Control of Dynamic Systems Using LSTM supported Neural Network

Steven Spielberg P
Zilun C. Peng
Tingke K. Shen
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4
stevenspielberg8@gmail.com
zilunpeng@gmail.com
kshentingke@gmail.com

February 19, 2020

Abstract

Model Predictive Control (MPC) is an effective control strategy that computes control action by solving an optimization objective. However, application of MPC can be computationally demanding, and requires estimating the hidden states of the system, which can be challenging in complex system. In this work we have designed a different Neural Network (NN) architecture by combining Long Short Term Memory (LSTM) and NN, called LSTM supported NN (LSTMSNN) that accounts for past information and present behavior of the system to learn the complex policies of MPC. MPC is used to generate data during training time. This learned model acquires a policy that maps system output to control action without having to estimate the state and it is fast during test time. 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.

1 Introduction

The operation of dynamic system must tackle many challenges. The challenges arise due to non-linearity, disturbances and multivariate interactions. 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 (function of difference in target output and system output) while accounting for system dynamics (using a prediction model) and satisfying

output and control action constraints. MPC is robust to modeling errors [1] and has the ability to use high-level optimization objectives [5]. However, solving the optimization objective in real time is computationally demanding and often takes lot of time for complex systems. Moreover, MPC requires the estimation of hidden system states (hidden states characterize system dynamics) which can be challenging in complex non-linear dynamic system. Also it is not susceptible to actual-system model and mathematical model mismatch. Alternating methods to speed up the optimization process is linearizing [6] the non-linear system dynamics (or) approximating the complex system to simple system [7][8][9]. However, they do not account full dynamics of the system to perform MPC. Function approximations like neural networks were used to approximate complex system. However, they involve estimating the hidden states which are cumbersome. We propose to use an off-policy learning algorithm that can learn policies from MPC by only using system output (sensor data) without having to estimate the state. Also, it is computationally less demanding and fast at test time compared to MPC.

We propose LSTM supported NN model (LSTMSNN). The output of LSTMSNN is a weighted combination of outputs from LSTM and NN. We use this combination because the current control action depends on past control actions, current system output and target output. The LSTM part of LSTMSNN takes past control actions as input. Because there is temporal dependency between control actions, and we want to use LSTM to capture it. The NN part of LSTMSNN takes current system output and target output as input. Because we want to train NN to make a decision on control action by using current system and target output.

We use a supervised learning approach to train LSTMSNN. First, we use MPC to generate optimal control actions and system output under target output. Then, we use these data to train LSTMSNN. During test time, LSTMSNN makes a near optimal control action given previous control actions, system output and target output. In other words, it takes system output close to target output.

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), non-linear system using this approach. The training setup is one of the key benefits of our approach, since it allows training with optimal data and full state information but still produces a policy that uses only system output (sensor data) and not state estimation. Our trained LSTMSNN Model is computationally less expensive than MPC. It takes 0.0011 second on an average for our model to generate control action at test time. Since training our model can be paralleled, training can be done at a reasonable time.

The dynamic system in this project is the control of moisture content in a paper machine. We show that LSTMSNN can learn policies that are robust to a variety of perturbations and generalize well to various initial condition and varying target output.

2 Related Work

Model predictive control (MPC) is an effective and popular technique for control of dynamic systems [11][12][13] but it suffers from computational complexity in controlling complex systems. To overcome this, various functional approximations techniques were used on MPC [14],[15],[16]. Neural Networks was used to approximate MPC prediction step[7] and the optimization cost function. However, this again involved estimation of hidden system states which are tedious to estimate. NN were also used to approximate the nonlinear system dynamics [8] and then MPC was performed on the neural network model. However, NN generated system output tend to be oscillatory. Other approaches includes [9] using a two tiered recurrent neural networks for solving optimization objective based on linear and quadratic programming formulations. This again involved estimation of hidden states at each instant. In MPC guided policy search [3], MPC was used only at the time of searching optimal control policy and the policy were learned using Neural Networks. On the contrary, our approach is fully based on MPC control actions. NN was also used [10] to do the future state prediction steps with policies learned using Proportional-Integral-Derivative Controller (PID). However, the training of the model was carried out online. Hence, this model could not learn all the possible control actions and failed in specific test cases where it did not see the trained control actions during training. However, in our approach of training LSTMSNN, training was done offline using a MPC simulator and the predictions were found to be good even for the data (system outputs) it did not see during training. LSTMs are effective at capturing long term temporal patterns[2] and are used in number of applications like speech recognition, smart text completion, time series prediction etc. Our approach differs from the fact that LSTMSNN can have a track of past information about the MPC control actions and the current system output behavior. This leads to clever prediction of control output at the next instant with less oscillatory effect. Moreover, our approach does not involve the burden of estimating the hidden states that characterizes system dynamics.

