**ADTA 5560 Final Project**

Part I: A Time-series Data Sets

The study of sunspots and their effects on the Earth’s climate has been a heated topic in recent news. Observing these solar phenomena can highlight changes in the sun’s magnetic field and better understand coronal mass ejections. Astronomical researchers can gather insights from complex interactions between the sun and our planet to predict geomagnetic storms. The dataset provided came from the WDC-SILSO, Royal Observatory of Belgium, Brussels. Which focuses on Sunspots daily estimates from the period of 1818 – 2022.

Text

Description automatically generated with medium confidence

The variables consist of numeric and categorical data types in the Sunspot time-series datasets. In addition, the 748,759 datapoints had no inconsistence in its tabular format, classifying the dataset as structured data. Understanding the history of solar activity in a cyclical manner can help researchers predict disruptions of satellite communications, navigation systems and power grids.

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

This section will overview steps in creating Simple RNN with Sine Wave Data for educational purposes. Displaying RNN’s architecture can help the reader understand the basic concepts of neural networks and how it is applied to time-series data. Types of layers, activations functions and number of neurons can better viewed when applied to complex datasets. The range on the Sine Wave Data project is from 0 – 99, and the number of parameters is 1111. A trigonometric function of -1 and 1 was performed to visualize this concept.

Chart, histogram

Description automatically generated

Next, our dataset was load into the pandas data frame and 20% of the data points were reserved for testing the model. The normalized dataset was then split to training/testing sets and then ran through a TimeseriesGenerator. Afterwards the x array was flatten to prepare our data the data for Building, Training, and Testing.

Graphical user interface, text, application, table

Description automatically generated

Sine wave values of index x will be fit with 99 layers and 50 steps for the time-series prediction target. To compile the model will be optimized with the Adam method and Mean Square Errors will observe the cost of loss function. The summary shown below is the total parameters and trainable parameters totaled 10,999, non-trainable parameters were 0.

Table

Description automatically generated

Understanding the performance of lost data requires a critical grasp of data visualization. After building our training model using a 50-step time series, the data suffered a small loss, as indicated by the ideal time of 1.0. For a more accurate assessment of our initial batch, the data was then reshaped into a 3D array of 1x 50 x 1. Visualized the model’s performance after training was achieved with 5 epochs. The batches were then plotted to observe performance of cost of loss.

Chart, line chart

Description automatically generated

Lastly, an evaluation of our test batches was compared between our true predictions for accuracy. Using a FOR loop help reiterate our batches and placed into an empty list. This step is necessary to understand each iteration of testing, by dropping the first data point of the current input and appending the new predicted values. Next the Sine and Predictions values were placed into a 222 rows X 3 column dataset for visualization. Now a plot was executed to comparable values to test the accuracy of the sine wave data to our predictions.

Chart, histogram

Description automatically generated

In conclusion, the orange line displayed the prediction value, and the blue line predicted the sine value. The object is to predict the accuracy of our trained input values in a Simple Recurrent Neural Network.

Part III: RNN: LSTM Neural Network

--) Explain the vanishing gradient problem

Two frequent problems of neural networks are the vanishing gradient problem and the exploding gradient problem. Both issues stem from the way gradients spread throughout the network during backpropagation. When training artificial neural networks for machine learning, the vanishing gradient problem occurs in gradient-based learning techniques and backpropagation. The network's weights are modified based on the partial derivative of the error function with respect to the current weight throughout each training iteration. However, the gradient occasionally gets so slight that it prevents the weight from changing. Backpropagation uses the chain rule to compute gradients, multiplying multiple small gradients and worsening the vanishing gradient problem.

--) Explain the exploding gradient problem

In contrast to the vanishing gradient problem is the exploding gradient problem. That happens when the loss function's weight-related gradients grow exponentially as they move backward through the layers. This can occur when the learning rate is too high or when weights initialize with large values—resulting in weights to updated quickly during training when the gradients are substantially high. Again, an unstable network can prevent the algorithm from further learning. If the eigenvalue is less than 1, the gradient will vanish and if the eigenvalue is more than 1 the gradient explodes. Adding appropriate weights and using the correct activation function, can help mitigate these gradient problems.

