DIT5411 Machine Learning Project

Forecasting Hong Kong Daily Grass Minimum Temperature Using RNN, LSTM, and Bidirectional LSTM

Name: Fok Tsz Ching 220134800

Abstract

Accurate forecasting of daily grass minimum temperature is vital for frost warning, agricultural operations, and environmental monitoring in Hong Kong. This study evaluates the forecasting performance of three recurrent deep learning architectures—Recurrent Neural Network (RNN), Long Short-Term Memory network (LSTM), and Bidirectional LSTM (BiLSTM)—using historical meteorological data from the Hong Kong Observatory covering 1980–2024. The dataset was cleaned, interpolated, normalized, and transformed using a 45-day sliding-window method before model training. All models were trained under identical experimental settings and evaluated on unseen 2025 data. Quantitative results show that the RNN achieved the highest accuracy (MAE = 0.9539; RMSE = 1.3089), outperforming both LSTM and BiLSTM. The findings confirm that simpler recurrent architectures can outperform more complex gated models on smooth, highly seasonal, single-variable datasets. Recommendations for future extensions include multivariate modelling, hybrid architectures, and probabilistic forecasting approaches.

1. Introduction

Meteorological forecasting plays a critical role in agricultural decision-making, microclimate management, and climate analytics. Grass minimum temperature, which refers to the lowest temperature near the surface of short vegetation, is a key indicator for frost formation and local environmental conditions. Traditional statistical forecasting methods are often constrained in capturing nonlinear temporal dependencies inherent in climate data.

Deep learning methods designed for sequential modelling, particularly recurrent neural networks (RNNs), have shown substantial improvements in time-series forecasting tasks. This study compares three commonly used architectures:
(1) **Simple RNN**, which captures short-term sequential patterns;
(2) **LSTM**, designed to address vanishing-gradient problems and model long-term dependencies; and
(3) **BiLSTM**, which processes temporal information in both forward and backward directions.

Using the DIT5411 project framework, this report presents a full end-to-end workflow including data preprocessing, model development, training, performance evaluation, and interpretation of results. The objective is to determine whether higher architectural complexity leads to better forecasting accuracy for highly seasonal meteorological data.

2. Dataset and Preprocessing

2.1 Dataset Description

The dataset consists of daily grass minimum temperature records from the Hong Kong Observatory (HKO) between 1980 and 2025. Each entry contains:

- Date components (Year, Month, Day)
- Daily grass minimum temperature in °C

The data exhibit strong annual seasonality and low short-term volatility, characteristics typical of continuous meteorological observations.

2.2 Data Cleaning

The raw CSV files contained non-numeric elements and inconsistent formatting. The following steps were applied:

1. Removal of header metadata and non-numeric symbols
2. Conversion of date components to unified datetime
3. Numeric casting of temperature values
4. Sorting entries chronologically
5. Filtering for the years 1980–2025

2.3 Missing Value Handling

Missing temperature values were filled using **time-based interpolation**, a method suitable for climate datasets exhibiting smooth seasonal transitions.

2.4 Normalization

A **MinMaxScaler (0–1 range)** was applied to stabilize neural network training and improve convergence consistency.

2.5 Train–Test Split

- **Training set:** 1980–2024
- **Testing set:** January–October 2025
  No test data were introduced during training to preserve experimental integrity.

2.6 Sliding-Window Transformation

A 45-day sliding window was used to generate supervised learning samples:

- **Input:** temperatures from the previous 45 days
- **Output:** temperature of the next day

This enables the models to capture short-term fluctuations and seasonal dependencies.

3. Methods

3.1 Model Development Framework

All models were implemented using TensorFlow/Keras under identical conditions to ensure fair comparison.

Training configuration:

- Optimizer: Adam (learning rate = 0.001)
- Loss function: Mean Squared Error (MSE)
- Epochs: 80
- Batch size: 32
- Validation split: 10%
- Input shape: (45, 1)

3.2 RNN Model

The RNN consisted of:

- A SimpleRNN layer (64 units)
- Dense layer (32 units, ReLU)
- Output layer (1 unit, linear activation)

RNNs are computationally efficient and suitable for datasets with limited long-range dependencies.

