Robot Modelling Using LSTMs and BP

Wang, Pengyu, 0425157 pwang2@laurentian.ca

Zhang, Haokun, 0424660 hzhang12@laurentian.ca

Zhu, Ziping, 0422426 zzhu4@laurentian.ca

Department of Engineering & Computation, Laurentian University

CPSC 5616: Machine and Deep Learning

Instructor: Dr. Meysair Zeinali

April 20, 2023

Table of Content

I. Abstract

Robot models are used all over the world every day, there are so many types of robots that can help us not only in academic or industrial fields but also in daily life, such as automatic drive, autonomous drones, cleaning bots, bionic arms, etc. The technique is mainly implemented by one of the Recurrent Neural Networks (RNN), the Long Short-Term Memory (LSTM), which belongs to the deep learning area. In this paper, our group used LSTM and Multi-Layer Perceptron (MLP) to train the dataset, then predict the future motion of robots. There are three features, position, acceleration and velocity as the input, torque and force are the output. Both of these two neural networks use backpropagation(BP) to be trained. Especially, LSTM useful in the engineering field because it chose to memorize or neglect past steps or behaviour. We built networks and implemented algorithms with MATLAB. Based on the results after running the algorithm, we will discuss the performance of LSTM and then make a comparison with traditional MLP neural networks.

***Keywords*— Robot, Modelling, Deep Learning, Recurrent Neural Network, Long Short-Term Memory, Multi-Layer Perceptron, Backpropagation**

II. Introduction

Artificial Intelligence and Deep Learning have been popularly used in modern robotic development, and they become the keys to robotic development. Robots are not only widely used in people's daily life, such as autonomous cars, autonomous drones, cleaning bots, assistant robots, etc. Nowadays, autonomous robots have been very popular in several fields, since autonomous means, based on their intelligent feature, can handle tasks and operate without human supervision and control. With technological leaps and bounds, robots are being more intelligent than ever past day. However, they are not only wildly use in daily life, but also in other fields, and their appearances show frequently. For example, in industry, industrial robots are used in manufacturing, because robots can cause fewer errors than humans, and on assembly lines, robots can assemble complex parts faster than humans. Autonomous robots giantly increase the efficiency of industrial manufacturing.

According to[1], Robot models simulate the kinematic and dynamic properties of manipulator robots and other rigid body systems. The models are *rigid body tree* objects containing *rigid body* and *rigid body joint* elements with joint transformations and inertial properties. In robot modelling, deep learning generally uses non-linear models, which are well capable of processing the learning period. With a given dataset, then it can learn directly from data. This approach is high efficiency and simple to implement. And we use supervised learning to train the Long Short-Term Memory (LSTM) model because there are inputs and outputs, obtained from the simulation of a dynamic model of the robots, said Patel et al., 2021 [2]. We will use the Multi-Layer Perceptron (MLP) by using backpropagation to train as well, then we will compare these two models' differences by the results and performances. The dataset is provided by Zeinali named *Robot Dataset_with_6 inputs and 2 Outputs.xlsx*, there are six inputs and two outputs, which are two arms with position, acceleration, and velocity as the input, and force and torque value as the output.

LSTM belongs to Recurrent Neural Network (RNN). RNNs take sequential input with any length and share the same weights in each timestamp. Furthermore, they can optionally produce output at each step. On the other hand, the biggest difference between traditional neural networks is that their structure can be folded, and the previous step output can affect subsequent input, because, this network uses the previous output or cell state as the input of the current cell or step. Based on its feature, this makes they can be utilized in handwriting recognition, speech recognition, natural language processing tasks, etc. Compared with Convolutional Neural Networks (CNN), RNNs have fewer features of compatibility,

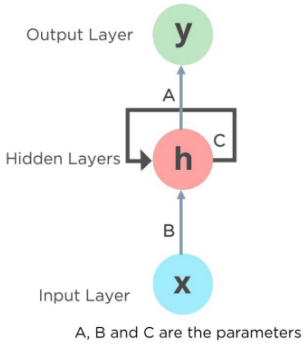
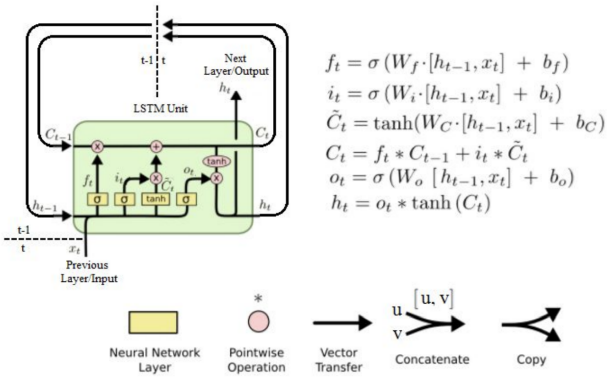


