CS 4375
PROJECT REPORT

Name of students in your group:
James Hooper (jah171230)
Hritik Panchasara (hhp160130)

Number of free late days used: 2

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

- *Last page of report*

CS 4375 Project Report

James Hooper
UTD Computer Science Major
Dallas, Texas
NETID: jah171230

Hritik Panchasara
UTD Computer Science Major
Dallas, Texas
NETID: hhp160130

***Abstract*— This report explores the topic of Recurrent Neural Networks by going into the LSTM architecture utilized to forecast time-series data for a regression task. The goal given the dataset used is explained side-by-side with the efforts given towards preprocessing the data for the model implementation. A walkthrough of the architecture is given to understand the underlying structure and calculations of the model utilized. The results from the model given the time-series dataset is then shown and analyzed to showcase the best parameters. Ultimately, a guideline and expectation of how to build off of this project in future works to fully explore the concepts of Recurrent Neural Networks is given.**

Keywords—RNN, LSTM, Network, time-series, architecture, model

I. Introduction

Forecasting data is one of the major concerns for society and thus a central focus for Machine Learning. To be able to predict data accurately and efficiently is key to being able to utilize modern data manipulation in such a way that can optimally benefit society (e.g. companies, governments, etc.). One of the most popular types of datasets to be able to predict/model is time-series data which is ordered in order of which an event has occurred temporally. A general focus of machine learning models is being able to transform data in such a way to properly predict the correct output/class. In the case of a time-series dataset the goal would be to take input from a certain sequence of time and then predict the next state to occur. For this project our task is regression based where we will attempt to predict a continuous output. Specifically, we aim to predict the Global Active Power put out by individual households given a certain set of attributes of their respective electric power consumption [1]. This dataset is laid out by averaging out the data over a series of discrete time steps. A key tool to accurately forecast data is the utilization of Recurrent Neural Networks that can manipulate temporally sequenced data. This type of tool will enable the ability for the model to learn from previous states that can then have a say in the future determined outputs. The architecture we used for this project was the LSTM architecture which improves upon the vanilla RNN structure.

II. Technique/Algorithm Theory

First The overarching technique used throughout this project for our forecasting problem was Recurrent Neural Networks, specifically the LSTM architecture. To make this work the first action to do is to set up the data in such a way that it can be fed through the architecture to create the model. Since we only want to predict one output from multiple input attributes the model, we are creating is a many-to-one model [3]. This will enable our regression calculation to use a set of inputs from a discrete time-step to then forecast the desired output for the next predicted time-step. Now in a vanilla RNN the hidden states will keep track of the sequential input thus to have a sense of memory throughout the data manipulation towards the future state output. The main issue with this is that vanilla RNNs are prone to the vanishing gradient problem which will have the change in weights decay to such an insignificant degree that layers further down the sequence will be inconsequentially affected [7]. A simple way to look at this is that throughout time events that occur further, and further back will have a continuously decreasing amount of influence over future outputs [7]. A great way to solve this is through the utilization of LSTM architecture which enables the model to have a greater ‘memory’.

Diving into the LSTM architecture the first focus is the input that enters the LSTM cell. The two inputs of the LSTM are the previous cell state and the compilation of information from the previous hidden state and the current input [2]. From the LSTMs cell input it’s obvious that we must manipulate this data in such a way to output the new/current cell state and the new/current hidden state which will then be fed to the next LSTM cell accompanied by the next time-step input. To accomplish this, we first must focus on the path to achieve the current cell state. Again, with the memory analogies, a calculation must be made to determine what information the cell should keep while sequentially progressing. This occurs through the forget gate which will send the information from the previous hidden state and the current input through the sigmoid function where information that is close to 0 will be ‘forgotten’ while information closer to 1 will be ‘remembered’ [2]. The previous cell state will then be multiplied by this forget-gate output, we’ll call this the new forget gate value. Next is the input gate which will take that same forget-gate concept putting the input values through a sigmoid function that will then be multiplied by the input values that have been sent through a tanh function meant for regulating the data (i.e. taking away the chance for an exploding value) [2]. Let’s call this new value the new input value (rather than the input value which consists of the previous hidden state added to the current input of that time-step). To determine the current/new

cell state the only thing left to do is to add the new forget value and the new input value. Lastly, we must utilize the output gate to yield the new hidden state. First, we send the initial input value through the sigmoid function with a similar forget idea and at the same time send the current cell state through the tanh function with a similar regularization idea [5]. These two values will then be multiplied with one another to give off the new hidden state value that will be sent along to the next LSTM state.

