Predicting Coffee Machine Energy Consumption

Christos Panourgias

Contents

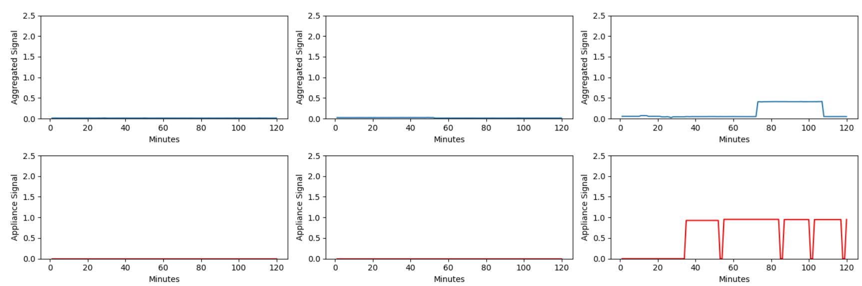
1 Introduction

The ability to accurately forecast energy requirements not only facilitates informed decision-making, but also enables efficient resource allocation and demand-side management. In the domestic context, coffee machines are among the most frequently used devices, contributing to a significant part of energy consumption. Therefore, accurately predicting energy usage can lead to significant energy savings and improved user experience.

Traditional time series analysis techniques often fall short in capturing the complex, non-linear relationships inherent in energy consumption patterns. To overcome this limitation, recurrent neural networks (RNN) have emerged as powerful tools for modeling sequential data due to their ability to capture temporal dependencies. In particular, the LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Units) architectures have gained prominence for their ability to handle long-term dependencies and mitigate the problem of vanishing gradients.

In this work, the application of RNN, LSTM and GRU models to predict 120-minute coffee machine consumption time series was investigated, using whole-house energy consumption information for each 120-minute period.

Below are graphs of home consumption and coffee machine consumption for three random 120-minute windows.



2 Applied Methods

2.1 Data Normalization

Normalization was first applied to the data by dividing by the true maximum value. This is necessary because time series data often exhibit significant variations in size and scale. Normalization helps bring the data into a consistent range, and ensures that all features contribute proportionally to the analysis and model training process, preventing any particular feature from dominating due to its larger values. Also, outliers can distort statistical measures and affect model performance.

2.2 Data Separation

The normalized data is divided into training set, validation set and test set with percentages of 70%, 20% and 10% respectively. Thus it will be possible to tune the hyper-parameters and evaluate the model on new data that is independent of the data it was trained on.

2.3 Data Preprocessing

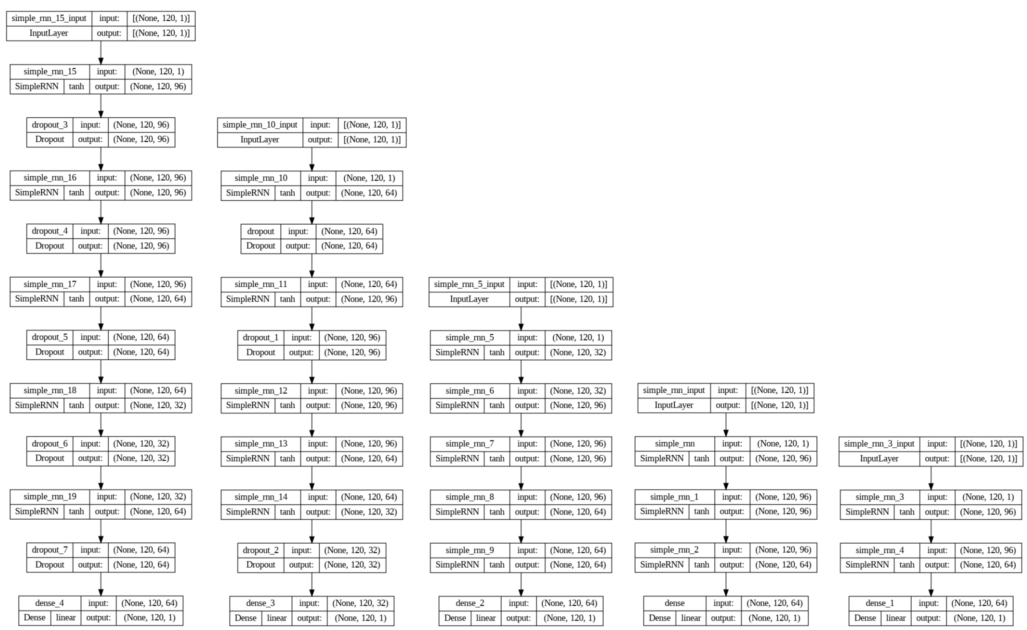
The keras library models consider 120 minute time series as an attribute, so an extra dimension is added to the data in order for it to be inputted into the models.

