th}}$, $(i+1)^{\text{th}}$, …, n^{th} are also masked. In this manner, two advantages are expected: (1) only upstream subsystems' inputs are allowed to affect the i^{th} subsystem's output, better representing the physics and (2) by masking upstream subsystems' outputs, the model only captures the dependencies between the inputs and specifically the i^{th} subsystem's outputs at the respective training phase.

To illustrate the WS-RNN training, consider a system comprising of 3 subsystems ($n = 3$) with an original data set represented as DS. The development of a TL-RNN model for the described system is as follows:

1. Mask all inputs and outputs of subsystems $2 \rightarrow 3$ in the training data set to create a new data set denoted as DS_{M_1}, where DS_{M_i} denotes all inputs of subsystems $i+1$ and above as well as all outputs other than that of subsystem i being masked.
2. Develop a conventional RNN model with the dataset DS_{M_1}.
3. For subsystem 2, mask all inputs of subsystem 3 and all outputs of subsystems 1 and 3 in the training data set to create a new data set denoted as DS_{M_2}.
4. Retrain the developed RNN model of Step 2 following Pseudocodes below.
5. For subsystem 3 (i.e., n^{th}), mask the outputs of subsystems 1 and 2 to create a data set DS_{M_3} to retrain the model developed in Step 4.

The above training procedure is depicted in Fig. 2 for a system with 3 subsystems, where each subsystem is characterized by 2 state variables and 1 input variable. Specifically, subsystem 1 consists of states x_1 and x_2 and manipulated input variable u_1, subsystem 2 consists of states x_3 and x_4 and manipulated input variable u_2, and subsystem 3 consists of states x_5 and x_6 and manipulated input variable u_3. The RNN model takes as input the 6 state variables and 3 manipulated inputs for a total of 9 variables to predict the 6 state variables in the future, which are the RNN model output variables.

Remark 3. The knowledge transferred between models are (a) the core dynamics of the process as well as (b) interconnectivity between subsystems within the overall system. Specifically, the reactions within each reactor do not change and the dynamics are captured by the initial model built using a large data set from the first reactor. This knowledge is transferred in the proposed weight-sharing approach since the initial model is only fine-tuned in subsequent training phases with low values of the learning rate. With respect to interconnectivity between subsystems, the fact that upstream processes and tasks affect downstream ones but not vice versa is enforced at every training phase and the knowledge of how the state prediction of each subsystem is dependent on its inputs is transferred between training phases. For example, in Fig. 2, the effects of x_1, x_2, u_1 on x_1^+ and x_2^+ are already captured in the initial RNN model and transferred to the second model in the training of subsystem 2. Since state predictions x_1^+ and x_2^+ are masked in training phase 2, the effects of x_1, x_2, u_1 on only x_3^+ and x_4^+ are captured in the second training phase. Similarly, the in the third and final training phase, the knowledge of the first 6 variables' dependence on the first four states' predictions that has already been captured is transferred.

4. Neural network-based model predictive control (MPC)

This section describes the integration of an RNN model into a Lyapunov-based model predictive controller (LMPC), with the RNN model providing state predictions for the cost function estimation. The MPC optimization problem is formulated as follows:

$$\mathcal{J} = \min_{u \in S(\Delta)} \int_{t_k}^{t_{k+H_P}} L(\tilde{x}(t), u(t))\, \mathrm{d}t \tag{6a}$$

$$\text{s.t.} \quad \dot{\tilde{x}}(t) = F_{nn}(\tilde{x}(t), u(t)) \tag{6b}$$

$$\tilde{x}(t_k) = x(t_k) \tag{6c}$$

$$u(t) \in U, \ \forall\ t \in [t_k, t_{k+H_P}) \tag{6d}$$

$$\dot{V}(x(t_k), u) \leq \dot{V}(x(t_k), \Phi(x(t_k))), \quad \text{if } x(t_k) \in \Omega_\rho - \Omega_{\rho_{nn}} \tag{6e}$$

$$V(\tilde{x}(t)) \leq \rho_{nn}, \ \forall\ t \in [t_k, t_{k+H_P}), \ \text{if } x(t_k) \in \Omega_{\rho_{nn}} \tag{6f}$$

where the predicted state trajectory is donated by $\tilde{x}$. $S(\Delta)$ represents the set of constant piecewise functions with period Δ. For this MPC scheme, the prediction horizon is denoted by H_P. The function $\dot{V}(x,u)$ in Eq. (6e) represents the time-derivative of the Lyapunov function V (i.e., $\frac{\partial V(x)}{\partial x} F_{nn}(x,u)$). The optimal input series $u^*(t)$ is computed by the LMPC across the given prediction horizon $t \in [t_k, t_{k+H_P})$. The system is instructed to implement the first optimal input $u^*(t_k)$ for the upcoming sampling period, after which, the LMPC receives the new

Pseudocode 1: Transfer-learning based RNN model construction via weight-sharing

Load the Model
{
Pre-trained model=load.model('RNN model')
}
Training Data
{
New training data=load('DS_{M_i}')
}
Create new RNN model with the same architecture as the pre-trained model
{
input layer:
layer class: Input
units: (number of input features in vector u)
input shape: (number of data points in the input data sequence, number of input features in vector u)
connected to: hidden layer

hidden layer:
layer class: Long Short-term Memory
units: (Number of inputs) × (Number of outputs)
return sequences: true
activation function: hyperbolic tangent function (tanh)
recurrent activation function: sigmoid
recurrent initializer: orthogonal
use bias: true
connected to: output layer

output layer:
layer class: Dense
units: number of output
activation function: linear
output shape: (number of data points in the output data sequence, number of outputs in vector x)
}

Weights sharing
{
New model weights= get.weights('Pre-trained model')
}

Pseudocode 2: Transfer-learning based RNN model training

Model compile
{
optimizer: Adam (other optimizers: RMSprop, SGD, etc.)
loss function: mean squared error (other metrics can be utilized)
}

Early stop
{
monitor: validation loss
early stop condition: 1×10^{-8}
}

Split of data set DS_{M_i}
{
x_t, y_t, x_v, y_v=*train_test_split*(DS_{M_i}, *split_ratio=0.8*)
}

