PROJECT REPORT

King Mongkut's Institute of Technology Ladkrabang
School of Engineering – Robotics and AI Engineering: Section 1

01416632 SPACE SCIENCE
Academic Year 2024

Forecasting the Disturbance Storm Time Index (DST) with Deep Learning

Instructed by: Mr. Gleb Mutasov, Mr. Efren Martin Alban Cuestas

Members:

Jadetapat Tuntipitukkul	65011307
Jing Chaichan	65011312
Pannathorn Panomtham	65011426

Date of submission
Mar 1, 2025

Introduction

The Dst-index (Disturbance Storm Time index) is a key measure of geomagnetic activity, representing the strength of Earth's ring current. It is calculated based on the average horizontal magnetic field observed at four low-latitude geomagnetic observatories. When the Dst-index decreases (more negative values), it indicates the presence of a geomagnetic storm, which can disrupt Earth's magnetosphere. These storms are caused by solar wind disturbances, such as coronal mass ejections (CMEs) and high-speed solar wind streams, which interact with Earth's magnetic field.

Predicting the Dst-index is important because geomagnetic storms can have serious effects on technology and infrastructure. They can cause power grid failures, damage satellites, interfere with GPS signals, and affect radio communications. In extreme cases, strong geomagnetic storms can even increase radiation exposure for astronauts and high-altitude flights. By developing accurate models to predict the Dst-index, scientists can improve space weather forecasting and help prevent damage to critical systems. This project focuses on selecting the most relevant features from solar wind and interplanetary magnetic field (IMF) data to build a reliable predictive model for the Dst-index.

Github: https://github.com/sincerem00n/DST_PRED_ZERO/

Methodology

1. Feature Selection

In this project, we use distinctive features to predict the **Dst-index**, which measures geomagnetic activity caused by interactions between the solar wind and Earth's magnetosphere. **Scalar_B**, the total magnetic field strength, is included because stronger interplanetary magnetic fields (IMF) can transfer more energy into the magnetosphere. The **X, Y, and Z components of the IMF** in **GSE (Geocentric Solar Ecliptic) and GSM (Geocentric Solar Magnetospheric) coordinates** help to describe the direction and structure of the magnetic field. Among these, **BZ_GSM** is particularly important because when it is negative (southward), it allows strong interaction between the solar wind and Earth's magnetic field, which can cause geomagnetic storms.

Also, the **proton density** in the solar wind affects the pressure on Earth's magnetic field, which influences storm strength. The **solar wind plasma temperature** is another key factor because higher temperatures mean more energetic particles, which can make geomagnetic activity stronger. The **solar wind speed** is also important because faster winds bring more energy into Earth's magnetosphere and affect the **Dst-index**. These features are selected to make sure the model can predict geomagnetic storms well. This can help us to forecast space weather and reduce its impact on technology and communication systems on Earth.

In the feature selection phase of our DST prediction model, we employed a correlation analysis approach to identify the most relevant input parameters. The correlation matrix (Figure 1) provided critical insights into the relationships between various solar wind and interplanetary magnetic field parameters and the DST index.

1.1. Correlation Analysis

We calculated the Pearson correlation coefficients between all features to quantify their linear relationships. The heat map visualization revealed several important patterns:

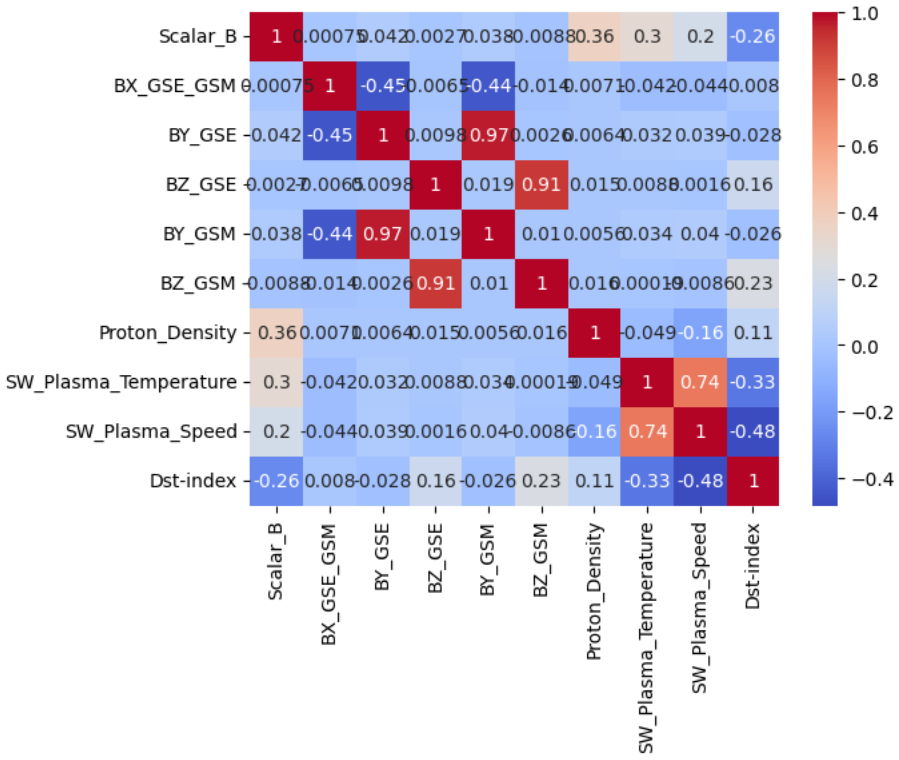

Figure 1: The correlation matrix of features

Solar Wind Plasma Parameters: Solar Wind Plasma Speed showed a moderate negative correlation (-0.48) with the DST index, indicating that higher solar wind speeds are associated with more negative DST values (stronger geomagnetic storms). Similarly, Solar Wind Plasma Temperature exhibited a negative correlation (-0.33).

