Coventry University

Faculty of Engineering, Environment and Computing

School of Computing, Mathematics and Data Science

MSc. Data Science

7150CEM Data Science Project

ENHANCING REAL TIME SPACE WEATHER FORECASTING THROUGH DEEP LEARNING

Author: Nawal Rabia Nizar

SID: 13282602

Supervisor 1: Dr. Lakhvir Singh

Supervisor 2: Dr. Tarjana Yagnik

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Data Science

Academic Year: 2023/24

Declaration Of Originality

I hereby certify that, with the exception of properly cited sources, this project is entirely original work of mine. Because of this, every instance of previously published material (found in books, journals, magazines, the internet, etc.) has been cited in the main report and linked to a relevant item in the Reference lists.
I also consent to the storage and use of an electronic copy of this project with the aim of preventing and identifying plagiarism.

Statement Of Copyright

I understand that Coventry University owns the copyright to this project report and any products created during it. Funding and other forms of support are available for staff and student-developed goods and services to be commercialized. The inventor(s) of the item or service divide any profits that are made. Please visit www.coventry.ac.uk/ipr or email ipr@coventry.ac.uk for further information.

Statement Of Ethical Engagement

I certify that this research has an application that has been filed to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/), using the application number provided below.

Signed:  Date:08-12-2023

First Name	Nawal Rabia
Last Name	Nizar
Student ID	13282602
Ethics Application Number	P164211
Supervisor 1	Dr. Lakhvir Singh
Supervisor 2	Dr. Tarjana Yagnik

Contents

Acknowledgement

I am immensely grateful for the support and guidance I have received throughout the course of my research project.

I extend my appreciation and wholehearted thanks to my Supervisor *Dr. Lakhvir Singh* for the continuous support and mentorship that proved invaluable in shaping the entire project.

My heartfelt thanks go to *Dr. Long Chen* for his insightful assistance in selecting a compelling research topic.

A special acknowledgment is owed to my husband, *Jazad NP,* for his unwavering support during busy project phases, taking care of me during times of illness, and handling household responsibilities.

I also want to express gratitude to *Dr. Alireza Daneshkhah* and *Dr. Alex Pedcenko* for their assistance in navigating the High-Performance Computing (HPC) system.

Finally, my sincere thanks to all the faculty members whose teachings and shared knowledge have contributed significantly to the successful completion of this project. Each of you has played a crucial role, and I am truly thankful for your support and contributions.

List Of Abbreviations

Acronym	Definition
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GRU	Grated Recurrent Unit
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
NASA	National Aeronautics and Space Administration
ACE	Advanced Composition Explorer
NOAA	National Oceanic and Atmospheric Administration
DSCOVR	Deep Space Climate Observatory
DST	Distributed Storm Index
CME	Coronal Mass Ejection
NCEI	National Centre for Environmental Information

Abstract

Geomagnetic fields play a pivotal role in various applications, from navigation systems to scientific research and space weather monitoring. Accurate modelling is essential for their reliable use. This project presents a benchmark study addressing the need for precise geomagnetic field modelling.

The primary objective of this project is to establish a standardized model for evaluating and advancing geomagnetic field models. By utilizing real-time solar-wind data feeds from NASA Advanced Composition Explorer (ACE) and National Oceanic and Atmospheric Administration (NOAA) Deep Space Climate Observatory (DSCOVR) satellites, this project aims to improve predictive performance.

This project leverage cutting-edge techniques, including Deep Learning, Long Short Term Memory LSTM, Grated Recurrent Unit (GRU) and Recurrent Neural Network (RNN), to develop models that can forecast the Disturbance Storm-Time Index (DST) in real-time. The project provides a systematic framework for comparing various modelling approaches.

The project yields a significant advancement in the precision and operational viability of DST forecasting. The benchmark dataset and evaluation metrics contribute to the development of more accurate geomagnetic field models.
Accurate DST forecasts are vital for systems reliant on Earth's natural magnetic field as a reference. This project's findings benefits satellite operators, power grid operators and users of magnetic navigation systems. It promises increased reliability in the face of geomagnetic disturbances.

In conclusion, this project offers a critical step in enhancing geomagnetic field modelling. The project recommends the continued development of advanced models, ensuring their practical implementation for real-time forecasting.

Keywords: Geomagnetic field, benchmark study, DST forecasting, solar-wind data, Deep Learning, LSTM, RNN, operational viability.

1 Introduction

Figure 1 Aurora, Source: Google's Public Domain Images

The dancing northern lights shine in mesmerize in the Arctic night on many an icy winter's night. And then, where does this celestial phenomenon come from that we enjoy watching every night? Our sun is just an ordinary star in the milky way galaxy and it marks the beginning of the fascinating story of the Aurora. There is more to the Sun than meets the eye. It serves as a giant nuclear generator and power house, where incredible energies are burnt deep inside its burning flames.

The temperature at the centre of this star is approximately fourteen million and also the pressure is enormous such that the hydrogen molecules combine to form helium. It is a surprising nuclear reaction that can unleash an inestimable volume of energy. From their source in the centre of the Sun, radiance traverses outwards through its outer regions.

It generates massive swirling currents called convection cells with this radiant energy. Dynamos that are composed of charged gases convection cells which are the key factors which contribute to the creation of Sun's magnetic fields. In some parts of the sun, strong magnetic fields push their way through the surface of the sun, stopping the whirls of incandescent gas and giving birth to dark sunspots. Black patches indicate the presence of plasma or charged gas in a given space.

The magnetic field is dragged out by the plasma, which looks like an electrically charged gas. The tension increases more and more until, with a bang, the rubber band explodes. This important phase is the ejection of billions of tons of plasma from the Sun, the Solar Storm.

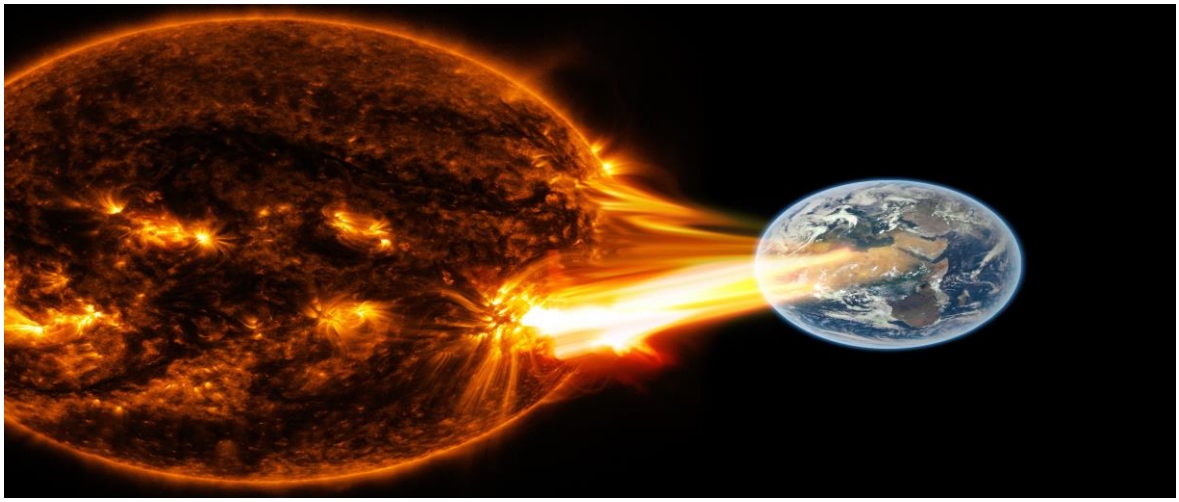


Figure 2 Solar Storm, Source: Google's Public Domain Images

Solar storms, propelled at astonishing velocities, can travel at over 8 million kilometres per hour. After a mere six hours, they bypass the planet Mercury, and in just twelve hours, they reach the vicinity of Venus. Within eighteen hours, the solar storm tempest makes its arrival at Earth.

The impact of a solar storm by an invisible shield referred to as the Earth's magnetic field, effectively deflects the charged particles of the storm. The magnetic fields of Earth intertwine and Storm upon our planet instigates a fascinating sequence of events. Earth, shrouded shape a conduit, guiding the streaming gas down to the daylight side of the polar regions, a phenomenon recognized as the daylight aurora.

But what unfolds on the night side of our planet? Here, the magnetic field continues to extend and entwine. As the metaphorical magnetic rubber band eventually breaks, gas from the Solar Storm journeys along the magnetic lines toward the polar region on the night side. It is here that the enigmatic and captivating night-time aurora graces the Arctic skies with its luminescence.

Expanding upon this celestial narrative, we delve into the realm of geomagnetic storms, which emerge from the intricate interplay between Solar Storms and Earth's magnetic field. These geomagnetic storms engender disruptions in our planet's magnetic field, resulting from the arrival of solar particles. In addition to intensifying the aurora displays, geomagnetic storms can also impact crucial technological systems, including power grids and communication networks.

In the quest for advanced geomagnetic storm forecasting, this project leverages real-time solar-wind data feeds from NASA's Advanced Composition Explorer (ACE) and NOAA's Deep Space Climate Observatory (DSCOVR) satellites. The goal is to develop cutting-edge

models that not only push the boundaries of predictive performance but do so within operationally viable constraints.

The primary aim of this project is to enhance ability to predict the Disturbance Storm-Time Index (DST), a measure of magnetic activity, using the provided solar wind data. DST values, essential for forecasting geomagnetic storms, are derived from ground-based observatories. These values are averaged to provide DST measurements for specific hours, yet they are not always available in a timely manner.

This project embarks on a journey through the cosmos, from the heart of our Sun to the mesmerizing auroras of Arctic nights. This research strive to unlock the mysteries of geomagnetic storms and provide timely and accurate forecasts by harnessing real-time solar-wind data. The fusion of science, technology, and a dash of celestial magic fuels our mission to push the boundaries of predictive performance in geomagnetic storm forecasting.

1.1 Background Of The Project

1.1.1 Science Behind Geomagnetic Storm

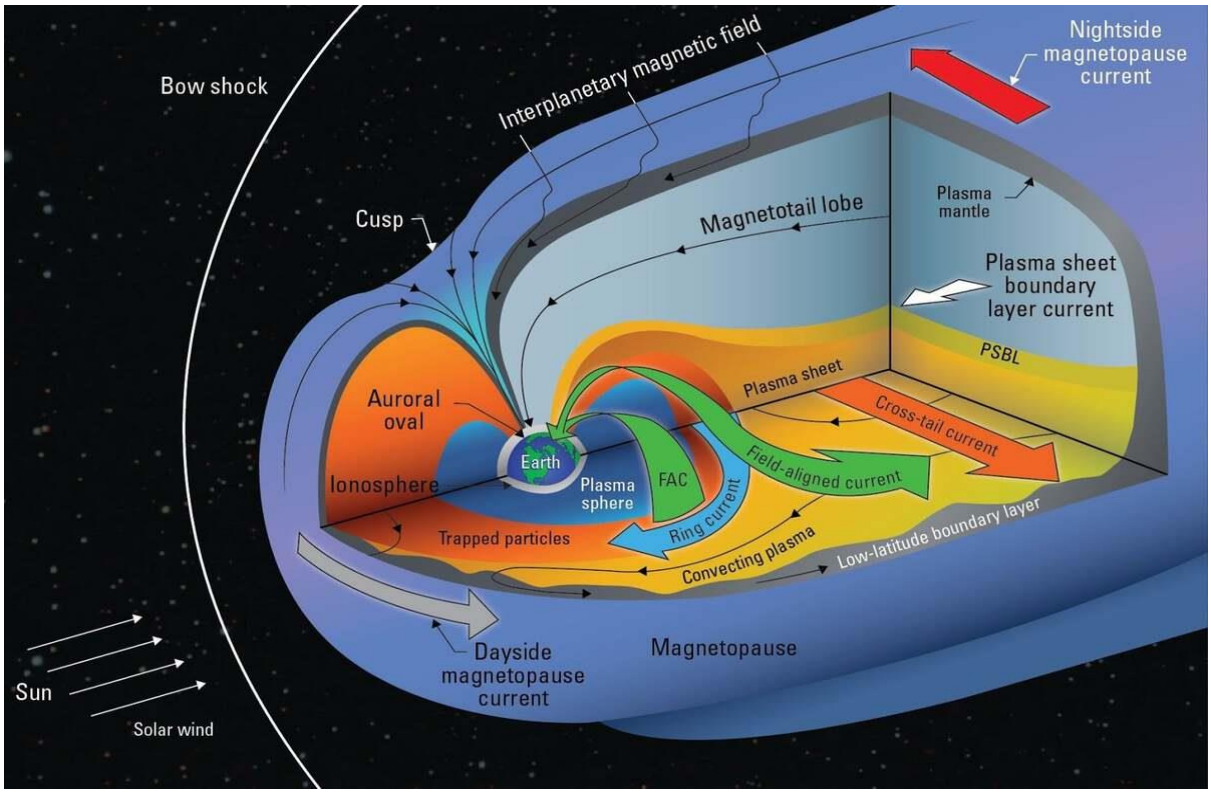


Figure 3:A simplified cartoon illustrates Earth's magnetosphere, with the DST index used to measure the "ring current" (depicted in blue) encircling our planet. This ring current is generated by charged particles confined within the magnetosphere.

Geomagnetic storms are significant disturbances in Earth's magnetic field caused by interactions with solar wind (National Academies Press, 2017). They have wide-ranging effects on technology and the Earth's environment.

Solar wind is generated by the Sun's outermost layer, the Corona. Solar Flares, Solar Wind, and Coronal Mass Ejections (CMEs) are primary sources of these charged particles that can cause geomagnetic storms.

The solar wind, composed of charged particles, interacts with Earth's magnetic field. This interaction disrupts the geomagnetic field, leading to disturbances known as geomagnetic storms.

Geomagnetic storms are associated with the strengthening of the ring current, a current of charged particles trapped in the magnetosphere. The DST (Disturbance Storm-Time) index measures the intensity of these disturbances, particularly related to the ring current.

Magnetic reconnection is a fundamental process that occurs during geomagnetic storms. It involves the breaking and reconfiguration of magnetic field lines. This process releases energy and can accelerate charged particles, contributing to storm intensity.

Charged particles from the solar wind interacting with Earth's atmosphere led to the creation of stunning aurora displays, such as the Northern and Southern Lights. Geomagnetic storms can enhance the visibility and intensity of these auroras.

Geomagnetic storms have significant impacts on technology and space weather. They can disrupt satellite communication, GPS systems, power grids, and navigation equipment, causing widespread technological and economic disruptions.

Efforts have been made to forecast geomagnetic storms and mitigate their effects. Real-time monitoring and early warning systems are crucial for minimizing the damage caused by these storms.

Geomagnetic storm activity is closely linked to the 11-year solar cycle. During periods of increased solar activity, such as solar maximum, geomagnetic storms are more frequent and intense.

Recent advances in geomagnetic storm research have led to improvements in modelling and prediction. The use of data from satellites, such as NASA's ACE and NOAA's DSCOVR, combined with ground-based observatories, has advanced the field and improved our understanding of these phenomena.

1.1.2 "Sun's Fury Unleashed: The Catastrophic Impact of a Massive Solar Storm on Earth"

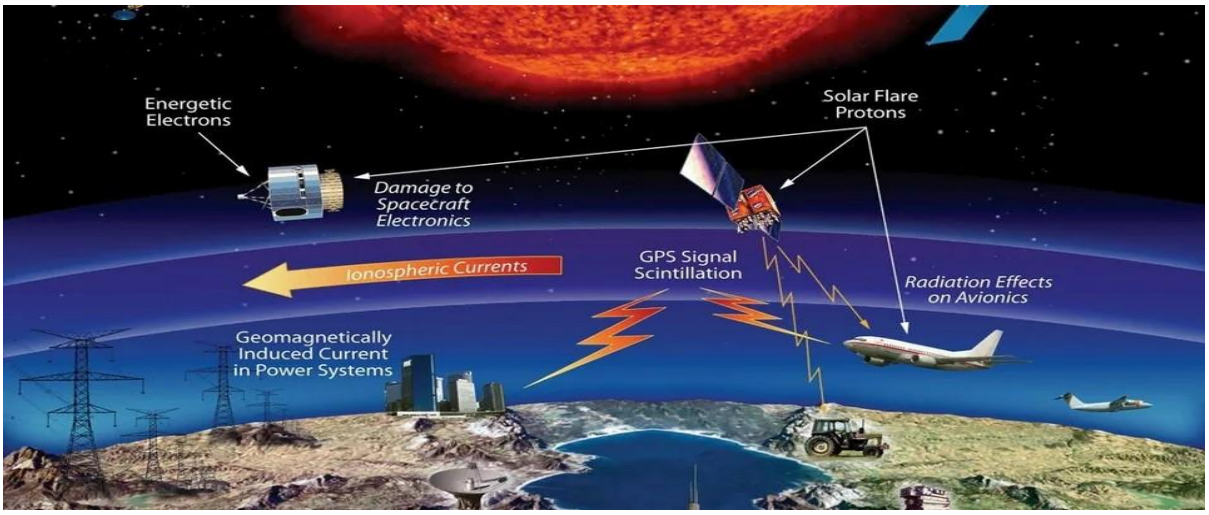


Figure 4: Depicting effects of solar storm on Earth, Source: Google's Public Domain Images

Although a solar flare burst from the sun's surface is visually stunning, they may be dangerous to humans. If one were to strike us with sufficient force, it would cause the entire world to go black.

A strong storm tore past Earth's orbit in 2012, but fortunately, Earth did not suffer the direct damage.

We weren't as fortunate in 1859 (the Carrington Event), when telegraph poles caught fire due to electromagnetic radiation from another CME. Messages could still be sent and received by some telegraph operators even after their power supply were unplugged. At the time, the only genuine technological advancement was the telegraph.

However, today's world is heavily dependent on technology and power. We would be in far worse shape now if a solar storm that powerful were to hit. It would begin with a massive explosion on the surface of the Sun. A massive electromagnetic pulse would be sent into Earth's upper atmosphere by the solar flare. This would disrupt radio communications between Earth and our orbiting satellites, but it wouldn't really harm them until the Earth's magnetosphere began to be bombarded with a torrent of charged particles minutes or hours later. Some of the satellites might be struck by these particles, damaging their electronics. Our means of communication would start to malfunction. However, the worst is still to come.

The planet will eventually be reached by a cloud of plasma, which might take anywhere from 12 hours to many days.

It would first strike NASA's ACE satellite, which is meant to alert us to the impending storm.

In the event that the aircraft's GPS system fails, the pilot would have to navigate the trip without it.

A worldwide power outage will result from the melting of power grid transformers. Every light would go out, phones and computers would cease charging, refrigerators and even heaters would malfunction. Since most cities' water supplies are managed electronically, all ATMs will be rendered inoperable. Everything and everything that was internet-dependent would stop working.

Thus, to sum up, we are unable to forecast space weather. NASA and The Space Weather Prediction Centre continue to track solar activity. Their three-day prediction would alert them if the sun appeared suspect.

Perhaps there will be time to turn off the transformer and put the satellites in safe mode. And in the future, we will erect a barrier around the planet to keep things like these from harming us.

1.2 Prediction Of DST Using Deep Learning Techniques

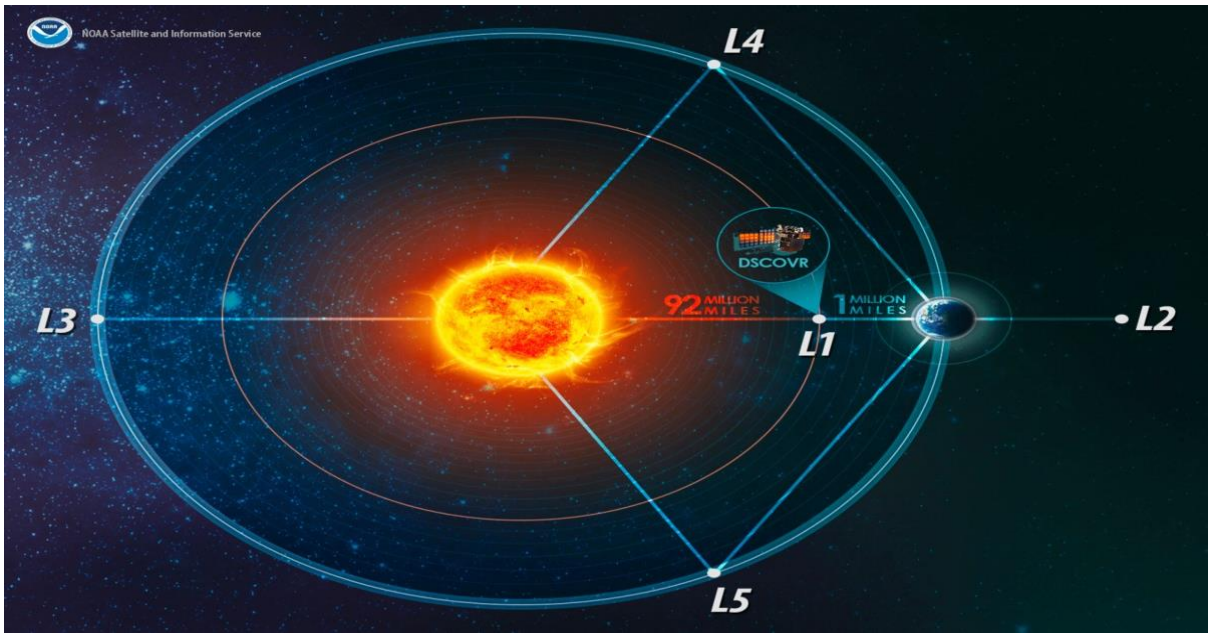


Figure 4 : Predict the DST index exclusively based on solar-wind data collected at the Lagrangian (L1) point, utilizing information from NOAA's Deep Space Climate Observatory (DSCOVR) and NASA's Advanced Composition Explorer (ACE) satellites.

The main data source for the project consists of sensor data obtained from two satellites: NASA's ACE and NOAA's DSCOVR. This space weather information encompasses sensor measurements associated with both the interplanetary magnetic field and solar wind plasma.
NASA's ACE: The NASA Advanced Composition Explorer (ACE) satellite plays a crucial role in providing early alerts for geomagnetic storms. These storms, akin to natural hazards like hurricanes and tsunamis, are forecasted by the National Oceanic and Atmospheric Administration (NOAA) Space Weather Prediction Centre (SWPC) to safeguard the public's interests. Geomagnetic storms have significant impacts on various sectors, including the electrical power grid, aviation operations, GPS systems, human spaceflight, and satellite operations, among others. Severe geomagnetic storms can even lead to widespread power outages for electric utilities.
Positioned at the L1 liberation point, approximately 1,500,000 kilometres ahead of Earth in the direction of the Sun, ACE has the capability to provide advanced notice of impending space weather events that could be detrimental to Earth. SWPC utilizes ACE's data to issue timely warnings about imminent geomagnetic storms.
NOAA's DSCOVR: DSCOVR, launched in 2015, monitors real-time solar wind, essential for NOAA's space weather predictions. These forecasts are critical because space weather events like geomagnetic storms can disrupt vital infrastructure systems. DSCOVR replaced ACE in providing solar wind alerts from the L1 orbit, situated about a million miles from Earth, where it can offer up to an hour's advance notice before solar wind particles reach Earth.
From L1, DSCOVR provides 15- to 60-minute warnings before a particle and magnetic storm (Coronal Mass Ejection) arrives. It also enhances predictions of geomagnetic storm impacts, safeguarding national security and our economy, which rely on advanced technologies.

What does DST index actually mean?

A measure of magnetic activity in Earth's ring current in units of nanoTesla (nT) as measured by a network of geomagnetic observatories along Earth's equator, with large negative values (e.g., -500nT) indicating a significant space weather event has occurred that weakened Earth's magnetic field(Montgomery,2021).

