Developing RNN and LSTM Models for Hong Kong Daily Minimum Temperature Forecasting

Course: DIT5411 Machine Learning

Name: Fok Tsz Ching

Abstract

This study investigates the use of deep learning models—Simple Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), and Bidirectional LSTM (BiLSTM)—for forecasting Hong Kong's daily grass minimum temperature. Using historical meteorological data from 1980 to 2024, models were trained to predict daily temperatures for the test period spanning January to October 2025. The LSTM model achieved the best performance, with a Mean Absolute Error (MAE) of 0.92°C and a Root Mean Squared Error (RMSE) of 1.26°C. The results indicate that LSTM networks capture temporal dependencies more effectively than Simple RNNs, especially in climate datasets featuring long-term seasonal patterns. This report also discusses model limitations, error patterns, and potential directions for future development.

1. Introduction

Daily grass minimum temperature is a critical environmental indicator monitored by the Hong Kong Observatory. As temperatures near the ground cool faster than the air, grass minimum temperature serves as an essential metric for agricultural planning, frost warnings, and urban climate research. Accurate forecasting of such temperature trends supports timely decision-making across multiple fields.

Traditional statistical models often struggle with long-term temporal dependencies. Deep learning, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, offers modern approaches capable of modelling sequential dependencies effectively (Hochreiter & Schmidhuber, 1997). This project aims to compare the performance of RNN, LSTM, and BiLSTM architectures in forecasting Hong Kong's daily grass minimum temperature using a univariate time-series dataset covering 45 years.

2. Literature Review

Recurrent neural networks are commonly applied in time-series forecasting due to their recurrent structure, which allows them to process sequential data (Lipton, 2015). However, simple RNNs suffer from vanishing gradients, limiting their ability to learn long-term patterns (Bengio et al., 1994).

LSTM networks address these limitations through memory cells and gating mechanisms, allowing them to retain information over long sequences. They have demonstrated strong performance in climate prediction, weather modelling, and environmental forecasting tasks (Shi et al., 2015).

BiLSTM extends the LSTM architecture by integrating forward and backward temporal representations. Although powerful in tasks where future context is relevant (e.g., NLP), its effectiveness in unidirectional forecasting is still debated.

This project builds upon such findings and evaluates the performance of these architectures on real-world climatic data from Hong Kong.

3. Dataset and Preprocessing

3.1 Dataset Description

The dataset originates from the Hong Kong Observatory's publicly available Grass Minimum Temperature records. Each observation includes:

- Date (YYYY-MM-DD)
- Grass Minimum Temperature (°C)

The full dataset spans from 1980 to 2025. Following project requirements, the training set contains data from **1980–2024**, and the testing set covers **2025-01-01 to 2025-10-30**.

3.2 Preprocessing Steps

The following steps were implemented in data_preprocessing.py:

1. Loading and cleaning raw CSV records.
2. Converting year, month, and day columns into a unified datetime index.
3. Removing missing or invalid entries.
4. Interpolating missing temperature values using time-based interpolation.
5. Filtering data to match the required time range.

A MinMaxScaler (range 0–1) was fitted on the training set only to avoid data leakage. Sliding windows of **45 days** were used to generate input sequences for model training.

4. Methodology

4.1 Sequence Generation

The sequence_generator.py script converts temperature values into overlapping sliding windows. For a window size of 45 days:

- Input shape: (45, 1)
- Output: Temperature of the next day

This sequence-to-one structure allows each model to learn patterns from the preceding 45 days.

4.2 Model Architectures

Simple RNN (Baseline)

- Two SimpleRNN layers (100 units each)
- Activation: tanh
- Optimizer: Adam
- Loss: MSE

LSTM (Advanced Model)

- Two LSTM layers with dropout
- Memory cells capture long-term dependencies
- Improved gradient flow over long sequences

Bidirectional LSTM (Extension)

- Two BiLSTM layers
- Capable of leveraging both forward and backward dependencies during training
- May not provide substantial benefit in strictly forward-predictive tasks

4.3 Training Configuration

All models were trained with:

- Epochs: 80
- Batch size: 32
- Validation split: 0.1
- Loss function: MSE
- Metrics: MAE