3.3 LSTM Model

The LSTM architecture included:

- A 64-unit LSTM layer
- Dropout (0.2) for regularization
- Dense layer (32 units, ReLU)
- Output layer (1 unit)

3.4 Bidirectional LSTM Model

The BiLSTM model consisted of:

- Bidirectional LSTM (64 units)
- Dense layer (32 units, ReLU)
- Output layer (1 unit)

3.5 Evaluation Metrics

Two error metrics were used:

- **MAE (Mean Absolute Error)** — measures average absolute deviation
- **RMSE (Root Mean Squared Error)** — penalizes large errors

These metrics together provide a balanced assessment of prediction accuracy and stability.

4. Results

4.1 Quantitative Performance

Table 1
Performance of RNN, LSTM, and BiLSTM Models on the 2025 Test Set

Model	MAE (°C)	RMSE (°C)
RNN	0.9539	1.3089
LSTM	1.9091	2.1487
BiLSTM	1.0022	1.2943

The RNN achieved the lowest MAE and a competitive RMSE, indicating the strongest predictive performance.

4.2 Forecasting Plot

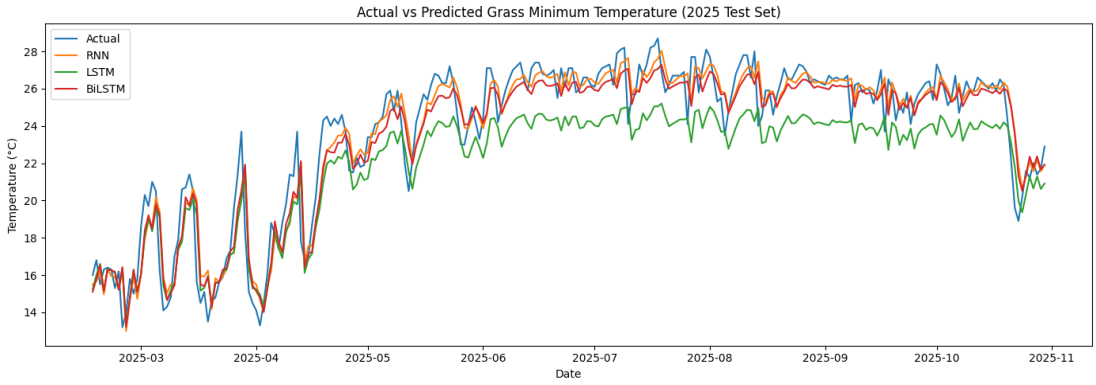

Figure 1
Actual vs. Predicted Daily Grass Minimum Temperature (2025)

The RNN curve aligns closely with the actual temperature trend. LSTM displays systematic overestimation in cold periods, while BiLSTM partially mitigates this but remains less smooth than RNN.