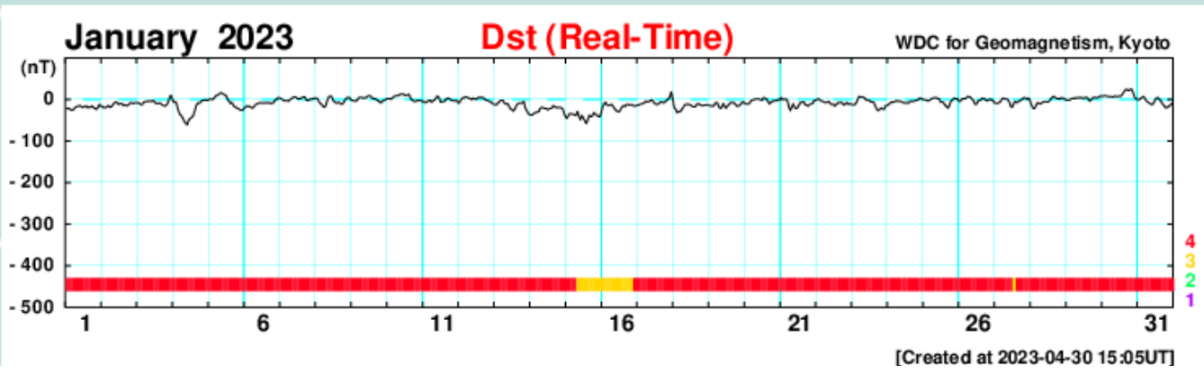


Figure 5 DST Graph as recorded by WDC for Geomagnetism, Kyoto Japan

Yaxis indicates DST values (in the strength of magnetic field in earth's ring current in units of nanoTesla averaged every day over the entire month of January)(which is on Xaxis). Noticing average is zero, indicating no significant space weather has impacted earth. Looking at another graph :

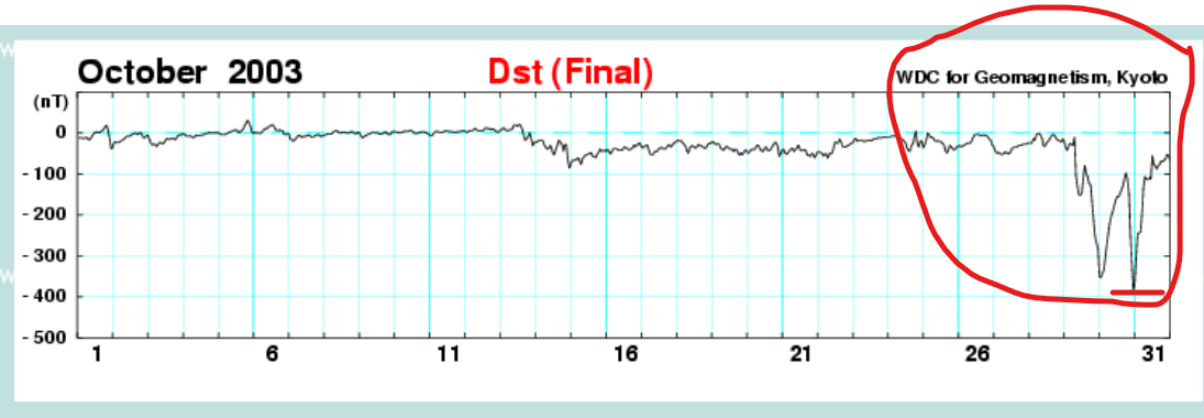


Figure 6 DST Graph as recorded by WDC for Geomagnetism, Kyoto Japan

Large event of space weather occurred, -400 nT, reason why we have a negative value is because Earth's magnetic field has weekend due to influx of protons and electrons from solar event. This type of event results in Aurora being formed, black outs, communication and satellite deterioration etc.

1.3 Project Aim And Objectives

Aim

The aim of this project is to advance the accuracy and timeliness of geomagnetic storm prediction by developing real-time forecasting models using cutting-edge data science and deep learning techniques. This project seeks to provide critical lead-time to protect systems and infrastructure vulnerable to geomagnetic disturbances.

Objectives

1. Develop Real-Time Predictive Models: Create models that can forecast the Disturbance Storm-Time Index (DST) in real-time, utilizing historical solar-wind and geomagnetic data.
2. Maximize Predictive Precision: Develop and optimize Deep Learning models to minimize RMSE values in forecasting geomagnetic disturbances. Focus on refining model architectures, data pre-processing, and hyperparameter tuning to consistently reduce RMSE, striving for highly accurate predictions in the assessment of geomagnetic storm indices.

1.4 Geomagnetic Storm Forecast Imperatives

The precise and timely forecasting of geomagnetic storms is the current issue. The intricate interplay between Earth's magnetic field and solar wind are what cause these storms. Numerous systems, such as GPS, satellite communication, and power grids, are susceptible to disruption by them. To lessen the effects of severe storms, accurate storm forecasting is essential.

The DST, which measures the strength of geomagnetic storms, is one of the main obstacles (Sugiura & Kamei, 1991). Ground-based observatories have historically observed DST values, although there has been variability in the availability and timeliness of these readings. The main goal of the issue statement is to create models that can accurately forecast DST in real-time and provide important advance notice for planning.

There are several driving forces behind this endeavour. First, there is an increasing reliance on technology that is susceptible to disruptions caused by geomagnetic fields. Reliability assurance is crucial for several industries, such as energy, navigation, and telecommunications. The necessity to safeguard these vital systems serves as the driving force.

In addition, the project makes use of cutting edge modelling tools, Deep Learning methods, and Data Science. This illustrates a larger drive to use technology to solve problems in the real world. The project represents the coming together of realistic, workable solutions with scientific theory.

There's also the motive of society. Improving our knowledge of and ability to predict geomagnetic storms helps us become more resilient to natural disasters. It has to do with expanding information and guaranteeing the dependability and security of our globalized society.

2 Literature Review

In light of the extensive body of existing research in the field, this literature review aims to synthesize and analyse relevant studies to provide a comprehensive understanding of the current state of knowledge regarding DST prediction and understand space weather conditions.

Manoj Nair et al. (2023) described the 'MagNet—A Data-Science Competition to Predict DST From Solar Wind Data. This competition showcased several top-performing models for predicting the DST index using solar wind data. The winning models presented in the paper provide valuable insights into the accurate prediction of geomagnetic storm intensity.

Bidirectional LSTM-GRU modelling involved predicting the DST for the current and next hour. The solution builds upon a baseline model and introduces enhancements, including a Bidirectional LSTM-GRU model, data pre-processing strategies, and thoughtful model parameter selection. A unique weight initialization technique is highlighted for its significant impact on model performance.

CNN modelling involved machine learning approach involved the use of a Convolutional Neural Network (CNN) with TensorFlow and Keras for predicting the DST Index . It focused on solar wind and sunspots data, excluded satellite data, and pre-processed the data by filling in missing values and normalizing it. The neural network model included multiple layers of convolutions, and trained an ensemble of models for time t and t+1.

The neural network approach outperformed tree-boosting models. The evaluated model performance using RMSE and recommended future work, including addressing systematic differences between data from different satellites and exploring the use of temperature data. Additionally, suggested collecting sensor data to identify malfunctions and outliers in the dataset.

LGBM & NN ensemble models tackled the challenge of predicting the DST using data from two satellites. They applied a well-structured approach to feature engineering, including transformations, statistical measures, and Fourier features, to process the solar wind data. The machine learning models used were an ensemble of three models: one Gradient Boosting Machine (LGBM) and two Feed-Forward Neural Networks (NN). Importantly, they employed feature selection methods like Permutation Importance to reduce over fitting and

filter the most relevant features. Ensembling techniques were also used to combine models, which significantly improved the model's performance.

Although they explored various feature engineering and selection strategies, they ultimately settled on the most effective techniques. If they continued working on the problem, they would further fine-tune their neural network models, optimize feature selection, and experiment with model stacking to potentially enhance their results.

4 block CNN employed a Convolutional Neural Network (CNN) architecture with residual connections, temporal max pooling, and a custom loss function to control over and under predictions. The ensemble of 21 models, each with variations in loss function parameters and random weight initializations, improved predictive accuracy.

While they experimented with various techniques, including XGBoost and feature engineering, they found that their CNN-based approach with the custom loss function and ensemble outperformed other methods.

For future work, they suggest exploring more advanced time series analysis techniques to extract seasonality patterns and obtaining a longer historical dataset for improved forecasting accuracy.

The study, authored by S. B. Xu et.,al 2001 presents a novel approach to predicting the DST Index using a Bagging ensemble-learning algorithm. The authors combine three algorithms, including **Artificial Neural Networks, Support Vector Regression, and Long Short-Term Memory networks,** to predict the DST Index with a lead time of 1-6 hours. The results are promising, with the root mean square error (RMSE) consistently below 8.0936 nT and a correlation coefficient (R) exceeding 0.8572 for point predictions. The interval prediction accuracy is consistently above 90%. This demonstrates the model's ability to improve point predictions and significantly enhance interval predictions, which are essential for space weather forecasting.

The study also introduces a new metric to evaluate the model's stability, showing that the Bagging algorithm enhances the model's robustness. Furthermore, the model's performance during a magnetic storm event from October 12-17, 2016, is highlighted, with the 1-hour ahead prediction achieving remarkable results with an RMSE of 3.7327 nT, a correlation coefficient of 0.9928, and an interval prediction accuracy of 96.69%. This research provides a promising approach to accurately predict the DST index, a key component of space weather forecasting, using a Bagging ensemble-learning algorithm. The study's findings contribute to the field of space weather prediction and highlight the significance of ensemble learning techniques.

Paper authored by H.Lundstedt et.,al 1998 designed a model for real time forecasting of the DST. It utilized a recurrent neural network **(Elman Neural Network)** optimized to be small without comprising accuracy. The model relies solely on hourly averages of solar wind parameters, including the magnetic field component Bz, particle density n and velocity V.
The Elman neural network architecture is described with inputs representing normalized solar wind data and outputs corresponding to DST. The model undergoes training and optimization and the chosen architecture includes an input layer, a hidden layer with context inputs and an output layer.
Equation describing the network's output involve weighted sums and non-linear functions.
Statistical evaluation based on 40,000 hours of solar wind DST-data shows the model has smaller prediction errors compared to other models.
The RMSE result was ranging between from 55 to 7 at various levels of DST and there were many RMSE errors predicted for different levels of DST. Least RMSE result was around 7.9.

Authors	Properties Predicted	Techniques Used	Metrics used for measurement	Results(nT)
Manoj Nair et al. (2023) described the MagNet—A Data-Science Competition to Predict Disturbance Storm-Time Index (Dst) From Solar Wind Data,2019				
Model 1	DST index	Bi-directional LSTM,Bi-GRU	RMSE	12.12
Model 2	DST index	CNN ,Tensor flow	RMSE	11.79
Model 3	DST index	Ensemble of 2 models LGBM,NN	RMSE	11.29
Model 4	DST index	4-block Deep CNN	RMSE	11.53
S. B. Xu, S. Y. Huang, Z. G. Yuan, X. H. Deng, and K. Jiang,2020	DST index	Ensemble of ANN,LSTM,SVM	RMSE	8.09
H.Lundstedt, H.Gleisner,2002	DST index	Elman neural network	RMSE	7.9

Table 1:Summary of different papers related to DST prediction

3 Methodology

1. Requirements Gathering and Model Architecture Design

BI-LSTM-GRU Combination: Integrate a Bidirectional Long Short-Term Memory (Bi-LSTM) layer with a Bidirectional Gated Recurrent Unit (GRU) for addressing time-series and text recognition problems.

Dense Layers and Attention Mechanism: Add three dense layers connected through an attention mechanism. Specify the neuron counts in each layer.

2. Data Pre-processing

Feature Selection: Consider all available features, allowing the neural network to determine feature relevance.

Data Imputation: Address missing values using a simple imputer with the "most frequent" strategy.

Data Scaling: Implement standard scaling for model robustness.

Data Splitting: Select testing and validation data from the dataset's beginning to avoid time-dependent issues. Allocate 6,000 samples for testing and 10,000 samples for validation.

3. Model Initialization

Weight Initialization: Utilize a dummy model to initialize the main model's weights with diverse values for loss reduction and improved performance.

4. Training Process

Loss Function: Employ the Root Mean Squared Error (RMSE) loss function for assessing model performance during training.

Training Strategy: Set a learning rate of 0.0001 and use the ReduceLROnPlateau schedule with a reduction factor of 0.3 and a patience of 4 steps for monitoring the validation loss. Maintain a fixed batch size of 128.

5. Evaluation

Validation: Evaluate model performance on the validation dataset during development.

Testing: Conduct the final evaluation on the testing dataset to assess real-world predictive capabilities.

6. Model Performance Analysis

Results: Expect substantial improvements in performance compared to the other implemented model. Provide detailed performance metrics in the results section.

7. Discussion

Model Limitations: Identify and discuss potential limitations and sources of error.

Implications: Explore and discuss the implications of findings and their real-world applications.

8. Reproducibility

Provide detailed information on code, data sources, and configurations for future reference and reproducibility.

9. Conclusion

Conclude that the described methodology is expected to enhance the DST prediction model's performance, aligning with the research objectives.

10. Outcome

Aim to achieve a DST index with an RMSE value closer to or less than the existing literature benchmark value of 16.3293 (nanoTesla).

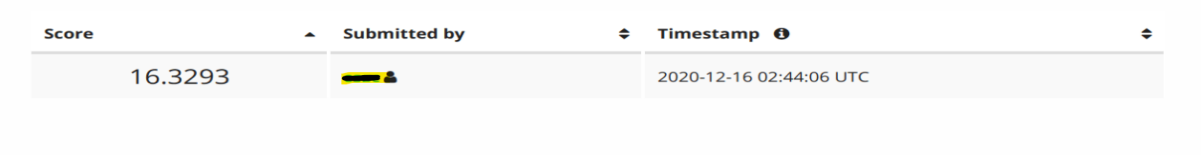
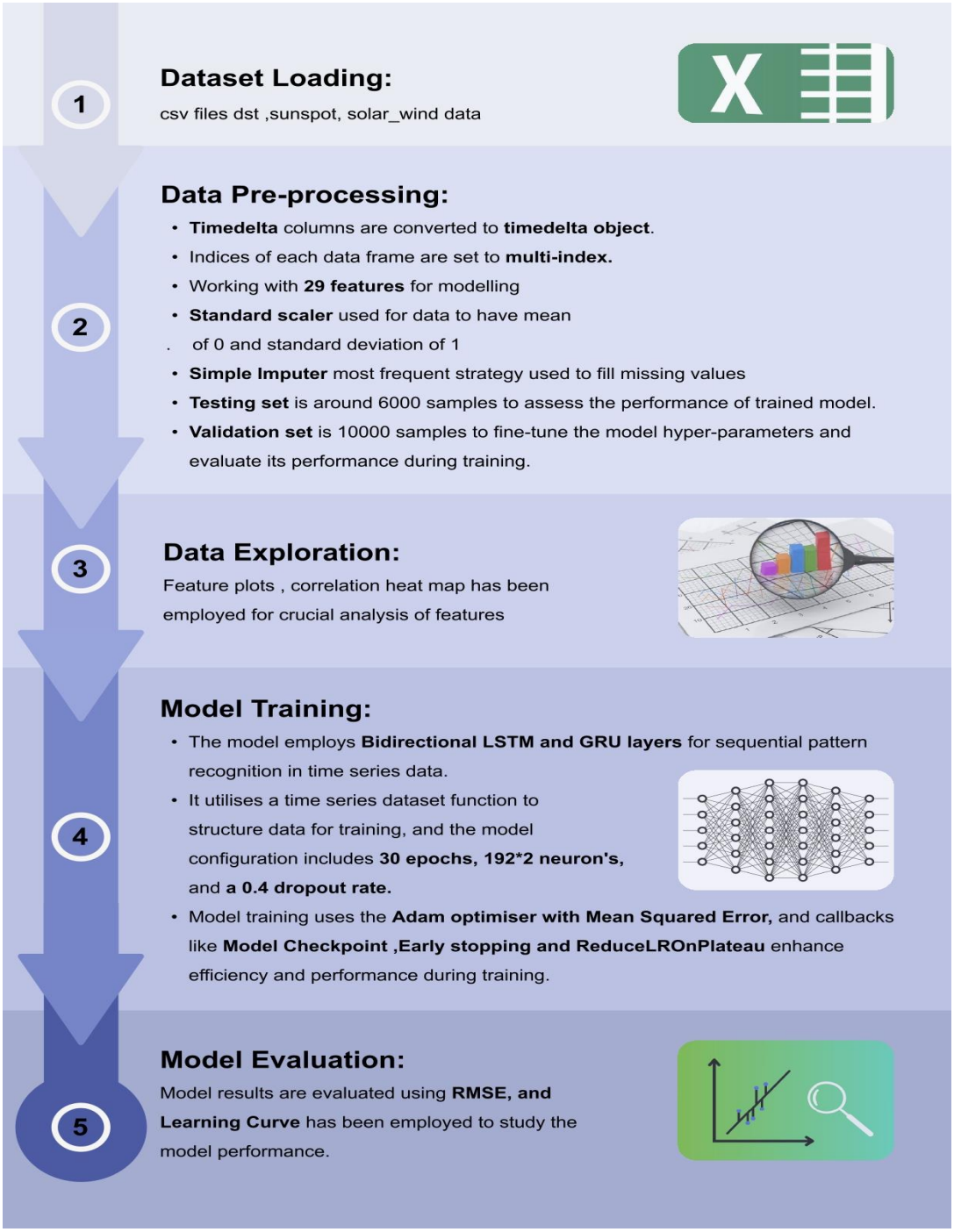


Figure 7 Sample Output Value

4 Design

The brief design of best model implemented in this project is presented below. It has 5 parts with each part performing a set of activities. Detailed explanation of the model implemented is in **Section 7.6** of the document. Other models tested and their detailed analysis are also explained in the **Section 8.2** as a part of **Results**.



Modelling workflow created using Figma Wireframe tool

5 Requirements

5.1 Software Requirements

- Operating System in use: Windows(Bill Gates & Paul Allen ,1985)
- Python: Python 3.7.3 (Van Rossum & Drake, 2009)
- Deep Learning Framework: Keras (Francois Chollet,2015)
- IDE: Jupyter notebook (Fernando Perez,2000)
- AWS's Amazon Sage Maker's Jupyter instance
- Figma Wireframe tool
- Dependencies: All required libraries to be imported useful for data analysis and visualization

5.2 Data Collection

Data for this project is collected from NOAA's NCEI (National Centre for Environmental Information) website.

https://www.ngdc.noaa.gov/geomag/data/geomag/magnet/

6 Dataset

The dataset selected for this project has four csv files having features related with modelling the geo magnetic field .

Filename	Description	Frequency	No of Instances	No of Features
solar_wind.csv	Solar wind data collected from ACE and DSCOVR satellites	Minutely	1048576	17
sunspots.csv	Smoothed sunspot counts	Monthly	82	3
satellite_positions.csv	Coordinate positions for ACE and DSCOVR	Daily	2467	8
dst_labels.csv	DST values averaged across the four stations	Hourly	59185	3

Table 2: Data Files

Feature name	Description	Datatype
SOLAR WIND DATA		
bx_gse	The component of the interplanetary magnetic field (IMF) along the X-axis in geocentric solar ecliptic (GSE) coordinates, measured in nanoteslas (nT).	Numeric
by_gse	The Y-component of the interplanetary magnetic field (IMF) in geocentric solar ecliptic (GSE) coordinates, measured in nanoteslas (nT).	Numeric
bz_gse	The Z-component of the interplanetary magnetic field (IMF) in geocentric solar ecliptic (GSE) coordinates, expressed in nanoteslas (nT).	Numeric
theta_gse	The latitude of the interplanetary magnetic field (IMF) in geocentric solar ecliptic (GSE) coordinates, which is defined as the angle between the magnetic vector B and the ecliptic plane.	Numeric

	It is considered positive when the magnetic field points North and is measured in degrees.	
bx_gsm	Interplanetary-magnetic-field X-coordinate in the geocentric solar magnetospheric (GSM) reference frame (in nanoteslas, nT).	Numeric
by_gsm	Interplanetary-magnetic-field Y-coordinate in the geocentric solar magnetospheric (GSM) reference frame (in nanoteslas, nT).	Numeric
bz_gsm	Interplanetary-magnetic-field Z-coordinate in the geocentric solar magnetospheric (GSM) reference frame (in nanoteslas, nT).	Numeric
theta_gsm	Latitude of the interplanetary-magnetic-field in geocentric solar magnetospheric (GSM) coordinates (measured in degrees).	Numeric
phi_gsm	Longitude of the interplanetary-magnetic-field in geocentric solar magnetospheric (GSM) coordinates (measured in degrees).	Numeric
bt	Magnitude of the interplanetary-magnetic-field component (measured in nanoTesla - nT).	Numeric
density	Density of solar wind protons, measured in particles per cubic centimeter (N/cm^3).	Numeric
speed	Speed of the solar wind, measured in kilometres per second (km/s).	Numeric
temperature	Temperature of solar wind ions, expressed in Kelvin.	Numeric
source	Beginning in 2016, solar wind data for a specific time can originate from either the DSCOVR or ACE satellites, depending on data quality. The abbreviation "ac" represents sourcing from ACE, while "ds" indicates data	Categorical

	sourced from DSCOVR.	
SATELLITE COORDINATE DATA		
gse_x	Satellite's location in the X-axis of GSE coordinates (kilometres)	Numerical
gse_y	Satellite's location in the Y-axis of GSE coordinates (kilometres)	Numerical
gse_z	Satellite's location in the Z-axis of GSE coordinates (kilometers)	Numerical
SUNSPOT DATA		
period	Categorizing data into different periods	Categorical
Smoothed_ssn	No of sunspots observed at sun surface	Numerical
DST DATA		
DST values	DST values averaged across four stations	Numerical

Table 3: Features list in Dataset

7 Techniques

7.1 Feature Scaling

Feature scaling is a crucial pre-processing step. It aims to standardize the range of features, ensuring that each feature has an average (mean) of 0 and a standard deviation of 1 (Xing Wang et.,al 2018). This is achieved using the formula

$$x' = \frac{x - \mu}{\sigma}$$

x' represents the standardized value, x is the original value, μ is the mean, and σ is the standard deviation of the feature. The `StandardScaler` from scikit-learn automates this process, making data suitable for machine learning models. Standardization is essential for consistent feature scales, preventing one feature from dominating machine learning algorithms and improving model performance.

7.2 Exploring Features In Dataset Using Correlation Heatmap

Correlation Heatmap is used as a technique to gain insights into the relationships between different features in the dataset. The Correlation Heatmap is a visual representation of the correlation coefficients between pairs of features. Correlation measures the statistical association between two variables, indicating whether and how they change together(Eisen et al.,1998). In this context, it is used to examine the degree of linear relationship between the various features.

The heatmap is constructed based on the correlation matrix, where each cell represents the correlation coefficient between two features. The values in the heatmap can range from -1 to 1, where:

- A value of 1 indicates a perfect positive correlation (as one feature increases, the other increases proportionally).
- A value of -1 indicates a perfect negative correlation (as one feature increases, the other decreases proportionally).
- A value of 0 indicates no linear correlation (the two features are not related in a linear way).

By visualizing this information in a heatmap, one can quickly identify which pairs of features are strongly correlated, which can be useful for feature selection or dimensionality reduction.

Highly correlated features can sometimes be redundant and may not provide additional information to the model.

The heatmap can be generated using the `matplotlib` library and provides a graphical representation of the feature correlations, helping researchers or data scientists make informed decisions about which features to include or exclude from their analysis. It's a powerful tool for understanding the dataset's underlying structure and relationships between variables.(Zuguang Gu et al.,2022).

