Research Draft: Final Project Draft

B351 / Q351

Basic Information

Project Title
How to predict the Stock market price trends with AI model. (Tesla, Apple, Amazon)

Main Research Project Question

We are trying to find the correlation from the data set with the historical statistic data set and use the AI for predicting the stock price trends and trying to find out which machine learning architecture is the most accurate and similar to prediction. And wanted to figure out if this AI model works for predicting the stock trend. So we tried to find with LSTM, GRU, LSTM + GRU, Lasso, and MLP to see if it works.

Team Members

1. Name : Anthony Dove
2. Name : Chima Philips
3. Name : Daeyeop Kim

1 Summary of Work

1. Since we have tried to figure out the trend of the stock market prices with several data sets: Tesla, Amazon, and Apple. So that we could compare if the implemented model is working with several different sets and see which one is better. We brought the dataset from GitHub or Yahoo! finance to train the models. For this, we might use LSTM and GRU to learn to figure out the trends of the stock and check the MSE for which machine learning architecture is more accurate. And tried with MLP for LSTM if there is any improvement for the model. And find out if there is another way to make the regression prediction which we chose to do Lasso regression for the regression model.

2 Algorithms and Implementation

2.1 (a) What algorithms are you using?

1. In this research, We used 5 machine learning architectures or methods for the prediction. For Daeyeop, use the LSMT, GRU, And LSTM + GRU. Anthony used the Lasso regression, and Chima added multiple layers on LSTM for the MLP the predict the trends of the stock price.

2.2 (b) How do they work?

GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) are both types of Recurrent Neural Network (RNN) architectures that are designed to handle the vanishing gradient problem commonly encountered in traditional RNNs when learning long-range dependencies in sequence data. For the GRU has two gates: update and reset, that manage the flow of information. It contains two different components, Candidate Hidden State which represents the potential new hidden state and is a combination of the current input with the past hidden state, modulated by the reset gate, and Current Hidden State, which is an interpolation between the old hidden state and the candidate hidden state. LSTM has 5 gates: Forget Gate to decide what information should be discarded from the cell state, Input Gate to decide which value will be updated in the cell state, Cell State the internal memory that carries the sequential processing, Output gate to determine the next hidden state, Hidden state for

the output that gets passed to next time step. And are two different components Candidate and Final memory cells which filter the version of the input and result from the forget gate to the previous cell state, and the input gate to the candidate memory cell, merging past knowledge with new insights.

Both GRU and LSTM cells use various gating mechanisms to control the flow of information, reduce the impact of short-term dependencies, and capture long-term relationships in sequence data. GRUs are generally considered to be more efficient than LSTMs as they have fewer parameters (due to merging the forget and input gates), but LSTMs may perform better on more complex tasks that benefit from their more expressive learning capabilities.

2.3 (c) What is the time/space complexity?

2. Time Complexity of an LSTM:
   T = LSTM cell processing a sequence of length
   The time complexity for one forward pass is O(T)
   the size of input = I, the number of hidden unit H.
   LSTM cell performs matrix multiplications that are O(IH + HH)
   the input at each time step is multiplied by the input weights (dimension I x H) and the previous hidden state is multiplied by the recurrent weights (dimension H x H)

3. For each time step: Input-to-hidden multiplication: O(IH)
   Hidden-to-hidden multiplication: $O(H^2)$.
   For T time steps: Overall time complexity: $O(T * (IH + H^2))$
   If you have L layers in your LSTM network,
   you will need to multiply this by L because each layer's output is used as input for the next layer:
   With L layers and T time steps: $O(L * T * (IH + H^2))$
   GRU Time complexity : $O(TN)$
   T is the length of the sequence
   N is the number of parameters in the model

4. Space Complexity of an LSTM and GRU:

   Space complexity refers to the amount of memory needed to store the weights, biases, and intermediate states of the LSTM.

   For the weights and biases: There are four sets of weights for the input and four sets for the recurrent connections because of the four gates (input, forget, cell/update, and output) within an LSTM cell. The weights for the gates have dimensions I x H for input-to-hidden and H x H for hidden-to-hidden, and each set of biases has dimension H.

   Thus, for a single layer, the space complexity of the weights and biases is approximately:
   Weights: $(4 * (I \times H + H^2))$
   Biases: (4 x H)

   For intermediate states, such as the hidden state and cell state vectors, for each time step the space complexity is O(H), and thus for T time steps it is (T x H). If we have L layers, we have to again multiply by the number of layers L.

5. the overall space complexity of a single LSTM layer is $(4 * (I \times H + H^2) + L * T \times H)$

2.4 (d) What are the limitations?

