International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 09 | Sep 2020 www.irjet.net p-ISSN: 2395-0072

LSTM-RNN for Radar Data Processing

Varsha Nimbaragi[1], Dr. Smitha Sasi[2], V jithesh[3]

[1]Student, Dept. of Telecommunication Engg, Dayananda Sagar College of Engg, Bengaluru, Karnataka, India
[2]Professor, Dept. of Telecommunication Engg, Dayananda Sagar College of Engg, Bengaluru, Karnataka, India
[3]Scientist E, LRDE, DRDO, C. V. Raman Nagar, Bengaluru, Karnataka, India

---***---

Abstract - *A Radar data processor connects the obtained measurement data and tracks and predicts target parameters such as the target position (radial distance, azimuth, and pitch angle) and the motion parameters (velocity and acceleration, etc.) The data processing is complicated by target maneuvers, closely spaced targets, limited resolution of the radar and missing measurements. Neural networks like Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) are able to model problems with multiple input variables. This is very beneficial in time series forecasting, where the other classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems. The aim of this project is to explore the feasibility of using deep learning techniques such as LSTM-RNN in radar data processing to assist in scenario generation and tracking.*

Key Words: Long short term memory, Recurrent neural network, TensorFlow, keras, Benchmark Scenarios.

1. INTRODUCTION

Artificial Intelligence (AI) is showing all sorts of progress in many fields and can be applied to many applications. Advancement of AI technology made deep learning techniques adaptable in many applications. **Deep Learning** is a subfield of machine learning which is concerned with algorithms and it is inspired by the structure and function of the brain called **Artificial Neural Networks (ANN).** The ability of Deep Learning is that it gives the model more flexibility to choose the way to decide how to use data to best effect. The application of Knowledge-based expert system (KBS) technology is the recent advancement in Radar. These classical methods of neural networks proved successful in many tasks but complicate the data processing as they cannot solve problems with multiple input variables. Neural networks like Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) are able to solve problems including multiple input variables which is beneficial in time series forecasting.

Recurrent Neural Networks (RNNs) are networks with loops in them, allowing information to continue to exist. LSTMs are designed to avoid the long-term dependency problem. LSTMs remember the information for long periods of time. They don't struggle to learn. The Long Short-Term Memory network or LSTM network is a type of recurrent neural network employed in deep learning because very large architectures will be successfully trained[1]. In this paper, it is discovered how to develop LSTM networks in Python using the Keras deep learning library to scenario generation and time-series prediction problem. How to implement this model has been studied in [2] - [7]

Blaire, et. al, had introduced 6 benchmarking trajectories in their paper These benchmarking problems are now widely used for benchmarking Radar Trackers[8].

The objectives are as follows: (1) Develop a Deep Learning Model to generate realistic target scenarios with different maneuvering conditions. The scenario shall contain target state information such as position, velocity & acceleration. These generated scenarios can be used in evaluating a radar data processor. (2) Develop a Deep Learning model to predict target states and compare these results against a multi-model Kalman Filter output. This model may assist in coasting when measurement data is not available. The predicted states can also be used for fusing with the Kalman filter outputs to achieve better accuracy. Google TensorFlow Deep Learning framework along with Keras library will be used in this study. The development will be carried out in Python under the Anaconda environment.

2. IMPLEMENTATION

The approach followed for implementing the LSTM based tracker is described in this section. The LSTM based tracker program has two parts: a Trainer and a Predictor. The Trainer program gets trained on a dataset containing a training trajectory and saves the trained model. The predictor model predicts the next position of the target.

2.1 Training Data Generation

A comprehensive set of scenario data of depicting various target maneuvering conditions such as constant velocity, constant acceleration, level turn, climb, turn with acceleration, turn with climb and turn with climb & acceleration has to be generated first. This will be performed though the implementation of motion models in MATLAB. This data is used to develop the LSTM-RNN model. The training dataset represent a target trajectory, which consists of the target's position, velocity & acceleration components and time.

The following hyper parameters are to be considered during the implementation : Number of Epochs, number of LSTM Layers, number of Neurons per hidden layer, LSTM memory length, batch size, Learning Rate, Optimization Algorithm, loss function and dropout (Yes or No).

2.2 Load Dataset

The first step is to define the number of features in the dataset, number of values to be memorized (represented by the variable 'lookback') by the LSTM and the percentage of data to be used for training. The lookback is set as 100 and 70% of the data is used for training and the rest of the data is used for testing. And next step is to load the dataset into memory. The data file is loaded into memory using the Pandas read_csv() function.

