

Application of Machine Learning (Using Artificial Neural Network & LSTM) in the GPS time Series data

Training Report

Submitted by

AMAN KUMAR

(Reg. No: 22MC0011)

As Part of Master of Science & Technology

In

APPLIED GEOPHYSICS



Department of Applied Geophysics

Indian Institute of Technology

(Indian School of Mines)

DHANBAD, JHARKHAND

UNDER THE GUIDANCE OF:

Dr. Rajesh S. (Scientist-D)

Wadia Institute of Himalayan Geology, Dehradun



10 June 2024 to 10 July 2024

ACKNOWLEDGEMENTS

I hereby express my profound sense of gratitude to my project supervisor **Dr. Rajesh S. (Scientist-D)** who has been helping me throughout the internship period, by supervising me in building the foundation of this work. I express my sincere thanks to him for spending his valuable time for guiding me in every way during the period of my project.

I extend my sincere thanks to **Dr. Kalachand Sain (Director)** Wadia Institute of Himalayan Geology, Dehradun for providing me the opportunity to do my project work at WIHG.

I am thankful to **Prof Sanjit Kumar Pal** Head of the Department of Applied Geophysics, Indian Institute of Technology (Indian school of mines), Dhanbad, Jharkhand for granting the permission to undertake my project at WIHG.

Lastly, I express my hope and enthusiasm for any future opportunities to continue learning and collaborating with **Dr. Rajesh S.** I would eagerly embrace the chance to further expand my knowledge under his mentorship.

I am grateful to lab mate Dr. Sanjay Kumar Verma at the GPS & Gravity lab for their valuable contributions, discussions, criticisms, and enjoyable moments.

I extend my heartfelt appreciation to everyone mentioned above, as well as anyone else who has contributed to my research and learning experience. Their support has been invaluable in the successful completion of this project.

Table of contents

S.No.	Title	Page no
1.	Introduction	5
2.	Overview of GPS (i)Space Segment (ii)Control Segment (iii)User Segment	6-10
3.	Methodology (i) Artificial Neural Network (ii)Activation Function (iii)Architecture of LSTM	11-19
4.	Result & Discussion (i) Case Study-1 (ANN) (ii) Case Study-2 (LSTM) (iii) Normal Interpolation Method (iv) Discussion	20-27
5.	Bibliography	28
6.	Appendix (i)Python code of ANN Algorithm (ii) Python Code of LSTM Algorithm	29-42

List of figures:

Fig 2.1.1 Space segment (Satellite)

Fig 2.1.2 Expandable 24-Slot satellite constellation, as defined in the SPS Performance Standard.

Fig2.2.1 Control Segment global Network

Fig 2.3.1 flow chart of the different segment of GPS

Fig 3.1.1 Raw data format of GPS time series of CHLM station

Fig 3.2.1 Architecture of ANN

Fig 3.2.2 Comparison of Artificial neurons vs biological neurons

Fig 3.2.3 Representation of Backpropagation

Fig 3.3.4.1 Different types of activation function

Fig 3.4.1 Architecture of LSTM Network

Fig 3.4.2 Expanded form of LSTM Network

Fig 4.1.1 Interpolated result of North component by ANN

fig 4.1.2 North component with missing point and predicted point

Fig 4.1.3 Interpolated result of East component by ANN

fig 4.1.4 East component with missing point and predicted point

Fig 4.1.5 Interpolated result of Z component by ANN

fig 4.1.6 Z component with missing point and predicted point

Fig 4.2.1 Interpolated result of N component by LSTM

Fig 4.2.2 North component with missing point and predicted point by LSTM

Fig 4.2.3 Interpolated result of E component by LSTM

Fig 4.2.4 E component with missing point and predicted point by LSTM

Fig 4.2.5 Interpolated result of Z component by LSTM

Fig 4.2.6 Z component with missing point and predicted point by LSTM

Fig 4.3.1 Bar plot of RMSE error by ANN & LSTM

Chapter-1

Introduction

In the field of geospatial analysis, GPS data plays a crucial role in providing accurate positioning information for various applications such as navigation, surveying, and environmental monitoring. However, GPS data often suffers from missing values due to technical issues, atmospheric conditions, or signal obstructions. Traditional interpolation techniques like linear, polynomial, spline, and nearest neighbours have been widely used to address this issue by estimating the missing values based on available data points. While these methods provide satisfactory results, the advent of machine learning offers a more sophisticated approach to handle such challenges.

In this report, we explore the application of machine learning, specifically neural networks and recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) units, for interpolating missing GPS data. These state-of-the-art techniques are particularly well-suited for time series data due to their ability to capture complex patterns and dependencies over time. By leveraging the capabilities of neural networks, we aim to enhance the accuracy of GPS data interpolation by precisely estimating the displacement of the North, East, and Vertical components at each point.

The primary challenge in GPS data interpolation is to accurately estimate the missing values based on the surrounding time series data. The missing data points often occur due to temporary loss of signal or other disruptions, resulting in gaps in the recorded GPS measurements. To address this, we employ neural network models that learn from the previous and subsequent data points, effectively filling in the gaps and providing a continuous and reliable dataset.

This report will detail the methodology used for implementing neural network and LSTM models for GPS data interpolation, compare their performance with traditional interpolation techniques, and discuss the advantages and limitations of using machine learning for this purpose. Through this approach, we aim to demonstrate the potential of machine learning in enhancing the accuracy and reliability of GPS data, paving the way for more robust geospatial analyses and applications.

Chapter-2

2. Overview of GPS

Global Positioning System (GPS)

GPS is the fully functional radiometric space-based Global Navigation Satellite System (GNSS) developed by the U.S. Department of Defence for military applications in the 1970's. In 1983, it became available for civilian usage. Since then, its usage has expanded to include estimation of accurate, real-time three-dimensional position and velocity in a common reference system. In addition, it is broadly employed for achieving high precision timing for communication purposes (Misra & Enge 2006). The GPS system is composed of three distinct segments such as control segment, space segment, and user segment, described in detail in the following sections.

2.1 Space Segment

The GPS space segment consists of a constellation of satellites transmitting radio signals to users.

The United States is committed to maintaining the availability of at least 24 operational GPS satellites, 95% of the time.

To ensure this commitment, the U.S. Space Force has been flying 31 operational GPS satellites for well over a decade.

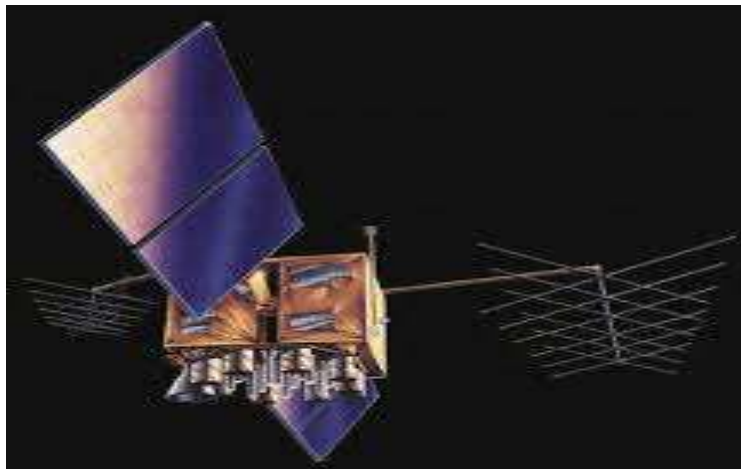


Fig 2.1.1 Space segment (Satellite)

Constellation Arrangement

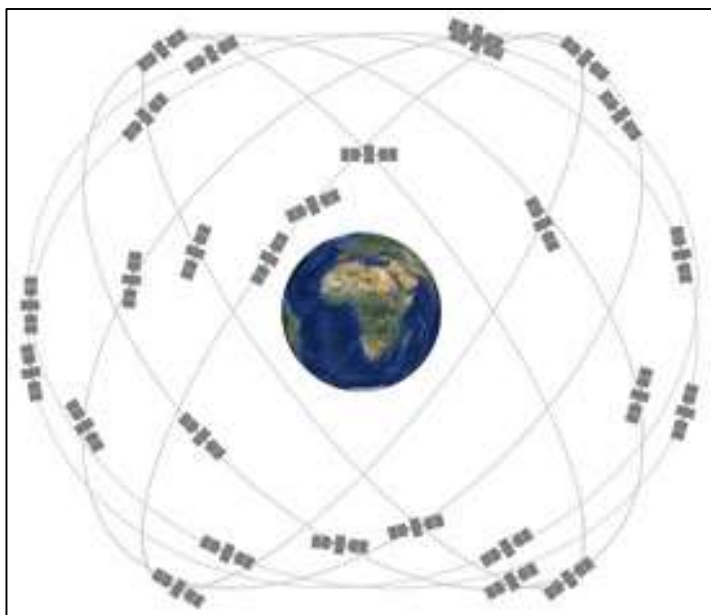


Fig 2.1.2 Expandable 24-Slot satellite constellation, as defined in the SPS Performance Standard.

GPS satellites fly in medium Earth orbit (MEO) at an altitude of approximately 20,200 km (12,550 miles). Each satellite circles the Earth twice a day.

The satellites in the GPS constellation are arranged into six equally-spaced orbital planes surrounding the Earth. Each plane contains four "slots" occupied by baseline satellites. This 24-slot arrangement ensures users can view at least four satellites from virtually any point on the planet.

The Space Force normally flies more than 24 GPS satellites to maintain coverage whenever the baseline satellites are serviced or decommissioned. The extra satellites may increase GPS performance but are not considered part of the core constellation.