6. Gated Recurrent Units (GRUs) and Long Short-Term Memory Networks (LSTMs) are both popular types of recurrent neural networks (RNNs) used to handle sequences of data for tasks such as natural language processing, time series analysis, and speech recognition. Despite their successes, they both have limitations:

7. There were 4 main Limitations of GRUs: Complexity and Computation Costs, Difficulty with Very Long Sequences, Hyperparameter Sensitivity, Limited Expressiveness for certain tasks.

8. There were 5 main Limitations of LSTMs: Even More Complexity and Computation Costs, Learning Long-Term Dependencies, Gradient Issues, Overfitting Risk, Becoming Outmoded

9. For both GRUs and LSTMs: Training Time, Sequential Computation, Data Sparsity and Bias, Memory Limitations

2.5 (e) What were some alternatives?

10. The LSTM or GRU is a transformer that has global attention so that does not suffer from the memory problem.

2.6 (f) How do the algorithms apply to/answer your research questions?

11. Since our research questions were based on predicting the stock price and trends, We are searching for which algorithm has the best model for predicting the stock price and trends.

3 Processing for this research

3.1 Describe how your solution models human thought processes

The human is good at sequential data and good at mapping trends. That is why we choose to use the LSTM, GRU, LSTM + GRU, MLP, and lasso regression. So humans have limited memory of thought processes which we can think of using the LSTM, GRU, or other machine learning architectures or models to calculate the prediction by using the sequential data.
Since we had limited time to figure this out, we separated works like Daeyeop doing the LSTM, GRU, and LSTM work Chima choosing MLP, and Anthony choosing the Lasso regression

4 Result and conclusion explanation

4.1 (a) Present your findings in graphs and/or tables (NOT a giant paragraph of text).

These figures of position print in wrong for some reason...

4.1.1 LSTM, GRU, LSTM+GRU

4.1.2 MLP

4.1.3 Lasso

4.2 (b) Why did you get these results?

From LSTM, GRU, LSTM + GRU
MSE LSTM Open price: 490.4127965971428
MSE GRU Open price: 250.36214289323135
MSE LSTM +GRU Open price: 643.0077644469684
MSE LSTM Close price: 206.37829918675038
MSE GRU Close price: 286.313021189539
MSE LSTM + GRU Close Price: 336.5722089225555
From Lasso MSE : 1365.4239272414393
From MLP Mean Squared Error on Test Set: 0.001780773396603763