2.3 LSTM Data Pre-processing and Preparation

Here, the next step is to prepare the dataset for the LSTM. This involves framing the dataset as a supervised learning problem and normalizing the input variables. All the features are normalized using a min-max scalar to fit into the range [0, 1], and then the dataset is transformed into a supervised learning problem. The entire training dataset has to be converted to arrays of two sets, where the first set containing number of samples equal to 'lookback' and the second set contains the next sample.

2.4 Define the LSTM Model

In this section, fit an LSTM on the multivariate input data. First, split the prepared dataset into train and test sets, then splits the train and test sets into input and output variables. Finally, the inputs (X) are reshaped into the 3D format expected by LSTMs, namely [no. of samples, lookback, no. of features].

Now LSTM model can be defined and fit it for the training data. The model definition varies as the no. of layers and the no. of neurons per layer are hyper parameters. Different values have to be tried to fine-tune these parameters to arrive at the optimum number of layers and number of neurons. A 3-hidden layer LSTM model with 150 neurons in the first hidden layer, 150 neurons in the second hidden layer and 100 neurons in the last layer is considered here. The output layer has 9 neurons representing the 9 features in the scenario. Mean Absolute Error (MAE) loss function with the efficient "*RMSprop"* version of stochastic gradient descent is used in this implementation.