5. Results

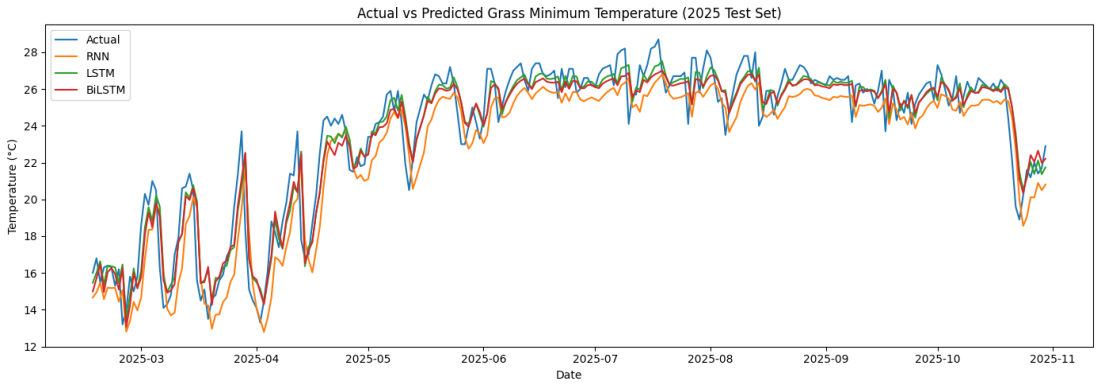
5.1 Performance Metrics

Model	MAE (°C)	RMSE (°C)
RNN	1.4411	1.7119
LSTM	**0.9216**	**1.2611**
BiLSTM	0.9520	1.2650

The LSTM achieved the best overall performance, with the lowest MAE and RMSE values.

5.2 Actual vs Predicted Temperature Plot

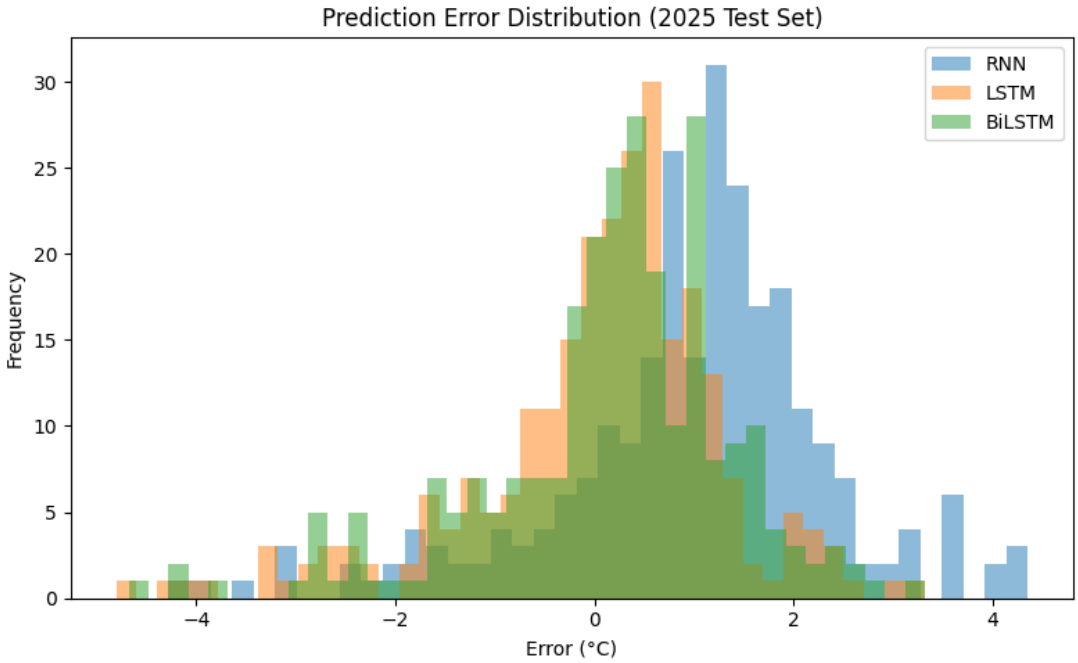
Figure 1



This plot compares actual 2025 temperature values with predictions from all three models.

5.3 Error Distribution Plot

Figure 2



LSTM and BiLSTM errors center more tightly around zero, whereas the RNN shows a wider and less stable distribution.

6. Discussion

6.1 Why LSTM Outperforms RNN

LSTM networks possess memory gates enabling them to capture long-term seasonal trends, such as Hong Kong's winter-summer cycles. In contrast, simple RNNs struggle with long sequences due to vanishing gradients, leading to poorer accuracy and less stable predictions.

6.2 BiLSTM Performance

Although BiLSTM generally excels in tasks requiring bidirectional context, temperature forecasting depends strictly on **past** data. Therefore, the forward-backward structure provides minimal additional information, resulting in performance nearly identical to standard LSTM.

6.3 Error Patterns

Error spikes were observed during seasonal transition periods (March–April), when temperatures fluctuate rapidly. All models tend to underpredict sharp temperature changes, a known limitation in univariate climate models.