Fig. 1: Simple Recurrent Neural Networks (RNN) from [3]

and CNNs are more potent than RNNs. For RNNs, they accept arbitrary input, so their output length is arbitrary as well. However, CNNs are the opposite. RNNs, unlike feed-forward neural networks, can use their memory feature to process arbitrary lengths of inputs, because of their working principles and particular structures.

LSTM is the abbreviation of Long Short-Term Memory, since they are a type of RNN, they have the characteristics of RNNs, then they can process an arbitrary size of data with entire sequences, not only single data. They were proposed by Hochreiter and Schmidhuber in 1997 as a solution to the problem of vanishing gradients, the crucial part of LSTM, widely cited [4] since 2003, in the Journal of machine learning research, written by Gers et al. (2003) But they were known to the public since 2013 after Graves and Hinton brought this concept to Google [5]. They used deep RNNs for speech recognition. There are four main parts of LSTM: Forget Gate; Input Gate; Output Gate; and Cell State. And there are four activation functions used in it, three sigmoid and one tanh.



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

Fig. 2: Structure and activation functions of LSTM [6]

We will explain how they work and their structure in the following section.

III. Problem Formulation and Calculation

A. Multi-layer Perceptron

In neural networks, the basic network is the MLP, and we will train the model first by using backpropagation (BP) to MLP (Fig. 3). Since in the real world, most of the problems are non-linear and not linearly separable. The gradient of a composition function is computed by using BP, which is to search through all the possible parameters then to minimize the cost error function. The concepts of MLP and RNN, the difference between them is that RNNs allow for loops in the network. And their weights are shared in each timestamp.

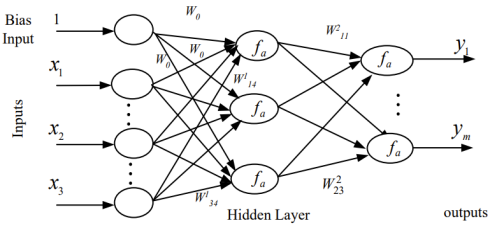


Fig. 3. MLP Structure [7]

Based on the results of MLP and LSTM, we will compare their performances. And choosing the best model to train and predict the motion of robots.

B. Gates and states

III.B.1 Feed-Forward

In the beginning, we need to clarify the notation and symbolic's meaning in Fig. 2: f_t is the Forget gate output; i_t is the Input gate output; $\hat{C}_t$ is noted as New candidate values; C_t is the New cell state; C_{t-1} is noted as the Previous cell state; o_t is the Output gate's output; and h_t is the Hidden state. The key point to understanding LSTM is to treat it as a conveyor, all the things are on a conveyor. After this, we look at the structure, the first one of it is the Forget Gate, which takes h_{t-1} the previous hidden state from the previous step and the input x_t at the current timestamp, then it will output **0** or **1**, as the output f_t of the forget gate, **0** means it will forget this "thing", **1** means remember this, the equation is shown as Eq. 1. It is noted that W and U are both weights, and their weights are not the same, they do not share the value for all the gates and states. Next, we walk through to Input Gate, then the cell state

$$f_t = \sigma(W_f \bullet h_{t-1} + U_f \bullet x_t + b_f) \quad (1)$$

will be updated upon the output of the Input Gate and candidate value. There are two sections of the Input Gate, the Input layer, which receives the previous hidden state and value of the current input, is the same as Forget Gate. Then we forward to the tanh layer, which outputs a vector of new candidate values. After this, the outputs from the tanh layer and Input Gate layer will be put together. It will be used to update the state of the cell, the equations are shown in Eq. 2 and 3. In the third step, we will focus on the cell state, by

$$i_t = \sigma(W_i \bullet h_{t-1} + U_i \bullet x_t + b_i) \quad (2)$$

$$\hat{C}_t = \tanh(W_c \bullet h_{t-1} + U_c \bullet x_t + b_c) \quad (3)$$

values received from the Input layer and tanh layer, use these values to update the old cell state, C_{t-1} to get the new cell state C_t. To do this, it multiplies the previous state with the output of Forget Gate and then adds the value after candidate $\hat{C}t$ multiplies with the value of Input Gate i_t, the result will be C_t, and the equation as shown in Eq. 4. Finally, we are at the end