2.5 Train the Model

The model is then trained for 40 epochs with a batch size of 32. During this training, 30% of the training data is used for validation. During training Callbacks are used to get a view on internal states of the model. The tf.keras.callbacks.ModelCheckpoint call back allows to continually save the model both during and the end of the training. The model is saved only when there is an improvement in the validation accuracy. Once the training phase was completed, the trained LSTM model with best validation accuracy got saved in to a hierarchical data format (.hd5).

```
---
Epoch 00039: val_acc did not improve from 0.99716
Epoch 40/40
12320/12320 [==============================] -
1815s 147ms/step - loss: 3.9384e-05 - acc: 0.9960 - val_loss:
1.3530e-05 - val_acc: 0.9977

Epoch 00040: val_acc improved from 0.99716 to 0.99773,
saving model to
Scenario_Predictions_Scaled_Circular_3Features_1500U_100
LB_Noisy.hdf5
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

Plots showing the model accuracy & model loss during training & validation are shown below:

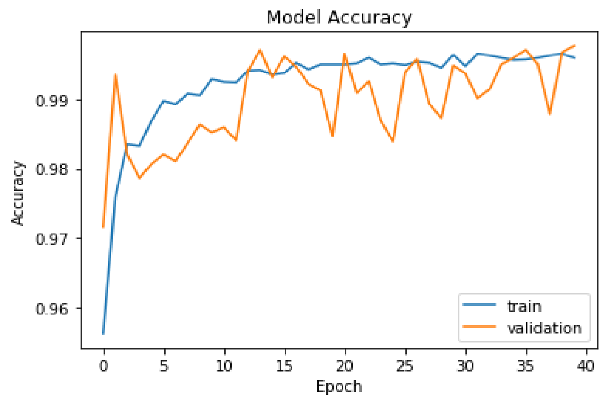

Fig -1: Accuracy Plot

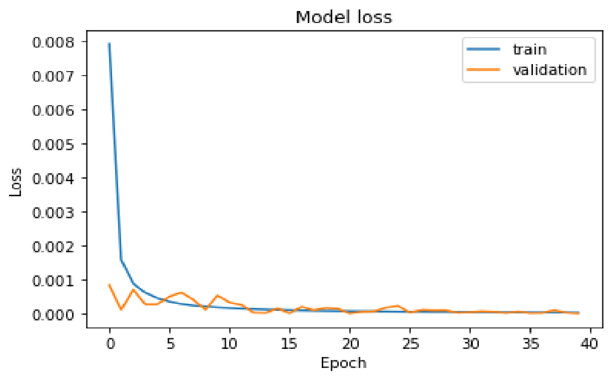

Fig -2: Loss plot

2.6 The LSTM Predictor

This is the second module of the LSTM tracker. This program loads the trained LSTM model and the measurements. It then pre-processes the measurements to the format required by LSTM. It may be noted that, as our LSTM model has a lookback of 100, the LSTM based tracking can start only after receiving 100 measurements. It will start predicting the 101th position based on the first 100 measurements, 102th measurement based on the previous 100 measurements, and so on. As the data is scaled before passing to the LSTM model, an inverse scaling is performed after the prediction to bring back the predicted position to the original scale. These predicted positions are then saved to a *.csv* file and is used to compare against the measurements and the output of an IMM based tracker.

2.7 IMM Filter Implementation

The Interacting Multiple Model (IMM) algorithm, assumes that the transition between different models is subject to the finite Markov chain of the given transition probability, and obtains the state estimate of the target by accounting for the interaction between several models. The models interact with one another to track the maneuver of a target. The IMM algorithm implemented here uses e filters: a Constant Velocity (CV) Kalman Filter, a Constant Acceleration (CA) Kalman Filter and a Constant Turn (CT) Kalman Filter. The MATLAB toolbox namely, Sensor Fusion and Tracking Toolbox is used for the implementation.

3. EVALUATION

In order to evaluate the trained LSTM model, a two-step approach has been followed. In the first step, the trained model was asked to predict the last 30% of the training dataset. In the second step, first 100 samples (as the look back was taken as 100) of six radar tracker benchmark scenarios were given to the trained model to predict the 101th samples, samples 2 to 101 were given to predict sample 102, and so on so that the entire scenario gets predicted. In both cases the performances of the LSTM model were compared against that of a 3-model Kalman Filter based IMM tracker. The prediction output for the last 30% of the unused training data is shown in below figures.

