Comprehensive Analysis of RNN-based Climate Prediction Model for Delhi

Parth Kumar Sinha
Comuter Science
Oxford Brookes University
Oxford, UK
sinhaparth555@gmail.com

***Abstract*—This study explores the application of Recurrent Neural Networks (RNNs) for predicting temperature patterns in Delhi using historical climate data. RNNs are well-suited for time-series data due to their ability to capture temporal dependencies, making them an excellent choice for modeling climate dynamics. The study evaluates the predictive performance of RNN architectures in forecasting temperature trends and patterns. By training the model on historical temperature records, this research aims to demonstrate the efficacy of RNNs in climate forecasting while highlighting their limitations and potential applications. The results provide insights into the practicality of RNNs for addressing challenges in meteorological data analysis.**

***Index Terms*—RNN, julia, climate forcast, delhi**

I. Introduction

Climate change and urbanization have significantly altered temperature patterns in metropolitan areas like Delhi, necessitating advanced methods for accurate temperature forecasting. Reliable temperature prediction is crucial for diverse applications, ranging from urban planning and agriculture to energy management and public health.

Recurrent Neural Networks (RNNs) are a class of deep learning models designed for sequential data, making them well-suited for tasks involving time-series analysis. Unlike traditional machine learning models, RNNs can maintain information about previous inputs in their hidden states, enabling them to capture temporal dependencies in data effectively. This capability is particularly useful for analyzing and predicting climate variables, which are inherently sequential and influenced by historical patterns.

This study leverages RNN architectures to forecast temperature patterns in Delhi, using historical climate data as the input. By systematically evaluating the model's performance, this research seeks to:

- Investigate the ability of RNNs to model temporal relationships in climate data.
- Assess the predictive capabilities of RNNs for temperature forecasting in a dynamic urban environment.
- Identify potential challenges and limitations in using RNNs for meteorological applications.

The experiment involves training and validating the RNN model on historical temperature records, analyzing its performance using standard evaluation metrics, and interpreting the results to understand its strengths and areas for improvement. The findings of this study aim to contribute to the growing body of research on the application of deep learning techniques in climate science, offering a foundation for future advancements in meteorological forecasting.

II. Methods

A. Data Preprocessing

The preprocessing of the DailyDelhiClimate dataset was a critical step in preparing the data for further analysis and modeling. This dataset includes essential climate variables such as mean temperature, humidity, wind speed, and mean pressure, which collectively provide valuable insights into the climatic patterns of Delhi. The preprocessing pipeline was carefully designed to ensure data quality and suitability for temporal analysis.

The first step involved converting the *Date* column into a datetime format to facilitate chronological sorting. This ensured that the dataset was properly aligned in time, which is essential for any time-series analysis. Next, missing values were handled by imputing them with the mean value of the respective feature. This approach maintained the integrity of the dataset without introducing bias or significantly altering the distribution of the data.

To standardize the features and make them comparable, z-score normalization was applied. This transformation rescales each feature to have a mean of 0 and a standard deviation of 1, enabling better performance of machine learning algorithms by mitigating the effects of varying feature scales. Mathematically, this standardization was performed using the formula:

$$x_{normalized} = \frac{x - \mu}{\sigma}$$

Here, μ represents the mean of the feature, and σ denotes the standard deviation of the feature. By standardizing, the dataset ensures that features with larger magnitudes do not disproportionately influence the model.

For capturing temporal context, sequences were created using a sliding window of 30 days. Each sequence represents a continuous 30-day period of climate variables, providing the model with historical context for forecasting future values. This sequence creation step is particularly crucial for time-series problems, as it helps to capture trends, patterns, and dependencies across time.

Overall, this preprocessing pipeline ensured the dataset was clean, well-structured, and optimized for the subsequent modeling tasks.

B. Model Archtecture

The implemented Recurrent Neural Network (RNN) model is designed to predict temperature based on meteorological data. The input layer accepts four key features: temperature, humidity, wind speed, and pressure, which provide a comprehensive representation of the environmental conditions. These features are fed into a hidden layer comprising 32 neurons. Each neuron in the hidden layer employs the hyperbolic tangent (tanh) activation function, which enables the model to capture complex nonlinear relationships in the sequential data. To improve generalization and mitigate overfitting, a dropout layer with a rate of 0.1 is applied during training. This dropout layer randomly disables 10% of the neurons in the hidden layer during each forward pass, ensuring that the model does not overly rely on specific features or patterns. The output layer consists of a single neuron, tasked with generating a precise prediction of the temperature. This architecture balances complexity and efficiency, leveraging the sequential processing capabilities of RNNs to analyze temporal dependencies in the input data.