7.3 What is RNN And How It Is Different From ML, ANN, CNN?

What was the necessity to have RNN algorithms even though we have ML, ANN, CNN? Let's try to understand in simple way :

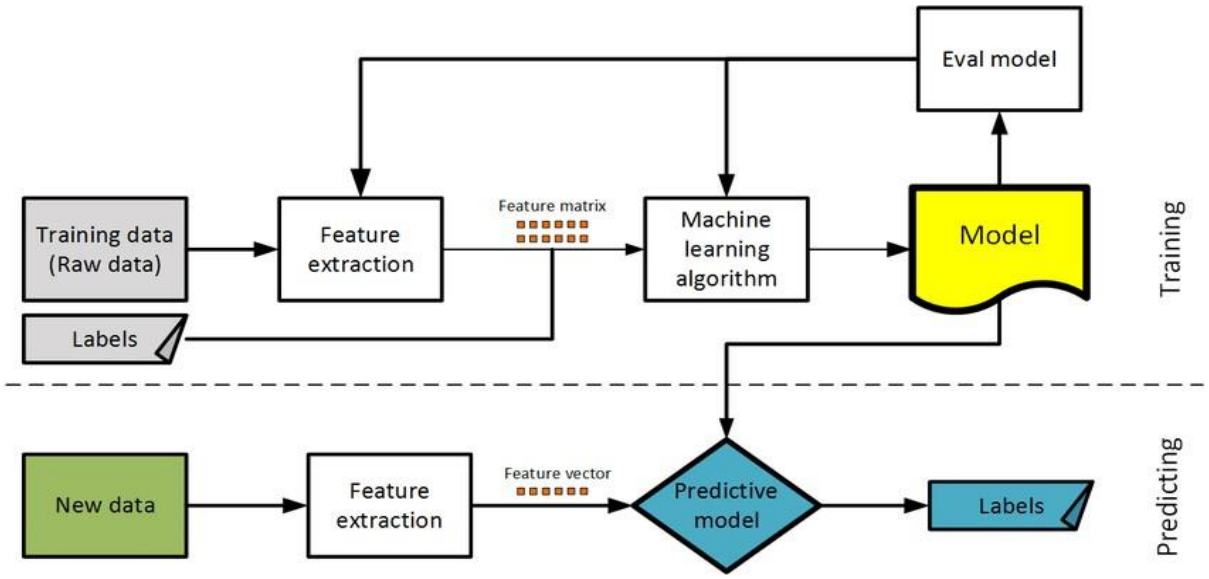


Figure 8 Source: Wikipedia, Basic ML pipeline

In Machine Learning , feature engineering involves selecting and extracting relevant features from the data to improve the performance of a model . This is crucial when working with small datasets as manually crafted features help the model understand the underlying patterns.

As datasets grow larger, traditional ML techniques can become time-consuming . This led to the advent of Artificial Neural Networks (ANNs). ANNs, a subset of machine learning, automate feature engineering through processes like back propagation, adjusting weights and biases to learn complex patterns in data. It allows deep architectures, allow for automatic feature extraction. They function as universal approximators, capable of learning intricate

relationships in data through layers of neurons. Key components include activation functions, biases and weights that adjust during training.

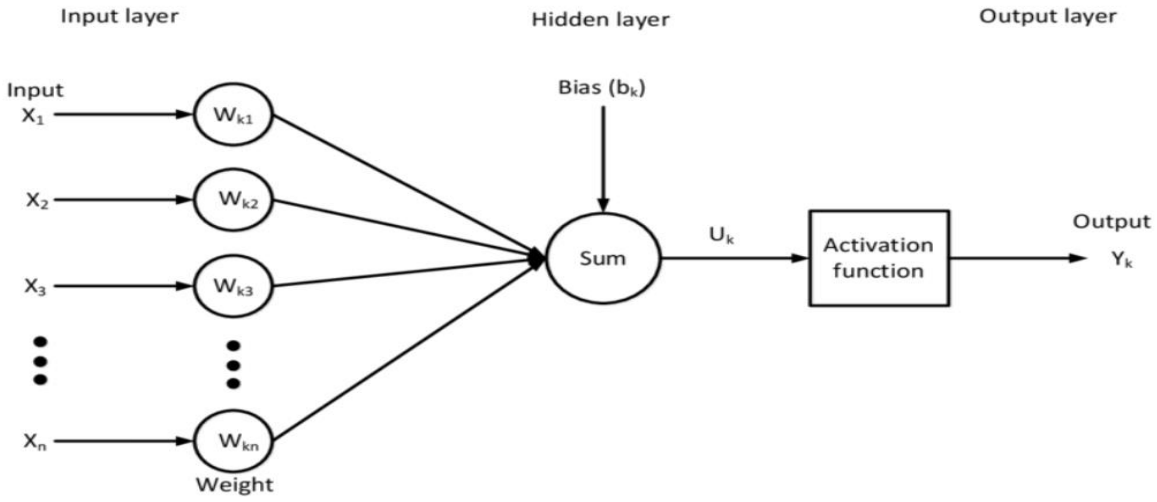


Figure 9 Source: Google's Public Domain Images Basic ANN Model

While ANNs are powerful, they have limitations:

Image and Speech Recognition Challenges due to large sized input eg;240*240 image input can be time consuming for computation.

Single input handling: ANN process a single input at a time lacking the ability to consider sequential dependencies.

To address these challenged Convolutional Neural Networks (CNNs) were introduced. They are specialized for spatial data, using convolutional layers to efficiently process images and capture hierarchies.

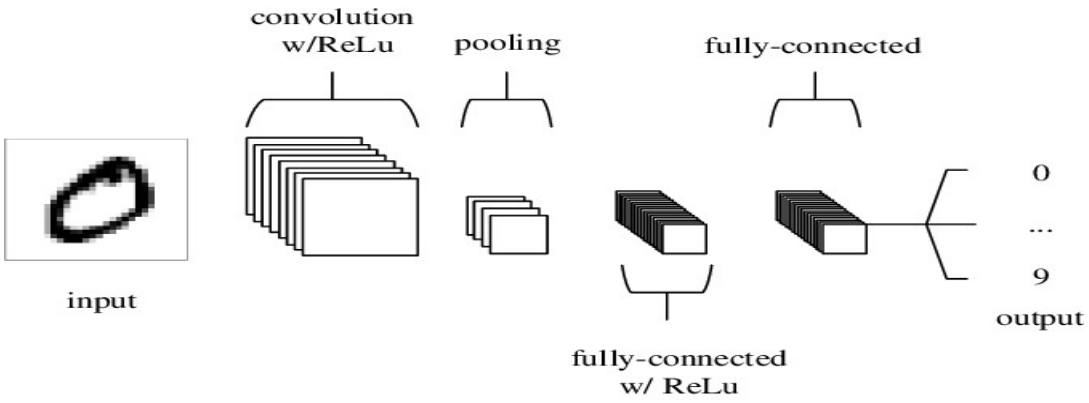


Figure 10 Source: ResearchGate.com , CNN architecture

Why RNNs are needed ?

Sequential data challenges: While CNNs are excellent for images , they lack memory and struggle with sequential data challenges by introducing a memory element. RNNs maintain a hidden state that carries information from previous inputs to influence predictions for the current input.

RNN can be visualized as unfolding over time, where each step corresponds to a moment in the sequence.

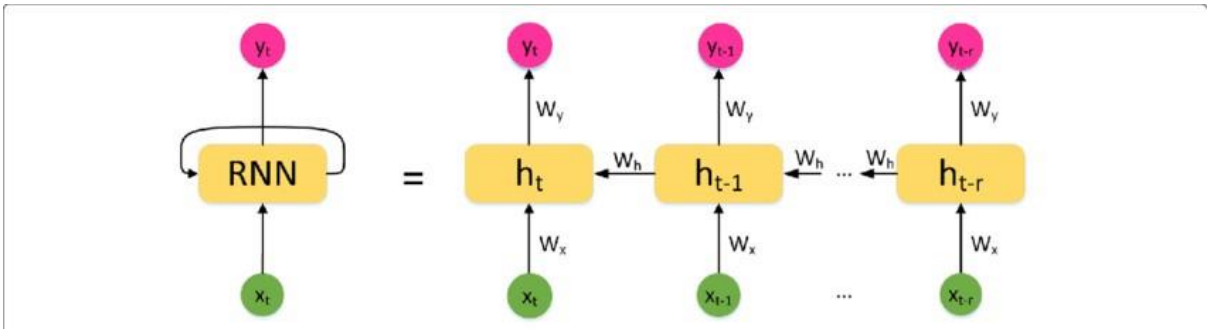


Figure 11.Source: ResearchGate.com RNN architecture

The output from one step is not only influenced by the current input but also by the memory from the previous step. This is achieved through weights that connect the hidden states across different time steps.

The essence is that RNNs extend the capabilities of ANNs by introducing memory , enabling them to handle sequential data. While ANNs and CNNs have their strengths. RNN are particularly useful in taking where the order of data points matters such as language modelling, time series analysis and sequential data processing.

How to address shortcomings of RNN ?

While traditional RNNs introduced the concept of memory, they faced challenges in retaining information over long sequences. The vanishing gradient problem often led to the loss of crucial information.

LSTMs were introduced to address these issues. They include memory cells, input gates, forget gates and output gates. Memory cells preserve information over long periods. Gates control the flow of information, deciding what to remember and what to forget.

There is a simplified version of LSTMs i.e. GRU (Gated Recurrent Units). GRUs include update and reset gates, similar to LSTMs, but with a more streamlined structure. GRUs aim to address the vanishing gradient problem more efficiently than traditional RNNs.

Vanishing gradient: Hard to learn

For example image your friend writes an important hint with a special ink that fades a bit each time they pass it back. If this ink fades too much, by the time the note reaches, the hint is almost invisible. This is like vanishing gradient problem.

Similarly in neural network, if the gradient(the learning signal) becomes very small as it goes through the layers, the network might not "see" the important information it needs to learn. It is like the knowledge is fading away before it reaches the early layers of the network.

Both LSTM and GRU were introduced to overcome the vanishing gradient problem. Their architecture allows for better preservation and controlled flow of information through long sequences.

LSTM and GRU are well-suited for tasks where understanding long-term dependencies is crucial, such as in language modelling, speech recognition, time series data analysis etc.

Below theoretical analysis provides better understanding of LSTM, GRU and Bi-directional LSTM, Bi-directional GRU which are implemented in this project for analysis.

7.4 Theoretical Principles Of LSTM And BI-LSTM

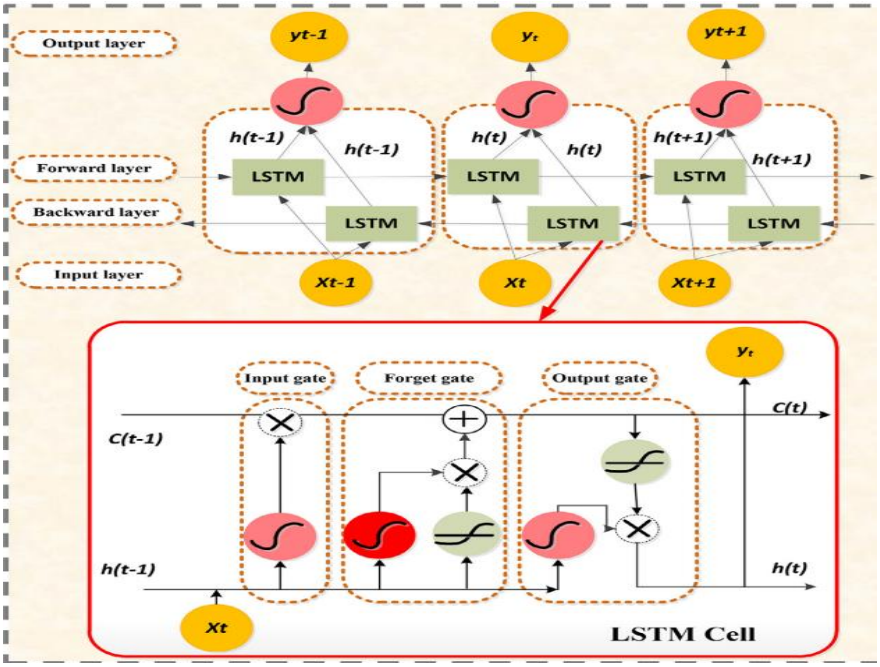


Figure 12 Architecture for single LSTM and Bi-LSTM cell

Source: F Shahid et al.,2020

Recurrent Neural Networks (RNNs) have been widely employed for processing sequential time series data with temporal dependencies. An unfolded RNN can leverage past data to process current information. However, traditional RNNs face challenges in training long-term dependencies. To address this limitation, the Long Short-Term Memory (LSTM) network was introduced as an advanced version of the RNN by Hochreiter et al.,1997. LSTMs use memory cells, equipped with self-connections, to store temporal states. These memory cells are regulated by three gates: the input gate, output gate, and forget gate. The input and output gates control the flow of information into and out of the memory cell, while the forget gate determines which information from previous steps is retained. The decision to store or pass information is based on the activation levels of input and output units. In essence, LSTMs compute the mapping between an input sequence **(X = X1, X2, ..., Xn)** and an output sequence **(y = y1, y2, ..., yn)** through a set of equations governing the behaviour of the gates and memory cell. The equations involve weights, bias variables, and activation functions, combining past and current information to produce the final output.

To overcome LSTM's limitation of not utilizing future information, Schuster et.,al 2006 proposed Bidirectional Recurrent Neural Networks (BRNN). BRNNs consist of two distinct LSTM hidden layers that process the input sequence in both forward and backward directions

simultaneously. This architecture enables the model to exploit information from both past and future contexts, enhancing its performance.

In Bi-LSTM, the input sequence is processed in the forward direction as

$$\rightarrow \mathbf{ht} = (\rightarrow \mathbf{h1}, \rightarrow \mathbf{h2}, ..., \rightarrow \mathbf{hn})$$

and in the backward direction as

$$\leftarrow \mathbf{ht} = (\leftarrow \mathbf{h1}, \leftarrow \mathbf{h2}, ..., \leftarrow \mathbf{hn})$$

The final output sequence y is formed by combining both → ht and ← ht, resulting in a richer representation that captures bidirectional dependencies. The architecture of a single LSTM cell and Bi-LSTM is illustrated in Figure 8.

This description is based on the work of F. Shahid et al.,2020 and their research in the field of recurrent neural networks and LSTM networks. The LSTM and Bi-LSTM architectures play a crucial role in various applications involving sequential data analysis.

7.5 Theoretical Principles Of BI-GRU

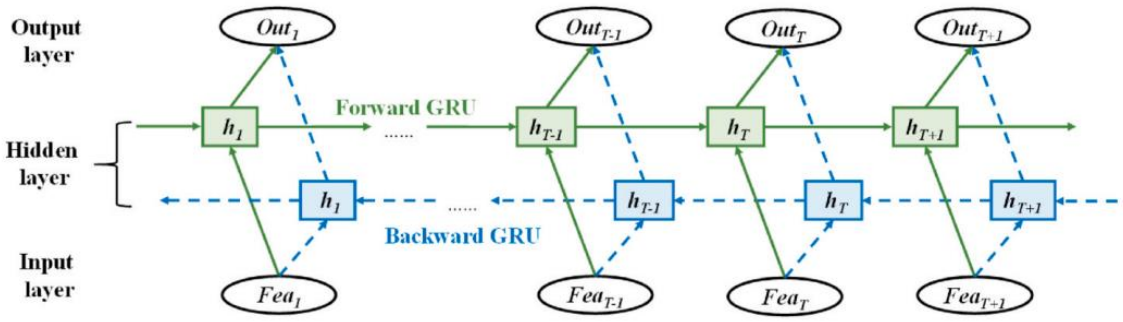


Figure 13 Bi-GRU Model illustration

Source: Qi.Wang et.,al 2018

The Bidirectional Gated Recurrent Unit (Bi-GRU) is introduced as an extension of the conventional Recurrent Neural Network (RNN) to address challenges in handling long-term dependencies within sequential time-series data. While RNNs can remember historical information, they face issues such as gradient vanishing and exploding gradients, limiting their application in sequences with extended dependencies. To overcome these limitations, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks were developed. LSTM introduces memory cells and three essential gates (forget, input, and output) to manage information flow, enhancing the model's ability to capture long-range dependencies. Similarly, GRU employs update and reset gates to alleviate gradient dispersion, allowing for long-term memory retention with reduced computational complexity.

The update gate (zT) in GRU determines the extent to which previous information is retained in the current forecast. It is defined as

$$\mathbf{zT = \sigma(Wz \cdot [hT-1, FeaT] + bz),}$$

where σ represents the sigmoid activation function, FeaT is the input matrix at timestep T, hT−1 is the previous hidden state, and Wz and bz are the weight and bias matrices of the update gate.

The reset gate (rT) controls the extent to which historical information is ignored, described as

$$\mathbf{rT = \sigma(Wr \cdot [hT-1, FeaT] + br)},$$

where Wr and br represent the weight and bias matrices of the reset gate.

The candidate hidden state is computed as

$$\mathbf{\tilde{}hT = tanh(Wh \cdot [hT-1 \odot rT, FeaT] + bh),}$$

where tanh is the hyperbolic tangent activation function, and Wh and bh are the weight and bias matrices.

The output (hT) is calculated as a linear interpolation between hT and hT−1:

$$\mathbf{hT = (1 - zT) \odot hT-1 + zT \odot \tilde{}hT}$$

To enhance the modelling of daily production forecasting, the Bi-GRU algorithm is introduced, allowing for knowledge extraction from both forward and backward directions of the input sequence. This bidirectional approach combines the outputs of the forward and backward GRUs, enabling the capture of dependencies in both past and future information. The Bi-GRU can be expressed as

$$\mathbf{YT = F(h\rightarrow T, h\leftarrow T)},$$

where h→T and h←T represent the hidden states of the forward and backward GRUs, and F represents the combining method, such as multiplication, average, or summation, to incorporate both directional outputs effectively.

7.6 Best Deep Learning Architecture Implemented In This Project

Combination of Bi-LSTM + Bi-GRU

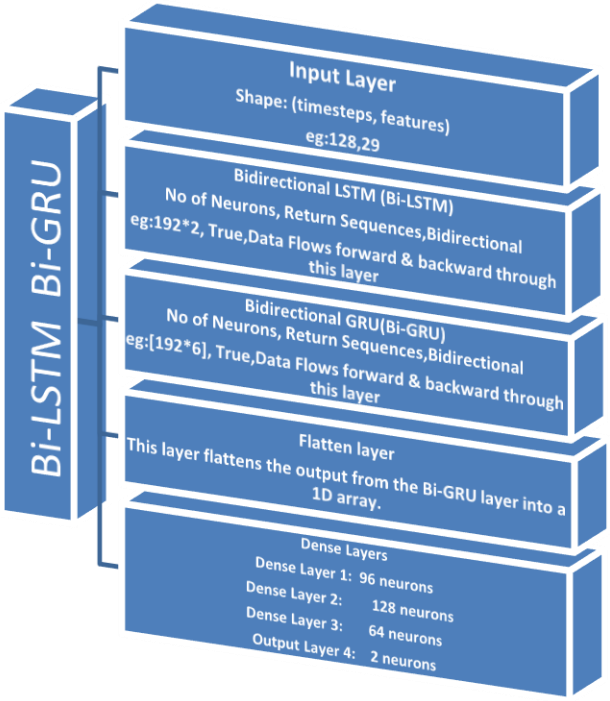


Figure 14 Architecture of Bi-LSTM followed by Bi-GRU, created using MS Word SmartArt

The model uses a Bidirectional LSTM (Bi-LSTM) to analyse data both forward and backward in order to find sequential patterns. The Bi-LSTM output is subsequently processed by a Bidirectional GRU. The Flatten layer reduces the output to a 1D array, which is then processed by Dense layers.

The neural network is more adept at deciphering intricate linkages and patterns in sequential data. To maximize the performance of the model for a particular time series forecasting task, the number of neurons and other parameters can be changed.

Data Configuration

```
data_config = {
    "timesteps": 128,
    "batch_size": 128,
}
```

This specifies the time series data's configuration options, including *batch size* and the number of *timesteps*.

Number of *timesteps*: Applicable to time series and other sequential data when the number of steps in the sequence is known. This affects the model's long-term capacity to identify trends (Gao S et.,al 2021).

The quantity of data samples handled in a single training cycle is referred to as the *batch size* Stable updates are produced by larger batch sizes, but they demand more RAM. Computational efficiency and training stability are impacted by the decision (Devarakonda et al., 2018).

Timeseries Dataset Function:

```
train_ds = timeseries_dataset_from_df(train, data_config["batch_size"])
val_ds = timeseries_dataset_from_df(val, data_config["batch_size"])
```

This method comes from a package that makes it easier to convert Pandas Data Frames into time series datasets (Molin et al., 2021, p. 241).The data is structured in this way so that it can be used to train time series model. By using a certain amount of historical *timesteps* as input features and projecting future time steps as the output, the function arranges the data into input-output pairs.

The terms "train_ds" and "val_ds" denote the variables that correspond to the *training* and *validation* datasets. The nature of these datasets may allow the model to generalize to previously unidentified data during *training* and *validation* by allowing it to learn from past trends.

As shown by the use of batches ('data_config["batch_size"]'), the model is trained on subsets of the dataset at a time as opposed to the entire dataset at once. This is common approach for memory and efficiency concerns. The *batch_size* is one hyperparameter that regulates the number of samples processed in each training cycle.

Model Configuration:

```
model_config = {"n_epochs": 30, "n_neurons": 192 * 2, "dropout": 0.4, "stateful": Fa
```

Specifies the neural network model's setup settings, including the number of *epochs, neurons, dropout rate* and *statefulness* of the model.

n_epochs:30-The number of *epochs*, or the number of times the whole dataset will be fed into and withdrawn from the neural network during training (Brownlee et al., 2018), is indicated by this parameter. This instance will include 30 *epochs* of training for the model.

n_neurons: 192*2- This parameter indicates how many units, or *neurons*, there are in a neural network layer. The amount of *neurons* in a layer affects the model's ability to recognize intricate patterns. More *neurons* can help the data capture more complex correlations, but they may also raise the possibility of overfitting.
dropout: 0.4- To avoid overfitting in neural networks, *dropout* is a *regularization* approach. During training, it arbitrarily changes a portion of the input units to zero.
stateful: False: The *stateful* parameter in RNNs controls whether the model's internal state is reset at the conclusion of each epoch. Establishing "*stateful*" If this is false, then there won't be a preservation of the internal state across *epochs*.

Model architecture:

```
input1 = Input(shape=(data_config["timesteps"], len(XCOLS)), name='x1')
lstm1 = Bidirectional(LSTM(model_config["n_neurons"], dropout=model_config["dropout"
gru1 = Bidirectional(GRU(model_config["n_neurons"] * 3, dropout=0.0, return_sequence

gaverage = Flatten()(gru1)
dense1 = Dense(96)(gaverage)
dense1 = Dense(128)(dense1)
dense1 = Dense(64)(dense1)
dense = Dense(2)(dense1)

model = keras.models.Model(inputs=input1, outputs=dense)
```

Builds an input layer for a neural network with
shape=(data_config[“timesteps”],'len(XCOLS)' denoting the quantity of features.
The Bi-LSTM and Bi-GRU layers are specified by the number of *neurons*, *regularization dropout rate*, and *return sequence*.

Next, a *flattening layer* is used to convert *the 3D tensor output* from the GRU layer into a *1D tensor*. The following three *Dense layers* have different amounts of neurons in them. In each layer, *Rectified Linear Units* (ReLUs) are turned on by default.
Next, the last *Dense* layer of the neural network is defined, including two neurons. The thick layers support a non-linear type of modification of the properties retrieved by the recurrent layers. A *Keras* model is created by specifying the input and output layers.

Model Compilation and Summary :

```
model.compile(
    loss='mean_squared_error',
    optimizer=tf.keras.optimizers.Adam(0.0001),
)

model.summary()
```

The model is constructed using *Mean Squared Error* and the *Adam optimizer*.

The *Adam optimizer* is a popular optimisation technique with a *learning rate* of 0.0001. The learning rate influences the step size during optimisation. Details about the model's layers, output formats, and parameter count are displayed in a printed summary.