Model fit using DS_{M_i} **data set**
{
training:(x_t, y_t) *:*
x_t*: python list (input training set for input layer)*
y_t*: python list (output training set for output layer)*
batch size: 32 (defaults value)
epochs: 50 (user choice, other numbers can be used)
validation:(x_v, y_v) *:*
x_v*: python list (input validation set for input layer)*
y_v*: python list (output validation set for output layer)*

callbacks: early stop
}

state measurements and uses them to solve the control optimization problem once more during the subsequent sampling period. Moreover, the objective of the MPC optimization problem is to minimize $L(\tilde{x}, u)$, which is the time integral of the cost function, as represented in Eq. (6a), over the prediction horizon H_P, while fulfilling the constraints of Eqs. (6b)–(6f). The first constraint of Eq. (6b) is the RNN model of Eq. (4) that is utilized to predict the evolution of the closed-loop state. In Eq. (6b), $x(t_k)$ is used as the initial condition of the RNN model state vector $\tilde{x}$. As for the inputs, their constraints are represented by Eq. (6d), which are applied throughout the entire prediction horizon.

To maintain closed-loop stability, the contractive constraint of Eq. (6e) is activated when $x(t_k) \in \Omega_\rho - \Omega_{\rho_{nn}}$, i.e., when the state is not yet within the final desired level set $\Omega_{\rho_{nn}}$. This constraint forces the Lyapunov function of the closed-loop states to decrease under the LMPC and, as a result, the actual state will approach the origin in finite time, eventually entering $\Omega_{\rho_{nn}}$. Once the state $x(t_k)$ enters the desired region $\Omega_{\rho_{nn}}$, then the predicted closed-loop state will be maintained within this region for the entire prediction horizon as per Eq. (6f). The work of Wu et al. (2019) demonstrated that, when using RNN-based LMPC to control a nonlinear system as that of Eq. (2), the closed-loop state is guaranteed to be bounded within the stability region Ω_ρ for all times, and ultimately will converge to a small neighborhood around the origin under the assumption that the modeling error ν is sufficiently small.

Remark 4. A state observer can be used to estimate the unmeasured states from the measured ones in the event that $x_1, \ldots, x_n$ are not entirely available via sensors. Two distinct machine learning-based state estimators were created in an earlier work (Alhajeri et al., 2021) within the framework of ML-based LMPC for nonlinear systems. It was demonstrated that all state trajectories starting from different initial conditions converged to the steady-state under the LMPC, with accurate state estimates being obtained by both the hybrid-model based estimator and the ML-based estimator.

5. Application to chemical process example

In this section, we use a chemical process example to evaluate the proposed weight-shared RNN-based LMPC. Firstly, we develop a dynamic model for a chemical process using first-principles modeling principles. Subsequently, a time-series dataset of the process operation is collected to train and test the RNN models via extensive open-loop simulations of the first-principles model. Finally, open- and closed-loop simulations under the RNN-model based LMPC are carried out and the results discussed.

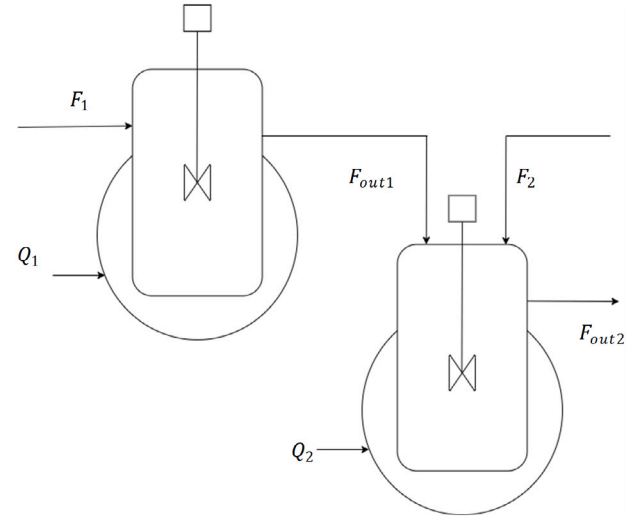

Fig. 3. Two continuous-stirred tank reactors in series.

Reactants A and B are used as raw materials in the production process of the desired chemical D, as shown in Fig. 3. This process involves two successive non-isothermal, well-mixed continuous stirred tank reactors (CSTR) with a second-order, exothermic, irreversible primary reaction and two side reactions. The mechanism of these three reactions is described in Eq. (7) below:

$$A + B \rightarrow D \quad \text{(main)} \tag{7a}$$

$$A + D \rightarrow UD \tag{7b}$$

$$B + UD \rightarrow 2\ D \tag{7c}$$

By applying the tools of mass and energy balances, the first-principles models for the CSTRs can be developed. Specifically, the dynamic model of the first CSTR is represented by the following ODEs:

$$\frac{dC_{A_1}}{dt} = \frac{F_1 C_{Ao_1} - F_{out_1} C_{A_1}}{V_1} - r_1 - r_2 \tag{8a}$$

$$\frac{dC_{B_1}}{dt} = \frac{F_1 C_{Bo_1} - F_{out_1} C_{B_1}}{V_1} - r_1 - r_3 \tag{8b}$$

$$\frac{dC_{D_1}}{dt} = \frac{-F_{out_1} C_{D_1}}{V_1} + r_1 - r_2 + 2r_3 \tag{8c}$$

$$\frac{dC_{UD_1}}{dt} = \frac{-F_{out_1} C_{UD_1}}{V_1} + r_2 - r_3 \tag{8d}$$

$$\frac{dT_1}{dt} = \frac{(T_{0_1} F_1 - T_1 F_{out_1})}{V_1} + \sum_{j=1}^{3} \frac{-\Delta H_j}{\rho_1 C_p} r_j + \frac{Q_1}{\rho_1 C_p V_1} \tag{8e}$$

while the dynamic model of the second reactor is given by the ODEs below:

$$\frac{dC_{A_2}}{dt} = \frac{F_2 C_{Ao_2} + F_{out_1} C_{A_1} - F_{out_2} C_{A_2}}{V_2} - r_1 - r_2 \tag{9a}$$

$$\frac{dC_{B_2}}{dt} = \frac{F_2 C_{Bo_2} + F_{out_1} C_{B_1} - F_{out_2} C_{B_2}}{V_2} - r_1 - r_3 \tag{9b}$$

$$\frac{dC_{D_2}}{dt} = \frac{F_{out_1} C_{D_1} - F_{out_2} C_{D_2}}{V_2} + r_1 - r_2 + 2r_3 \tag{9c}$$

$$\frac{dC_{UD_2}}{dt} = \frac{F_{out_1} C_{UD_1} - F_{out_2} C_{UD_2}}{V_2} + r_2 - r_3 \tag{9d}$$

$$\frac{dT_2}{dt} = \frac{(T_{0_2} F_2 + T_1 F_{out_1} - T_2 F_{out_2})}{V_2} + \sum_{j=1}^{3} \frac{-\Delta H_j}{\rho_2 C_p} r_j + \frac{Q_2}{\rho_2 C_p V_2} \tag{9e}$$

where the reaction rates are functions of concentrations and temperature, and can be computed via the functions below, with $i = 1, 2$:

Table 1
Parameter and steady-state values for the CSTRs.

$T_{1_o} = 350$ K	$T_{1_s} = 310.523$ K
$T_{2_o} = 350$ K	$T_{2_s} = 430.542$ K
$C_{A_1} = 4.2455$ kmol/m^3	$C_{A_2} = 0.3254$ kmol/m^3
$C_{A_o} = 4$ kmol/m^3	$C_{B_o} = 5$ kmol/m^3
$C_{B_1} = 5.3532$ kmol/m^3	$C_{B_2} = 1.3841$ kmol/m^3
$C_{D_1} = 0.1854$ kmol/m^3	$C_{D_2} = 3.8744$ kmol/m^3
$C_{UD_1} = 9.1426 \times 10^{-7}$ kmol/m^3	$C_{UD_2} = 0.0058$ kmol/m^3
$k_1 = 1.528 \times 10^6$ m^3/kmol/s	$F_1 = 43.2$ m^3/h
$k_2 = 2.778 \times 10^6$ m^3/kmol/s	$F_2 = 91.079$ m^3/h
$k_3 = 0.4167$ m^3/kmol/s	$Q_{1\ s} = 911.455$ kJ/s
$E_1 = 71160$ kJ/kmol	$Q_{2\ s} = 6835.270$ kJ/s
$E_2 = 83680$ kJ/kmol	$T_{2_o} = 300$ K
$E_3 = 62760$ kJ/kmol	$V_1 = 60$ m^3
$Cp = 2.411$ kj/kg/K	$V_2 = 60$ m^3
$R = 8.314$ kJ/(kmol K)	$\Delta H_1 = -1.04 \times 10^6$ kJ/kmol
$\rho_1 = 683.7$ kg/m^3	$\Delta H_2 = -1.02 \times 10^6$ kJ/kmol
$\rho_2 = 607.5040$ kg/m^3	$\Delta H_3 = -5.05 \times 10^2$ kJ/kmol

$$r_1 = k_1 e^{-\frac{E_1}{RT_i}} C_{A_i} C_{B_i} \tag{10a}$$

$$r_2 = k_2 e^{-\frac{E_2}{RT_i}} C_{D_i} C_{A_i} \tag{10b}$$

$$r_3 = k_3 e^{-\frac{E_3}{RT_i}} C_{UD_i} C_{B_i} \tag{10c}$$

F_i and F_{out_i} represent the inlet and outlet flow rates, respectively, of stream i (where $i = 1, 2$) of the system. Furthermore, the notations T_i, and Q_i represent the reactor temperature, and the heat supply rate, respectively, of the i^{th} reactor. For the reactants and products concentrations, we use the notation C_{l_i}, where $l = A, B, D, UD$. The subscript "o" is used to indicate the initial state of the process variables. V_i is the volume of the reacting liquid, which has a density of ρ and a heat capacity of C_p that are constant for both reactors. R is the ideal gas constant. ΔH_j, k_j, and E_j denote the j^{th} reaction's enthalpy, pre-exponential constant, and activation energy, respectively, and these parameters are unchanged for both reactors. Process parameter values are listed in 1.

The manipulated inputs for this process are the heat supply rate to both reactors (i.e., Q_1 and Q_2), which are represented in deviation form from their steady-state values as $u_1 = Q_1 - Q_{1s}$ and $u_2 = Q_2 - Q_{2s}$. The upper and lower bounds on the inputs are $[U^{\max}, U^{\min}] = [5, -5] \times 10^5$ kJ/h, respectively. Also the ten states are represented in deviation fashion from their steady-state values, such that the origin is the equilibrium point of the state-space representation of the underlying system.

5.1. Data generation and RNN models development and selection

It is well known that a large amount of data is required for the development of neural network models. Furthermore, generally, if the data is independent and identically distributed, the larger the size of the data set, the more accurate the model can be (Sugiyama, 2015; Wu et al., 2021). Hence, in this work, data was collected by simulating the first-principles model of Eqs. (8)–(10) under different inputs while recording all inputs and generated output values. Numerical integration was used, specifically, Euler's integration method with a small enough integration step h_c to ensure stability and accuracy of the integration.

Using the machine learning library, Keras, the two RNN models are constructed and trained regularly following Wu et al. (2019), where all the process inputs and states at $t = t_k$ are used as inputs for the model to predict the process states at $t = t_k + \Delta$.

For the TL-RNN, we developed different models based on different portions of the training data set, as demonstrated in Fig. 4. Furthermore, each model was trained over two phases, where in phase one, only the first CSTR's input and five states are considered. This step is done by masking the second CSTR's states and input, $x_6 - x_{10}$, and u_2.

