A-01

Applying RNN into system identification

Chen Cheng†　　Yukikazu Nakamoto†

Abstract

In the field of embedded control systems, we often use the method of system identification to model a plant that is a control target. System identification requires usually statistical methods to build mathematical models of dynamical systems from measured data and tune many parameters for that model. These are troublesome. Another approach to model a plant is to use neural network, especially recurrence neural networks (RNN). A user can build a model using RNN without. We describe preliminary experiment in applying RNN into modeling a plant.

1. Introduction

In the field of control systems, we often use the method of system identification to model a plant. System identification uses statistical methods to build mathematical models of dynamical systems from measured data. A common approach is to start from measurements of the behavior of the system (outputs) and the external influences (inputs) and try to determine a mathematical relation between them without details of what is actually happening inside the system. It requires, however, selecting an appropriate mathematical model and tuning many parameters for that model. These are troublesome. Another approach to model a plant is to use neural network, especially recurrence neural networks (RNN) (e.g. [1]). There are, however, unclear points in its design comparing with CNN. This paper addresses a preliminary design of RNN to model plants.

2. Method

This paper addresses how to construct a machine-learning program to model plants. To achieve this purpose, we make a machine-learning carry out the following steps.

1. Prepare input and output data of a plant: We collect the plant data by executing a model containing plants and controllers in MATLAB/Simulink.
2. Construct a machine-learning program: We make neural networks in different machine learning method, such as Recurrent Neural Network (RNN) and Convolution Neural Network (CNN) for comparison. Using the plant data as training data, we have trained the neural networks.
3. Execute prediction of neural networks: The neural networks receive the time series of input and output data from / to a plant until time *t-1* and an input value at time *t,* and output a plant values at time *t*.
4. Compare By import these models back to Simulink, we can compare the performance of them. Finding the differences of the several methods in control systems is also our purpose.

We choose RNN for machine learning because RNN is considered to be suitable for prediction with time series data such as a behavior of a plant.

3. Building and training Recurrence Neural Network

In this section, we present how to build and train RNN model using training data.

Preparation of training data

The first step is to prepare the dataset for the RNN. We prepare data set for RNN by using an automatic transmission controller as a plant model in MATLAB/Simulink [2] (See Fig.1). The version of MATLAB is R2019b. The input of the model is throttle and brake while the output is vehicle speed. We extract data of vehicle speed, throttle, and brake of the plant model as input values. After the extraction, we need to normalize the input variables.

We implement RNN as shown Fig.2 using Python 3, Tensorflow 2.0 and Keras 2.3.1.using Long Short-Term Memory (LSTM) [3] cells and train RNN with Then, The RNN predict a vehicle speed at the current time with vehicle speed, throttle and brake data during the previous period.

We define the LSTM with 300 neurons in one hidden layer with initializing parameters to zero and 1 neuron in the output layer for predicting vehicle speed. We use the Mean Square Error (MSE) loss function and the efficient Adam version of stochastic

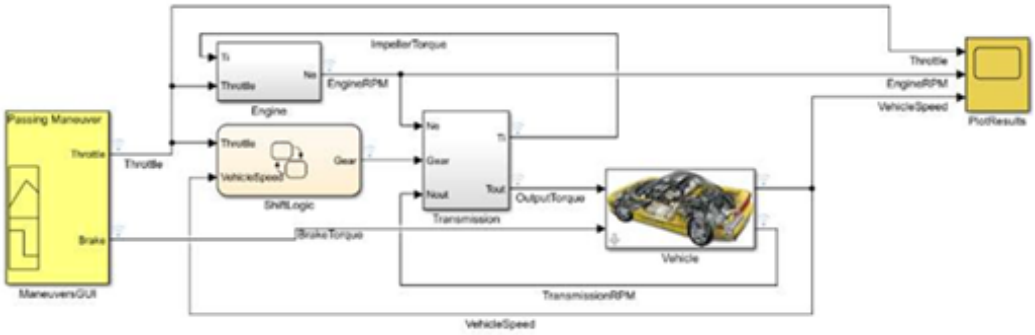

Figure 1. An automatic transmission controller model in MATLAB/Simulink [2]

†University of Hyogo

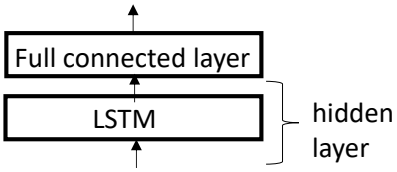

Figure 2. Architecture of RNN

gradient descent. The model be fit for 100 training epochs with a batch size of 32.