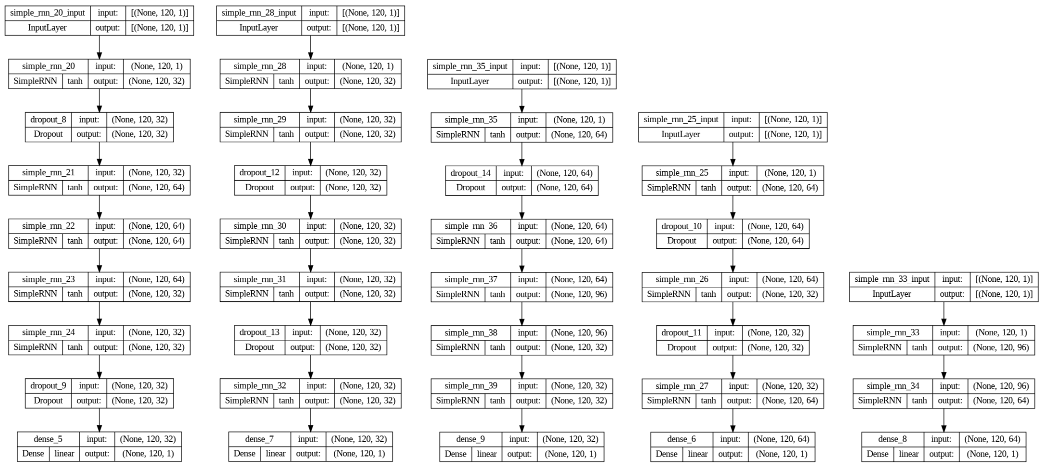
2.4 Building Different Architectures

For each type of model, 10 different models are created by randomly choosing the layers, the units each layer has, and the initial value of the learning rate.

Below are the architectures that were constructed for each type of RNN.

