Robots Pouring Time Series Regression Using RNN

Zihe Ye

***Abstract*— Liquid Pouring is a daily basic motion performed by human time to time, the accuracy of pouring is an important metric to measure how well the motion is done. As human can finish this task very naturally, it is not as easy for AI to do such a job. In this paper, we use RNN (recurrent neural network) to build a network predicting water volume after each time step for different trials, and this regression problem helps us better understand how well the robots perform in the trial samples. The paper tries different network layouts, and different normalization methods, with a final RMSE loss of 0.041 on 33 test sequences which have never been seen by the model.**

***Keywords*—Recurrent Neural Network, Data Normalization, Regression problem, Learning rate, LSTM, Masking, Optimizers.**

I. Introduction

Robotics fields has been growing significantly in recent several years, [1] mentioned robot cooking back to 2006, which indicates robot cooking has always been a task which scientists are devoted into. Among different procedures that a robot cook needs to do, pouring is a very basic but meantime one of the most important tasks. According to [2-3], pouring motion is the second most frequent used manipulation in robot cooking scenarios.

Pouring is a very easy task for healthy human to perform because human learn from experience since childhood with strong multi-sense but pouring accurately for machine is not as easy. Researchers in [4-5] use deep neural network, with a pouring PID controller and RGB-D point cloud with PID controller separately to predict water volume poured into a cup. Another work [6] uses reinforcement learning to help understand pouring policy, then verify the output on both simulation and real robots. The figure 1 below shows an example of robot pouring beer.

Figure 1 [10]

This paper is a partial repetition of [7-8], the two papers start from building a RNN network model, then tuning the model to the state with a loss of 10^{-5} level. Then the researchers in the two papers accommodate the model to predict and help real robots learn how to better pour liquids under different conditions combination.

There can be many ways to solve a regression problem. As the dataset we use for this paper is a time series dataset, RNN is inherently suitable for such scenario. Furthermore, with the situation that our time sequence is long, LSTM [9] mentioned in 1997 is very good at dealing with long time series regression problem.

The Section II talks about data and preprocessing in which normalization is deployed and discussed. The Section III is the Methodology section which displays the final model network layout and talks about the methods and tricks that are tried and used in this project. The section IV is the Evaluation and Results part, which talks about the final model loss on the test set, as well as comparison between different network layouts, normalization, hyper parameters tuning. The final section V is the discussion section which lists the key points concluded after the project, and some potential ways to tune the model to lower the loss.

II. Data and Preprocessing

A. Data

The Dataset for this paper includes a whole dataset with a layout of 688*700*7 NumPy array, the 688 is the number of samples, which is divided into three parts: 413 trials for training set, 138 samples for valid set, 137 samples for testing set. The final testing set is composed of 33 sequences provided by the project teacher assistant Juan. The second dimension 700 is 700 timesteps with 60Hz data capture frequency, the trials with less than 700 real timesteps are padded with zeros to 700. The third dimension is the number of features, the details are shown in figure 2 below.

[

$\theta(t)$	rotation angle at time t (degree)
$f(t)$	weight at time t (lbf)
f_{init}	weight before pouring (lbf)
f_{target}	weight aimed to be poured in the receiving cup (lbf)
h_{cup}	height of the pouring cup (mm)
d_{cup}	diameter of the pouring cup (mm)
$\dot{\theta}(t)$	velocity of the pouring cup (rad/s)

]

Figure 2 [11]

The f(t) is the target output which the model will predict, all other fields can be used as input. Only the first dimension-rotation angle at time t(degree), the second dimension-the

targeted output f(t), and the last dimension-velocity of the pouring cup are changing with respect to time, all other fields are constant.

Before dividing the original dataset into training, validation, testing sets or do any preprocessing, we plot the structure of all seven features as figure 3 below shows.

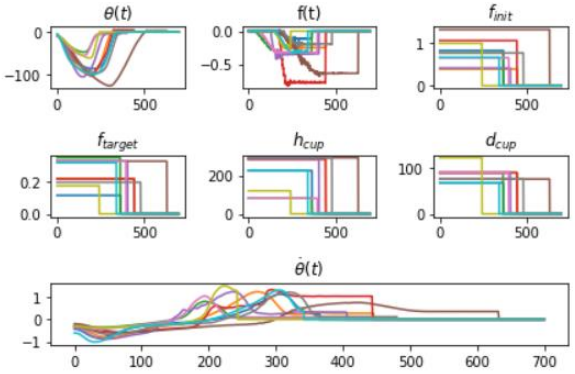

Figure 3

The figure 3 above shows that there are 4 possible input features have very large scale and variation which might affect our model training, therefore we try two different commonly used normalization, the figure 4 shows the result of standardization normalization, and the figure 5 shows the result of min max normalization. Both figure 4 and figure 5 are data after retracting the target predicting output.

