A Deep Learning Architecture for Predictive Control

Steven Spielberg Pon Kumar * **Bhushan Gopaluni** ***
Philip Loewen ***

* *University of British Columbia, Vancouver, Canada (e-mail: spiel@chbe.ubc.ca).*
** *University of British Columbia, Vancouver, Canada (e-mail: bhushan.gopaluni@ubc.ca)*
*** *University of British Columbia, Vancouver, Canada (e-mail: loew@math.ubc.ca)*

Abstract: Model Predictive Control (MPC) is a popular control strategy that computes control action by solving an optimization objective. However, application of MPC can be computationally demanding and sometimes requires estimation of the hidden states of the system, which itself can be rather challenging. In this work, we propose a novel Deep Neural Network (NN) architecture by combining standard Long Short Term Memory (LSTM) architecture with that of NN to learn control policies from a model predictive controller. The proposed architecture, referred to as LSTM supported NN (LSTMSNN), simultaneously accounts for past and present behavior of the system to learn the complex control policies of MPC. The proposed neural network architecture is trained using an MPC and it learns and operates extremely fast a control policy that maps system output directly to control action without having to estimate the states. We evaluated our trained model on varying target outputs, various initial conditions and compared it with other trained models which only use NN or LSTM.

Keywords: Artificial intelligence, Process Control, Neural Nets, Parallel Computation.

1. INTRODUCTION

Controlling complex dynamical systems in the presence of uncertainty is rather challenging. These challenges arise due to non-linearities, disturbances, multivariate interactions and model uncertainties. A control method well-suited to handle these challenges is MPC. In MPC, the control actions are computed by solving an optimization objective that minimizes a cost function while accounting for system dynamics (using a prediction model) and satisfying input-output constraints. MPC is robust to modeling errors [1] and has the ability to use high-level optimization objectives [4]. However, solving the optimization objective in real time is computationally demanding and often takes lot of time for complex systems. Moreover, MPC sometimes requires the estimation of hidden system states which can be challenging in complex stochastic non-linear systems and in systems which are not observable. Also, standard MPC algorithms are not designed to automatically adapt the controller to any model plant mismatch. Several algorithms exist to speed up the MPC optimization through linearization [5] of the non-linear system and through approximation of the complex system to a simpler system [5][6][7]. However, these algorithms do not account for the complete non-linear dynamics of the system. In this article, we propose using the deep neural network function approximator to represent the complex system and the corresponding MPC control policy. Once the control policies of MPC are learned by the proposed deep neural network, no optimization or estimation is required. The deep neural network is computationally less demanding and runs extremely fast at the time of implementation.

We propose a novel neural network architecture using standard LSTM supported by NN models (LSTMSNN). The output of LSTMSNN is a weighted combination of outputs from LSTM and NN. This novel neural network architecture is developed in such a way that its output depends on past control actions, current system output and the target output. The LSTM part of LSTMSNN architecture uses past control actions as input to capture the temporal dependency between control actions. The NN part of LSTMSNN architecture uses current system output and target output as inputs to predict the control action. Our simulations show that the proposed deep neural network architecture is able to learn complex nonlinear control policies arising out of prediction and optimization steps of MPC.

We use a supervised learning approach to train LSTMSNN. First, we use MPC to generate optimal control actions and system output under a given target trajectory. Then, we use these data to train LSTMSNN and learn the MPC control policies. Once the control policies are learned, the LSTMSNN model can completely replace the MPC. This approach has the advantage of not having to solve any online optimization problems. In addition, we believe that LSTMNN model can be used to update the control policies without having to re-identifying the

model. In this article, we restrict ourselves to developing the LSTMNN model.

Our main contribution is the combined LSTM and NN architecture that can keep track of past control actions and present system information to take near optimal control actions. Since LSTMSNN uses deep neural networks accounting past and present information, we can train complex, high-dimensional states (system with large number of states) in stochastic non-linear systems using this approach. One of the unique features of our method is that the network is trained using an MPC. The trained LSTMSNN Model is computationally less expensive than MPC because it does not involve an optimization step and does not require estimation of hidden states. Since, we use a Graphical Processing Unit (GPU) to parallelize the computation, the prediction of optimal control actions is done rather rapidly.