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 x1 (InputLayer)             [(None, 128, 29)]         0

 bidirectional_2 (Bidirecti  (None, 128, 768)          1271808
 onal)

 bidirectional_3 (Bidirecti  (None, 128, 2304)         13284864
 onal)

 flatten_1 (Flatten)         (None, 294912)            0

 dense_4 (Dense)             (None, 96)                28311648

 dense_5 (Dense)             (None, 128)               12416

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 2)                 130

=================================================================
Total params: 42889122 (163.61 MB)
Trainable params: 42889122 (163.61 MB)
Non-trainable params: 0 (0.00 Byte)
```

Training the Model:

```
history = model.fit(
    train_ds,
    batch_size=data_config["batch_size"],
    epochs=model_config["n_epochs"],
    verbose=2,
    callbacks=[checkpoint, lr_reducer],
    shuffle=False,
    validation_data=val_ds,
)
```

```
Epoch 1/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1703s - loss: 400.0068 - val_loss: 374.8605 - lr: 1.0000e-04 - 1703s/epoch - 8s/step
Epoch 2/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1690s - loss: 342.7743 - val_loss: 273.0658 - lr: 1.0000e-04 - 1690s/epoch - 8s/step
Epoch 3/30
224/224 - 1667s - loss: 144.6048 - val_loss: 139.7158 - lr: 1.0000e-04 - 1667s/epoch - 7s/step
Epoch 9/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1680s - loss: 140.2694 - val_loss: 137.3792 - lr: 1.0000e-04 - 1680s/epoch - 8s/step
Epoch 10/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1685s - loss: 134.0841 - val_loss: 132.2754 - lr: 1.0000e-04 - 1685s/epoch - 8s/step
Epoch 11/30
```

Using both training and validation data, the model is trained via the *"fit"* approach. During training, *callbacks* such as *ReduceLROnPlateau* and *ModelCheckpoint* are utilised to save the optimal model and modify the learning rate.

In the context of training neural networks, *callbacks* are functions that may be executed at different points in the training process. These functions enable the performance of activities like model saving, learning rate adjustment, and training termination depending on predefined criteria.

"ModelCheckpoint" and *"ReduceLROnPlateau"* are two often used *callbacks* that help make training more successful and efficient.

7.7 Metrics: Root Mean Square Error (RMSE)

The root mean squared error (RMSE) is the square root of the mean of the squared differences between predicted and actual values. RMSE is widely used as an error metric for assessing performance in regression analysis (Neill et al.,2018). The formula for RMSE is as follows,

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2}$$

with Oi representing observations, Si representing predicted values, and n representing the number of observations used in the analysis.

```
test_ds = timeseries_dataset_from_df(test, data_config["batch_size"])
mse = model.evaluate(test_ds)
print(f"Test RMSE: {mse**.5:.2f}")
```

Models implemented are analysed using RMSE performance metrics.

8 Results

This section comprises three subsections that analyse the outcomes of exploratory data analysis and the prediction error values derived from deep learning models respectively.

8.1 Exploratory Data Analysis

8.1.1 Data Description

Three distinct datasets containing time-series data are selected as features.

- **Data on solar wind obtained from ACE and DSCOVR satellites**
  The main feature dataset is available in this, containing solar wind measurements obtained from both the ACE and DSCOVR satellites which has 1048576 instances and 17 features.
- **Sunspot counts that have been subjected to a smoothing process**
  The Sun experiences a solar cycle, which is a recognised cyclic pattern that lasts for around 11 years and is characterised by variations in the number of sunspots on its surface. Significant geomagnetic storms often occur most frequently at the apex of these solar cycles. Smoothed sunspot numbers are available, which enables model modifications based on solar cycle. Monthly access to sunspot counts that have been smoothed is available. In dst_labels.csv, sunspots are classified according to how aligned they are with the corresponding day. 3 features and 82 instances make up the file.
- **DST values**
  The labels represent hourly DST values and are organized with a multi-index based on the period and timedelta. The file has 59185 instances and 3 features.

8.1.2 Summary Statistics

In this section, key statistical characteristics of dataset, aimed at providing a clear overview of its central tendencies and variability. For numerical variables, fundamental statistics such as the mean, median, standard deviation, as well as the minimum and maximum values are obtained. These statistics shed light on the typical value, dispersion, and range of our numerical data. Understanding these summary statistics is crucial for gaining insight into the dataset's general properties and can guide onto subsequent data analysis and modelling efforts. The following tables present a comprehensive picture of these statistical aspects for dataset.

	dst						
	count	mean	std	min	25%	50%	75%	max
period								
test_a	25560.0	-20.139710	26.814563	-422.0	-30.0	-16.0	-4.0	63.0
test_b	21168.0	-10.981623	17.856649	-147.0	-18.0	-8.0	0.0	77.0
test_c	12456.0	-5.634875	11.096726	-59.0	-11.0	-4.0	2.0	40.0

Figure 15 Summary Statistics of DST data

The big dataset includes more than 140,000 hourly observations of the DST index spanning more than 15 years. Notably, the test_b and test_c periods have almost twice as many observations as the test_a period. Furthermore, the test_a period appears to correspond with a period of greater geomagnetic activity based on its lower mean DST value and larger standard deviation. The majority of these DST values are negative, indicating large disturbances to the magnetic field. These disruptions are often directed towards Earth and are measured in nanoTeslas (nT). In calm conditions, DST values often hover around or slightly below the zero line, while they can sometimes occasionally be extremely positive.

	period	test_a	test_b	test_c
bx_gse	count	1.483618e+06	1.241728e+06	7.374180e+05
	mean	-1.242373e+00	4.468653e-02	-3.303193e-01
	std	4.907424e+00	3.288608e+00	2.777060e+00
	min	-4.348000e+01	-2.465000e+01	-1.731000e+01
	25%	-4.930000e+00	-2.430000e+00	-2.550000e+00
...	...	...	...	...
temperature	min	1.000000e+04	1.000000e+04	2.000000e+03
	25%	6.009200e+04	3.716000e+04	2.588000e+04
	50%	1.102600e+05	6.480100e+04	4.596900e+04
	75%	1.839400e+05	1.132320e+05	9.232050e+04
	max	5.675500e+06	6.158688e+06	4.076898e+06

Figure 16 Summary Statistics of Solar Wind Data

The test_a period's mean and standard deviation of DST values clearly show a greater degree of geomagnetic strength than the other two periods, as would be predicted.

It's also important to keep in mind that the dataset includes numbers on a variety of scales. For example, the temperature values are often greater than hundreds of thousands of Kelvin.

The Interplanetary Magnetic Field (IMF) measurements, on the other hand, are primarily negative and much less in magnitude.

	period	test_a	test_b	test_c
smoothed_ssn	count	35.000000	29.000000	17.000000
	mean	134.025714	82.458621	3.676471
	std	39.084857	13.670237	1.295439
	min	71.000000	45.700000	1.800000
	25%	94.450000	84.300000	2.700000
	50%	136.000000	86.300000	3.500000
	75%	173.200000	88.100000	4.700000
	max	180.300000	98.300000	6.200000

Figure 17 Summary Statistics of Sunspots Data

The average and level of variation in the data within the "test_a" period tend to be higher compared to the other two periods.

8.1.3 Data Distribution

This session discusses the data distribution analysis of some of features in the dataset.

i. Distribution of DST across periods

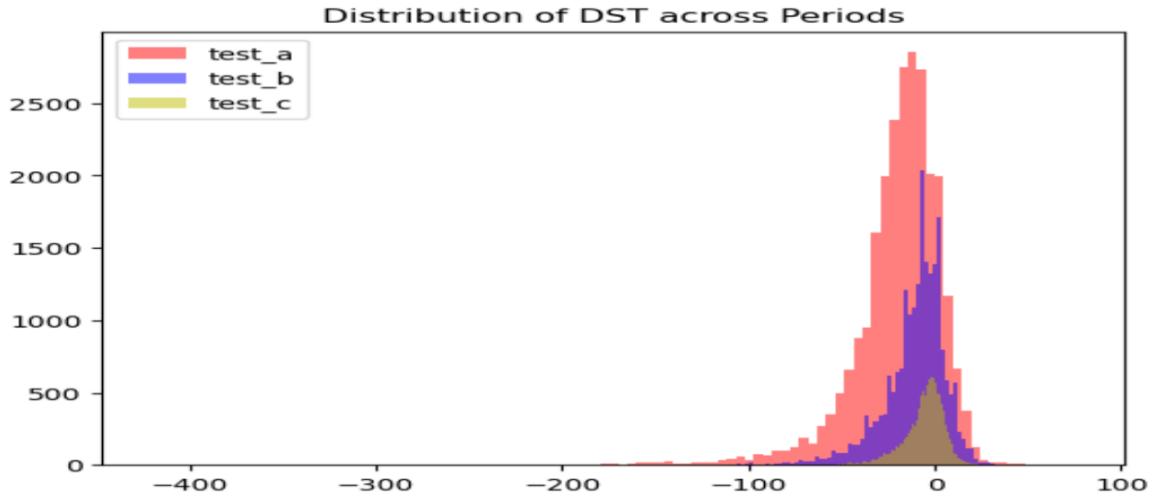


Figure 18 DST across three periods a, b, c

test_a clearly has the most intensity among the times, with test_b close behind. This is to keep this distribution difference in mind while performing data partitioning.

ii. Distribution of few of solar wind features

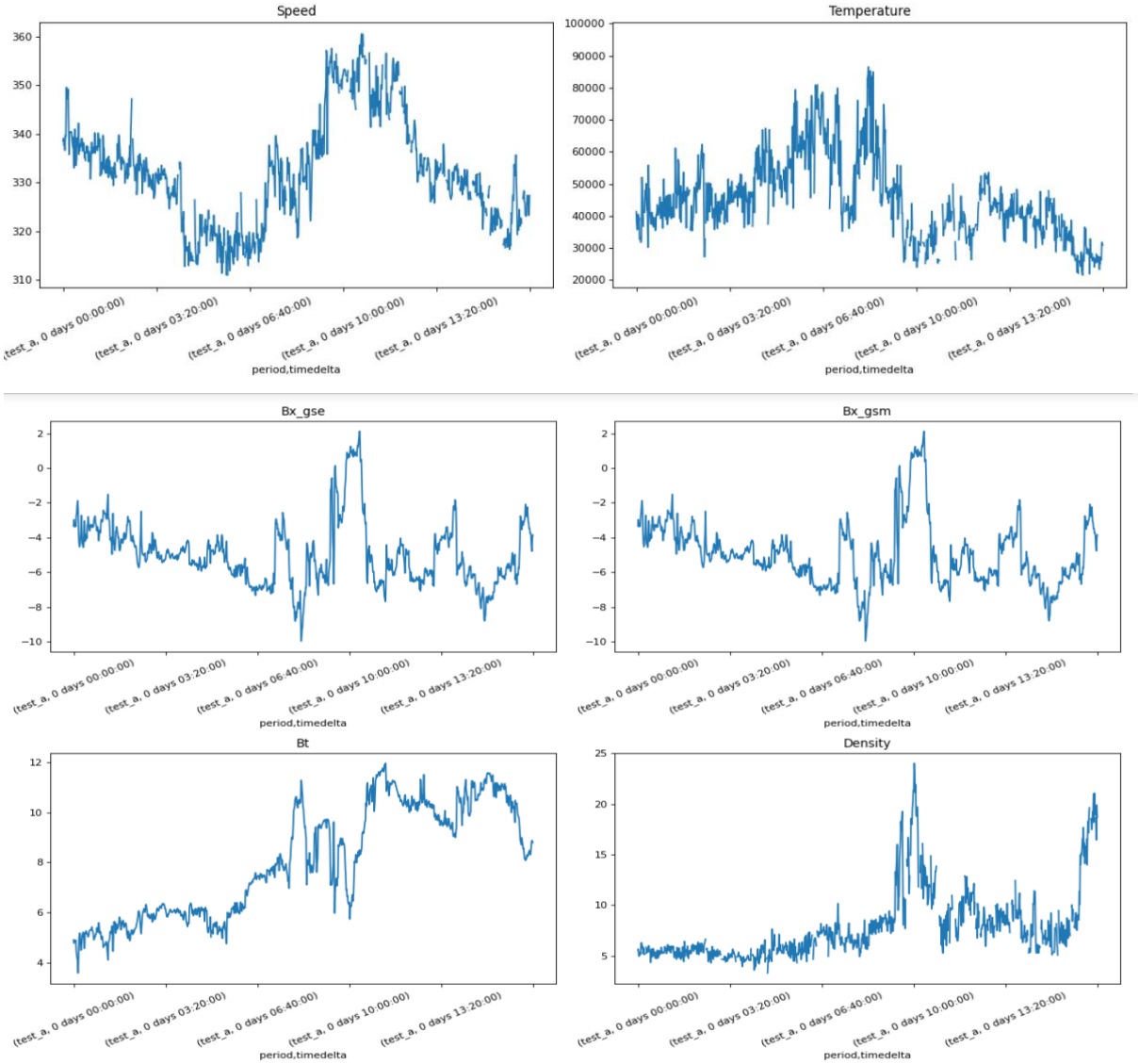


Figure 19 Different Solar wind features plots

The visualizations clearly reveal the presence of gaps in the data. The proportion of missing values differs across features, indicating that each feature will need some form of imputation. Evidently, sensor readings from space can be unreliable due to various space weather disturbances. Hence there is a to devise a reasonable strategy for handling these gaps. Additionally, it's worth noting that the IMF features bx_gsm and bx_gse exhibit a strong correlation. This might lead to multi-collinearity problems if both features are included in model.

8.1.4 Missing Data and Strategies For Handling Missing Data

```
solar_wind_data.isna().sum()
```

```
bx_gse           88276
by_gse           88276
bz_gse           88276
theta_gse        88276
phi_gse          88276
bx_gsm           88276
by_gsm           88276
bz_gsm           88276
theta_gsm        88276
phi_gsm          88276
bt               88276
density         264734
speed           270811
temperature     307377
source           78265
dtype: int64
```

```
dst_data.isna().sum()
```

```
dst    0
dtype: int64
```

```
sunspots_data.isna().sum()
```

```
smoothed_ssn    0
dtype: int64
```

While the degree of missing data varies among features, each feature necessitates some form of imputation. It appears that sensor readings from space can be inconsistent due to the influence of unpredictable space weather conditions. Hence, it is necessary to devise a practical approach for addressing these missing values.

In this project two different ways are implemented to handle missing data. The methods are specified using 'fillna' function.

```
solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="pad")
```

- 'fillna(method="pad")': This method fills missing values with the previous valid value along each column (forward fill). It's also known as "ffill". If there are consecutive missing values in a column, this method will replace them with the most recent valid value in that column.