Magnetic Field Components: The analysis revealed strong correlations between BY_GSE and BY_GSM (0.97), as well as between BZ_GSE and BZ_GSM (0.91). This high collinearity suggested potential redundancy in these features.

Scalar_B: The magnetic field magnitude (Scalar_B) showed a weak negative correlation (-0.26) with the DST index, suggesting some relationship with geomagnetic activity.

Proton Density: This parameter displayed a weak positive correlation (0.11) with the DST index, indicating a limited direct linear relationship with storm intensity.

1.2. Selection Criteria

Based on the correlation analysis, we selected features that:

- Demonstrated meaningful correlations with the DST index
- Provided non-redundant information (avoiding highly correlated pairs)
- Represented different physical aspects of the solar wind-magnetosphere coupling process

This approach allowed us to optimize our input feature set, reducing dimensionality while preserving the most relevant geophysical information for our CNN-LSTM model. The selected features captured both the magnetic field orientation and solar wind plasma conditions that drive geomagnetic storm development.

2. Dataset Preparation

We have compiled our dataset from multiple sources. The Dst index is sourced from WDC Kyoto [1], interplanetary magnetic field data [2], and solar wind parameters [3] from ACE Science Center (ASC). The available periods have some differences between the three datasets, so we decided to acquire the data when all three sources' periods intersected which is the period between 1st January 1999 to 12th of March 2023. The datasets have an hourly resolution, with 212088 data points (Figure 2). The three datasets are exported in text file format and then prepared for training by utilizing Pandas, a Python Data Analysis Library.

	YEAR	DOY	HR	Scalar_B
0	1999	1	0	6.792
1	1999	1	1	6.884
2	1999	1	2	7.073
3	1999	1	3	6.644
4	1999	1	4	6.645
...	...	...	..	...
212083	2023	71	19	5.172
212084	2023	71	20	5.205
212085	2023	71	21	4.985
212086	2023	71	22	4.824
212087	2023	71	23	4.858

Figure 2 A total of 212,088 data points between 1st January 1999 to 12th of March 2023 in hourly resolution and Day of Year format (DOY).

The three datasets had an inconsistent amount of whitespace between columns and contained unnecessary header information and descriptive data. The aforementioned issues were resolved with manual striping and Pandas' read_table function with certain arguments given (Figure 3). The previous header names are dropped in favor of new, more concise header names.

```
mag = pd.read_table("mag.txt", sep=r'\s+', header=None,
                    names=["YEAR", "DOY", "HR",
                           "Scalar_B","BX_GSE_GSM","BY_GSE","BZ_GSE",
                           "BX_GSM_redundant","BY_GSM","BZ_GSM"])
```

Figure 3 Pandas' function reading magnitudes of solar wind

While parsing through each feature, we noticed that BX_GSE and BX_GSM have identical values, so we decided to drop the redundant feature.

```
mag = mag.drop(columns=["BX_GSM_redundant"])
```

Figure 4 Pandas' function was used to drop the redundant feature.

After all three data sets were prepared, we concatenated them together to finalize our data preparation processes.

After inspecting the final database which contains 23 years of hourly data and 9 different features, we have found some missing data points. Each database filled those missing points with extremely low values such as -999.99 or -9999.99 (see Figure 5). Unfortunately, there is no single best way to handle missing data points of a time series dataset. Deleting periods where data points are missing will cause time discontinuities which is undesirable in the forecast training dataset, while leaving them as is will also disrupt the continuity and temporal structure of the dataset.

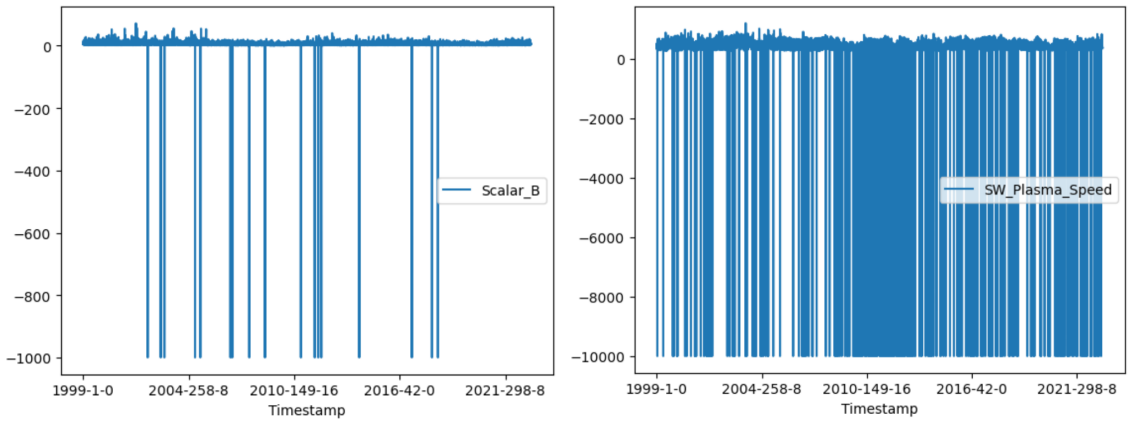

Figure 5 Graph visualization of the missing data across some features, missing data is replaced with extremely low value by the database owner.