2. RELATED WORK

Model predictive control (MPC) is an industrially successful algorithm for control of dynamic systems [9][10] but it suffers from high computational complexity when controlling large dimensional complex non-linear systems - (1) the optimization step while returning optimal control action could take very long time especially for higher dimensional non-linear systems (2) the estimation of states could be burdensome. To overcome this, various functional approximation techniques were developed for use with MPC [11],[12]. In addition, model free methods such as reinforcement learning techniques [14] were also developed. NN were used to approximate the MPC prediction step [6] and the optimization cost function. NN were also used to approximate the nonlinear system dynamics [7] and then MPC was performed on the neural network model. However, NN generated system predictions tend to be oscillatory. Other approaches include [8] using a two tiered recurrent neural network for solving optimization objective based on linear and quadratic programming formulations. However, these approaches involve estimation of hidden states at each instant. LSTMs are effective at capturing long term temporal patterns [2] and are used in a number of applications like speech recognition, smart text completion, time series prediction etc. Our approach differs from the fact that LSTMSNN is based on the past MPC control actions and the current system output. Moreover, our approach does not involve the burden of estimating the hidden states that characterize system dynamics.

3. PRELIMINARIES

3.1 Neural Network

Many practical problems can be formulated as requiring a computer to perform the mapping $f : X \mapsto Y$, where X is an input space and Y is an output space. The arbitrary function f transforms the inputs x to predicted outputs $\hat{y}$ using weights w that are learnt by solving an appropriate optimization problem.

Vanilla Neural Networks: The neural networks are constructed by repeating matrix multiplications and element-wise non-linearities. As an example, a 2-layer neural network would be implemented as $f(x) = W_2\sigma(W_1x)$ where W_1, W_2 are matrices and σ is an element-wise non-linearity (eg. sigmoid or tanh). Common settings for non-linearities are tanh, the sigmoid function, $1/1(1 + e^{-x})$, and the rectified linear unit (ReLu), max(0,x). A 3-layer network would have the form $f(x) = W_3\sigma(W_2\sigma(W_2\sigma(W_1x)))$. Also the last layer of the neural network normally does not contain the non-linearity.

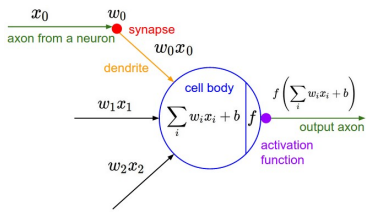


Fig. 1. A diagram of the biological inspiration behind a single neuron. Inputs x_i interact multiplicatively with the synapses w_i, the cell body accumulates the sum and then fires an output signal after the activation function

Back Propogation To learn parameters or weights (w), we construct a loss function using the difference between measured data and predictions, then evaluate the gradient of the loss function and use a gradient descent algorithm to minimize it. Through this process we find an approximation of the mapping f that is consistent with the data used for training. Backpropogation is the process by which we efficiently compute gradients of scalar valued functions with respect to their inputs. It is a recursive application of chain rule from calculus. The weights of the neural network are learnt using back propogation. The gradients of the structure shown in Fig. 1 are shown in Fig. 2.

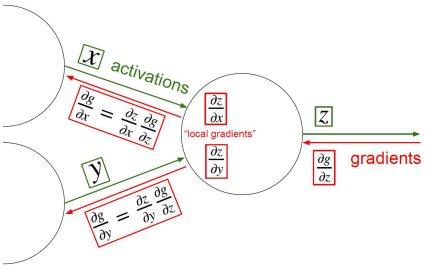


Fig. 2. An example of backpropogation along a computational graph. During forward pass x and y take on specific values and the vector z is computed using some fixed function $(eg. z = x \odot y)$, where $\odot$ is the element wise multiplication. Then Jacobian matrices $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ are computed. The backward pass proceeds in the reverse order, recursively applying the chain rule to find the influence of all inputs of the graph on the final output. The chain rule states that to backpropogate we should take the global gradient on z, $\frac{\partial g}{\partial z}$ and multiply it onto the local gradient for each input. For example, the global gradient for x will become $\frac{\partial g}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial g}{\partial z}$. The gradient is then recursively chained, in turn through the functions that produced the values of x and y until the inputs are reached.