--) Discuss the limitations of the SimpleRNN neural network

RNNs' inherent interconnectedness makes parallelizing their training or combining it with those of other models difficult. RNNs utilize the output of the previous node to compute on the current node, in other words. Since the computational expense of parallelizing or stacking RNNs with other models is not fulfilled by any potential accuracy gains, this interconnection makes it challenging. Furthermore, slow and Complex training procedures are one of the fundamental problems with RNN's, which can take a lot of time to train when models become complex. Short-term memory is another issue that may arise when remembering patterns from earlier sequences. In addition, processing longer sequence is challenging, especially when using tanh activation functions. Lastly, the computational expense to train large datasets may limit accuracy predictions to save on cost.

--) 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.

To address the vanishing and exploding gradient problem, Long Short-Term Memory (LSTM) algorithms can be used as a viable solution. For instance, LSTM uses memory cells that allow information to flow unchanged and provide a path for the gradient propagation without vanishing or exploding its gradient. The memory cell consists of three gates: forget gate, input gate, and output gate. The forget gate controls which information is maintained or forgotten, the input gate controls what information is permitted to enter the memory cell. The output gate, in the end, regulates the data output from the memory cell. Long-term data dependencies can be learned more effectively via LSTM. This is so that the LSTM may selectively store and utilize important information over time while deleting irrelevant information using the memory cell and gates. Overall, the LSTM neural network provides a reliable solution to the gradient vanishing and exploding problem. However, LSTM have limitations with intricate structures when compiling several gates and memory cells. Effective hyperparameter tuning and network optimization may also be harder due to the heightened complexity.

Part V: RNN: LSTM with Time-Series Data

The depth of a Long Short Term Memory neural network model typically refers to its number of layers. Having a complete understanding of the architecture can help clarify the flow process in machine learning. The model consists of two layers, a hidden layer, and an output layer.

The hidden layer is designed to learn complex representations of the input data and extract meaningful features from the output layer. The previous layer, consisting of the input and weighted sum, is passed through the hidden layer to achieve this. In addition to the gates, LSTM modules also contain a "memory cell" that holds the previous hidden state and a "hidden state" that is the current output of the module. These values updated at each time step by a series of mathematical operations that learned during training.

The output layer is the final layer of the network, and its primary purpose is to generate predictions based on the features learned from the hidden layer. In other words, the hidden layer acts as a feature extractor, and the output layer uses those features to produce the final output of the model.

Diagram

Description automatically generated

The LSTM with Time-Series Data project is directed to predict Tesla stock price. By observing 5 years of historic data, we can analysis the direction of future prices more accurately. Although the forecast prediction was inaccurate, the method used could be restructured for better results. To start this process, deep-learning API Keras was the main install for the learning process. Pandas, Numpy, Matplotlib, Sci-Learn, TensorFlow, and Keras, and imported into Jupyter to assist in forecasting and visualization. I utilized a test set comprising 20% of the available data. To prepare the data for constructing, training, and evaluating Simple RNN models, I implemented normalization and transformation techniques using a MinMaxScaler. Additionally, I formatted the data into a time series array, with these steps being vital to the overall success of the modeling process. After exploratory analysis of the structure time-series data. A plot figure was shown to ensure that close price accuracy for yahoo finance. The data consist of 1141 days of TSLA pricing and will be compared to 1826 days of TSLA pricing for comparison of our prediction.

The decision to use an input sequence of 60 was to ensure that I had sufficient information for complex learning of long term dependences. Experimenting with various lengths can help optimize the performance of the model on training sets. Next, I took 20% of the TSLA data ‘test\_percent’ and split the training and testing sets through indexing. data\_test = df.iloc[split\_index - length60 :]. To normalize the machine learning preprocess data, a MinMaxScaler function rescaled our numbers between 0 and 1, by subtracting the minimum value of each feature and dividing the range. This step is vital to improving performance of the models since it reduces the differences in scale between values. A batch size of 32 to fine tune the TSLA dataset while iterating. Larger batch size can lead to better forecasting, and through parallelization can process many samples simultaneously. If the batch size is too large, there might be inefficiency. The function TimeseriesGenerator process the training data for normalization: train\_tsGenerator60 = TimeseriesGenerator(normalized\_train, normalized\_train, length=length60, batch\_size=batch\_size32)