6.4 Limitations

- Only grass minimum temperature was used; adding humidity, wind speed, and cloud cover could improve forecasts.
- Model performance may further benefit from hyperparameter tuning or hybrid architectures (e.g., CNN-LSTM).
- Data prior to 1980 was excluded for consistency; including more historical data may enhance long-term trend learning.

7. Conclusion

This project successfully implemented and compared RNN, LSTM, and BiLSTM models for forecasting Hong Kong's daily grass minimum temperature. The LSTM model demonstrated superior performance, confirming the importance of memory-based architectures in long-term environmental forecasting. Future work may explore multivariate inputs, attention mechanisms, or transformer-based models to further enhance predictive accuracy.

References (APA Format)

Bengio, Y., Simard, P., & Frasconi, P. (1994). *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 5(2), 157–166.

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural Computation, 9(8), 1735–1780.

Lipton, Z. C. (2015). *A critical review of recurrent neural networks for sequence learning*. arXiv:1506.00019.

Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). *Convolutional LSTM network: A machine learning approach for precipitation nowcasting*. Advances in Neural Information Processing Systems.

Hong Kong Observatory. (2025). *Daily Grass Minimum Temperature Dataset*. https://data.gov.hk/

Appendix A — sequence_generator.py

```
import numpy as np

import pandas as pd
```

```
from sklearn.preprocessing import MinMaxScaler


TRAIN_END = "2024-12-31"
TEST_START = "2025-01-01"
TEST_END = "2025-10-30"


def create_sequences(values, window):
    X, y = [], []
    for i in range(len(values) - window):
        X.append(values[i:i+window])
        y.append(values[i+window])
    return np.array(X), np.array(y)


def get_train_test_data(window, path="processed_HKO_GMT_ALL.csv"):
    df = pd.read_csv(path, parse_dates=["date"]).sort_values("date")
    train = df[df["date"] <= TRAIN_END]["Value"].values.reshape(-1,1)
    test = df[(df["date"] >= TEST_START) & (df["date"] <=
TEST_END)]["Value"].values.reshape(-1,1)


    scaler = MinMaxScaler()
    train_scaled = scaler.fit_transform(train)
    test_scaled = scaler.transform(test)


    X_train, y_train = create_sequences(train_scaled, window)
    X_test, y_test = create_sequences(test_scaled, window)
    return X_train, y_train, X_test, y_test, scaler
```

Appendix B — models_rnn_lstm.py

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


def create_rnn_model(window):
    model = Sequential([
        Input(shape=(window,1)),
        SimpleRNN(100, return_sequences=True),
        SimpleRNN(100),
        Dense(1)
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                  loss="mse", metrics=["mae"])
    return model


def create_lstm_model(window):
    model = Sequential([
        Input(shape=(window,1)),
        LSTM(100, return_sequences=True),
        Dropout(0.2),
        LSTM(100),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
```

```
                    loss="mse", metrics=["mae"])
    return model


def create_bilstm_model(window):
    model = Sequential([
        Input(shape=(window,1)),
        Bidirectional(LSTM(100, return_sequences=True)),
        Dropout(0.2),
        Bidirectional(LSTM(100)),
        Dropout(0.2),
        Dense(1)
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                    loss="mse", metrics=["mae"])
    return model
```

Appendix C — train_and_evaluate.py

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error

from sequence_generator import get_train_test_data
from models_rnn_lstm import create_rnn_model, create_lstm_model,
create_bilstm_model

def main():
    window = 45
```

```
    X_train, y_train, X_test, y_test, scaler = get_train_test_data(window)
    y_true = scaler.inverse_transform(y_test)

    rnn = create_rnn_model(window)
    rnn.fit(X_train, y_train, epochs=80, batch_size=32, verbose=0)
    pred_rnn = scaler.inverse_transform(rnn.predict(X_test))

    lstm = create_lstm_model(window)
    lstm.fit(X_train, y_train, epochs=80, batch_size=32, verbose=0)
    pred_lstm = scaler.inverse_transform(lstm.predict(X_test))

    bilstm = create_bilstm_model(window)
    bilstm.fit(X_train, y_train, epochs=80, batch_size=32, verbose=0)
    pred_bilstm = scaler.inverse_transform(bilstm.predict(X_test))

    print("RNN MAE:", mean_absolute_error(y_true, pred_rnn))
    print("LSTM MAE:", mean_absolute_error(y_true, pred_lstm))
    print("BiLSTM MAE:", mean_absolute_error(y_true, pred_bilstm))

if __name__ == "__main__":
    main()
```