Recurrent Neural Networks In many practical applications the input and output measurements contain se-

quences. For instance, a series of past inputs to the controller. A recurrent neural network (RNN) is a connectivity pattern that processes a sequence of vectors $x_1, x_2, ..., x_T$ using recurrence formula of the form $h_t = f_\theta(h_{t-1}, x_t)$, where f is a function that we describe in detail below and θ are parameters that are independent of time, allowing us to process sequences with an arbitrary number of vectors. The hidden vector h_t can be interpreted as a running summary of all vectors x_t until that time step and the recurrence formula updates the summary based on the previous vector. It is common to use $h_0 = \vec{0}$. The precise mathematical form of the recurrence $(h_{t-1}, x_t) \mapsto h_t$ varies from model to model and we describe these details below.

The Vanilla Recurrent Neural Network (RNN) uses a recurrence of the form,

$$h_t = tanh\big(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\big) \tag{1}$$

That is, the previous hidden vector and the current input are concatenated and transformed linearly by the parameter vector W. This is equivalent to writing $h_t = tanh\big(W_{xh}x_t + W_{hh}h_{t-1}\big)$, where the two matrices W_{xh}, W_{hh} concatenated horizontally are equivalent to the matrix W above. The tanh nonlinearity can be replaced with ReLU. If the input vectors x_t have dimension D and the hidden states dimension H, then W is a matrix of size $\big[H \times (D+H)\big]$. Interpreting the equation, the new hidden states at each time step are a non-linear function of linear combination of elements of x_t, h_{t-1}. In vanilla RNN, the gradients tend to either vanish or explode over long time periods.

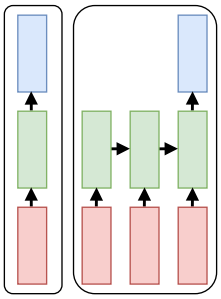


Fig. 3. An ordinary neural network (left) might take an input vector (red), transform it through some hidden layer (green), and produce an output vector (blue). In these diagrams boxes indicate vectors and arrows indicate functional dependencies. RNN allows us to process sequences of vectors, for example: 1) at the output, 2) at the input 3) either serially or in parallel. This is facilitated by a recurrent hidden layer (green) that manipulates a set of internal variables h_t based on previous hidden state h_{t-1} and the current input using a fixed recurrence formula $h_t = f_\theta(h_{t-1}, x_t)$, where θ are parameters we can learn

Long Short-Term Memory: The LSTM network is designed to address the limitations of the vanilla RNN. Its recurrence formula has a form that allows the inputs x_t and h_{t-1} to interact in a more computationally complex manner that includes multiplicative interactions. The LSTM recurrence uses additive interactions over time steps that more effectively propagate gradients backwards in time. In addition to a hidden state vector vector h_t, LSTMs also maintain a memory vector c_t. At each time step the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms. The precise form of the update is as follows:

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} sigm \\ sigm \\ sigm \\ tanh \end{bmatrix} W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \tag{2}$$

$$c_t = f \odot c_{t-1} + i \odot g \tag{3}$$

$$h_t = o \odot tanh(c_t) \tag{4}$$

Here, the sigmoid function sigm and tanh are applied element-wise, and if the input dimensionality is D and the hidden state has H units then the matrix W has dimension $[4H \times (D + H)]$. The three vectors $i, f, o \in \mathcal{R}^H$ are thought of as binary gates whose control state is revealed respectively in the hidden vector. The activations of these gates are based on sigmoid function and hence allowed to range smoothly between zero and one to keep the model differentiable. The vector $g \in \mathcal{R}^H$ ranges between -1 and 1 and is used to additively modify the memory contents. This additive interaction allows gradients on the memory cells c to flow backwards through time uninterrupted for long time periods.

