# Long Short-Term Memory RNN for estimation of the Robotic Liquid Pouring

Hemanth Potlabathini

***Abstract*— For Cooking robots, estimating the amount of liquid it is pouring from one container to another is an important thing a cooking robot must be capable of doing. While cooking many recipes need some or other kinds of liquids like water, sauces etc., and each one is of different density and so the amount of liquid transferring may differ for different densities of liquids, for same angle of rotation. Recurrent neural networks have been giving good results for such kind of problems. In this paper we have implemented an RNN using LSTM for finding the amount of liquid poured by robot. LSTM have been providing some good results in majority of the problems now a days. The Root mean square error we got for this problem on the test data provided is 0.02202 and the Root mean square error on the test data generated from the original train data is 0.0039.**

***Keywords – Recurrent Neural Networks, LSTM, Robot Manipulation tasks, Robot grasping.***

## I. Introduction

Cooking robots need a lot of things to come together to work perfectly. One such thing is grasping. Grasping is a phenomenon, where a cooking robot can hold a physical object [2] and perform several manipulation techniques on it. Cooking robots, if perfected will be of massive help in our day to day life. They could save a lot of time and reduces our physical work by a huge margin. Once the robot is able to grasp the physical object it should also able to understand, what it is holding and should be able to perform, required manipulations to the objects, it has been holding. Suppose for a recipe it is required to add some liquid, then it must be able to grasp the liquid container and while adding the liquid to the recipe, it should know, how much quantity, it is putting into the pan. So, basically for performing most of the cooking, robots need to have an estimate of the quantity of the ingredients they will be using for cooking.

Performing such manipulation tasks are not at all easy as the robots not only needs to hold the container physically, it must also know the features of the container. Suppose if it is holding a cup, then it must have an idea about the diameter of the cup, the height of the cup and if the cup contains liquid then it must also have the idea about the density of the liquid. Suppose if you were transferring water to another container, it may take very less time than to transfer another liquid with high density. So, for robots to learn the liquid pouring [1] task, features like weight of the cup before pouring and weight of the cup after pouring will help them in understanding the amount of liquid transferred.

## II. Methodology

### A. *Problem challenge*

The major shortcoming of the traditional neural networks is that it starts from the scratch every time. But that is not the case with the Recurrent Neural Networks. Recurrent neural networks always try to remember from what it has learnt previously and relate it to what it is about to predict. This will have a lot of impact on the applications as we don’t do it from scratch every time as we have mentioned above. For the past few years, application of Recurrent Neural networks for a variety of problems have led to very successful results. In a recurrent neural network at a time step it accepts an input and output is forwarded to the next step. So, for the next step it looks for the input of that current step and the output of the previous step and tries to produce an output. Recurrent neural networks can be considered as traditional neural networks in loops.

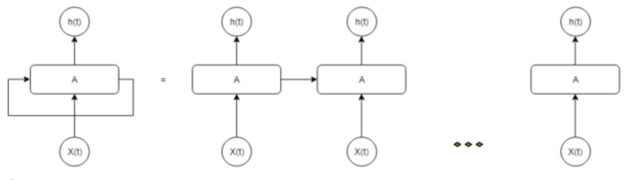

Fig 2.1 RNN Architecture

But standard Recurrent neural networks are not that good for long time dependencies. This is because the in standard recurrent neural networks gradient of the loss function decays exponentially and to solve this problem LSTM [4] were introduced, which stand for Long Short Time Memory. LSTM’s consists of special memory cells. These memory units along with standard units will help to maintain required information for a very long time. They also have special gates which help them to control what, it is storing, and it also takes care of information that needs to be forgotten.

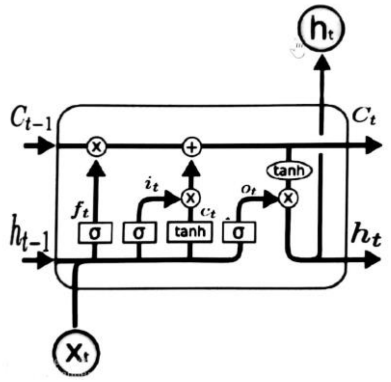

Fig 2.2 LSTM Architecture

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$C_t = tanh(W_C.[h_{t-1}, x_t] + b_c)$$
$$C_t = f_t * C_{t-1} + i_t * C_t$$
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Given a time step t, here f represents forget gate, I represent the input gate, c stands for cell, h represents the output, sigma represents the sigmoid activation function.