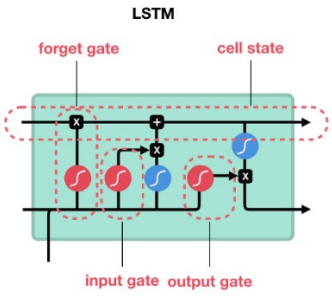

Figure 1: LSTM Cell [2]

There are also some topics to mention that do not need a full dive into to gather the grand idea and how they are fundamental to the model architecture. Before leaving the LSTM cell, there are two core neural network ideas, we should be clear to overview, Forward Propagation and Backward Propagation. Now the previous part of this section just went through the Forward Propagation for this architecture which is utilized to compute the loss value [6]. The key idea of Backward Propagation is the calculations for the gradients themselves, but the difference with traditional neural networks and recurrent neural networks is the need to use a technique called Backpropagation Through Time (BPTT) [3]. To understand BPTT one must understand the unfolded RNN structure which allows us to see each hidden state sequentially. The BPTT algorithm will calculate the gradients by relying on the sequence of the states [3]. The gradients from each time-step will be utilized alongside the derivative of the loss function/calculation to compute the final gradient computation [3]. The last topic that should be mentioned that was utilized for the project model was the use of the dropout method where before computing the gradients a percentage of values are "dropped out". The goal of this method is to prevent overfitting for the final testing results. The final technique that should be mentioned is that for the data inputted into the model the requirement of preprocessing the data by transforming the time-series data into a supervised dataset. This is done mainly by attaching a copy of a shifted version of the desired output, the Global Active Power, as the output column. This shift is done so that the current time-step's attributes are set to predict the next G.A.P. value of the next time step.

III. Results and Analysis

A. Pre-Processing

Before running the data through scaling, splitting, and finally the model we must analyze the raw attribute data. In *Figure 2* we graphed each attribute for every time-step value found within the dataset. There are two major things that are discerned here. One, Voltage is immediately seen to have a very low variance. This type of attribute may improve results if removed given that it will do little to affect the final forecasted result. Two, Global Active Power and Global Intensity have the same overall structure. To really get an absolute idea that they are correlated with one another we can take our sights to *Figure 3* where we generate a Heatmap of the Attribute Correlation resampled over the for the sum. This graph clearly shows a 1:1 correlation between the Global Active Power and Global Intensity. Since our goal here is to predict the Global Active Power we decided to choose the Global Intensity as a possible attribute to remove.

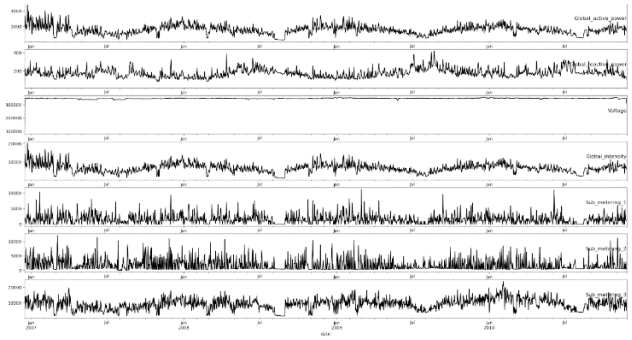

Figure 2: Plotting of Attribute Data

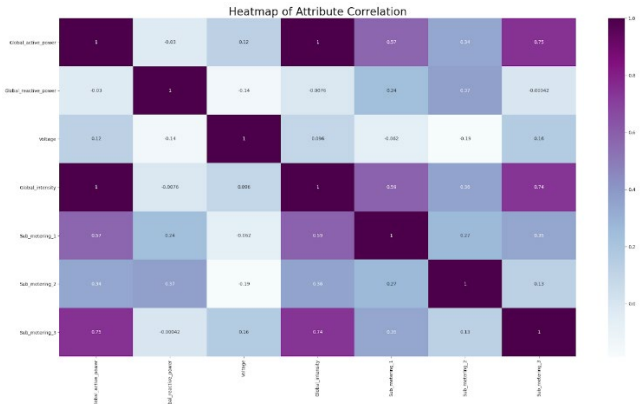

Figure 3: Heatmap of Attribute Correlation

B. Data Results

The following data is given in a tabular format and have some constant parameters used when compiling:

- The loss function used is Mean Squared Error.
- The optimizer used is Adam.
- AA: means All Attributes. This is a Boolean that will determine if we drop the columns we deemed to be unnecessary or to keep all attributes.
- Resample: Ex. (hour/sum) i.e. resample data over the hour for sum.