3.2 MPC

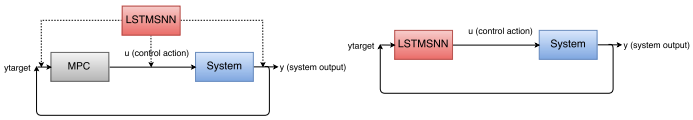
MPC is a multivariate control algorithm that uses an internal dynamic model of the process, history of past control moves to yield optimal control actions. The optimization objective function is given by,

$$\min_u \sum_{i=1}^{N} w_{y_i}(y_{target} - y_i)^2 + \sum_{i=1}^{N} w_{u_i}\Delta u_i^2 \tag{5}$$

where
y_i denotes i-th system output
y_{target} denotes required target value
u_i denotes i-th control action
w_{y_i} denotes weighting coefficient reflecting the relative importance of y_i
w_{u_i} denotes the weighting coefficient penalizing difference in u_i at successive instances

4. MODEL

In this work, we have proposed a different neural network architecture called LSTMSNN. We use LSTMSNN (weighted linear combination of LSTM and NN) to learn the complex behaviour of MPC. The block diagrams for training and testing phases are shown in Figure 4. In the training phase we use LSTMSNN model to learn the behaviour of MPC. Once the model has learnt, it can be used to predict optimal control action required to control the plant as given in Fig. 4 (right). The different neural network models used for training MPC behaviour are discussed below.



. Fig. 4. Left: Training setup; Right: Test setup

4.1 LSTM

The LSTM with sequence length of 5 is trained with data generated from MPC. The input to the model are past MPC control actions, plant output and target whereas the output is the control action to be taken at the next time step. Fig. 5 illustrates the LSTM model.

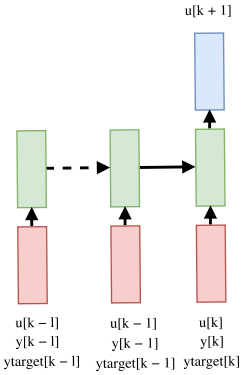


Fig. 5. Architecture of LSTM-only model

4.2 NN

The input to NN are the previous system output and target output. The output of NN is the control action at the next time step. The considered NN model has 3 layers. Fig. 6 illustrates the NN model.

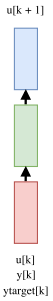


Fig. 6. Architecture of NN-only model

4.3 LSTMSNN

This is the new proposed architecture with an LSTM (sequence length of 5) and NN (3 layers) part. The motivation for this design is MPC control actions depend on both current system output and past input trajectories. Hence, the LSTMSNN output is a weighted combination of LSTM (which takes past input into account) and NN (takes current plant output and required setpoint) to learn the optimal control action given past input trajectories, current plant output and required setpoint. The best configuration of LSTMSNN resulted after tuning by simulation is shown in Fig. 7:

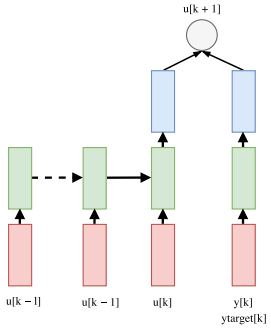


Fig. 7. Architecutre LSTMSNN model

5. EXPERIMENTAL SETUP

5.1 Physical system

The physical system we choose to study is a model from a paper machine. The target output is the desired moisture content of the paper sheet. The control action is the steam flow rate. The transfer function of the system is

$$G(z) = \frac{0.05z^{-4}}{1 - 0.6z^{-1}}. \tag{6}$$

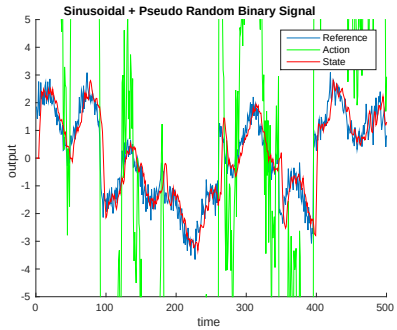
The time step used for simulation is 1 second.

5.2 Implementation

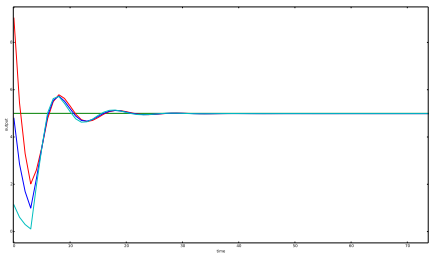
The neural networks are trained and tested using the deep learning framework Tensorflow [13]. In Tensorflow, the networks are constructed in the form of a graph and the gradients of the network required during training are computed using automatic differentiation. Training and testing were done on GPU (NVIDIA Geforce 960M) using CUDA. We ran the MPC and generated artificial training data in Matlab.

