Time Series Prediction of the Western Sports and Recreation Centre Attendance Using Long Short Term Memory

Colin Brown
Department of Computer Science
Western University
London, ON
cbrow388@uwo.ca

***Abstract*—In this paper, I will investigate the usefulness of a long short-term memory (LSTM) model for time series prediction that aims to forecast the attendance of people in the Western Sports and Recreation Centre (WSRC) third floor weight room. Taking as input 5 hours of weight room (WR) attendance, the model can forecast the rest of the day's attendance, allowing for updated forecasts to be made as the day progresses. I will compare this model against two other models. The model is good at predicting days that are close to normal, but days that deviate strongly from the normal cause it to predict that the day will gradually return closer to the normal rather than remain abnormal.**

***Index Terms*—long short term memory, time series analysis, recurrent neural networks, deep learning, prediction algorithms**

I. Introduction

Toward the end of my third year of university, I started regularly going to the gym on campus. I eventually learned that the staff post half-hour updates of the live attendance statistics on Twitter/X (@WesternWeightRm) about the number of people in three different sections of the gym, the weight room being the busiest and the one that concerned me the most. After discovering this helpful data, I initially found it difficult to tell whether "80" in the weight room meant it was busy or not because I had no frame of reference for that number. So, I wrote a script to scrape as much of their tweet data as possible and then graph it.

After I had done that, I ended up with a month and a half of tweet data to graph. I used Excel to sort them by month, then day of the week, then hour, and then averaged all the values in each hour. I kept up with this during my fourth year and accumulated almost 12 full months of gym attendance data, see Fig. 1. A clear pattern could be seen in the data I had collected. I called this my day of the week average (DotWA) model. It has seven day of the week predictions for every month of the year, coming to 84 predictions total. However, I still found this method of checking the busyness insufficient. I could check a few tweets from the Twitter/X page and then compare the numbers to the spreadsheet to see if it was normally busy or abnormally busy, but that did not really help me figure out what it might look like later in the day if things changed or were abnormal, a prediction model is what I wanted.

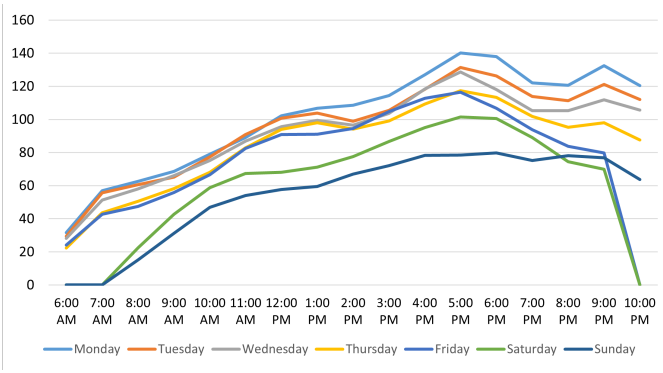

Fig. 1. A portion of the DotWA model. This shows the months September to April combined for the weight room attendance for a given day of the week.

Research objectives and result. I set out to create an efficient time series prediction model. This desire of mine presented me some issues; I would need a lot of data to train a machine learning model, there are many aspects of university life that impact how busy the gym is, and there are many models for time series analyses that could work. To implement the model, I needed to format my data in a way that the model can digest, develop a script to train many models and save the best one, another to test the model's efficacy, and a third to run a user defined input through the model and display the prediction. I compiled lots of data, over 55 000 tweets from a span of more than 10 years, and knowing that recurrent neural networks (RNNs) work well with processing lots of sequential data, I tried one of those for the architecture of my model. I found that a long short-term memory (LSTM) model was great at making the predictions I was looking for with my weight room data. It preforms better than my averaged data as it can forecast based on previous inputs.

Impact of results. This research further proves the effectiveness of LSTM models on time series analysis and prediction. The model could be used by the WSRC to better communicate with students what to expect if they plan to

attend the gym on a given day. It could also be provided directly to students to input any day in the future of their choosing to see what it might be like.