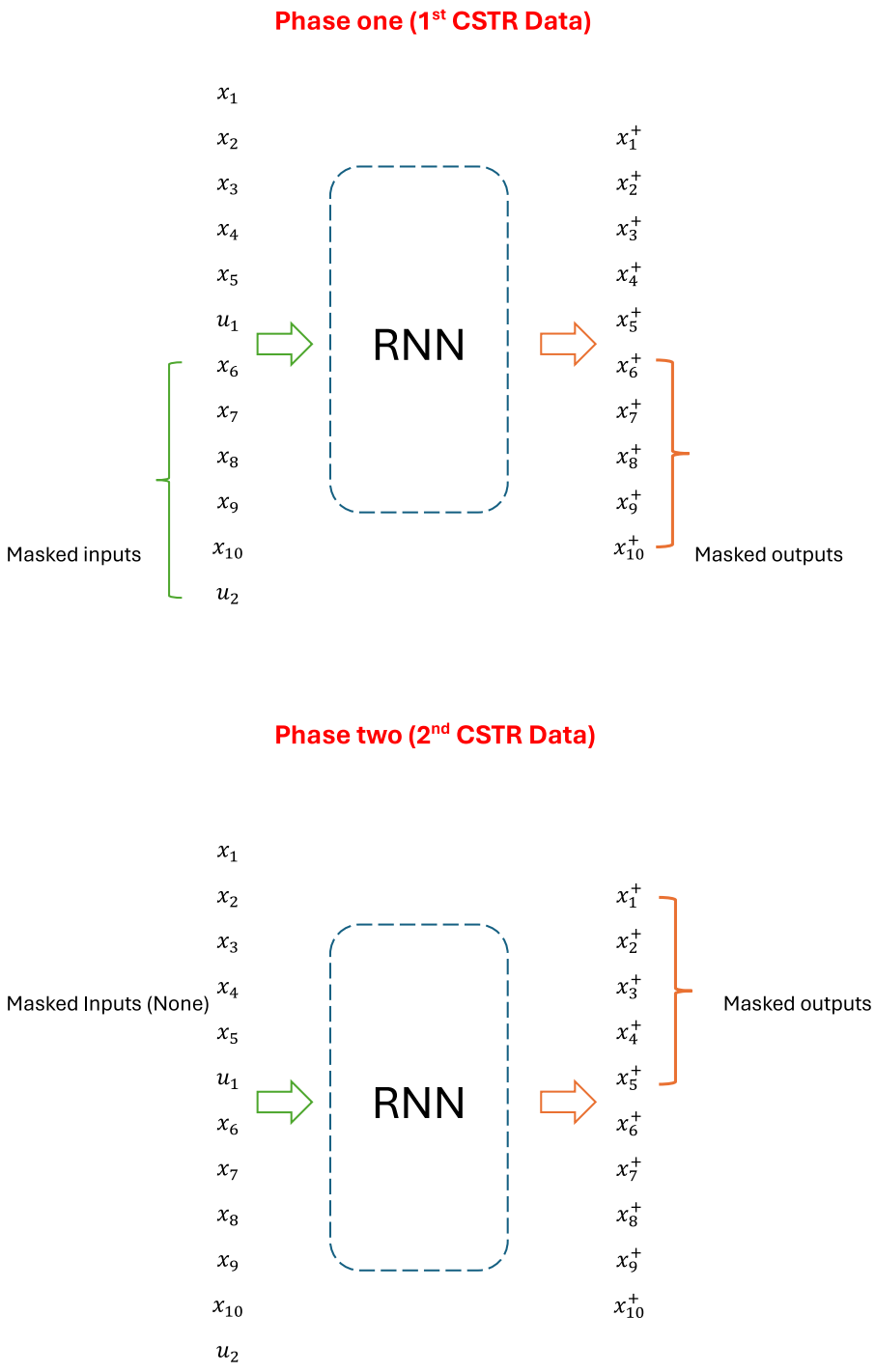

Fig. 4. TL-RNN model training schematic, where x_i and u_i at $t = t_k$, and x_i^+ is at $t = t_k + \Delta$.

Subsequently, for phase two, the developed model was retrained but for predicting the dynamics of the second reactor's states, with the first CSTR states and inputs masked. Technically, the weights of this model were basically transferred/shared from phase one to phase two, thus resulting in a transferred knowledge model via weight-sharing. This model is anticipated to be able to predict the dynamics of both CSTRs more accurately than the conventional RNN model.

Following model development, the investigation of how data size (i.e., data availability) affects the accuracy of the TL-RNN model compared to the C-RNN models is performed. We develop several models based on the two aforementioned methods using different training data set sizes, and then test all of them on a designated data set that is not included in the training phase. Fig. 5 shows the overall testing mean squared error (MSE) values for this study.

This comparison is essential in order to see how well each model performs on a set of unseen initial conditions. The total training dataset has 1,000,000 initial conditions and we take different, random subsets of the 1,000,000 data points to train the different models and evaluate their performance accordingly. Notably from Fig. 5, the TL-RNN model outperforms the C-RNN at all data sizes. In addition, the difference

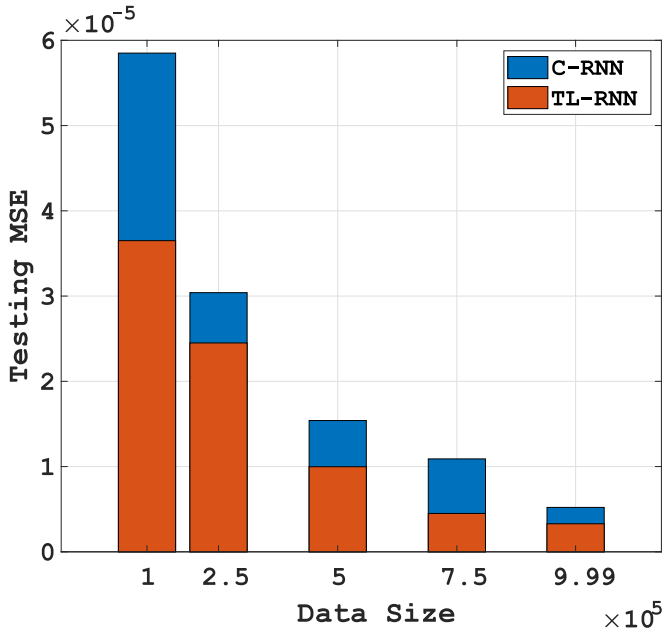

Fig. 5. Overall testing MSE results of both the TL-RNN and the C-RNN models based on different training data sizes.

between the two models increases exponentially as the training data size decreases, which aligns with our motivation of using data more efficiently.