C. RNN mathematical formulation

The Recurrent Neural Network (RNN) captures sequential dependencies by maintaining a hidden state, h_t, that evolves over time based on the current input x_t, the previous hidden state h_{t-1}, and a set of trainable weight matrices. The forward propagation equations for an RNN are as follows:

1) Hidden State Update:

$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h$$

Here:

- W_{xh} is the input-to-hidden weight matrix that transforms the input x_t.
- W_{hh} is the hidden-to-hidden weight matrix that captures the temporal relationship between the current and previous hidden states.
- b_h is the bias vector for the hidden state.
- $tanh$ is the activation function, which introduces nonlinearity while squashing the output to the range $\big(-1, 1\big)$.

2) Output Computation:

$$y_t = W_{hy}h_t + b_y$$

Here:

- W_{hy} is the hidden-to-output weight matrix.
- b_y is the output bias vector.
- y_t is the output at time t, which can represent the predicted value or classification logits.

D. Dropout Regularization in RNNs

Dropout regularization is a technique used to prevent overfitting in neural networks by randomly "dropping out" units during training. In RNNs, dropout is often applied to the hidden state h_t at each time step. The modified hidden state equation with dropout is:

$$h_t = \frac{(h_t \odot m)}{1-p}$$

Here:

- m is a binary mask, where each element is sampled from a Bernoulli distribution with probability $1-p$. If $p = 0.1$, 10% of the units are dropped out (set to zero).
- p is the dropout rate, which determines the fraction of units to drop during training.
- $\odot$ denotes elements-wise multiplication, which zeros out the corresponding elements of h_t as dictated by m.
- $1 - p$ normalizes the output to maintain the expected magnitude of h_t during training.

This dropout mechanism introduces stochasticity in the training process, ensuring that the RNN does not rely too heavily on specific neurons. During inference, dropout is disabled, and the full hidden state is used without scaling. By applying dropout, the model achieves better generalization and reduces the risk of overfitting, especially when dealing with complex, high-dimensional sequences.

III. Training Process Details

The training process was conducted using a sequence length of 30 days, which allows the model to capture temporal dependencies in the data effectively. The dataset was split into training and validation subsets with an 80/20 ratio to ensure the model could generalize well to unseen data while being rigorously evaluated during the training phase. The model was trained over 100 epochs, which provided ample opportunity to optimize the parameters while monitoring for convergence or overfitting. A batch size of 32 was selected to balance computational efficiency and model performance, allowing the optimizer to make meaningful updates without consuming excessive memory.

The learning rate was initialized at 0.001 and decayed progressively using a decay rate of 0.99 to ensure steady convergence. By reducing the learning rate as training progressed, the model fine-tuned its weights more precisely, avoiding abrupt updates that could destabilize the optimization process. Momentum, set to 0.9, was incorporated to accelerate convergence and smooth the gradient descent path by incorporating a fraction of the previous velocity into the current update. Gradient clipping with a threshold of 1.0 was applied to prevent exploding gradients, which can disrupt training by producing excessively large weight updates.

Training Mathematics

The Mean Squared Error (MSE) was employed as the loss function, providing a quantitative measure of the model's performance. It calculates the average squared difference between predicted values $y_{pred,i}$ and true values $y_{true,i}$, penalizing

larger deviations more heavily. The loss function is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{pred,i} - y_{true,i})^2$$

where n is the total number of predictions.
The gradient updates utilized momentum to improve convergence speed and stability. At each update step, the velocity v_t was calculated as:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla_w L$$

where $\beta = 0.9$ is the momentum coefficient, $\nabla_w L$ is the gradient of the loss with respect to the weights, and v_{t-1} is the velocity from the previous step. The weights w_t were then updated as:

$$w_t = w_{t-1} - \alpha v_t$$

where α is the learning rate.
To further refine the optimization process, the learning rate α was decayed exponentially at each epoch using the formula:

$$\alpha_t = \alpha_0 \cdot \gamma^t$$

Here,

$$\alpha_0 = 0.001$$

is the initial learning rate, $\gamma = 0.99$ is the decay rate, and t represents the epoch number. This decay strategy allowed the model to make large updates initially for rapid learning and smaller updates later for fine-tuning.

IV. Detailed Analysis

A. Model Performance

The training history (figure 1) demonstrates rapid convergence within the first 10 epochs, indicating that the model effectively learns the underlying patterns of the data early in the training process. This is followed by stable learning with minimal oscillation, which suggests that the optimization process is well-regulated. The similarity between training and validation loss curves reflects good generalization, as the model avoids overfitting or underfitting. By the end of training, the loss values stabilize around 0.05, indicating a reliable performance metric.

