Predicting the Future Energy Consumption of UVA Smart Buildings with Machine Learning Models

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Ruthvik Gajjala

Spring, 2023

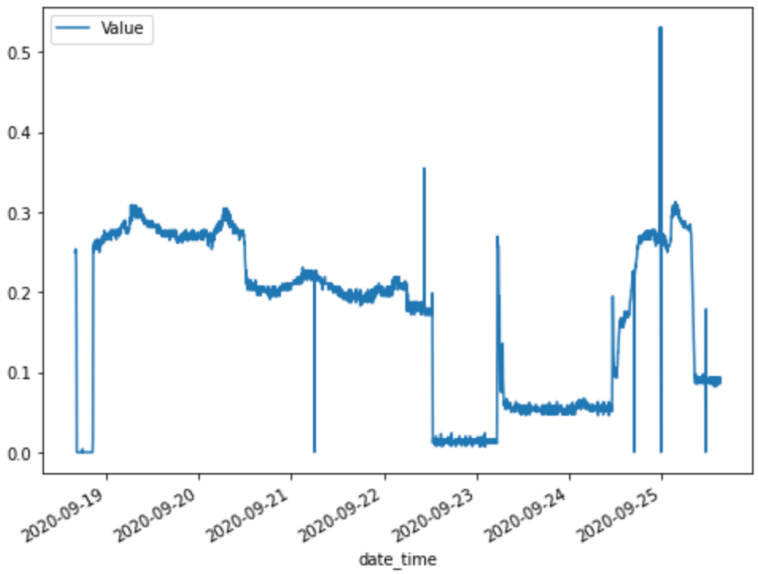
On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

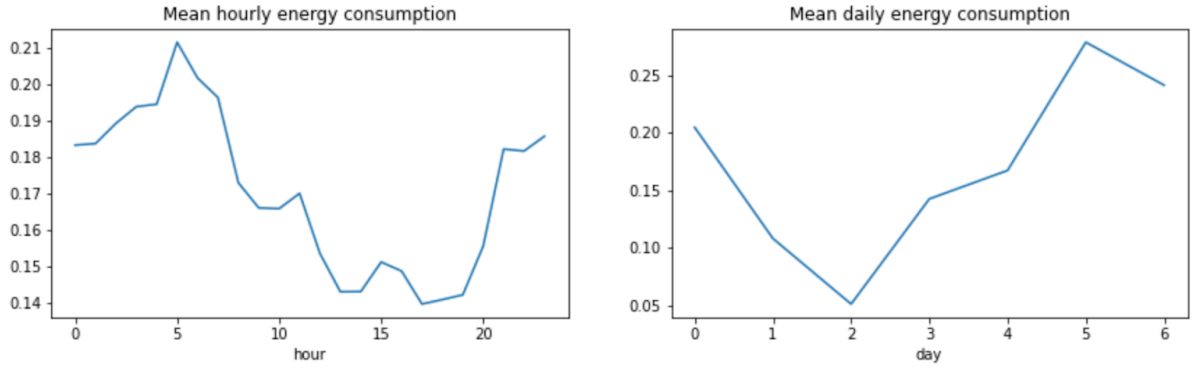
Haiying Shen, Department of Computer Science