A way to remediate this problem is to use imputation on the dataset. Imputation is a technique that replaces missing data in a dataset with estimated values. It's a way to preserve the majority of a dataset's information and structure. Among other imputation techniques, we chose “fill forward (ffill)” as the way to pad the missing data points in each feature. The reason is that the data generated will impose a minimal trend toward the model, while other data points will continue to contribute normally.

```
# Replace -999.9 values in scalar B, BX_GSE_GSM, BY_GSE, BZ_GSE, BY_GSM, BZ_GSM column with Forward fill
df["Scalar_B"] = df["Scalar_B"].replace(-999.9, method="ffill")
df["BX_GSE_GSM"] = df["BX_GSE_GSM"].replace(-999.9, method="ffill")
df["BY_GSE"] = df["BY_GSE"].replace(-999.9, method="ffill")
df["BZ_GSE"] = df["BZ_GSE"].replace(-999.9, method="ffill")
df["BY_GSM"] = df["BY_GSM"].replace(-999.9, method="ffill")
df["BZ_GSM"] = df["BZ_GSM"].replace(-999.9, method="ffill")
#print(df.describe())

# Replace -9999.9 values in Proton_Density, SW_Plasma_Speed column with Forward fill
df["Proton_Density"] = df["Proton_Density"].replace(-9999.9, method="ffill")
df["SW_Plasma_Speed"] = df["SW_Plasma_Speed"].replace(-9999.9, method="ffill")
# print(df.describe())

# Replace -9.999900e+03 values in SW_Plasma_Temperature column with Forward fill
df["SW_Plasma_Temperature"] = df["SW_Plasma_Temperature"].replace(-9.999900e+03, method="ffill")
```

Figure 6 the process of using the forward fill method to generate missing data

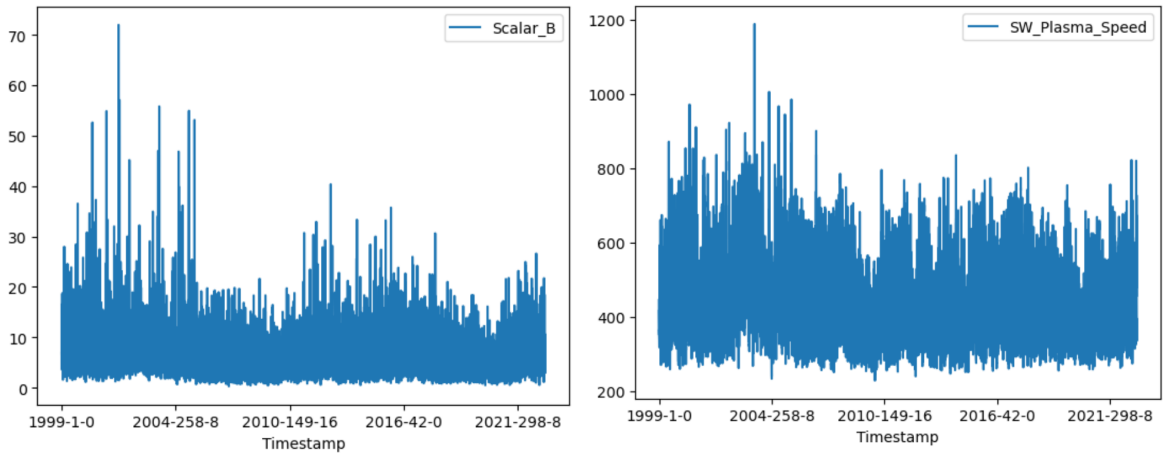

Figure 7 Graph visualization of some features after the missing data was replaced with forward fill.

3. Data reconstruction

To ensure the model could effectively learn from the data, a comprehensive data preparation and reconstruction pipeline was implemented. This pipeline included data normalization, splitting, and sequence generation to transform the raw data into a format suitable for training and evaluating the model.

Data Splitting

To ensure the model performance, we split the data into 80:20 for train and a valid set. In time series forecasting, it is critical to maintain the temporal order of the data. Splitting the data into training and testing sets sequentially (without shuffling) ensures that future data (testing set) is not used to train the model. This prevents data leakage, which can lead to overly optimistic performance metrics.

Data Normalization

The raw data was normalized to ensure that all features were on a similar scale, which is crucial for the stability and convergence of deep learning models.

```
# Scale X
scaler_X = MinMaxScaler(feature_range=(-1, 1))
X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled = scaler_X.transform(X_test)

# Scale y (if regression)
scaler_y = MinMaxScaler(feature_range=(-1, 1))
y_train_scaled = scaler_y.fit_transform(y_train)
y_test_scaled = scaler_y.transform(y_test)
```

Figure 8 MaxMinScaler from sklearn. Preprocessing is implemented to train and test the dataset.

The *MinMaxScaler* from the *sklearn.The preprocessing* library was used to scale the features and target values to the range [−1,1]. This scaling range was chosen to preserve the sign of the data, which is important for time series forecasting tasks like DST index prediction.

Sequence Splitting

```
# Split the data into sequences
def sequence(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, :-1]) # all columns except the last
        y.append(data[i+seq_length, -1])  # target is the last column
        # print(f'X:{np.array(X)}, Y:{np.array(y)}')
    return np.array(X), np.array(y)

SEQ_LENGTH = 24
X_train, y_train = sequence(train_data, SEQ_LENGTH)
X_test, y_test = sequence(test_data, SEQ_LENGTH)
```

Figure 9 data sequence function for splitting the data

To leverage the temporal nature of the DST index data, the dataset was split into sequences of a fixed length *SEQ_LENGTH = 24*. This step transformed the data into a format suitable for training sequence-based models like LSTM.

Final Dataset Structure

The final training and testing datasets were structured as follows:

- Input (X): A 3D array of shape (*num_samples, SEQ_LENGTH, num_features*), where *num_samples* is the number of sequences, *SEQ_LENGTH* is the sequence length, and *num_features* is the number of input features.
- Target (y): A 1D array of shapes (*num_samples*,), representing the DST index value to be predicted for each sequence.

This structured dataset was then used to train and evaluate the model, ensuring that the temporal dependencies in the data were effectively captured.