Next, we investigate the performance of predicting individual CSTR states. During testing, we recorded the MSE values for each CSTR, which are reported in Table 2. At larger data sizes, both models perform relatively similar, which is as expected since there are enough data points to capture the first CSTR's input–output relationship. However, as we decrease the data size, the C-RNN performs much worse than the TL-RNN for predicting the first CSTR's states. The reason for that result roots to the fact that C-RNN model takes all the inputs and relate them to all the outputs without relying on the existence of known physical relations. This is more notable in the smaller data set scenarios.

Finally, for the second CSTR state predictions, the same procedure was followed to find and tabulate the MSE values. As seen from Table 2, TL-RNN always achieves a lower MSE than C-RNN, typically an order of magnitude lower. This might be due to the fact that TL-RNN only gives updates weight values to the relevant inputs to the neural network based on the integrated physical knowledge (e.g., we know that $x_1 - x_5$, and u_1 do not affect the second CSTR's states directly as only the outputs, $x_1^+ - x_5^+$, are relevant).

Remark 5. Although the proposed WS-RNN approach may seem similar to the partially connected RNNs approach (Alhajeri et al., 2022; Xu et al., 2021), in the WS-RNN, we are not using different RNN structure or connections and weights, but instead we are sharing weights by masking to achieve this desired accuracy via incorporating the physical knowledge through the training data. Specifically, the physical knowledge is incorporated into the data set and training rather than network architecture.

5.2. Open-loop & closed-loop simulation results:

After developing both RNN models, we run an open-loop simulation under random time varying inputs utilizing three models, the 2 RNN models and the first-principles model, and depict the results in Fig. 6. From the results, it can be noticed that both C-RNN and TL-RNN models provide good agreement with the FP model, yet the TL-RNN model is more accurate and its trajectory is closer to the trajectory of the true process model.

Table 2
Testing mean squared error for each reactor modeling by the two approaches.

Data size/model	CSTR 1		CSTR 2	
	C-RNN	TL-RNN	C-RNN	TL-RNN
10%	4.18×10^{-6}	5.22×10^{-7}	1.12×10^{-4}	5.25×10^{-5}
25%	3.40×10^{-6}	2.2×10^{-7}	5.74×10^{-5}	4.87×10^{-6}
50%	6.91×10^{-7}	3.71×10^{-7}	3.02×10^{-5}	1.69×10^{-5}
75%	4.31×10^{-7}	2.58×10^{-7}	2.13×10^{-5}	6.41×10^{-6}
99%	2.05×10^{-7}	9.58×10^{-8}	1.02×10^{-5}	6.27×10^{-6}

Subsequently to the open-loop simulation, we implement the TL-RNN model into the MPC scheme discussed in Section 4 and run closed-loop simulations. The Lyapunov function V is used in the form of $V = x^T Px$, where x is the state matrix and P is a positive definite matrix tuned by trail-and-error in order to generate the largest stability region under the feedback stabilizing controller. The LMPC objective function is expressed in the form of $L(x,u) = x^T \mathbf{Q}\, x + u^T \mathbf{R}\, u$, where $\mathbf{Q}$ and $\mathbf{R}$ are diagonal penalty matrices for the set-point error and control actions, respectively. The two matrices need to be properly tuned because they have a significant impact on the LMPC's performance. The MPC tuning recommendations given in Alhajeri and Soroush (2020) were followed.

Figs. 7–9 summarize the closed-loop simulation results based on varying data availability, specifically 50%, 75%, and 99%, respectively. In these closed-loop simulations, both RNN based LMPCs are successfully able to achieve the main objective to drive the system to the steady state and stabilize the system within a small neighborhood around the origin. Fig. 7 shows the process dynamics under the three LMPCs using only 50% of the available data set. The TL-RNN based LMPC is clearly performing better in terms of its responsiveness, as it responds more quickly and smoothly with notably less oscillation compared to the C-RNN. From Figs. 8 and 9, as the data availability increases to 75% and also up to 99% of total data set size, the two RNN-based MPCs' performance improved overall, which aligns with the theoretical expectations. However, the C-RNN-based MPC is still having noticeably oscillatory behavior and a slower response even when using larger training data size, in contrast to the TL-RNN, which shows better performance when using larger training data size.

During the closed-loop simulation session, the cost function $L(x,u)$ values were recorded for each model under the different data fraction scenarios. Fig. 10 shows how L changes with data size. It is noticeable that, with more data, the two RNN models tend to provide relatively smaller and closer L values. The importance of the TL-RNN model is clear in the smaller data fractions, for example at 25%, where the L value associated with TL-RNN model is 15% smaller than that associated with the C-RNN model. From these results, TL-RNN model has shown to be more cost efficient in all scenarios and especially under lower data availability.

6. Conclusion

In this work we investigated the utilization of transfer learning in the framework of RNN-based MPC. The model accuracy of the proposed TL-RNN and effect of data size were discussed. Open-loop and closed-loop simulations under a model predictive controller were carried out considering a process consisting of two CSTRs in series. The results demonstrated improvement in model accuracy by using transfer learning to develop the RNN model for a process with similar subsystems, especially when dealing with small amount of data. Moreover, closed-loop state trajectories under the TL-RNN-based MPC were in close agreement with the first principles-based MPC trajectories with overall smoother, faster, and less oscillatory responses than that under the C-RNN-based MPC. Hence, for large-scale systems comprising of similar subsystems, especially when the data set does not contain equal