Figure 1: $LSTMTeslaOpen$

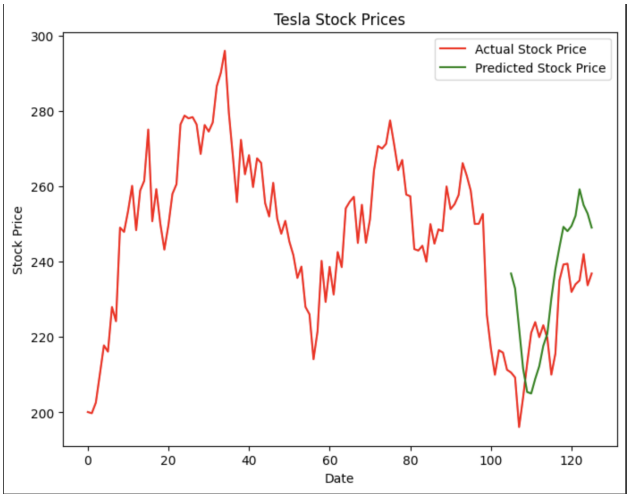

Figure 2: $TeslaOpenGRU$

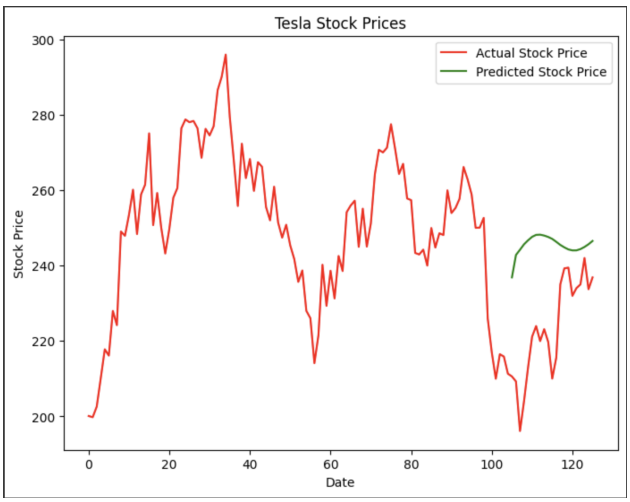

Figure 3: $TeslaOpenLSTM + GRU$

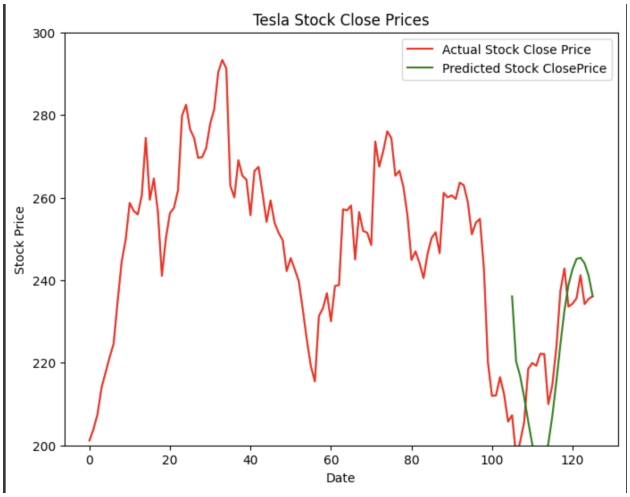

Figure 4: $TeslaCloseLSTM$

Figure 5: $Tesla_Close_GRU$

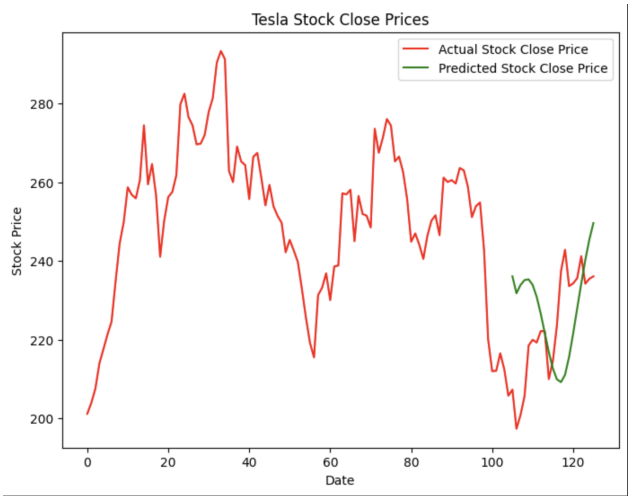

Figure 6: $TelsaCloseLSTM + GRU$

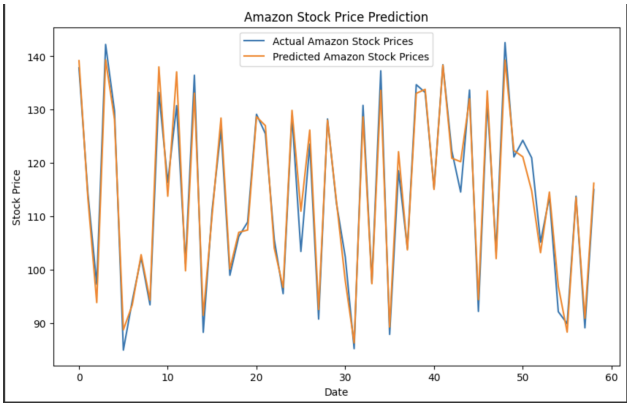

Figure 7: $MLPwithAMZN$

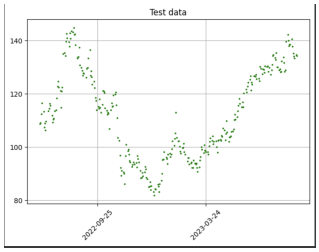

Figure 8: Lasso Prediction with scatter

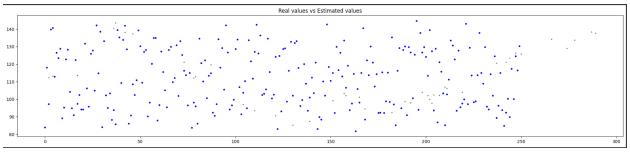

Figure 9: prediction with lasso

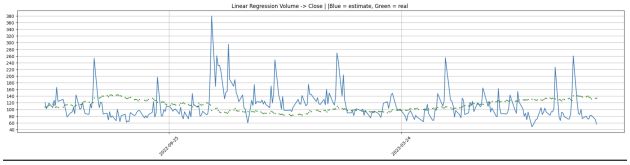

Figure 10: Linear Regression with Volume

4.3 (c) Were the results what you expected?

We were expected to see the graph for LSTM, GRU, and LSTM + GRU graph and mse to LSTM + GRU ¡ LSTM ¡ GRU so that it could improve the machine learning architecture for some reason. We did expect to get an improved mse for the Lasso and We expected to improve the result with MLP because it is multilayer which may increase.

4.4 (d) What do the results imply about the problem space?

