Trisa Nguyen

October 17, 2022

ADTA 5560

Final Project

**PART I: A Time Series Data Set**

This time series dataset from Kaggle contains climate data collected in the city of Delhi, India (Figure 1). The time period spans from 1/01/2013 to 4/24/2017. The dataset contains mainly numeric data and a column for date data. The columns include: the date (YYYY, MM, DD), mean temperature (averaged out from multiple 3 hour intervals per day), humidity, windspeed, and mean pressure. The dataset is divided into a training sub-set and a testing sub-set. The training subset spans from 1/01/2013 to 12/31/2016 for a total of 1462 rows. The testing sub-set spans from 1/01/2017 to 4/24/217 for a total of 114 rows. The dataset can be downloaded from this link [here](https://www.kaggle.com/code/advaypatil/dehli-climate-time-series-analysis/data).

Timeline

Description automatically generated

Figure 1. Delhi, India climate dataset preview (Kaggle, 2022).

**PART II: RNN: Simple RNN with Sine Wave Data**

**Model Architecture:**

A

A

UNROLLED SIMPLE RNN

FEED FORWARD NEURAL NETWORK

Figure 2. Diagram of Simple RNN model architecture.

**Results:**

**Training:**

During the training phase the sequential model was trained with 125 neurons with 1 input feature (sine data) using 50 historical data points to predict the next sequence. The model was then compiled with an “adam” optimizer and loss was measured in mean squared error. Then the model was fit using fit\_generator for 5 epochs. Figure 3 displays the epoch history with the loss at each epoch. Figure 4 plots the loss. It can be seen that the loss is minimized after each epoch, which is what we want to see with a neural network.

Chart

Description automatically generated

Figure 3. Epoch callback history for Simple RNN model.

Chart, line chart

Description automatically generated

Figure 4. Plot of loss data.

**Evaluation:**

To evaluate the model on test data, the model’s predictions (orange) were plotted with the true values (blue) (Figure 5). To gauge how well the model performed we can observe how the predicted sine wave is plotted with the true sine wave (Figure 5). The gap between the two waves represents model error. If the predicted is close to the true values that we can say that the model performed well, due to low error. If there is a large gap between the two waves then we could say that the model did not perform as well, due to higher error. For this model, I would say that the model performed okay as the gap is not too large, however it is apparent that the model under predicted most of the time.

Chart, line chart

Description automatically generated

Figure 5. Visualization of predicted sine wave with true sine wave.

**PART III: RNN: LSTM Neural Network**

**Question 3.1:**

**Explain the vanishing gradient problem and the exploding gradient problem.**

The vanishing gradient problem is a problem that occurs during back propagation in a recurrent neural network. Gradients are values used to update weights that are applied to a neural network. With the vanishing gradient problem, the gradient shrinks as it back propagates each time through time step t. The gradient value shrinks extremely small that it stops contributing to the network’s learning and therefore, the network stops learning all together. This is problematic if the network is large with many layers and the sequence is long. RNNs can “forget” what it has learned in the earlier time steps of long time sequences, thus having “short term memory”. In contrast to the vanishing gradient problem, the exploding gradient problem occurs when the gradient increases exponentially as it back propagates through each time step until it eventually explodes, meaning the network becomes unstable and unable to learn.

Why does this occur? Certain activation functions, like the sigmoid function, have a large difference between the variance of their inputs and outputs. So they shrink and transform a larger input space into a smaller output space that lies between the range of [0,1]. With larger inputs, it saturates at 0 or 1 with a derivative very close to 0. Thus, when back propagation occurs, the network till has no gradient to propagate back into the network, and whatever little residual gradient exists keeps on diluting as the algorithm progresses towards the later layers. So, this leaves nothing for the earlier layers. Similarly, in some cases the initial weights assigned to the network generate some large loss and now the gradients can accumulate during back propagation. However, the result will be very large gradients which eventually results in large updates to the network weights and leads to an unstable network that can no longer learn.