Text

Description automatically generated Text

Description automatically generated

The importance of normalization is to improve accuracy, interpretability, and stability of the models before building, training, and testing the data. The number of features is 1, ‘Closing Price’ and ‘model’ was defined for the sequential function. Using sequential data in a timeseries captures long term dependencies between the learning process. Three layers that consist of 50 LTSM cells, two dropout layers and the ‘relu’ function were used for the training process. After each dropout the remaining memory works smarter to analysis the information. In addition, a fourth out layer ‘Dense’ provide the final results. Next, the model was complied using an optimizer ‘adam’ and loss fuction of ‘mse’ give our model a total of 50,851 parameters. Lastly, 100 epochs hyper-tuned the model for fitting, which can optimize performance in machine learning.

Graphical user interface

Description automatically generated with low confidence

Keras contains ‘loss’ information about the model’s performance during training. Retrieving the loss history keys can provide valuable insights into the behavior and performance of the model. After running the epoch function, visualization of the data can help recognize fine-tuning metrics.

Chart, line chart, histogram

Description automatically generated

Using TimeseriesGenertor of our normalized data played a vital role for predicting the forecast. Using a range of 913 – 1141 provide our predictions numbers that consist of 228 historic datapoints in the TSLA dataset. To measure the accuracy of our prediction, a plot was displayed from the original data. Confirming our model was not overfit or underfit for further analysis.

Chart

Description automatically generated

The following step is to normalize the full timeseries TSLA dataset to forecast the next 117 days of Tesla’s pricing. forecasting. The FOR loop method iterated batches and a forecast index with a range of 1141 – 1258, determined the pricing trend. Lastly, the forecast data was plotted for insights and accuracy.

Chart, histogram

Description automatically generated

Overall, our forecast prediction showed that Tesla stock price is projected to increase throughout the next 117 days. However, actual data shows that from 9-13-2022 to 3-2-2022 had the TSLA stock price decreased. Machine learning can be a powerful tool to help make price predictions of stocks when analyzing historical data, but many other financial indicators, such as sentiment, earnings, geopolitical events, and news headlines, can alter market trends. In addition, emotional factors can distort confidence in participating in trading the market, and it is crucial to understand the risk when investing capital into a business. Comparing the actual stock data with prediction stock data can help identify the inaccuracy of the TSLA price prediction model.

A new TSLA dataset was uploaded from comparative analysis of the predicted data. The data ranged from 3-5/2018 to 3-2-2022. By review actual data to our predictive values, adjustments can be made to better accuracy. After indexing the data to observe ‘close’ pricing, a FOR loop method structed the data to view both sets of data in parallel.

Graphical user interface, application, table

Description automatically generated

A significant difference can be observed, through the tail end of the tabular data. Lastly the restructured data was plotted using plt.plot(df\_JUL\_DEC\_2022, 'b', linewidth=2, alpha=0.4,) and labeled for clarity. The visualization displays historic pricing in dark blue. The light purple lines indicate the rising trend of our prediction and falling trending of actual prices.

Chart, histogram

Description automatically generated

Summary of Core Parameters

The core parameters for the LSTM time series model trained on the TSLA stock price data are: date range from 3-5-2018 to 3-2-2022, of a forecast period of 117 days. The dropout rate is 0.20, or a 20% percent reduction on un-useful data and has 50,851 total parameters. In addition, a batch size of 32, a testing percent of 20%, and an input sequence length of 60 days were all configured.

Part V: Redesign the Neural Network

Determine a prediction of stock price can be difficult since many financial factors can change market trends. Even forecast pricing from machine learning (ML) can lead to incorrect confluences when deciding to invest. It is important to fine-tune a ML model to ensure optimization. Re-designing the architecture of a Long Short-term Memory neural network can provide you with better insight of your data. Highlighting areas of improvement can be changing the percentage of your testing data (30%), keeping the same layers, increasing the number of neurons (75), the sequential length (100), and raising the dropout layer to (0.25), which randomly drops 25 percent neurons during training.