Technical Report

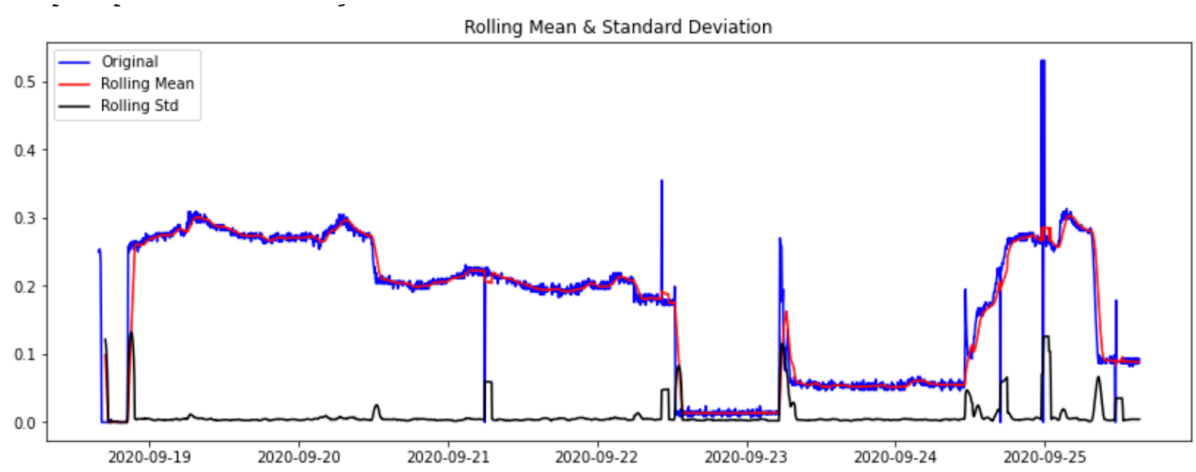
To start off the modeling process, I first had to explore the data in order to examine how it needed to be altered in order to fit the task of fitting an LSTM model. The first big step was shrinking the size of the energy dataset to only the relevant pieces of data, which was a big task considering the sheer size of data. Since I knew that the data was labeled under "real energy data", I knew that I could filter the dataset and keep only the data labeled under this name. After doing so, I soon realized that the dataset stored the data for various energy detection devices placed around Rice Hall, and it would be difficult to create a model for each of them because they each contained different trends and different detected energy values over time. I also found that most of these devices did not detect anything at all, and were thus unusable for the prediction task. This, and the fact that we only had data in one week at a time, made the prediction task very difficult overall. Predicting one day of data was essentially predicting nearly 20% of the dataset at a single time, so I instead decided on predicting shorter multi step intervals, such as 1 hour to 12 hours into the future. The data itself was taken in 5-minute intervals, so I then had to figure out which device I should use to perform the prediction task. Initially, I had tried to create a model based on the sum of detected energy output from all the devices, but this was not possible due to the huge variance in energy consumption between devices. To decide on a device to produce the most reasonable predictions, I tried to find the devices with the most individual variance across their detected energy output. One of the devices that had such high variance was device ELEUV0214_SM14_0214HMSBB_14, with a standard deviation of 0.001652 across all of its detected energy data. This is not very high at all, but relative to the other devices in the dataset it was very high. After choosing this particular device, I found that the data was collected cumulatively rather than measuring the energy output at that particular time step. The data was not particularly well-labeled, and this made even predicting outputs very difficult during my time working with this data. To combat this, I 1-lag differenced the data initially to calculate the energy usage at each particular timestep. After doing so, I was able to see that the energy output detected from this device looked like the graph below.



Unfortunately, no data for any device that I could find showed any clear trends over the days of the week, and the energy output seemed to be very sporadic. This already presented the notion that even with an LSTM, the results of multistep predictions may not be accurate at all. In addition, after all the filtering that had to be done to the data, the dataset size shrunk from over a million entries to just over 2000, which is not nearly enough for reliable multistep predictions; this also limited the trends that could be found in the data for accurate predictions. Following this, I extensively visualized the data, from rolling averages to day-by-day averages, to understand the energy consumption detected from the specific device I was using.



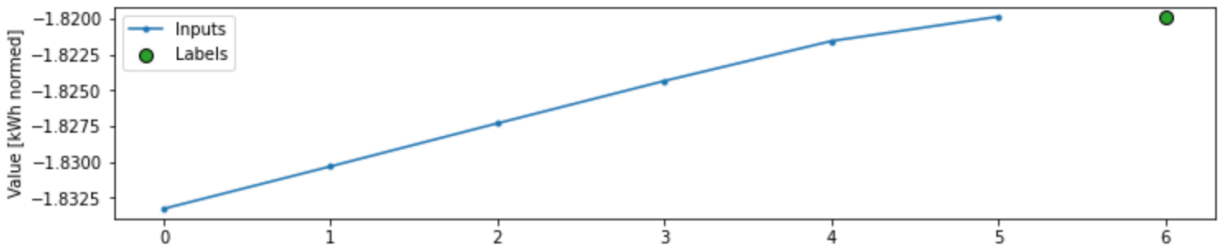
Given that I was relatively new to using neural networks and machine learning in general, visualizing the data greatly helped my understanding of what was going on. I also checked for the stationarity of the data I was using, as this is important for predicting time series data.