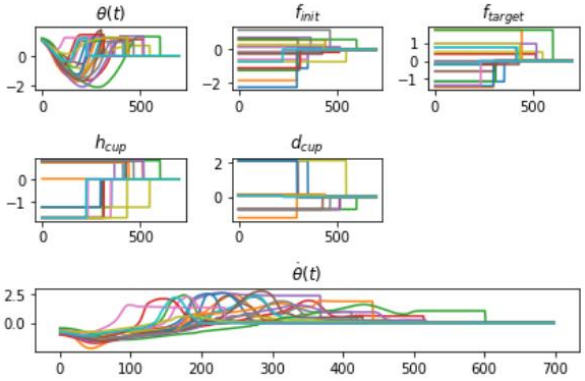

Figure 4

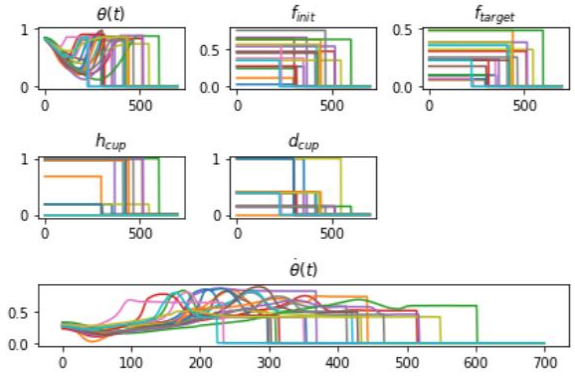

Figure 5

It can be seen from the two figures above that different features are closer in a small range after two types of normalization, the evaluation section will talk about the result of trials with two different types of normalization.

The target output we are going to predict stay as original without any preprocessing.

B. *Data preprocessing*

This subsection talks about the data preprocessing we use in this project. The preprocessing part in this section is all mostly about normalization, as known as regularization in some other terms, the other part is about handling zero paddings. The point to do normalization is to balance the weight of input to better train the model without much bias to one or a few of all features.

As the output may vary and the project requirement states, the second feature (as known as the output) are not to be normalized, hence the code extract the second row in the third dimensions for all three sets before normalization, and the extracted dimension are assigned to three ground truth NumPy array for later loss calculation use.

The program first calculates the padding zero starting index for all sequences (all features in one sequence have same paddings) from back, then store the stop indexes into a list for later use.

The program tries two different types of normalization, one is standardization based on global training set values for all six possible input features, each non-padding field will be normalized by subtracting global feature mean first, then divide by global feature standard deviation. Another normalization method is also based on global training set values for all six input features, the only difference is subtracting every non-padding field with global feature min and divide by the difference of global feature max and global feature min. After calculating the normalization scalars for all six features, these scalars will be used for any incoming datasets, including preprocessing validation set, testing set.

III. Methodology

This part is to introduce the steps building up the model, tuning methods and the main training and validation part, as well as the testing part.

The basic unit in Keras of Tensorflow is layer, in this project I used LSTM in Keras, the input are all six features. I did different trials on the number of LSTM Layers, LSTM units' number, Dropout Layers, loss function for model, optimizer, and learning rate. The final model is with four LSTM layers, each layer is composed of 16 units, there is a Dropout Layer after the final LSTM layer, and the Dropout rate is 0.2. There are two fully connected layers after the Dropout Layer, the following figure 6 is the final layout of the model submitted with this paper:

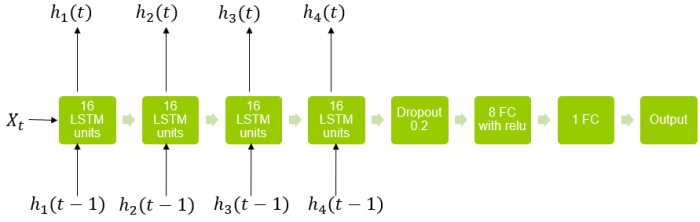

Figure 6

The methodology for building and tuning a good deep learning model is to try different combinations based on experience and other people's suggestions.