Since the output or results were not consistent, we can conclude that these models were sufficient to not. The problem is more complex than simple models like LSTM, GRU, and the architectures or models we provided for the research. So, it may be difficult to generalize models to other companies or stocks. We could think that might be necessary to include more features to generalize the model.

4.5 (e) Anything else of interest?

It was pretty impressive that LSTM, GRU, and other architectures that were not consistent in the output. It was surprising that Lasso did work for the regression in this research with bad prediction because Lasso usually does not fit this regression problem. And the MLP shows a pretty high percentage of the accuracy it was quite impressive.

Code explanation

4.6 Daeyeop's code work

```
## for importing libraries
import pandas as pd
import numpy as np

## import the dataset
## for 1 year set
telsa_train_data = pd.read_csv("/content/TLSA_train.csv") ## Daeyeop used the Google colab for research and uploaded train data and test data from the yahoo finance for learning (train = 1 yr, test = 6 month long Daily)
# printing dataset header
telsa_train_data.head() ## for checking is right data set

## by checking the info see if there is the missing values
telsa_train_data.info()

## Compare two different prices: Open and Close
## The output shows that our dataset consists of seven columns. However, in this section, we are only interested in the Open and Close columns. Therefore, we will select the Open and Cloase columns from the dataset.
#filtering open column
training_processed = telsa_train_data[['Open']].values
#filtering close column
training_processed2 = telsa_train_data[['Close']].values
#Scaling feature
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0, 1))
training_scaled = scaler.fit_transform(training_processed) ## open
training_scaled2 = scaler.fit_transform(training_processed2) ## close
print(len(training_scaled)) ### open
print(len(training_scaled2)) ### close
```

```
# Our feature set will consist of 60 timesteps of 1 feature. The feature set basically consists of the opening stock price of the past 60 days
#training features contained data of last 60 days
#training labels contain data of 61st day
training_features= []
training_labels = []
for i in range(60, len(training_scaled)):
  training_features.append(training_scaled[i-60:i,0])
  training_labels.append(training_scaled[i, 0])
#converting training data to numpy arrays
X_train = np.array(training_features)
y_train = np.array(training_labels)
## and check the shape of train set
print(X_train.shape)
print(y_train.shape)
## did the same thing to the Close set like Open set.
#training features contained data of last 60 days
#training labels contain data of 61st day
training_features2= []
training_labels2 = []
for i in range(60, len(training_scaled2)):
  training_features2.append(training_scaled2[i-60:i,0])
  training_labels2.append(training_scaled2[i, 0])
#converting training data to numpy arrays
X_train2 = np.array(training_features2)
y_train2 = np.array(training_labels2)
print(X_train2.shape)
print(y_train2.shape)
## We need to reshape our input feature into a three-dimensional format
## for opening case
#Converting data into 3D shape
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
#Converting data into 3D shape ## for closing
X_train2 = np.reshape(X_train2, (X_train2.shape[0], X_train2.shape[1], 1))
# The following script creates our LSTM model. We have four LSTM layers with 100 nodes each. Each LSTM layer is followed by a dropout layer to avoid overfitting. The final dense layer has one node since the output is a single value.
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Activation, Dense, Flatten, Dropout,  Flatten, LSTM
from tensorflow.keras.models import Model
#defining the LSTM network
## Open

input_layer = Input(shape = (X_train.shape[1], 1))
fc1 = Dense(input_layer.shape[1],activation='relu')(input_layer)
lstm1 = LSTM(100, activation='relu', return_sequences=True)(fc1)
do1 = Dropout(0.2)(lstm1)
lstm2 = LSTM(100, activation='relu', return_sequences=True)(do1)
do2 = Dropout(0.2)(lstm2)
lstm3 = LSTM(100, activation='relu', return_sequences=True)(do2)