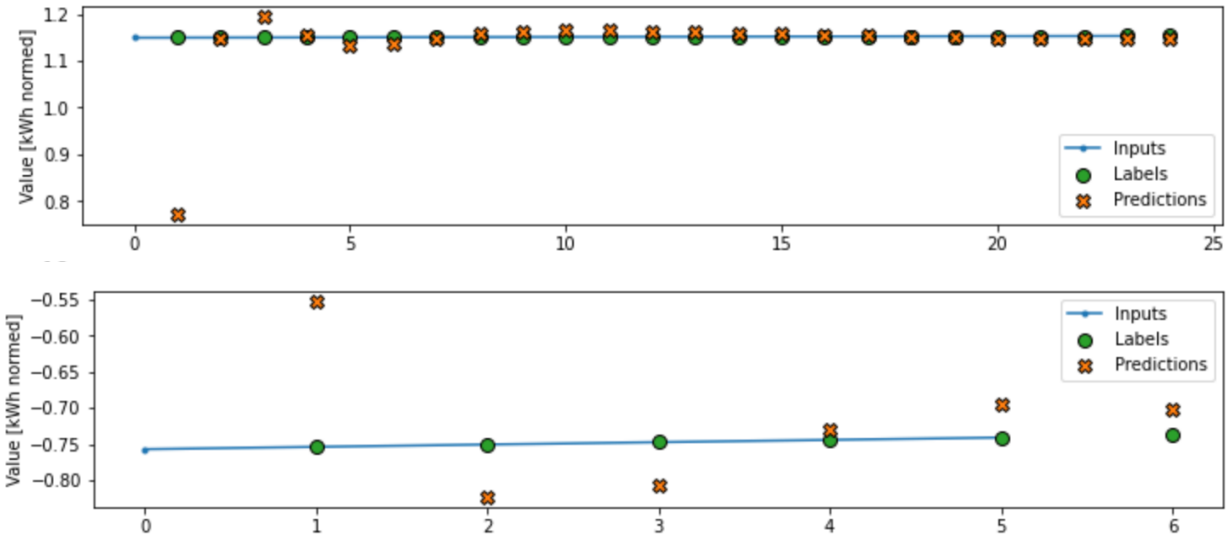

```
<Results of Dickey-Fuller Test>
Test Statistic                  -2.454293
p-value                          0.126995
```

My next task was then to simply visualize what LSTM predictions may look like, and I spent a solid couple of weeks understanding how LSTM's work. This is mainly due to the fact that my main objective of the research course was to learn about how LSTM's work, given that I had very little knowledge on neural networks and time series predictions coming into the research course. In the Energy_Data_Analysis.py file, which was my first attempt at creating a model, I basically had decided to slowly build up creating a vanilla LSTM network and use it to make basic predictions on data windows taken from the device data; this was based on one of the

many tutorials that I had went through to better understand neural networks and TensorFlow in general. This was done by creating a Window class in python and using a very basic LSTM model to plot predictions on small windows of data. The process of understanding the dimensions of data required by TensorFlow took very much time, especially considering my limited knowledge of dimensionality of ML models prior to the course. As shown below, my basic model would simply try to predict the label based on a small window of data given to it.