In June 2011, the Air Force successfully completed a GPS constellation expansion known as the "Expandable 24" configuration. Three of the 24 slots were expanded, and six satellites were repositioned, so that three of the extra satellites became part of the constellation baseline. As a result, GPS now effectively operates as a 27-slot constellation with improved coverage in most parts of the world. [**LEARN MORE AT SPACEFORCE.MIL**](#)

Technical details about the orbits, coverage, and performance of the GPS satellite constellation are documented in the GPS Performance Standards. [**VIEW DOCUMENTS**](#)

2.2 Control Segment

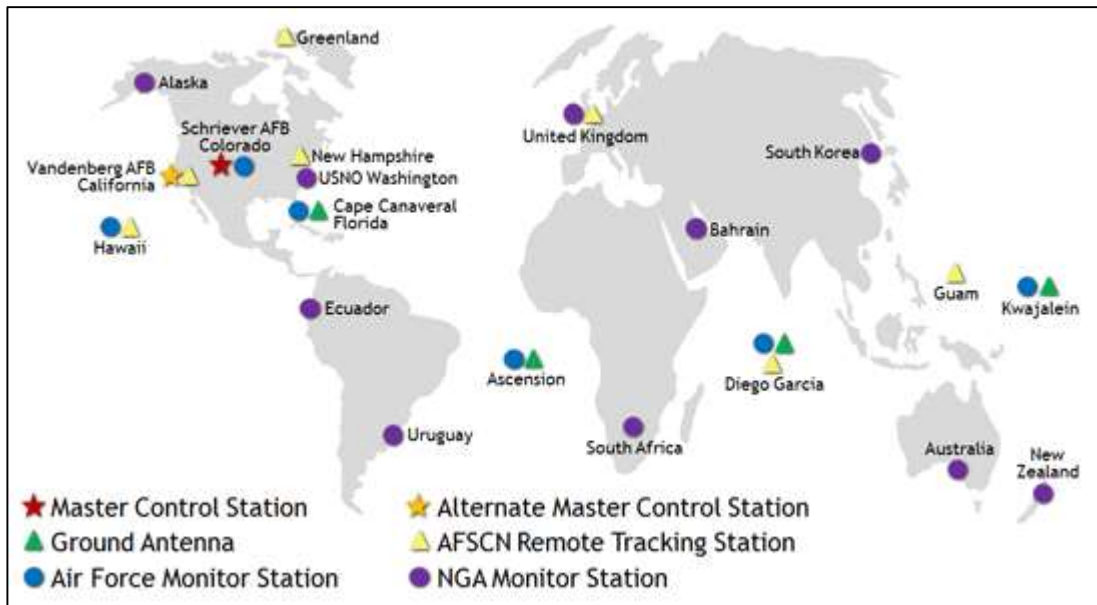
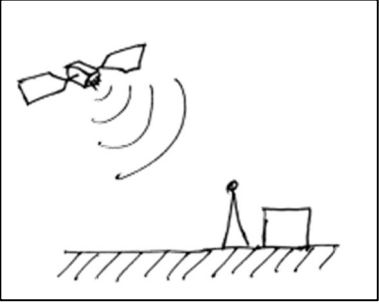
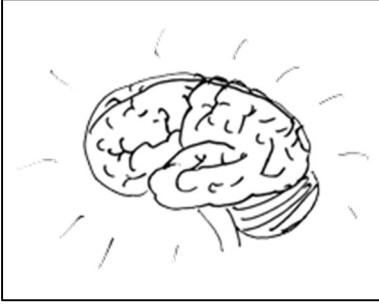



Fig2.2.1 Control Segment global Network

The GPS control segment consists of a global network of ground facilities that track the GPS satellites, monitor their transmissions, perform analyses, and send commands and data to the constellation.

The current Operational Control Segment (OCS) includes a master control station, an alternate master control station, 11 command and control antennas, and 16 monitoring sites. The locations of these facilities are shown in the map above. [VIEW AS PDF \(111 KB\)](#)

2.2(a) Control Segment Elements

		
<p>Monitor Stations</p> <ul style="list-style-type: none"> ▪ Track GPS satellites as they pass overhead ▪ Collect navigation signals, range/carrier measurements, and atmospheric data ▪ Feed observations to the master control station ▪ Utilize sophisticated GPS receivers ▪ Provide global coverage via 16 sites: 6 from the Air Force plus 10 from NGA 	<p>Master Control Station</p> <ul style="list-style-type: none"> ▪ Provides command and control of the GPS constellation ▪ Uses global monitor station data to compute the precise locations of the satellites ▪ Generates navigation messages for upload to the satellites ▪ Monitors satellite broadcasts and system integrity to ensure constellation health and accuracy ▪ Performs satellite maintenance and anomaly resolution, including repositioning satellites to maintain optimal constellation ▪ Currently uses separate systems (AEP & LADO) to control operational and non-operational satellites ▪ Backed up by a fully operational alternate master control station 	<p>Ground Antennas</p> <ul style="list-style-type: none"> ▪ Send commands, navigation data uploads, and processor program loads to the satellites ▪ Collect telemetry ▪ Communicate via S-band and perform S-band ranging to provide anomaly resolution and early orbit support ▪ Consist of 4 dedicated GPS ground antennas plus 7 Air Force Satellite Control Network (AFSCN) remote tracking stations

2.3 The User Segment

The user segment is composed of receiver technologies for computing local position in the Earth-centered fixed WGS84 reference system and for calculating the receiver clock offset. The GPS applications in the civilian community have expanded as a result of the decreasing cost of accessibility to GPS technology in the recent years.

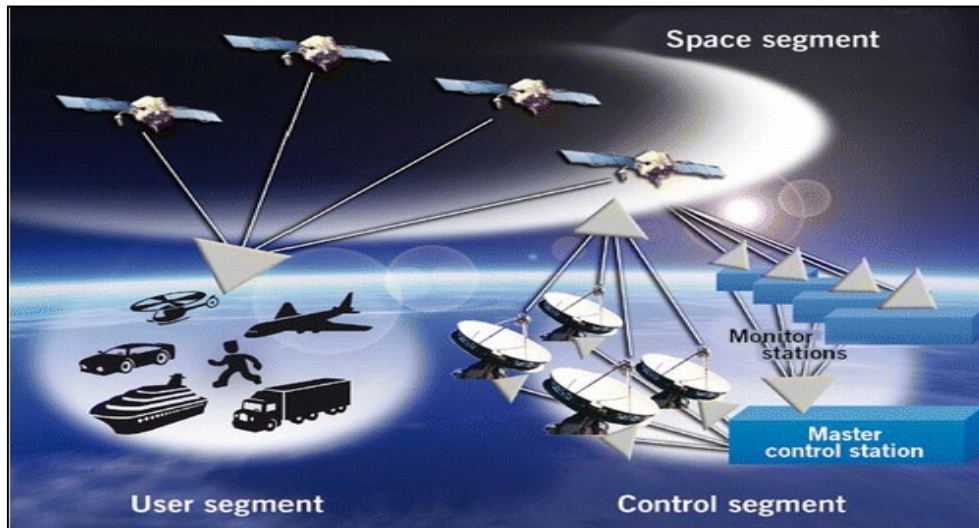


Fig 2.3.1 flow chart of the different segment of GPS

2.3(a) GPS Applications

Like the Internet, GPS is an essential element of the global information infrastructure. The free, open, and dependable nature of GPS has led to the development of hundreds of applications affecting every aspect of modern life. GPS technology is now in everything from cell phones and wristwatches to bulldozers, shipping containers, and ATM's.

GPS boosts productivity across a wide swath of the economy, to include farming, construction, mining, surveying, package delivery, and logistical supply chain management. Major communications networks, banking systems, financial markets, and power grids depend heavily on GPS for precise time synchronization. Some wireless services cannot operate without it.

GPS saves lives by preventing transportation accidents, aiding search and rescue efforts, and speeding the delivery of emergency services and disaster relief. GPS is vital to the Next Generation Air Transportation System (NextGen) that will enhance flight safety while increasing airspace capacity. GPS also advances scientific aims such as weather forecasting, earthquake monitoring, and environmental protection.

Finally, GPS remains critical to U.S. national security, and its applications are integrated into virtually every facet of U.S. military operations. Nearly all new military assets -- from vehicles to munitions -- come equipped with GPS.

Chapter-3

3. Methodology

3.1 Raw Data Format

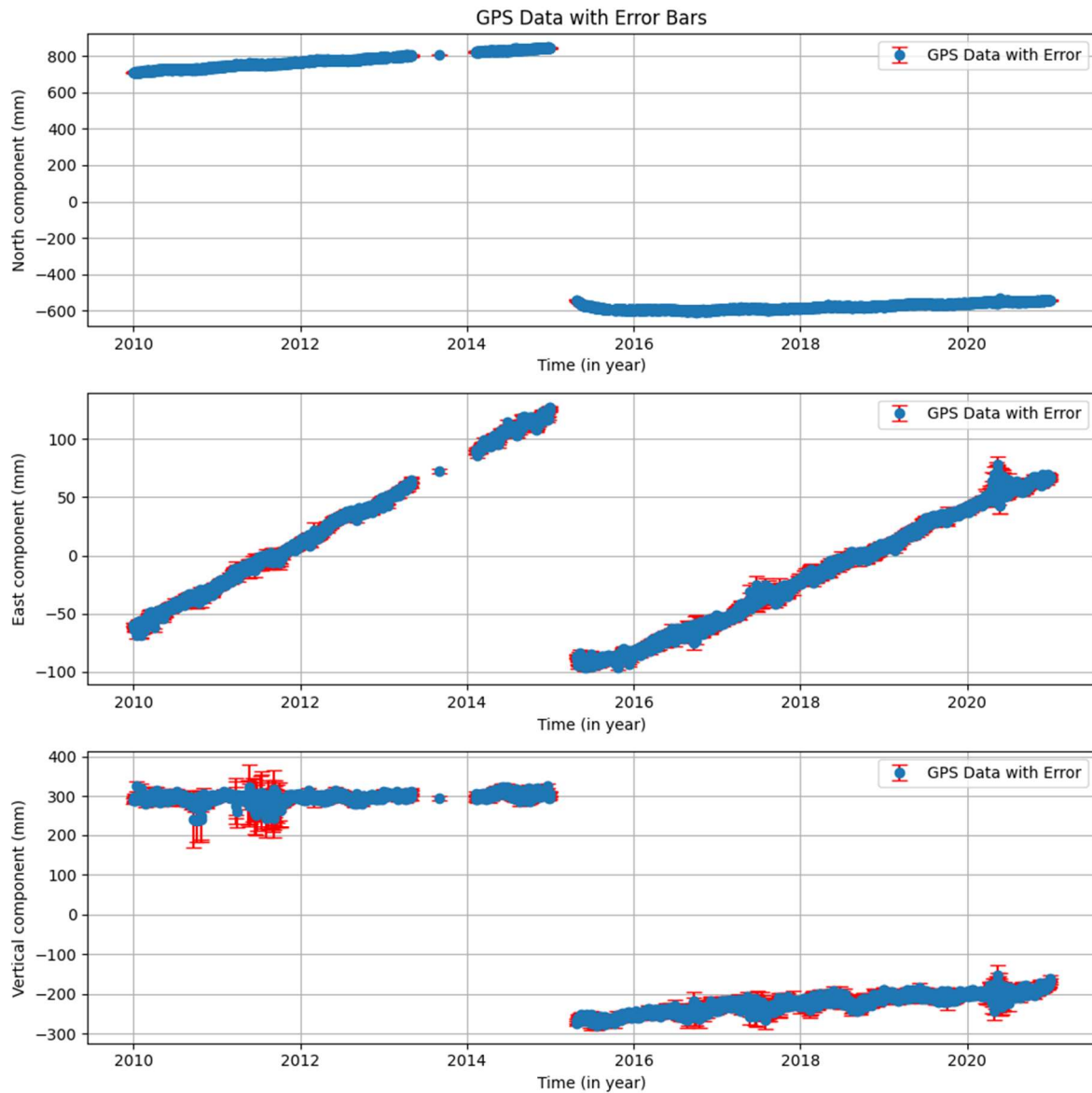


Fig 3.1.1 Raw data format of GPS time series of CHLM station

We are clearly seeing that there is data gap in each component. We have to fill that data gap accurately. We know Many interpolation Technique like linear interpolation , polynomial interpolation , spline interpolation and Nearest neighbours interpolation But here we want apply machine learning Technique Artificial Neural Network and Recurrent Neural Network (LSTM). And we will see the result. First of all Lets see the procedure of our algorithm.