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \quad (4)$$

of the conveyor, Output Gate. First, like what it does at Input Gate and Forget Gate, it takes the hidden state and current input values to the sigmoid activation function. Then put the cell state through tanh, and after doing this, multiply with the sigmoid function value, we got the new state of the hidden layer. The equation is shown below in Eq. 5 and 6. After all

$$o_t = \sigma(W_o \bullet h_{t-1} + U_o \bullet x_t + b_o) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

the cells end of processing, the final output depends on the hidden state h_t, the calculation is shown below

$$\hat{y} = w_y * h2 + b_y \quad (7)$$

As shown in Eq. 7, $\hat{y}$ is the value of the final output. However, we must get the error and backpropagate the signal to update the weights. The error cost function we use is Means Squared Error (MSE) in Eq. 8.

$$E = \frac{1}{2}(y - \hat{y})^2 \quad (8)$$

So far, we have done the feed-forward step. Next, we will go through the feed-backward step.

III.B.2 Feed-Backward

For the feed-backward step, according to Graves et al., 2013 [8], chain rules and derivative knowledge are applied here. The first pace along here is to calculate the gradient of the error with the output we get:

$$\frac{\partial E}{\partial ot} = \frac{\partial E}{\partial ht}\frac{\partial ht}{\partial ot} = (y - \hat{y})(-w_y)\tanh(C_t) = \delta o_t \quad (9)$$

$$\frac{\partial E}{\partial ot} = (y - [w_y(h_t)+b_y])(-w_y)(o_t)(1-\tanh^2(C_t)) = \delta o_t \quad (10)$$

Eq. 10 is the equation after expanding Eq. 9. Then we go back to gates and states. There are three gates, Input Gate, Forget Gate and Output Gate. For Input Gate as shown in Eq. 11, Forget Gate as shown in Eq. 12, the new cell state is shown in Eq. 13, the previous cell state C_{t-1} as shown in Eq. 14:

$$\frac{\partial E}{\partial it} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial it} = \delta_{Ct}\hat{C}_t = \delta_{it} \quad (11)$$

$$\frac{\partial E}{\partial ft} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial ft} = \delta_{Ct}C_{t-1} = \delta_{ft} \quad (12)$$

$$\frac{\partial E}{\partial \hat{C}t} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial \hat{C}t} = \delta_{Ct}i_t = \delta_{at} \quad (13)$$

$$\frac{\partial E}{\partial Ct} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial Ct} = \delta_{Ct}f_t = \delta_{Ct-1} \quad (14)$$

After going through states and gates, firstly, we need to get the input $\delta_{\hat{a}t}$ to the system, as shown in Eq. 15, then the net input of the Input Gate $\delta\hat{\imath}$ (Eq. 16), Forget Gate (Eq. 17) and Output Gate (Eq. 18) need to be calculated by the derivative of the sigmoid function as well. Based on the concepts and

$$\frac{\partial E}{\partial netCt} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial \hat{C}t}\frac{\partial \hat{C}t}{\partial netCt} = \delta_{at}(1 - \tanh^2(net_{ct})) = \delta_{\hat{a}t} \quad (15)$$

$$\frac{\partial E}{\partial netit} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial it}\frac{\partial it}{\partial netit} = \delta_{it}i_t(1-i_t) = \delta_{\hat{\imath}t} \quad (16)$$

$$\frac{\partial E}{\partial netft} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial ft}\frac{\partial ft}{\partial netft} = \delta_{ft}f_t(1-f_t) = \delta_{\hat{f}t} \quad (17)$$

$$\frac{\partial E}{\partial netot} = \frac{\partial E}{\partial Ct}\frac{\partial Ct}{\partial ot}\frac{\partial ot}{\partial netot} = \delta_{ot}o_t(1-o_t) = \delta_{\hat{o}t} \quad (18)$$

calculations, we implemented LSTM by MATLAB. The results are shown in the section. III.