### C. Dataset

The dataset[5][6] is a NumPy array with the shape of (1307,1099,10). Here 1307 is the number of sequences and 1099 is the number of time steps and 10 represents the number of features for each time step. The dataset[5][6] is a tabulation of experiments performed in the laboratory. A cup holder was fitted with sensors and the experiment is performed, then all the values indicated by the sensor is tabulated.

$$\theta(t) = rotation\ angle\ at\ time\ t\ (degree)$$
$$f(t) = weight\ at\ time\ t\ (lbf)$$
$$f_{init} = weight\ before\ pouring\ (lbf)$$
$$f_{empty} = weight\ while\ cup\ is\ empty\ (lbf)$$
$$f_{final} = weight\ after\ pouring\ (lbf)$$
$$d_{cup} = diameter\ of\ the\ receiving\ cup\ (mm)$$
$$h_{cup} = height\ of\ the\ receiving\ cup\ (mm)$$
$$d_{ctn} = diameter\ of\ the\ pouring\ cup\ (mm)$$
$$\rho = material\ density/water\ density\ (unitless)$$

Our target is to estimate the force at given time t. And the data also consists of padding. So, the maximum time steps for a sequence is 1099, so if a sequence consists of less than 1099 then zeros are padded for that sequence.

Once the model is prepared, we have used a completely new test set for checking the accuracy of the network. This dataset consists of 289 sequences, each with 839-time steps and all 10 dimensions. They also include zero padding as the previous dataset.

### D. Data Preprocessing

We have tried to standardize the data by doing couple of data standardization techniques. We have tried to flatten the data by converting the three-dimensional array to two-dimensional array. We have used Pandas data frames for doing all the manipulations to the data. Once the data is converted to two-dimensional array, we have removed the padded zeroes. Then we have divided the data by 10. Then we have done Minmax scaling for the data. This will bring the entire data in the range of 0 to 1. Then we have divided the data into train, test and split. Test is of twenty percentage of the original data. Valid is twenty percentage of the remaining data after splitting test data. Then we have reshaped the data to 3d and passed it to the LSTM [4].

### E. Model

The model consists of three LSTM layers and 1 dense layer in total. The first layer consists of 32 memory cells and then a drop out of 10 percent is added to the model. The second layer consists of 16 cells and the third one consists of 8 cells. The dense layer in the end consists of 1 layer. Optimizer used was Adam [3] and Mean Squared error is the loss function used

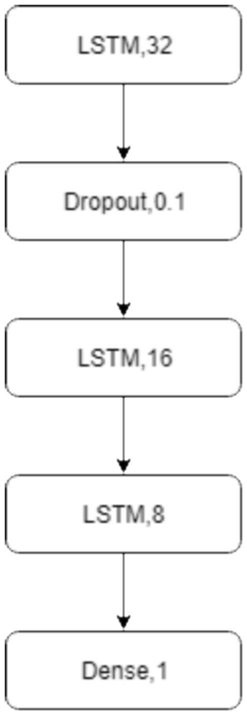

Fig 2.3 Model structure

## III. EXPERIMENTS AND RESULTS

Experiment 1: For the first experiment, the data standardization was similar to what I have done for my final model except that padding was not removed and the data was not divided by 10. This model consists of two LSTM layers with memory units of 16 and 16 respectively. The loss function used was Mean square error and the optimizer used was Adam. The model was run for 50 epochs and the loss were not converging and they aren't even close by. The number of parameters used were 3, The features removed are diameter of the pouring cup, diameter of the receiving container, height of the receiving container, force while the cup is empty, material density, height of the pouring container. The MSE error was 0.05 for this experiment

Experiment 2: After the first experiment, I have tried different things for experiment 1 like changing the optimizer, changing the loss function, changing the number of epochs and the number of memory cells in each LSTM layers but it has not fetched me good results. So, I decided to work on the data again. So, this time instead of converting the data into two-dimensional array and then removing the padding for the data, I have decided to keep the data as it is in three-dimensional array. After splitting the data into train, test and valid datasets, I have found the length of the minimum timesteps in each of the three datasets and for each sequence of a dataset, I have only taken time steps up-to the length of the least number of time steps in that dataset. This will automatically remove padding from all the three datasets. Then the features used were the same as the previous experiment. The number of LSTM layers were increased from 2 to 3. First layer contains 32 cells, and a drop out of 10% was added then second layer contains 16 cells, and the third LSTM layer contains 8 cells. Then a dense layer was added at the end. We have increased the number of epochs this time to 150. The results were good compared to the last time, and loss vs valid loss graph also indicated that the losses have converged. But the Mean squared error was 0.004, but the predicted values were completely different from the actual values. So, I must discard this model as well.