do3 = Dropout(0.2)(lstm3)
lstm4 = LSTM(100, activation='relu')(do3)
do4 = Dropout(0.2)(lstm4)

output_layer = Dense(1)(do4)
model = Model(input_layer, output_layer)
```

```
model.compile(optimizer='adam', loss='mse')

## I create the basically LSTM, GRU, LSTM+GRU for OPEN and Close.
#defining the LSTM network
## Close
input_layer = Input(shape = (X_train2.shape[1], 1))
fc1 = Dense(input_layer.shape[1],activation='relu')(input_layer)
lstm1 = LSTM(100, activation='relu', return_sequences=True)(fc1)
do1 = Dropout(0.2)(lstm1)
lstm2 = LSTM(100, activation='relu', return_sequences=True)(do1)
do2 = Dropout(0.2)(lstm2)
lstm3 = LSTM(100, activation='relu', return_sequences=True)(do2)
do3 = Dropout(0.2)(lstm3)
lstm4 = LSTM(100, activation='relu')(do3)
do4 = Dropout(0.2)(lstm4)

output_layer = Dense(1)(do4)
model_1 = Model(input_layer, output_layer)
model_1.compile(optimizer='adam', loss='mse')



##Next, we need to convert the output y into a column vector.
# for Open
print(X_train.shape)
print(y_train.shape)
y_train= y_train.reshape(-1,1)
print(y_train.shape)

# For Close
print(X_train2.shape)
print(y_train2.shape)

y_train2= y_train2.reshape(-1,1)
print(y_train2.shape)
#training the model
## for Open / LSTM
model_history = model.fit(X_train, y_train, epochs=100, verbose=1, batch_size = 32)
#training the model
## for Close / LSTM
model_1_history = model_1.fit(X_train2, y_train2, epochs=100, verbose=1, batch_size = 32)

## Create the GRU
## for open
from tensorflow.keras.layers import Input, Activation, Dense, Flatten, Dropout,  Flatten, GRU
input_layer = Input(shape = (X_train.shape[1], 1)) ## need to change the different format of
↪  the input
gru1 = GRU(100, activation='relu', return_sequences=True)(input_layer)
do1 = Dropout(0.2)(gru1)
gru2 = GRU(100, activation='relu', return_sequences=True)(do1)
do2 = Dropout(0.2)(gru2)
gru3 = GRU(100, activation='relu', return_sequences=True)(do2)
do3 = Dropout(0.2)(gru3)
gru4 = GRU(100, activation='relu')(do3)
do4 = Dropout(0.2)(gru4)

output_layer = Dense(1)(do4)
model1 = Model(input_layer, output_layer)
```

```
model1.compile(optimizer='adam', loss='mse')
## same process like the LSTM...
## For opening
print(X_train.shape)
print(y_train.shape)

y_train= y_train.reshape(-1,1)
print(y_train.shape)
## For close
from tensorflow.keras.layers import Input, Activation, Dense, Flatten, Dropout,  Flatten, GRU
input_layer = Input(shape = (X_train2.shape[1], 1)) ## need to change the different format of the input
gru1 = GRU(100, activation='relu', return_sequences=True)(input_layer)
do1 = Dropout(0.2)(gru1)
gru2 = GRU(100, activation='relu', return_sequences=True)(do1)
do2 = Dropout(0.2)(gru2)
gru3 = GRU(100, activation='relu', return_sequences=True)(do2)
do3 = Dropout(0.2)(gru3)
gru4 = GRU(100, activation='relu')(do3)
do4 = Dropout(0.2)(gru4)

output_layer = Dense(1)(do4)
model1_1 = Model(input_layer, output_layer)
model1_1.compile(optimizer='adam', loss='mse')
print(X_train2.shape)
print(y_train2.shape)

y_train2= y_train2.reshape(-1,1)
print(y_train2.shape)
###training the GRU model Open
model1_history = model1.fit(X_train, y_train, epochs=100, verbose=1, batch_size = 32)
#training the GRU model Close
# for close
model1_1_history = model1_1.fit(X_train2, y_train2, epochs=100, verbose=1, batch_size = 32)

## For LSTM + GRU
## for open
from tensorflow.keras.layers import Input, Activation, Dense, Flatten, Dropout,  Flatten, GRU, LSTM
input_layer = Input(shape = (X_train.shape[1], 1))
fc1 = Dense(input_layer.shape[1],activation='relu')(input_layer)
lstm1 = LSTM(100, activation='relu', return_sequences=True)(fc1)
do1 = Dropout(0.2)(lstm1)
lstm2 = LSTM(100, activation='relu', return_sequences=True)(do1)
do2 = Dropout(0.2)(lstm2)
lstm3 = LSTM(100, activation='relu', return_sequences=True)(do2)
do3 = Dropout(0.2)(lstm3)
lstm4 = LSTM(100, activation='relu', return_sequences=True)(do3)
do4 = Dropout(0.2)(lstm4)
gru1 = GRU(100, activation='relu', return_sequences=True)(do4)
do5 = Dropout(0.2)(gru1)
gru2 = GRU(100, activation='relu', return_sequences=True)(do5)
do6 = Dropout(0.2)(gru2)
gru3 = GRU(100, activation='relu', return_sequences=True)(do6)
do7 = Dropout(0.2)(gru3)
gru4 = GRU(100, activation='relu')(do7)
do8 = Dropout(0.2)(gru4)