Structure of paper. In the first part of this paper I will discuss how LSTM models work and what makes it a good choice for my use case. After that I will discuss some time series prediction models, their upsides and downsides, and will use one to compare my results with. In the second part of this paper I will outline the objectives of my implementation, how I accomplished them, and discuss problems I encountered. In the third section I will present the results of my implementation, how they compare to other models, and patterns I observed. Finally, I will consider use cases and improvements of my result.

II. Background and Related Work

In this section I will discuss the LSTM architecture further, outlining its benefits, its downsides, and how it works. I will also discuss some other related work to time series forecasting involving weather and the WSRC.

A. LSTM Architecture

Knowing I wanted my model to be able to handle sequential data well, a RNN made sense for me as they are a fundamental linear data structure [1]. But, concerned with the issues of other conventional RNNs, a LSTM model is beneficial for a couple reasons. Short-term memory is meant for saving representations of the latest events and long-term memory is exemplified by gradually shifting weights. LSTM models make the best of both [AAA]. Additionally, in a LSTM model there is no occurrence of the vanishing gradient problem seen in other conventional RNN models [2]. This is the reason the architecture is the basis for many state-of-the-art models in literature [3]. The vanishing gradient problem occurs when gradients become extremely small and vanish as they "flow backwards in time" to the other layers in the model [2]. Gradients propagate back in the model to improve the output of the layers and connect them for sequential analysis. The rate at which they propagate depends on the size of the weights [2]. Another problem for some RNNs is that the gradients could oscillate, changing direction or sign frequently and significantly from layer to layer [2]. LSTM models overcome these errors and can bridge time intervals of more than 1000 steps by using a gradient-based algorithm which implements a constant error flow through the internal states [2].

There are three inputs to a LSTM cell, refer to Fig. 2. The current input, represented by x_t, is the data at the current time step t. It is a vector representing the information the LSTM should process at a specific moment. The previous hidden state, represented by h_{t-1}, is a vector representing the information the model has learned from the previous time steps in the sequence. It is essentially the memory of the past relevant to processing the current input and makes up the short-term memory for the model. The previous cell state, represented by c_{t-1}, is another vector that represents the long-term memory of the cell. Unlike the hidden state which often carries information about the recent past, the cell state is designed to retain information over longer sequences. For the outputs of a cell, the current hidden state, represented by h_t, is the main output of the cell at time step t. It's a vector that contains information about the current input and the influence of the past. It is what make the prediction at the current time step and is also passed to subsequent layers. The current cell state, represented by c_t, is the updated long-term memory of the cell that will get passed to the next cell. It could have some information added or removed from the cell state given by the previous cell [2].

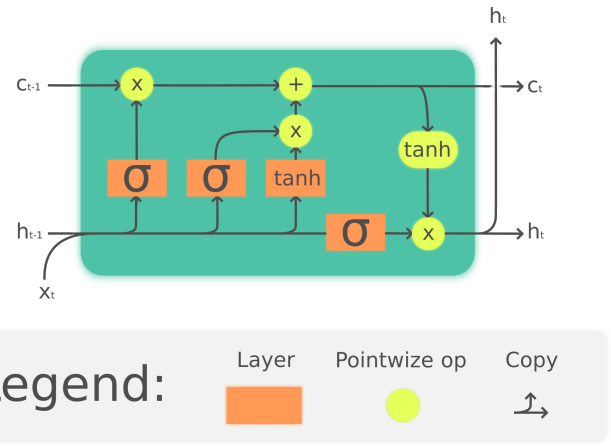

Fig. 2. LSTM cell. This visualization is freely available and is licensed under the CC-BY License, by Guillaume Chevalier. [CCC]

To account for the need of long-term and short-term memory, lots of data is needed for LSTM models. The architecture is "designed to allow the network to learn much longer-range dependencies" [3]. If there is not enough data, then a LSTM model will not be useful, as it is likely to overfit to the training dataset and not be efficient on unseen data. Other kinds of classical methods of models, such as autoregressive integrated moving average (ARIMA), simple exponential smoothing (SES) and moving average (MA), would be more efficient on data sets that are smaller [3]. As my training dataset ended up being comprised of 36 036 tuples, I am not concerned with this shortcoming.