3 Preliminaries

3.1 MPC

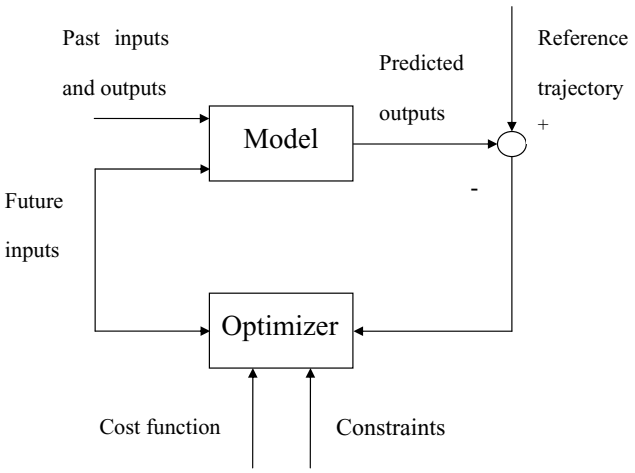


Figure 1: Basic structure of MPC

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,

$$\min_{u} \sum_{i=1}^{N} w_{y_i}(ytarget - y_i)^2 + \sum_{i=1}^{N} w_{u_i} \Delta u_i^2 \tag{1}$$

Where,
y_i denotes i-th system output
$ytarget$ denotes required target value
u_i denotes ith 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

3.2 LSTM

Figure 2 shows the flow of data through a LSTM unit or cell. Here x_t, h_t, C_t are the input, hidden state, and cell state respectively. σ represents the sigmoid function $\frac{1}{1+e^{-x}}$ while tanh is the hyperbolic tangent; they are the point-wise non-linear activation functions. Equations 8 through 7 are the corresponding equations for figure 2. W and b are weight matrices to be learned and $\odot$ is the point-wise multiplication operator.

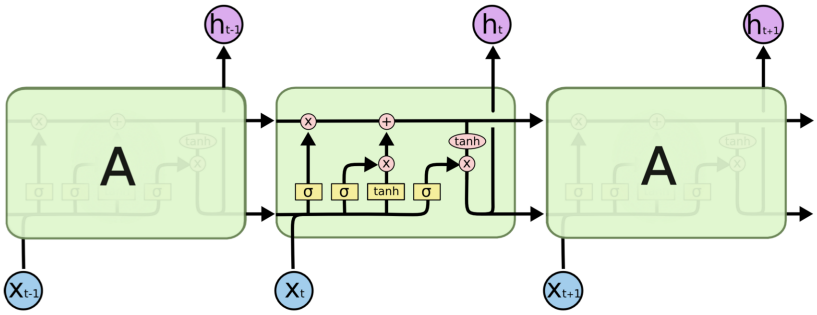


Figure 2: This figure shows the schematic of a LSTM unit used in the recurrent neural network. Image courtesy of
`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad forget\ gate \tag{2}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad input\ gate \tag{3}$$

$$\widetilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \quad input\ state \tag{4}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \quad cell\ state \tag{5}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad output\ gate \tag{6}$$

$$h_t = o_t \odot tanh(C_t) \quad hidden\ state \tag{7}$$

4 Model

In this work, we used different neural network architectures to construct LSTM-SNN. First the state-of-art LSTM and deep Neural Network (NN) were used. We then introduced a new architecture which is the weighted linear combination of LSTM and NN, LSTMSNN, to learn the complex behaviour of MPC. The block diagrams for training and testing phase are Figure 3 and Figure 4:

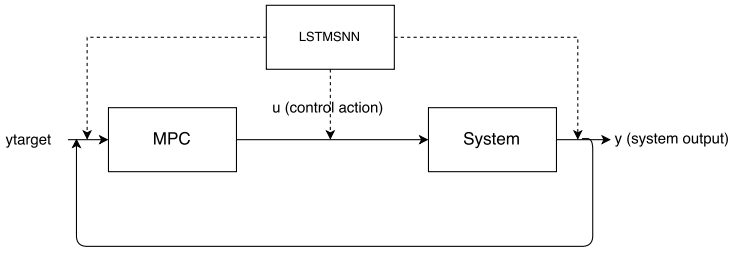


Figure 3: Training Phase

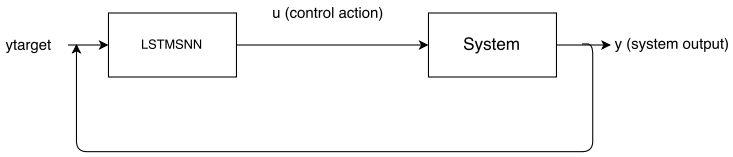


Figure 4: Test Phase

4.1 LSTM

The LSTM is trained with data generated from MPC. The input to the model are past MPC control actions and the output is the control action to be taken at the next time step. Number of past MPC control actions to be considered is the number of cells in the first LSTM layer. Figure 5 illustrates the LSTM part of LSTMSNN.

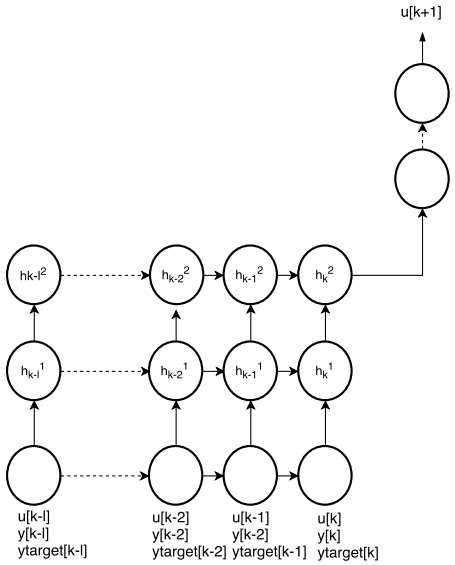


Figure 5: Architecture of LSTM-only model

4.2 NN

The input to NN are the previous system output and target output. Note that NN only takes one previous system output and one current target output. Output of NN is the control action at the next time step. Figure 6 illustrates the structure of NN part of LSTMSNN.

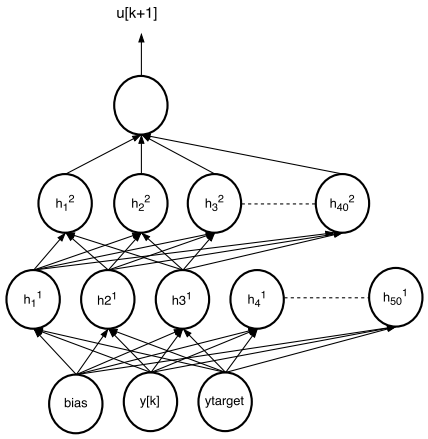


Figure 6: Architecture of NN-only model

4.3 LSTMSNN

LSTM takes past control actions into account. NN takes past system output and target output into account. MPC control actions depend on both current system output and past input trajectories. So we take a weighted combination of the two outputs from LSTM and NN and this will be the control action in the next time step. The best configuration of LSTMSNN resulted after tuning is shown in Figure 7:

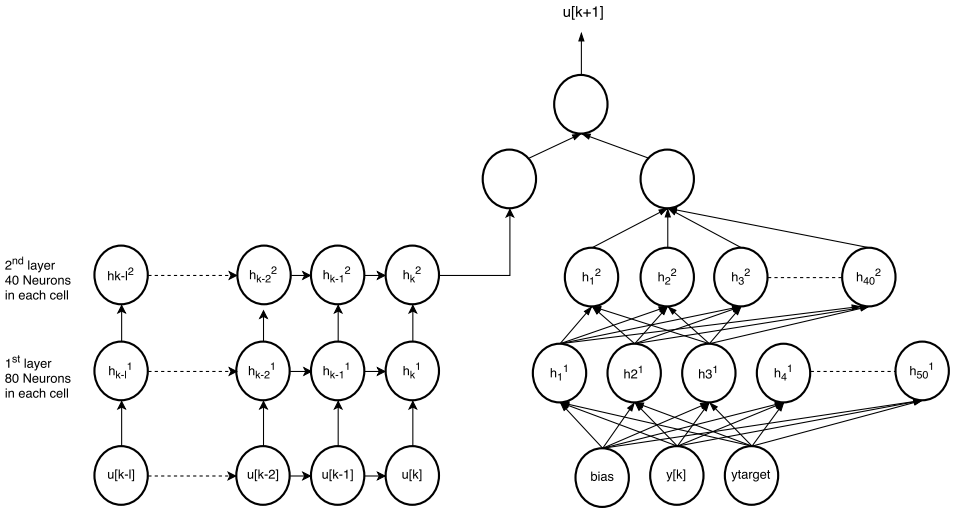


Figure 7: Best configuration of LSTMSNN model

5 Experimental setup

5.1 Physical system

The physical system we chose to study is the manufacture of paper in a paper machine. The target output is the desired moisture content of the paper sheet. The control action is the steam flow rate. The system output is the current moisture content. The transfer function of the system is

$$G(z) = \frac{0.05z^{-4}}{1 - 0.6z^{-1}} \tag{8}$$

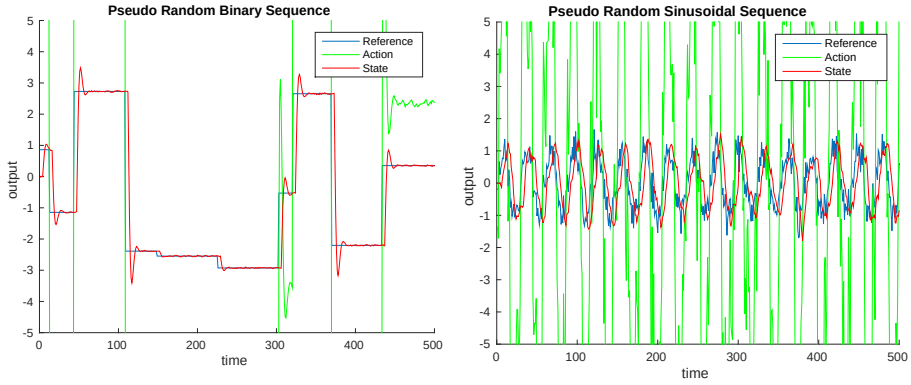
The time step used for simulation is 1 second.

5.2 Implementation

The neural networks were trained and tested using the deep learning library for Theano. 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 output, system output, and control action. All the data used for training was artificially generated in Matlab. We chose the target outputs and then used MPC to find the optimal action actions and resulting system outputs. There are three different sets of data used in this study. Each set of data had 1000,000 training points.



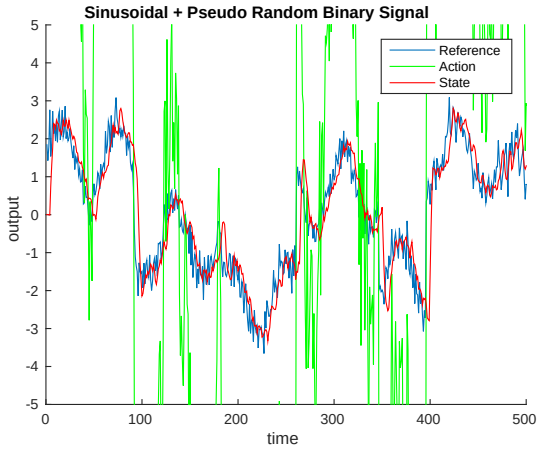


Figure 9: This figure shows 500 time steps snippets of the three types of training files used. Blue is the target output, red the system output and green the control action. The binary, sinusoidal, and combined sequences are shown on the top left, top right, and bottom respectively