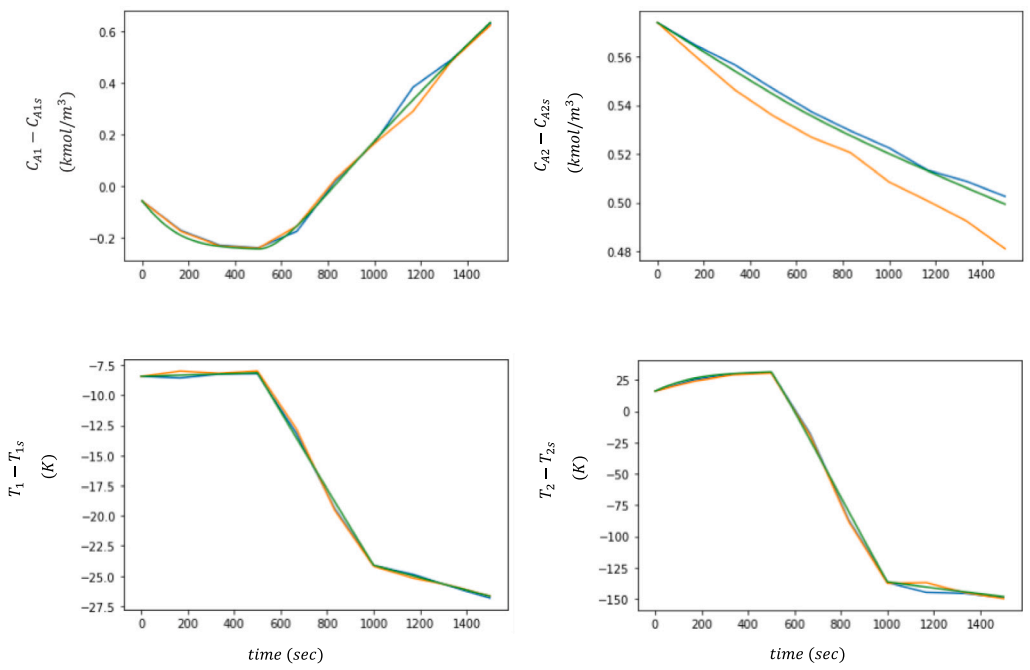

Fig. 6. Open-loop simulation results under random time-varying inputs, where the orange, blue, and green trajectories indicate the C-RNN, TL-RNN, and the FP models, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

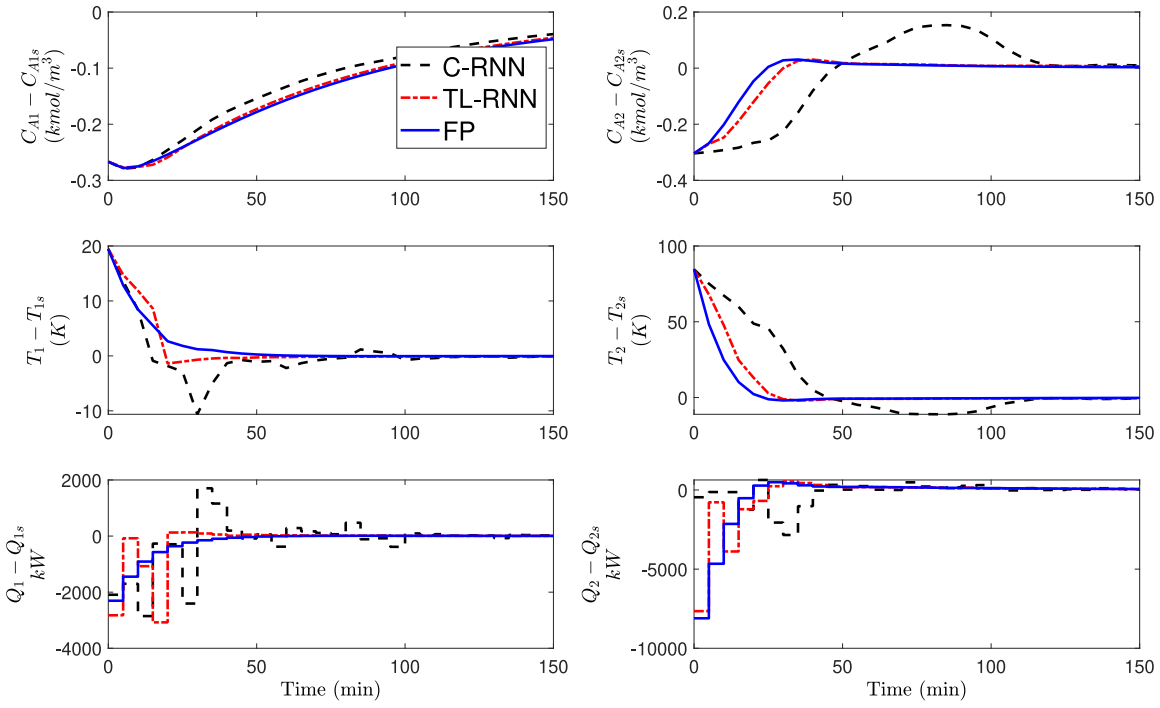

Fig. 7. Closed-loop simulation with 50% of the data size.

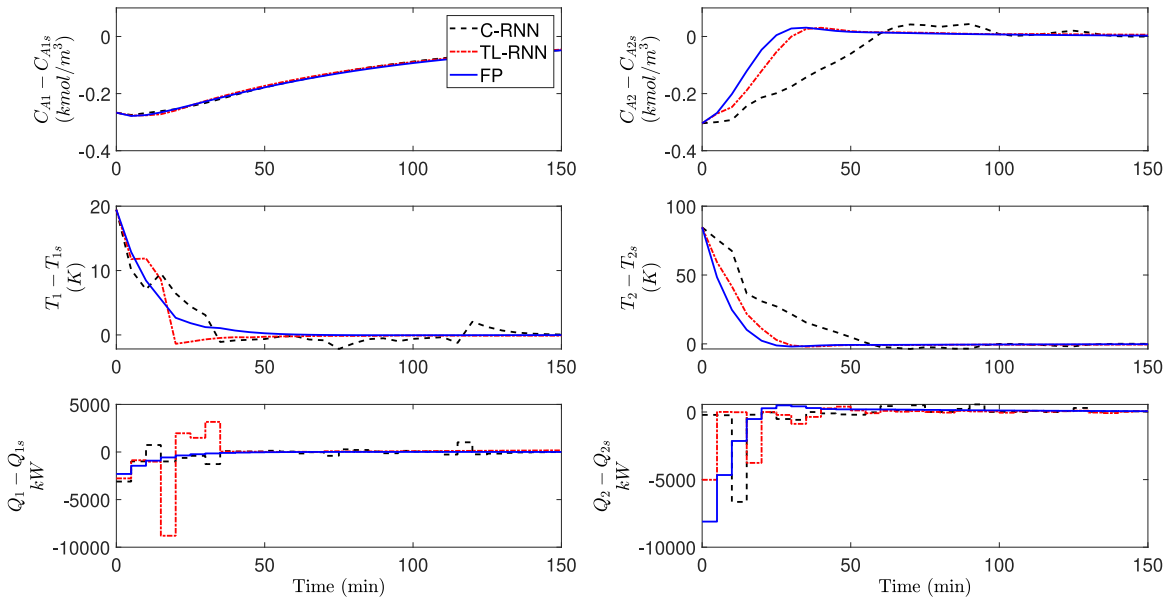

Fig. 8. Closed-loop simulation with 75% of the data size.