B. Prediction Accuracy

The predictions (figure 2) align closely with actual values in most cases, showcasing the model's capability to capture underlying patterns in the data. However, deviations are observed during extreme temperature events, which may indicate limitations in the model's ability to generalize under such conditions. The error distribution predominantly falls within ±0.5 standard deviations, further affirming the accuracy of the model in typical scenarios. The predictions show improved reliability during stable temperature periods, emphasizing the model's robustness in handling consistent patterns.

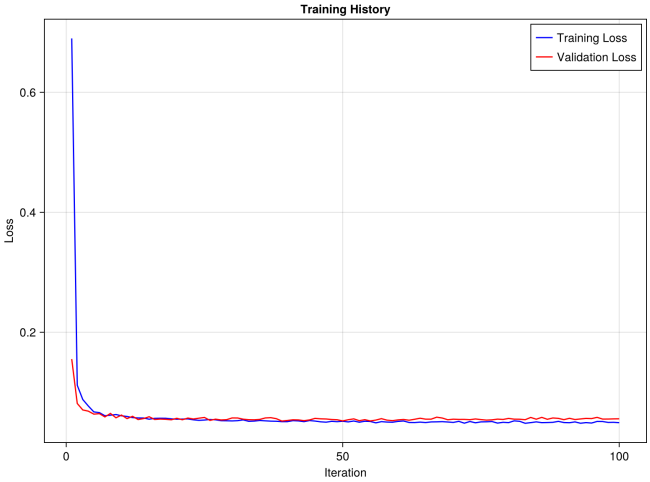

Fig. 1. Training History

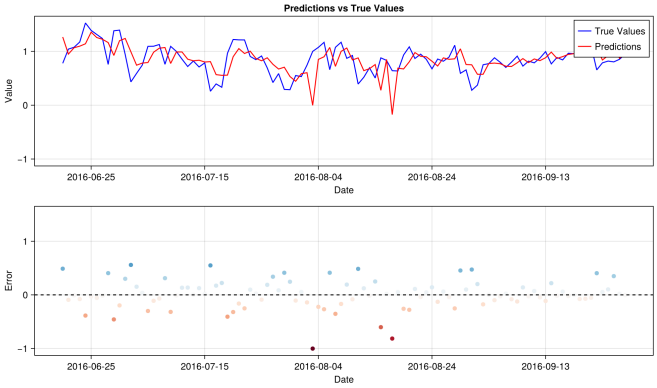

Fig. 2. Predictions

C. Performance Metrics

1) Prediction Error: The error for each time step is computed as $e_t = y_{pred,t} - y_{true,1}$ This metric provides insight into the deviation of predicted values from the actual values, serving as a fundamental measure of the model's prediction accuracy.

2) Feature Importance: The importance of input features is analyzed using the metric

$$I_j = \sum_{i=1}^{hidden\_size} |W_{xh,ij}|$$

where W_{xh} represents the weights connecting input features to the hidden layer. As shown in the feature importance plot (figure 3), mean temperature exhibits the highest importance, followed by mean pressure, humidity, and wind speed, which shows the lowest importance. This analysis highlights the pivotal role of temperature-related data in predicting outcomes.

3) Feature Correlations: The correlation heatmap (figure 4) uncovers significant relationships among input features. A