**Question 3.2:**

**Discuss the limitations of the Simple RNN neural network.**

The main limitation of simple RNNs is that it suffers from “short term memory” via the vanishing/exploding gradient problem. RNNs can “forget” what it has learned in the earlier time steps of long-time sequences, thus having “short term memory”. The vanishing/exploding gradient problem occurs during back propagation in an RNN. Gradients are values used to update weights that are applied to a neural network. With the vanishing gradient problem, the gradient shrinks as it back propagates each time through time step t. The gradient value shrinks extremely small that it stops contributing to the network’s learning and therefore, the network stops learning all together. For exploding gradients, the gradient increases so quickly that the network is unstable and can no longer contribute to network learning.

**Question 3.3:**

**Explain how the LSTM neural network can provide powerful solutions to both gradient problems: (Vanishing and Exploding) and address the limitations of the SimpleRNN neural network.**

LSTM neural networks have two key features that address the limitations of the vanishing/exploding gradient problem of Simple RNNs: 1) cell state and 2) gates. In addition to a hidden state, the cell state is also used to process information in an LSTM cell. The Cell state can use information carried in from a previous Cell state without any reduction, as long as the information is still relevant and significant. This occurs because there are no weights involved with the Cell state and can stay as it is through multiple layers or time steps. The cell state can also be considered as in information transporting channel where information is transferred from the earliest time steps all the way to the end of a sequence. LSTM gates regulates the flow of information that is moving along the cell state. These two key features mitigate the problem of short-term memory.

The cell state of a LSTM neural network has two different points of view. The first is “snapshot of the state of a cell”. In this point of view, the Cell (C) state can “memorize” the information in a cell at each time step, regardless of how early the information is in the sequence. This means that the value of the cell state in a cell at any timestep can accurately reflect or described the state of the cell regardless of how early the cell is in the sequence. The second is “flow in a channel”. With this point of view information can flow freely, not being reduced throughout the channel from the start until the end of the sequence as long as the information if significant and relevant (Figure 2). This occurs because there are no weights involved with the Cell state and can stay as it is through multiple layers or time steps. Since information is flowing freely moving along the cell state, gates are present to regulate that flow.

Shape, rectangle

Description automatically generated

Figure 6. LSTM: Cell state Snapshot of the state of a cell

Diagram

Description automatically generated

Figure 7. LSTM: Cell state flow in a channel.

There are three gates in an LSTM neural network: Forget, Input, and Output. The Forget gate decides which information of the previous cell state can be removed or kept during training. The input gate decides what information from the inputs are useful and should be added or saved into the new current cell state. The last gate is the output gate. This gate computes and extracts new current cell state Ct at the time step t. Besides the tanh function, each cell also contains a sigmoid activation function, and it helps the network update or “forget” the data during the training phase. Due to how these gates function together, learning/forgetting information, they can be viewed as a neural network to that can learn to keep relevant information or to “forget” or dispose of irrelevant information. Thus, by being able to preserve the gradient and regulate the data that flows into the cell, LSTM RNNs can address the issue of short-term memory (vanishing and exploding gradient problem) seen with Simple RNNs.

Diagram, schematic

Description automatically generated

Figure 8. Example of the forget gate in a LSTM cell.

Diagram

Description automatically generated

Figure 9. LSTM: Cell state flow in a channel.

Diagram

Description automatically generated

Figure 10. Example of input gate in a LSTM cell.

Diagram

Description automatically generated

Figure 11. Example of output gate in an LSTM cell.

**PART IV: RNN: LSTM with Time-Series Data**

**Model Architecture:**

This LSTM RNN model is composed of 6 layers: 3 LSTM, 2 drop out, and 1 fully connected output layer. There are 50 neurons in each LSTM cell and the drop out percentage is 20%. Figure 12 visualizes the architecture of the model as well as the LSTM cell. For this model, t represents time instance anywhere between the 1st neuron to 50th neuron.

LSTM (50 Neurons)

DROP OUT 20%

LSTM (50 Neurons)

DROP OUT 20%

LSTM (50 Neurons)

DENSE

INPUT

Inside the LSTM cell

x

x

x

+

tanh

σ

σ

σ

tanh

ht-1

ht

xt

Ct-1

ht

Ct

Figure 12. Diagram of RNN LSTM architecture.