3.2 Artificial Neural Network

Artificial Neural Networks contain artificial neurons which are called **units**. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyse or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

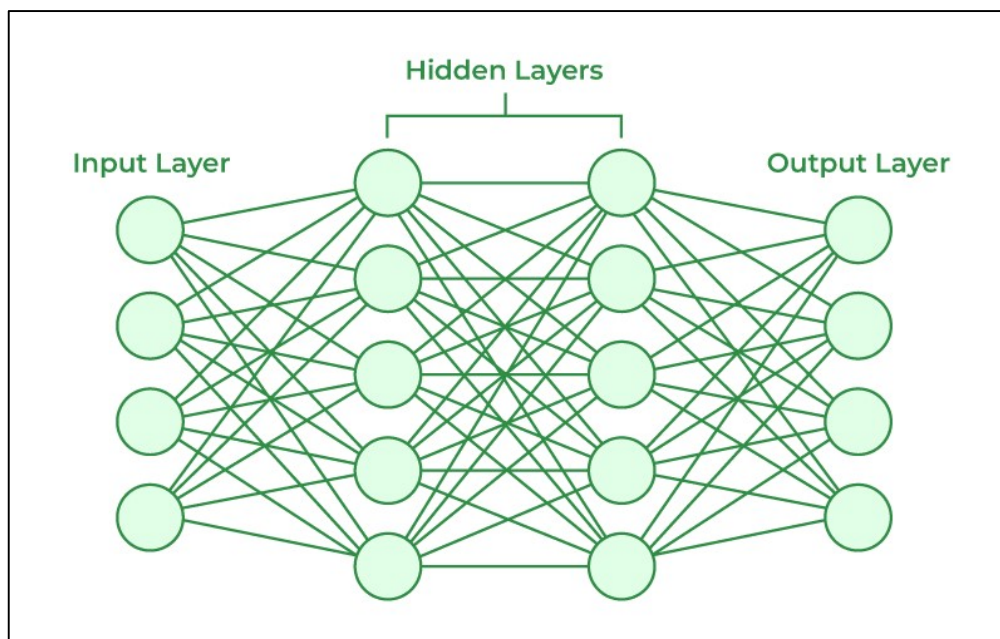


Fig 3.2.1 Architecture of ANN

The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

3.2(a) Artificial neurons vs Biological neurons

Activation: In biological neurons, activation is the firing rate of the neuron which happens when the impulses are strong enough to reach the threshold. In artificial neural networks, A mathematical function known as an activation function maps the input to the output, and executes activations.

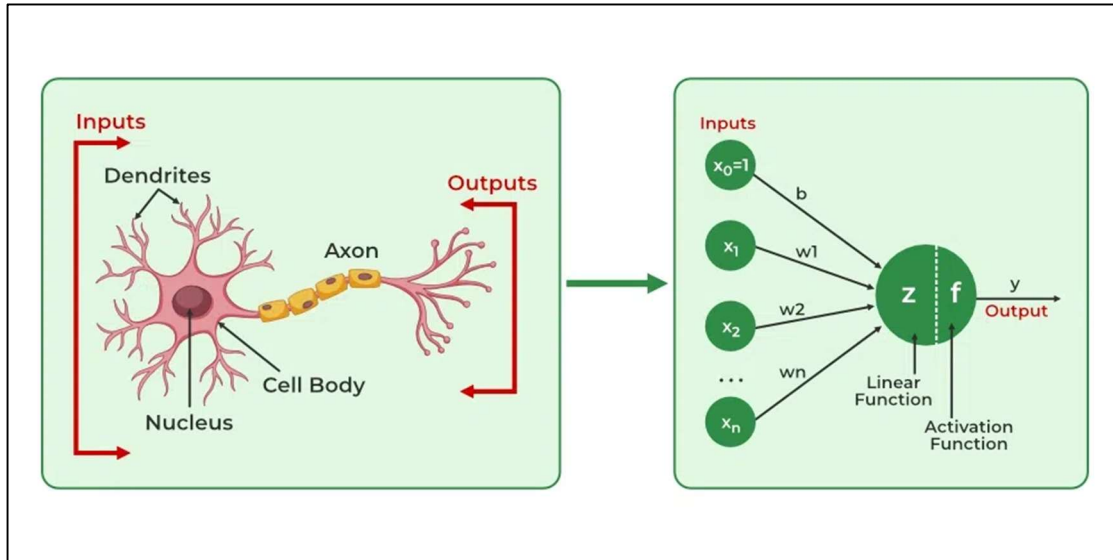


Fig 3.2.2 Comparison of Artificial neurons vs biological neurons

3.2(b) How do Artificial Neural Networks learn?

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then [back-propagation](#) is used to adjust whatever it has learned during training. [Backpropagation](#) is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates.

3.2(c) What are the types of Artificial Neural Networks?

- **Feedforward Neural Network**: The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist. So the feedforward neural network has a front-propagated wave only and usually does not have backpropagation.
- **Convolutional Neural Network**: A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit. But a CNN has one or more than one convolutional layer that uses a convolution operation on the input and then passes the result obtained in the form of output to the next layer. CNN has applications in speech and image processing which is particularly useful in computer vision.
- **Radial basis function Neural Network**: Radial basis functions are those functions that consider the distance of a point concerning the centre. RBF functions have two layers. In the first layer, the input is mapped into all the Radial basis functions in the hidden layer and then the output layer computes the output in the next step. Radial basis function nets are normally used to model the data that represents any underlying trend or function.
- **Recurrent Neural Network**: The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.

3.2(d) What is backpropagation?

- In machine learning, backpropagation is an effective algorithm used to train artificial neural networks, especially in feed-forward neural networks.
- Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as [gradient descent](#) or [stochastic gradient descent](#).

- Computing the gradient in the backpropagation algorithm helps to minimize the [cost function](#) and it can be implemented by using the mathematical rule called chain rule from calculus to navigate through complex layers of the neural network.

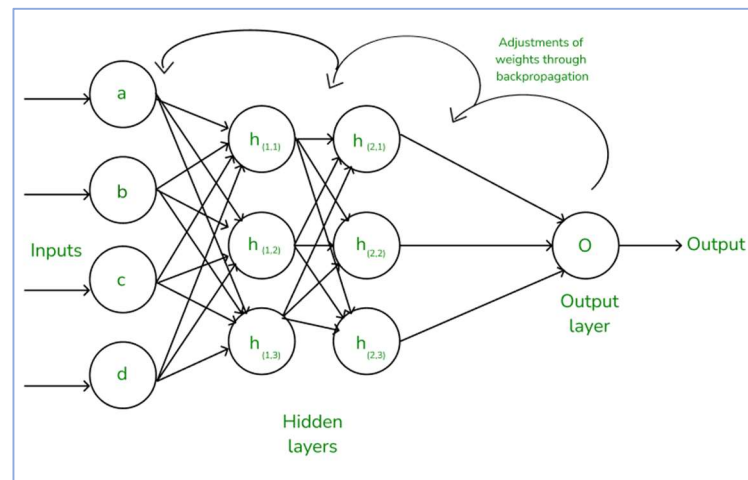


Fig 3.2.3 Representation of Backpropagation

3.3 Activation Function

3.3.1 What is Activation Function?

It's just a thing function that you use to get the output of node. It is also known as **Transfer Function**.

3.3.2 Why we use Activation functions with Neural Networks?

It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

The Activation Functions can be basically divided into 2 types-

1. Linear Activation Function
2. Non-linear Activation Functions

3.3.3 Why We use Non-linear activation Function?

Non-Linearly Separable Case:

Consider the XOR problem mentioned

(0,0) -> 0

(0,1) -> 1

(1,0) -> 1

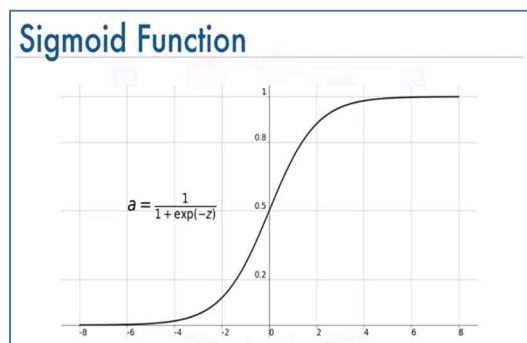
(1,1) -> 0

A neural network without non-linear activation functions cannot solve this problem because it cannot learn the necessary complex boundary. However, with non-linear activation functions, the network can learn to combine features in a way that models the XOR relationship.

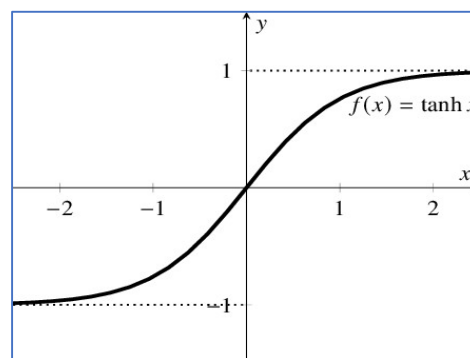
□ **Linear Activation Function:** Suitable for linearly separable data. The model is simple and cannot capture complex patterns.

□ **Non-Linear Activation Function:** Necessary for non-linearly separable data. They enable the neural network to capture complex relationships and patterns in the data by introducing non-linearity.

3.3.4 Different types of Activation function:



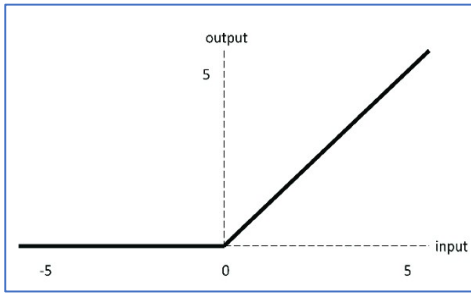
Sigmoid function



Tanh Function

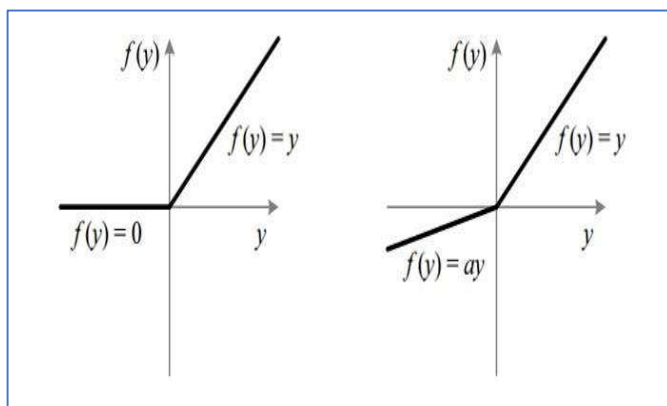
$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

ReLu Function



$$\sigma(x) = \begin{cases} \max(0, x) & , x \geq 0 \\ 0 & , x < 0 \end{cases}$$

Leaky Relu Vs ReLu



$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

Fig 3.3.4.1 Different types of activation function

3.4 Architecture of LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. It was proposed in 1997 by **Sepp Hochreiter** and **Jurgen schmidhuber**. Unlike standard feed-forward neural networks, LSTM has feedback connections. It can process not only single data points (such as images) but also entire sequences of data (such as speech or video).