B. Weather Forecasting

Lots of work has gone into the problem of time series forecasting for its many use cases. Weather, economics, stock market analysis, energy consumption, and more are all important applications of time series forecasting that have seen great impact on how they are being accomplished in recent years [4]. As better models are created, more overall data is collected, and overall compute power becomes greater, time series forecasting is only going to become better and more widespread.

Weather forecasts are a time series forecast that everyone uses just about every day. However, most current forecasts found on a weather app are made with physics-based models

that perform vast sums of mathematic calculations on real-world data inputted into supercomputers. Despite this, there are lots of companies working on new machine learning models. GraphCast, developed by Google DeepMind, is one that has seen lots of promise. It has proven to perform on par with, or better than, the best physics-based models 90 percent of the time [5]. In one case, it predicted the path of Hurricane Lee three days before traditional models. Additionally, these machine learning models are much faster than physics-based models, creating a weather forecast in a couple minutes rather than two to three hours [5].

There are some issues with these new forecasting models. They are best at predicting widespread systems rather than localized weather. GraphCast works on 28 square kilometre chunks of the Earth [5]. This limitation reduces its effectiveness at predicting storm and rainfall intensity [6]. As well, because it is trained on historical data, a changing climate may reduce its effectiveness. Rare and never-before-seen events are unlikely to be predicted by a model of this kind. Furthermore, because of the black-box nature of machine learning models, there is no way to tell how the model arrived at a forecast in the same way physics-based models do. There is little way to determine a specific issue and fix it to improve the model. These models also can only produce one forecast without probabilities, something we are very used to with our current forecasts [5].

Weather forecasters currently see these new models as an additional tool to consider when making their forecasts. It is unlikely that that they will replace current methods in the coming decade [5].

C. WSRC Forecasting

A case where time series forecasting was implemented on the WSRC in the past is with Western Gym Monitoring by Robot, also known as, GyMBRo. This was a boosted tree model that used LightGBM, developed by Demetri Pananos in late 2019. It was used to predict the number of people at the gym for each hour of a given day. Twice every hour, the program would post on Twitter/X a graph with a prediction line and plot points of the attendance actually being observed, see Fig. 3. Pananos did not publish a loss calculation for the model, but from my observation, it seems to often do quite well with its predictions before it was shutdown in April of 2023. Some days a very noticeable deviance from the prediction and observation can be seen, especially with summer predictions or days just after holidays. Pananos notes that what improved the model the most in their feature engineering was features for holidays. One for the holiday coming up, one for how many days until the holiday, and another for if it is currently a holiday. It was also noted that weather data was not found to be useful for predictions [7].

III. Methods

In this section I will discuss the objectives for the implementation of my model and how I achieved each.

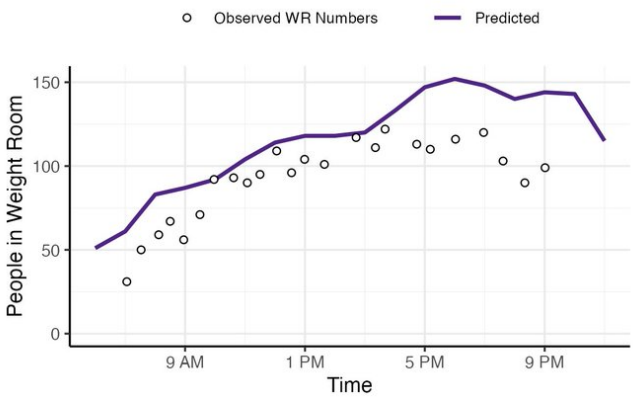

Fig. 3. Prediction made by GyMBRo for April 18, 2023. Purple line is boosted tree prediction, black circles are observed attendance plot points [8].

A. Research Objectives

The objectives I set out for in the implementation of my model are as follows:

- **O1**: Collect lots of data so that the data expensive LSTM model can accurately recognize patterns in the data and better forecast a day's attendance.
- **O2**: Feature engineer the data so that it is readable by the machine for training, validation, and testing purposes and provides the best information for prediction purposes.
- **O3**: Create scripts to train and validate a model, test it, and use it to make a user defined prediction.
- **O4**: Create the best model with my engineered data by fine tuning the hyperparameters that beats my DotWA model.