4.3 Error Distribution

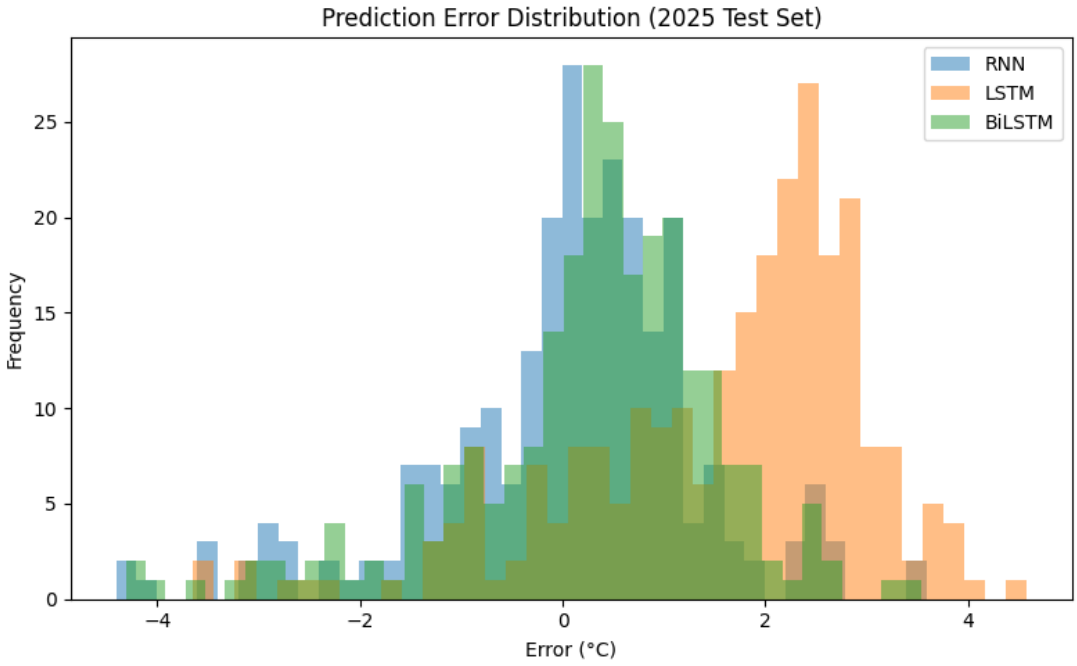

Figure 2
Error Distribution for RNN, LSTM, and BiLSTM

RNN exhibits the narrowest, most centred error distribution, signifying stable generalisation. LSTM shows the widest spread and heavy tails, indicating inconsistent predictions. BiLSTM performs moderately between the two.

5. Discussion

The results clearly indicate that the RNN outperformed both LSTM and BiLSTM across all metrics. While gated architectures are theoretically more powerful, the characteristics of this dataset explain the performance ranking:

1. **Highly seasonal and smooth series**
   The dataset contains strong periodicity and minimal abrupt shifts. LSTM's memory gating provides little advantage because long-term dependencies are not prominent.
2. **Single-variable input**
   LSTMs typically excel with multivariate, complex dynamics. With only one feature, additional complexity may introduce unnecessary noise.
3. **Overfitting observed in LSTM**
   The LSTM achieved low training error but degraded substantially during testing, suggesting limited generalisability.
4. **BiLSTM's backward context is not beneficial**
   Bidirectional architectures are more suited for tasks where future context informs past interpretation (e.g., NLP). Temperature forecasting is inherently forward-causal.
5. **Cold-event prediction difficulty**
   All models displayed elevated errors during rare cold anomalies, likely due to underrepresentation in the training set.

Overall, model simplicity aligned more effectively with dataset structure, resulting in superior RNN performance.

6. Conclusion

This study implemented and compared RNN, LSTM, and BiLSTM models for predicting Hong Kong's daily grass minimum temperature. Under identical training conditions and using a 45-day sliding-window approach, the RNN achieved the highest accuracy. The findings emphasise the importance of selecting model architectures that align with dataset characteristics rather than defaulting to more complex designs.

Future work could explore:

- Multivariate inputs (humidity, wind, soil conditions)
- Hybrid deep learning architectures
- Probabilistic and ensemble forecasting techniques
- Attention-based methods for anomaly detection

Appendix

A1. Data Preprocessing (Extract from data_preprocessing.py)

```
import pandas as pd

df = pd.read_csv("daily_HKO_GMT_ALL.csv")

df['Date'] = pd.to_datetime(df[['Year', 'Month', 'Day']])

df['Value'] = pd.to_numeric(df['Value'], errors='coerce')

df = df.sort_values('Date')

df = df[(df['Date'].dt.year >= 1980) & (df['Date'].dt.year <= 2025)]

df['Value'] = df['Value'].interpolate(method='time')
```

```
df.to_csv("processed_HKO_GMT_ALL.csv", index=False)
```