5.3 Data collection

We required three variables for training: the target, system output, and the control action. All the data used for training was artificially generated in Matlab by giving reference trajectories as discussed below.



. Fig. 8. Figure shows 500 time steps of a mixture of binary, sinusoidal and combined sequences used for training. Blue is the target output, red the system output and green the control action



. Fig. 9. LSTMSNN's system output under various initial conditions. Green line is the target output. Other colors denote different system outputs under varying initial conditions.

The data set used for training is a mixture of pseudorandom binary sequences. The target output randomly jumps to a value in the range $\{(-3,-0.2),(0.2,3)\}$. For each 1000 time steps, a sine function with period $(10,1000)$ time steps was chosen. Gaussian noise was added to the data with noise-to-signal ratio of 10. The motivation for using pseudorandom binary data to train is that it excites all frequencies of the system. Furthermore, physical systems are often operated with a step function reference. Sinusoidal data are also used during train to allow the model to learn periodic behavior. Hence, the reference data used are a mixture of random steps, sinusoids and random jumps data. The training data had 100,000 training points.

6. SIMULATION RESULTS

We conducted experiments to show effectiveness of LSTMSNN over other comparison models and to be a potential replacement to MPC after training.

Note that LSTMSNN uses a NN with many layers because we found that a simpler structure does not perform well during test time. We think this is because our training data sets are large, which limits a simpler structure from getting the most out of training data. However, decreasing the size of training data decreases LSTMSNN's performances during test time, especially under varying target output. So there is a trade off between complexity of architecture and size of training data.

Table 1: Performance comparison between methods

Model	Target Output	MSE	OE
LSTMSNN	2	0.06	0.01
NN-only	2	0.07	0.23
LSTM-only	2	5.56	Did not converge
LSTMSNN	5	0.078	0.01
NN-only	5	0.53	0.7
LSTM-only	5	62.42	Did not converge
LSTMSNN	10	0.01	0.01
NN-only	10	2.21	1.3
LSTM-only	10	167.78	Did not converge

Our set of experiments show the effectiveness of LSTMSNN by comparing with NN-only and LSTM-only. We use mean square error (MSE) and offset error (OE) to measure the performances. We ran the trained model for 1000 time steps for each model and the performance is shown in Table 1. We can see from the OE value that the LSTMSNN has learnt MPC behaviour and can track setpoint well compared to other models. The reason is, it has learnt the hidden states of the plant in the hidden unit and the optimization, prediction steps are learnt within the weights of the neural network using the past control action, current plant output and setpoint data.

Fig. 10 shows the system outputs by LSTMSNN and NN-only. The difference in OE between LSTMSNN and NN-only architectures also reflects the deficiency of using NN-only. It is very promising that LSTMSNN maintains an OE close to 0.01. LSTMSNN performs much better than other models in all other cases. A target output of 10 (Fig. 10. (d))is completely out of the range of our training data, but LSTMSNN still can track that setpoint. The reason why LSTMSNN was able to track setpoint is that it has learnt and generalized the MPC behaviour, optimization step, prediction step within the neural network and the hidden states are learnt as an abstract information within the hidden neural network units.

The next set of experiments show LSTMSNN's ability to handle varying targets. It can be seen from Fig. 11 that the LSTMSNN can still track setpoint despite the complex changes in setpoint. Fig. 9 shows LSTMSNN tracking setpoint under various different initial conditions. This demonstrates that LSTMSNN is a robust deep neural network architechture compared to other architectures.

7. CONCLUSION

We have developed a novel neural network architecture to learn the complex behaviour of MPC. This approach eliminates the burden of state estimation, optimization step and prediction step in MPC, since they are learnt as an abstract information in the hidden layers and hidden units of the network. Also, since the implementation uses GPU, computing optimal control action is relatively fast. Hence, the proposed approach can be used to learn the behaviour of MPC offline and the trained model can be deployed in production to control any complex high dimensional stochastic nonlinear system.