**Summary of Core Parameters:**

|  |  |
| --- | --- |
| * 1. **Percentage of data for testing:** | * 1. 10% |
| * 1. **Number of layers in LSTM:** | * 1. 6 |
| * 1. **Number of neurons in LSTM Cell:** | * 1. 50 |
| * 1. **Number of drop out layer(s):** | * 1. 2 |
| **Percentage of drop out:** | * 1. 20% |
| * 1. **Length of input sequence:** | * 1. 1315 |
| * 1. **Batch size for training:** | * 1. 32 |
| * 1. **Number of epochs for training:** | * 1. 100 |
| * 1. **Optimizer:** | * 1. adam |
| * 1. **Loss:** | * 1. mse |

Table 1. Summary of parameters for LSTM RNN.

**Results - Building, training, testing, re-train, forecast**

The initial LSTM RNN model is composed of 6 layers: 3 LSTM, 2 drop out, and 1 fully connected output layer (Figure 13). There are 50 neurons in each LSTM cell and the drop out percentage is 20%. The length of the input is 1315 data points. The model was trained with a batch size of 32 with 100 epochs. 10% of the dataset was reserved for testing. The summary of core parameters can be found above. Figure 12 visualizes the architecture of the model as well as the LSTM cell. For this model, t represents time instance anywhere between the 1st neuron to 50th neuron. It should also be noted that there is only 1 feature for this model, which is the mean temperature.

Table

Description automatically generated

Figure 13. Model summary after model building and compiling.

The model was trained using the TimeseriesGenerator which produces time series batches. The input and output for the TimeseriesGenerator is the normalized training data. The length is 60 historical points and the batch size was 32. The TimeseriesGenerator was then fit to the model for 100 epochs. In Figure 14, it can be seen that the model minimizes loss early on in training and continues to fluctuate at very tiny intervals at each epoch.

Chart, histogram

Description automatically generated

**Figure 14.** Visualization of loss data from initial LSTM RNN model.

The model was then tested to generate predictions on mean temperature using a batch size of 1 and another iteration of the TimeseriesGenerator, but with the test data this time. The length was held at 60 historical points. This produced 146 rows of predictions which was then plotted with 1461 data points of the true values (Figure 15). The predictions seem to match the trend of the true values. To confirm how well the model performed, another plot was created to visualize the predictions and compare them to the actual mean temperature (Figure 16).

A picture containing chart

Description automatically generated

Figure 15. Plot of true values of mean temperature with predicted mean temperatures.

A picture containing graphical user interface

Description automatically generated

Figure 16. Plot comparison of predictions against actual mean temperature.

Before forecasting, we need to re-train the model. Another TimeseriesGenerator was created using a normalized version of the complete dataset for the input and output. The length was 60 historical points, the batch size was 32, and the model was fit to 100 epochs. Now for the actual forecast- the time period was 114 business days (1/01/2017 – 4/24/2017). Figure 17 visualizes the forecast which shows a steep decrease in mean temperature and then a sharp increase. To compare how well the model forecasted, another plot was created to show the true mean temperatures for the same 114 business days plotted with the prediction forecast (Figure 18). In Figure 18, we can see that the model was able to forecast the trend of the temperature, however it failed to correctly forecast the daily mean temperature.

Chart

Description automatically generated

Figure 17. Plot of forecast of mean temperature in Delhi from 1/01/2017 to 4/24/17.