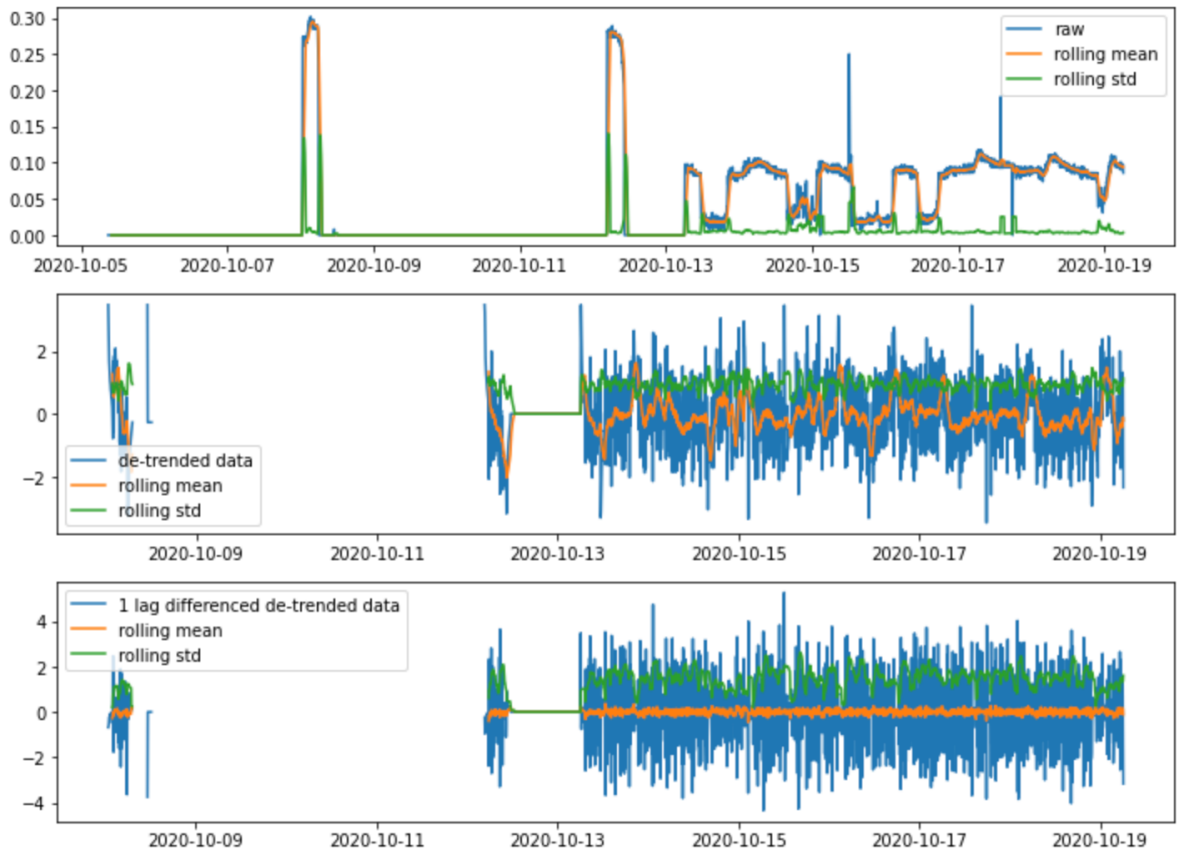


The end result of this initial attempt were various visualizations showing the expected prediction of the vanilla LSTM model versus the actual result. The basic LSTM contained a single hidden layer of 32 neurons, and some of the results are pictured below. The lookback value varied based on the window size.



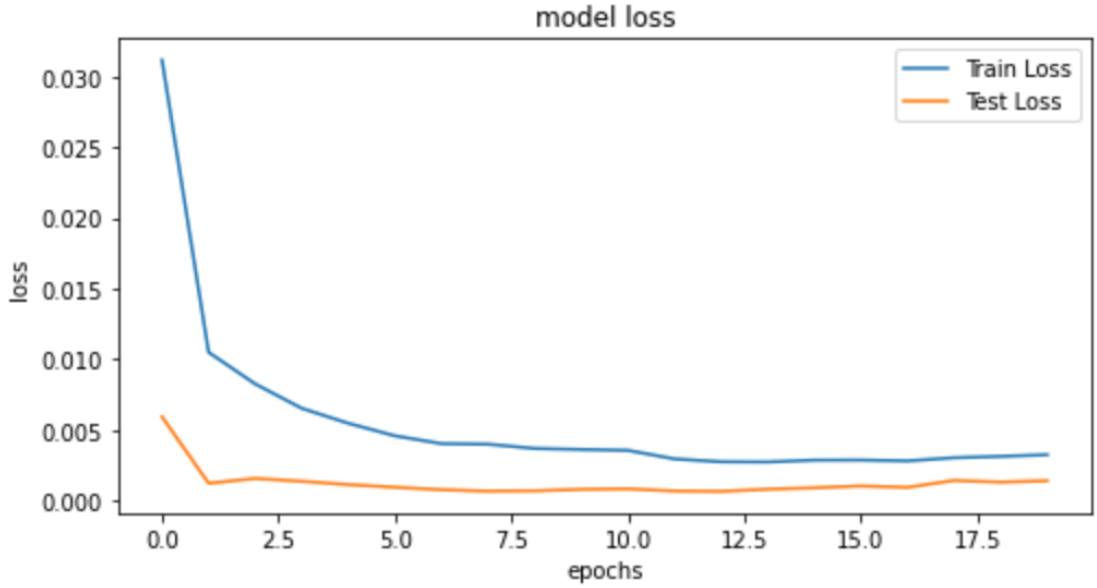
Every point on the line is an energy consumption value from the device in question, with a 5-minute interval between each point. I was also able to change the data window for the predictions to take place. Rather than looking to reduce specific RMSE values, this initial attempt, based off of a TensorFlow tutorial that I used to learn about machine learning, was primarily to help me understand the prediction process. Thus, I spent most of my hours in this initial process learning the ins and outs of LSTM's, reading articles, watching videos, and learning how to apply them to time series data in an effective manner. Once I was able to get a basic understanding of how to create an LSTM in TensorFlow, I then used another week of energy data to repeat the process I had already tried, but this time with a slightly more concrete single step LSTM prediction model. First and foremost, the new week of energy consumption