I have tried different combinations with factors including different normalization methods, different number of features input to the RNN, different LSTM layers, different LSTM units, with and without Dropout layer, Dropout ratio change, different number of fully connected layers, and change of learning rate as well as optimizer choice. The comparison between different model layout and hyperparameters choice will be shown in the following Evaluation and Results section.

IV. Evaluation and Results

A. *Evaluation*

This section illustrates in details different configurations I tried with a results analysis and summary. Figure 7 shows the comparison between ground truth and prediction of the model, while Figure 8 shows the training and valid set loss in 50 epochs.

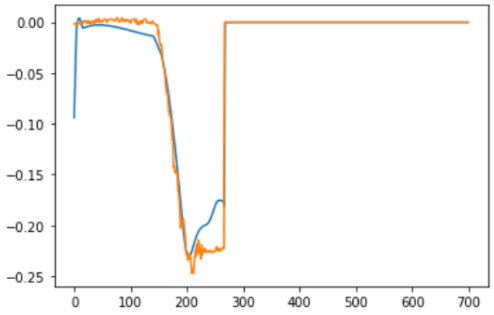

Figure 8

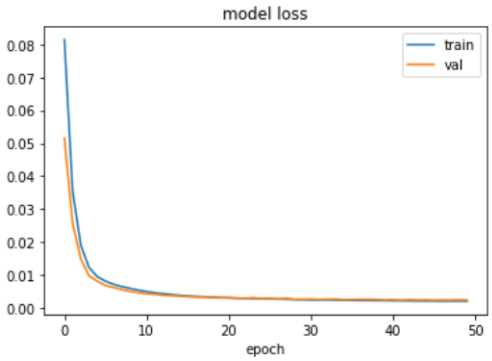

Figure 9

The changes I did can be divided into change the model, change the normalization part, and change hyperparameters.

The following table 1 is the change of model and results, and table 2 is the result of change on normalization type (data preprocessing), the table 3 is the result of change on hyperparameters.

Model Settings	Training Loss	Validation Accuracy/Loss	#Epoch
16 units LSTM* 1	0.0019	0.0024	50
12 units LSTM* 1	0.0024	0.0031	50
16 units LSTM* 4	0.0015	0.0023	50
16 units LSTM* 4 w. Maskin g	0.0013	0.0015	50
w. Dropou 0.5, all other are same to the block above	0.0019	0.0034	50

Table 1

The table above tries from one layer to many layers, as well as 12 units and 16 units. Besides, masking layer is considered as one of the possible changes of the network, and the dropout layer is the other. All networks above are with a 8 fully connected layers of relu activation and a 1 fully connected layer as output. The takeaway from table1 is that in 50 epochs, Masking layer and more LSTM layers perform better than other choices, Dropout is useful but not at the beginning. Dropout layer and ratio play an important role when reaching a large number of epochs with occurrence of possible overfitting, and I deployed a Dropout layer of 0.2 dropout ratio between LSTM and Fully connected layers after different trials.

Normalization Type	Training Loss	Validation Loss	#Epoch
Standardization	0.0022	0.0023	50
Min Max	0.0030	0.0040	50

Table 2

The above table shows the affect of normalization method, the standardization and min max normalization methods are only executed on input features that are not padded zeroes. All other layouts are as follow: 1 masking, 4 * 16 units LSTM, 1 Dropout of 0.2, 8 Fully connected layer, 1 fully connected layer. The table indicates that standardization could be better than min max as a normalization tool, but this is not certain. Both methods are very commonly used, but they are for different scenarios, after different trials I find there is no absolute advantage over Min Max by Standardization, but my program eventually deploys the standardization for two

reasons: 1, The standardization does a better job in squeezing data into a relatively uniformed range. 2, The standardization preprocessing does a good job for later training use.