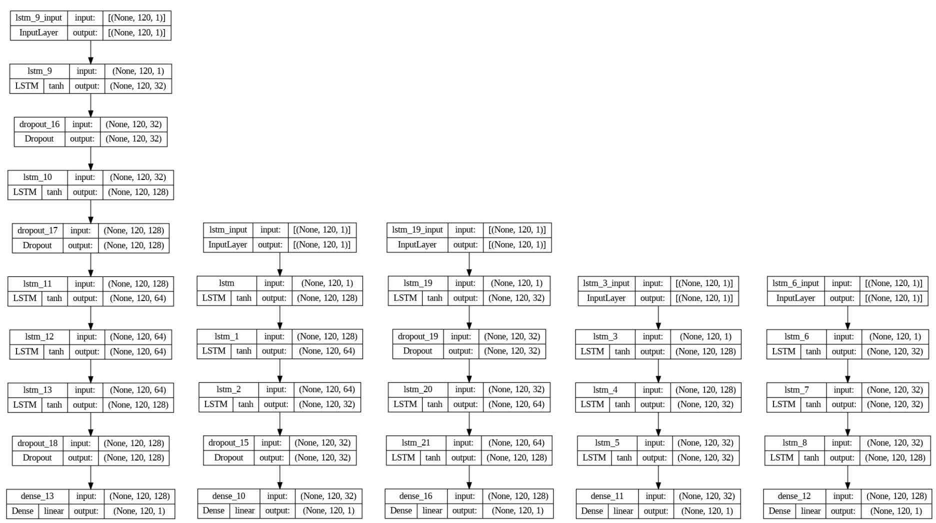
2.4.1 RNN



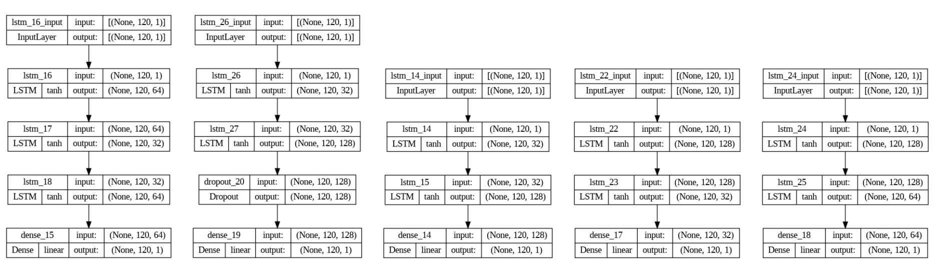


simple_rnn_20_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
simple_rnn_20 | SimpleRNN | tanh | input: (None, 120, 1) | output: (None, 120, 32)
dropout_8 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
simple_rnn_21 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 64)
simple_rnn_22 | SimpleRNN | tanh | input: (None, 120, 64) | output: (None, 120, 64)
simple_rnn_23 | SimpleRNN | tanh | input: (None, 120, 64) | output: (None, 120, 32)
simple_rnn_24 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dropout_9 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
dense_5 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
simple_rnn_28_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
simple_rnn_28 | SimpleRNN | tanh | input: (None, 120, 1) | output: (None, 120, 32)
simple_rnn_29 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dropout_12 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
simple_rnn_30 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
simple_rnn_31 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dropout_13 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
simple_rnn_32 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dense_7 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
simple_rnn_35_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
simple_rnn_35 | SimpleRNN | tanh | input: (None, 120, 1) | output: (None, 120, 64)
dropout_14 | Dropout | input: (None, 120, 64) | output: (None, 120, 64)
simple_rnn_36 | SimpleRNN | tanh | input: (None, 120, 64) | output: (None, 120, 64)
simple_rnn_37 | SimpleRNN | tanh | input: (None, 120, 64) | output: (None, 120, 96)
simple_rnn_38 | SimpleRNN | tanh | input: (None, 120, 96) | output: (None, 120, 32)
simple_rnn_39 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dense_9 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
simple_rnn_25_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
simple_rnn_25 | SimpleRNN | tanh | input: (None, 120, 1) | output: (None, 120, 64)
dropout_10 | Dropout | input: (None, 120, 64) | output: (None, 120, 64)
simple_rnn_26 | SimpleRNN | tanh | input: (None, 120, 64) | output: (None, 120, 32)
dropout_11 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
simple_rnn_27 | SimpleRNN | tanh | input: (None, 120, 32) | output: (None, 120, 64)
dense_6 | Dense | linear | input: (None, 120, 64) | output: (None, 120, 1)
simple_rnn_33_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
simple_rnn_33 | SimpleRNN | tanh | input: (None, 120, 1) | output: (None, 120, 96)
simple_rnn_34 | SimpleRNN | tanh | input: (None, 120, 96) | output: (None, 120, 64)
dense_8 | Dense | linear | input: (None, 120, 64) | output: (None, 120, 1)

2.4.2 LSTM

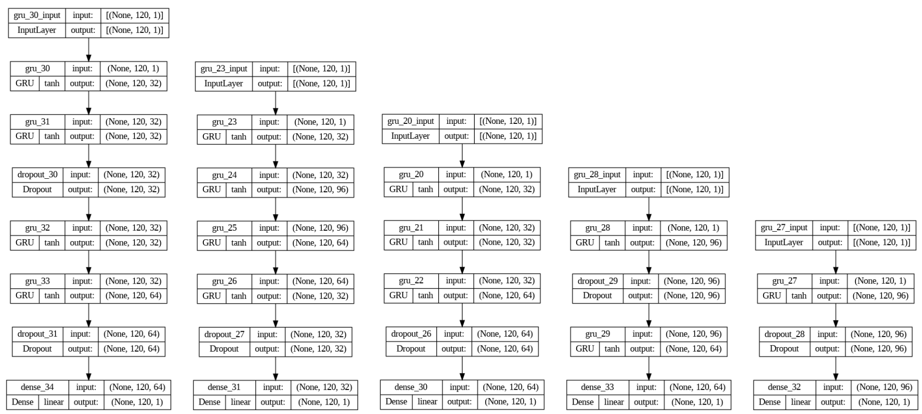


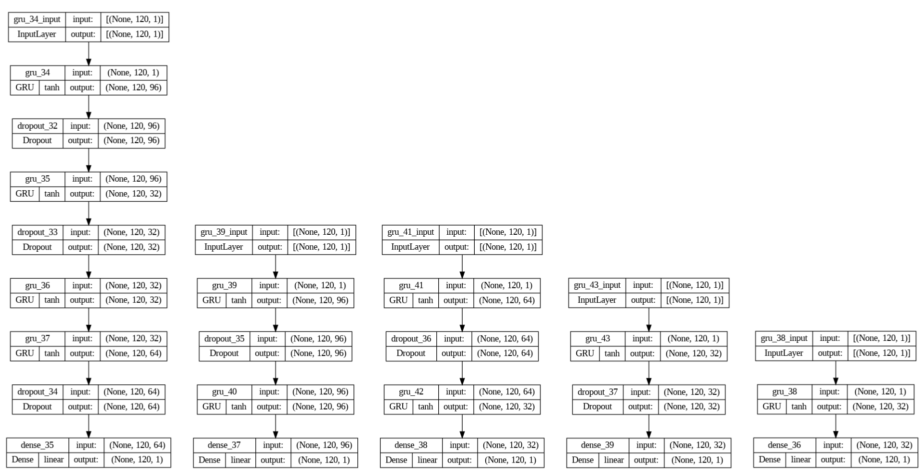
lstm_9_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_9 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 32)
dropout_16 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
lstm_10 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 128)
dropout_17 | Dropout | input: (None, 120, 128) | output: (None, 120, 128)
lstm_11 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 64)
lstm_12 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 64)
lstm_13 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 128)
dropout_18 | Dropout | input: (None, 120, 128) | output: (None, 120, 128)
dense_13 | Dense | linear | input: (None, 120, 128) | output: (None, 120, 1)
lstm_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 128)
lstm_1 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 64)
lstm_2 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 32)
dropout_15 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
dense_10 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
lstm_19_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_19 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 32)
dropout_19 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
lstm_20 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 64)
lstm_21 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 128)
dense_16 | Dense | linear | input: (None, 120, 128) | output: (None, 120, 1)
lstm_3_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_3 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 128)
lstm_4 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 32)
lstm_5 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 32)
dense_11 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
lstm_6_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_6 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 32)
lstm_7 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 32)
lstm_8 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 128)
dense_12 | Dense | linear | input: (None, 120, 128) | output: (None, 120, 1)