The first set is a pseudorandom binary sequence (RandomJump). The target output randomly jumps to a value in the range $\{(-3,-0.2),(0.2,3)\}$. The target output then remains constant for 10 to 100 time steps before jumping again. The second set is a pseudorandom sinusoidal sequence. 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 last set of data is the combination of the first two sets, a pseudorandom binary-sinusoidal sequence (SineAndJump). At each time step, the target outputs from the two data sets were added. The motivation for using pseudorandom binary data to train is that a step function is the building block for complex functions. Any function can be approximated by a series of binary steps. Furthermore, physical systems are often operated with a step function reference. The motivation to train on sinusoidal data is that our model should learn periodic behavior.

6 Simulation Results

We conducted experiments to show the effects of data set on LSTMSNN's performance, effectiveness of LSTMSNN over other comparison models and the robustness of LSTMSNN under various initial conditions and varying target outputs.

We trained LSTMSNN by using RMSprop [4]. We also used the two parts of LSTMSNN, LSTM and NN, as comparison models in our experiments. They are denoted as LSTM-only and NN-only. Details of model architectures are in Table 1. 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: Details of comparison models

Model	Optimizer	Number of layers	Sequence Length
LSTMSNN	RMSprop	2 layers of LSTM and 4 layers of NN	5
LSTM-only	RMSprop	2	5
NN-only	Stochastic Gradient Descent	3	N/A

Our first set of experiments intend to show the effect of training data to LSTMSNN's performances. We test the performances of LSTMSNN by specifying an initial condition. LSTMSNN will generate a control action in each time step and get a system output. Previous control action, system output and a constant target output will be the input to LSTMSNN in the next time step. We run this process for 1000 time steps. We use mean square error (MSE) and offset error (OE) to measure the performances. MSE is the average of squared errors between model's system output and target output during the process. OE is the difference between model's system output and target output after system output converges. We will compare the performances of LSTMSNN by training on RandomJump and SineAndJump, as shown in Table2.

Table 2: Performance comparison by testing different data set

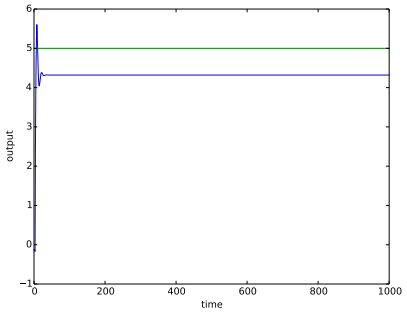
Data Set	Target Output	MSE	OE
RandomJump	2	0.02	0.01
SineAndJump	2	0.06	0.01
RandomJump	5	0.176	0.18
SineAndJump	5	0.078	0.01
RandomJump	10	1.28	1.8
SineAndJump	10	0.01	0.01

Although training on RandomJump outperforms SineAndJump when target output is 2, LSTMSNN is able to maintain a low OE as the target output increases. And it is more important for a controller to maintain a smaller OE during the process. We think the size and diversity of data set causes the performance difference. SineAndJump allows LSTMSNN to learn the optimal control actions under more system and target outputs, whereas RandomJump cannot.