B. Research Methodology

O1. For the collection of my data, I originally wrote a python script to scrape the data on Twitter/X from @WesternWeightRm. However, due to the API changes made in February 2023, the free tier no longer has access to reading tweets and can only post tweets. To get around this, I used a Python package called Tweety-ns, a reverse engineered front-end API that could scrape the most recent 800 tweets for me. After continuously running that script for some months, I ended up with 5746 tweets from May 2024 to April 2025.

While planning for the project, I reached out to the WSRC and asked if they had any additional data, they got back to me saying they only had the data I already had. Then, while implementing the training script, I discovered the GitHub repository for GyMBRo which contained an SQLite datebase file containing 50 133 more tweets from @WesternWeightRm from January 2014 to April 2023 excluding COVID months March 2020 to August 2022. After combining that with my data, I ended up with 55 879 tweets to use to train my model. I also used this data to improve my DotWA model.

O2. For feature engineering, I first took this dataset and added to it tuples with a zero WR value for times the WSRC is closed, such as the morning, evening, and holidays only if

there was not a tuple for that hour already. Then, I transformed my tuples from (YYYY-MM-DD hh:mm, WR value) format to (year, month, week, day, hour, day of the week, WR value) format where "week" is the week of the year out of 53 and "day" is the day of the month out of 31. All attributes would be an input, a WR value would be the only output. Next, so as not to confuse the model with a year feature, I removed it and combined any tuples with the same date features by averaging the WR outputs together. I did not want there to be multiple tuples with the same date features and different associated WR values. Finally, I had a script group the dataset by their date features and take 80 percent of the dates for the training data. This set had 1337 different days and 36 036 tuples. Then, with the remaining 20 percent, I split the dates evenly into two sets for validation and testing. For each of dates in these datasets I ran a linear interpolation function on them so there was no missing hours of data in case the WSRC did not post data during an hour. This was so the loss function would always have something to calculate for every hour. These datasets each had 167 days in them; 3961 tuples for validation and 4009 tuples for testing. In the end, I was left with 44 006 tuples total for training, validation, and testing.

O3. When creating my scripts, making the training script train.py wasn't too much of an issue. I referred to online literature for guidelines on how to create the model with PyTorch in Python and what I could expect from the framework and the model [9] [10]. The script has five hyperparameters. These are the: number of neurons (HIDDEN_SIZE), number of layers (NUM_LAYERS), batch size (BATCH_SIZE), learning rate (LEARNING_RATE), and sequence length (SEQUENCE_LENGTH). The total number of input features (INPUT_SIZE) was 6.

To ensure numerical stability and improve model convergence, all selected features were scaled to a [0, 1] range. A separate MinMaxScaler was fitted specifically on the 'WR' column of the training data to prevent data leakage from the validation set; this scaler is crucial for inverse-transforming the model's output back to the original 'WR' scale during validation and prediction. The time-series data was transformed into sequences suitable for LSTM input using a sliding window approach. Each input sequence consisted of SEQUENCE_LENGTH consecutive time steps, encompassing all selected features. The corresponding target for each sequence was the scaled 'WR' value at the subsequent time step (t+1). Optimization was performed using the Adam optimizer with a learning rate of LEARNING_RATE. Training utilized mini-batches of size BATCH_SIZE using PyTorch's DataLoader, which also shuffled the training sequences in each epoch to improve generalization. Each training process would have 100 epochs, and I would save the model with the best loss rate on the validation dataset. This model would then be tested with the same loss strategy.

Validation and testing were what I knew would be an issue. For them I used mean squared error (MSE) as my loss function to disproportionately punish worse predictions greater. y_i is the observed WR value at hour i. $\hat{y}_i$ is the predicted WR value at hour i. n is the number of predictions made.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Both the validation and test functions used the same method to calculate a MSE value. Each would first pick a random start hour at least SEQUENCE_LENGTH hours from the first tuple for every date in their respective datasets. Then they would input the previous SEQUENCE_LENGTH hours of tuples into the model, iteratively predict the remaining hours of the day, and calculate the MSE of those predictions and the actual observed data in their datasets. I chose to do this method because in reality, someone wanting a prediction could choose any random start point in the day for their forecast and I wanted the validation and test functions to mirror that. However, because it was random, that meant the MSE value I would get was not a concrete number for each dataset, and could be higher or lower than the true MSE of the model. To remedy this, I had the validation function run 5 times and the test function run 50 times, each time averaging the sum of the outputs. Along with the fact that the validation and test datasets were quite large, I believe that my resulting MSE would be roughly the true MSE according to the central limit theorem.