```

```
output_layer = Dense(1)(do8)
model3 = Model(input_layer, output_layer)
model3.compile(optimizer='adam', loss='mse')
 ## samething like the close case too. and learning the model
 model3_history = model3.fit(X_train, y_train, epochs=100, verbose=1, batch_size = 32)

### Testing the Stock Prediction Model

#The test data should also be converted into the right shape to test our stock prediction model. We will do that later. first, import the data and then remove all the columns from the test data except the Open and Close columns.
#creating test set
testing_complete_data = pd.read_csv("/content/TLSA_test.csv")
testing_processed = testing_complete_data[['Open']]. values
testing_processed2 = testing_complete_data[['Close']]. values
## concatenate the training and test sets. I do this because to predict the first value in the test set, the input will be the data from the past 60 days, which is basically the data from the last 60 days in the training set.
# for opening
all_data = pd.concat((telsa_train_data['Open'], testing_complete_data['Open']), axis=0)
test_inputs = all_data [len(all_data) - len(testing_complete_data) - 60:].values
print(test_inputs.shape)
test_inputs = test_inputs.reshape(-1,1)
test_inputs = scaler.transform(test_inputs)
print(test_inputs.shape)
test_features = []
for i in range(60, 80):
  test_features.append(test_inputs[i-60:i, 0])
X_test = np.array(test_features)
print(X_test.shape)
#converting test data into 3D shape
X_test = np.reshape(X_test, (X_test.shape[0], X_test. shape[1], 1))
X_test2 = X_test.copy()
X_test3 = X_test.copy()
print(X_test.shape) ## LSTM
print(X_test2.shape) ## GRU
print(X_test3.shape) ## LSTM + GRU
#making predictions on test set
y_pred = model.predict(X_test) ## for LSTM
y_pred1 = model1.predict(X_test2) ## for GRU
y_pred2 = model3.predict(X_test3) ## LSTM + GRU
#converting scaled data back to original data
y_pred = scaler.inverse_transform(y_pred) ## for LSTM
y_pred1 = scaler.inverse_transform(y_pred1) ## for GRU
y_pred2 = scaler.inverse_transform(y_pred2) ## for LSTM + GRU
## for the MSE and Pred len is 20 so it goes -20
real_value_open  = testing_processed[-20:]
from sklearn.metrics import mean_squared_error
mse_value_lstm = mean_squared_error(real_value_open, y_pred)
mse_value_gru = mean_squared_error(real_value_open, y_pred1)
mse_value_lstm_gru = mean_squared_error(real_value_open, y_pred2)
print("MSE LSTM Open price:", mse_value_lstm)
print("MSE GRU Open price:", mse_value_gru)
print("MSE LSTM + GRU Open price:", mse_value_lstm_gru)
#plotting original and predicted stock values
#Finally, to compare the predicted output with the actual stock price values, you can plot the two values
## LSTM
```

```
plt.figure(figsize=(8,6))
plt.plot(testing_processed, color='red', label='Actual Stock Price')
data_len, pred_len = len(testing_processed), len(y_pred)

y_p = [i[0] for i in y_pred]
x_pred = [k for k in range(data_len-21, data_len, 1)]

plt.plot(x_pred, [testing_processed[-1]] + y_p, color='green', label='Predicted Stock Price')
# plt.plot(y_pred, color = 'green', label = 'Predicted Stock Price')
plt.title('Tesla Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()

## Basically same thing for the Close case for the LSTM and other GRU, GRU+LSTM for Open,
↪  CLose case.
```

4.7 Antony's code work

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.ticker as ticker
import datetime as get
from matplotlib.ticker import MultipleLocator
from sklearn.model_selection import train_test_split
#files go here
train_data =
↪  pd.read_csv('https://raw.githubusercontent.com/CeeBee99/AI_Final_Project/main/AMZN(training).csv')
test_data =
↪  pd.read_csv('https://raw.githubusercontent.com/CeeBee99/AI_Final_Project/main/AMZN(testing).csv')
train_dates = pd.to_datetime(train_data['Date'])
test_dates = pd.to_datetime(test_data['Date'])
print(train_dates)
def simple_regression(data_x,data_y):
    x = 0
    y = 0
    xy = 0
    x2 = 0
    n = len(data_x)
    for q in range(n):
        x += data_x[q]
        y += data_y[q]
        xy += data_x[q] * data_y[q]
        x2 += data_x[q] * data_x[q]
    m = (n*(xy-(x*y) )) / (n *(x2 - (x*x)))
    b = (y-(x*m))/n
    return lambda k: m*k + b
## train data for Open

train_data['Open']

##plot of training data by

plot_x = train_dates
plot_y = train_data['Close']
```

```
plt.plot(plot_x, plot_y, color='green', marker='o', linestyle='none',linewidth=1,
↪  markersize=1.5)