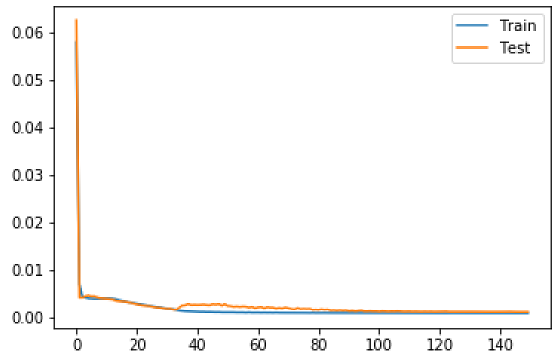

Fig 3.3 loss Vs Val loss for experiment 2

Experiment 3:

After trying many things to the experiment 2, then we decided to go back to the data preprocessing, we have used for the first experiment, but we have increased the number of layers to 3, similar to the second experiment. This time we have increased the number of features used to five. The features removed were height of the pouring cup, height of the receiving cup, diameter of the pouring cup, diameter of the receiving cup. The loss function used was Mean square error and the optimizer used was Adam. From all the experiments I have done these two yielded the best results. So, for major part of my experimentation I have used Adam as the optimizer and Mean square error as the loss function. The number of epochs has been increased to 250. I have also tried to run it for a greater number of epochs, but there wasn't much improvement with the increased number of epochs. The Root mean square error for the test data I have created was 0.0039. The losses have also converged when you observe the graph. The Root Mean squared error on the given test set was 0. 02202. Adam is the optimizer and MSE is the loss function.

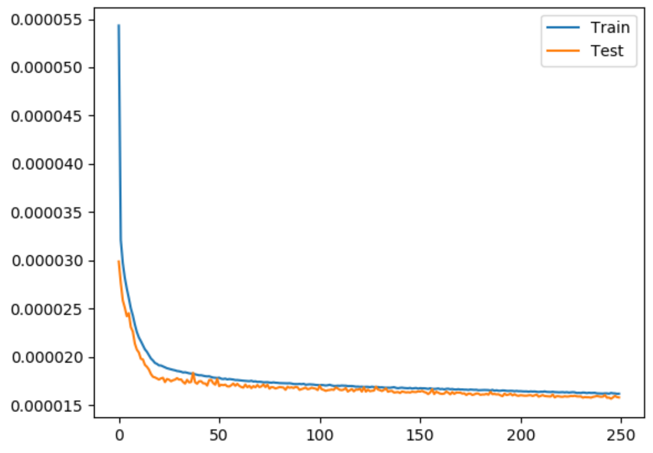

Fig 3.2 loss vs Val loss for experiment 3

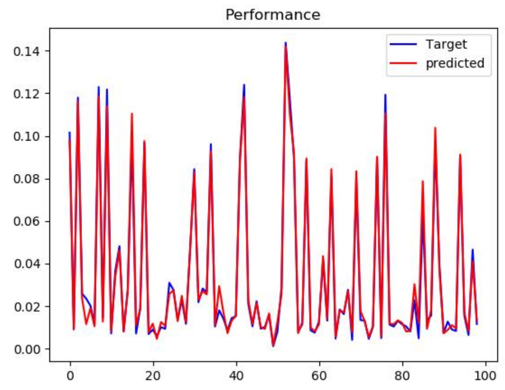

Fig 3.3 Target Vs Predicted for a sequence on own test data.

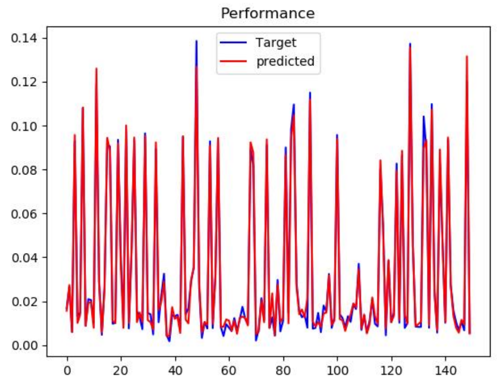

Fig 3.4 Target Vs Predicted for a sequence on own test data.