4. Model Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN_LSTM                                 [32, 1]                   --
├─Conv1d: 1-1                            [32, 32, 24]              896
├─MaxPool1d: 1-2                         [32, 32, 12]              --
├─LSTM: 1-3                              [32, 12, 256]             823,296
├─Dropout: 1-4                           [32, 256]                 --
├─Linear: 1-5                            [32, 1]                   257
==========================================================================================
Total params: 824,449
Trainable params: 824,449
Non-trainable params: 0
Total mult-adds (M): 316.84
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.98
Params size (MB): 3.30
Estimated Total Size (MB): 4.31
==========================================================================================
```

Figure 10 CNN-LSTM Model Architecture for Dst-Index Prediction from Pytorch's model summary

Input Layer	Takes in data with a batch size of 32 and a sequence of 9 features.
Conv1d Layer (1-1)	- Output shape: [32, 32, 24] - Parameters: 896 - This applies 32 filters with a kernel that preserves temporal patterns in our time series data - The Conv1d layer captures local patterns in the solar/geomagnetic data
MaxPool1d Layer (1-2)	- Output shape: [32, 32, 12] - No trainable parameters - Reduces the sequence length by half (from 24 to 12)
LSTM Layer (1-3)	- Output shape: [32, 12, 256] - Parameters: 823,296 - Processes the sequence data with 256 hidden units, maintaining the sequence length of 12 - The LSTM layer models long-term dependencies in the time series
Dropout Layer (1-4)	- Output shape: [32, 256] - No parameters - Helps prevent overfitting
Linear Layer (1-5)	- Output shape: [32, 1] - Parameters: 257 - Final output layer that produces a single prediction value for each sample

Table 1 CNN-LSTM Model by-layer explanation

Time series nature of geomagnetic data: DST (Disturbance Storm Time) is a geomagnetic index that captures variations in the Earth's magnetic field. This model combines CNNs and LSTMs specifically to handle the temporal patterns in this data.

Multi-scale feature extraction:

- The Conv1d layer captures local, short-term patterns in solar wind parameters and interplanetary magnetic field components that directly influence geomagnetic activity
- The LSTM component models longer-term dependencies and the progressive development of geomagnetic storms

Handling complex solar-terrestrial interactions: Geomagnetic storms result from complex interactions between solar wind, the interplanetary magnetic field, and Earth's magnetosphere. The multi-layered approach can model these nonlinear relationships.

Efficient dimensionality reduction: The MaxPool1d layer reduces sequence length without losing critical information, making the model computationally efficient while preserving key features.

The balance between complexity and efficiency: With 824,449 parameters, the model is sophisticated enough to capture complex patterns but small enough (4.31 MB) to deploy practically.

Dropout for robustness: Space weather prediction involves inherent uncertainties; the dropout layer helps the model generalize rather than overfit the specific storm patterns in the training data. For this model, we set dropout at 0.2