data had to first be filtered once again in order to find a viable device that has energy values that can be predicted. In this particular case, device ELEUV0214_SM95_0214HMR2_13 showed the highest variance in energy consumption, so this made it the ideal device to predict future usage. I also tried merging the previous weeks data with the new weeks data for the same device in order to have a large dataset to train the model from, but for some odd reason the energy consumption between weeks was not collected accurately, and there were large discrepancies in energy consumption. For this reason, I decided to train the data only using the new week of data that I had received. Once again, I created all of the necessary visualizations and rolling averages, but this time in the new_eneg_analysis.py file, signifying that I was now working with a new week of data.

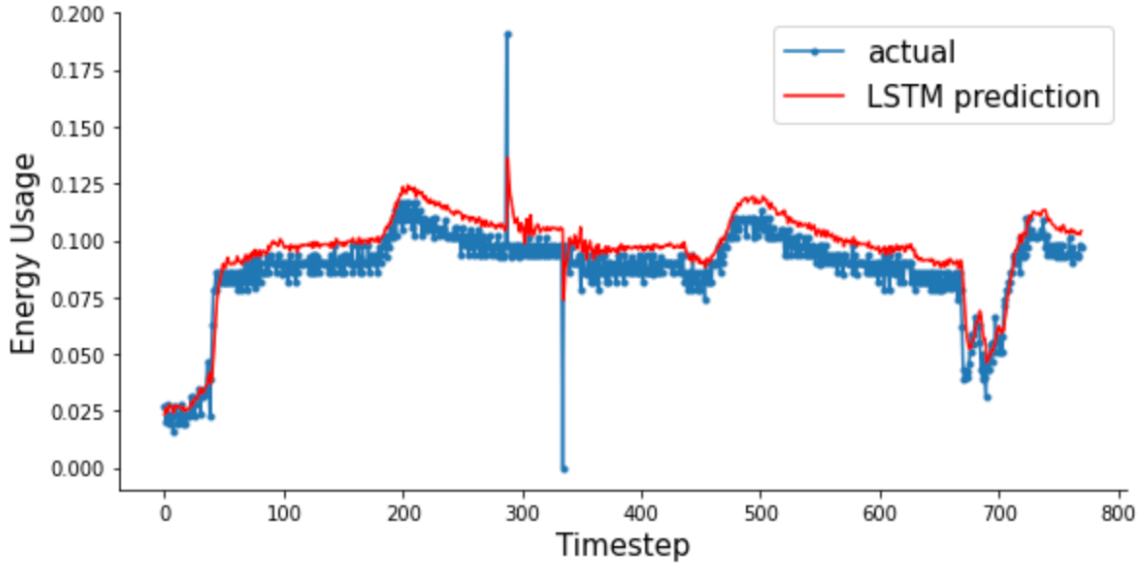


In this case, when testing for the stationarity of the new device I was predicting, I found that the data had an acceptable p-value of 0.000009 in the dickey-fuller test, making it appropriate for time series predictions. Before moving any further, I decided to create a baseline naive model that simply uses the previous time step as the predicted result. This would then allow me to compare with the LSTM I created to see if it is really predicting the next timestep any better than the naive model. The model I created used a hidden LSTM layer with 100 neurons, a dropout layer with 20% dropout, and a single dense output neuron to predict the next timestep. After experimenting with the hyperparameters of the model, I found that using 20 epochs, 100 batches, and a lookback of 28 timesteps was most ideal in terms of balancing

efficiency and accuracy. Training the device data on this model gave the following loss over epochs graph. An 80-20 train-test split was used to train this data and test it.