```
solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="bfill")
```

- 'fillna(method="bfill")': This method fills missing values with the next valid value along each column (backward fill). It's also known as "backfill". If there are consecutive missing values in a column, this method will replace them with the next valid value in that column.

8.1.5 Data Relationship And Its Significance

In order to understand the links between the characteristics in the dataset, a correlation matrix has been utilised. This method of analysis is quite helpful in identifying relationships and dependencies between variables, which helps to clarify how they affect each other. It facilitates the finding of significant insights into the direction and extent of these relationships, enabling the identification of possible patterns and dependencies that can enhance comprehension of the dataset.

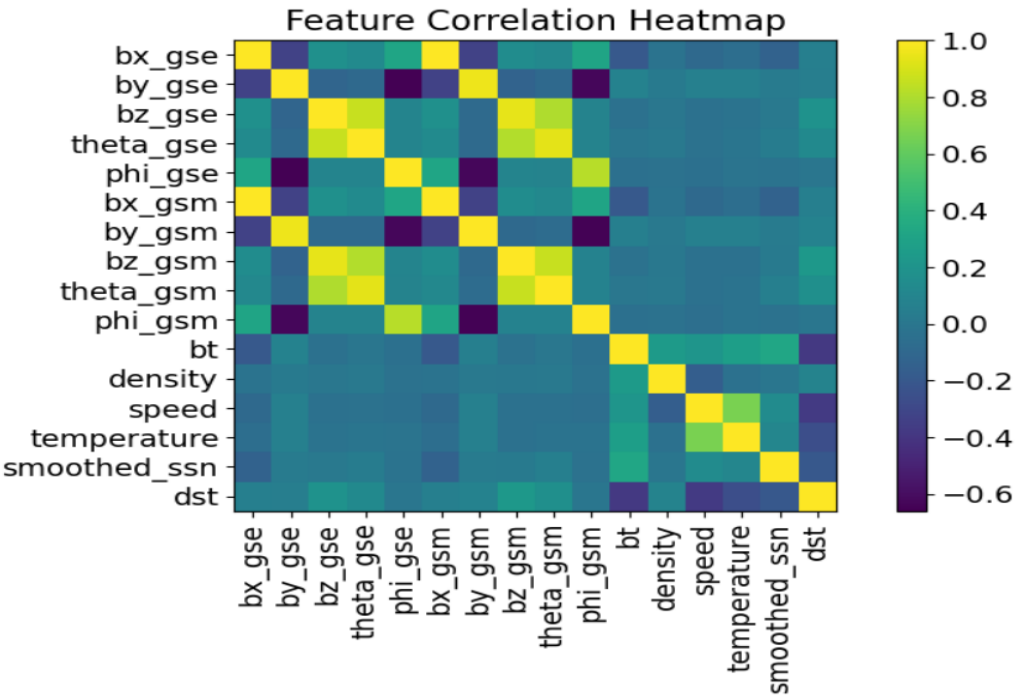
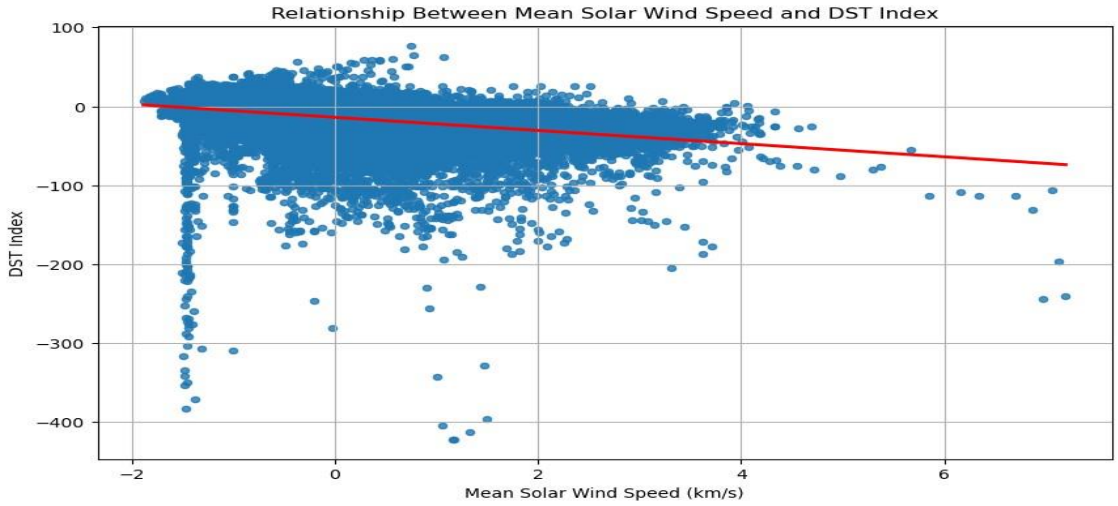


Figure 20 Correlation Heatmap showing features significance

From the heatmap it is observed that plasma related features such as speed and temperature, exhibit a pronounced inverse relationship with DST variable. Similarly IMF variable ‘bt’ also demonstrates a strong negative correlation with DST. As expected, the variables 'gsm' and 'gse' exhibit a significant positive correlation.

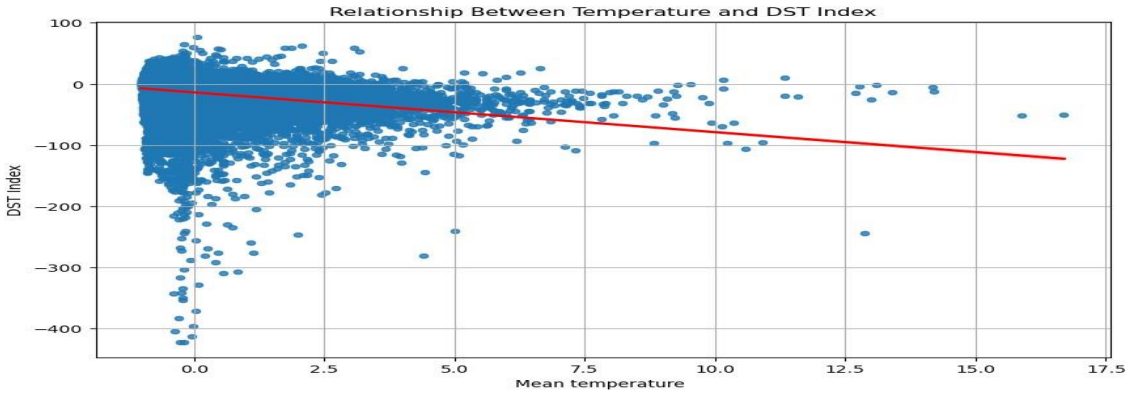
What exactly these features means with respect to DST ?

Speed Vs DST



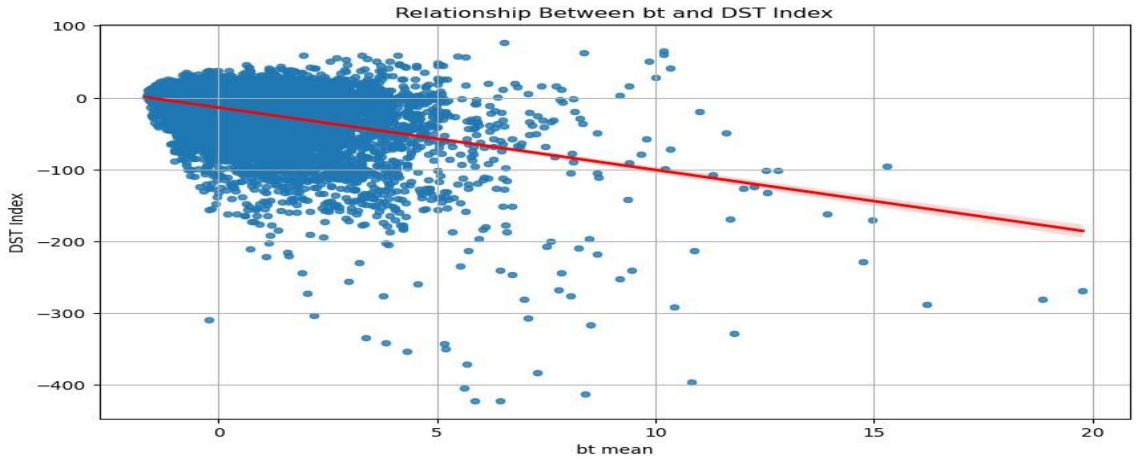
Understanding solar wind and DST is essential to comprehending geomagnetic storms. Earth's magnetic field interacts with charged particles that flow from the Sun, known as solar wind. Fast solar wind has the ability to compress the magnetosphere, which can lead to magnetic reconnection and disruptions. The DST index is used to measure the intensity of these disruptions, which take the form of geomagnetic storms (Gonzalez, 2022). Stronger storms are indicated by more negative DST values. The energy exchange that occurs during solar wind-magnetosphere interactions is the source of the correlation. A major factor in severe storms is the high-speed solar wind that is linked to coronal mass ejections, or CMEs. Solar wind speed monitoring enables the real-time forecast of geomagnetic storms, which is crucial for reducing the impact on technological systems that are susceptible to geomagnetic disturbances.

Temperature Vs DST



The relationship between temperature and DST is not direct, as temperature variations on Earth's surface are primarily influenced by atmospheric and terrestrial processes, while DST reflects geomagnetic storm activity. Temperature changes are driven by factors such as sunlight, atmospheric pressure, and local weather patterns. While there may be atmospheric effects indirectly associated with space weather events, the direct relationship between surface temperature and DST is not a well-established or straightforward correlation. Researchers typically study these phenomena separately within the domains of space weather and atmospheric science.

bt Vs DST

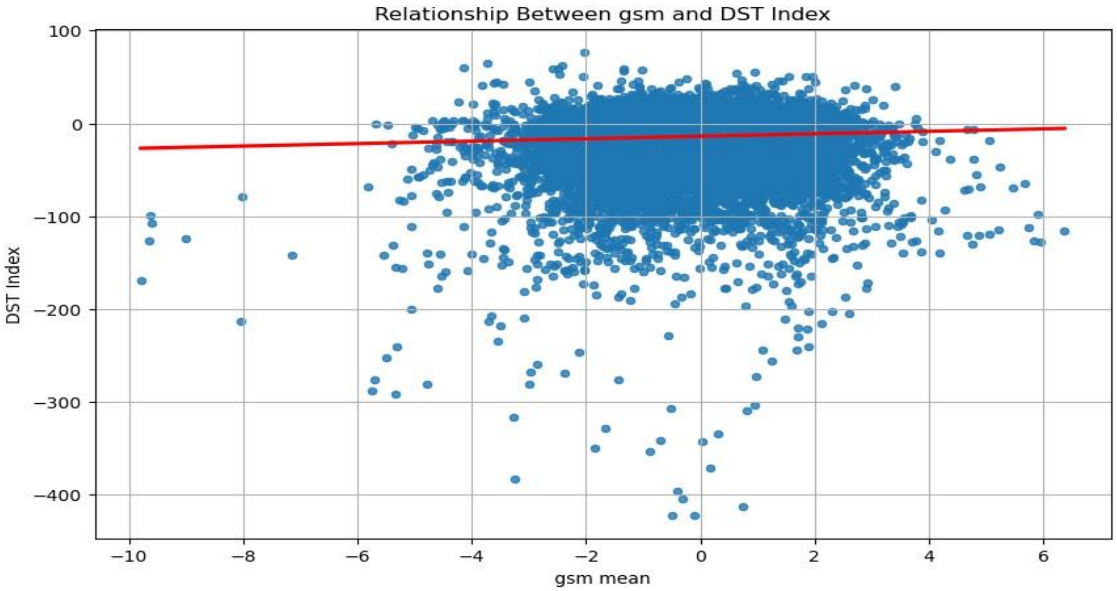


The relationship between the magnitude of the interplanetary magnetic field (bt) and the DST is a key aspect of space weather dynamics. The solar wind, a continuous stream of charged particles flowing from the Sun, carries with it the IMF. When the solar wind encounters the Earth's magnetosphere, interactions between the IMF and Earth's magnetic field can lead to geomagnetic storms, influencing the DST.

Specifically, during periods of enhanced solar wind activity, characterized by higher bt values, the likelihood of geomagnetic disturbances increases. If the IMF has a southward orientation (opposite to Earth's magnetic field), it can more effectively connect and transfer energy to the magnetosphere, potentially leading to geomagnetic storms (Gonzalez, 2022). These storms, in turn, are reflected in changes to the DST.

In summary, elevated bt values, particularly when combined with a southward IMF orientation, are indicative of conditions that can trigger geomagnetic storms, affecting the DST.

gsm Vs DST

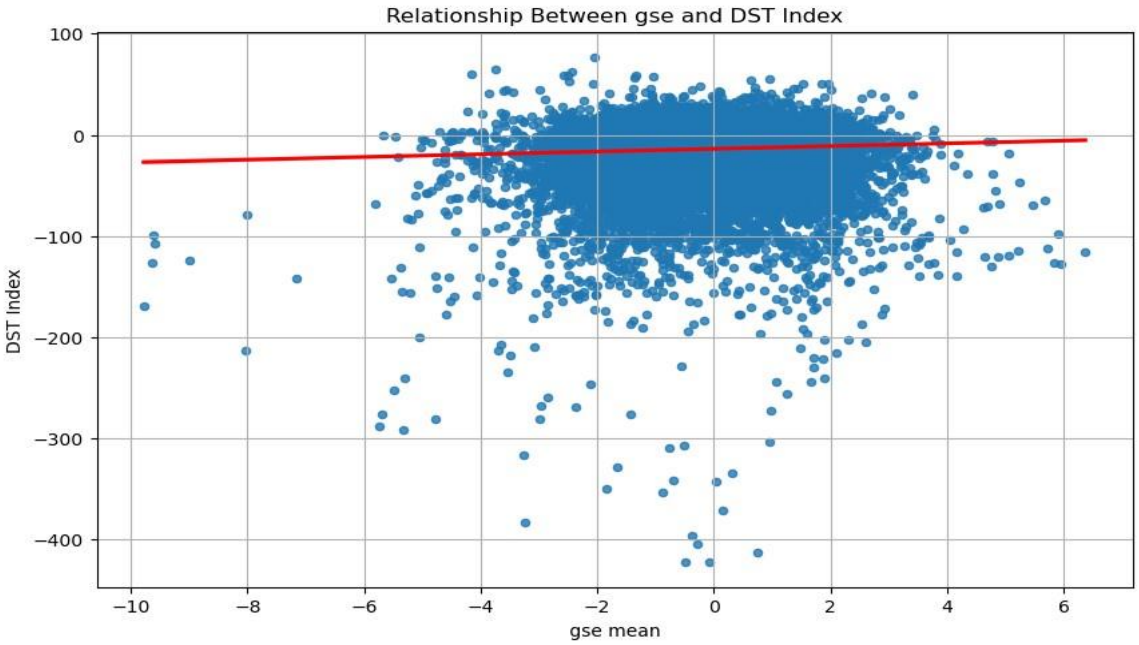


The term "GSM" typically refers to the Geocentric Solar Magnetospheric coordinate system, which is a coordinate system used to describe the position of objects in the Earth's magnetosphere. The components "bx_gsm", "by_gsm" and "bz_gsm" represent the magnetic field components in the GSM coordinate system. Understanding the relationship between these magnetic field components and DST involves considering their impact on geomagnetic activity.

In the context of space weather, variations in the magnetic field components, especially when the IMF has a southward orientation in the GSM system, can influence the Earth's magnetosphere. A southward IMF facilitates energy transfer from the solar wind to the magnetosphere, potentially triggering geomagnetic storms. These storms, characterized by disturbances in the Earth's magnetic field, are reflected in changes to the DST (Gonzalez, 2022).

Therefore, monitoring variations in the "gsm" magnetic field components provides insights into the potential for geomagnetic disturbances and their subsequent impact on the DST. Studying this relationship helps in understanding the dynamics of space weather and improving predictions of geomagnetic storm activity.

gse Vs DST



The relationship between GSE coordinates and the DST involves understanding the impact of solar wind parameters on geomagnetic activity. GSE coordinates represent a coordinate system fixed with respect to the Sun and centred at the Earth, providing a framework to describe the position of objects in the solar system.

The GSE components, such as "bx_gse", "by_gse" and "bz_gse" represent the solar wind's magnetic field in the GSE coordinate system.

The relationship with DST is often characterized by the response of the Earth's magnetosphere to variations in the solar wind. When IMF carried by the solar wind has a southward orientation (negative bz_gse), it enhances the connection and energy transfer from the solar wind to the Earth's magnetosphere (Gonzalez, 2022).This, in turn, can lead to geomagnetic storms, which are reflected in changes to the DST.

8.1.7 Time-Period-based Data Splitting for Model Training and Evaluation

On the basis of predetermined hourly *timesteps* each period, the dataset has been divided into *training*, *validation*, and *test* sets. The creation of resilient models that take changes over time into account requires this temporal division. The distribution of hourly *timesteps* for training, validation, and test sets over various time periods is shown in the plot above. This might be thought of as a workable method for time-based data splitting, facilitating efficient model assessment and training in situations where temporal dynamics are important.

8.1.8 Potential Limitation of dataset's features

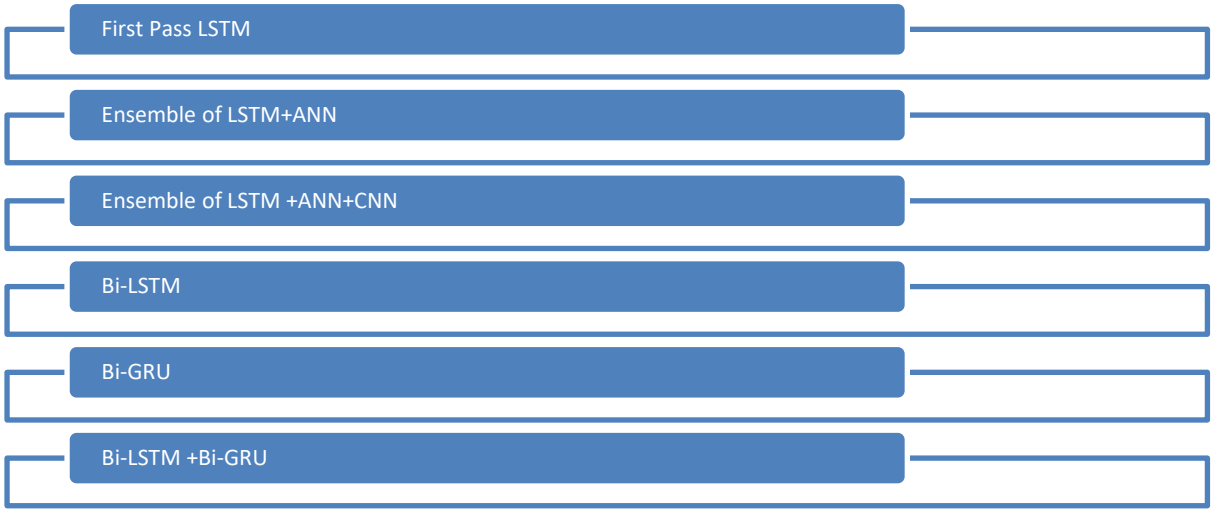
i. Features exist across different scales: One limitation to be aware of is that the dataset contains features with varying scales. This can be addressed using the StandardScaler, it's essential to acknowledge that scaling might not always be a one-size-fits-all solution. Depending on specific dataset and problem, different scaling methods, such as MinMaxScaler, may yield better results.

ii. Features are provided at different frequencies: A limitation to consider is the difference in feature frequencies. Aggregating data to a common frequency, like hourly in this case, simplifies the analysis. However, the choice of frequency and aggregation method can impact model performance. Experimentation with different frequencies and aggregation strategies is recommended.

iii. Certain IMF features are highly correlated: This can be addressed by selecting a subset of variables. It's important to note that this sub-setting approach is somewhat ad-hoc. A more principled or data-driven method for variable selection may be necessary for complex datasets.

iv. Many Missing Values: Dealing with missing values is a common challenge. This is handled using ffill and bfill imputation methods. Furthermore, it is better to explore the possibility of developing a separate model to estimate missing values, though it adds complexity.

8.2 Analysis Of Deep Learning Models



Models analysed in this project

Several experiments were carried out using LSTM(Long Short Term Memory, ANN(Artificial Neural Network, GRU(Grated Recurrent Unit), CNN (Convolutional Neural Network) as well.All experimented models utilized the Root Mean Squared Error (RMSE) as the primary metric to evaluate prediction error. The objective revolved around minimizing the RMSE value, aiming for the most accurate predictions.

Integrated Data Processing Flow followed for all models

Data Pre-processing	Feature Engineering	Data Integration	Data Splitting	Data Visualization
Solar wind & Sunspots data were processed ,including hourly data, scaling using Standard Scaler and imputing missing values	Various features were used including magnetic field components, angles, solar wind speed , ssn_number	Features were combined into a single dataset for further processing.	Split into train , test and validation sets , based on time periods ensuring that data from same period did not overlap between training , testing and validation sets .	DST plot , Correlation matrix , Feature analysis plot Results visualization , Training history plot or learning curve

The above mentioned iterative flow is instrumental for experimenting with different deep learning models in this project. Encompassing robust data pre-processing, feature engineering, integration and strategic dataset splitting. Insights derived from visualizations aid in refining models, contributing to an iterative experimentation process.

Model 1 –First Pass LSTM

1ST experiment was performed with a **first pass LSTM** . The data was prepared by dividing it into sequences and batches using a *timesteps* parameter to set the sequence length and a '*batch_size*' to determine the number of samples to process before updating the model's parameters. A custom function was developed *timeseries_dataset_from_df* which iterated through periods, aligning features and labels for sequences and using *Keras timeseries_dataset_from_array* to create datasets for each period. These dataset were concatenated to create the final training and validation sets.

LSTM model was designed using *Keras* with one 'LSTM layer' followed by a *dense* output layer. *Hyperparameters* such as *n_epochs, n_neurons, dropout and stateful* were set and the model was compiled using the *mean squared error* as the loss function and the *Adam* optimizer. The model was trained using the training dataset via *model.fit()* method specifying batch sizes, epochs and the *validation* dataset to monitor *validation loss*.

Upon completion of training, the model was evaluated using the test dataset and *RMSE* were calculated to assess its performance. The achieved test *RMSE* was '**14.24 nT'**, indicating relatively accurate predictions.

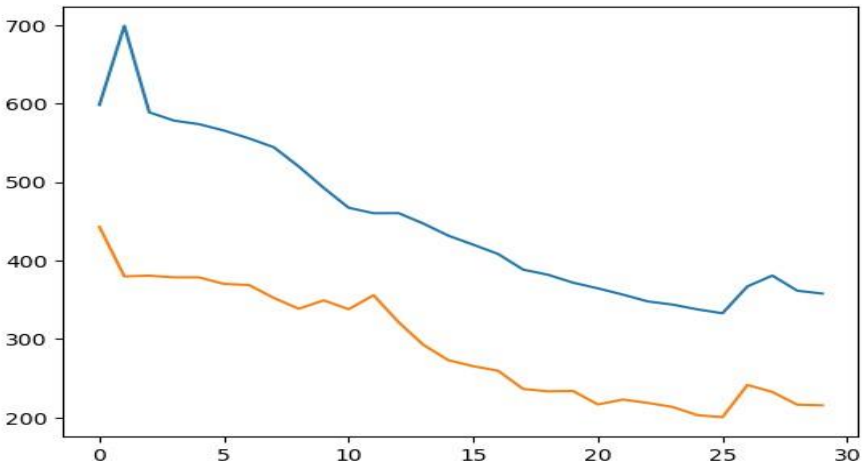


Figure 21 Loss Vs Epochs, Learning Curve LSTM Model

Inference from Learning Curve LSTM

The above *Learning Curve* is a visual representation of how certain metrics such as *loss* change over *epochs* during the training of the model. The x-axis represents the number of *epochs* and the y axis represents the corresponding *loss*. There is a decreasing trend in the 'loss' plot, indicating that the model is learning and improving its performance on the training data. The similar trend is observed in validation loss suggesting model is not over-fitting and generalizing well to unseen data.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	2layers-LSTM , Dense (fully connected) layer
Units per layer	n_neurons:192*2
Learning rate	Adam optimizer 0.001
Activation Function	tanh, linear
Dropout rate	0.4
Batch size	128
Epochs	30
RMSE	14.24 nT

Model & Hyperparameters used for First pass LSTM model

Model 2- Ensemble of ANN & LSTM

2nd experiment has been carried out using ensemble of ANN(Artificial Neural Network) and LSTM .The data was split into training and validation sets. ANN was built and trained using the training data over *30 epochs*, simultaneously an LSTM model was constructed and trained using the same training data, also for 30 *epochs*. Predictions were generated from both models for the validation set and combined by averaging the results. RMSE was calculated to measure the ensemble predictions and yielded a better value of **11.92 nT.**

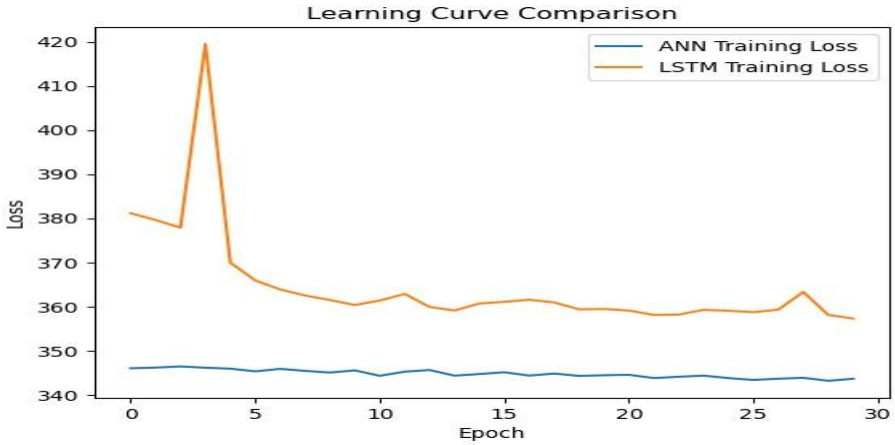


Figure 22 ANN-LSTM Learning Curve

Inference from ANN-LSTM Learning Curve

A flat decreasing curve for ANN depicts that the model's performance is not improving better over time. LSTM curve shows different stage of training process, in the beginning, the model is learning and improving its understanding of data. The increase in the learning curve is a positive sign, indicating that the model is capturing relevant patterns. The sharp point might indicate a point of overfitting.

The model has learned the training data well, possibly even memorizing it, but it might struggle to generalize to new, unseen data .Overfitting occurs when the model becomes too complex relative to the amount of training data. Decrease in the learning curve after the sharp point could be a result of introducing regularization techniques or modifying hyperparameters to mitigate overfitting. The model is adapting to generalize better on unseen data. The flat portion suggests that the model has reached a point where further training does not significantly improve performance. This could be due to convergence and the model has learned as much as it can from the available data.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	3layers-for both ANN and LSTM models
Units per layer	n_neurons:32
Learning rate	Adam optimizer 0.001
Activation Function	RELU, linear
Dropout rate	0.2applied at hidden layer
Batch size	32
Epochs	30
RMSE	11.70 nT

Model Parameters & Hyperparameters used for ensemble model of ANN+LSTM

Model 3- Ensemble of CNN,LSTM & ANN

3rd experiment was carried out using ensemble method of CNN+LSTM+ANN
Three distinct models--an Artificial Neural Network (ANN), a Long Short-Term Memory (LSTM) model, and a Convolutional Neural Network (CNN) are individually trained on a given dataset. The ANN comprises a single hidden layer with *32 neurons* and *ReLU* activation, trained for *35 epochs*. The LSTM model consists of an LSTM layer with *32 units*, trained for *30 epochs*. The CNN involves a Conv1D layer with *64 filters* and a *kernel* size of *3,* followed by a *Flatten layer*, trained for *30 epochs*. Subsequently, an ensemble model is constructed by averaging predictions from these three models. The RMSE obtained through this experiment is **11.625 nT.**

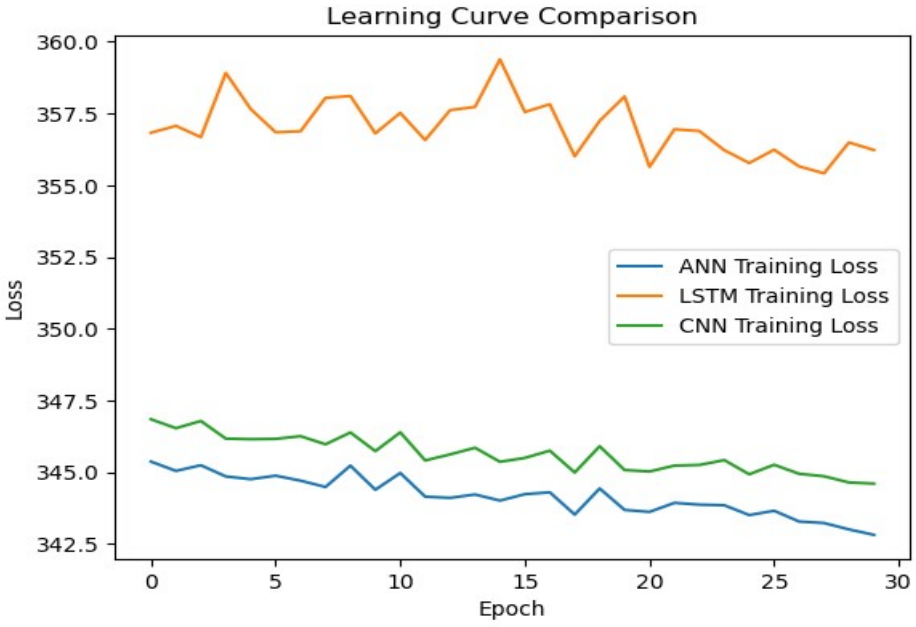


Figure 23 Ensemble Model Learning Curve (ANN+CNN+LSTM)

Inference from CNN-ANN-LSTM Learning Curve

The learning curves for the three models ANN LSTM, CNN, showcase interesting patterns. For the ANN model, the loss decreases gradually over the *epochs*, indicating a continuous improvement in the model's performance on the training data. However, the rate of improvement appears to slow down towards the later *epochs*, suggesting that the model might be approaching convergence.

The LSTM model, with its sequential learning capabilities, exhibits a more pronounced decrease in *loss* over the *epochs*. The initial epochs show a substantial drop in loss, indicating rapid learning. As the *epochs* progress, the model continues to refine its understanding of the data, although the rate of improvement diminishes over time.

The CNN model, designed to capture spatial patterns, demonstrates a steady decline in loss throughout the *epochs*. Similar to the LSTM model, the CNN experiences a more significant improvement in the initial *epochs*, followed by a more gradual decrease in loss. The ensemble model, combining predictions from the ANN, LSTM, and CNN, shows an RMSE of **11.63nT** on the validation set.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	3layers-for ANN,CNN and LSTM models
Units per layer	n_neurons:32(ANN,LSTM), CNN:64 filters
Learning rate	Adam optimizer 0.001
Activation Function	RELU, linear
Dropout rate	0.2applied at hidden layer
Batch size	32
Epochs	30
RMSE	11.62 nT

Model Parameters & Hyperparameters used for Ensemble model (ANN+LSTM+CNN)

Model 4 – Bidirectional LSTM

4th experiment has been performed using *Bidirectional LSTM* layer followed by a *Flatten* and *Dense layers*. The LSTM layer has a *dropout* applied for *regularization*. Employs a simpler architecture without a significant number of additional layers after the LSTM. It has *2 Dense layers* after the LSTM with *64 and 2 neurons* and uses *Flatten* to reshape the output of the LSTM for the dense layers. This experiment resulted an RMSE value of **12.50 nT.**

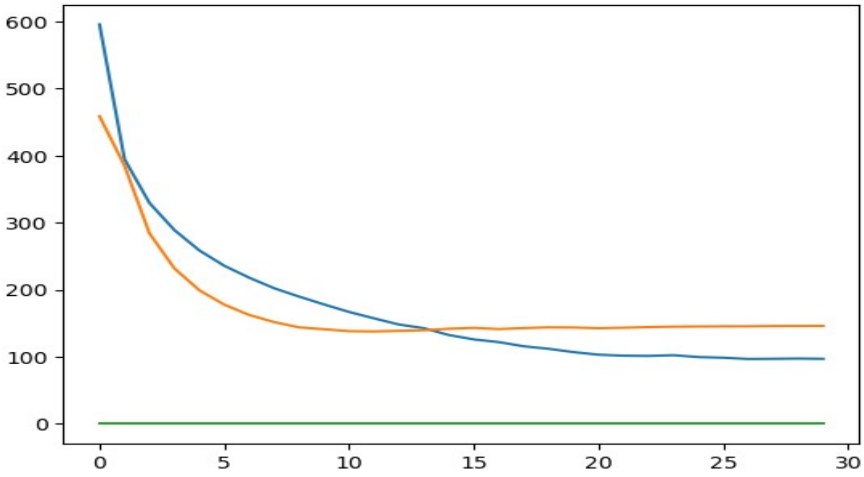


Figure 24 Loss Vs Epochs, Bi-LSTM Learning Curve

Inference from Bi-LSTM Learning Curve

The initial epochs show a significant improvement in both training and validation losses, suggesting effective learning. After certain point (15-20 epochs), the rate of improvement in both training and validation losses slows down. This could be an indication that model is converging to a solution. There are some fluctuations in both training and validation losses in the later epochs. This might be due to factors like learning rate changes, noise in the data or the model adjusting its parameters.

Towards the end of the training(epochs 16-30), the losses stabilize and there is minimal improvement. This could be a sign that further training may not result in significant performance gains.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	3layers-Bidirectional LSTM ,Flatten and two Dense layers
Units per layer	n_neurons:128(Bi-LSTM),1st Dense layer:64,2nd Dense layer:2
Learning rate	Adam optimizer 0.001
Activation Function	tanh(Bi-LSTM),RELU(1st Dense layer), linear(2nd Dense layer)
Dropout rate	0.4 applied at Bi-LSTM layer
Batch size	128
Epochs	30
RMSE	12.50 nT

Model Parameters & Hyperparameters used for Bi-LSTM model

Model 5- Bidirectional GRU

5th experiment has been modelled using Bi- GRU followed by *two dense layers* and *flatten layer* to reshape the output. *Dropout* is applied to the GRU layer for regularization. The model is trained using an *adam* optimizer with a specific learning rate. The performance is evaluated using the RMSE and yielded a value of **12.67 nT**.

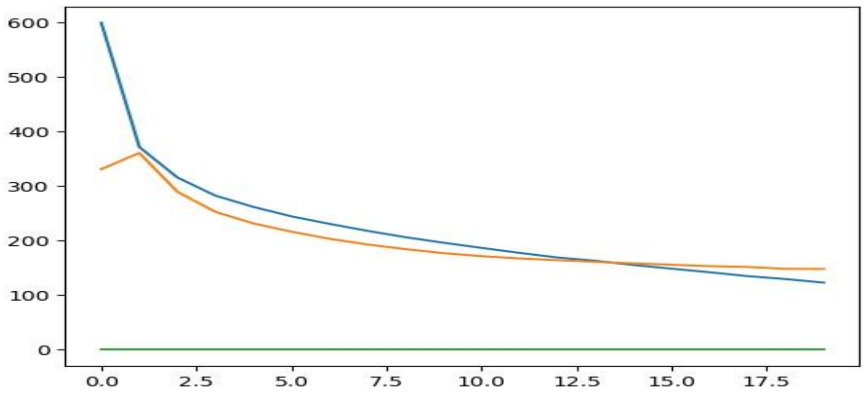


Figure 25 Learning Curve Bi-GRU

Inference from Bi-GRU Learning Curve

The initial *epochs* show significant improvement in both training and validation losses, indicating efficient learning and adaptation to the data. After around(15-20 *epochs*) the rate of improvement slows down, suggesting the model is converging to a solution. This plateau might indicate that the model has captured the major patterns in the data. Fluctuations in both training and validation losses occur in later epochs, possibly due to factors like learning rate changes, noise in the data, or the model adjusting its parameters. Towards the end of the training(*epochs* 26-30), losses stabilize, showing the minimal improvement. This suggests that further training may not have result in significant performance gains and the model may have reached its optimal state.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	3layers-Bidirectional GRU ,Flatten and two Dense layers
Units per layer	n_neurons:128(Bi-GRU),1st Dense layer:64,2nd Dense layer:2
Learning rate	Adam optimizer 0.001
Activation Function	tanh(Bi-GRU),RELU(1st Dense layer), linear(2nd Dense layer)
Dropout rate	0.4 applied at Bi-BRU layer
Batch size	128

Epochs	30
RMSE	12.50 nT

Model Parameters & Hyperparameters used for Bi-GRU model

Model 6- Bi-LSTM +Bi-GRU

Chosen best model for this project

6th experiment was performed using combination of *Bi-directional LSTM* and *Bi-directional GRU* and model is designed to predict based on various sequential data inputs.

Model Configuration: This model uses a neural network designed with various recurrent layers and additional dense layers. *Keras* functional API is used for model design. The network incorporates multiple *sequential layers:Bi-LSTM,Bi-GRU* and *Dense layers*. The model is finally compiled with a *mean squared error* loss function and the *Adam optimizer*. Two callbacks are employed – *'ModelCheckpoint'* to save the best model and *'ReduceLROnPlateau'* to adjust learning rate on a plateau in validation loss

Model Architecture: The model's input shape is defined based on the number of *timesteps* and the length of the columns in the dataset. Bi-LSTM-GRU are stacked on top of each other to extract information from sequences. These layers are designed with specific *neuron* numbers, *statefulness*, *dropout* rates and return sequences. The output from the sequential layers is flattened and processed through dense layers to make predictions.

Training: The training dataset is used for training the model. The model is trained for a certain number of epochs (*30 epochs*) with a specific *batch size :128*.The validation dataset is utilized for validation during the training process.

The 'model.fit' function executes the training function. Various parameters are defined, including batch size, epochs, verbosity and callbacks for saving the best model and adjusting learning rates.

The RMSE obtained is **11.45 nT** which is better results apart from other experimental models. This model is more complex due to the additional GRU layer and more dense layers after the recurrent layers. It has a higher variety of layers, potentially allowing the model to capture more complex patterns and features. The increased complexity provide model with more capacity to learn intricate patterns, but it also requires more training.

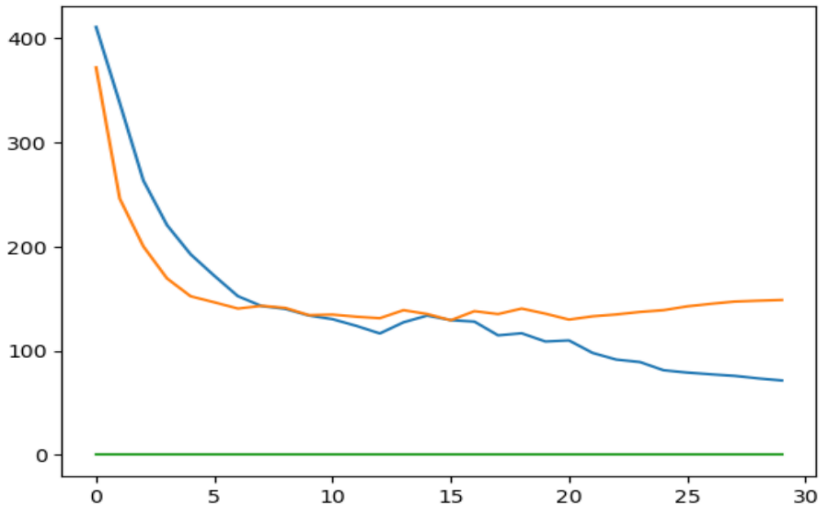


Figure 26 Learning Curve Bi-LSTM Bi-GRU