O4. I trained the best possible model by isolating hyperparameters, changing them individually, and observing which would lower the test MSE the most. Then, to calculate the effectiveness of my DotWA model, I computed the MSE of the DotWA model with the same test data I used to test the LSTM model. For this, I had the MSE function test only on predictions that were not zero so that the model did not unfairly get lots correct predictions of zero WR attendance, values that I entered manually in both sets, unfairly bringing down its MSE.

IV. Results

A. System requirements, architecture, and implementation

For all training, validation, testing, and predicting, I used a ASUS ROG Zephyrus Laptop with 16GB of DDR4 SDRAM, an Intel Core i7-11800H 8-core processor, and a NVIDIA GeForce RTX 3060 Laptop GPU running CUDA version 12.6.85. Code was run in Visual Studio Code. See the attached zip folder for the implementation and model. The PyTorch 2.6.0 framework was used for data preprocessing, model instantiation, training loop, validation, and model saving. Scikit-learn 1.6.1 was used for feature scaling.

B. Testing and Results

Testing different models revealed that a model with 64 neurons, 3 layers, a batch size of 128, a learning rate of 0.001, and a sequence length of 3 resulted in the best model. This model has a MSE of 541.6 and a root mean squared error (RMSE) of 23.3. The standard deviation of my WR values is 47.2, a noticeable difference. As seen in Fig. 4, the green model in the middle is the best model. To the left

and right of it, raising and lowering selected hyperparameters in grey increased the Test MSE. The RMSE shows the average difference between the prediction, $\hat{y}_i$ and the true observation y_i.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Neurons	64	64	64	32	64	64	64	128	64	64	64
Layers	3	5	3	3	2	3	4	3	3	3	3
Batch size	128	64	128	128	128	128	128	128	128	256	128
Learning rate	5E-04	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.005
Seq length	5	5	3	5	5	5	5	5	7	5	5
Validation MSE	620.8	631.6	662.9	620.9	622.2	611.8	635.5	620.8	640.4	599.3	632.5
Test MSE	556.8	559.1	547.3	562.9	551.9	541.6	646.3	574.6	572.6	589.4	696.1
Test RMSE	23.6	23.6	23.4	23.7	23.5	23.3	25.4	24.0	23.9	24.3	26.4

Fig. 4. Chart showing the validation and test MSEs and test RMSE of each best model made from the corresponding hyperparameters. Hyperparameters in green produced the best model. Grey hyperparameters differ from the best model. Green MSE is lowest, red is greatest.

A surprising result I observed was that lowering the sequence length to three did not significantly increase the test MSE. However, as this model had the highest validation MSE value, I did not pursue it for further use in my predictions.

Initially I trained my model on the 5746 tuples that I had collected before I got the additional 50 133 tuples. With that smaller dataset, I also observed that the same hyperparameters produced the best model, but it had a MSE of 991.3. The other models of differing hyperparameters had MSE values ranging around 1050 to 1150. This result further tells us that LSTM models benefit greatly from larger datasets, and by increasing my dataset, I reduced my best MSE by 45.4

When comparing the LSTM model to the DotWA model, I found that the LSTM model performs quite better on average. Calculating the MSE of the DotWA model with the test data resulted in a MSE of 752.2, an increase in MSE of 39.1% when compared to the best LSTM model.

I also manually compared a few predictions from GyMBRo to predictions of other models. Seen in Fig. 5 are predictions for April 18, 2023. I chose this date because Tuesday April 18 is a point not in my training or validation sets. My model's predictions are marked with the 'X' points. The black line is a full day prediction made at the start of the day and the red line is a prediction made at 11am with 5 hours of sequential WR data input. It can clearly be seen that as the model is fed information about the progressing day, the prediction moves closer to the green target observed values. My forecast at 11am has the best MSE of the 4 predictions made on the day. The next lowest MSE was the DotWA forecast, and GyMBRo's forecast was the worst with a 414% greater MSE than the 11am LSTM prediction.