Hyper Parameters	Training Loss	Validation Loss	#Epoch
Batch Size=1(default 1)	0.0033	0.0025	50
Batch Size=25	0.0039	0.0032	50
Batch Size=50	0.0036	0.0051	50
Learning rate=0.0001(batch size = 50)	0.0080	0.0043	50
Learning rate=0.001(batch size = 50)	0.0036	0.0051	50
Learning rate=0.01(batch size = 50)	0.0027	0.0017	50

Table 3

The Table3 shows that batch size and the learning rate affect the training loss and procedure. According to the result shown in the table, batch size does affect training loss, so a proper batch size can not only provide a better model but also can accelerate training process compared to smaller batch size. The learning rate is also one of the most important hyperparameters to tune, the result shown above is not the final answer and give a solid suggestion. But the general takeaway here is that bigger learning rate can reach to a small loss much faster than smaller learning rate, but it does not mean the final loss is also smaller. Smaller learning rate makes the model converges slower because of smaller step, but this can help prevent jumping effect without find the local or global minimum. The general solution is to try different epochs, batch size and learning rate and finally deploy a changing learning rate.

B. Results

After trying different combinations, I found the current model can reach to 0.0424 RMSE error on the 30% given test set at epoch 150 and can stay relatively stable after 100 epochs and before 250 epochs. Several helpful points I notice during building and tuning the model are as following: 1. Normalization does make a big difference compared to original input data, but which normalization method is better is still in doubt. 2. Masking Layer is useful, but it seems that the network can detect padding pattern and ignore in some methods. 3. Try different model layout and make it as complicated as possible before overfitting starts, then it is the time to add Dropout layer. 4. Should do early stop only until the validation loss stops decreasing. 5. Optimizer makes a huge difference in this problem, at the beginning I used SGD, and tried different model layout, but the valid and train loss can reach 10^{-3} level only after 300 epochs after many different trials, after changing to Adam optimizer the loss converges much faster than before.

V. Discussion

After finishing the project, I realized a lot of things I was not aware before, but there are even more things I am not clear about and need to be figured out in the future. RNN network is much faster than CNN network training, so it provides me more time and opportunities to try different combinations. I was stuck in optimizer for a few days which was the key factor stopped my loss going down, and I talked with other people since I am in doubt if my normalization is correct. But the truth is that I used an improper optimizer. This project teaches me that training deep neural network requires knowledge, searching ability, as well as patience just as any other program debugging procedure.

One of the important things I noticed and am curious about is the starting point of training. Even if everything except for the random set division is the same, every time a model training can have very different loss decreasing trend, which should be normal though. However, from what I observed that the possible starting point affects the training and validation final loss a lot in a few hundred epochs. So, at this point I have not figured out the theory behind this phenomenon and how to solve it correctly. One possible solution I can think of is to change the learning rate with steps, but from the data view, there can be local and global max and min, and they can be at very different point, so each time the network is trying to figure out a local extreme value, but different local extreme values can be very different. There should be a way to pick up a relatively good starting point, and I will devote myself to figure this out in the near future.

Acknowledgment

I want to show my gratitude to Dr. Sun for great lectures and Juan for being the TA of this class and provide me lots of great suggestions and help!

References

[1] F. Gravot, A. Haneda, K. Okada and M. Inaba, "Cooking for humanoid robot, a task that needs symbolic and geometric reasonings," Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006.

[2] David Paulius, Yongqiang Huang, Roger Milton, William D Buchanan, Jeanine Sam, and Yu Sun. Functional object-oriented network for manipulation learning. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2655–2662. IEEE, 2016.

[3] David Paulius, Ahmad B Jelodar, and Yu Sun. Functional object-oriented network: Construction & expansion. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 1–7. IEEE, 2018.

[4] Chau Do and Wolfram Burgard. Accurate pouring with an autonomous robot using an RGB-D camera. CoRR, abs/1810.03303, 2018.

[5] C. Schenck and D. Fox. Visual closed-loop control for pouring liquids. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 2629–2636, May 2017.
[6] Chau Do, Camilo Gordillo, and Wolfram Burgard. Learning to pour using deep deterministic policy gradients. In 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018.
[7] Yongqiang Huang and Yu Sun. Learning to pour. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 7005–7010. IEEE, 2017.
[8] Tianze Chen, Yongqiang Huang, and Yu Sun. Accurate pouring using model predictive control enabled by recurrent neural network. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019.
[9] Hochreiter, S. & Schmidhuber, J"urgen. Long short-term memory. Neural computation, 9(8), pp.1735–1780, 1997.
[10] Robots can already pour beer for you… – KUKA BLOG
[11] J. Wilches. Project 2: Pouring Dynamics Estimation Using Recurrent Neural Network, 2021.