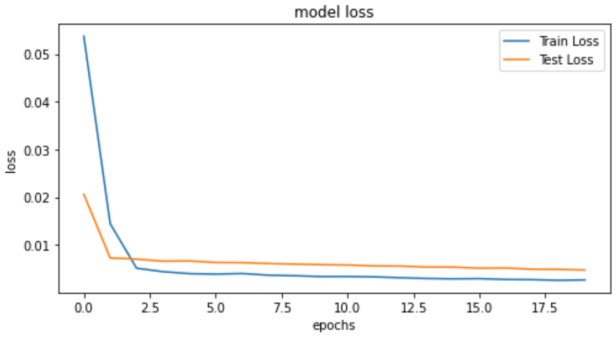


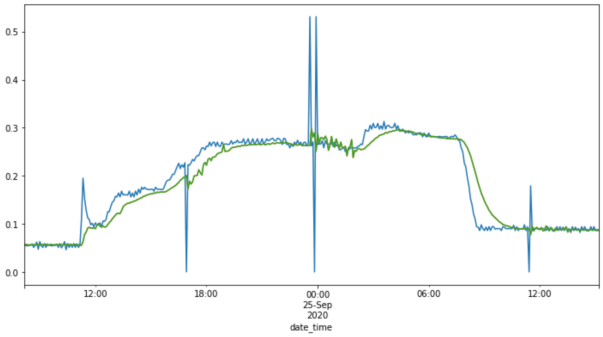
When predicting the results on the test set, I found that the RMSE of the LSTM model was much lower than that of the naive model. The LSTM reported an RMSE value of 0.095 on the test set, while the naive model was at 0.45, a huge difference. While a 0.095 error is not great, considering that I was completely new to creating models using TensorFlow this was a good start and reasonable considering the limitations of the dataset and sporadic data.



As seen in the graph above, the LSTM was not able to predict the outliers in the energy usage data, which is expected considering the volatile nature of the energy prediction data. The single step model was able to give decent results on this device in particular. For this model, I

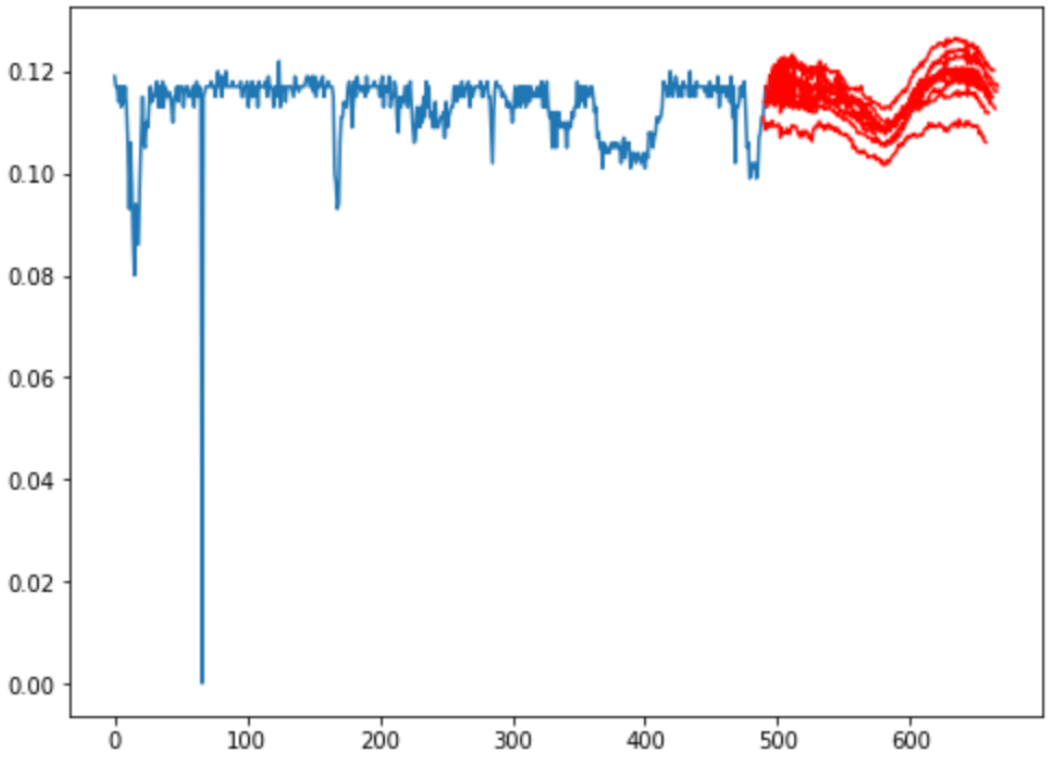
used the “adam” optimizer and relu activation function. I also attempted to create a basic multistep model with the same week of data in the same new_eneg_analysis.py file. This model was trained on random windows in the data, with a single LSTM layer with 100 neurons and a dense output layer with 14 neurons. The reason for 14 neurons was so that I could predict 1 single hour into the future, which is reasonable considering the small size of the dataset. However, evaluating the error of this model proved to be difficult so I instead decided to take an entirely different approach to multistep predictions of energy usage. Since I had a functioning single step model, I altered the original Energy_Data_Analysis.py file and added the single step model to observe the results on the first week of data. A similar result was shown, with an RMSE of .13 for the LSTM model and an RMSE of .41 for the naive model. While slightly higher than the week two data, it still gave a generally good result. With this file altered to include this model, the original prediction visualizations with the data windows no longer worked; this was fine because at this point I already knew the basics of how LSTM’s predict time series data. The model loss and predictions of the single step model are shown below; the actual raw energy usage is blue, and the single step predictions are green.