IV. Results and Discussion

A. Multi-layer Perceptron

When we first started the project, we tried to use the Logistic Sigmoid function to be the activation function. However, the actual outputs of the dataset are not from 0 to 1, so our predicted output image cannot match. And we used sequential training instead of batch training, which means reading the input and output from the given data set line by line. Implementing, it is easy. But the results of the training, and validation were not good.

So, there were two parts we lost for a while, for the activation function, we choose the sigmoid function, which related to the output value fluctuating between 0 and 1. The second failure was that we used sequential training, which decreased the learning efficiency, and increased the running time.

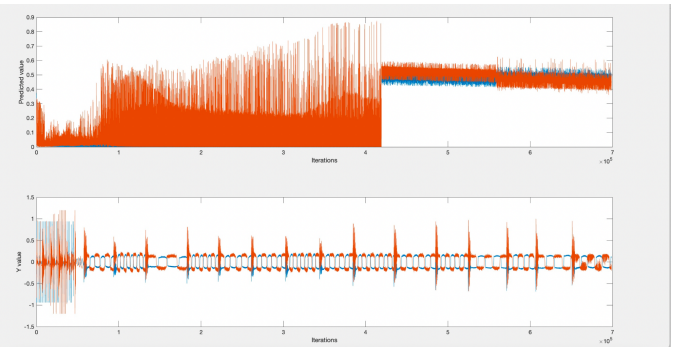


Fig. 4: Prediction value and Actual value by using the sigmoid function

The upper part of Fig.4 shows the predicted value after using the sigmoid function (including 60% training set, 20% validation set, and 20% test set). The number of hidden layers is only 1, and the number of hidden neurons is 5 in the training set, then adjusted to 6. The learning rate was 0.2 at the training set, then we adjusted the learning rate to 10^{-5} in the validation set. When we go further, we set the number of neurons to 10, the change was not obvious. Due to the lack of values from -1 to 0, the weights cannot be updated correctly after training, so the output results cannot match the actual output values in the lower part of Fig.4. At the same time, the results obtained by the validation set and test set are very different from the training set, which also belongs to high bias and high variance.

We have tried many models and after observing the output value from the dataset, we found the values were not from 0 to 1, but from -1 to 1. Finally decided to use MLP with the tanh activation function for training a prediction model of robot control data. In this model, we use 1 hidden layer with 6 neurons to form the neural network. The learning rate is 10^{-6} and we trained the network with 50000 iterations. All weights are randomly initialized with small values, and biases are set to 1. In order to maintain the validity of the model, we used the first half of the data set and left the remaining data untouched. In the first half of the data set, we separated it with a 6:2:2 ratio for the training set, validation set, and test set. The following picture shows the results of the model.

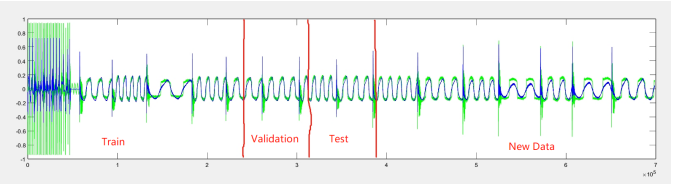


Fig. 5: Prediction value(Blue) and Actual value(Green) for output 1 by using the tanh activation function vs. Iteration

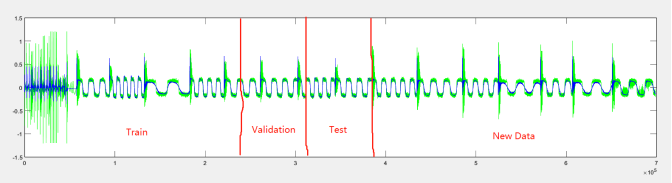


Fig. 6: Prediction value(Blue) and Actual value(Green) for output 2 by using the tanh activation function vs. Iteration

With these pictures, we can clearly see that the model has successfully learned the patterns in the data set. The model can predict a similar pattern in unseen data. Although we still observed loss decreasing while we were training, we met the vanishing gradient problem. The loss decreases too slowly and we consider the time limit and the ability of our computers. We decided to keep this model with a loss close to 0.0098. Since we didn't observe any overfitting, or increase in loss during the training, it only requires more iterations in training to obtain a more accurate model. The trained weights are as follow:

1.22227091	-0.8212307307	2.13933466	-0.491933373	1.646118721	2.171734636
1.597413129	3.742468615	1.28744379	1.565381345	-0.7406746898	0.3535640253
-1.023337958	2.061356051	0.8365300275	1.119180511	1.852907328	0.4493358898
0.8948551667	-1.713692731	0.2404084712	0.2448756997	-1.103282895	1.247970481
-0.01974427379	3.819676088	-1.895171429	0.1397812358	-1.009800391	-0.8598075
-0.6047949371	0.745767102	0.07842418334	0.3555842562	-0.2244583311	0.4143087331
0.1519313811	-0.2448478095	0.02610876636	-0.1847860716	-2.163181927	0.09806514444

Table. 1: Weights from the input layer to the hidden layer. Each column contains weights for one neuron in the hidden layer, and the first row contains weights for biases.

0.2974939938	0.1973135109
0.09550874684	-0.05455550554
-0.1774175214	0.2702229374
-0.5030793391	-0.3694719218
0.1382102322	-0.3584293826
0.09172003291	0.03563127661
0.1019672594	0.1452341155

Table. 2: Weights from the hidden layer to the output layer. Each column contains weights for one neuron in the output layer, and the first row contains weights for biases.

For this model, we compare the train loss and validation loss to determine whether the model has been over-trained or under-trained. In Fig. 7, the training loss(yellow) is higher than the validation loss(blue). This declares that the model has more accuracy on the training data set. The model does not meet an overfitting situation since we always have train loss higher than validation loss. If train loss is lower than validation loss and the difference between these losses is significant, the model is over-trained.

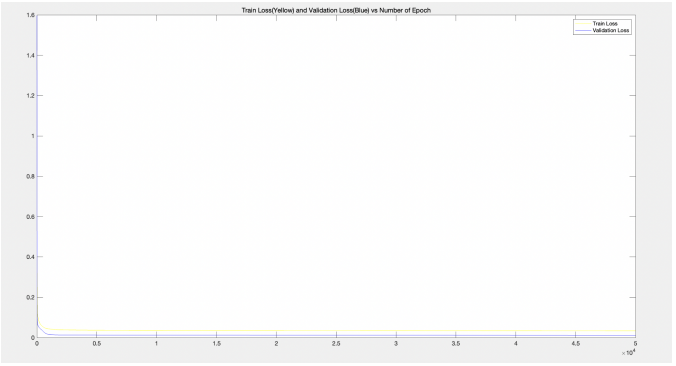


Fig. 7: Train loss(yellow) and validation loss(blue) vs. Iteration

B. Long short-term memory

In this project, we also tried to implement the LSTM model to complete the data prediction generated by the robot arms. In this model, we set the learning rate to 0.0001, the number of neurons to 6, and the bias to 1. At the same time, we divide the data set into three parts: training set (60%), validation set (20%), and test set (20%) to verify the accuracy of our prediction results. According to the diagram of LSTM, we used sigmoid and tanh functions as our activation functions at different gates. The figures below are the results we got.