lstm_16_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_16 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 64)
lstm_17 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 32)
lstm_18 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 64)
dense_15 | Dense | linear | input: (None, 120, 64) | output: (None, 120, 1)
lstm_26_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_26 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 32)
lstm_27 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 128)
dropout_20 | Dropout | input: (None, 120, 128) | output: (None, 120, 128)
dense_19 | Dense | linear | input: (None, 120, 128) | output: (None, 120, 1)
lstm_14_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_14 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 32)
lstm_15 | LSTM | tanh | input: (None, 120, 32) | output: (None, 120, 128)
dense_14 | Dense | linear | input: (None, 120, 128) | output: (None, 120, 1)
lstm_22_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_22 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 128)
lstm_23 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 32)
dense_17 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)
lstm_24_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm_24 | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 128)
lstm_25 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 64)
dense_18 | Dense | linear | input: (None, 120, 64) | output: (None, 120, 1)

2.4.3 GRU





2.5 Model Training and Selection

The model training part is divided into two parts, the first part where the models are trained and the best one is selected, and the second part where the best model is trained. In each part of the training the optimizer is Adam and the error that the model is trained on is Mean Squared Error.

2.5.1 Training to Find the Best Model

Initially, it is desirable, for each type, to find the most efficient model, but training each model so that the best one can be found is quite time-consuming. To mitigate this problem the training of each model will be limited to a random sample from the initial training data population of 10000 data. Before starting the training, the number of epochs and the batch size of the data are randomly selected

During training, the learning rate decreases per epoch, according to the formula $lr_0 \cdot e^{-\frac{lr_0}{5} \cdot epoch}$. Additionally if the validation set error has not decreased for 5 consecutive epochs, training is stopped. Finally, a checkpoint is used using the Callbacks API so that after the end of the training, the model is saved in the state where it has the minimum error in the validation set.

Therefore, once the training is completed, the model that had the best results is obtained.

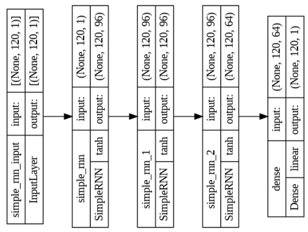
2.5.2 Training the Best Model

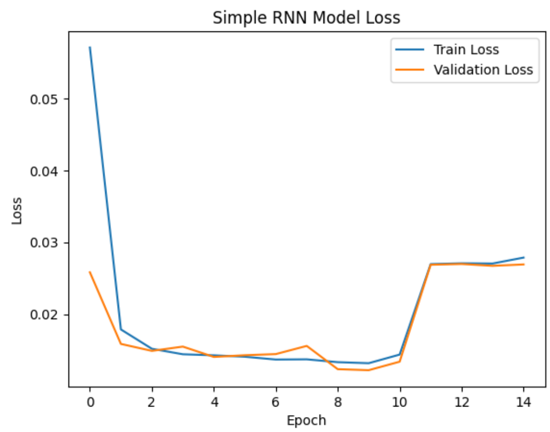
The parameters of the selected model of the first part are reset to be untrained, and are then trained on the entire training data set. Learning rate reduction, checkpoint and early stopping apply again. The initialization of the number of epochs is the same as in the first part while the initialization of batch size is such that the model processes as much data as it did in the training epochs of the first part.

3 Experimental Results

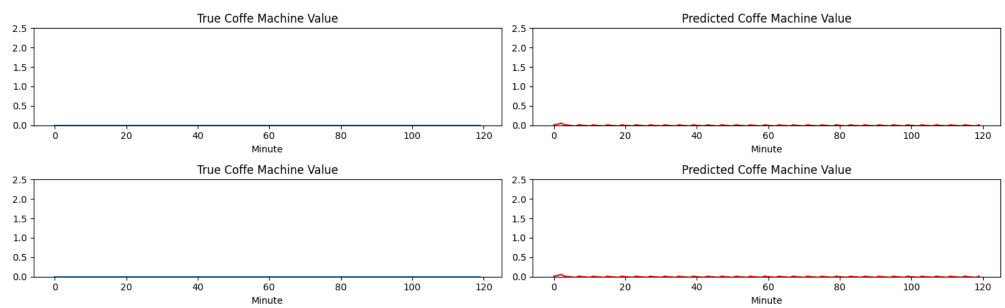
Following the above procedure, the models selected from each type are as follows:

RNN

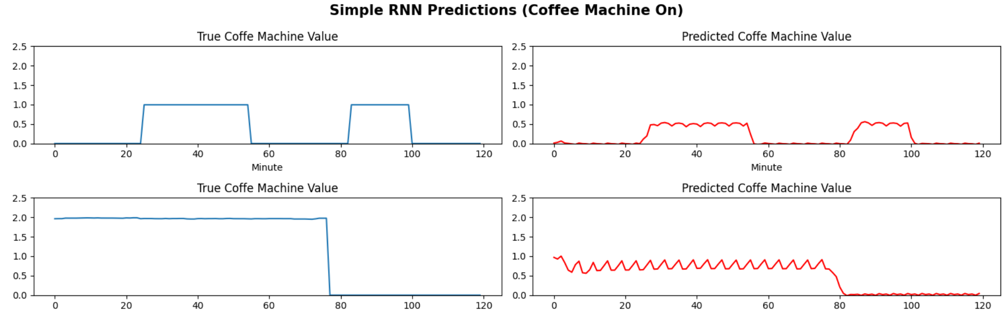


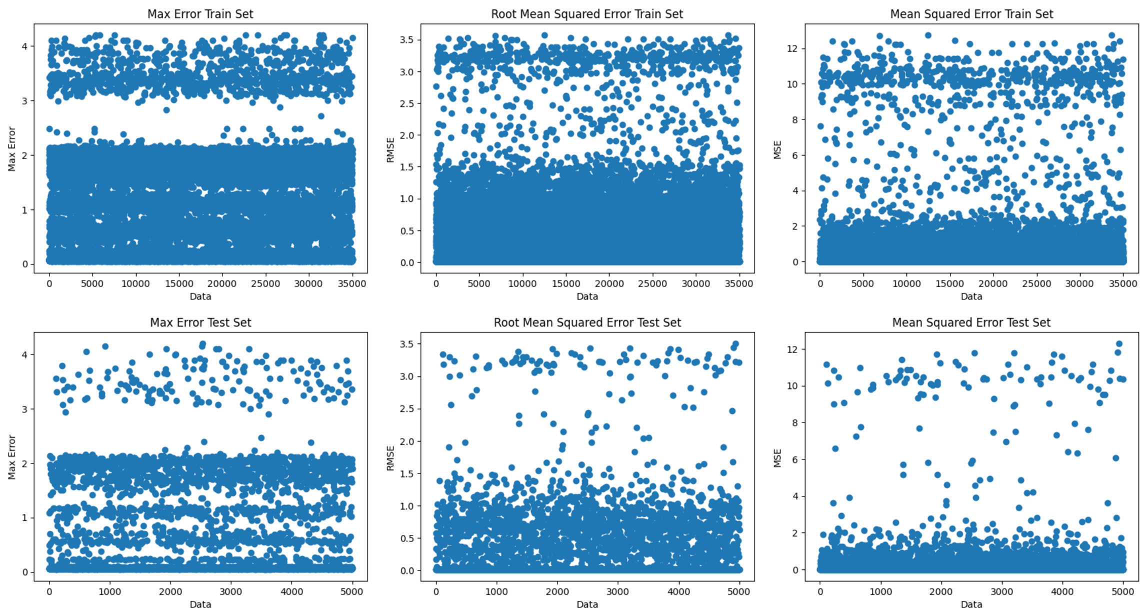


Simple RNN Predictions (Coffee Machine Off)



Simple RNN Predictions (Coffee Machine On)