Long dataset compatibility: A 20-year dataset (212,088 samples) provides exposure to multiple solar cycles and numerous storm types, which this architecture can effectively learn from.

5. Training Parameters

We trained the CNN-LSTM model, the best model configuration from various tests, using the following parameters:

- Number of epochs: 150
- Optimizer: AdamW with an initial learning rate of 1e-3 and weight decay of 0.01
- Loss function: Mean Squared Error (MSE)

Advanced Training Techniques

1. Gradient Accumulation

   We implemented gradient accumulation with 4 steps to effectively increase the batch size without requiring additional memory, allowing the model to see more diverse examples before each parameter update.

2. Mixed Precision Training

   To accelerate training and reduce memory usage, we employed mixed precision training using PyTorch's AMP (Automatic Mixed Precision) with CUDA support. This technique allowed computations in lower precision (FP16) where appropriate while maintaining numerical stability.

3. Learning Rate Scheduling
   We utilized the **OneCycleLR** scheduler with:
   - Maximum learning rate: 1e-4
   - Warmup period: 10% of total training steps
   - Division factors: 50 (initial) and 100 (final)
   - Cosine annealing strategy

   This scheduling approach helped the model converge faster and find better minima by dynamically adjusting the learning rate throughout training.

4. Gradient Clipping

   To prevent exploding gradients, which can be particularly problematic in recurrent neural networks like LSTMs, we applied gradient norm clipping with a maximum threshold of 1.0.

5. Early Stopping

   We implemented early stopping with:

   - Patience: 15 epochs
   - Monitoring: Test loss
   - Significant improvement threshold: 0.1%

   Training would terminate if no significant improvement were observed after 15 consecutive epochs, preventing overfitting, and reducing unnecessary computation.

Result

The model achieved an MSE of 60.01 on the test dataset. This result indicates that the CNN-LSTM architecture is capable of capturing the temporal dependencies and spatial features present in the DST time series data. The convolutional layer extracts local patterns from the input sequence, while the LSTM layer effectively models the long-term dependencies in the data. The dropout layer helps prevent overfitting, ensuring that the model generalizes well to unseen data.

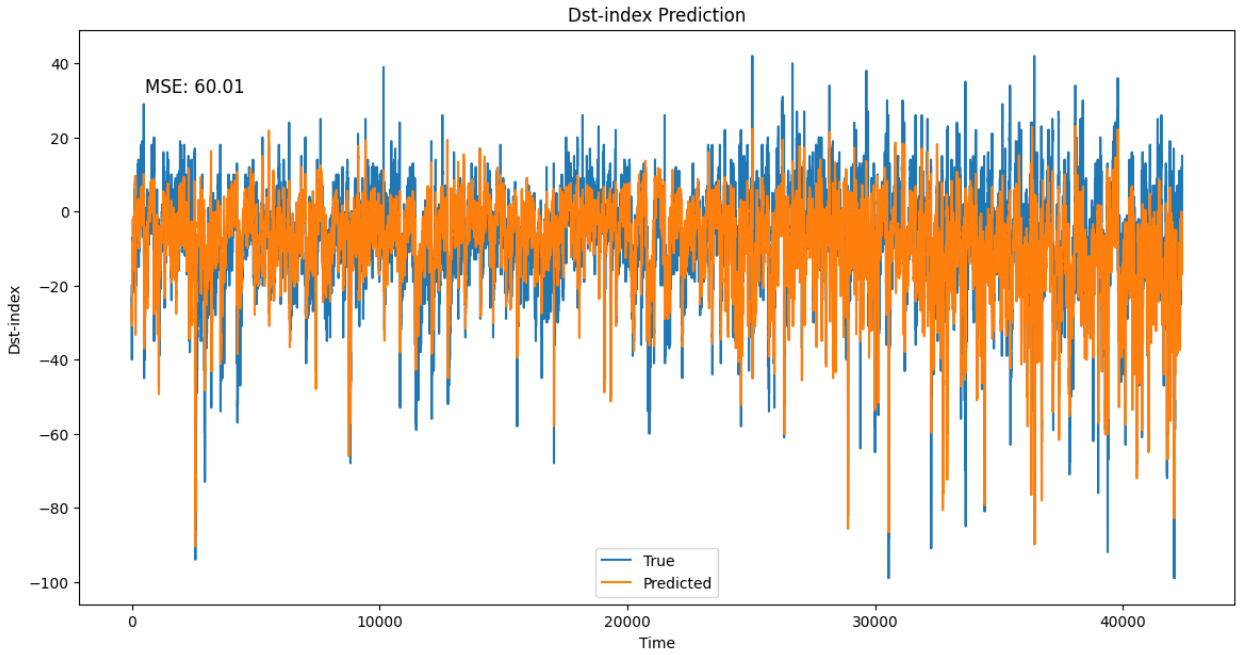

Figure 11 the result of the model predicting test data

The relatively low MSE value suggests that the model performs well in predicting the DST index, which is crucial for space weather forecasting. However, there is still room for improvement, as the MSE could potentially be reduced further by fine-tuning hyperparameters, increasing the complexity of the model, or incorporating additional features into the input data.

Discussion

1. Underfitting

The model exhibited significant underfitting, as indicated by the training loss consistently performing worse than the test loss. This suggests that the model was unable to capture the underlying patterns in the training data effectively. Despite extending the training duration, the model's learning stagnated, and training was often halted by the early stopping mechanism (with patience of 15 epochs), indicating no significant improvement in performance.

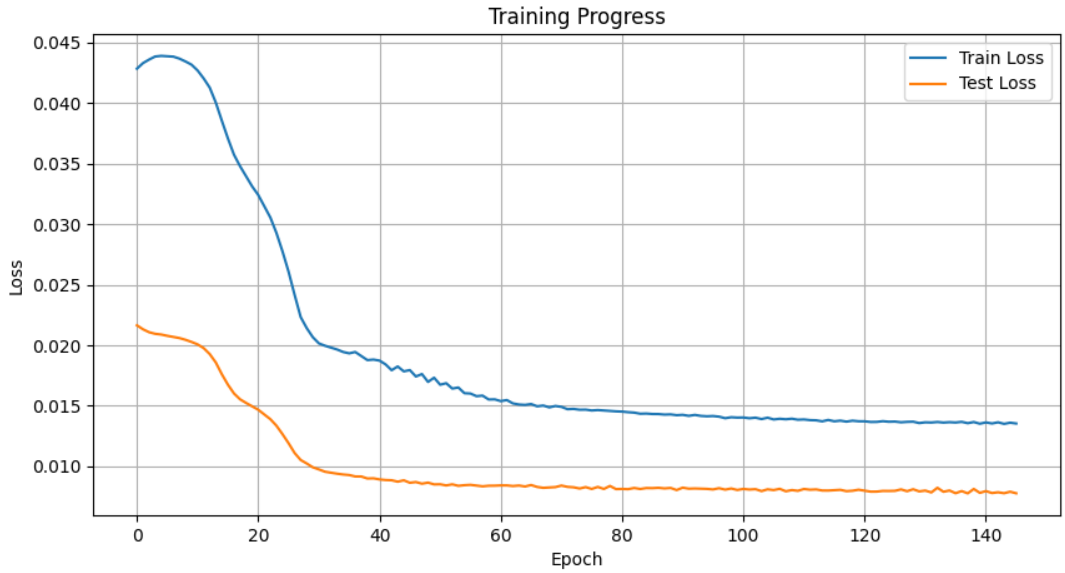

Figure 12 Learning curve graph of run10