A general **LSTM** unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals, and three gates regulate the flow of information into and out of the cell. LSTM is well-suited to classify, process, and predict the time series given of unknown duration.

Long Short- Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory.

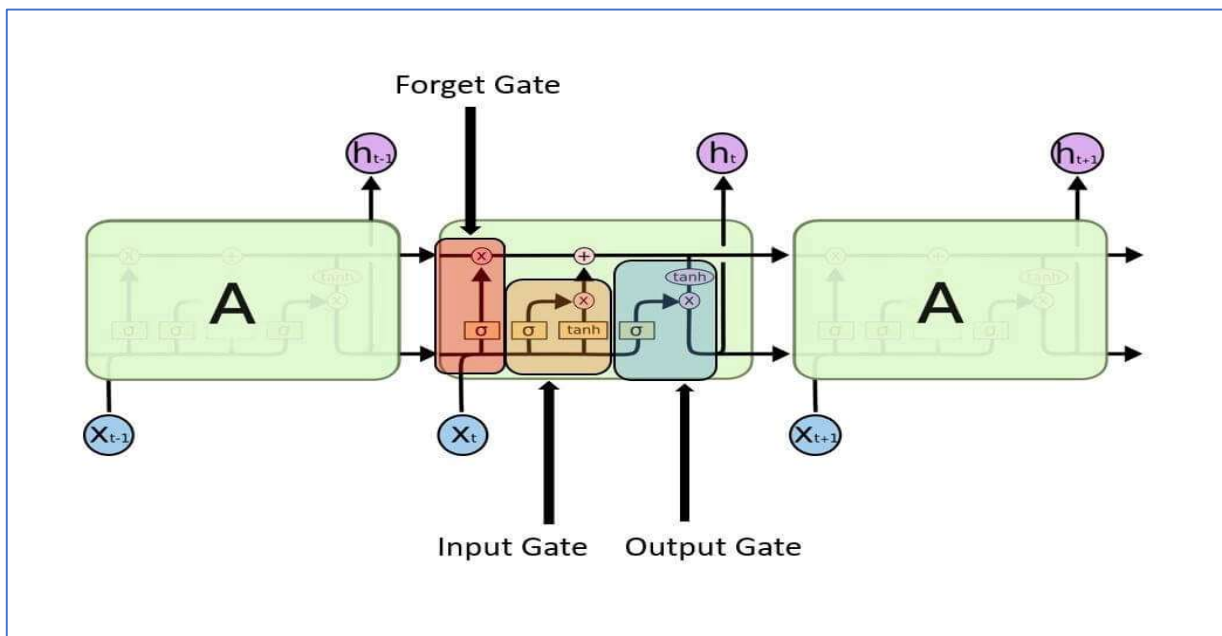


Fig 3.4.1 Architecture of LSTM Network

1. Input gate- It discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through 0 or 1. And **tanh** function gives weightage to the values which are passed, deciding their level of importance ranging from -1 to 1.

$$i_t = \sigma(W_i \cdot [h_t - 1, x_t] + b_i)$$

$$C_t = \tanh(W_c \cdot [h_t - 1, x_t] + b_c)$$

2. Forget gate- It discover the details to be discarded from the block. A sigmoid function decides it. It looks at the previous state (**ht-1**) and the content input (Xt) and outputs a number between 0(omit this) and 1(keep this) for each number in the cell state **Ct-1**.

$$f_t = \sigma(W_f \cdot [h_t - 1, x_t] + b_f)$$

3. Output gate- The input and the memory of the block are used to decide the output. Sigmoid function decides which values to let through 0 or 1. And **tanh** function decides which values to let through 0, 1. And tanh function gives weightage to the values which are passed, deciding their level of importance ranging from -1 to 1 and multiplied with an output of **sigmoid**.

$$O_t = \sigma(W_o[h_t - 1, x_t] + b_o)$$

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

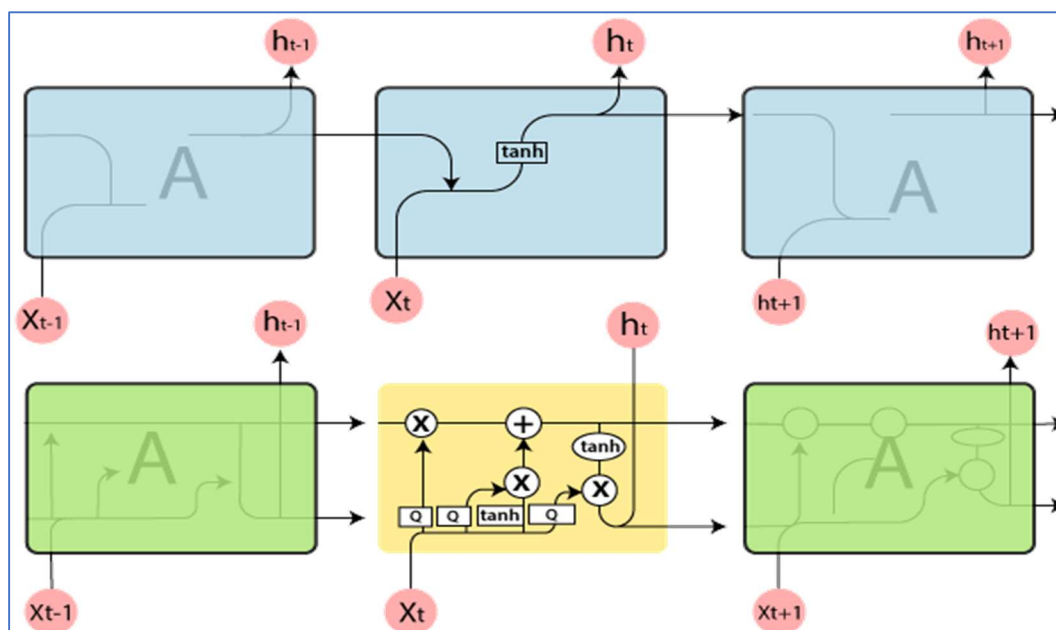


Fig 3.4.2 Expanded form of LSTM Network

Chapter-4

4.Result & Discussion:

4.1 Case Study-1

By applying Backpropagated neural network algorithm we get following result.

Firstly showing the result for North component

4.1(a) North component:

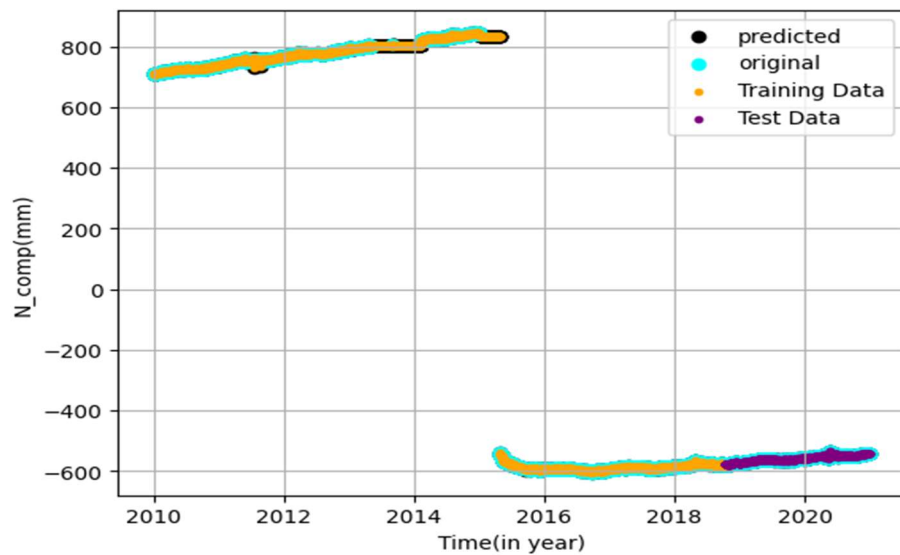


fig 4.1.1 Interpolated result of North component by ANN



fig 4.1.2 North component with missing point and predicted point

4.1(b) E-component:

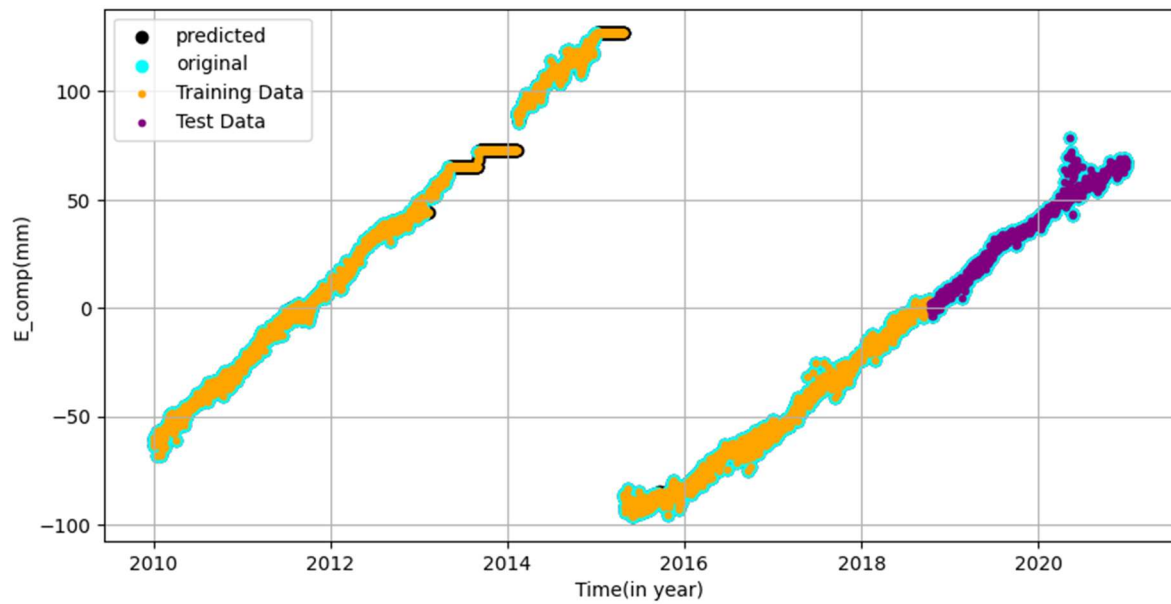


Fig 4.1.3 Interpolated value of East component by ANN

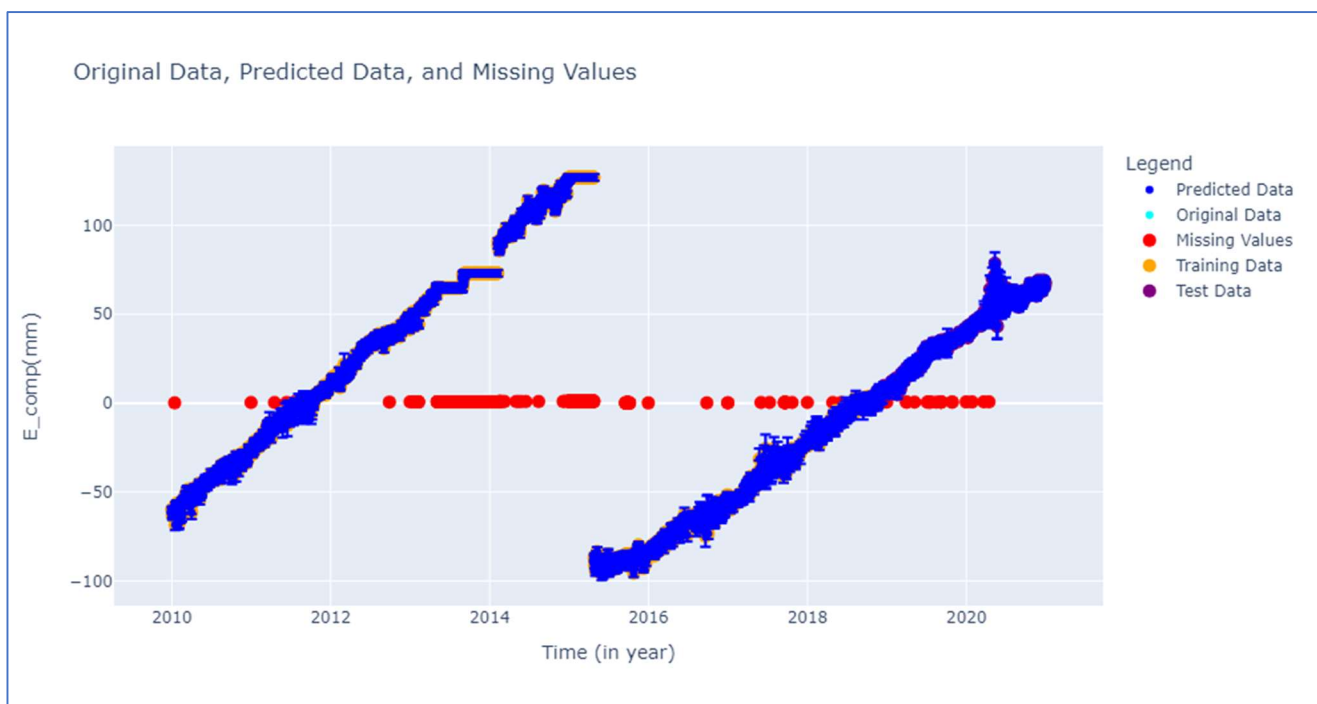


Fig 4.1.4 East component with missing point and predicted point

4.1(c) Z-component:

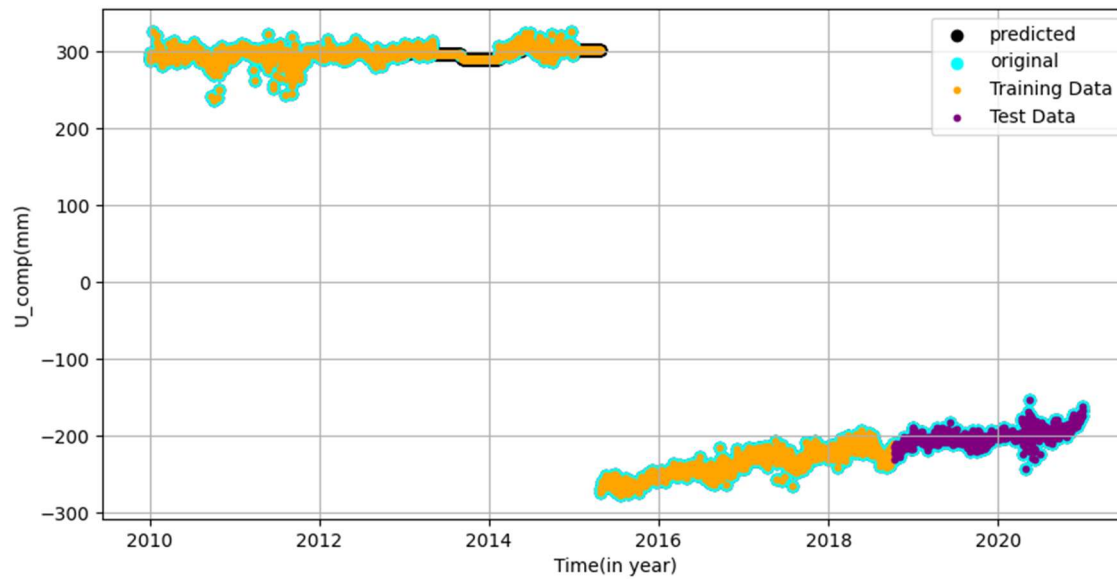


Fig 4.1.5 Interpolated value of Vertical component by ANN

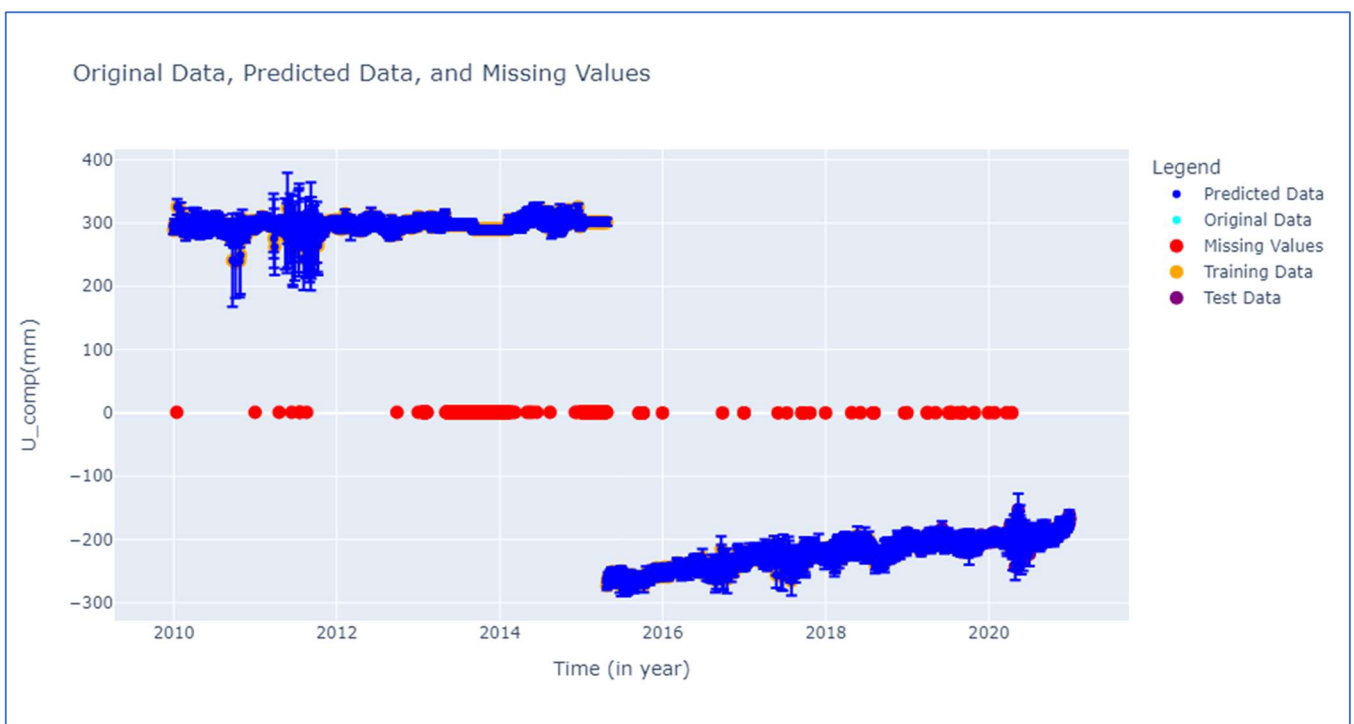


Fig 4.1.6 Vertical component with missing point and predicted point

4.2 Case Study-2

By applying Recurrent Neural Network with LSTM we get following result.

Here we are showing the result component wise-

4.2(a) N-component:

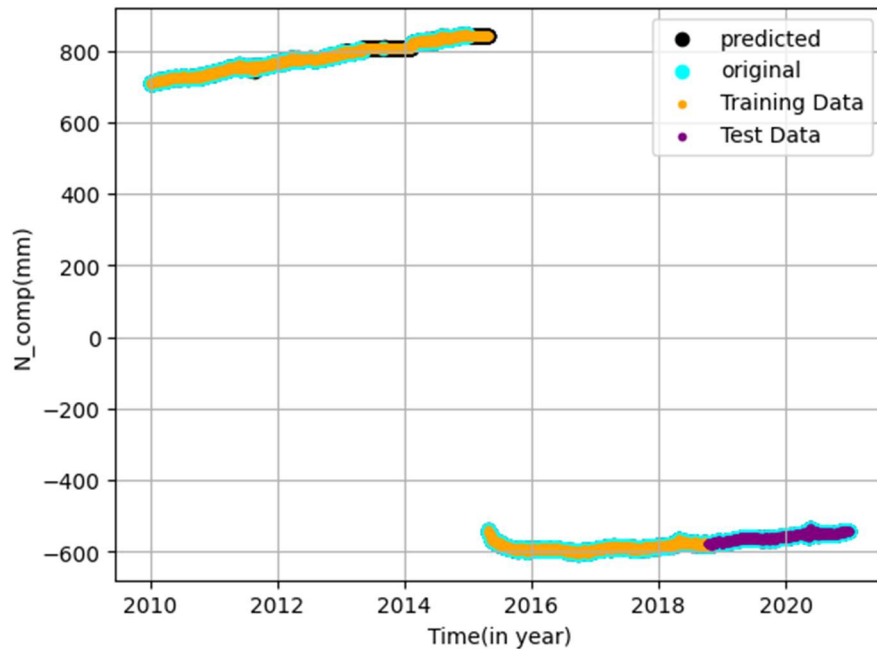


Fig 4.2.1 Interpolated result of N component by LSTM

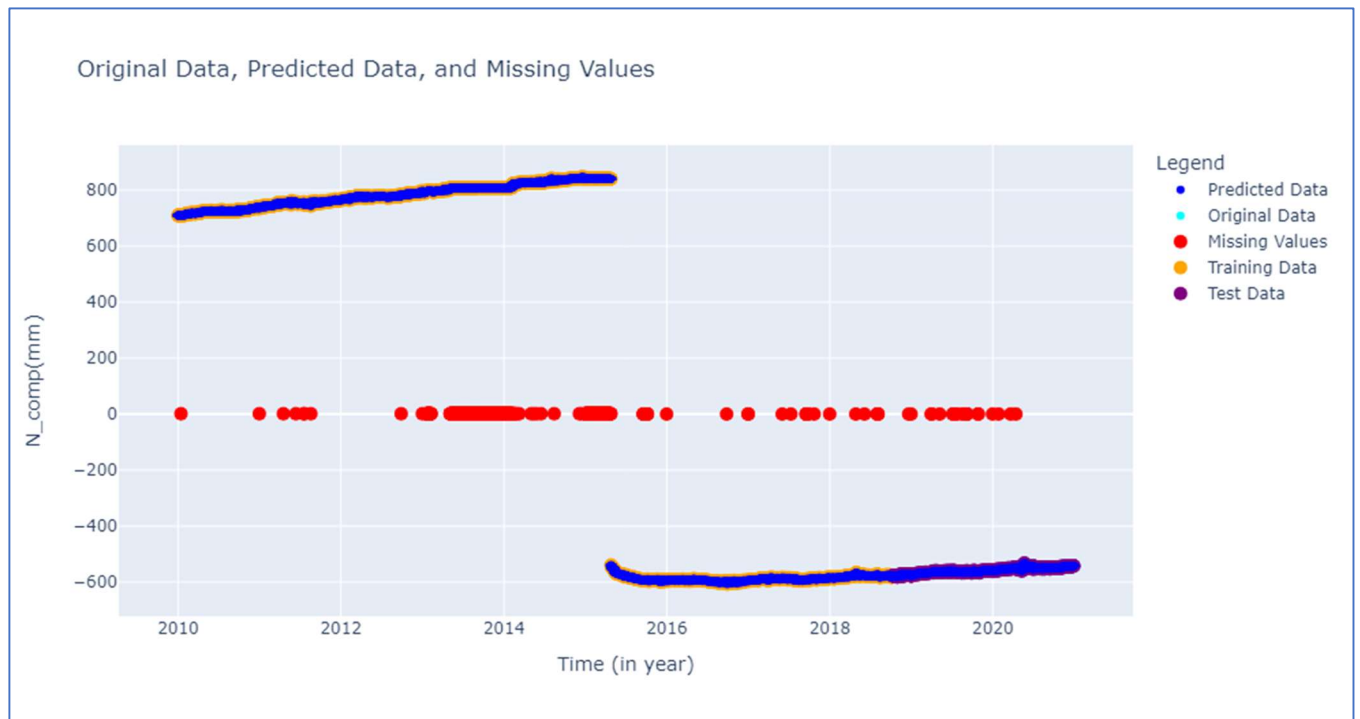


Fig 4.2.2 North component with missing point and predicted point by LSTM

4.2(b) E-component:

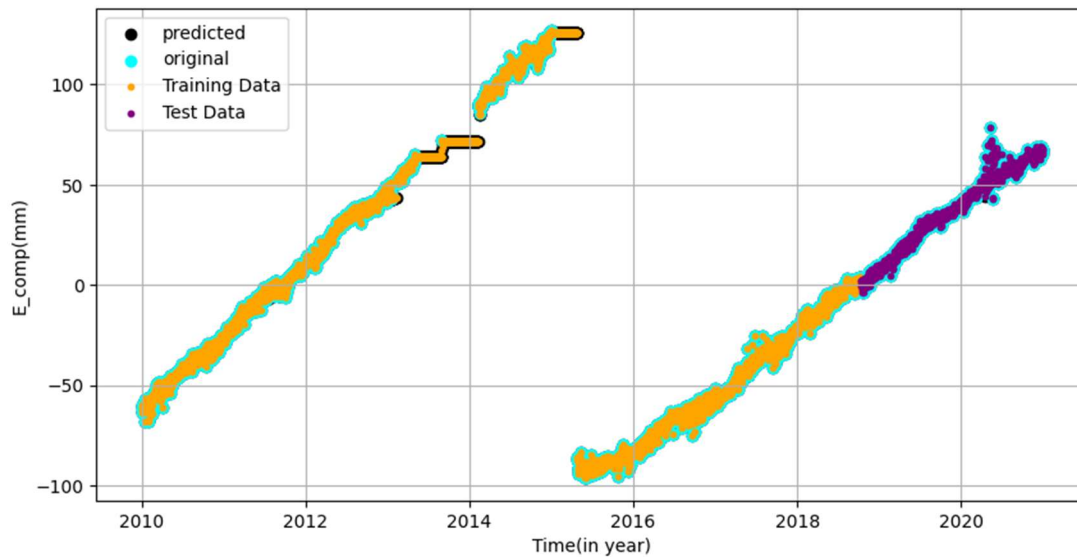


Fig 4.2.3 Interpolated result of E component by LSTM

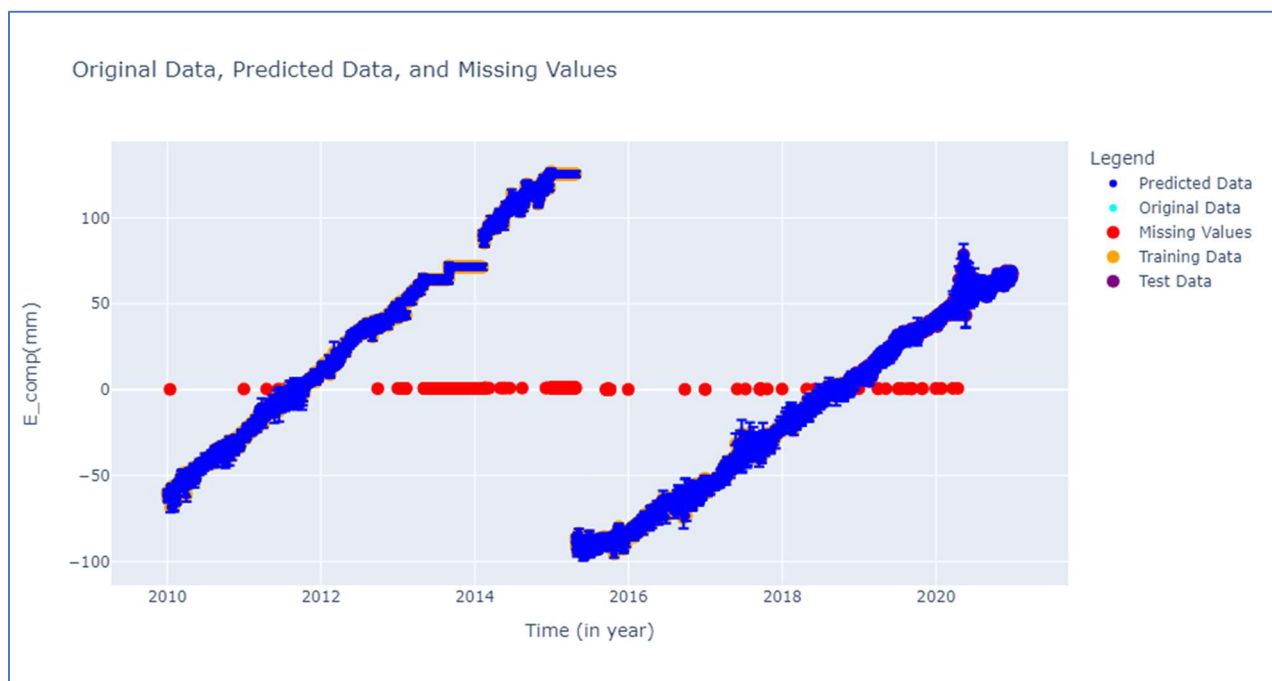


Fig 4.2.4 East component with missing point and predicted point by LSTM

4.2(c) Z-component:

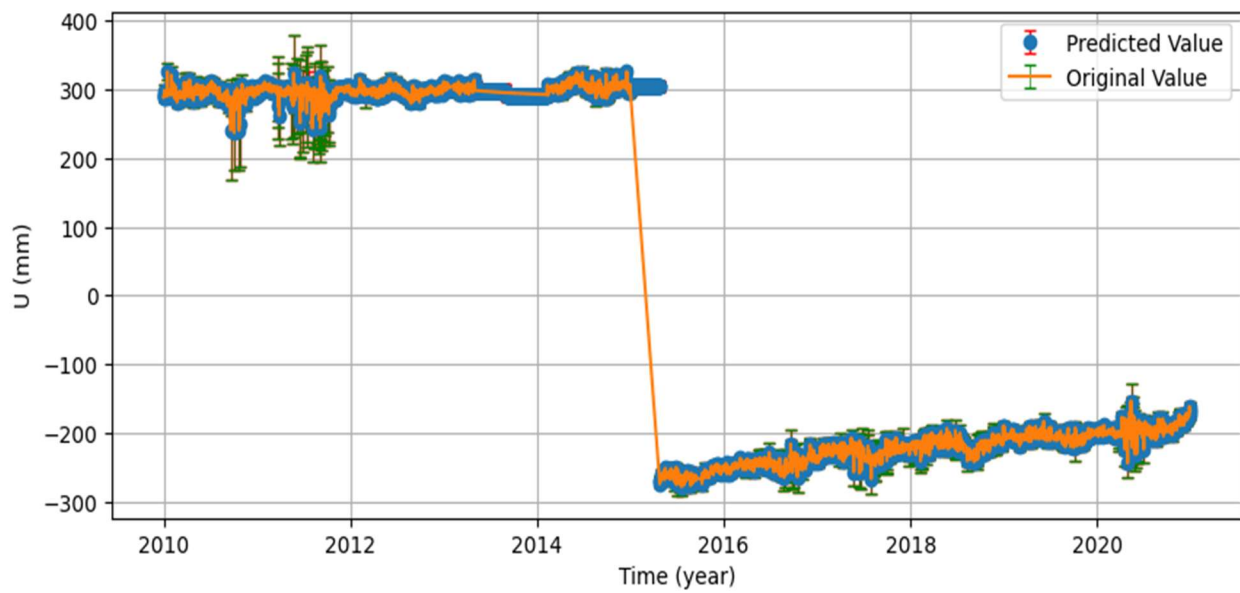


Fig 4.2.5 Interpolated result of E component by LSTM

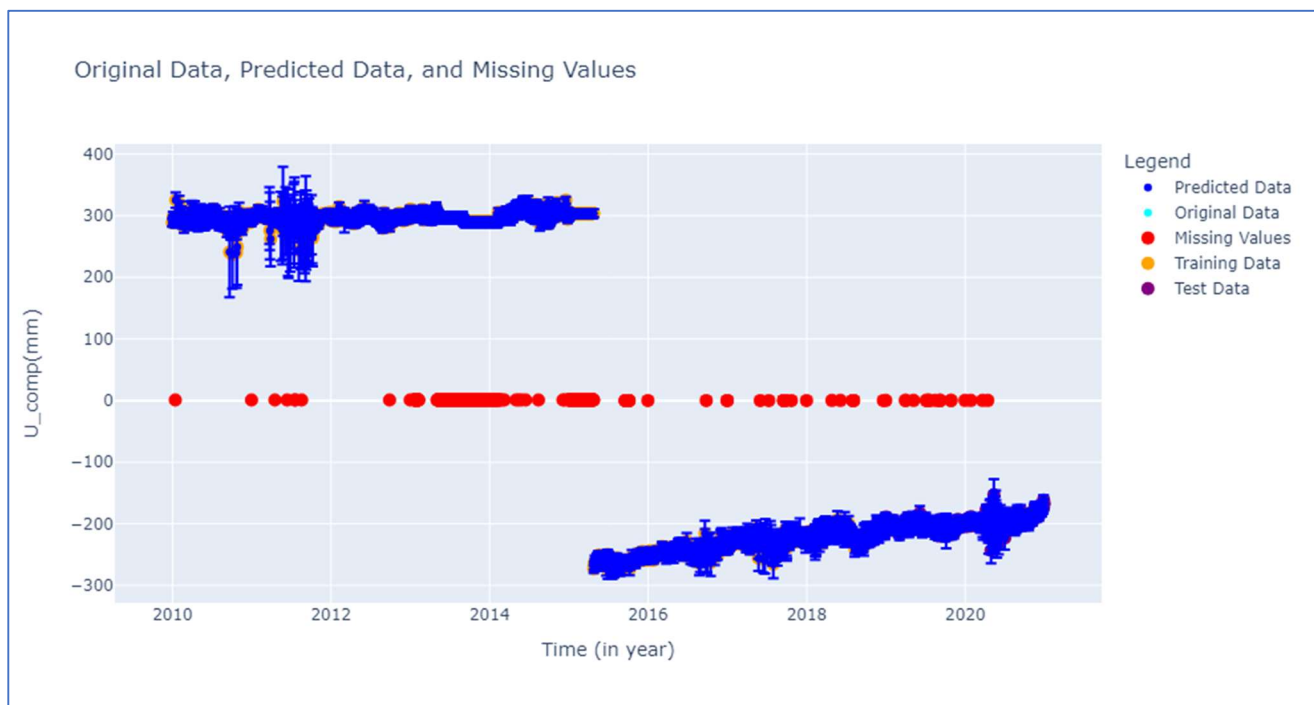


Fig 4.2.6 Z component with missing point and predicted point by LSTM

4.3 Comparison of ANN & LSTM:

We are plotting the RMSE error value for both algorithm for every parameter.

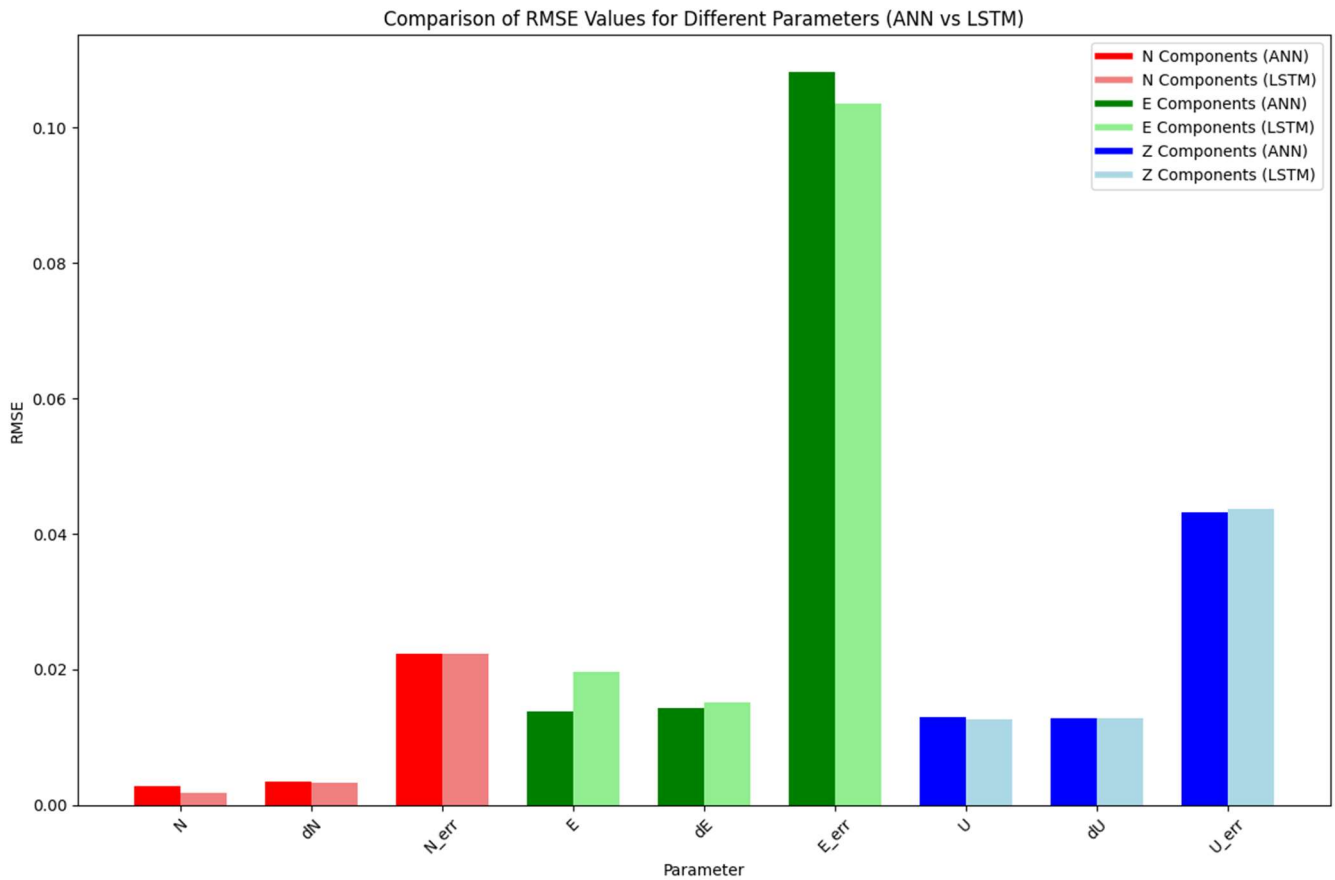


Fig 4.3.1 Bar plot of RMSE error by ANN & LSTM

- From the above bar plot we observe that LSTM algorithm is slightly better than ANN.
- Both Algorithm have almost same RMSE error in between the range of $\pm(0.0018 - 0.1082)$ mm
- So that's a good result we can working on it and improve it more.

4.4 Normal Interpolation Technique RMSE Error Component Wise

S.NO	Normal Interpolation methods	RMSE_N (+- mm)	RMSE_E (+- mm)	RMSE_Z (+-mm)
1.	KNN	1.7243	2.1339	6.6485
2.	RBF	2.2367	2.9099	9.6047
3.	Polynomial	2.2581	2.8602	9.4429
4.	Linear	2.4327	1.8086	0.4051

Table 4.4.1 Comparison of different normal interpolation method

4.5 Discussion:

In our analysis, we observed that among standard interpolation techniques, the K-Nearest Neighbours (KNN) method consistently yielded the lowest RMSE error, indicating superior performance in terms of accuracy. While linear interpolation is generally not effective across all datasets, it exhibited remarkable goodness-of-fit for our specific data, particularly with the Z-component, which had the lowest RMSE error using this method. However, these traditional interpolation methods fall short when compared to state-of-the-art techniques such as Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) units. Recognizing the limitations of classical methods, we extended our investigation to these advanced machine learning models.

Our preliminary work focused on one station, demonstrating that machine learning models, which are inherently data-driven, necessitate careful tuning of parameters and hyperparameters to optimize performance. The initial results underscore the potential of applying ANN and RNN models to achieve more robust and stable predictions. This approach can be scaled to multiple stations to comprehensively evaluate and compare the results across different locations.

Our ongoing effort aims to develop a robust and stable model by exploring and implementing various algorithms, including boosting and bagging techniques. These ensemble methods are expected to enhance the model's performance and stability. Despite the constrained timeline of one month for this preliminary work, our objective is to apply machine learning to this field, leveraging its potential to produce highly accurate and reliable results. The ultimate goal is to identify the most effective and robust method for our specific application, thereby advancing the state of data interpolation and prediction in our domain. This exploration will lay the groundwork for future research and application, potentially transforming how we handle and analyse such data.

Bibliography:

1. YILMAZ, M., GULLU, M., 2014. A comparative study for the estimation of geodetic point velocity by artificial neural networks. J Earth Syst Sci 123, 791–808. <https://doi.org/10.1007/s12040-014-0411-6>,
2. <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>
3. <https://www.gps.gov/systems/gps/space/>
4. Alinia, H. S. 2017. New GPS Time Series Analysis and a Simplified Model to Compute an Accurate Seasonal Amplitude of Tropospheric Delay.
5. <https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>
6. <https://suryansh-data.medium.com/activation-functions-in-neural-networks-45a8b035b14a>
7. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning

Appendix

Python code of ANN Algorithm of North component:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv(r"C:\Users\AmanKumar\Desktop\Wadiainternship\CHLM_ful.txt",delim_whitespace=True)
data['YYYYMMDD'] = pd.to_datetime(data['YYYYMMDD'], format='%Y%m%d')
data=data.set_index(data['YYYYMMDD'])
data = data.rename(columns={'+-.': 'N_err', '+-.1': 'E_err', '+-.2': 'U_err'})
data=data.asfreq(pd.infer_freq(data.index))
df=data.drop(['DecYr','YYYYMMDD','HHMNSC','MJD','F','F.1','F.2','dE','dU','E','E_err','U','U_err'],axis=1)
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Load your DataFrame (assuming df is already loaded)
# df = pd.read_csv('your_data.csv')

# Sort DataFrame by index (if not already sorted)
df.sort_index(inplace=True)

# Identify missing values
missing_mask = df['N'].isna()

# Fill missing values temporarily for normalization
df_filled = df.fillna(method='ffill').fillna(method='bfill')

# Normalize the data
scaler = MinMaxScaler()
df[['N', 'dN', 'N_err']] = scaler.fit_transform(df_filled[['N', 'dN', 'N_err']])

# Create sequences
def create_sequences(data, seq_length):
```

```

xs, ys = [], []

for i in range(seq_length, len(data) - 1):
    x = data[i-seq_length:i]
    y = data[i]
    xs.append(x)
    ys.append(y)

return np.array(xs), np.array(ys)

# Define sequence length
SEQ_LENGTH = int(input("Enter a sequence length:"))

# Create sequences
data_modified = df[['N', 'dN', 'N_err']].values
X, y = create_sequences(data_modified, SEQ_LENGTH)

# Filter out any sequences with NaN values
valid_indices = ~np.isnan(y).any(axis=1)
X, y = X[valid_indices], y[valid_indices]

# Reshape X for ANN input
X = X.reshape((X.shape[0], X.shape[1] * X.shape[2]))

# Split into training and test sets
split_idx = int(len(X) * 0.8)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

# Build the model
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(SEQ_LENGTH * 3,)))

```

```

#model.add(BatchNormalization())
#model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
#model.add(BatchNormalization())
#model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
#model.add(BatchNormalization())
#model.add(Dropout(0.2))
model.add(Dense(3)) # Predicting N, dN, N_err

model.compile(optimizer='adam', loss='mse')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

model_losses = pd.DataFrame(model.history.history)
model_losses.plot()

import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

# Assuming X_test and y_test are already defined

# Make predictions
y_pred = model.predict(X_test)

# Compute RMSE for each target variable (N, dN, N_err)
rmse_N = sqrt(mean_squared_error(y_test[:, 0], y_pred[:, 0]))
rmse_dN = sqrt(mean_squared_error(y_test[:, 1], y_pred[:, 1]))
rmse_N_err = sqrt(mean_squared_error(y_test[:, 2], y_pred[:, 2]))

