A Deep Learning Architecture for Predictive Control

Steven Spielberg Pon Kumar and Bhushan Gopaluni

University of British Columbia
Outline

- Introduction
- Related Works
- Background
- System Setup
- Model Architectures
- Simulation Results
- Observations
- Conclusion
Introduction
Model Predictive Control - Advantages

- Look ahead in future to take optimal control actions
- Desired objective can be fed in optimization objective
- Constraint can be imposed in MPC via constraints in optimization objective
- Can handle non-linearities, disturbances, multivariate interactions and model uncertainties
- More degrees of freedom to control the plant optimally compared to P, PI and PID controllers.
MPC - Drawbacks

• Solving optimization problem ‘online’ for complex non-linear systems takes time
• Computationally intensive in production stage
• Estimation of hidden states (difficult for complex stochastic non-linear system and for non-observable systems)
MPC – Existing Approaches to Tackle Drawbacks

**MPC - Drawbacks**

- Solving optimization problem ‘online’ for complex non-linear systems takes time
- Computationally intensive in production stage
- Estimation of hidden states (difficult for complex stochastic non-linear system and for non-observable systems)

**Current Approaches to tackle Drawbacks**

- Linearize a non-linearize system
- Approximate complex system with simple system
MPC – Existing Approaches to Tackle Drawbacks

**MPC - Drawbacks**

- Solving optimization problem ‘online’ for complex non-linear systems takes time
- Computationally intensive in production stage
- Estimation of hidden states (difficult for complex stochastic non-linear system and for non-observable systems)

**Current Approaches to tackle Drawbacks**

- Linearize a non-linear system
- Approximate complex system with simple system

Does not account the full dynamics of the system leading to decrease in performance
Motivation – Learning Control Problems with deep neural networks

Image Classification + Self Driving Cars

Human Level Control in Games

complex Go game \(1.74 \times 10^{172}\) states

Paint like an Artist

Machine Translation

Learning from humans via VR

Source: google images
Primary reasons for success of deep learning

- Increase in compute power (Moore’s law)
- Use of Graphical Processing Unit for parallel computation
Can we frame this specific control problem as a learning problem?

- Artificial Intelligence meets Process Control

- Next Generation exascale Controllers
Our Approach

- Learn complex control behavior of MPC
- Learn function of the mapping from plant output to optimal control input

Challenges

- Data collected with respect to time steps are not i.i.d (independently identically distributed)
- Control actions are dependent of current output and past control actions

Novel Architecture

- We propose a novel neural network architecture to learn the complex control behavior

Advantages

- Quick in predicting optimal control actions given plant output
- Does not involve solving optimization objective
- Does not involve estimating the hidden states
- All steps are learnt within hidden units of neural networks
Motivation - Summary

**Optimization based control**

- Computationally demanding (optimization objective)
- Estimating hidden states
- Solving optimization problem ‘online’ for complex non-linear systems takes time

**Current Approach**

* Linearize a non-linear system dynamics
* Approximate complex system with simple system
* Does not account full dynamics of the system

**Drawback**

1. Quicker at Test time
2. Does not involve solving optimization problem

* Learn the complex policies of MPC
* A new novel architecture combining recurrent network and feed forward nets
Related Works
Related Works

- NN was used to approximate prediction step [1]
- NN was used to approximate non-linear dynamics [2]
- Two tier RNN for solving optimization objective based on linear and quadratic formulations [3]
- NN was used to do the future state prediction steps with policies learnt using Proportional-Integral-Derivative Controller (PID) [4]
Background
**Linear Regression**

\[
\hat{y}_i = \theta_1 + \theta_2 \times x_i
\]

- \(\hat{y}_i\) is the predicted output, \(\theta_1\) and \(\theta_2\) are the parameters (weights) of the linear model.