Diagram

Description automatically generated

Again, we’ll be using TSLA.csv for the dataset and changing the parameters to observe differences of the original model. After importing the same libraries, Keras, pandas, numpy and matplotlib.pyplot, the summary statistics of TSLA data was displayed to confirm information was identical to the previous experiment. Sub-setting the 4th and 5th columns provided the ‘close’ variable and a plot was displayed showing our data is identical in the exploratory data analysis stage.

Chart, histogram

Description automatically generated

When predicting future stock prices, the model needs to consider not only the current price but also the price movements over the recent past. By increasing the input sequence length, the model can observe a more extended history of price data, which can help it identify and learn more complex patterns that may be missed with a shorter sequence length. Therefore I increased the input sequence length to 100 and reserved testing data at 30%. After normalizing my data to a scale of 0 -1, a TimeseriesGenerator can be implemented for creating batches. I increased my batchsize to 64, since a larger batch can result in more stable and accurate gradient estimates. Over the configuration had 799 normalized training data points and 11 batches for the machine learning process.

After increasing the input sequence length, adjusting the testing data, and strengthening the batch size, the data can be trained for predictive analysis. I increased the neural network to 75 LSTM cells and increased the dropout to 0.25 for better machine learning practices. The ADAM method optimized our compiled model with MSE identified our loss function. The total parameters sum to 201,701, which is an increase of 400% when compared to the original data parameters.

Table

Description automatically generated

To visualize the lost performance training data. A plot was displayed observing the lost\_history\_keys for performance. The line graph shows a sharp drop in compute after the 10th epoch loss function when compared to the original data.

Chart, line chart, histogram

Description automatically generated

After normalizing the data and setting a range index of 799 – 1141, I had 342 data points to compare my prediction to the actual TSLA dataset. The matplotlib function was able to visualize both datasets for comparative analysis. The orange line is the prediction value, and the blue line is the actual values. It’s also noted that the prediction value is not overfit or underfit. In addition, the prediction trend had similar results from the original dataset.

Histogram

Description automatically generated with medium confidence

A TimeseriesGenerator constructed the normalized data for forecasting the time series data. The period length ranged from 03/05/2022 - 12/15/2022, totaling 117 business days. A FOR loop method iterated the batches for evaluation. Next the normalized data was inversed back into its true values and plotted.

Chart, histogram

Description automatically generated

Summary of Updated Core Parameters

The final prediction was a success, which shows the forecast to accurately predict pricing for Tesla as a downward trend. Change the testing percentage (30%), adding neurons for the LTSM algorithm (75), increasing the dropout layer (0.25), and having longer input sequences (100) optimized my model’s performance to accurately forecast Tesla’s pricing throughout the next 117 days.

A new dataset of TSLA stock prices was uploaded following a comparative analysis of predicted values spanning the period from March 2018 to March 2022. By scrutinizing the actual data in relation to our predicted values, improvements can be made to enhance the accuracy of future predictions. Following the indexing of the dataset to observe 'close' pricing, a FOR loop method was employed to enable a concurrent view of both datasets.

Graphical user interface, application

Description automatically generated

A noticeable patter in similarity can be observed when viewing the tabular data. Subsequently, the reconfigured data was plotted utilizing the plt.plot() function with the specified arguments of df\_JUL\_DEC\_2022, 'b', linewidth=2, and alpha=0.4, and labeled for enhanced legibility. The graphical representation displays historical pricing in dark blue, while the light purple lines depict a descending trend of our prediction, similar to the declining direction of actual prices.

Chart, histogram

Description automatically generated

Part VI: Compare Network Performance

Each of these parameters plays a crucial role in the model's performance. The dropout rate helps prevent overfitting by randomly dropping out a percentage of not useful information and sets the following LSTM layer to learn more efficient during training. Having an optimal number of neurons can prevent your model from underfitting and overfitting. The number of neurons also known as parameters indicates the model's complexity. The batch size determines how many examples the model processes in each training step, and a larger batch size can result in faster training times. The testing percent is crucial to evaluating the model's performance, a common split of 80:20 can be used to split the data into training and testing sets. Finally, the input sequence length determines how many previous data points the model considers when predicting the next value. A longer input sequence allows the model to capture more complex patterns in the data.

To finding the best values for each parameter involves experimenting with different combinations and evaluating the model's training and testing data. The tabular data shown in below compares Model 1 and Model 2 parameters for comparative analyst in forecasting TSLA stock. The period of the forecast is 117 days and the actual price values were used for accuracy.