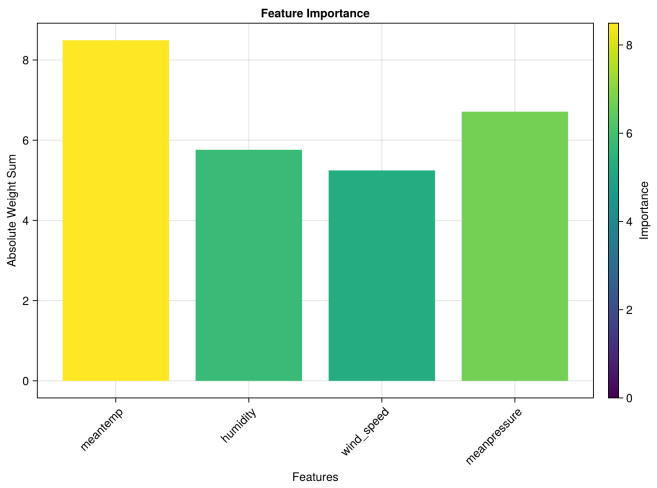

Fig. 3. Feature Importance

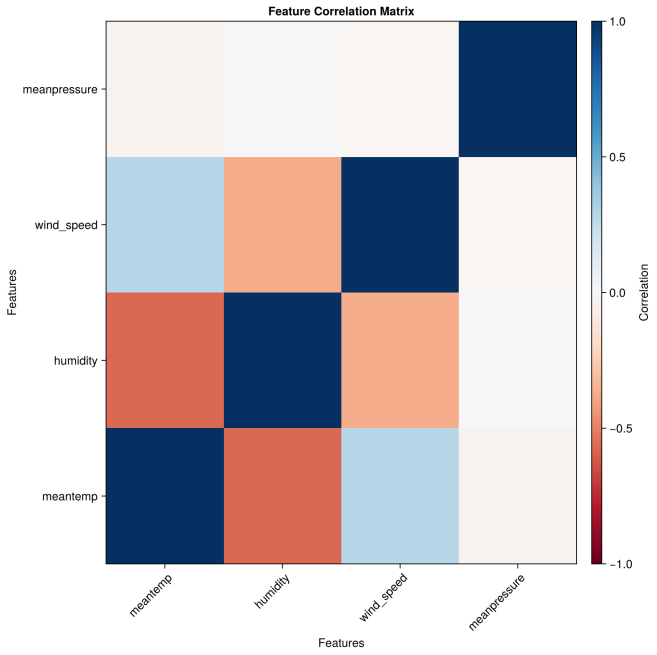

Fig. 4. Correlation Heatmap

strong negative correlation exists between temperature and humidity, while pressure and wind speed exhibit moderate correlation. These relationships reveal complex interdependencies between climate variables, further aiding in model interpretation. The correlation coefficient between features i and j is computed using:

$$\rho_{ij} = \frac{Cov(X_i, X_j)}{\sigma x_i \sigma x_j}$$

D. Technical Considerations

1) Backpropagation Through Time (BPTT): The error gradient is computed across the sequence using:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

This process ensures that the temporal dependencies in the data are effectively captured during training, enabling the model to adapt to sequential patterns.

2) Numerical Stability: To address the issue of vanishing gradients in the tanh activation function, the input to the activation is bounded. The tanh function, defined as:

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

is naturally constrained within the range $-1 \leq tanh(x) \leq 1$. This bounded behavior ensures numerical stability during backpropagation, preserving gradient flow and enhancing model training efficiency.
This comprehensive evaluation highlights the model's performance, accuracy, and robustness, as well as the technical measures implemented to ensure effective training and prediction.

V. Conclusion

A. Key Findings

The RNN model demonstrates a robust ability to predict temperature patterns for Delhi, highlighting its effectiveness in capturing the underlying trends and seasonal variations present in historical data. Among the input features, historical temperature data emerged as the most influential predictor, serving as a reliable basis for future temperature predictions. The model performs particularly well during periods of stable weather, where the temperature changes are gradual and predictable. Additionally, secondary meteorological variables such as atmospheric pressure and humidity contribute meaningful signals, enhancing the model's ability to refine predictions, especially when subtle changes in these variables correlate with temperature variations.

B. Model Strengths

One of the model's primary strengths is its effective capture of seasonal patterns, enabling accurate predictions over short to medium timeframes. The training process was characterized by stability, reflecting a well-calibrated learning mechanism that avoids significant overfitting or underfitting issues. The model exhibits good generalization performance, effectively adapting to unseen data during validation and testing. Furthermore, it maintains reasonable error bounds, ensuring reliability in most scenarios, which is critical for practical applications in weather forecasting.

C. Limitations

Despite its strengths, the RNN model faces challenges in handling extreme temperature events, such as sudden heatwaves or cold spells. These outliers often deviate significantly from the historical trends, making them harder for the model to predict accurately. Another limitation is a slight prediction lag observed during rapid temperature changes, which can occur due to abrupt shifts in weather patterns. Additionally, the model's capability for long-term forecasting is limited, as RNNs inherently struggle with retaining information over extended sequences, which reduces their effectiveness for predicting far-future trends.

D. Future Improvements

To enhance the model's performance and address its limitations, several avenues for future improvements can be explored. Increasing the sequence length used for training could allow the model to capture longer temporal dependencies, potentially improving its handling of long-term trends. Incorporating additional meteorological variables, such as wind speed, solar radiation, and precipitation, could provide a more comprehensive understanding of the factors influencing temperature, thereby improving prediction accuracy. Implementing attention mechanisms could help the model focus on the most relevant parts of the input sequence, reducing prediction lag and improving its ability to handle rapid changes. Finally, exploring ensemble approaches by combining the RNN with other models, such as CNNs or transformers, could leverage complementary strengths and further enhance predictive accuracy.

APPENDIX

GitHub Repository: https://github.com/sinhaparth5/climate_rnn_project
Delhi Daily Climate dataset: https://www.kaggle.com/datasets/sukhmandeepsinghbrar/daily-delhi-climate