Chart

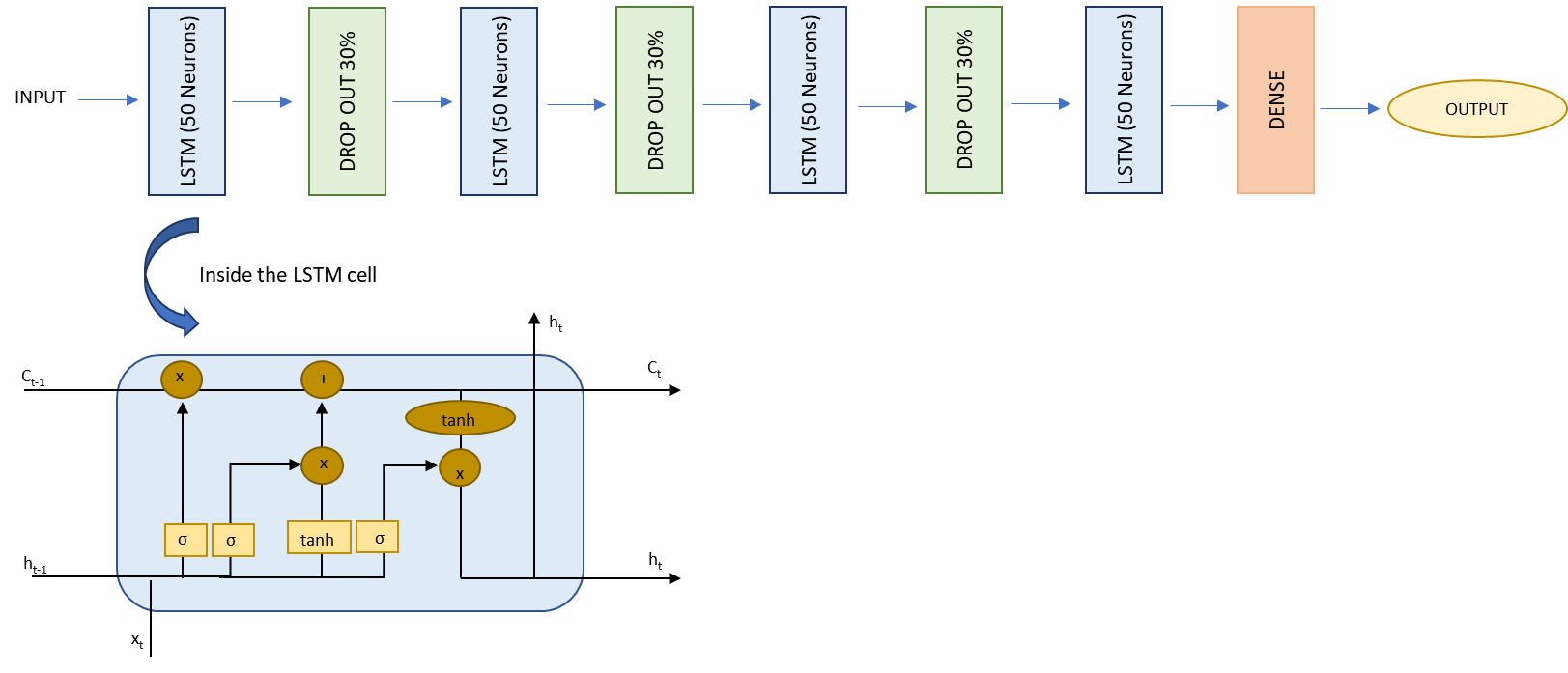
Description automatically generated

Figure 18. Plot of forecast compared with actual mean temperature.

**PART V: Redesign with Neural Network**

**Redesign model architecture:**

The redesign LSTM RNN model is composed of 8 layers: 4 LSTM, 3 drop out, and 1 fully connected output layer (Table 2). There are 50 neurons in each LSTM cell and the drop out percentage is 30%. The figure below visualizes the architecture of the model as well as the LSTM cell. For this model, t represents time instance anywhere between the 1st neuron to 50th neuron.



**Summary of Updated Core Parameters:**

|  |  |
| --- | --- |
| * 1. **Percentage of data for testing:** | * 1. 10% |
| * 1. **Number of layers:** | * 1. 8 |
| * 1. **Number of neurons in LSTM Cell:** | 50 |
| * 1. **Number of drop out layer(s):** | * 1. 3 |
| **Percentage of drop out:** | * 1. 30% |
| * 1. **Length of input sequence:** | * 1. 1315 |
| * 1. **Batch size for training:** | * 1. 32 |
| * 1. **Number of epochs for training:** | * 1. 50 |
| * 1. **Optimizer:** | * 1. adam |
| * 1. **Loss:** | * 1. mse |

Table 2. Summary of parameters for redesigned LSTM RNN.