Experiment Number	*Parameters Chosen*	*Results*

1	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 20 **Train/Test Split**: 90/10 **AA**: False **Resample**: hour/sum **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 100,401	**Epoch 20/20**: loss = .0089, val_loss = .0061 **Train RMSE**: 37.88 **Train R^2**: 0.517 **Test RMSE**: 30.64 **Test R^2**: 0.478
2	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Train/Test Split**: 90/10 **AA**: True **Resample**: hour/sum **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 101,201	**Epoch 20/20**: loss = .0089, val_loss = .0061 **Train RMSE**: 37.929 **Train R^2**: 0.515 **Test RMSE**: 30.54 **Test R^2**: 0.4816
3	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 20 **Train/Test Split**: 80/10 **AA**: False **Resample**: hour/sum **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 100,401	**Epoch 20/20**: loss = .0091, val_loss = .0076 **Train RMSE**: 38.25 **Train R^2**: 0.528 **Test RMSE**: 34.055 **Test R^2**: 0.385
4	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 20 **Train/Test Split**: 90/10 **AA**: False **Resample**: hour/mean **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 100,401	**Epoch 20/20**: loss = .0091, val_loss = .0063 **Train RMSE**: 0.6304 **Train R^2**: 0.5185 **Test RMSE**: 0.5101 **Test R^2**: .4792
5	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 20 **Train/Test Split**: 90/10 **AA**: False **Resample**: day/mean **Train On**: 1296 Samples **Validate On**: 145 Samples **Total Params**: 100,401	**Epoch 20/20**: loss = .0117, val_loss = .0066 **Train RMSE**: 0.3302 **Train R^2**: 0.3766 **Test RMSE**: 0.255 **Test R^2**: 0.4016
6	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 50 **Train/Test Split**: 90/10 **AA**: False **Resample**: hour/mean **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 100,401	**Epoch 50/50**: loss = .0090, val_loss = .0063 **Train RMSE**: 0.6249 **Train R^2**: 0.5267 **Test RMSE**: 0.5097 **Test R^2**: 0.480
7	LSTM RNN: **Layers**: LSTM (100), Dropout(.2), LSTM(80), Dropout(.2), LSTM(80), Dropout(.2), Dense(1) **Epochs**: 20 **Train/Test Split**: 90/10 **AA**: False **Resample**: hour/mean **Train On**: 31129 Samples	**Epoch 50/50**: loss = .0089, val_loss = .0063 **Train RMSE**: 0.6298 **Train R^2**:0.5193 **Test RMSE**: 0.5099 **Test R^2**: 0.4796

	Validate On: 3459 Samples **Total Params**: 151,921	
8	LSTM RNN: **Layers**: LSTM (70), Dropout(.2), LSTM(40), Dropout(.2), Dense(1) **Epochs**: 50 **Train/Test Split**: 90/10 **AA**: False **Resample**: hour/mean **Train On**: 31129 Samples **Validate On**: 3459 Samples **Total Params**: 39,081	**Epoch 50/50**: loss = .0090, val_loss = .0063 **Train RMSE**: 0.6224 **Train R^2**: 0.5305 **Test RMSE**: 0.5089 **Test R^2**:0.4815

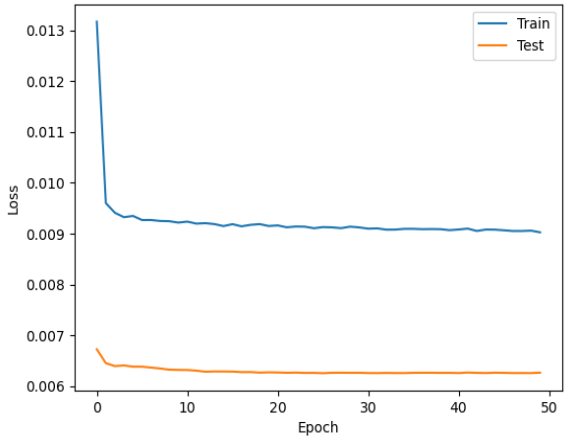

Figure 4: Train Dataset & Validation Dataset, Loss vs Epochs

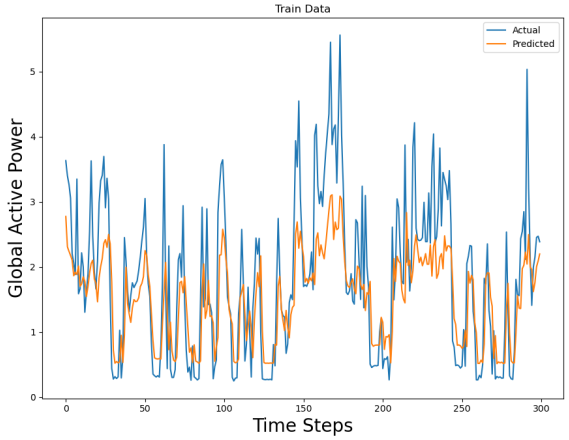

Figure 5: Train Dataset Final Graphical Results over the first 300 time-steps.