4. Evaluation of Model

After the model is fit, we can predict for the entire test dataset. We compare the prediction values with the test dataset and invert the scaling from the normalized data. We also invert scaling on the test dataset with the expected vehicle speed. With the predictions and the actual values in their original scale, we calculate an error score Root Mean Squared Error (RMSE) for the model. The result in Fig.3. shows that there is no difference between the actual values and the predicted ones. In Fig.3, X-axis is time (second) and Y-axis is a vehicle speed.

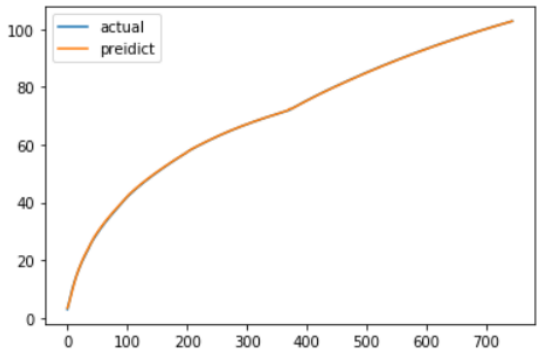

Figure 3. Actual values and predicted values by RNN

Next, we change the number of neurons in the first hidden layer to see how it effects the training. Besides 300, we have also tried build RNN with100 and 200 neuros. We show the result in Fig.4. Y-axis is training loss while X-axis is training epoch. From the figure, we can consider that we can get better results the larger number of neurons in the same training epoch.

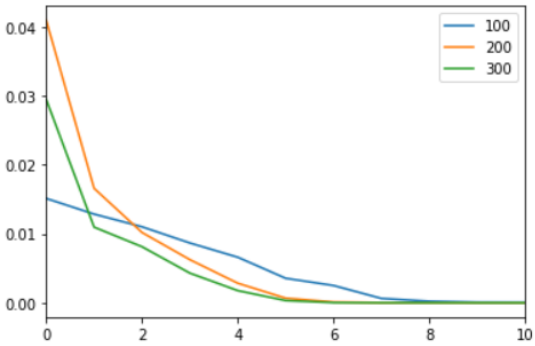

Figure 4. Training loss in RNN (from epoch 0 to 10)

For the comparison with CNN, we also use the same data to train and test CNN model. CNN is constructed by Flatten layer and Dense layer (Fig 5). In the same way as we do with the RNN model, we obtain results of the prediction and actual values. Unlike RNN, we can clearly find the difference between them. As mentioned earlier, we use RMSE to quantify this difference. RMSE of CNN model is 0.21 while RNN is only 0.17.

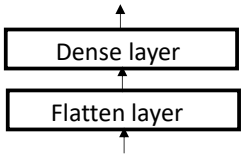

Figure 5. Architecture of CNN

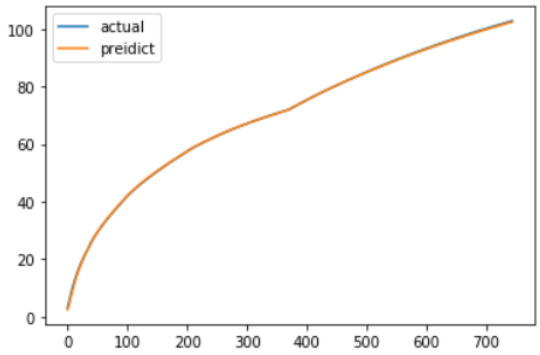

Figure 6. Actual values and predicted values by CNN

5. Conclusion

In this research, we consider that it is feasible to simulate a control system with a deep learning method, which is our goal. About the difference between RNN and CNN, there is almost no difference in the result of training step. It could be caused by the complexity of this system and need more research. Besides this, it is also significant how to implement such model with other platform like MATLAB/Simulink. This is the future research.

References

[1] G. Pillonetto, F. Dinuzzo, T. Chen, G. Nicolao and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," 2014, Automatica, vol.50, no.3, 657-682.

[2] Mathworks, “Modeling an Automatic Transmission Controller,” 2020. URL: https://jp.mathworks.com/help/simulink/slref/modeling-an-automatic-transmission-controller.html?lang=en

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory". Neural Computation. vol.9, no.8, pp.1735–1780 (1997).