# format
plt.xticks(rotation=45)
plt.gca()#.autofmt_xdate()
plt.rcParams["figure.figsize"] = [25, 5]
ax = plt.gca()
ax.yaxis.set_major_locator(MultipleLocator(20))
ax.xaxis.set_major_locator(MultipleLocator(180))

plt.grid()
plt.tight_layout()
plt.title("Test data")
plt.show()

#plotting with the volume doesn't work, likely cause of it's data type
smaller_vol=[]
n = len(train_data['Volume'])
for x in range(n):
    smaller = train_data['Volume'][x]/1
    smaller_vol.append(smaller)
smaller_test_vol=[]
n = len(test_data['Volume'])
for x in range(n):
    smaller = test_data['Volume'][x]/1
    smaller_test_vol.append(smaller)

##plot of training data with normal regression line

plot_x = train_dates
plot_y = train_data['Open']
plt.plot(plot_x, plot_y, color='green', marker='o', linestyle='none',linewidth=1,
↪  markersize=1.5)

# format
plt.xticks(rotation=45)
plt.gca()#.autofmt_xdate()
plt.rcParams["figure.figsize"] = [25, 5]
ax = plt.gca()
ax.yaxis.set_major_locator(MultipleLocator(20))
ax.xaxis.set_major_locator(MultipleLocator(180))

RegVals = []
funct = simple_regression(smaller_vol,train_data['Close'])
for x in smaller_vol:
                          val = funct(x)
                          RegVals.append(val)


plt.grid()
plt.plot(plot_x,RegVals)
#plt.tight_layout()
plt.title("Linear Regression Volume -> Close | |Blue = estimate, Green = real ")
plt.show()
import math
def mean(vals):
    n = len(val)
```

```
    sum=0
    for x in vals:
        sum+=x
    return (sum/n)
def pearsonCC(data_x,data_y):
    n = len(data_x)
    xsum=0
    ysum=0
    xysum=0
    x2sum=0
    y2sum=0
    for q in range(n):
        xsum += data_x[q]
        ysum += data_y[q]
        xysum += data_x[q]*data_y[q]
        x2sum += data_x[q]**2
        y2sum += data_y[q]**2

    top = (n*xysum)-(xsum*ysum)
    bottom = math.sqrt( ( (n*x2sum)- (xsum**2) )*((n*y2sum)- (ysum**2)))
    return top/bottom
## for getting the
r = pearsonCC(train_data['Open'],train_data['Close'])
r
#residual sum of squares list of float,list of float -> float
def RSS(data_x,data_y):
    n = len(data_x)
    sum = 0
    EST_line = simple_regression(data_x,data_y)
    for q in range(n):
        sum += (data_y[q]-EST_line(data_x[q]))**2
    return sum

base = RSS(train_data['Open'],train_data['Close'])
print(base)

#should calculate a number to be a weigh for a variable in lasso
def MSEP(data_x,data_y,test_x,test_y):
    n = len(data_x)
    EST_line = simple_regression(data_x,data_y)
    base = RSS(data_x,data_y)/n
    ovr_sum = 0
    iter_avg = 0
    for q in test_y:
        for k in range(n):
            iter_avg += ( (base + EST_line(test_x)) - test_y)
        ovr_sum += (iter_avg/n)
    val = ovr_sum/len(test_y)
    return val

#testing the MSEP function
#returning a list of vals instead of the average of float
#also takes a long time to compute which shouldn't be happening
'''
q = MSEP(train_data['Open'],train_data['Close'],test_data['Open'],test_data['Close'])
print(q)
base = RSS(train_data['Open'],train_data['Close'])/n
weight = mean(q)
```

```
#val = base + (weight *train_data['Open'][1] )
#print(val)
print(train_data['Close'][1])
'''
train_data.head()
#files go here
train_data = pd.read_csv('https://raw.githubusercontent.com/CeeBee99/AI_Final_Project/main/AMZN(training).csv')
test_data = pd.read_csv('https://raw.githubusercontent.com/CeeBee99/AI_Final_Project/main/AMZN(testing).csv')
train_dates = pd.to_datetime(train_data['Date'])
test_dates = pd.to_datetime(test_data['Date'])

y_vals = train_data.drop(train_data.columns[[1,2,3,5,6]],axis = 1)
x_vals = train_data.drop(train_data.columns[[0,4]],axis = 1)
#training and testing data creation
#x_vals.head()
## splitting data
x_train, x_test, y_train, y_test = train_test_split(x_vals, y_vals, test_size=0.15)


y_test_dates = y_test.drop(y_test.columns[[1]],axis = 1,inplace = False)
y_test.drop(['Date'],axis =1,inplace = True)

y_train_dates = y_train.drop(y_train.columns[[1]],axis = 1,inplace = False)
y_train.drop(['Date'],axis =1,inplace = True)

print(len(x_test))
print(len(x_train))
y_test_dates = pd.to_datetime(y_test_dates['Date'])
#y_train_dates =pd.to_datetime(y_train_dates['Date'])