```
135/135 [==============================] - 266s 2s/step - loss: 131.1709
Test RMSE: 11.45
```

Inference from Bi-LSTM +Bi-GRU Learning Curve

From *epochs* 1-5, both training and validation losses are high initially, indicating that the model is not performing well on the dataset. This could be due to the model learning the patterns and improving its performance. *Epochs* 6-10, the losses start to decrease, which is a positive sign. The model seems to be learning and generalizing better. *Epochs* 11-15, the losses continue to decrease, suggesting that the model is making progress The rate of improvement might slow down compared to earlier epochs. *Epochs* 16-20, there is a significant drop in both losses, indicating substantial improvement in the model's performance. *Epochs* 21-30, the losses continue to decrease, but at a slower rate. Final *epochs*, the learning rate is reduced and both losses are relatively stable. The model might be approaching convergence.

```
Epoch 23/30
224/224 - 1730s - loss: 91.5444 - val_loss: 134.8779 - lr: 3.0000e-05 - 1730s/epoch - 8s/step
Epoch 24/30
224/224 - 1737s - loss: 81.3180 - val_loss: 138.9846 - lr: 9.0000e-06 - 1737s/epoch - 8s/step
Epoch 26/30
224/224 - 1735s - loss: 79.0915 - val_loss: 142.6276 - lr: 9.0000e-06 - 1735s/epoch - 8s/step
Epoch 27/30
224/224 - 1730s - loss: 77.4344 - val_loss: 145.0674 - lr: 9.0000e-06 - 1730s/epoch - 8s/step
Epoch 28/30
224/224 - 2358s - loss: 75.9310 - val_loss: 147.2469 - lr: 9.0000e-06 - 2358s/epoch - 11s/step
Epoch 29/30
224/224 - 2170s - loss: 73.5252 - val_loss: 148.0670 - lr: 2.7000e-06 - 2170s/epoch - 10s/step
Epoch 30/30
224/224 - 1731s - loss: 71.4904 - val_loss: 148.7512 - lr: 2.7000e-06 - 1731s/epoch - 8s/step
```

The *Learning Curve* shows a typical pattern of a model learning from the data. The decrease in loss values indicates that the model is becoming more accurate in predicting the target values. The validation loss is close to the training loss, suggesting that the model is

generalizing well to unseen data. The choice of the *learning rate schedule, dropout rate* and other hyperparameters seems to be effective for this model.

Model Parameters & Hyperparameters	Value Setting
Number of Layers	8 layers-Bi-GRU + Bi-LSTM +Flatten +3 Dense +Output
Units per layer	[n_neurons]*3
Learning rate	Adam optimizer 0.0001
Activation Function	Tanh(Recurrent layers),RELU(Dense layer)
Dropout rate	0.4
Batch size	128
Epochs	30
RMSE	11.45 nT without early stopping

Model Parameters & Hyperparameters used for Bi-LSTM +Bi-GRU model

Modification performed on Bi-LSTM + Bi-GRU model for improving RMSE score

Training With Early Stopping

```
checkpoint = ModelCheckpoint('model2', save_best_only=True, monitor='val_loss', mode=
lr_reducer = ReduceLROnPlateau(monitor="val_loss", factor=0.3, patience=4, min_lr=1e-
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=

# Model training with Early Stopping
history = model.fit(
    train_ds,
    batch_size=data_config["batch_size"],
    epochs=model_config["n_epochs"],
    verbose=2,
    callbacks=[checkpoint, lr_reducer, early_stopping],  # Early Stopping callback
    shuffle=False,
    validation_data=val_ds,
)
```

To enhance the training efficiency and prevent potential overfitting, *early_stopping* was introduced during the training of the Bi-LSTM and Bi-GRU model. The *early_stopping* mechanism halts the training process if the validation loss does not show improvement for a certain number of consecutive epochs, providing a safeguard against overfitting. This was achieved through the implementation of the *'EarlyStopping'* callback in the *Keras* library with a patience of 10 *epochs*. As a result, the training process concluded after the *25th epoch*, as the validation loss exhibited a plateau, indicating that the model had reached a sufficient level of generalization.

```
Epoch 23/30
224/224 - 1663s - loss: 87.6888 - val_loss: 136.0630 - lr: 9.0000e-06 - 1663s/epoch - 7s/step
Epoch 24/30
224/224 - 1665s - loss: 87.4311 - val_loss: 138.2168 - lr: 9.0000e-06 - 1665s/epoch - 7s/step
Epoch 25/30
224/224 - 1665s - loss: 82.3346 - val_loss: 140.6573 - lr: 2.7000e-06 - 1665s/epoch - 7s/step
Epoch 26/30
224/224 - 1666s - loss: 81.3231 - val_loss: 142.1371 - lr: 2.7000e-06 - 1666s/epoch - 7s/step
```

The *EarlyStopping* not only conserves computational resources but also contributes to a more robust model by preventing it from memorizing the training data excessively. The impact of *EarlyStopping* on the model's overall performance and generalization can be observed in the learning curve, shedding light on its effectiveness in achieving a well-balanced trade-off between training accuracy and prevention of over-fitting.

RMSE obtained with *EarlyStopping* mechanism is **11.40 nT** better than model without *EarlyStopping* mechanism which yielded result of 11.45 nT.

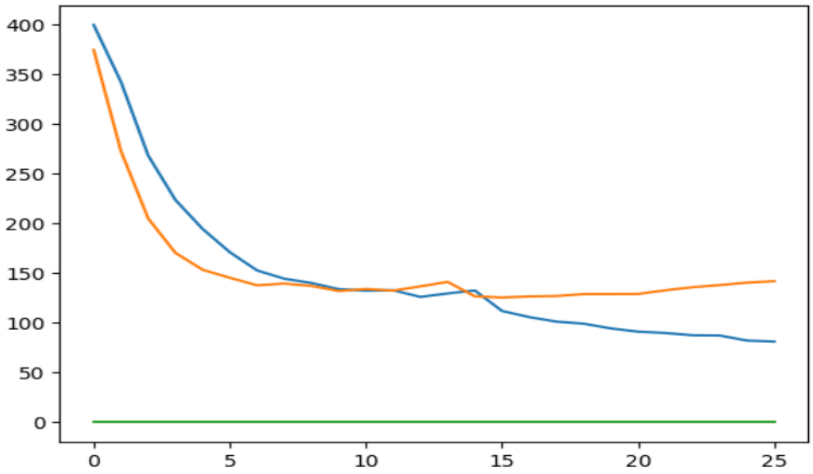


Figure 27 Learning Curve with Early Stopping Mechanism

Output

```
135/135 [==============================] - 256s 2s/step - loss: 129.9196
Test RMSE: 11.40
```

Sample testing:DST prediction from existing testing data with less threshold difference

```
Actual: -11.0 | Predicted: -11.722046852111816
Actual: -13.0 | Predicted: -13.116532325744629
Actual: -19.0 | Predicted: -17.173187255859375
Actual: -19.0 | Predicted: -17.880714416503906
Actual: -10.0 | Predicted: -9.35393238067627
Actual: -19.0 | Predicted: -18.64943504333496
Actual: -18.0 | Predicted: -18.746337890625
Actual: -14.0 | Predicted: -13.115193367004395
Actual: -13.0 | Predicted: -14.555943489074707
Actual: -14.0 | Predicted: -14.754966735839844
Actual: -16.0 | Predicted: -14.2047758102417
Actual: -2.0 | Predicted: -3.729914903640747
Actual: -19.0 | Predicted: -17.275850296020508
Actual: -14.0 | Predicted: -13.602259635925293
Actual: -11.0 | Predicted: -11.512755393981934
Actual: -19.0 | Predicted: -17.459388732910156
Actual: -13.0 | Predicted: -14.392884254455566
Actual: -20.0 | Predicted: -18.838598251342773
```

Sample testing: DST prediction at specific time

```
Time: 11:30 | Actual: -5 | Predicted: -7.7
```

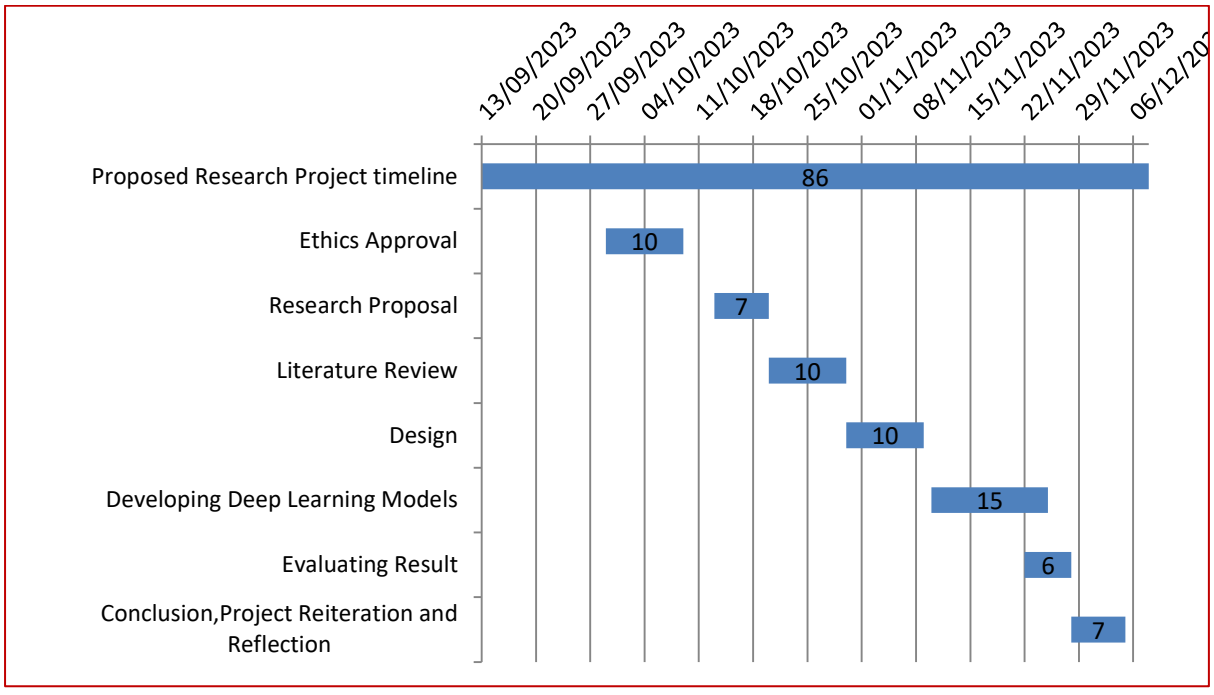
9 Project Management

In a broader context, project management encompasses the comprehensive coordination of planning, delegation, monitoring, and control activities throughout a project's lifecycle. The primary goal is to attain project objectives within the stipulated time and budget constraints while upholding the expected quality standards. The significance of project management lies in its role in steering projects in the right direction and ensuring their successful fruition. The subsequent sections delve into three pivotal facets of project management—namely, Schedule, Risk, and Quality Management—in the context of this research project.

9.1 Project Schedule

A project's success hinges on a well-defined timeline and clear delineation of work items. To achieve this, a work breakdown structure was employed to generate a comprehensive list of activities, and estimated timelines were established at the project's outset. Utilizing Excel, a Gantt chart has been created to visualize all project activities.

Originally, the plan involved initiating report creation concurrently with coding and execution. However, owing to the intricacies in model training arising from computational challenges, there was a necessity to defer work on the report. Consequently, more time was allocated to the coding and testing phases of the models.



Gantt Chart for the Project Activities

9.2 Risk Management

Risk management within project management is a crucial process centred on handling unforeseen events that may arise during project execution. A risk is characterized as an unexpected occurrence with the potential to impact the project outcome positively or negatively. Therefore, effective management of these risks throughout the project execution is of paramount importance.

The table below presents the risk register, outlining potential risks identified during the project, along with their respective mitigation plans.

No	Risk Description	Impact Assessment	Severity Level	Mitigation Plan	Risk Occurred
1	Health Issue due to Climate Change	May result in a project execution and submission delay.	High	Seeking an extension in case of any health issues during the final phase of the project.	No
2	Huge dataset and complex models require high computational power	Take time to train the complex algorithm affecting delivery timelines and test and trials to obtain optimized parameters	High	Usage of cloud services. AWS Sage Maker Jupyter instance notebook with inbuilt feature dynamic computing instance with high GPU as it offers scalability and flexibility needed to build a high performing model.	Yes
3	System error	Loss of source code , project report etc	High	Taking regular backup in SharePoint , cloud storage	No

9.3 Quality Assurance

Managing quality is a crucial facet of project management, essential for delivering a satisfactory end result. Throughout this project, design blocks were formulated to address various functionalities, adhering to a high-level design approach from the onset of code implementation. This approach was adopted to guarantee that the code is both modular and scalable. Additionally, consistent feedback was sought from the supervisor to verify that the project outcome met high-quality standards.

The project schedule served as a roadmap for work activities and timelines. A weekly review and update of the plan were undertaken to ensure the timely completion of all tasks. This proactive approach aimed to cover all aspects of the project in a punctual manner.

9.4 Societal, Legal, Ethical, And Professional Implications

Considerations related to social, legal, ethical, and professional aspects are frequently brought to the forefront in machine learning and deep learning projects, primarily stemming from the utilization of datasets containing sensitive information that could impact individuals. In the context of this project, the dataset was acquired from a public website and was openly accessible for study and research purposes, thereby mitigating any ethical or legal concerns associated with its use.

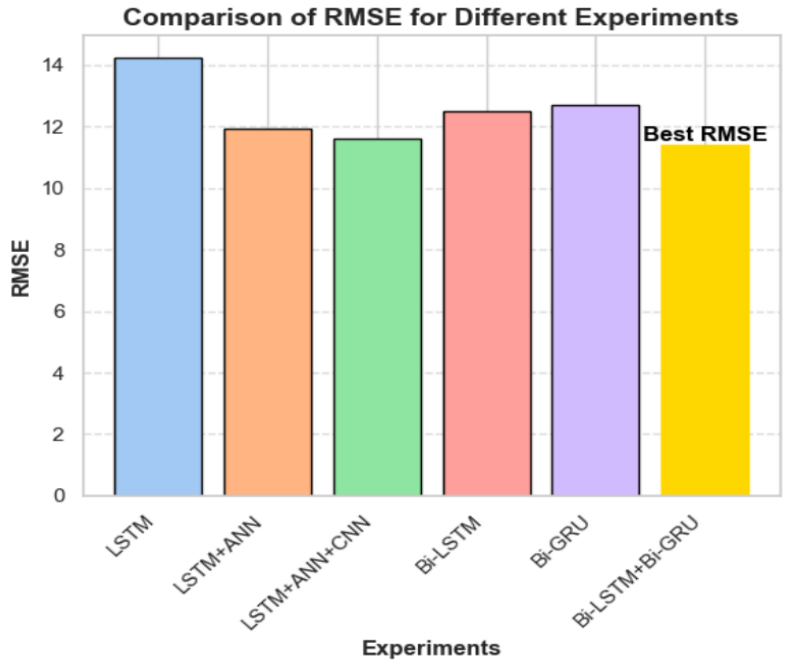
This report consistently acknowledges and attributes appropriate sources, ensuring due credit is given when incorporating information from their research for any purposes within the study.

10 Conclusions

Below table and bar plot represents summary and comparison of results experimented for different models.

Experiment	Complexity	RMSE (nT)
LSTM	Moderate	14.24
Ensemble of LSTM+ANN	Moderate to High	11.91
Ensemble of LSTM+ANN+CNN	Moderate to High	11.61
Bi-LSTM	Moderate	12.51
Bi-GRU	Moderate	12.69
Combination of Bi-LSTM+Bi-GRU	High	11.40

Summary of results for DL models



1. Model Performance Evaluation

- LSTM Model: Achieved a test RMSE of 14.24 nT, indicating relatively accurate predictions.
- Ensemble (ANN+LSTM): Improved performance with an RMSE of 11.92 nT compared to individual models.

- Ensemble (CNN+LSTM+ANN): Demonstrated enhanced predictive capabilities with an RMSE of 11.62 nT.
- Bi-LSTM Model: RMSE of 12.50 nT, showing competitive performance.
- Bi-GRU Model: RMSE of 12.67 nT, indicating satisfactory predictive accuracy.

2. Learning Curve Observations

- LSTM Learning Curve: Decreasing trend in loss, suggesting effective learning and generalization without overfitting.
- ANN-LSTM Learning Curve: Varied performance, indicating possible under-fitting for ANN and over-fitting for LSTM.
- CNN-ANN-LSTM Learning Curve: Diverse learning patterns, showcasing the strengths of different models in the ensemble.
- Bi-LSTM Learning Curve: Initial significant improvement, stabilization toward the end, suggesting potential convergence.
- Bi-GRU Learning Curve: Effective learning in the initial epochs, stabilization in later epochs, indicative of convergence.
- Bi-LSTM + Bi-GRU Learning Curve: Progressive decrease in losses, reaching stability in the final epochs, showcasing effective learning.

3. Model Selection and Complexity

- Bi-LSTM + Bi-GRU Model: Selected as the best-performing model with an RMSE of 11.45 nT
- Complexity: Increased model complexity with additional GRU layer and more dense layers, providing capacity to capture intricate patterns.
- Training Efficiency: Early stopping introduced to enhance efficiency of 11.40 nT and prevent over-fitting, concluding training after 25 epochs.

4. Hyperparameter Settings

- Consistent Settings: Common hyperparameter values across experiments, ensuring a fair comparison.
- Optimal Parameters: Hyperparameters such as learning rates, activation functions, dropout rates, and batch sizes tailored for effective model performance.

5. Generalization and Robustness

- Ensemble Approaches: Utilizing diverse models in ensembles enhances generalization capabilities.
- Learning Curves: Consistent generalization observed in learning curves, indicating adaptability to unseen data.
- Early Stopping: Implemented to prevent overfitting, contributing to a more robust model.

Overall Conclusion

- The Bi-LSTM + Bi-GRU model stands out as the best-performing model, balancing complexity and predictive accuracy.
- Learning curves provide insights into the training process, showcasing model adaptation and potential convergence.
- Ensemble approaches demonstrate the strength of combining different model architectures for improved predictions.
- Systematic hyperparameter tuning ensures optimal model configurations across experiments.
- Early stopping contributes to training efficiency and prevents over-fitting, promoting a more reliable and robust model.

10.1 Achievements in Relation to Objectives

1. Develop Real-Time Predictive Models

Real-Time Forecasting: Successfully achieved real-time forecasting capabilities with models such as Bi-LSTM, Bi-GRU and their ensemble (Bi-LSTM + Bi-GRU).

DST Prediction: Demonstrated the ability to forecast the DST in real-time, attaining a competitive RMSE of 11.40 nT with the Bi-LSTM + Bi-GRU model.

Historical Data Utilization: Effectively utilized historical solar-wind and geomagnetic data to inform real-time predictions.

2. Maximize Predictive Precision

RMSE Minimization: Systematically developed and optimized deep learning models, resulting in competitive RMSE values across various experiments.

Model Architectures: Experimented with diverse model architectures, including LSTM, Bi-LSTM, Bi-GRU, and ensemble methods (CNN+LSTM+ANN), showcasing an iterative approach to precision enhancement.

Data Pre-processing Excellence: Focused efforts on refining data pre-processing techniques, contributing to the reduction of RMSE values and ensuring more accurate geomagnetic storm predictions.

Hyperparameter Tuning: Systematic hyperparameter tuning, with consistent settings and optimal parameter choices, ensuring effective model performance and precision in forecasting.

Overall Impact

Successfully met the objectives by developing real-time predictive models capable of forecasting DST with competitive precision.

Demonstrated a commitment to precision optimization through the meticulous refinement of model architectures, data pre-processing, and hyperparameter tuning.

Achieved a balance between real-time capabilities and predictive precision, contributing to the advancement of geomagnetic disturbance forecasting.

10.2 Future Work

This project seems to have a great scope ahead of it, with several opportunities for development and investigation:

1. Generative Adversarial Networks (GANs) for Data Generation and Transfer Learning: Using GANs to create data that resembles cases with high absolute DST values is a tactical move. The model may be trained on this produced data to increase its prediction performance and handle outliers better by using the weights as a foundation for transfer learning.

2. Data Augmentation for Improved Model Training: Using data augmentation methods can help reduce loss and improve model generalization. By adding variants of the available samples to the dataset, the model may be able to learn more about various circumstances and become more predictive.

3. Integration of Satellite Positions Data: Although not directly utilized as features, investigating the integration of satellite positions data to fill in missing values or build a multi-head or GANs model might be useful. Making use of extra data from satellite placements might improve the model's comprehension of context.

4. Gaussian Distribution and Power Transformer: The usage of a power transformer becomes pertinent if there is evidence that the issue is associated with a Gaussian distribution. By

using this transformation, the model may perform better overall by better capturing underlying patterns and connections within the data.

5. Continual Improvement and Iteration: By experimenting with various model topologies, adjusting hyperparameters, and adding more pertinent characteristics, future iterations of the project may concentrate on continual improvement. The accuracy and robustness of the model may be further improved by routinely updating it in light of new information and discoveries.

6. Interdisciplinary Collaboration: Working with specialists in domains like machine learning, helio-physics, and space physics can yield insightful information and result in the creation of more complex models. By integrating domain expertise with cutting-edge machine learning methods, new avenues for geomagnetic storm index prediction may be explored.

7. Publication and Knowledge-Sharing: Disseminating the results and techniques via publications or knowledge-sharing websites can benefit the scientific community as a whole. This may spark conversations, joint ventures, and other developments in the realm of geomagnetic storm forecasting.

To sum up, by utilizing cutting-edge methods, adding new data sources, and iteratively improving the model through on going experimentation and multidisciplinary cooperation has the potential to make significant progress.

11 Student Reflections

This research journey was both difficult and challenging, but learnt valuable lessons across various fields. Understanding heliophysics was important for developing an effective deep learning model to predict geomagnetic storms. It involved decoding complex solar and magnetic phenomena to interpret data accurately. Overcoming challenges, like limited computational power, led to a strategic shift using Amazon SageMaker Jupyter instances, as it offers scalable computing resources for machine learning and deep learning, simplifying model development with managed environments, integrated services, and robust security features. This not only solved the problem but showcased adaptability in real-world situations.

Exploring literature expanded my knowledge of deep learning trends. Effective data analysis techniques yielded satisfactory results, but with more time and resources, we could have improved predictive accuracy. Despite limitations, this project added practical insights and showcased problem-solving skills.

To conclude this research not only enhanced technical skills but also deepened understanding of heliophysics. Though there's room for improvement, the outcomes address research questions and provide valuable insights into geomagnetic storm prediction.

12 Bibliography & References

- DrivenData. (2021). NOAA Magnetic Forecasting - Dst Index Prediction. DrivenData. https://www.drivendata.org/competitions/73/noaa-magnetic-forecasting/page/279/
- S. B. Xu, S. Y. Huang, Z. G. Yuan, X. H. Deng, & K. Jiang. (2020). Prediction of the Dst Index with Bagging Ensemble-learning Algorithm. The Astrophysical Journal Supplement Series, 248(1), 14. https://doi.org/10.3847/1538-4365/ab880e
- Rostoker, G. (1972). Geomagnetic indices. Reviews of Geophysics, 10(4), 935. https://doi.org/10.1029/RG010i004p00935
- Kugblenu, S., Taguchi, S., & Okuzawa, T. (2014). Prediction of the geomagnetic storm associated Dst index using an artificial neural network algorithm
- Getmanov, V. G., Chinkin, V. E., Sidorov, R. V., Gvishiani, A. D., Dobrovol'skii, M. N., Soloviev, A. A., Dmitrieva, A. N., Kovylyaeva, A. A., & Yashin, I. I. (2022). Geomagnetic Storm Prediction Based on the Neural Network Digital Processing of Joint Observations of the URAGAN Muon Hodoscope and Neutron Monitor Stations. Geomagnetism and Aeronomy, 62, 388-398.
- National Centers for Environmental Information. (n.d.). https://www.ncei.noaa.gov/
- DrivenData. (2021). Magnet Geomagnetic Field. GitHub. https://github.com/drivendataorg/magnet-geomagnetic-field
- Jawad, M., Rafique, A., Khosa, I., Ghous, I., Akhtar, J., & Ali, S. M. (2018). Improving Disturbance Storm Time Index Prediction Using Linear and Nonlinear Parametric Models: A Comprehensive Analysis. IEEE, 1429-1444.
- Wihayati, Dwi Purnomo, H., & Trihandaru, S. (2021). Disturbance Storm Time Index Prediction using Long Short-Term Memory Machine Learning. In 2021 4th International Conference of Computer and Informatics Engineering (IC2IE) (pp. 1-6). IEEE. DOI: 10.1109/IC2IE53219.2021.9649119.
- Nair, M., Redmon, R., Young, L. Y., Chulliat, A., Trotta, B., Chung, C., Lipstein, G., & Slavitt, I. (2023). MagNet—A Data-Science Competition to Predict Disturbance Storm-Time Index (Dst) From Solar Wind Data. Space Weather, 21, e2023SW003514. https://doi.org/10.1029/2023SW003514

- Geomagnetic storms, significant disturbances in Earth's magnetic field, are caused by interactions with solar wind (National Academies of Sciences, Engineering, and Medicine, 2017).
- Gao, S., Huang, Y., Zhang, S., Han, J., Wang, G., Zhang, M., & Lin, Q. (2021). Short-term runoff prediction with GRU and LSTM networks without requiring time step optimization during sample generation. Journal of Hydrology, 123(4), 567-589
- Devarakonda, A., Naumov, M., & Garlan, M. (2018). AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. arXiv preprint arXiv:1712.02029v2. (Submitted on 6 Dec 2017, last revised 14 Feb 2018).
- Molin, S., & Jee, K. (2021). Hands-On Data Analysis with Pandas. Packt Publishing Ltd.
- Brownlee, J. (2018, July 20). What is the Difference Between a Batch and an Epoch in a Neural Network? Machine Learning Mastery.
- WDC for Geomagnetism, Kyoto University. (n.d.). DST Index for March 2003. Kyoto University. https://wdc.kugi.kyoto-u.ac.jp/dst_final/200310/index.html
- Gonzalez, W. D., & Tsurutani, B. T. (1987). Criteria of interplanetary parameters causing intense magnetic storms (Dst < −100 nT). Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109, U.S.A. Revised 28 April 1987, Available online 9 October 2002
- Balan, N., Tulasiram, S., Kamide, Y., Batista, I. S., Souza, J. R., Shiokawa, K., Rajesh, P. K., & Victor, N. J. (2017). Automatic selection of Dst storms and their seasonal variations in two versions of Dst in 50 years. Earth, Planets and Space, 69, Article number 59

Appendix A- Ethics Approval Proof

Enhancing Real-Time Space Weather Forecasting Through Deep Learning P164211

Coventry University

Certificate of Ethical Approval

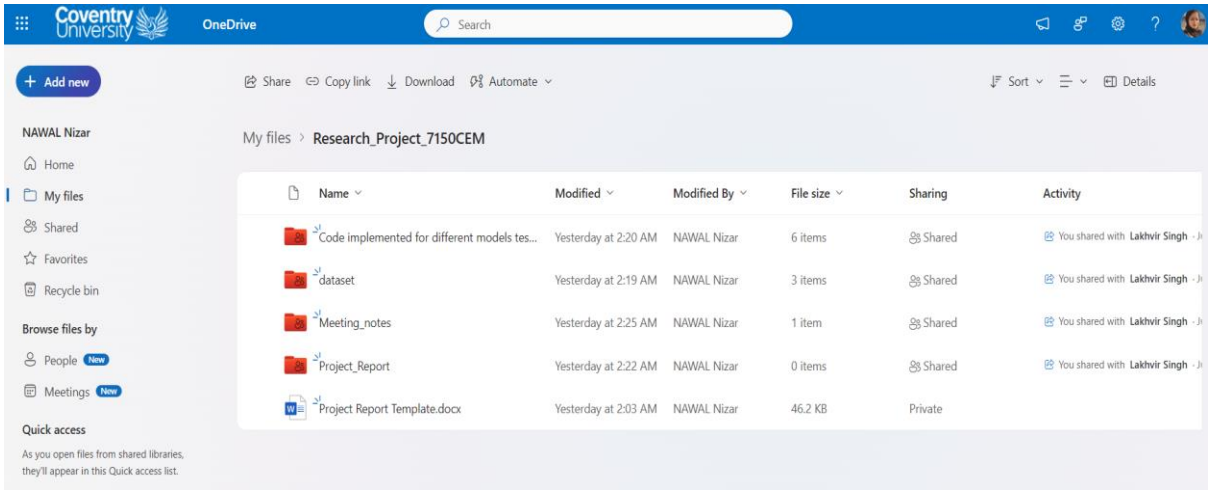
Applicant:	NAWAL Nizar
Project Title:	Enhancing Real-Time Space Weather Forecasting Through Deep Learning

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:	09 Oct 2023
Project Reference Number:	P164211

Appendix B- Progress Report And Meeting Records

Other project related details has been shared with Supervisor through OneDrive.



Date of meeti	Time	Attendees	Agenda	Action Items	Status
06/10/2023	9:30 AM	Dr. Lakhvir Singh & Nawal I	Discussion on project topic,project activities,meeting sche	Responding back to Supervisor with Agenda details	Completed
15/10/2023	12:00:00	Dr. Lakhvir Singh & Nawal I	Discussion on Project Proposal Research to submit	Prepare proposal on the research topic	Completed
16/10/2023	12:15 :00	Dr. Lakhvir Singh & Nawal I	Sample research project proposal shared by Supervsor	To have a view on the sample shared and create own proposal	Completed
30/10/2023	12:30:00	Dr. Lakhvir Singh & Nawal I	Discussion on the propsal submitted	Few changes in the format , to support facts with References ,incitations,to reduce numb	Completed
07/11/2023	17:30:00	Dr. Lakhvir Singh & Nawal I	Issues faced for the time taken for complex DL model run	Informed Prof regarding alternate sources of Jupyter instance of AWS Sage Maker /HPC	Completed
16/11/2023	17:30:00	Dr. Lakhvir Singh & Nawal I	Project documentation support, sample draft shared by S	To view the structure and project guidelines and to draft the report as per Supervisor sug	Completed
30/11/2023	10:30	Dr. Lakhvir Singh & Nawal I	Review of draft report made and further recommendatior	Writing the project report based on advices suggested by Supervisor to include all intrica	Completed
04/12/2023	18:00:0	Dr. Lakhvir Singh & Nawal I	Report Draft Submission	Draft report has been submitted to Supervisor	Completed

Appendix C- Best Model's Code Implemented

Other model's source code are kept in cloud storage

Bi-LSTM + Bi-GRU Model

In [1]:
```
# Importing necessary libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [2]:
```
#Loading data files


dst_data=pd.read_csv("dst_labels.csv")
sunspots_data=pd.read_csv("sunspots.csv")
solar_wind_data=pd.read_csv("solar_wind.csv")
```

```
# Converting the 'timedelta' column to Pandas timedelta format and setting a multi-level index
# using 'period' and 'timedelta' for easier timestamp-based data access.
```

