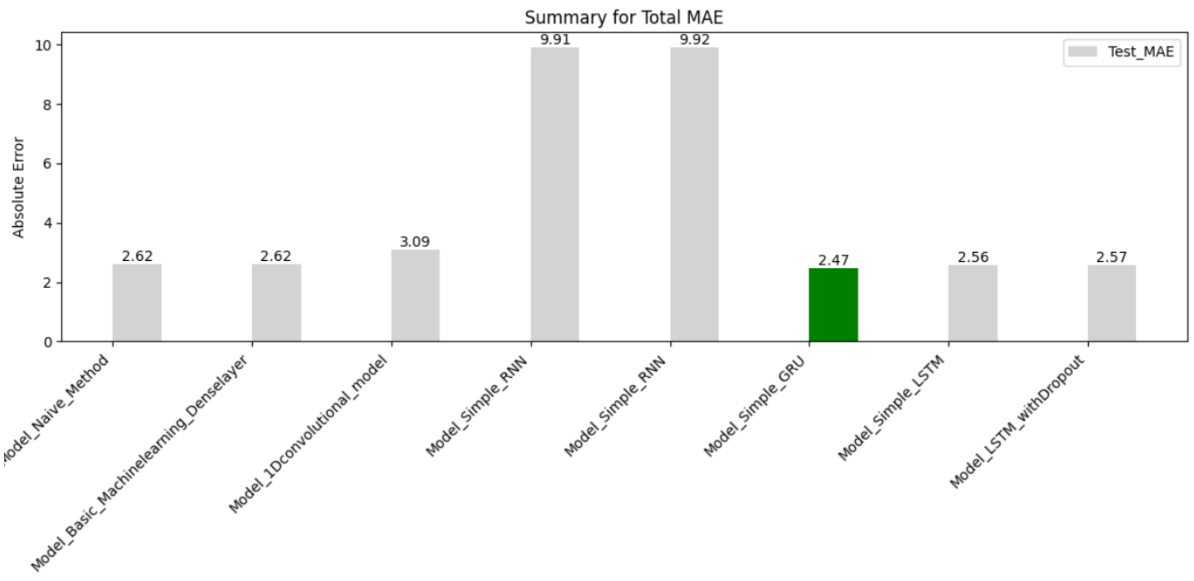
Assignment 3 Weather Time Series Forecasting Using RNN architecture

Name: Kandarp Barot

Date: 7/20/2024

In this assignment we have created 14 Different time series models for weather forecasting. We have considered 10days data point to create these models. The first 8 Models are samples of different model architectures. We will see basically which architecture is best in weather forecasting.



Here are the various model and their MAE. The **Naive Method** uses the last observed value as a prediction with no complex architecture; **Basic Machine Learning with Dense Layer** employs fully connected layers to learn from input features; **1D Convolutional Model** captures spatial hierarchies using convolutional and pooling layers; **Simple RNN** handles sequential data with recurrent layers maintaining a hidden state; **Simple GRU** improves RNNs with gating mechanisms to control information flow; **Simple LSTM** uses gates to manage long-term dependencies; and **LSTM with Dropout** includes dropout layers to prevent overfitting.

- **Best Model**: **Simple GRU** (2.470000 MAE) - Effective in capturing sequential dependencies with gating mechanisms.