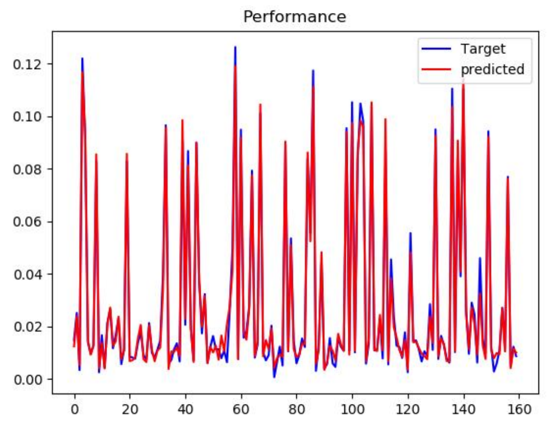

Fig 3.5 Target Vs Predicted for a sequence on own test data.

Then we have plotted the graph for different sequences in the test data provided as you can see above and new unseen test data. We have observed that is a lot of difference between the predicted and actual value when the model sees a new dataset. For the test data we have separated from the original dataset, both test and predicted are almost similar and the error is very less, We have observed that the error itself doesn't describe the network properly as, we can see we got a RMSE of 0.02 for unseen test set but the difference is huge when we look at a single sequence. Upon observing the dataset 2, it is clear that the features in this dataset certainly seemed to be different when compared to the dataset used for training, that may be the reason for such huge difference in the prediction. Moreover, due to the overfitting issue with the model, data may be predicted with such difference.