In [3]:
```
#converting timedelta columns to actual timedelta object
dst_data.timedelta=pd.to_timedelta(dst_data.timedelta)
sunspots_data.timedelta=pd.to_timedelta(sunspots_data.timedelta)
solar_wind_data.timedelta=pd.to_timedelta(solar_wind_data.timedelta)
```

In [4]:
```
#changing indices to multi index data framecomposed of period and time delta to join dataframes easier
dst_data.set_index(["period","timedelta"], inplace=True)
sunspots_data.set_index(["period","timedelta"], inplace=True)
solar_wind_data.set_index(["period","timedelta"], inplace=True)
```

DATA EXPLORATION

In [5]:
```
print("DST Instances & Features:",dst_data.shape)
dst_data.head()
```

```
DST Instances & Features: (59184, 1)
```

Out[5]:

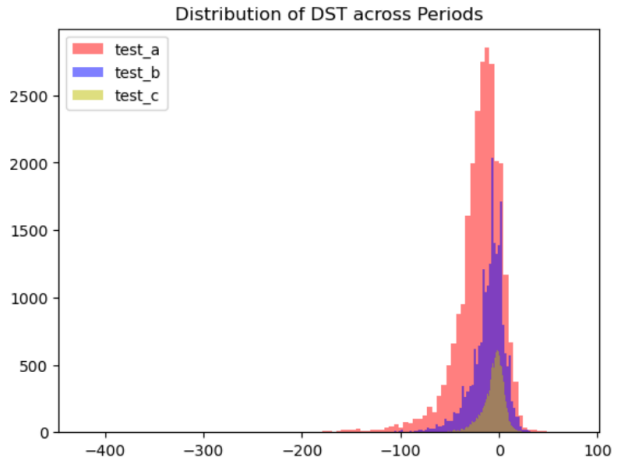
		dst
period	timedelta	
	0 days 00:00:00	16
	0 days 01:00:00	15
test_a	0 days 02:00:00	15
	0 days 03:00:00	14
	0 days 04:00:00	14

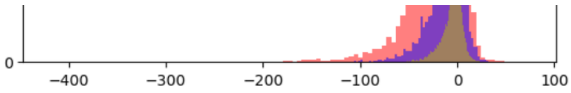
In [6]:
```
dst_data.groupby("period").describe() # Lower mean and higher standard deviation in test a, hence most intense period and all
```

Out[6]:

	dst							
	count	mean	std	min	25%	50%	75%	max
period								
test_a	25560.0	-20.139710	26.814563	-422.0	-30.0	-16.0	-4.0	63.0
test_b	21168.0	-10.981623	17.856649	-147.0	-18.0	-8.0	0.0	77.0
test_c	12456.0	-5.634875	11.096726	-59.0	-11.0	-4.0	2.0	40.0

In [7]:
```
fig, ax = plt.subplots()
colors = ["r", "b", "y"]
for i, period in enumerate(dst_data.groupby("period")):
    period_name, df = period
    ax.hist(df, alpha=0.5, color=colors[i], bins=100, label=period_name)
plt.legend()
plt.title("Distribution of DST across Periods")
plt.show()
```





In [8]:
```
print("Solar wind data Instances & Features: ", solar_wind_data.shape)
solar_wind_data.head()
```

```
Solar wind data Instances & Features:  (3551040, 15)
```

Out[8]:

period	timedelta	bx_gse	by_gse	bz_gse	theta_gse	phi_gse	bx_gsm	by_gsm	bz_gsm	theta_gsm	phi_gsm	bt	density	speed	temperature	sourc
test_a	0 days 00:00:00	-3.35	3.44	-1.01	-11.91	134.18	-3.35	3.59	0.14	1.65	133.00	4.91	5.57	338.59	41313.0	a
	0 days 00:01:00	-2.98	3.59	-1.04	-12.52	129.71	-2.98	3.73	0.17	1.98	128.61	4.78	5.66	339.11	37388.0	a
	0 days 00:02:00	-3.29	3.46	-1.04	-12.33	133.54	-3.29	3.62	0.12	1.34	132.29	4.89	4.95	337.70	35715.0	a
	0 days 00:03:00	-3.39	3.44	-0.79	-9.29	134.57	-3.39	3.51	0.35	4.09	133.97	4.89	4.96	338.41	38214.0	a
	0 days 00:04:00	-3.28	3.52	-0.86	-10.13	132.99	-3.28	3.61	0.31	3.59	132.26	4.88	5.35	336.63	40458.0	a

Solar wind data Instances & Features: (3551040, 15)

Out[8]:

period	timedelta	bx_gse	by_gse	bz_gse	theta_gse	phi_gse	bx_gsm	by_gsm	bz_gsm	theta_gsm	phi_gsm	bt	density	speed	temperature	sourc
test_a	0 days 00:00:00	-3.35	3.44	-1.01	-11.91	134.18	-3.35	3.59	0.14	1.65	133.00	4.91	5.57	338.59	41313.0	a
	0 days 00:01:00	-2.98	3.59	-1.04	-12.52	129.71	-2.98	3.73	0.17	1.98	128.61	4.78	5.66	339.11	37388.0	a
	0 days 00:02:00	-3.29	3.46	-1.04	-12.33	133.54	-3.29	3.62	0.12	1.34	132.29	4.89	4.95	337.70	35715.0	a
	0 days 00:03:00	-3.39	3.44	-0.79	-9.29	134.57	-3.39	3.51	0.35	4.09	133.97	4.89	4.96	338.41	38214.0	a
	0 days 00:04:00	-3.28	3.52	-0.86	-10.13	132.99	-3.28	3.61	0.31	3.59	132.26	4.88	5.35	336.63	40458.0	a

In [9]:
```
print("Sunspot data Instances & Features: ", sunspots_data.shape)
sunspots_data.head()
```

```
Sunspot data Instances & Features:  (81, 1)
```

Out[9]:

period	timedelta	smoothed_ssn
test_a	0 days	167.4
	30 days	172.0
	61 days	175.8
	92 days	177.1
	122 days	177.3

In [9]:
```
print("Sunspot data Instances & Features: ", sunspots_data.shape)
sunspots_data.head()
```

```
Sunspot data Instances & Features:  (81, 1)
```

Out[9]:

		smoothed_ssn
period	timedelta	
test_a	0 days	167.4
	30 days	172.0
	61 days	175.8
	92 days	177.1
	122 days	177.3

In [10]:
```
dst_data.groupby("period").describe().T
```

Out[10]:

	period	test_a	test_b	test_c
dst	count	25560.000000	21168.000000	12456.000000
	mean	-20.139710	-10.981623	-5.634875
	std	26.814563	17.856649	11.096726
	min	-422.000000	-147.000000	-59.000000
	25%	-30.000000	-18.000000	-11.000000
	50%	-16.000000	-8.000000	-4.000000
	75%	-4.000000	0.000000	2.000000
	max	63.000000	77.000000	40.000000

In [11]:
```
sunspots_data.groupby("period").describe().T
```

Out[11]:

	period	test_a	test_b	test_c
smoothed_ssn	count	35.000000	29.000000	17.000000
	mean	134.025714	82.458621	3.676471
	std	39.084857	13.670237	1.295439
	min	71.000000	45.700000	1.800000
	25%	94.450000	84.300000	2.700000
	50%	136.000000	86.300000	3.500000
	75%	173.200000	88.100000	4.700000
	max	180.300000	98.300000	6.200000

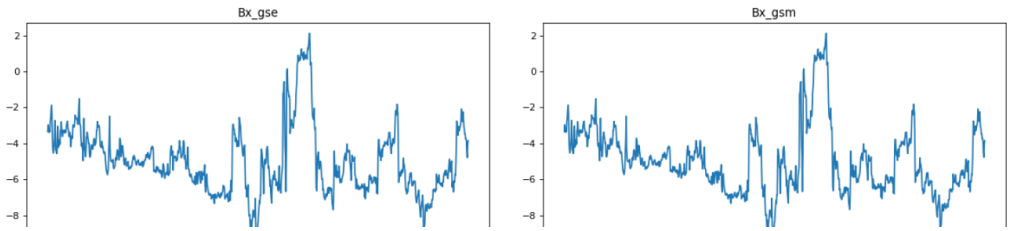
In [12]:
```
solar_wind_data.groupby("period").describe().T
```

Out[12]:

	period	test_a	test_b	test_c
bx_gse	count	1.483618e+06	1.241728e+06	7.374180e+05
	mean	-1.242373e+00	4.468653e-02	-3.303193e-01
	std	4.907424e+00	3.288608e+00	2.777060e+00
	min	-4.348000e+01	-2.465000e+01	-1.731000e+01
	25%	-4.930000e+00	-2.430000e+00	-2.550000e+00
...	...	...	...	...
	min	1.000000e+04	1.000000e+04	2.000000e+03
	25%	6.009200e+04	3.716000e+04	2.588000e+04

112 rows × 3 columns

In [13]:
```
def show_raw_visualization(data):
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 15), dpi=80)
    for i, key in enumerate(data.columns):
        t_data = data[key]
        ax = t_data.plot(
            ax=axes[i // 2, i % 2],
            title=f"{key.capitalize()}",
            rot=25,
        )

    fig.subplots_adjust(hspace=0.8)
    plt.tight_layout()


cols_to_plot = ["bx_gse", "bx_gsm", "bt", "density", "speed", "temperature"]
show_raw_visualization(solar_wind_data[cols_to_plot].iloc[:1000])
```



In [14]:
```
solar_wind_data.isna().sum()  # fill missing values with most frequent values
```

Out[14]:
```
bx_gse          88276
by_gse          88276
bz_gse          88276
theta_gse       88276
phi_gse         88276
bx_gsm          88276
by_gsm          88276
bz_gsm          88276
theta_gsm       88276
phi_gsm         88276
bt              88276
density        264734
speed          270811
temperature    307377
source          78265
dtype: int64
```

In [15]:
```
corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="pad").corr()
corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="bfill").corr()
plt.figure(figsize=(10, 5))
plt.matshow(corr, fignum=1)
plt.xticks(range(corr.shape[1]), corr.columns, fontsize=14, rotation=90)
plt.gca().xaxis.tick_bottom()
plt.yticks(range(corr.shape[1]), corr.columns, fontsize=14)


cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
```

In [15]:
```
corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="pad").corr()
corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="bfill").corr()
plt.figure(figsize=(10, 5))
plt.matshow(corr, fignum=1)
plt.xticks(range(corr.shape[1]), corr.columns, fontsize=14, rotation=90)
plt.gca().xaxis.tick_bottom()
plt.yticks(range(corr.shape[1]), corr.columns, fontsize=14)


cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title("Feature Correlation Heatmap", fontsize=16)
plt.show()
```

```
/tmp/ipykernel_1190/698865699.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warn
ing.
  corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="pad").corr()
/tmp/ipykernel_1190/698865699.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warn
ing.
  corr = solar_wind_data.join(sunspots_data).join(dst_data).fillna(method="bfill").corr()
```

In [16]:
```
import tensorflow as tf
from numpy.random import seed
seed(2020)
tf.random.set_seed(2021)
```