Table

Description automatically generated

Both models had significant forecast predictions. An increase in TSLA stock price was observed in Model 1’and a decrease in TSLA stock price was observed in Model 2’. The actual values of TSLA stock price had a downward trend during the 117-day period, making the Model 2’s algorithm more accurate. After concluding the inaccuracy of Model 1, I analyzed how change my parameters to achieve better results. Overall, a 10% increase in test percentage gave Model 2 more data to learn. I kept the layer the same to emphasize other changes in Model 2, the neurons were adjusted to 100, which can be helpful for complex learning, the dropout percent increased to 25% and made the following layers learn more efficient. I decide that double my input sequence to a length of 100 would lead the model to better understand of TSLA stock historical pricing. In addition, the batch size of 100 helped Model 2 train faster, and Epochs decreased to 50, since neuron and dropout percent can optimize predictions. Lastly the total parameters of Model 2 show a 400 percent increase from Model 1 parameters. This can lead to improving the relationship of the input data and provide more variety of sequencing complex patterns.

Part VII: Project Report | Intro - Summary

Neural networks have gained traction for businesses to help improve in decision-making and optimize operational efficiency through forecast predictions. This report reviews the importance of neural networks and the impact of adjusting various parameters, such as test percentages, layers, neurons, dropout percentage, input sequence, batch size, epochs, and total parameters for performance. The project focuses on two types of Neural networks, Simple Recurring Neural Networks and Long Short-Term Memory Neural Networks (LSTM). The data used in our first project Simple RNN uses sine wave data and reviews the basic concepts of neural network architecture. The second project will focus on LSTM neural network and highlight methods for improving forecast predictions on TSLA stock over 117 days. The use of neural networks can help businesses identify patterns and trends in large and complex data sets that may be difficult to understand using conventional statistical techniques.

The architecture of the RNN is explained to better understand how neural networks are applied to time-series data. The dataset is loaded into the pandas data frame, and 20% of the data is reserved for testing the model. The normalized dataset is then split into training and testing sets and run through a TimeSeriesGenerator. The model is compiled with the Adam method and Mean Square Errors to observe the cost of the loss function. The performance of the model is evaluated through data visualization, and the accuracy of the predictions is compared to the true values using a FOR loop. The Sine Wave Data is used to predict the accuracy of the trained input values in a Simple RNN. Observing a Simple RNN with Sine Wave Data can be helpful for time-series data by providing insights into the basic concepts of neural networks. Understanding the several layer types, activation functions, and neuronal counts needed to evaluate complex datasets is made more accessible by understanding the architecture of the RNN. The comparison of Sine Wave Data can be used to predict the accuracy of the trained input values when analyzing similar time-series data. Additionally, visualizing the model's performance through data visualization techniques can help evaluate the model's performance and make future improvements. Overall, knowing the functionality of a Simple RNN can help you better understand exploding and vanishing gradients issues and the basic concepts parameters.

The second projects focus on Long Short-Term Memory (LSTM) neural networks for time series data of TSLA closing price that involves hidden layers and an output layer. The hidden layer learns complex representations of the input data and extracts meaningful features throughout the machine-learning process. The output layer generates predictions based on the features known from the hidden layer. Normalization is crucial in improving the model's accuracy, interpretability, and stability. The LSTM model for stock price prediction showed noticeable improvements by increasing the input sequence length of TSLA's closing price, which helped identify more complex patterns. The model's parameters, such as the dropout rate, number of neurons, batch size, testing percent, and input sequence length, all play a vital role in the model's performance.

By experimenting with different combinations and evaluating the model's training and testing data, I found better values for each parameter. Furthermore, my comparison revealed that Model 2's algorithm significantly improved on TSLA closing price predictions. Overall, expanding the testing percentage, increasing the number of neurons and dropout percent, input sequence length, and batch size can improve the relationship of the input data for sequencing complex patterns.

Resources

<https://www.kaggle.com/datasets/tavoglc/sunspots>

<https://finance.yahoo.com/quote/TSLA/history?p=TSLA>

YouTube Video

<https://youtu.be/y5sCdgWJ7Ig>.