The training loss and validation loss curves in Figure 12 show a gradual decrease initially, indicating that the model was learning from the data. However, the rate of improvement slowed down significantly after a few epochs, suggesting that the model struggled to further minimize the loss. This behavior suggests that the model was not able to fully capture the complexity of the training data, leading to suboptimal performance on both the training and validation sets.

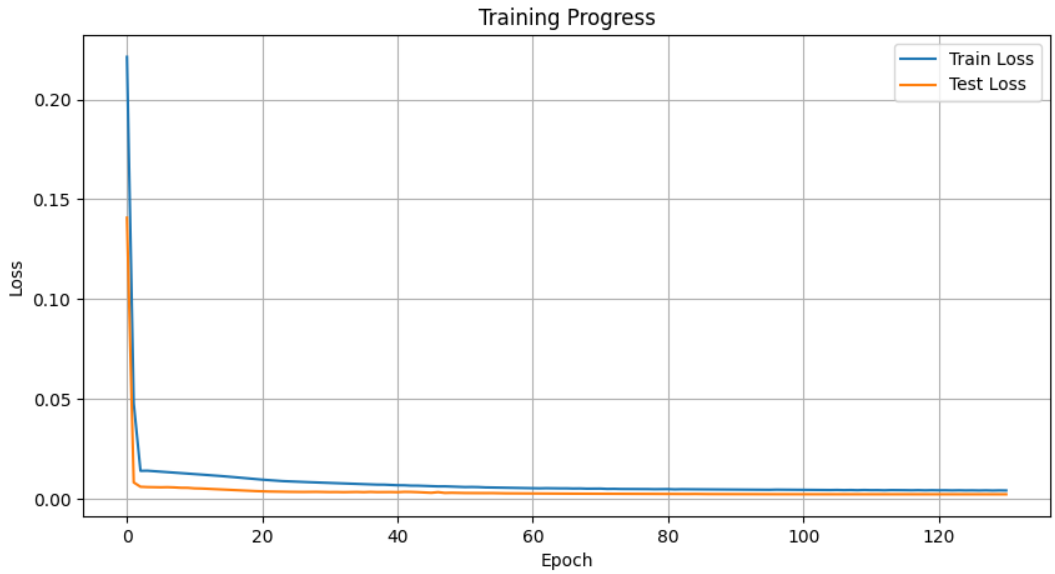

Figure 13 Learning curve graph of run8

Among the various configurations tested, *run8*, which included a hidden layer of size 128, demonstrated a unique behavior in its learning curve. While this configuration achieved the closest alignment between

training loss and test loss compared to other configurations, it resulted in a higher Mean Squared Error (MSE) of 70.41 (Figure 13) on the test dataset.

The inclusion of a hidden layer with 128 units helped reduce the gap between training and test losses, demonstrating improvement in generalization compared to other configurations. However, this improvement still came at the cost of higher overall error as reflected in increased MSE.

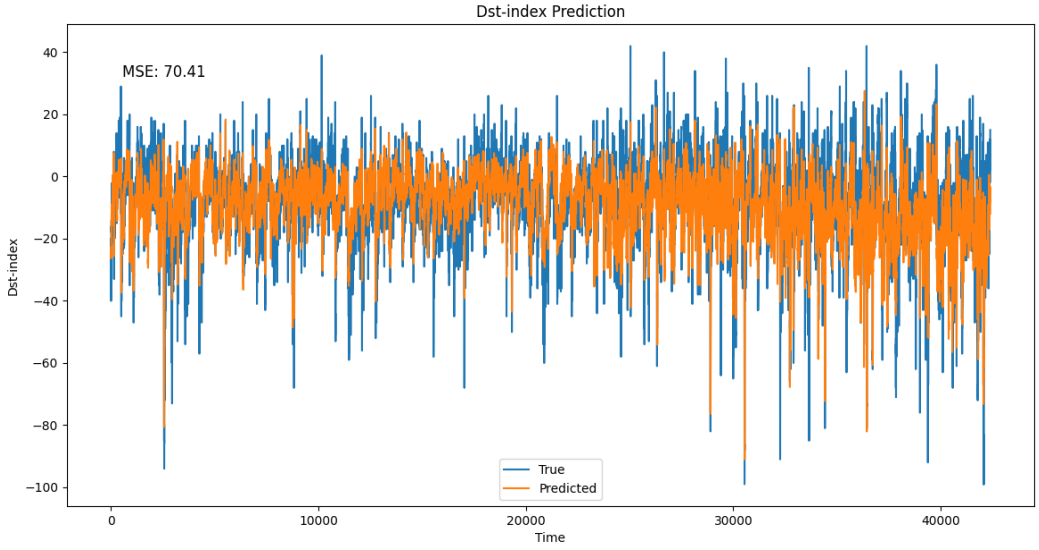

Figure 14 the result of the DST-index prediction of run8

2. Complexity Adjustment

To address underfitting, the model's complexity was increased by adding additional hidden layers. While this approach slightly reduced underfitting, it led to an increase in the Mean Squared Error (MSE). This trade-off suggests that while increasing complexity helped the model fit the training data better, it may have introduced noise or overcomplicated the learning process, resulting in poorer generalization of the test data.

In an effort to improve the model's performance, the DSTNET architecture in Figure 15 from [5] was experimented with as an alternative to the CNN-LSTM model. DSTNET, a more complex and specialized architecture designed for time series forecasting, was expected to better capture the complicated patterns in DST index data.

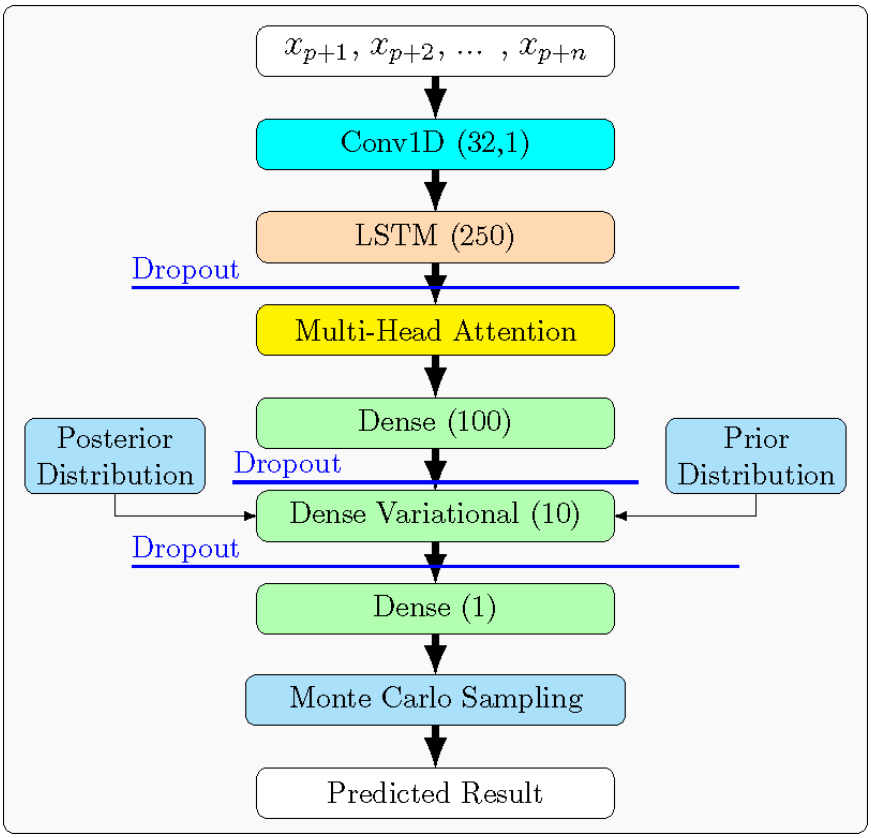

Figure 15 DSTNET model architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
DSTNET                                   [32, 1]                   --
├─Conv1d: 1-1                            [32, 32, 24]              896
├─LSTM: 1-2                              [32, 24, 256]             823,296
├─Dropout: 1-3                           [32, 24, 256]             --
├─MultiheadAttention: 1-4                [32, 24, 256]             263,168
├─Dropout: 1-5                           [32, 24, 256]             --
├─Linear: 1-6                            [32, 400]                 102,800
├─Dropout: 1-7                           [32, 400]                 --
├─Linear: 1-8                            [32, 100]                 40,100
├─Dropout: 1-9                           [32, 100]                 --
├─Linear: 1-10                           [32, 1]                   101
==========================================================================================
Total params: 1,230,361
Trainable params: 1,230,361
Non-trainable params: 0
Total mult-adds (M): 637.56
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 1.90
Params size (MB): 3.87
Estimated Total Size (MB): 5.79
==========================================================================================
```