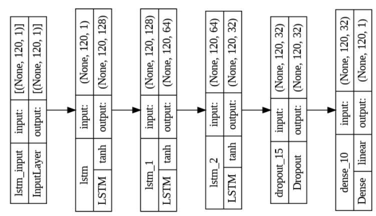


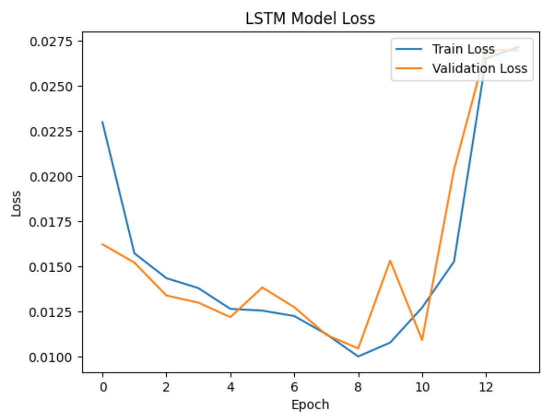


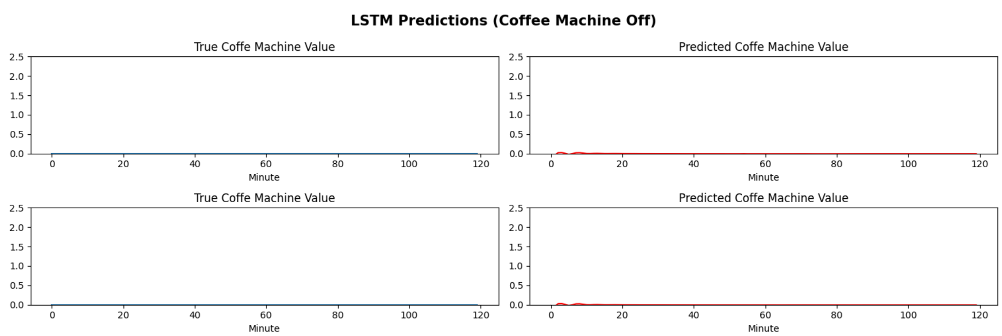
The first graph shows the architecture of the model which has three hidden RNN layers and the output and input layers. From the error plot it is observed that the minimum error of the validation set is obtained around the 8th to 9th epoch, where due to the checkpoint the model will be saved in that state.

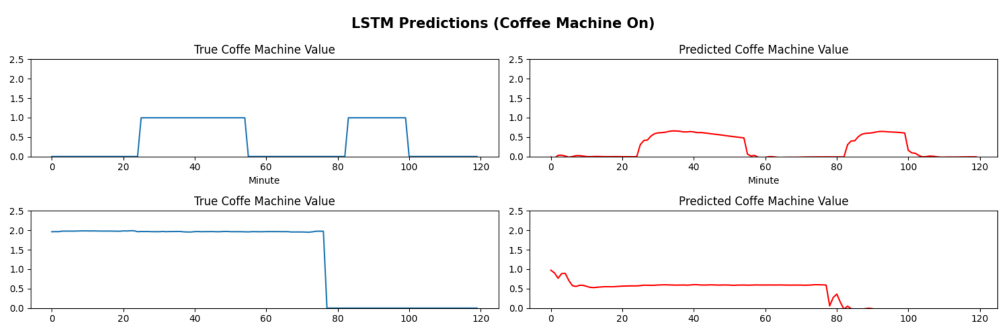
From the last two graphs it can be concluded that the model predictions when the coffee machine is off seem to be correct on average, while the model predictions when the coffee machine is on deviate quite a bit from the actual values with the predicted values being lower than real ones, this is also confirmed by the dispersion observed in the graphs of ME, MSE and RMSE of the test and training sets. However, it is worth noting that the model has captured the general pattern of the actual values.