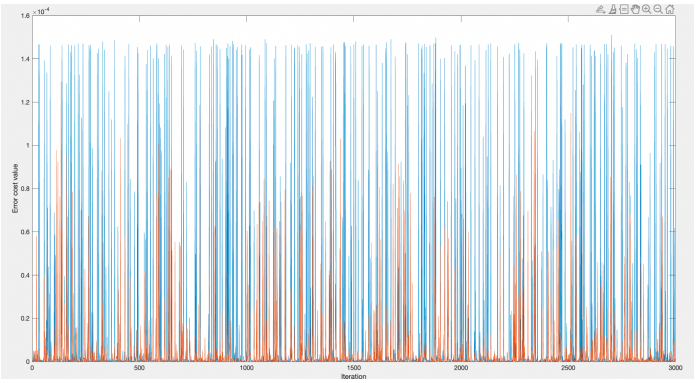


Fig. 8: Trend graph of error cost and iterations

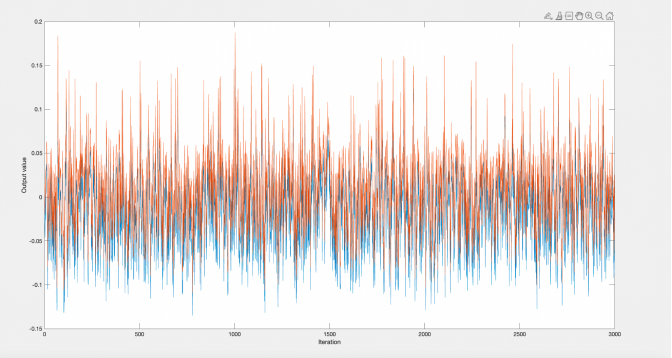


Fig. 9: Trend graph of prediction outputs and iterations

Fig. 8 shows the trend graph of the Error shown by the two outputs (the red and blue lines represent the two outputs, respectively) after the training iteration. Our training results do not show a downward trend in errors, and the gap between the predicted and actual results has not decreased. Fig. 8 shows the values predicted by the two outputs after the training iteration, which does not coincide with the actual values' image, indicating that we still need to implement the LSTM prediction model.

This is because when we update the weights by feed-backward, there was a problem when constructing the derivative function. We followed what we understand from the section. II, successfully get the results of the values of error by using the mean square error (MSE) function. In the calculation of derivatives at Forget Gate, Input Gate, Output Gate, and updating the cell states, we failed at calculation by using chain rules. So, only the feed-forward step was calculated successfully, then the backward step for updating the weights failed. We tried our best to do it.

V. Conclusion

This project evaluates the motion data set of the robot provided by Prof. Meysar Zeinali by using the machine learning method of long short-term memory networks (LSTM) and Multilayer perceptron (MLP). In this way, the model simulates the force and torque required by the robot's two arms to achieve different situations.

The prediction result of the MLP model in the project is very close to the actual data, and its training error curve is gradually decreasing. This shows that the model currently has moderate bias and moderate variance, proving the model simulation's accuracy. If the user adds more data features, the model can predict more complex situations to help the robot make complex actions.

The LSTM model is currently running feedforward normally and can give reasonable results. However, in the model's backward step, the function cannot adapt to this model, resulting in the current model's inability to update the weights to reduce output errors correctly. The learning curve of the project also shows that the trend of predicted results cannot be kept similar to the trend of actual results.

In summary, the machine learning model accurately predicts robot behaviour. In the future development of robots, robots need to perceive the environment to perform corresponding actions. A large amount of data training through machine learning technology can help robots accurately complete the tasks they should perform, thereby improving their intelligence level and application scenarios.

VI. Acknowledgements

Thanks to Dr. Meysar Zeinali for instructing our group's project this term. Much appreciate every one of our team spared no efforts in the work of this research. During the research period, we face challenges but never give up, the spirit of our team can cheer us up forever. Finally, thanks for everyone coming and listening.

VII. References

[1] "Robot modelling and simulation," Robot Modeling and Simulation - MATLAB & Simulink. [Online]. Available: https://www.mathworks.com/help/robotics/modeling-and-simulation.html. [Accessed: 07-Apr-2023].

[2] R. Patel, M. Zeinali, and K. Passi, "Deep learning-based robot control using Recurrent NeuralNetworks (LSTM; GRU) and adaptive sliding mode control," *International Conference of Control, Dynamic Systems, and Robotics*, 2021.

[3] A. Biswal, "Recurrent neural network (RNN) tutorial: Types and examples [updated]: Simplilearn," *Simplilearn.com*, 14-Feb-2023. [Online]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn. [Accessed: 08-Apr-2023].

[4] F. A. G. IDSIA, F. A. Gers, Idsia, Nicol N. Schraudolph Inst. of Computational Science, N. N. Schraudolph, I. of C. Science, J. S. IDSIA, J. Schmidhuber, and O. M. V. A. Metrics, "Learning precise timing with LSTM recurrent networks," The Journal of Machine Learning Research, 01-Mar-2003. [Online]. Available: https://dl.acm.org/doi/10.1162/153244303768966139. [Accessed: 09-Apr-2023].

[5] A. Graves, A. -r. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 2013, pp. 6645-6649, doi: 10.1109/ICASSP.2013.6638947.

[6] "Understanding LSTM networks," *Understanding LSTM Networks -- colah's blog*. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 09-Apr-2023].

[7] Zeinali. M. (2023) CPSC 5616 Machine and Deep Learning, Lecture 3, Chapter 3: Artificial Neural Network, Neuron, Perceptron, and Backpropagation Algorithm[PowerPoint slides]

[8] A. Graves, A.-rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.