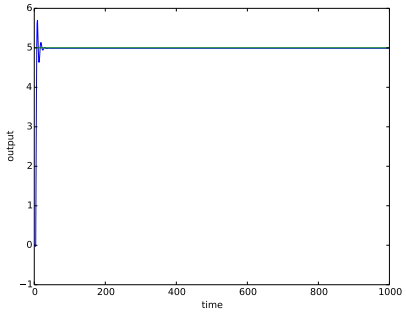
Table 3: Performance comparison between methods

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

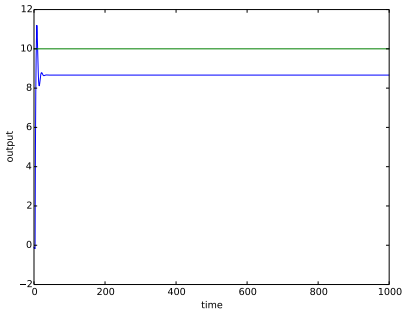
Our second set of experiments show the effectiveness of LSTMSNN by comparing with NN-only and LSTM-only. In experiment, each model starts with the same initial condition and receives a system output after making a decision on control action in each time step. The current system output, fixed target output and current control action are the input to LSTMSNN and LSTM-only in next time step. The current system output, fixed target output are the input to NN-only in the next time step, because NN outputs the next control action without using past control action according to Section 4.2. We run this process for 1000 time steps for each model. We use SineAndJump to train each model. Results are shown in Table 3



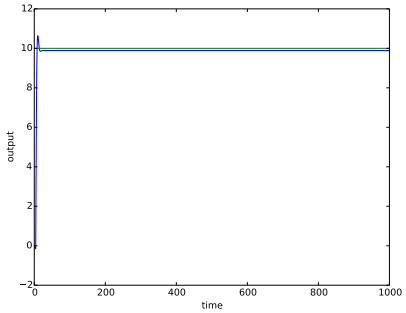
(a) NN-only's system output for target output=5



(b) LSTMSNN's 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

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

Figure 10 shows the system outputs by LSTMSNN and NN-only. LSTMSNN does not outperform NN-only much in MSE when target output is 2. But NN-only has a constant gap between its system output and target output, whereas LSTMSNN does not. The difference in OE between LSTMSNN and NN-only also reflects the flaw of NN-only. It is very promising that LSTMSNN maintains an OE close to 0.01 as time step increases. LSTMSNN performs much better than other models in all other cases. A target output of 10 is completely out of the range in our training data, but LSTMSNN still can output system outputs with low MSE and OE. The reason behind that is LSTMSNN takes control actions calculating the trend of the past control actions using LSTM as well as present system output with NN. The weighted combination of LSTM and NN in LSTMSNN has learnt the time series and mapping (map from output to control actions) trend so well that even though the data set range is -3 to -3 it is able to make the system output reach target output of 10 with less oscillatory effect.

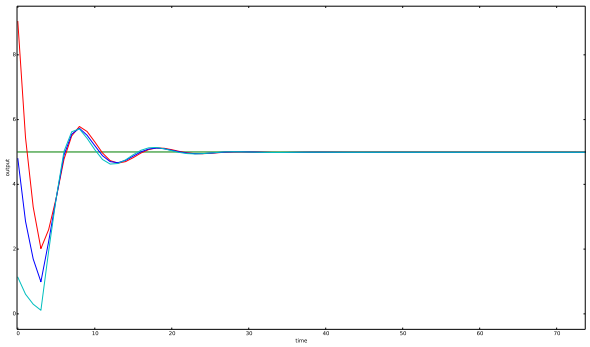
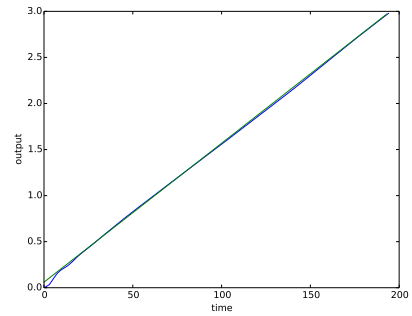
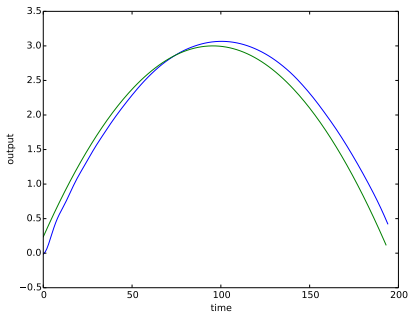


Figure 11: LSTMSNN's system output under various initial condition. Green line is the target output. Other color denotes a different system output under different varying initial condition.