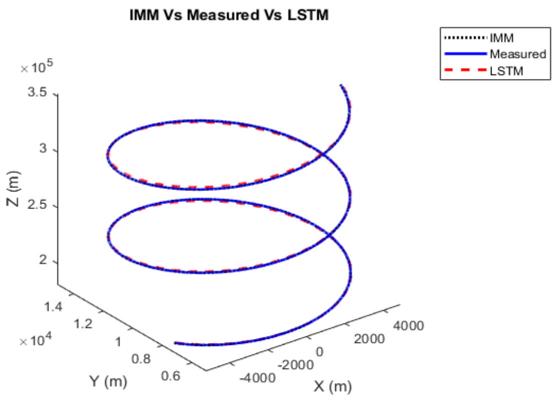

Fig -3: IMM Vs Measured Vs LSTM

Here 'Measured' is the true data, 'IMM' is the data predicted by a 3-model IMM filter and 'LSTM' is the output of our LSTM model. Each time step in x-axis is 100 ms.

4. RESULTS AND DISCUSSIONS

The results show that there is very good agreement between the three datasets.

4.1 Performance on Benchmark Scenarios

Blaire, et. al, had introduced 6 benchmarking trajectories in their paper. These benchmarking problems are now widely used for benchmarking Radar Trackers.

The targets under this benchmark perform as much as seven *gs* of lateral acceleration and two *gs* of longitudinal acceleration. Target range can vary from 20 km to 120 km, while the target elevation angle varies from 2º to 80º. Since only one radar face is used, the bearing of the target will be confined to ±60º.

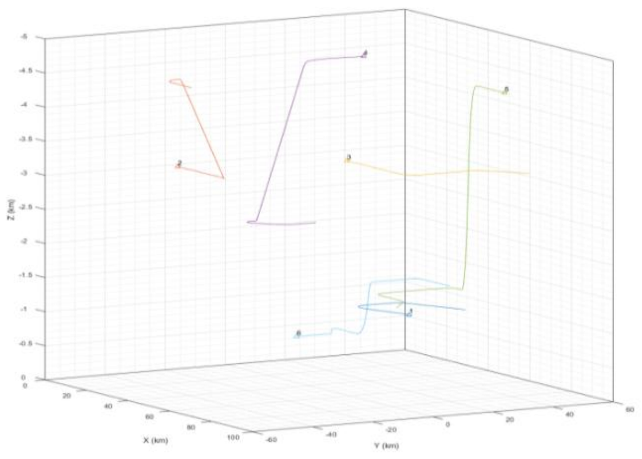

Fig -4: 3D View of the 6 Benchmark Scenarios

LSTM & IMM outputs for few benchmark scenarios are shown below:

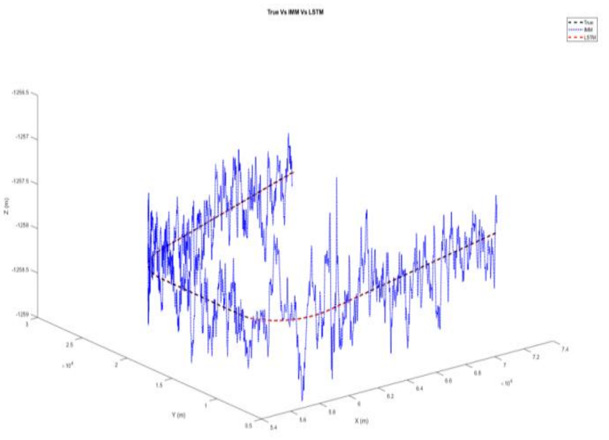

Fig -5: Comparison of LSTM Results with IMM Results & True Scenario for Benchmark 1

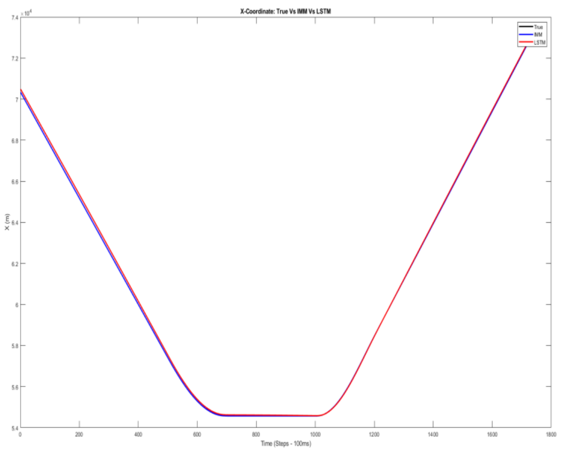

Fig -6: Comparison of LSTM Results with IMM Results & True Scenario for Benchmark 1 (x-coordinate)

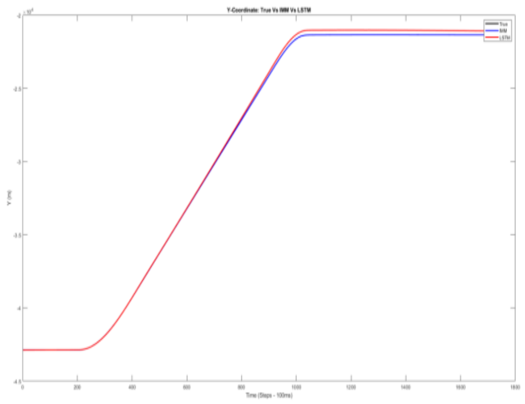

Fig -7: Comparison of LSTM Results with IMM Results & True Scenario for Benchmark 2 (y-coordinate)

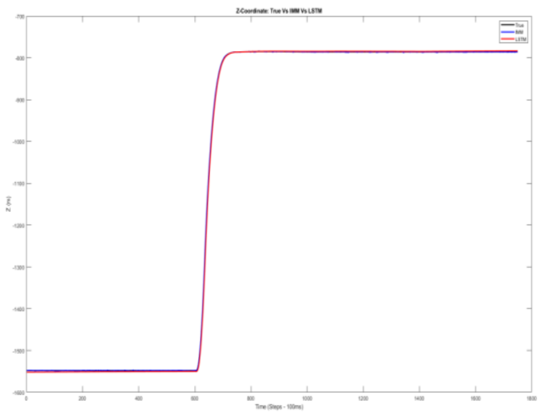

Fig -8: Comparison of LSTM Results with IMM Results & True Scenario for Benchmark 6 (z-coordinate)

It is evident that in all the six cases, LSTM based model provided results as good as a state-of-the-art tracking algorithm like IMM.

5. CONCLUSION

The objective of this work is to explore the feasibility of using deep learning in Radar tracking. The results obtained with the benchmark trajectories are very much promising and are as good as the results of a 3-model IMM Kalman tracker. As no trace of any previous work could be found in the literature, I can proudly say that this is very first study on the usage of AI techniques in radar target tracking. The results can further be improved by adding velocity components, acceleration components and target heading as training inputs. But this will add more complexity to the LSTM model.

REFERENCES

[1] Christopher Olah, Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 27, 2015.

[2] Bakker, Indra den, Python Deep Learning Cookbook, Packt Publishing Ltd, 2017.

[3] Francois chollet, Deep Learning With Python, Manning Publications, 2018.

[4] J.Patterson,A.Gibson, Deep Learning: A Practitioner's Approach, O'Reilly Media, 2017.

[5] Nikhil Buduma, Nicholas Locascio, Fundamentals of Deep Learning, O'Reilly Media, 2017.

[6] Nishant Shukla, Kenneth Fricklas, Machine Learning With TensorFlow, Manning Publications Co, 2017.

[7] Phil Kim, MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence, APRESS, 2017.

[8] W.D. Blair, G. A. Watson, T. Kirubarajan, Y. Bar-Shalom, "Benchmark for Radar Allocation and Tracking in ECM." Aerospace and Electronic Systems IEEE Trans on, vol. 34. no. 4. 1998.

[9] https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/