These predictions were generated on the last bit of energy usage entries for the week so that I could closely observe the results. With single step predictions out of the way, the last step was to generate multistep predictions, as I was unable to before. For this purpose, I created a separate file multi-step.py in order to generate multistep predictions; these predictions could range anywhere from 1 hour after the current timestep, to 1 day after the current timestep. This multistep LSTM model was based off of a tutorial that I used to learn about multistep time series predictions, which is linked in the sources below. I went through the tutorial step by step and did not have to change much in order to generate reasonable predictions. Like usual, I had to do the same data filtering steps to choose a viable device to predict. Once found, the major change that had to be made was to 1-lag difference the data before making it stationary, as the data was cumulative and I did not want to predict the cumulative energy consumption values. One other big change from previous models I had used was the shift to first converting the data into a supervised learning problem, which was then transferred into a multistep problem. For this model, a single LSTM layer with 100 neurons was used, as well as a dense output layer with 168 neurons. The reason for 168 is that 168 time steps is 12 hours into the future, and the model is able to calculate the RMSE at each of these timesteps. Only a single batch was used with the data with a lookback value of 28; since only the previous 28 timesteps were considered, I did not need to train the model on the entire dataset as this was taking too long. Rather, I used the last 500 values of the week’s data and predicted 12 hours into the future for the last 10 data points. Surprisingly, this model outperformed the single step model for one time step into the future,

with an RMSE value of .003 for 100 epochs. In addition, at one hour into the future, the RMSE value was .005. This is expected, as when the model was predicting further into the future larger errors were occurring. The error peaked at around 7 hours into the future, with an RMSE of .01. Overall, the predictions were relatively accurate considering the extreme volatility of the energy consumption data and the limitations in size of the dataset, with only a single week being able to be considered at a time. The multistep predictions for the last 10 timestep are shown in the graph below. The actual data is in blue, and the forecasted time steps are in red. While the forecasted time steps cannot be reliably predicted, this is good enough for the limitations of data. It would be much easier to forecast time series with clear trends based on seasonality and human factors.



References

Brownlee, J. (2019, August 21). *How to convert a time series to a supervised learning problem in Python*. MachineLearningMastery.com. Retrieved from https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

Brownlee, J. (2020, August 27). *Multistep time series forecasting with LSTMs in python*. MachineLearningMastery.com. Retrieved from https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/

Brownlee, J. (2020, December 9). *How to create an Arima model for time series forecasting in Python*. MachineLearningMastery.com. Retrieved from https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

Brownlee, J. (2021, November 18). *Multi-step LSTM time series forecasting models for power usage*. MachineLearningMastery.com. Retrieved from https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

Crocker, B. (2019, November 10). *How to predict a time series part 1*. Medium. Retrieved from https://towardsdatascience.com/how-to-predict-a-time-series-part-1-6d7eb182b540

Olah, C. (2015, August 27). *Understanding LSTM networks*. Understanding LSTM Networks -- colah's blog. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Singh, A. (2023, February 9). *Stock prices prediction using machine learning and Deep Learning Techniques (with python codes)*. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/

Time Series forecasting : Tensorflow Core. TensorFlow. (n.d.). Retrieved from https://www.tensorflow.org/tutorials/structured_data/time_series