### Goal
- Learn the parameters/weights \(\theta_1\) and \(\theta_2\) using the data we have

### Prediction
- If \(\theta_1\) and \(\theta_2\) are known, Output can be predicted.

Activation function (linear for linear regression)

- \(u_1\) - output of neurons before activation
- \(o_1\) - output of neurons after activation

Since activation function is linear,

\[
\hat{y}_i = o_1 = u_1 = \theta_1 + \theta_2 \times x_i
\]
Learning Algorithm

- The loss is given by,

\[
    Loss = \sum_{i=1}^{n} (y_i^{true} - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i^{true} - \theta_1 + \theta_2 \times x_i)^2
\]

- \(y_i^{true}\) is the data we have in our hand

- \(\hat{y}_i\) is the data that our model predicts

Find the values of \(\theta_1\) and \(\theta_2\) where loss is minimum,

\[
    \min_{\theta_1, \theta_2} \sum_{i=1}^{n} (y_i^{true} - \theta_1 + \theta_2 \times x_i)^2
\]

Goal

- We want our predictions to be closer to actual data

Summary

- If \(\theta_1\) and \(\theta_2\) are known, output can be predicted
Drawback - Linear Regression

\[ \hat{y}_i = \theta_1 + \theta_2 \times x_i \]

- \( \hat{y}_i \) is the predicted output, \( \theta_1 \) and \( \theta_2 \) are the parameters (weights) of the linear model.

Linear Regression - Output is not bounded to [0,1]
Logistic Regression

\[ \hat{y}_i = \frac{1}{1 + e^{-(\theta_1 + \theta_2 \times x_i)}} \]

- \( \hat{y}_i \) is the predicted output, \( \theta_1 \) and \( \theta_2 \) are the parameters (weights) of the linear model.

Sigmoid function, \( f(x) = \frac{1}{1 + e^{-x}} \)

Linear Regression - Output is bounded to [0,1]
Neural Networks – For Regression

\[ u_{11} = \theta_1 \times 1 + \theta_2 \times x_i \]

\[ u_{12} = \theta_2 \times 1 + \theta_3 \times x_i \]

\[ o_{11} = \frac{1}{1 + e^{-u_{11}}} \]

\[ o_{12} = \frac{1}{1 + e^{-u_{12}}} \]

\[ u_{21} = \theta_5 \times 1 + \theta_6 \times o_{11} + \theta_7 \times o_{12} \]

\[ \hat{y}_i = o_{21} = u_{21} \]
Neural Network Compact Representation
Input vector layer

Hidden Layer

Output vector Layer

\[ u[k + 1] \]

\[ u[k] \]

\[ y[k] \]

\[ y_{target[k]} \]
Recurrent Neural Network (RNN)
Drawback of Recurrent Neural Networks

- Because of many layers in time steps,
  
  (i) if gradient is less than zero, while backpropogating to first layer the gradients becomes zero (‘vanishing gradient’)

  (ii) If gradient is gradient is greater than 1, the gradient explodes (‘exploding gradient’)

- To overcome that ‘Long Short Term Memory’ was proposed. In addition to hidden state vector it uses memory vector to overcome the ‘vanishing’ and ‘exploding’ gradient problems.

- In terms of architecture, LSTM is similar to RNN but with extra model complexity to handle ‘vanishing’ and ‘exploding’ gradient problems.

Long Short Term Memory (LSTM)

\[
\begin{align*}
  f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{forget gate} \\
  i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{input gate} \\
  C_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \quad \text{input state} \\
  \tilde{C}_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{cell state} \\
  o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{output gate} \\
  h_t &= o_t \odot \tanh(C_t) \quad \text{hidden state}
\end{align*}
\]
Model Predictive Control

- Objective function: \( \min_u \sum_{i=1}^{N} w_y (y_{\text{target}} - y_i)^2 + \sum_{i=1}^{N} w_{u_i} \Delta u_i^2 \)
- \( y \) – outputs, \( w \) – weighing coefficient, \( u \) – inputs, \( N \) – Prediction horizon
System Setup
System Setup

Training

\[
\text{ytarget} \rightarrow \text{MPC} \rightarrow \text{System} \rightarrow y \text{ (system output)}
\]

\[
\text{LSTMSNN} \rightarrow u \text{ (control action)}^{\text{\rightarrow}} \text{LSTMSNN}
\]