I consistently found that my LSTM model performed better than the GyMBRo model on average, especially when my model makes its prediction later in the morning when observations about the attendance have been made. However,

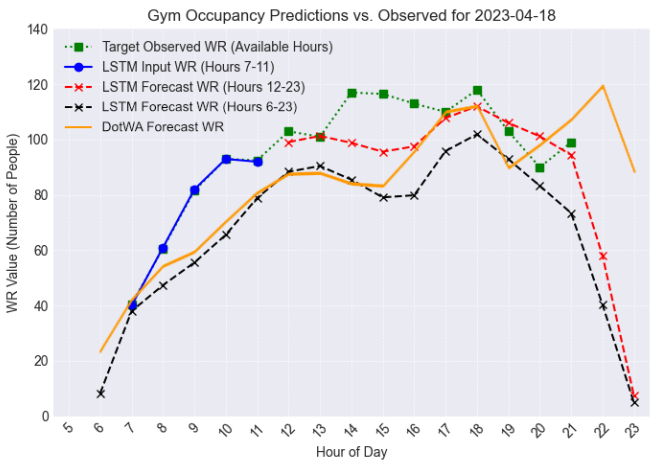

Fig. 5. Graph showing four predictions made for Tuesday April 18, 2023. In green/blue is the target observed values. In black is a LSTM prediction made at 5am with only 0 valued input data (MSE: 456.2). In red is an updated LSTM prediction made at 11am with five hours of observed input data from 7-11am in blue (MSE: 119.5). In orange is an average of Tuesdays in April from the DotWA model (MSE: 294.3). Not graphed: GyMBRo (MSE: 615.2) see Fig. 3.

because GyMBRo did not publish its testing results, I cannot definitively confirm this.

Another observation I made is that if a day is abnormally busy or quiet, the LSTM model tends to gravitate towards an average later in the day instead of remaining at the same abnormal difference of the input. This phenomenon can be seen in Fig. 6 for March 30, 2023, another date not in my training or validation datasets. The green observation line is significantly above the orange DotWA prediction, showing abnormality, however, when I input five abnormal WR inputs, the model forecasts far lower than what occurs. The model does correctly increase its prediction above the 5am prediction for most of the day, but they both eventually trend toward a similar WR value close to the DotWA line. For this day's prediction GyMBRo does very well, perhaps due to its extra features as March 30th is near to the end of the term.

V. Conclusions and Future Work

A. Conclusion

Overall, I found that the LSTM model created is the preferred model for forecasting the number of people in the weight room at the WSRC. I was able to successfully collect lots of data points and feature engineer them so that the model performs noticeably better than the DotWA model in testing and in general use. I observed that the LSTM model tended to converge on a similar average despite its input. This is likely due to its short-term memory gradient becoming diminished in later cells as the prediction continues.

I confirmed that LSTM models perform better with more data, as is discussed in literature [3]. With only a year's worth of data, the best trained LSTM model was not able to outperform the rudimentary DotWA model.

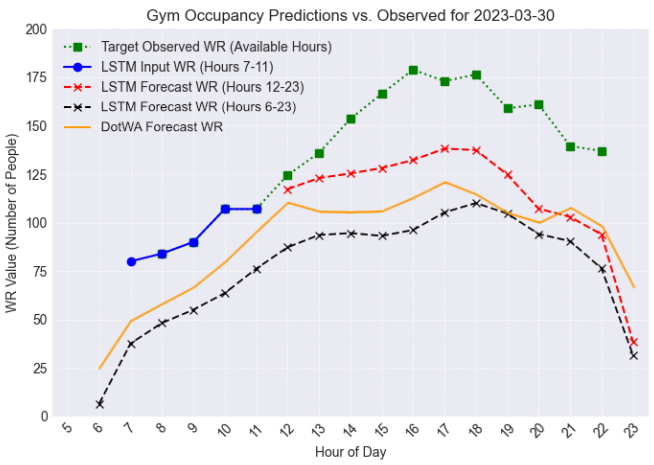

Fig. 6. Graph showing four predictions made for Thursday March 30, 2023. In green/blue is the target observed values. In black is a LSTM prediction made at 5am with only 0 valued input data (MSE: 3112.8). In red is an updated LSTM prediction made at 11am with five hours of observed input data from 7-11am in blue (MSE: 1331.2MSE). In orange is an average of Thursdays in March from the DotWA model (MSE: 1921.5). Not graphed: GyMBRo (MSE: 394.9) [11].