**Results - Build, train, test, forecast**

To possibly improve model performance, I tuned the following parameters for the redesign. 1) Added 1 additional LSTM layer to add another level of abstraction of input observations over time. 2) Increased the dropout percentage to 30% and added 1 additional drop out later to further reduce overfitting during training. 3) Decreased epochs to 50 to reduce run time. All other parameters were held constant. A summary of all core parameters for the redesigned model can be found in Table 2.

The model was trained using the TimeseriesGenerator package which produces time series batches. The input and output for the TimeseriesGenerator is the normalized training data. The length is 60 historical points and the batch size was 64. The TimeseriesGenerator was then fit to the model for 50 epochs. In Figure 19, it can be seen that the model minimizes loss early on in training and continues to fluctuates up and down at very tiny intervals at each epoch.

Chart, histogram

Description automatically generated

Figure 19. Visualization of loss data from redesigned LSTM RNN model.

The model was then tested to generate predictions on mean temperature using a batch size of 1 and another iteration of the TimeseriesGenerator, but with the test data this time. The length was held at 60 historical points. This produced 146 rows of predictions which was then plotted with 1461 data points of the true values. To confirm how well the model performed, a plot was created to visualize the predictions and compare them to the actual mean temperature (Figure 20).

A picture containing graphical user interface

Description automatically generated

Figure 20. Plot comparison of predictions against actual mean temperature in redesigned RNN LSTM.

During re-training the model, another TimeseriesGenerator was created using a normalized version of the complete dataset for the input and output. The length was 60 historical points, the batch size was 64, and the model was fit to 50 epochs. The forecast was kept constant at 114 business days (1/01/2017 – 4/24/2017). To compare how well the model forecasted, a plot was created to show the true mean temperatures for the same 114 business days (Figure 21). In Figure 21, we can see that there was an improvement in forecasting the trend of the temperature, however it still failed to correctly forecast the daily mean temperature.

Chart, line chart

Description automatically generated

Figure 21. Plot of redesigned model’s forecast compared with actual mean temperature.

**PART VI: Compare Network Performance**

If we compare the original LSTM RNN model with the redesign, I will say that the redesign performed marginally better than the original in regards to forecasting. To re-iterate, the original model was a 6 layer RNN LSTM model with 50 neurons in each cell. There were 2 drop out layers, each at 20%. The batch size was 32 for training at 50 epochs. A table summarizing the core parameters for the models in PART IV and V can be viewed in Table 3. During training, both models performed equally (Figure 16, 20). However, if we look at the final time series forecasting plot, the redesigned model was able to predict the trend marginally better than the original and it’s noted that the trend was tailing up indicating there was more successful abstraction of input data in the redesign model (Figure 22). However, the re-design still failed at predicting the daily mean temperature. Perhaps there is too much variability in the data for the model to process or, alternatively, perhaps a more robust dataset is needed so the model can train over a longer time span instead of 4 years.

Chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 22. Plot of original (left) and redesigned (right) LSTM RNN model’s forecast compared with actual mean temperature.

|  |  |  |
| --- | --- | --- |
| * 1. **PARAMETERS** | * 1. **ORIGINAL** | * 1. **REDESIGN** |
| * 1. **Percentage of data for testing:** | * 1. 10% | * 1. 10% |
| * 1. **Number of layers in LSTM:** | * 1. 6 | * 1. 8 |
| * 1. **Number of neurons in LSTM Cell:** | * 1. 50 | * 1. 50 |
| * 1. **Number of drop out layers:** | * 1. 2 | * 1. 3 |
| * 1. **Percentage of drop out:** | * 1. 20% | * 1. 30% |
| * 1. **Length of input sequence:** | * 1. 1315 | * 1. 1315 |
| * 1. **Batch size for training:** | * 1. 32 | * 1. 32 |
| * 1. **Number of epochs for training:** | * 1. 100 | * 1. 50 |
| * 1. **Optimizer:** | * 1. adam | * 1. adam |
| * 1. **Loss:** | * 1. mse | * 1. mse |

Table 3. Summary of core parameters of original and redesigned LSTM RNN.