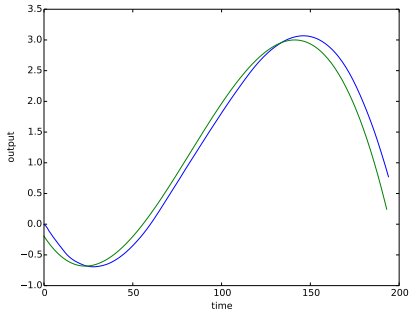
The third set of experiments show LSTMSNN's ability to handle various initial conditions and varying target outputs. We use SineAndJump to train LSTMSNN. We give LSTMNN a set of random initial conditions and set a fixed target output. From Figure 11, we can see that LSTMSNN is able to adjust and produce system output close to target output after some time steps. We also give LSTMSNN varying target outputs, and we can observe that LSTMSNN can accurately follow the varying target output from Figure 12. This demonstrates that LSTMSNN is a robust model. System outputs of LSTMS are very good when varying target output is smooth. We did find that LSTMSNN's system outputs are not as good when target output has some sharp corners. In those cases, LSTMSNN has poor performances at those sharp corners but it starts to perform well when target output becomes smooth again. We think a sudden change in target output (sharp corner) causes LSTMSNN some time steps to adjust its output.



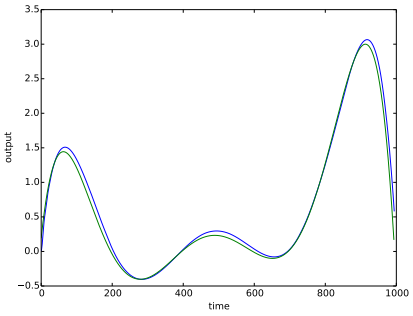
(a) Target output is linear



(b) Target output is quadratic



(c) Target output is cubic



(d) Target output is polynomial of degree is 6

Figure 12: LSTMSNN's system output under varying target output. Green line is target output. Blue line is LSTMSNN's system output.

7 Conclusion and Future Work

In this project, we combine LSTM and NN to build a robust model LSTMSNN. LSTMSNN learns from training data generated by MPC and is able to produce good system output under new target output. We conduct experiments to show the effectiveness of LSTMSNN over other methods and the robustness of LSTMSNN under varying conditions.

This study was limited to training and testing on artificially generated data. Further studies should examine the use of LSTM networks in controlling real systems and training on real data either offline or online. It may also be of interest to investigate the effectiveness of LSTM networks in controlling multiple-input-multiple-output (MIMO) systems (which we did not have time to do ourselves). In such systems, artificially generating data becomes difficult because the number of possible pseudorandom parameters (e.g. the sinusoidal sequence's frequencies) grows exponentially with output dimension. Hence, real data is

required. Nevertheless, the results of this study suggest that artificial data may even augment performance in MIMO systems. Finally, we were unable to investigate multiple-agent systems which we sought to study in our project proposal. In such a dynamic system, the agent would try to maintain some system state while the actions of an adversary changes in time. This is an extension of a MIMO system and could be studied in the future.

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] Zhang, T., Kahn, G., Levine, S. & Abbeel, P. (2015) Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search

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

[5] 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

[6] 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

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

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

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

[10] Draeger, Andreas, Sebastian Engell, and Horst Ranke. "Model predictive control using neural networks." Control Systems, IEEE 15.5 (1995): 61-66.

[11] 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. 1812–1827, Aug 2012.

[12] Wallace, M.; Das, B.; Mhaskar, P.; House, J.; Salsbury, T. Offsetfree model predictive control of a vapor compression cycle. J. Process Control 2012

[13] Flores-Tlacuahuac, A.; Moreno, S. T.; Biegler, L. T. Global optimization of highly nonlinear dynamic systems. Ind. Eng. Chem. Res. 2008

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

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

[16] Akpan, Vincent A., and George D. Hassapis. "Nonlinear model identification and adaptive model predictive control using neural networks." ISA transactions 50.2 (2011): 177-194.