ACKNOWLEDGEMENTS

We would like to thank Smriti Shyammal of McMaster University and Tingke Shen, Zilun Peng of University of British Columbia for helpful discussions.

REFERENCES

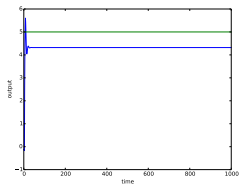
[1] D. Q. Mayne, M. M. Seron and S. V Rakovic, "Robust model predictive control of constrained linear systems with bounded disturbances", Automatica, vol. 41, no. 2, Feb. 2005

[2] Klaus, G., Rupesh, K.S., Jan, K., Bas, R.S. & Jurgen, S. (2015) LSTM: A Search Space Odyssey

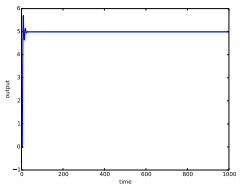
[3] Tieleman, T. and Hinton, G. Lecture 6.5RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012

[4] E. Todorov, T. Erez, and Y. Tassa, MuJoCo: A physics engine for model-based control, in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012

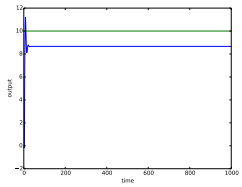
[5] Kuhne, Felipe, Walter Fetter Lages, and Joao Manoel Gomes da Silva Jr. "Model predictive control of a mobile robot using linearization." Proceedings of mechatronics and robotics. 2004



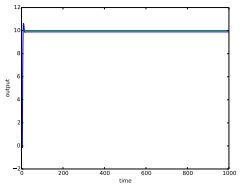
(a) NN-onlys system output for target output=5



(b) LSTMSNNs system output for target output=5

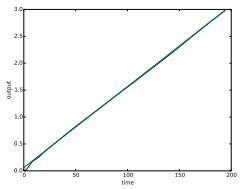


(c) NN-only's system output for target output=10

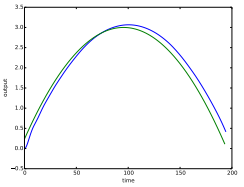


(d)LSTMSNN's system output for target output=10

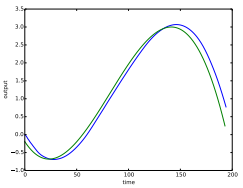
. Fig. 10. Performance comparison between LSTMSNN and NN-only for different target output.Green line is the target output. Blue line is system output



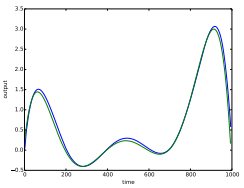
(a) Target output is linear



(b) Target output is quadratic



(c) Target output is cubic



(d) Target output is polynomial of degree 6

. Fig. 11. LSTMSNNs system output under varying target output. Green line is target output. Blue line is LSTMSNNs system output

[6] Paisan Kittisupakorn, Piyanuch Thitiyasook, M.A. Hussain, Wachira Daosud, "Neural Network based model predictive control for a steel picking process", Journal of Process Control

[7] Piche, Stephen, et al. "Neural network based model predictive control." Advances in Neural Information Processing Systems. 2000.

[8] Yunpeng Pan and Jun Wang, "Two Neural Network Approaches to Model Predictive Control" , American Control Conference, 2008

[9] K. Alexis, G. Nikolakopoulos, and A. Tzes, Model predictive quadrotor control: attitude, altitude and position experimental studies, Control Theory Applications, IET, vol. 6, no. 12, pp. 18121827, Aug 2012.

[10] M Wallace, SS Pon Kumar and P Mhaskar, Offset-Free Model Predictive Control (MPC) with Explicit Performance Specification, Industrial and Engineering Chemistry Research, 2016

[11] Zhong, Mingyuan, et al. "Value function approximation and model predictive control." Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on. IEEE, 2013.

[12] Akesson, Bernt M., and Hannu T. Toivonen. "A neural network model predictive controller." Journal of Process Control 16.9 (2006): 937-946.

[13] Abadi, Martn, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016)

[14] S.P.K Spielberg, Bhushan Gopaluni and Philip D. Loewen, Deep Reinforcement Learning Approaches for Process Control, Advanced Control of Industrial Processes, Taiwan, 2017