**PART VII: Project Report**

This final project covers simple recurrent neural networks and long short term memory recurrent neural networks. We briefly discuss how simple RNNs function and build, train, and test a simple RNN model with sine wave data. We then discuss common pitfalls with simple RNNs and then the solution to those pitfalls, LSTM RNNs. We briefly discuss what LSTM RNNs are and proceed to build, train, and test a model LSTM RNN model to then use it in a time series forecasting model. Lastly, we will discuss what was learned from the models and what was experienced while working with them.

PART II covers simple RNNs. A simple RNN model was built, trained, and tested using sine wave data. The model was built and trained with 125 neurons with 1 input feature (sine data) using 50 historical data points to predict the next sequence. The model was then compiled with an “adam” optimizer and loss was measured in mean squared error. Then the model was fit using fit\_generator for 5 epochs. To evaluate the model on test data, the model’s predictions (orange) were plotted with the true values (blue) (Figure 5). To gauge how well the model performed we can observe how the predicted sine wave is plotted with the true sine wave (Figure 5). The gap between the two waves represents model error. If the predicted is close to the true values that we can say that the model performed well, due to low error. If there is a large gap between the two waves then we could say that the model did not perform as well, due to higher error. It was concluded that the model performed adequately as the gap was not too large, however it is apparent that the model under predicted most of the time.

PART III of project covers concepts regarding the limitations of simple RNNs and how LSTM RNNs address those limitations. Those limitations include the vanishing gradient problem and the exploding gradient problem. We then discussed how cell states and cell gates of LSTM RNN provide powerful solutions to both gradient problems and address the limitations of simple RNNs.

PART IV involves building a LSTM RNN model for time series forecasting. The dataset used was from PART I of this project that had climate data from Delhi, India from 1/01/2013 to 4/24/2017. The goal was to predict daily mean temperature. The model was a 6 layer LSTM model with 3 LSTM layers each having 50 neurons, 2 drop out layers each configured at 30%, and 1 fully connected dense layer. The model was trained with a batch size of 32 and for 100 epochs. A plot for model loss was generated and it was seen that the model minimized loss early in the training process. Next the model was tested using 10% of the whole dataset. Visualization plots were generated after training to gauge how well the model was able to predict daily mean temperature. When compared to the actual mean temperature, it was concluded that the model adequately. Lastly, the LSTM model was retrained to forecast the daily mean temperatures for the next 114 periods (1/01/2017 – 4/24/2017). It was concluded that the model was only able to forecast the trend and failed to forecast daily mean temperature accurately.

PART V was to propose improvements for the model created in PART IV. The following was implemented to possibly improve the model. 1) Added 1 additional LSTM layer to add another level of abstraction of input observations over time. 2) Added 1 additional dropout layer 3) Increased percent of all drop out layers to 30% to further reduce overfitting during training. 4) Decreased epochs to 50 to reduce run time. All other parameters were held constant. A summary of all core parameters for the redesigned model can be found in Table 2.

In this last part of the project, we will discuss what was experienced and learned while working with the models. While LSTM are powerful models that can address simple RNN’s pitfalls with vanishing/exploding gradient problem, it was difficult to produce a LSTM model that would perform adequately on the time series data I had. Furthermore, it was difficult to reproduce the same results even if the same parameters were used.

To summarize, this project covered Simple RNNs and LSTM RNNs. We discussed the architectures of both types of RNNs, the limitations of simple RNNs, how LSTM RNNs can address those limitations, built, trained, and tested both types of RNNs. Furthermore, we used the dataset from PART I of the project to attempt to create a time series forecast using the LSTM model and its redesign to forecast daily mean temperatures in Delhi, India. Though the forecasts were not successful in regard to predicting daily temperatures, we observed that parameter tuning is critical in regard to model performance and efficiency.

**PART VIII: Final Presentation Videos: YouTube Links**

[Final Project Presentation Video Trisa Nguyen](https://youtu.be/CrMjuQX1o70)