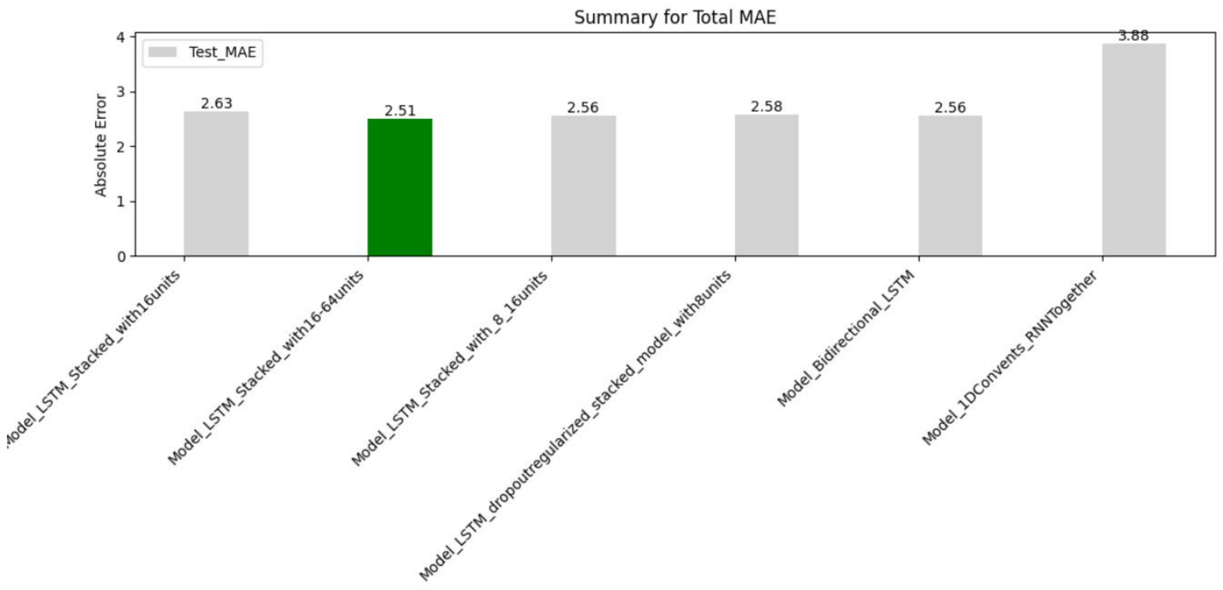
- **Worst Model**: **Simple RNN** (9.910000 and 9.920000 MAE) - Struggles with long-term dependencies and vanishing gradient issues.

As we understood the best architecture for the weather forecast is GRU, but we will still proceed with LSTM and try different modifications in it to get better results.

Assignment Task:

1. Adjusting the number of units in each recurrent layer in stacked setup & Using layer_lstm() instead of layer_gru(): Created 5 different models “LSTM_stacked with 16 units”, “LSTM_stacked with 16 -64 units” , “LSTM_stacked with 8-16 units”, “LSTM_stacked with 8 units dropout regularization”

3. Using a combination of 1d_convnets and RNN.: Created “Model_1DConvents_RNNTogether”



- **Best Model**: **LSTM_stacked with 16 -64 units** (2.510000 MAE)
- **Worst Model: Model_1DConvents_RNNTogether** (3.88 MAE)

Conclusion:

We created a total of 14 models. The first one, the Naive Method baseline, yielded a Mean Absolute Error (MAE) of 2.62. Subsequently, we developed a basic machine learning model using a dense layer, which also resulted in an MAE of 2.62, indicating that the added complexity did not enhance predictive power for this dataset. More complex models are expected to perform better. We also tried a convolutional model, but it produced poor results because the convolutional approach treated every segment of the data uniformly, and pooling destroyed the order of information.

Recognizing the need for a specific architecture for time series data, we turned to Recurrent Neural Networks (RNNs). RNNs can use information from previous steps in their current decision-making process, capturing dependencies and patterns within sequential data. The internal state of an RNN serves as a memory of previous inputs, making it capable of modeling sequences of arbitrary lengths. However, **the simple RNN proved too simplistic and was the worst performer due to the vanishing gradient problem**, which makes deep networks untrainable. **To address this, we used LSTM and GRU RNNs, which are designed to handle this issue and are available in Keras. The simple GRU showed the best results among all models due to its ability to capture long-range dependencies in sequential data while being computationally less expensive than LSTMs.**

LSTMs are renowned for handling time series data, and we experimented with six different LSTM models by varying the units in stacking recurrent layers to 8, 16, and 64. The model with 64 units performed the best. We also incorporated recurrent dropout to prevent overfitting and used bidirectional data to present the same information to the network in different ways, enhancing accuracy and mitigating the forgetting issue. All these models had MAE values close to each other and lower than the Naive Method model, as confirmed by the MAE evaluation graph.

Finally, we built a model combining a 1D convolutional model and RNN, but it resulted in a poor MAE of 3.88, likely due to the convolutional approach disrupting the order of information.

Recommendations

As observed simple RNNs struggle with the vanishing gradient problem, making them less effective at capturing long-term dependencies. Therefore, **it's advisable to use more advanced RNN architectures like LSTM and GRU**, which are designed to address these issues. While LSTM is a popular choice for handling time series data due to its ability to capture long-term dependencies, experiments have shown that GRU can be a more efficient option. **I recommend optimizing GRU by tuning hyperparameters such as the number of units in the stacked recurrent layers, the recurrent dropout rate, and the use of bidirectional data.**

Based on my results, the combination of 1D convolution and RNN did not perform well. Given the limitations of the convolutional approach in preserving the order of information in your time series data, **it’s advisable to focus on architectures specifically designed for handling sequential data, such as pure RNNs.**