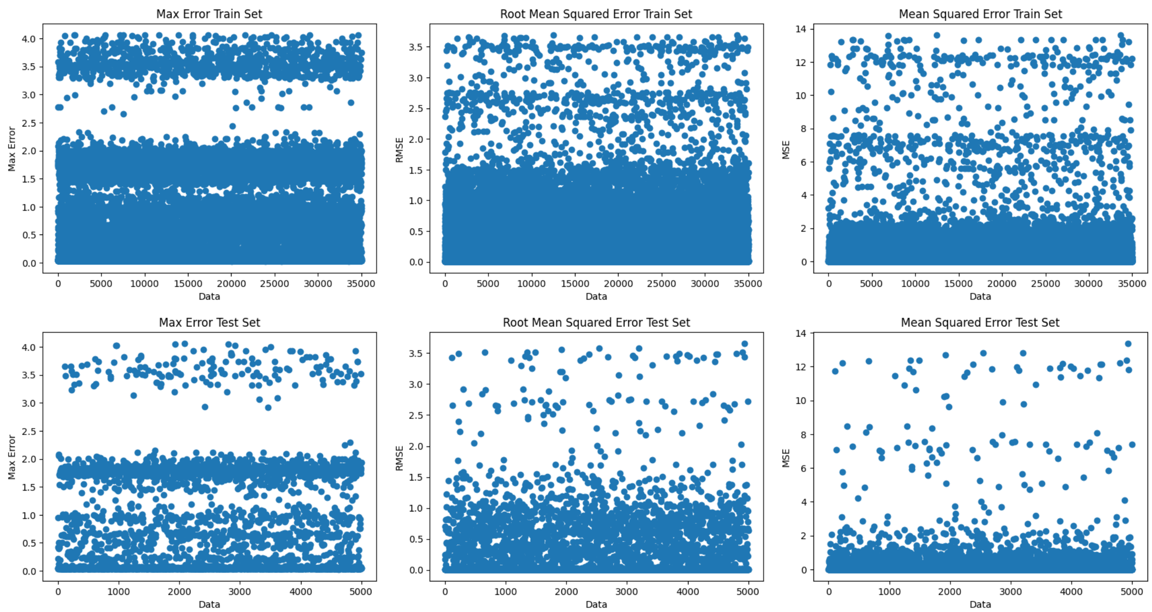
LSTM


lstm_input | InputLayer | input: [(None, 120, 1)] | output: [(None, 120, 1)]
lstm | LSTM | tanh | input: (None, 120, 1) | output: (None, 120, 128)
lstm_1 | LSTM | tanh | input: (None, 120, 128) | output: (None, 120, 64)
lstm_2 | LSTM | tanh | input: (None, 120, 64) | output: (None, 120, 32)
dropout_15 | Dropout | input: (None, 120, 32) | output: (None, 120, 32)
dense_10 | Dense | linear | input: (None, 120, 32) | output: (None, 120, 1)


LSTM Model Loss
Train Loss
Validation Loss
Loss
Epoch


LSTM Predictions (Coffee Machine Off)
True Coffe Machine Value
Predicted Coffe Machine Value
Minute


LSTM Predictions (Coffee Machine On)
True Coffe Machine Value
Predicted Coffe Machine Value
Minute

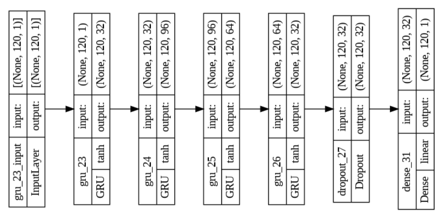


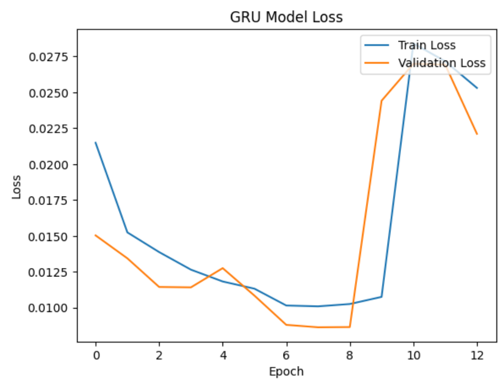
The model architecture has three hidden LSTM layers after the input layer and a Dropout layer which is connected to the output layer.

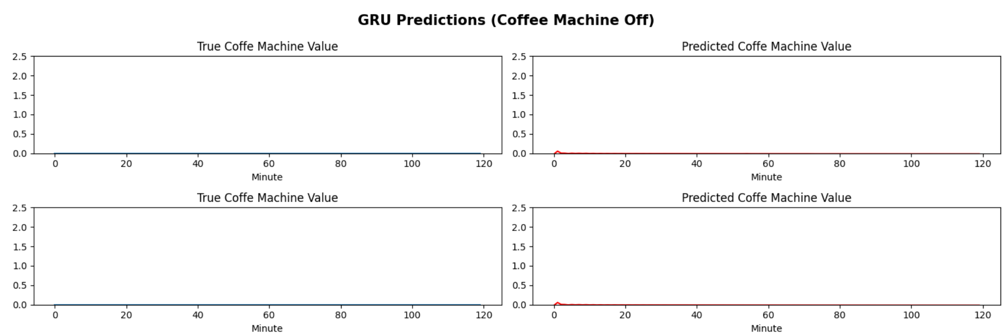
The minimum error of the validation set occurs at epoch 8. As before, because of the checkpoint the model is saved in that state.

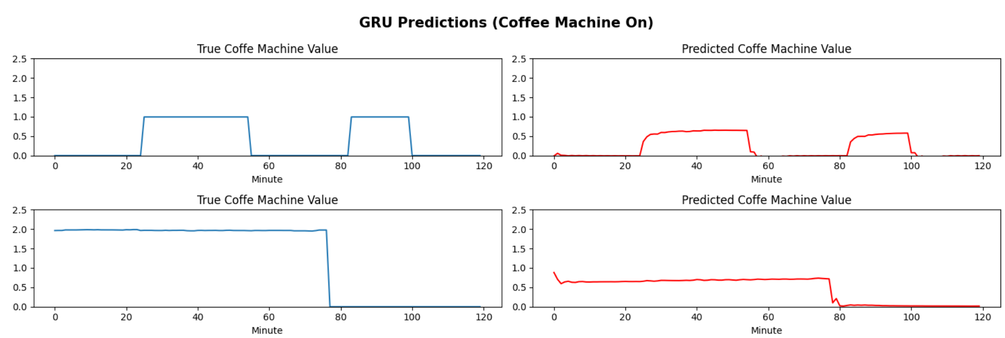
Next come the time series predictions where, like RNN, LSTM predicts the values of the coffee machine when it is off quite well. When the coffee machine is on it is observed that LSTM makes smoother predictions that are closer to the true form of the values in contrast to RNN, however as with RNN it is observed that the predictions have lower values than the real ones which is also evident from the dispersion observed in the graphs of ME, MSE and RMSE.