Testing

\[
\text{ytarget} \rightarrow \text{LSTMSNN} \rightarrow \text{System} \rightarrow y \text{ (system output)}
\]

\[
\text{LSTMSNN} \rightarrow u \text{ (control action)}^{\text{\rightarrow}} \text{LSTMSNN}
\]
Model Architectures
Model 1 – Long Short Term Memory
Model 2 – Neural Networks
Model 3 – New Architecture - LSTMSNN

u[k + 1]

u[k]

u[k - 1]

y[k]
ytarget[k]
Simulation Results
Experimental Setup – Physical System

- Control of manufacture of paper in a paper machine
- Target output, $y_{set}$, is the desired moisture content of the paper sheet
- Control action, $u$, is the steam flow rate
- System output, $y$, is the current moisture content
- Model,

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

• Basic Python numpy implementation with cudapy for GPU computation

• Trained and tested on NVIDIA Geforce 960M 4GB

• MPC simulation data were generated in Matlab

• Extended implementation with tensorflow
Data Collection

- Training Samples - 100,000 points
- Random jump – \{(-3, -0.2), (0.2,3)\}
- Sine function with period (10, 1000)
- Gaussian noise with noise to signal ratio of 10
- Sine and Random jump combined fed as a reference trajectory to MPC
Training

- Compared LSTM, NN, LSTMSNN
- Details of training

<table>
<thead>
<tr>
<th>Model</th>
<th>Optimizer</th>
<th>Number of layers</th>
<th>Sequence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTMSNN</td>
<td>RMSprop</td>
<td>2 layers of LSTM and 4 layers of NN</td>
<td>5</td>
</tr>
<tr>
<td>LSTM-only</td>
<td>RMSprop</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>NN-only</td>
<td>Stochastic Gradient Descent</td>
<td>3</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Simulation Results

(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
Simulation Results
Simulation Results

(a) Target output is linear

(b) Target output is quadratic

(c) Target output is cubic

(d) Target output is polynomial of degree is 6
# Performance Comparison

<table>
<thead>
<tr>
<th>Model</th>
<th>Target Output</th>
<th>MSE</th>
<th>OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTMSNN</td>
<td>2</td>
<td>0.06</td>
<td>0.01</td>
</tr>
<tr>
<td>NN-only</td>
<td>2</td>
<td>0.07</td>
<td>0.23</td>
</tr>
<tr>
<td>LSTM-only</td>
<td>2</td>
<td>5.56</td>
<td>Did not converge</td>
</tr>
<tr>
<td>LSTMSNN</td>
<td>5</td>
<td>0.078</td>
<td>0.01</td>
</tr>
<tr>
<td>NN-only</td>
<td>5</td>
<td>0.53</td>
<td>0.7</td>
</tr>
<tr>
<td>LSTM-only</td>
<td>5</td>
<td>62.42</td>
<td>Did not converge</td>
</tr>
<tr>
<td>LSTMSNN</td>
<td>10</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>NN-only</td>
<td>10</td>
<td>2.21</td>
<td>1.3</td>
</tr>
<tr>
<td>LSTM-only</td>
<td>10</td>
<td>167.78</td>
<td>Did not converge</td>
</tr>
</tbody>
</table>
Observations
Observations

- Predicting hidden states step is learnt as an abstract information in the hidden units of neural network.
- Optimization step is learnt as an abstract information in the hidden units of neural network.
- MPC prediction step is learnt within the hidden units and now prediction model is learnt directly with plant output without the need of hidden states.
- Computing with GPU makes it relatively fast.
Conclusion
Conclusion

- Combined deep learning with process control
- Different Neural Network architecture ‘LSTMSNN’
- Eliminates burden of state estimation, optimization step, prediction step are learn in hidden layers of neural networks
- Computing with GPU makes it relatively fast
- Trained model can be deployed in production to gain advantage of faster retrieval of optimal inputs


3. Yunpeng Pan and Jun Wang, "Two Neural Network Approaches to Model Predictive Control" , American Control Conference, 2008