Figure 16 DSTNET model architecture from Pytorch's model summary

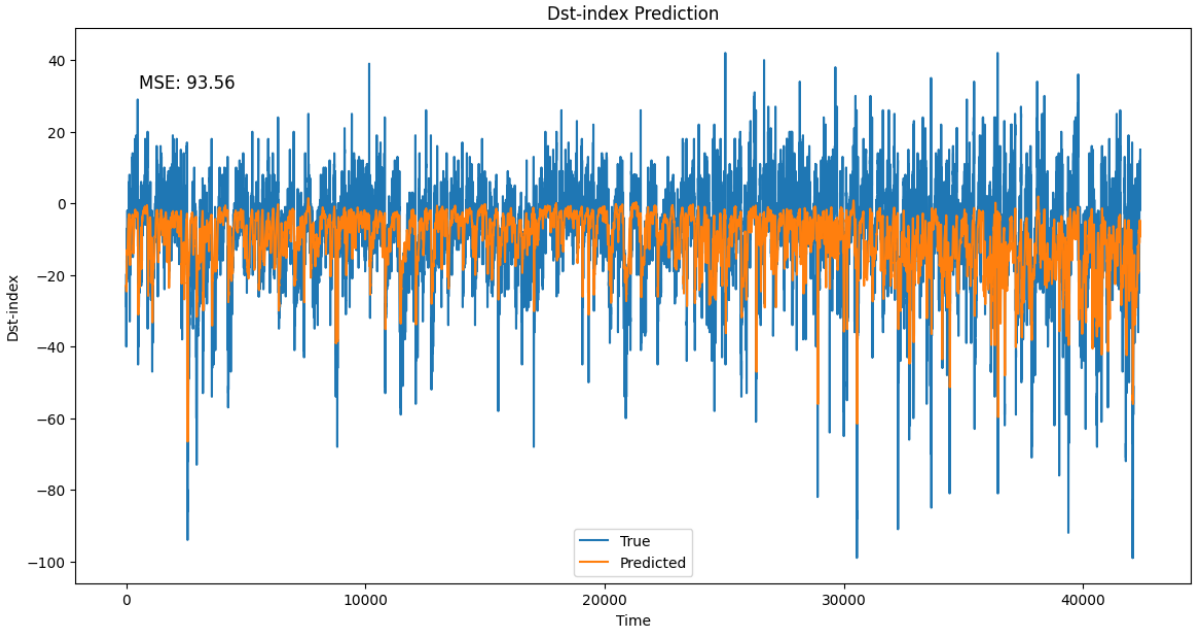

Figure 17 the result of DST-index prediction from the DSTNET model

However, the result was unreasonable, with the model achieving a significantly higher MSE of 93.56 compared to the CNN-LSTM model's MSE of 60.01