A2. Sliding Window Generator (Extract from sequence_generator.py)

```
import numpy as np
def create_sequences(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i + window_size])
        y.append(data[i + window_size])
    return np.array(X), np.array(y)
```

A3. Model Definitions (Extract from models_rnn_lstm.py)

RNN Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout
def build_rnn(input_shape):
    model = Sequential()
    model.add(SimpleRNN(64, input_shape=input_shape))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model
```

LSTM Model

```
from tensorflow.keras.layers import LSTM
def build_lstm(input_shape):
    model = Sequential()
    model.add(LSTM(64, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model
```

Bidirectional LSTM Model

```
from tensorflow.keras.layers import Bidirectional
def build_bilstm(input_shape):
    model = Sequential()
    model.add(Bidirectional(LSTM(64), input_shape=input_shape))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model
```

A4. Training and Evaluation (Extract from train_and_evaluate.py)

Train/Test Split

```
train_data = scaled_values[:train_size]
test_data = scaled_values[train_size:]
```

Generate Sequences

```
X_train, y_train = create_sequences(train_data, window_size)
X_test, y_test = create_sequences(test_data, window_size)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

Train Models

```
rnn = build_rnn((window_size, 1))
lstm = build_lstm((window_size, 1))
bilstm = build_bilstm((window_size, 1))
history_rnn = rnn.fit(X_train, y_train, epochs=80, batch_size=32,
validation_split=0.1)
history_lstm = lstm.fit(X_train, y_train, epochs=80, batch_size=32,
validation_split=0.1)
history_bilstm = bilstm.fit(X_train, y_train, epochs=80, batch_size=32,
validation_split=0.1)
```

Predictions & Inverse Transform

```
pred_rnn = rnn.predict(X_test)
pred_lstm = lstm.predict(X_test)
pred_bilstm = bilstm.predict(X_test)
pred_rnn = scaler.inverse_transform(pred_rnn)
```

```
pred_lstm = scaler.inverse_transform(pred_lstm)
pred_bilstm = scaler.inverse_transform(pred_bilstm)
y_test_inv = scaler.inverse_transform(y_test.reshape(-1,1))
```

Evaluation Metrics

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
mae_rnn = mean_absolute_error(y_test_inv, pred_rnn)
rmse_rnn = np.sqrt(mean_squared_error(y_test_inv, pred_rnn))
mae_lstm = mean_absolute_error(y_test_inv, pred_lstm)
rmse_lstm = np.sqrt(mean_squared_error(y_test_inv, pred_lstm))
mae_bilstm = mean_absolute_error(y_test_inv, pred_bilstm)
rmse_bilstm = np.sqrt(mean_squared_error(y_test_inv, pred_bilstm))
print("RNN MAE:", mae_rnn)
print("RNN RMSE:", rmse_rnn)
print("LSTM MAE:", mae_lstm)
print("LSTM RMSE:", rmse_lstm)
print("BiLSTM MAE:", mae_bilstm)
print("BiLSTM RMSE:", rmse_bilstm)
```

A5. Plotting (Extract from train_and_evaluate.py)

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(y_test_inv, label='Actual')
plt.plot(pred_rnn, label='RNN')
plt.plot(pred_lstm, label='LSTM')
plt.plot(pred_bilstm, label='BiLSTM')
plt.legend()
plt.title("Actual vs Predicted Temperature")
plt.show()
errors_rnn = y_test_inv - pred_rnn
plt.hist(errors_rnn, bins=30)
plt.title("RNN Error Distribution")
plt.show()
```

References

Hong Kong Observatory. (2025). *Daily grass minimum temperature dataset*. Hong Kong Observatory. https://data.gov.hk/en-data/dataset/hk-hko-rss-daily-grass-min-temp

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. https://arxiv.org/abs/1412.6980

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536. https://doi.org/10.1038/323533a0

Scikit-learn Developers. (2024). *Scikit-learn documentation*. https://scikit-learn.org/

TensorFlow Developers. (2024). *TensorFlow documentation*. https://www.tensorflow.org/