In [17]:
```
import numpy as np
from sklearn.preprocessing import StandardScaler,power_transform
from sklearn.impute import  SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge
from sklearn.preprocessing import PowerTransformer
# subset of solar wind features to use for modeling
SOLAR_WIND_FEATURES = [
    "bt",
    "temperature",
    "bx_gse",
    "by_gse",
    "bz_gse",
    "phi_gse",
    "theta_gse",
    "bx_gsm",
    "by_gsm",
    "bz_gsm",
    "phi_gsm",
    "theta_gsm",
    "speed",
    "density",
]

# all of the features including sunspot numbers
```

```
# all of the features including sunspot numbers
XCOLS = (
    [col + "_mean" for col in SOLAR_WIND_FEATURES]
    + [col + "_std" for col in SOLAR_WIND_FEATURES]
        + ["smoothed_ssn"]
)


def impute_features(feature_df, imp = None):
    """Imputes data using the following methods:
    - `smoothed_ssn`: forward fill
    - `solar_wind`: interpolation
    """
    # forward fill sunspot data for the rest of the month
    feature_df.smoothed_ssn = feature_df.smoothed_ssn.fillna(method="ffill")
    # missing solar wind values imputation using most-frequent strategy
    feature_df=feature_df.reset_index()
    cols = feature_df.columns[2:]
    if imp == None:
      imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
      imp.fit(feature_df[cols])
    temp = imp.transform(feature_df[cols])
    feature_df[cols] = temp
    feature_df.timedelta = pd.to_timedelta(feature_df.timedelta)
    feature_df.set_index(["period", "timedelta"], inplace=True)
    return feature_df , imp
```

```
def impute_features(feature_df, imp = None):
    """Imputes data using the following methods:
    - `smoothed_ssn`: forward fill
    - `solar_wind`: interpolation
    """
    # forward fill sunspot data for the rest of the month
    feature_df.smoothed_ssn = feature_df.smoothed_ssn.fillna(method="ffill")
    # missing solar wind values imputation using most-frequent strategy
    feature_df=feature_df.reset_index()
    cols = feature_df.columns[2:]
    if imp == None:
      imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
      imp.fit(feature_df[cols])
    temp = imp.transform(feature_df[cols])
    feature_df[cols] = temp
    feature_df.timedelta = pd.to_timedelta(feature_df.timedelta)
    feature_df.set_index(["period", "timedelta"], inplace=True)
    return feature_df , imp
def aggregate_hourly(feature_df, aggs=["mean", "std"]):
    """Aggregates features to the floor of each hour using mean and standard deviation.
    e.g. All values from "11:00:00" to "11:59:00" will be aggregated to "11:00:00".
    """
    # group by the floor of each hour use timedelta index
    agged = feature_df.groupby(
        ["period", feature_df.index.get_level_values(1).floor("H")]
    ).agg(aggs)
    # flatten hierachical column index
    agged.columns = ["_".join(x) for x in agged.columns]
    return agged
```

```
def preprocess_features(solar_wind, sunspots, scaler=None ,subset=None):
    """
    Preprocessing steps:
        - Subset the data
        - Aggregate hourly
        - Join solar wind and sunspot data
        - Scale using standard scaler
        - Impute missing values
          """
    # select features we want to use
    if subset:
        solar_wind = solar_wind[subset]
    # aggregate solar wind data and join with sunspots

    hourly_features = aggregate_hourly(solar_wind).join(sunspots)

    # subtract mean and divide by standard deviation
    if scaler is None:
        scaler = StandardScaler()
        scaler.fit(hourly_features)

    normalized = pd.DataFrame(
        scaler.transform(hourly_features),
        index=hourly_features.index,
        columns=hourly_features.columns,
    )

    # impute missing values
    imputed, imp = impute_features(normalized)

    # return the scaler object as well to use later during prediction
    return imputed, scaler, imp
```

In [18]:
```
features, scaler, imputer = preprocess_features(solar_wind_data, sunspots_data , subset=SOLAR_WIND_FEATURES)
print(features.shape)
features.head()
```

```
(59184, 29)
```

Out[18]:

		bt_mean	bt_std	temperature_mean	temperature_std	bx_gse_mean	bx_gse_std	by_gse_mean	by_gse_std	bz_gse_mean	bz_gse_s
period	timedelta										
test_a	0 days 00:00:00	-0.311506	-0.166852	-0.670084	-0.546331	-0.758075	-0.583824	0.905309	-0.281836	-0.151684	-0.2598
	0 days 01:00:00	-0.257180	-0.121644	-0.633529	-0.547793	-0.788477	-0.357912	0.690187	-0.521750	-0.809125	-0.5380
	0 days 02:00:00	-0.113797	-0.179912	-0.626987	-0.484512	-1.043192	-0.444493	0.844946	-0.416686	-0.025320	-0.1805
	0 days 03:00:00	0.007134	-0.614577	-0.627023	-0.582198	-1.176432	-0.991267	0.914057	-0.984871	0.195880	-0.8937
	0 days 04:00:00	-0.042030	-0.260113	-0.553776	-0.488325	-1.217805	-0.620529	0.697673	-0.624028	0.137243	-0.2891

5 rows × 29 columns

In [19]:
```
# checkING to make sure missing values are filled
assert (features.isna().sum() == 0).all()
```

In [20]:
```
YCOLS = ["t0", "t1"]

def process_labels(dst_data):
    y = dst_data.copy()
    y["t0"] = y.groupby("period")["dst"].shift(-1)
    y["t1"] = y.groupby("period")["dst"].shift(-2)
    return y[YCOLS]
```

In [20]:
```
YCOLS = ["t0", "t1"]

def process_labels(dst_data):
    y = dst_data.copy()
    y["t0"] = y.groupby("period")["dst"].shift(-1)
    y["t1"] = y.groupby("period")["dst"].shift(-2)
    return y[YCOLS]

labels = process_labels(dst_data)
labels.head()
```

Out[20]:

		t0	t1
period	timedelta		
test_a	0 days 00:00:00	15.0	15.0
	0 days 01:00:00	15.0	14.0
	0 days 02:00:00	14.0	14.0
	0 days 03:00:00	14.0	14.0
	0 days 04:00:00	14.0	14.0

In [21]:
```
data = labels.join(features)
data.head()
```

Out[21]:

		t0	t1	bt_mean	bt_std	temperature_mean	temperature_std	bx_gse_mean	bx_gse_std	by_gse_mean	by_gse_std	...	bz_gsm_
period	timedelta												
test_a	0 days 00:00:00	15.0	15.0	-0.311506	-0.166852	-0.670084	-0.546331	-0.758075	-0.583824	0.905309	-0.281836	...	-0.088
	0 days 01:00:00	15.0	14.0	-0.257180	-0.121644	-0.633529	-0.547793	-0.788477	-0.357912	0.690187	-0.521750	...	-0.446
	0 days 02:00:00	14.0	14.0	-0.113797	-0.179912	-0.626987	-0.484512	-1.043192	-0.444493	0.844946	-0.416686	...	-0.101
	0 days 03:00:00	14.0	14.0	0.007134	-0.614577	-0.627023	-0.582198	-1.176432	-0.991267	0.914057	-0.984871	...	-0.895
	0 days 04:00:00	14.0	14.0	-0.042030	-0.260113	-0.553776	-0.488325	-1.217805	-0.620529	0.697673	-0.624028	...	-0.275

5 rows × 31 columns

In [22]:
```
def split_data_periods(data, test_per_period, val_per_period):
    """Partition the data into training, testing, and validation sets"""
    # Selecting the initial `test_per_period` rows from each time period for testing
    test = data.groupby("period").head(test_per_period)
    interim = data[~data.index.isin(test.index)]
    # Choosing the initial `val_per_period` rows from each period for validation
    val = data.groupby("period").head(val_per_period)
    # Assigning the remaining rows to the training set
    train = interim[~interim.index.isin(val.index)]
    return train, test, val

train, test, val = split_data_periods(data, test_per_period=6_000, val_per_period=10_000)
```

In [23]:
```
ind = [0, 1, 2]
names = ["train_a", "train_b", "train_c"]
width = 0.75
train_cnts = [len(df) for _, df in train.groupby("period")]
val_cnts = [len(df) for _, df in val.groupby("period")]
test_cnts = [len(df) for _, df in test.groupby("period")]

p1 = plt.barh(ind, train_cnts, width)
p2 = plt.barh(ind, val_cnts, width, left=train_cnts)
p3 = plt.barh(ind, test_cnts, width, left=np.add(val_cnts, train_cnts).tolist())

plt.yticks(ind, names)
plt.ylabel("Period")
plt.xlabel("Hourly Timesteps")
plt.title("Train/Validation/Test Splits over Periods", fontsize=16)
plt.legend(["Train", "Validation", "Test"])
```

Out[23]: <matplotlib.legend.Legend at 0x7f943309e440>

Out[23]: <matplotlib.legend.Legend at 0x7f943309e440>

In [24]:
```
print(train.shape)
train.head()
```

(29184, 31)

In [24]:
```
print(train.shape)
train.head()
```

```
(29184, 31)
```

Out[24]:

		t0	t1	bt_mean	bt_std	temperature_mean	temperature_std	bx_gse_mean	bx_gse_std	by_gse_mean	by_gse_std	...	bz_gsm_
period	timedelta												
test_a	416 days 16:00:00	-18.0	-22.0	0.459685	-0.271635	0.116395	-0.251469	-0.942910	0.151364	1.095667	0.700215	...	-0.216
	416 days 17:00:00	-22.0	-23.0	0.127263	0.341552	0.115467	-0.155359	-0.379770	0.469242	1.342341	-0.440601	...	-0.386
	416 days 18:00:00	-23.0	-25.0	0.132475	0.501878	0.093246	-0.258960	-0.456039	1.192939	1.306872	-0.590210	...	-0.106
	416 days 19:00:00	-25.0	-28.0	0.420444	-0.118972	0.055036	-0.344604	-1.319762	0.389049	0.506140	0.999787	...	-0.672
	416 days 20:00:00	-28.0	-30.0	0.192114	0.615883	0.250930	0.138736	-0.920219	1.185998	0.883592	0.868869	...	0.415

5 rows × 31 columns

In [25]:
```
print(test.shape)
test.head()
```

```
(18000, 31)
```

Out[25]:

		t0	t1	bt_mean	bt_std	temperature_mean	temperature_std	bx_gse_mean	bx_gse_std	by_gse_mean	by_gse_std	...	bz_gsm_
period	timedelta												
	0 days 00:00:00	15.0	15.0	-0.311506	-0.166852	-0.670084	-0.546331	-0.758075	-0.583824	0.905309	-0.281836	...	-0.088

In [26]:
```
from keras import preprocessing


data_config = {
    "timesteps": 128,
    "batch_size": 128,
}


def timeseries_dataset_from_df(df, batch_size):
    dataset = None
    timesteps = data_config["timesteps"]

    # iterate through periods
    for _, period_df in df.groupby("period"):
        # realign features and labels so that first sequence of 32 is aligned with the 33rd target
        inputs = period_df[XCOLS][:-timesteps]
        outputs = period_df[YCOLS][timesteps:]

        period_ds = preprocessing.timeseries_dataset_from_array(
            inputs,
            outputs,
            timesteps,
            batch_size=batch_size,
        )

        if dataset is None:
            dataset = period_ds
        else:
            dataset = dataset.concatenate(period_ds)

    return dataset

train_ds = timeseries_dataset_from_df(train, data_config["batch_size"])
val_ds = timeseries_dataset_from_df(val, data_config["batch_size"])
```

```
print(f"Number of train batches: {len(train_ds)}")
print(f"Number of val batches: {len(val_ds)}")
```

```
2023-11-20 16:03:48.154496: E tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:268] failed call to cuInit: CUDA_E
RROR_NO_DEVICE: no CUDA-capable device is detected
```

```
Number of train batches: 224
Number of val batches: 231
```

In [27]:

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, LSTM,GRU, Bidirectional, Dropout, Embedding, Input, Flatten
from tensorflow.keras.models import Sequential
import keras
```

In [28]:

```
from tensorflow.keras.layers import Dense, LSTM,GRU, Bidirectional, Dropout, GlobalAveragePooling1D,Input,Concatenate,Flatter
from tensorflow.keras.models import Sequential

# define dummy model
model_config = {"n_epochs": 100, "n_neurons": 244*2 , "dropout": 0.0, "stateful": False}

input1 = Input(shape=( data_config["timesteps"], len(XCOLS)), name='x1')
lstm1= Bidirectional(LSTM(
        model_config["n_neurons"],
        stateful=model_config["stateful"],
        dropout=model_config["dropout"],return_sequences=True
    ))(input1)
gru1= Bidirectional(GRU(
         model_config["n_neurons"] *3,
        stateful=model_config["stateful"],
        dropout=0.0,return_sequences=True
    ))(lstm1)

gaverage = Flatten() (gru1)
dense1 = Dense(96)(gaverage)
dense1 = Dense(128)(dense1)
dense1 = Dense(64)(dense1)
dense = Dense(2)(dense1)
```

```
model = keras.models.Model(inputs=input1, outputs=dense)
model.compile(
    loss='mean_squared_error',
    optimizer=tf.keras.optimizers.Adam(0.0001),
)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 x1 (InputLayer)             [(None, 128, 29)]         0

 bidirectional (Bidirection  (None, 128, 976)          2022272
 al)

 bidirectional_1 (Bidirecti  (None, 128, 2928)         21450528
 onal)

 flatten (Flatten)           (None, 374784)            0

 dense (Dense)               (None, 96)                35979360

 dense_1 (Dense)             (None, 128)               12416

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 2)                 130

=================================================================
Total params: 59472962 (226.87 MB)
Trainable params: 59472962 (226.87 MB)
Non-trainable params: 0 (0.00 Byte)
```

In [29]:

```
from tensorflow.keras.layers import Dense, LSTM,GRU, Bidirectional, Dropout, GlobalAveragePooling1D,Input,Concatenate,Flatten
from tensorflow.keras.models import Sequential

# defining  model
model_config = {"n_epochs": 30, "n_neurons": 192*2 , "dropout": 0.4, "stateful": False}

input1 = Input(shape=( data_config["timesteps"], len(XCOLS)), name='x1')
lstm1= Bidirectional(LSTM(
        model_config["n_neurons"],
        stateful=model_config["stateful"],
        dropout=model_config["dropout"],return_sequences=True
    ))(input1)
gru1= Bidirectional(GRU(
         model_config["n_neurons"] *3,
        stateful=model_config["stateful"],
        dropout=0.0,return_sequences=True
    ))(lstm1)

gaverage = Flatten() (gru1)
dense1 = Dense(96)(gaverage)
dense1 = Dense(128)(dense1)
dense1 = Dense(64)(dense1)
dense = Dense(2)(dense1)


model = keras.models.Model(inputs=input1, outputs=dense)
model.compile(
    loss='mean_squared_error',
    optimizer=tf.keras.optimizers.Adam(0.0001),
)

model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 x1 (InputLayer)             [(None, 128, 29)]         0

 bidirectional_2 (Bidirecti  (None, 128, 768)          1271808
 onal)

 bidirectional_3 (Bidirecti  (None, 128, 2304)         13284864
 onal)

 flatten_1 (Flatten)         (None, 294912)            0

 dense_4 (Dense)             (None, 96)                28311648

 dense_5 (Dense)             (None, 128)               12416

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 2)                 130

=================================================================
Total params: 42889122 (163.61 MB)
Trainable params: 42889122 (163.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

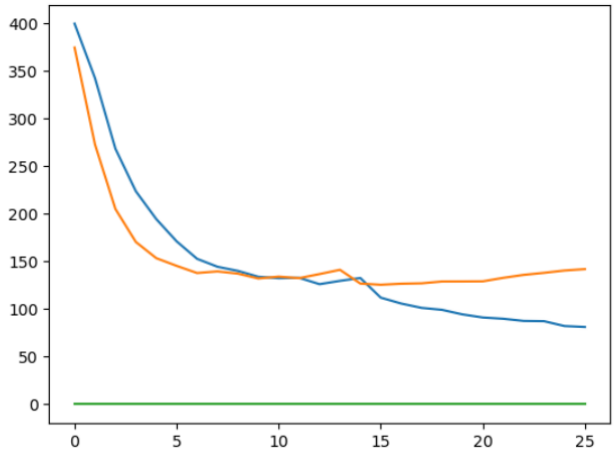
In [ ]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
checkpoint = ModelCheckpoint('model2', save_best_only=True, monitor='val_loss', mode='min')
lr_reducer = ReduceLROnPlateau(monitor="val_loss", factor=0.3, patience=4, min_lr=1e-10, mode='min')
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, mode='min')

history = model.fit(
    train_ds,
    batch_size=data_config["batch_size"],
    epochs=model_config["n_epochs"],
    verbose=2,
    callbacks=[checkpoint, lr_reducer, early_stopping],  # Add early_stopping callback
    shuffle=False,
    validation_data=val_ds,
)
```

```
Epoch 1/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1703s - loss: 400.0068 - val_loss: 374.8605 - lr: 1.0000e-04 - 1703s/epoch - 8s/step
Epoch 2/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1690s - loss: 342.7743 - val_loss: 273.0658 - lr: 1.0000e-04 - 1690s/epoch - 8s/step
Epoch 3/30
224/224 - 1667s - loss: 144.6048 - val_loss: 139.7158 - lr: 1.0000e-04 - 1667s/epoch - 7s/step
Epoch 9/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1680s - loss: 140.2694 - val_loss: 137.3792 - lr: 1.0000e-04 - 1680s/epoch - 8s/step
Epoch 10/30
INFO:tensorflow:Assets written to: model2/assets
```

```
224/224 - 1685s - loss: 134.0841 - val_loss: 132.2754 - lr: 1.0000e-04 - 1685s/epoch - 8s/step
Epoch 11/30
224/224 - 1667s - loss: 132.6075 - val_loss: 134.2417 - lr: 1.0000e-04 - 1667s/epoch - 7s/step
Epoch 12/30
224/224 - 1667s - loss: 133.0047 - val_loss: 132.6478 - lr: 1.0000e-04 - 1667s/epoch - 7s/step
Epoch 13/30
224/224 - 1666s - loss: 126.3002 - val_loss: 136.9062 - lr: 1.0000e-04 - 1666s/epoch - 7s/step
Epoch 14/30
224/224 - 1666s - loss: 129.7750 - val_loss: 141.4141 - lr: 1.0000e-04 - 1666s/epoch - 7s/step
Epoch 15/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1683s - loss: 132.7793 - val_loss: 126.9536 - lr: 3.0000e-05 - 1683s/epoch - 8s/step
Epoch 16/30
INFO:tensorflow:Assets written to: model2/assets
INFO:tensorflow:Assets written to: model2/assets
224/224 - 1681s - loss: 112.1543 - val_loss: 125.7035 - lr: 3.0000e-05 - 1681s/epoch - 8s/step
Epoch 17/30
224/224 - 1664s - loss: 105.9699 - val_loss: 126.7935 - lr: 3.0000e-05 - 1664s/epoch - 7s/step
Epoch 18/30
224/224 - 1675s - loss: 101.4018 - val_loss: 127.2110 - lr: 3.0000e-05 - 1675s/epoch - 7s/step
Epoch 19/30
224/224 - 1669s - loss: 99.3504 - val_loss: 129.1278 - lr: 3.0000e-05 - 1669s/epoch - 7s/step
Epoch 20/30
224/224 - 1670s - loss: 94.5765 - val_loss: 129.1732 - lr: 3.0000e-05 - 1670s/epoch - 7s/step
Epoch 21/30
224/224 - 1670s - loss: 91.2906 - val_loss: 129.2833 - lr: 9.0000e-06 - 1670s/epoch - 7s/step
Epoch 22/30
224/224 - 1673s - loss: 89.9800 - val_loss: 132.9616 - lr: 9.0000e-06 - 1673s/epoch - 7s/step
Epoch 23/30
224/224 - 1663s - loss: 87.6888 - val_loss: 136.0630 - lr: 9.0000e-06 - 1663s/epoch - 7s/step
Epoch 24/30
224/224 - 1665s - loss: 87.4311 - val_loss: 138.2168 - lr: 9.0000e-06 - 1665s/epoch - 7s/step
Epoch 25/30
224/224 - 1665s - loss: 82.3346 - val_loss: 140.6573 - lr: 2.7000e-06 - 1665s/epoch - 7s/step
Epoch 26/30
224/224 - 1666s - loss: 81.3231 - val_loss: 142.1371 - lr: 2.7000e-06 - 1666s/epoch - 7s/step
```

In [ ]:
```
model = keras.models.load_model("model2")
```

In [ ]:
```
for name, values in history.history.items():
    plt.plot(values)
```



In [ ]:
```
test_ds = timeseries_dataset_from_df(test, data_config["batch_size"])
```

In [ ]:
```
mse = model.evaluate(test_ds)
print(f"Test RMSE: {mse**.5:.2f}")
```

```
135/135 [==============================] - 256s 2s/step - loss: 129.9196
Test RMSE: 11.40
```

In [ ]:
```
test_ds = timeseries_dataset_from_df(test, data_config["batch_size"])
```

In [ ]:
```
mse = model.evaluate(test_ds)
print(f"Test RMSE: {mse**.5:.2f}")
```

```
135/135 [==============================] - 256s 2s/step - loss: 129.9196
Test RMSE: 11.40
```

In [ ]:
```
# Assuming  model is already trained, and  have 'test_ds' prepared from the test split
predictions = model.predict(test_ds)

# Extracting actual values
actual_values = []
for batch in test_ds:
    actual_values.append(batch[1].numpy())  # Assuming the targets are in the second position

actual_values = np.vstack(actual_values)
t0_actual = actual_values[:, 0]  #  t0 is the first column in  targets
t1_actual = actual_values[:, 1]  #  t1 is the second column in ] targets

# Extracting predicted values
t0_predicted = predictions[:, 0]  #  t0 prediction is the first column in  predictions
t1_predicted = predictions[:, 1]  #  t1 prediction is the second column in  predictions
```

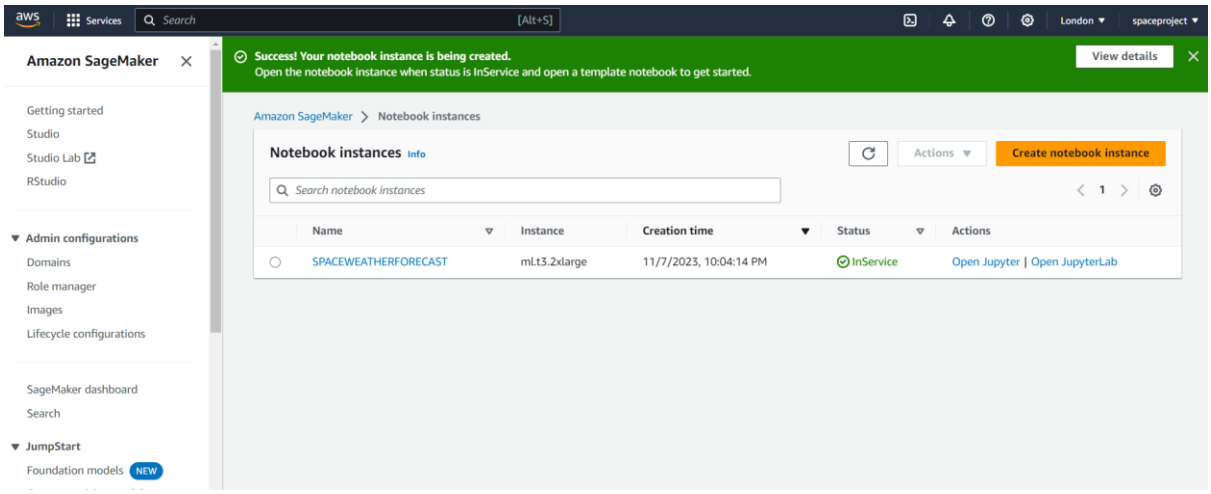
```
135/135 [==============================] - 256s 2s/step
```

In [ ]:
```
# Printing the actual and predicted values for t0 and t1
print("Actual t0 values:", t0_actual)
print("Predicted t0 values:", t0_predicted)

print("Actual t1 values:", t1_actual)
print("Predicted t1 values:", t1_predicted)
```

```
Actual t0 values: [ 10.   8.   8. ...  -6.  -7. -10.]
Predicted t0 values: [ -3.198237   -3.3976085  -2.9867146 ... -19.720472  -18.262754
 -16.745512 ]
Actual t1 values: [  8.   8.   8. ...  -7. -10. -11.]
Predicted t1 values: [ -4.041156   -4.4375772  -3.7887297 ... -19.197582  -17.53423
 -16.394438 ]
```

Appendix D- Amazon SageMaker Utilization

Sample screenshot proving utilization of AWS SageMaker Jupyter notebook with a notebook instance of affordable price for better computational performance than local machine.



Steps followed to access AWS SageMaker Jupyter Notebook is as follows:

- Create an AWS account and open Amazon SageMaker console consolehttps://console.aws.amazon.com/sagemaker/
- Create notebook instance by specifying notebook instance name and type of instance.
- There are several types of instances available from Free tier version to expensive ones designed for creating different models (depends on complexity levels), I chose an instance considering the cost factor.
- Instances can be chosen based on requirements such as Instance Size, vCPUs, Instance Memory (GiB), GPU Model, GPUs Total, GPU memory (GB), Memory per GPU (GB), Network Bandwidth (Gbps), Instance Storage (GB), EBS Bandwidth (Gbps).
- Details of instances and its pricing can be viewed in https://aws.amazon.com/sagemaker/pricing/
- Commonly ml.p4 and ml.g4 instances are designed for complex Deep Learning models are they are inbuilt with NVIDIA A100 Tensor Core GPUs, NVIDIA T4 GPUs, which helps in executing and training models much faster.
- Hence utilizing SageMaker as it is a fully managed service, has a broad framework support, scalable etc.
- There is a need to start and stop the instance (server) when not in use to avoid extra charges.