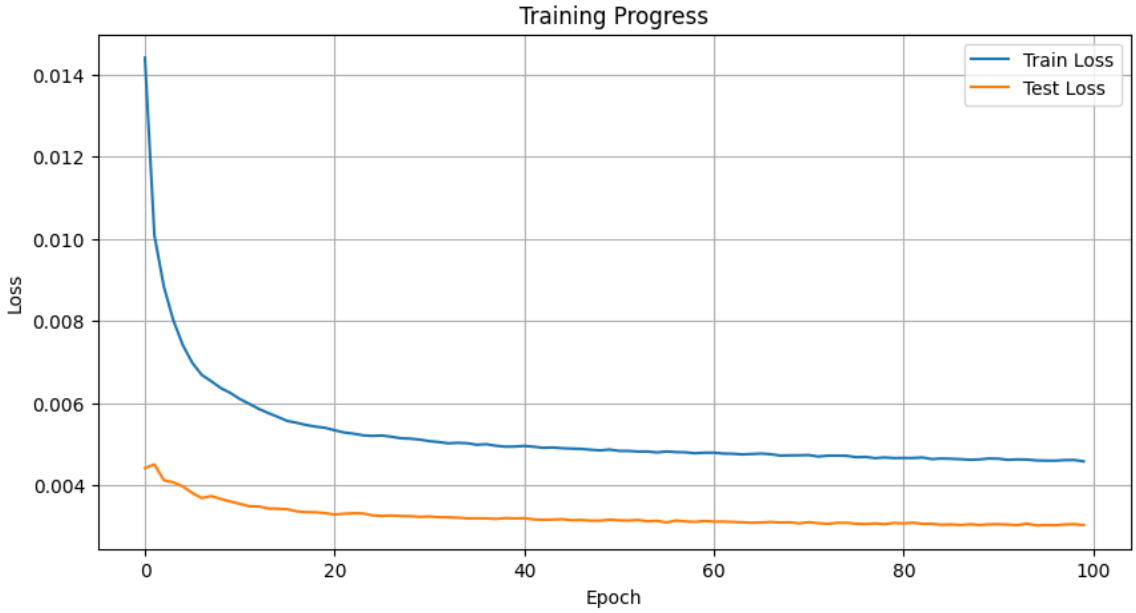

Figure 18 Learning curve graph for the DSTNET model

The learning curve for DSTNET displayed that the model was underfitting, as both the training and validation losses remained high and closely aligned throughout the training process. This behavior indicates that the model was incapable of learning meaningful patterns from the data.

Despite its advanced design, DSTNET underperformed, suggesting that increased model complexity does not always translate to better results. The training process for DSTNET may have been hindered by suboptimal hyperparameters, insufficient training data, or difficulties in converging to a meaningful solution. These challenges highlight the importance of balancing model complexity with the dataset's characteristics.

3. Trade-off Between Generalization and Performance

In Figure 14, *run8* highlights a trade-off between achieving better alignment of training and test losses (reduced underfitting) and minimizing the overall error. While the model generalized better, its predictive performance was compromised, resulting in a higher MSE.

4. Early Stopping

The training process was usually interrupted by the early stopping with patience of 15 epochs, as the model failed to show significant improvement in the validation loss over consecutive epochs. This incident further confirms that the model is unable to learn effectively from certain points even though we want to have the model to learn more as increasing epochs.

5. Lack of improvement in Loss

Both the training and validation loss curves plateaued relatively early in the training process, indicating that the model reached a local minimum and was unable to escape it which this behavior aligns with the underfitting issue observed during experimentation.

6. Missing data and imputation method

Scalar_B	0.000571
BX_GSE_GSM	0.000571
BY_GSE	0.000571
BZ_GSE	0.000571
BY_GSM	0.000571
BZ_GSM	0.000571
Proton_Density	0.380022
SW_Plasma_Temperature	0.128635
SW_Plasma_Speed	0.010995
Dst-index	0.005337

Figure 19 Percentage of missing data within each data

Some of the data were missing and were filled in with an imputation technique called “fill forward”. However, a better technique can be used to generate more representative data.

7. Lack of High correlation features

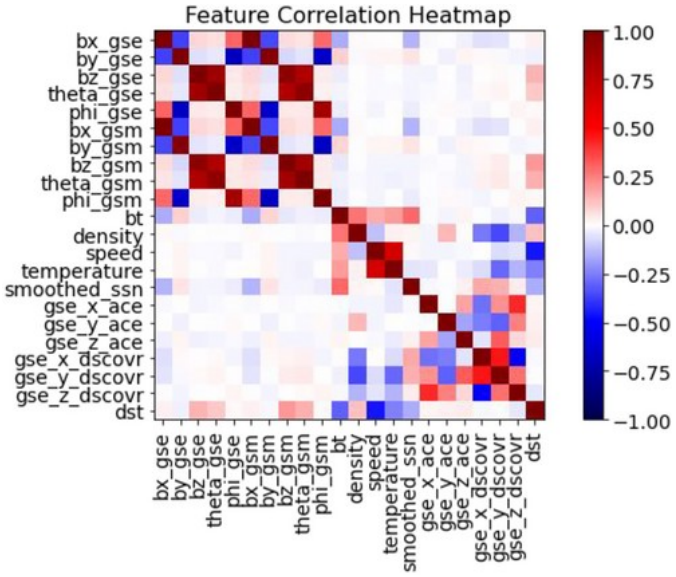

Figure 20 Correlation graph of DST index from MagNet—A Data-Science Competition to Predict Disturbance Storm-Time Index (Dst) From Solar Wind Data

From MagNet—A Data-Science Competition to Predict Disturbance Storm-Time Index (Dst) From Solar Wind Data, There is an example of Feature Correlation (Figure 20), indicating that some features with high correlations to the DST index are missing in this project such as *bt - Interplanetary magnetic field magnitude* and *smoothed_ssn - Monthly sunspot numbers, smoothed.* These features are likely important factors for DST prediction, including these features in training the model could improve the performance.

Conclusion

In this study, a CNN-LSTM model was developed to forecast the Disturbance Storm Index (DST), achieving an MSE of 60.01. Despite this result, the model exhibited underfitting, with training loss consistently higher than test loss, and struggled to improve beyond the early stopping threshold. Increasing model complexity by experimenting with DSTNET architecture, further confirming underfitting issues. The learning curve for all models indicated that neither architecture fully captured the complexity of the DST index data, highlighting the challenges of balancing model capacity and generalization.

Future work should focus on hyperparameter tuning, advanced architectures, and feature engineering to address these limitations. This study underscores the potential of deep learning for DST forecasting while emphasizing the need for further refinement to achieve more accurate and reliable predictions.

Reference

[1] https://wdc.kugi.kyoto-u.ac.jp/wdc/Sec3.html

[2] https://izw1.caltech.edu/ACE/ASC/level2/lvl2DATA_MAG.html

[3] https://izw1.caltech.edu/ACE/ASC/level2/lvl2DATA_SWEPAM.html

[4] https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2023SW003514

[5] https://github.com/deepsuncode/Dst-prediction