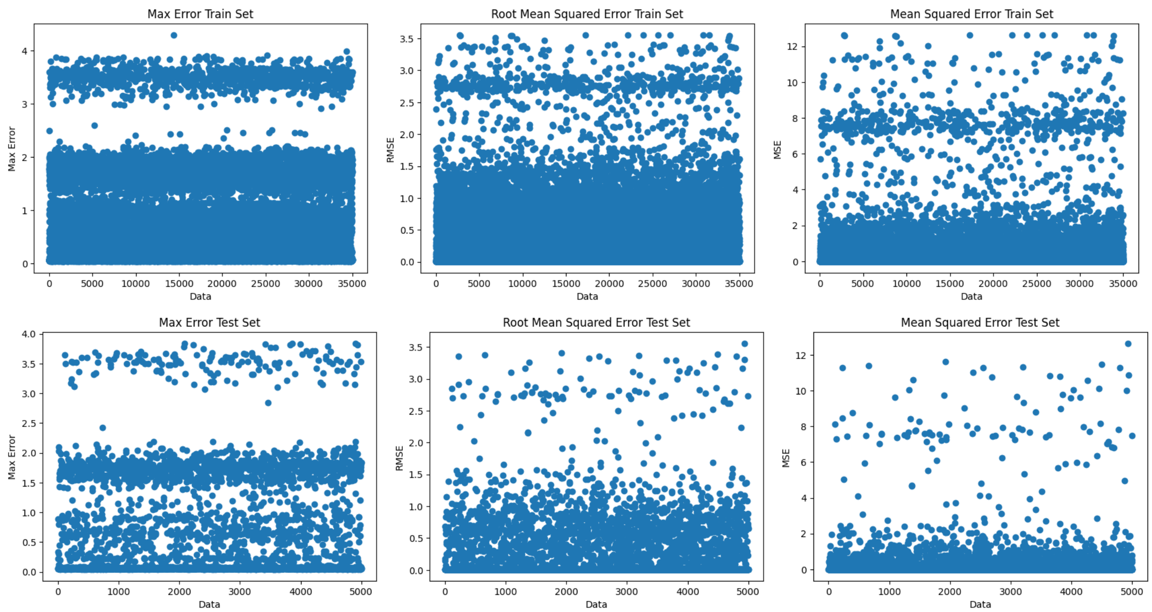
GRU


gru_23_input | InputLayer | input: | output: | [(None, 120, 1)] | [(None, 120, 1)]
gru_23 | GRU | tanh | input: | output: | (None, 120, 1) | (None, 120, 32)
gru_24 | GRU | tanh | input: | output: | (None, 120, 32) | (None, 120, 96)
gru_25 | GRU | tanh | input: | output: | (None, 120, 96) | (None, 120, 64)
gru_26 | GRU | tanh | input: | output: | (None, 120, 64) | (None, 120, 32)
dropout_27 | Dropout | input: | output: | (None, 120, 32) | (None, 120, 32)
dense_31 | Dense | linear | input: | output: | (None, 120, 32) | (None, 120, 1)


GRU Model Loss
Train Loss
Validation Loss
Loss
Epoch


GRU Predictions (Coffee Machine Off)
True Coffe Machine Value
Predicted Coffe Machine Value
Minute


GRU Predictions (Coffee Machine On)
True Coffe Machine Value
Predicted Coffe Machine Value
Minute



GRU consists of 5 GRU layers and a Dropout layer which leads to the output layer.

The minimum value of the error in the validation set is observed at epoch 8 where the model is stored.

From the predictions it is observed that they do not differ greatly from the LSTM predictions. However, a small noise reduction is observed in the prediction of the ”turning off” of the coffee machine. GRU, like the previous ones, has an error in predicting the actual values, predicting lower values in their place, as mentioned in the analysis of the results of the previous models, this is also evident from the graphs of ME, MSE and RMSE.

4 Conclusions

In conclusion, the study on predicting coffee machine consumption time series using RNN, LSTM and GRU architectures provided valuable insights into their performance and effectiveness. Among the three architectures, GRU emerged as the most effective in predicting coffee machine consumption, followed by LSTM and then RNN with the largest difference being shown between RNN and the rest. The ability of the GRU model to handle long-term dependencies and mitigate the problem of vanishing gradients likely contributed to its superior performance. LSTM, with its specialized memory cell structure, also demonstrated strong predictive abilities. While RNN showed reasonable performance, it lagged behind the other two architectures due to its limitations in capturing long-term dependencies.

Overall, the results highlight the importance of leveraging advanced recurrent neural network architectures for accurate time series forecasting tasks. Further research can explore optimizations and improvements to these architectures, potentially leading to even more accurate predictions and facilitating better energy management in households.