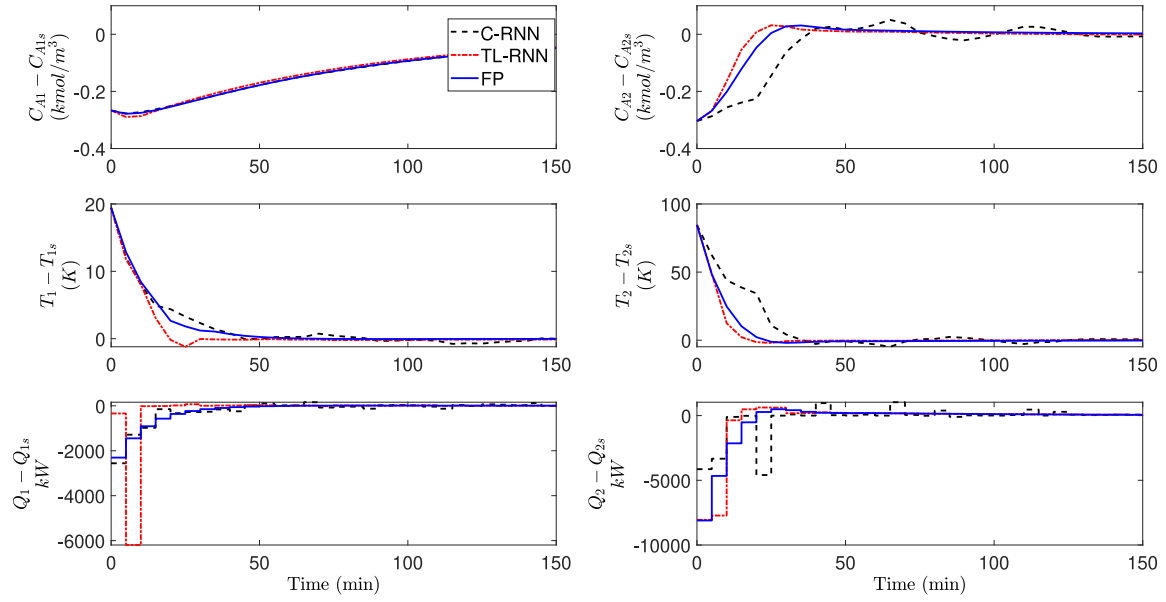

Fig. 9. Closed-loop simulation with 99% of the data size.

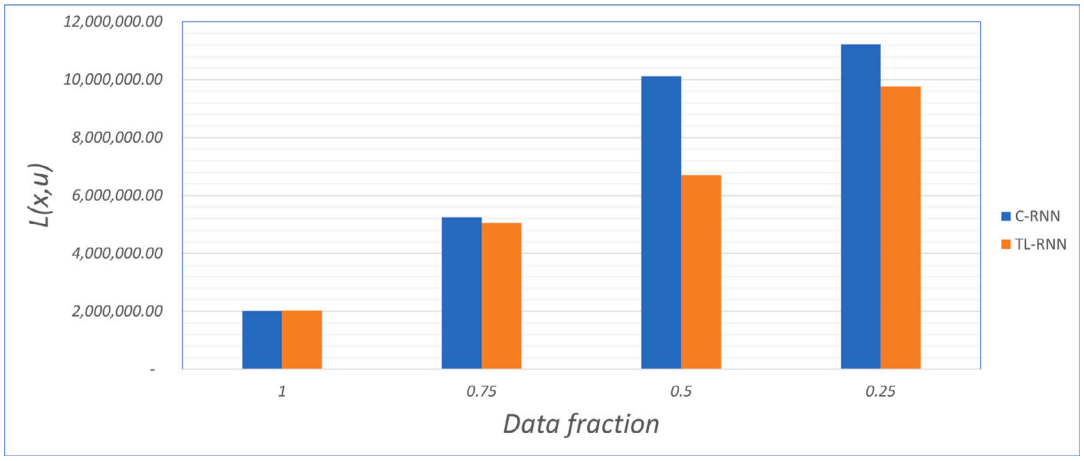

Fig. 10. Closed-loop cost function values based on data size and RNN modeling method.

proportions of data from the various subsystems, the proposed TL-RNN methodology based on *a priori* process knowledge was shown to be a promising alternative to conventional RNNs that would be highly restricted in its ability to use such a data set.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Financial support from the National Science Foundation and the Department of Energy is gratefully acknowledged. Mohammed S. Alhajeri would like to express his genuine appreciation to Kuwait University for its support.