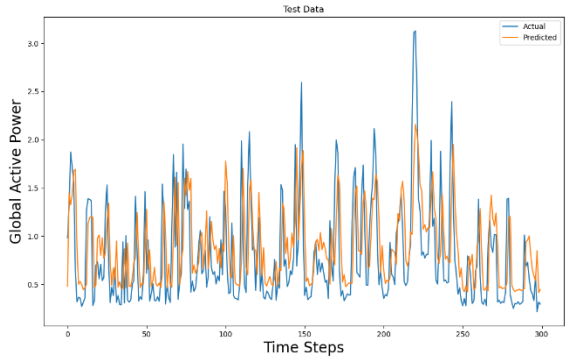

Figure 6: Test Dataset Final Graphical Results over the first 300 time-steps.

C. *Examination of Results*

From the given results we can determine that the best selection of parameters ended up being using two LSTM layers (70, 40 units respectively), with two dropout layers (.2 rate) in between, having a 90/10 train/test split, having a larger epoch size of 50, taking out the attributes that were deemed not needed, and by resampling over the hour for the mean. The results seem to be aligning with the desired output fairly well even though there is considerable improvement to be made. Again there are factors/concerns such as the RMSE vs the R^2 value being a good indicator of the best parameters and the time complexity being a limiter as well. But overall as seen in *Figure 4*, *Figure 5*, and *Figure 6* the last experiment seemed to have the best of both worlds for what we were aiming for. The major trends were that having two layers seemed sufficient, removing the attributes based on linearity and variance aided in decreasing the error, and resampling for the mean significantly decreased the root mean squared error allowing for better model prediction for the regression task. *Figure 4* showcases a direct trend of the loss decreasing over the epochs. *Figure 5* showcases that by running the training data through the trained model there is no case of extreme overfitting. *Figure 6* showcases that the predictive quality fits well. The trends align to exactly where one would hope, except there are a few glaring distortions in scale that could be adjusted with further optimization.

IV. Conclusion and Future Work

Overall, the results for forecasting the Global Active Power for the dataset over individual households ended up relatively accurate. Of course there can probably be further optimizations given better parameters and more epochs to decrease the error more and more. In fact, there could be a better organization of the model itself. But, overall the model created seems to be on the right track accuracy wise. Since this was a simple forecasting problem it allowed a deep understanding of the fundamental concepts of Recurrent Neural Networks and how the propagation of data is calculated. For the future there are a multitude of trajectories that can be sprouted off of this project's core accomplishments. One such trajectory is simply going through another time-series dataset meant for regression and seeing how the RNN model techniques learned here can be utilized in different scenarios. Of course through researching there were other forms of RNN that have been found namely the

inclusion of peephole connections, the GRU cell, and having LSTMs with attention [8]. These sets of techniques/architecture can be gone through in depth with other projects with similar form to this one. Another huge aspect in line with using Machine Learning for sequential data is the task of Natural Language Processing. By using a classification type dataset with predicting certain aspects of sentences such as tone, semantics, and syntax there is a whole other world of RNN topics that can be explored.

ACKNOWLEDGMENTS

None

REFERENCES

[1] UCI Machine Learning Repository, "Individual household electric power consumption Data Set", *Creative Commons Attribution 4.0 International (CCBY4.0).*[Online].Available: https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption. [Accessed: Aug. 1, 2020].

[2] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation", *Towards Data Science,* Sept. 24, 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed: Aug. 2, 2020].

[3] J. Brownlee, "Gentle Introduction to Models for Sequence Prediction with RNNs", *Machine Learning Mastery,* July 17, 2017. [Online]. Available: https://machinelearningmastery.com/models-sequence-prediction-recurrent-neural-networks/. [Accessed: Aug. 2, 2020].

[4] H. Kumar, "Backpropagation Through Time", Feb. 22, 2019. [Online]. Available: https://kharshit.github.io/blog/2019/02/22/backpropagation-through-time. [Accessed: Aug. 4, 2020].

[5] H. Kumar, "Backpropagation Through Time", Feb. 22, 2019. [Online]. Available: https://kharshit.github.io/blog/2019/02/22/backpropagation-through-time. [Accessed: Aug. 4, 2020].

[6] D. Britz, "Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs", *WildML*, Sept. 17, 2015. [Online]. Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. [Accessed: Aug. 4, 2020].

[7] Deep Learning on Medium, "Vanishing Gradients on Recurrent Neural Networks", *MC.AI,* Jun. 26, 2018. [Online]. Available: https://mc.ai/vanishing-gradients-in-recurrent-neural-networks-2/. [Accessed: Aug. 5, 2020].

[8] "5 Types of LSTM Recurrent Neural Networks and What to Do With Them", *Exxact*, Nov. 12, 2019. [Online]. Available: https://blog.exxactcorp.com/5-types-lstm-recurrent-neural-network/. [Accessed: Aug. 4, 2020].