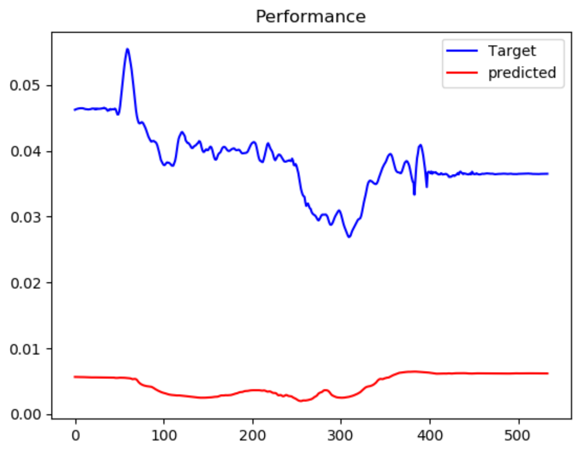

Fig 3.6 Target Vs Predicted for 27th Sequence of test data

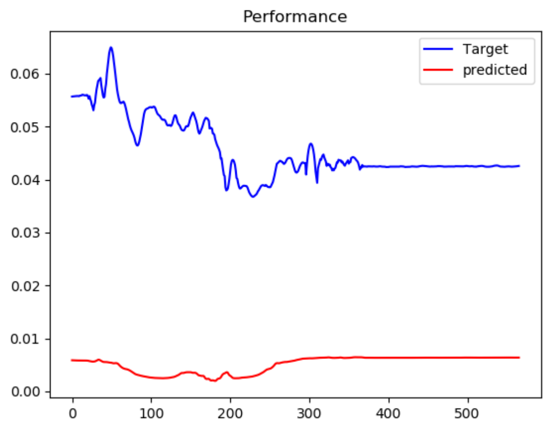

Fig 3.7 Target Vs Predicted for 60th Sequence of test data

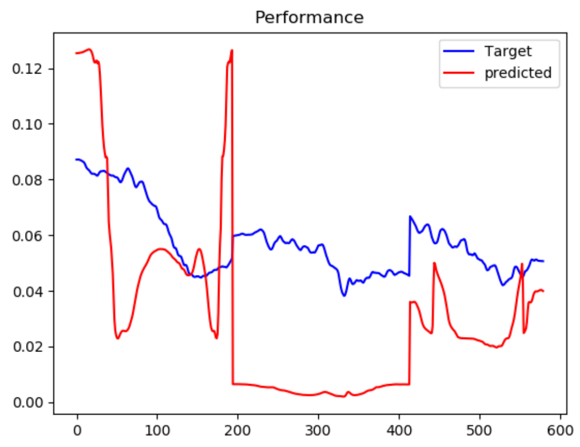

Fig 3.8 Target Vs Predicted for 98th Sequence of test data

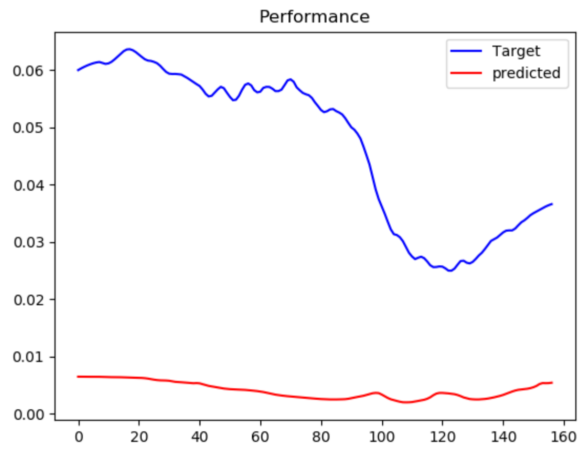

Fig 3.9 Target Vs Predicted for 177th Sequence of test data

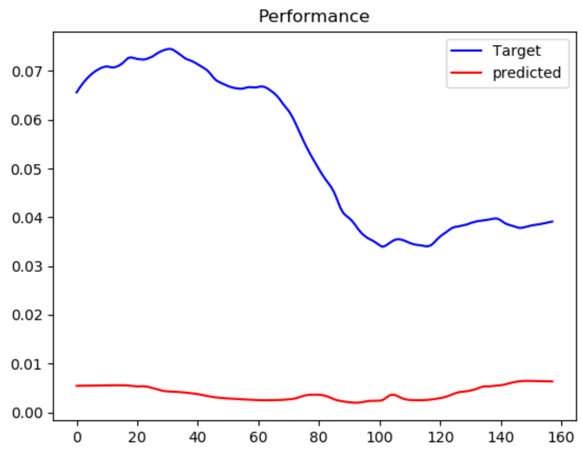

Fig 3.10 Target Vs Predicted for 226th Sequence of test data

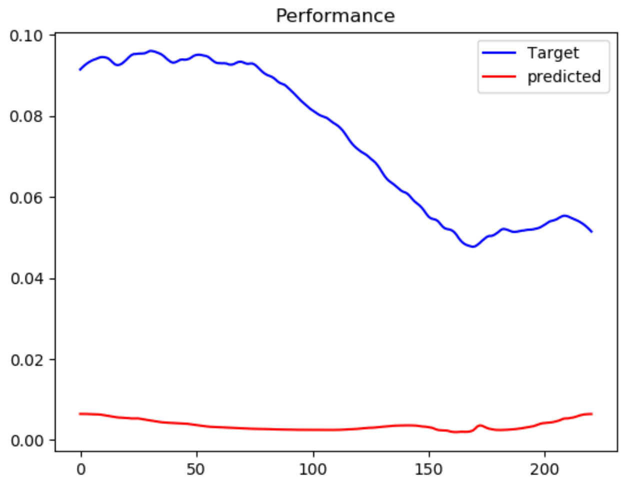

Fig 3.11 Target Vs Predicted for 248th Sequence of test data

## IV. Conclusion

In this paper we have used LSTM to estimate the weight of the cup at a time step t. We have used a total of 5 parameters, which we felt will be useful. The Root Mean square error for the test set separated from the dataset used for training is 0.0039, and the RMSE for the new unseen dataset is 0.022. The prediction for the unseen data has a bit difference. Overfitting may be the issue and the features of the dataset 2 may also be one of the reasons. So, feeding more data may increase the prediction accuracy and this means overfitting can be reduced in cases where huge data is available. Data preprocessing is also most important for such problems.

## References

[1] Y. Huang and Y. Sun, “Learning to pour,” in International Conference on Intelligent Robots and Systems (IROS), Sep 2017.

[2] Y. L. Yu Sun, Shaogang Ren, “Object–object interaction affordance learning,” in Robotics and Autonomous Systems, vol. 62, issue. 4, April 2014, pp. 487–496.

[3] D.P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in arXiv:1412.6980v9 [cs.LG], Dec 2014.

[4] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” in Journal of Neural Computation, vol. 9, issue. 8, Nov 15 1997, pp 1755-1780.

[5] Yongqiang Huang and Yu Sun. A dataset of daily interactive manipulation International Journal of Robotics Ressearch 2019.

[6] Yongqiang Huang and Yu Sun. A dataset of daily interactive Manipulation arXiv preprint arXiv:1807.00858, 2018.