References

Alhajeri, M.S., Alnajdi, A., Abdullah, F., Christofides, P.D., 2023. On generalization error of neural network models and its application to predictive control of nonlinear processes. Chem. Eng. Res. Des. 189, 664–679.
Alhajeri, M., Luo, J., Wu, Z., Albalawi, F., Christofides, P.D., 2022. Process structure-based recurrent neural network modeling for predictive control: A comparative study. Chem. Eng. Res. Des. 179, 77–89.
Alhajeri, M., Soroush, M., 2020. Tuning guidelines for model-predictive control. Ind. Eng. Chem. Res. 59 (10), 4177–4191.
Alhajeri, M.S., Wu, Z., Rincon, D., Albalawi, F., Christofides, P.D., 2021. Machine-learning-based state estimation and predictive control of nonlinear processes. Chem. Eng. Res. Des. 167, 268–280.
Amabilino, S., Pogány, P., Pickett, S.D., Green, D.V., 2020. Guidelines for recurrent neural network transfer learning-based molecular generation of focused libraries. J. Chem. Inf. Model. 60 (12), 5699–5713.
Baier, L., Jöhren, F., Seebacher, S., 2019. Challenges in the deployment and operation of machine learning in practice. In: Proceedings of the 27th European Conference on Information Systems. ECIS, Stockholm & Uppsala, Sweden.
Bozinovski, S., 2020. Reminder of the first paper on transfer learning in neural networks, 1976. Informatica 44 (3).
Briceno-Mena, L.A., Arges, C.G., Romagnoli, J.A., 2023. Machine learning-based surrogate models and transfer learning for derivative free optimization of HT-PEM fuel cells. Comput. Chem. Eng. 171, 108159.
Chen, T., Chen, H., 1995. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans. Neural Netw. 6 (4), 911–917.
Chuang, Y.-C., Chen, T., Yao, Y., Wong, D.S.H., 2018. Transfer learning for efficient meta-modeling of process simulations. Chem. Eng. Res. Des. 138, 546–553.
Fong, I.H., Li, T., Fong, S., Wong, R.K., Tallon-Ballesteros, A.J., 2020. Predicting concentration levels of air pollutants by transfer learning and recurrent neural network. Knowl.-Based Syst. 192, 105622.
Ghadami, A., Epureanu, B.I., 2022. Data-driven prediction in dynamical systems: recent developments. Phil. Trans. R. Soc. A 380 (2229), 20210213.
Golowich, N., Rakhlin, A., Shamir, O., 2018. Size-independent sample complexity of neural networks. In: Conference on Learning Theory. PMLR, pp. 297–299.
Gulli, A., Pal, S., 2017. Deep Learning with Keras. Packt Publishing Ltd.
Gupta, P., Malhotra, P., Narwariya, J., Vig, L., Shroff, G., 2020. Transfer learning for clinical time series analysis using deep neural networks. J. Healthc. Inform. Res. 4 (2), 112–137.
Laptev, N., Yu, J., Rajagopal, R., 2018. Reconstruction and regression loss for time-series transfer learning. In: Proceedings of the Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) and the 4th Workshop on the Mining and Learning from Time Series. MiLeTS, Vol. 20, London, UK.
Li, W., Gu, S., Zhang, X., Chen, T., 2020. Transfer learning for process fault diagnosis: Knowledge transfer from simulation to physical processes. Comput. Chem. Eng. 139, 106904.
Lin, Y., Sontag, E.D., 1991. A universal formula for stabilization with bounded controls. Systems Control Lett. 16 (6), 393–397.
Lindner, G., Shi, S., Vučetić, S., Mišković, S., 2022. Transfer learning for radioactive particle tracking. Chem. Eng. Sci. 248, 117190.
Narkhede, M.V., Bartakke, P.P., Sutaone, M.S., 2022. A review on weight initialization strategies for neural networks. Artif. Intell. Rev. 55 (1), 291–322.
Pan, Y., Wang, J., 2008. Two neural network approaches to model predictive control. In: 2008 American Control Conference. IEEE, Seattle, WA, USA, pp. 1685–1690.
Park, J., Sandberg, I.W., 1991. Universal approximation using radial-basis-function networks. Neural Comput. 3 (2), 246–257.
Ren, Y.M., Alhajeri, M.S., Luo, J., Chen, S., Abdullah, F., Wu, Z., Christofides, P.D., 2022. A tutorial review of neural network modeling approaches for model predictive control. Comput. Chem. Eng. 107956.
Rogers, A.W., Vega-Ramon, F., Yan, J., del Río-Chanona, E.A., Jing, K., Zhang, D., 2022. A transfer learning approach for predictive modeling of bioprocesses using small data. Biotechnol. Bioeng. 119 (2), 411–422.
Sugiyama, M., 2015. Introduction to Statistical Machine Learning. Morgan Kaufmann.
Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., Liu, C., 2018. A survey on deep transfer learning. In: Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27. Springer, pp. 270–279.
Wang, J., Zhang, W., Wu, H., Zhou, J., 2022. Improved bilayer convolution transfer learning neural network for industrial fault detection. Can. J. Chem. Eng. 100 (8), 1814–1825.
Wong, W.C., Chee, E., Li, J., Wang, X., 2018. Recurrent neural network-based model predictive control for continuous pharmaceutical manufacturing. Mathematics 6 (11), 242.
Wu, Z., Alnajdi, A., Gu, Q., Christofides, P.D., 2022. Statistical machine-learning-based predictive control of uncertain nonlinear processes. AIChE J. 68, e17642.
Wu, Z., Rincon, D., Gu, Q., Christofides, P.D., 2021. Statistical machine learning in model predictive control of nonlinear processes. Mathematics 9 (16), 1912.
Wu, Z., Tran, A., Rincon, D., Christofides, P.D., 2019. Machine learning-based predictive control of nonlinear processes. Part I: theory. AIChE J. 65 (11), e16729.
Wu, H., Zhao, J., 2020. Fault detection and diagnosis based on transfer learning for multimode chemical processes. Comput. Chem. Eng. 135, 106731.
Xiao, M., Hu, C., Wu, Z., 2023. Modeling and predictive control of nonlinear processes using transfer learning method. AIChE J. 69 (7), e18076.
Xiao, M., Wu, Z., 2023. Modeling and control of a chemical process network using physics-informed transfer learning. Ind. Eng. Chem. Res. 62 (42), 17216–17227.
Xu, J., Li, C., He, X., Huang, T., 2016. Recurrent neural network for solving model predictive control problem in application of four-tank benchmark. Neurocomputing 190, 172–178.
Xu, Y., Xie, L., Dai, W., Zhang, X., Chen, X., Qi, G.-J., Xiong, H., Tian, Q., 2021. Partially-connected neural architecture search for reduced computational redundancy. IEEE Trans. Pattern Anal. Mach. Intell. 43 (9), 2953–2970.
Yin, S., Kaynak, O., 2015. Big data for modern industry: Challenges and trends [point of view]. Proc. IEEE 103 (2), 143–146.
Yosinski, J., Clune, J., Bengio, Y., Lipson, H., 2014. How transferable are features in deep neural networks? Adv. Neural Inf. Process. Syst. 27.
Zarzycki, K., Ławryńczuk, M., 2021. LSTM and GRU neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors. Sensors 21 (16), 5625.
Zheng, Y., Wang, X., Wu, Z., 2022. Machine learning modeling and predictive control of the batch crystallization process. Ind. Eng. Chem. Res. 61, 5578–5592.
Zhou, Y., Jia, L., Zhang, Y., 2023. A transfer learning approach using improved copula subspace division for multi-mode fault detection. Can. J. Chem. Eng. 101 (12), 7015–7030.