# Print the RMSE values
print(f'RMSE for N: {rmse_N}')

```

```

print(f'RMSE for dN: {rmse_dN}')
print(f'RMSE for N_err: {rmse_N_err}')

# Function to fill missing values using the trained model
def fill_missing_values(df, model, seq_length):
    filled_df = df.copy()
    for i in range(len(df)):
        if missing_mask[i]:
            start_idx = max(0, i - seq_length)
            end_idx = i
            sequence = df[['N', 'dN', 'N_err']].iloc[start_idx:end_idx].values
            if len(sequence) < seq_length:
                sequence = np.pad(sequence, ((seq_length - len(sequence), 0), (0, 0)), 'constant', constant_values=0)
            sequence = sequence.flatten()
            sequence = np.nan_to_num(sequence)
            prediction = model.predict(sequence[np.newaxis, :])
            filled_df.loc[df.index[i], ['N', 'dN', 'N_err']] = prediction[0]
    return filled_df

# Fill missing values
filled_df = fill_missing_values(df, model, SEQ_LENGTH)

# Inverse transform to get original scale
filled_df[['N', 'dN', 'N_err']] = scaler.inverse_transform(filled_df[['N', 'dN', 'N_err']])

# Display the filled DataFrame
#print(filled_df)

plt.scatter(filled_df.index, filled_df.N, label='predicted', color='k')
plt.scatter(data['YYYYMMDD'], data['N'], label='original', color='cyan')

# Plot training data
# Scatter plot for training data
plt.scatter(df.index[:split_idx], filled_df['N'][:split_idx], label='Training Data', color='orange', s=10)

```



```

# Plot test data
plt.scatter(df.index[split_idx:], filled_df['N'][split_idx:], label='Test Data', color='purple', s=10)
plt.legend()

plt.ylabel("N_comp(mm)")
plt.xlabel("Time(in year)")
plt.grid()

import plotly.graph_objects as go

# Create figure
fig = go.Figure()

# Scatter plot for predicted data (filled data)
fig.add_trace(go.Scatter(
    x=filled_df.index,
    y=filled_df['N'],
    mode='markers',
    name='Predicted Data',
    marker=dict(color='blue')
))

# Scatter plot for original data
fig.add_trace(go.Scatter(
    x=data['YYYYMMDD'],
    y=data['N'],
    mode='markers',
    name='Original Data',
    marker=dict(color='cyan')
))

# Highlight missing values
fig.add_trace(go.Scatter(

```

```

x=df.index[missing_mask],
y=df['N'][missing_mask],
mode='markers',
name='Missing Values',
marker=dict(color='red', size=10)
))

# Scatter plot for training data
fig.add_trace(go.Scatter(
    x=df.index[:split_idx],
    y=filled_df['N'][:split_idx],
    mode='markers',
    name='Training Data',
    marker=dict(color='orange', size=10)
))

# Scatter plot for test data
fig.add_trace(go.Scatter(
    x=df.index[split_idx:],
    y=filled_df['N'][split_idx:],
    mode='markers',
    name='Test Data',
    marker=dict(color='purple', size=10)
))

# Add error bars for N_err component
fig.add_trace(go.Scatter(
    x=filled_df.index,
    y=filled_df['N'],
    error_y=dict(
        type='data',
        array=filled_df['N_err'],
        visible=True
    ),

```

```

mode='markers',
marker=dict(color='blue'),
showlegend=False
))

# Update layout
fig.update_layout(
    title="Original Data, Predicted Data, and Missing Values",
    xaxis_title="Time (in year)",
    yaxis_title="N_comp(mm)",
    legend_title="Legend",
    xaxis=dict(showgrid=True), # Enable grid for x-axis
    yaxis=dict(showgrid=True) # Enable grid for y-axis
)

# Show plot
fig.show()

```

Python code of LSTM Algorithm of Z- component:

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data=pd.read_csv(r"C:\Users\Aman Kumar\Desktop\Wadia internship\CHLM_ful.txt",delim_whitespace=True)

data
data['YYYYMMDD'] = pd.to_datetime(data['YYYYMMDD'], format='%Y%m%d')
data=data.set_index(data['YYYYMMDD'])

data = data.rename(columns={'+': 'N_err', '+.1': 'E_err', '+.2': 'U_err'})

data=data.asfreq(pd.infer_freq(data.index))

df=data.drop(['DecYr','YYYYMMDD','HHMNSC','MJD','F','F.1','F.2','dN','dE','N','N_err','E','E_err'],axis=1)

df

import pandas as pd

```

```

import numpy as np

from sklearn.preprocessing import MinMaxScaler

# Load your DataFrame (assuming df is already loaded)
# df = pd.read_csv('your_data.csv')

# Sort DataFrame by index (if not already sorted)
df.sort_index(inplace=True)

# Identify missing values
missing_mask = df['U'].isna()

# Fill missing values temporarily for normalization
df_filled = df.fillna(method='ffill').fillna(method='bfill')

# Normalize the data
scaler = MinMaxScaler()
df[['U', 'dU', 'U_err']] = scaler.fit_transform(df_filled[['U', 'dU', 'U_err']])

# Create sequences
def create_sequences(data, seq_length):
    xs, ys = [], []
    for i in range(seq_length, len(data) - 1):
        x = data[i-seq_length:i]
        y = data[i]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

# Define sequence length
SEQ_LENGTH = 10

# Create sequences
data_modified = df[['U', 'dU', 'U_err']].values
X, y = create_sequences(data_modified, SEQ_LENGTH)

```

```

# Filter out any sequences with NaN values
valid_indices = ~np.isnan(y).any(axis=1)
X, y = X[valid_indices], y[valid_indices]

# Split into training and test sets
split_idx = int(len(X) * 0.8)
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input

# Build the model
model = Sequential()
model.add(Input(shape=(SEQ_LENGTH, 3)))
model.add(LSTM(50, activation='relu', return_sequences=False))
#model.add(Dropout(0.2))
model.add(Dense(3)) # Predicting N, dN, N_err

model.compile(optimizer='adam', loss='mse')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_test, y_test))
model_losses = pd.DataFrame(model.history.history)
model_losses.plot()

import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

# Assuming X_test and y_test are already defined

```

```

# Make predictions
y_pred = model.predict(X_test)

# Compute RMSE for each target variable (N, dN, N_err)
rmse_U = sqrt(mean_squared_error(y_test[:, 0], y_pred[:, 0]))
rmse_dU = sqrt(mean_squared_error(y_test[:, 1], y_pred[:, 1]))
rmse_U_err = sqrt(mean_squared_error(y_test[:, 2], y_pred[:, 2]))

# Print the RMSE values
print(f'RMSE for U: {rmse_U}')
print(f'RMSE for dU: {rmse_dU}')
print(f'RMSE for U_err: {rmse_U_err}')

# Function to fill missing values using the trained model
def fill_missing_values(df, model, seq_length):
    filled_df = df.copy()
    for i in range(len(df)):
        if missing_mask[i]:
            start_idx = max(0, i - seq_length)
            end_idx = i
            sequence = df[['U', 'dU', 'U_err']].iloc[start_idx:end_idx].values
            if len(sequence) < seq_length:
                sequence = np.pad(sequence, ((seq_length - len(sequence), 0), (0, 0)), 'constant', constant_values=0)
            sequence = np.nan_to_num(sequence)
            prediction = model.predict(sequence[np.newaxis, :])
            filled_df.loc[df.index[i], ['U', 'dU', 'U_err']] = prediction[0]
    return filled_df

# Fill missing values
filled_df = fill_missing_values(df, model, SEQ_LENGTH)

# Inverse transform to get original scale
filled_df[['U', 'dU', 'U_err']] = scaler.inverse_transform(filled_df[['U', 'dU', 'U_err']])

```

```

import matplotlib.pyplot as plt

import pandas as pd

# Ensure 'YYYYMMDD' is in datetime format
data['YYYYMMDD'] = pd.to_datetime(data['YYYYMMDD'])

# Filter the data to start from January 1, 2010
start_date = '2010-01-01'
filtered_data = data[data['YYYYMMDD'] >= start_date]

plt.figure(figsize=(10, 4))

# Plotting the predicted values with error bars
plt.errorbar(
    x=filled_df.index,
    y=filled_df.U,
    yerr=filled_df.U_err,
    ecolor='red',
    label='Predicted Value',
    fmt='o',
    elinewidth=1,
    capsize=3
)

# Plotting the original values with error bars
plt.errorbar(
    x=filtered_data['YYYYMMDD'],
    y=filtered_data.U,
    yerr=filtered_data.U_err,
    fmt="",
    ecolor='green',
    elinewidth=0.5,
    capsize=3,
    label='Original Value'
)

```

```
)
```

```
plt.legend()
```

```
plt.xlabel("Time (year)")
```

```
plt.ylabel("U (mm)")
```

```
plt.grid()
```

```
plt.show()
```

```
import plotly.graph_objects as go
```

```
# Create figure
```

```
fig = go.Figure()
```

```
# Scatter plot for predicted data (filled data)
```

```
fig.add_trace(go.Scatter(  
    x=filled_df.index,  
    y=filled_df['U'],  
    mode='markers',  
    name='Predicted Data',  
    marker=dict(color='blue')
```

```
))
```

```
# Scatter plot for original data
```

```
fig.add_trace(go.Scatter(  
    x=data['YYYYMMDD'],  
    y=data['U'],  
    mode='markers',  
    name='Original Data',  
    marker=dict(color='cyan')
```

```
))
```

```
# Highlight missing values
```

```
fig.add_trace(go.Scatter(  
    x=df.index[missing_mask],
```



```

y=df['U'][missing_mask],
mode='markers',
name='Missing Values',
marker=dict(color='red', size=10)
))

# Scatter plot for training data
fig.add_trace(go.Scatter(
    x=df.index[:split_idx],
    y=filled_df['U'][:split_idx],
    mode='markers',
    name='Training Data',
    marker=dict(color='orange', size=10)
))

# Scatter plot for test data
fig.add_trace(go.Scatter(
    x=df.index[split_idx:],
    y=filled_df['U'][split_idx:],
    mode='markers',
    name='Test Data',
    marker=dict(color='purple', size=10)
))

# Add error bars for N_err component
fig.add_trace(go.Scatter(
    x=filled_df.index,
    y=filled_df['U'],
    error_y=dict(
        type='data',
        array=filled_df['U_err'],
        visible=True
    ),
    mode='markers',

```

```
marker=dict(color='blue'),
showlegend=False
))

# Update layout
fig.update_layout(
    title="Original Data, Predicted Data, and Missing Values",
    xaxis_title="Time (in year)",
    yaxis_title="U_comp(mm)",
    legend_title="Legend",
    xaxis=dict(showgrid=True), # Enable grid for x-axis
    yaxis=dict(showgrid=True) # Enable grid for y-axis
)

# Show plot
fig.show()
```