What is likely the best solution for forecasting the WR attendance for students is a combination of an LSTM model and another model, such as GyMBRo's boosted tree model. A prediction could be made with another model at the start of the day when there is no observed attendance yet, as my start of day predictions are weak. Then, as attendance is observed, a transition to the LSTM model can be made for more accurate predictions for the remainder of the day.

B. Future Work

As discussed by Pananos in his implementation of WRSC attendance time series forecasting, GyMBRo, engineering features for holidays turned out to be the most important features improving the model [7]. Implementing a similar tactic in an LSTM model may also produce great improvements in efficacy and is something that could be investigated.

Additionally, overfitting could be an issue in my model. Where there are days that have abnormally high or low attendance rates, the model tends to gravitate towards a similar average later in the day instead of remaining at the same abnormal difference of the input. This could potentially be addressed by the implementation of holiday features or perhaps other features such as indicator variables for anomalous days.

Improvement in the testing and validation methods could also be made. Whereas my model uses interpolated data for MSE calculations, a method could be created to instead calculate MSE exclusively on observed data, skipping unobserved hours. However, that does lead to the case where some hours may not be tested as often. Around closing time, observations seem to become less common by the WSRC staff. If observation and recording WR data for every hour was addressed, this issue would be addressed sufficiently.

REFERENCES

[1] G. Chevalier, "LARNN: Linear Attention Recurrent Neural Network," Aug. 17, 2018, arXiv: arXiv:1808.05578. doi: 10.48550/arXiv.1808.05578.

[2] S. Hocheriter and J. Shumidhuber, "Long Short-Term Memory," Neural Computation, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[3] A. R. S. Parmezan, V. M. A. Souza, and G. E. A. P. A. Batista, "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model," Information Sciences, vol. 484, pp. 302–337, May 2019, doi: 10.1016/j.ins.2019.01.076.

[4] M. Kolambe, "Forecasting the Future: A Comprehensive Review of Time Series Prediction Techniques," Journal of Electrical Systems, Apr. 2024, doi: 10.52783/jes.1478.

[5] L. Leffer, "AI Weather Forecasting Can't Replace Humans–Yet," Scientific American. Accessed: Apr. 07, 2025. [Online]. Available: https://www.scientificamerican.com/article/ai-weather-forecasting-cant-replace-humans-yet/

[6] R. Schumacher and A. Hill, "AI and machine learning are improving weather forecasts, but they won't replace human experts," Colorado State Univerity Source. Accessed: Apr. 07, 2025. [Online]. Available: https://source.colostate.edu/ai-and-machine-learning-are-improving-weather-forecasts-but-they-wont-replace-human-experts/

[7] D. Pananos, GyMBRo. (Apr. 29, 2023). R. Accessed: Apr. 07, 2025. [Online]. Available: https://github.com/Dpananos/GyMBRo

[8] Western GyMBRo [@WesternGymBot], "Predictions for 2023-04-18," Twitter. Accessed: Apr. 07, 2025. [Online]. Available: https://x.com/WesternGymBot/status/1648470972880801796

[9] A. Tam, "LSTM for Time Series Prediction in PyTorch," MachineLearningMastery.com. Accessed: Apr. 07, 2025. [Online]. Available: https://www.machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/

[10] A. Anis, "How to apply LSTM using PyTorch," Intel Tiber AI Studio. Accessed: Apr. 07, 2025. [Online]. Available: https://cnvrg.io/pytorch-lstm/

[11] Western GyMBRo [@WesternGymBot], "Predictions for 2023-03-30," Twitter. Accessed: Apr. 07, 2025. [Online]. Available: https://x.com/WesternGymBot/status/1641576423377846275