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
## data preprocessing
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
## use the Lasso regression
lasso_reg = Lasso(alpha = 0.01, max_iter=100000)
lasso_reg.fit(x_train_scaled,y_train)
estimate_vals = lasso_reg.predict(x_train_scaled)
print(estimate_vals)
vals = []
for k in estimate_vals:
    vals.append(k)

plot_x = y_test_dates
plot_y = y_test['Close']
plt.plot(plot_y, color='green', marker='o', linestyle='none',linewidth=1, markersize=1.5)

plot_x = y_train_dates
plot_y = vals

plt.plot(plot_y, color='blue', marker='o', linestyle='none',linewidth=1, markersize=3)
```

```

plt.title("Real values vs Estimated values")
plt.show()
```

4.8 Chima's code work

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import pandas as pd
# Load the data from a URL
def load_data(url):
    # Read stock price data from the provided URL
    df = pd.read_csv(url)
    # Extract the closing prices for simplicity
    data = df['Close'].values
    return data

# Preprocess the data
def preprocess_data(data):
    # Normalize the data using Min-Max scaling
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data.reshape(-1, 1))

    return data_scaled, scaler

# Split the data into training and testing sets
def split_data(data):
    X = data[:-1]  # Features (all data except the last point)
    y = data[1:]   # Target is the next data point

    # Split the data into 80% training and 20% testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    return X_train, X_test, y_train, y_test

# Build the MLP model
def build_model(input_size):
    model = Sequential()
    model.add(Dense(64, input_dim=input_size, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='linear'))  # Output layer with linear activation for
    ↪  regression

    optimizer = Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    return model
# Train the model
def train_model(model, X_train, y_train, epochs=50, batch_size=32):
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1)
# Evaluate the model on the test set and display predictions with a graph
def evaluate_model(model, X_test, y_test, scaler):
```

```
    predictions = model.predict(X_test)
    predictions = scaler.inverse_transform(predictions)  # Inverse transform to get original
    ↪  scale
    y_test_orig = scaler.inverse_transform(y_test.reshape(-1, 1))
    plt.figure(figsize=(10, 6))
    plt.plot(y_test_orig, label='Actual Amazon Stock Prices')
    plt.plot(predictions, label='Predicted Amazon Stock Prices')
    plt.title('Amazon Stock Price Prediction')
    plt.xlabel('Date')
    plt.ylabel('Stock Price')
    plt.legend()
    plt.show()
    loss = model.evaluate(X_test, y_test)
    print(f'Mean Squared Error on Test Set: {loss}')

    return predictions
def main():
    # Load stock price data from a URL
    stock_data_url =
    ↪  'https://raw.githubusercontent.com/CeeBee99/AI_Final_Project/main/AMZN(training).csv'
    stock_data = load_data(stock_data_url)

    # Preprocess the data
    processed_data, scaler = preprocess_data(stock_data)

    # Split the data
    X_train, X_test, y_train, y_test = split_data(processed_data)

    # Build the MLP model
    input_size = 1  # Since we only have one feature (closing price)
    model = build_model(input_size)

    # Train the model
    train_model(model, X_train, y_train)

    # Evaluate the model and display predictions with a graph
    predictions = evaluate_model(model, X_test, y_test, scaler)

    return predictions

if __name__ == '__main__':
    predictions = main()
    print("Predictions:", predictions)

```

Improvement

4.9 (a) What improvements would you make to the AI?

In this research, we tried several things to improve the AI. Specifically, we took a hybrid approach between the LSTM and GRU models to investigate if any part of the architectures lent performance to each other. What we found was that it did not have a strong impact on the RMSE compared to the pure architectures. However, these models could always be improved in several ways. One way would be to increase the amount of training data we feed to the model, as well as to change more hyper-parameters, like the learning rate, batch size, as well as using a learning-rate scheduler to dynamically adapt the learning rate. Similarly, we might also experiment with different Loss Functions or optimizers, such as AdamW, SGD, or RMSProp. We might also take a different approach to training the models, such as predicting masked regions of stock

values.

4.10 (b) What lessons did you learn?

From these lessons, we learned that AI is still hard to solve real-world problems. And there were a lot more complex problems that AI more machine learning could not solve. Even if there were a fully given data set for training, it still needs more info to increase the accuracy.

4.11 (c) Anything else of interest?

We were curious about different types of machine learning architectures that were working better than LSTM, GRU, MLP, or Lasso. Hopefully, could find real complex problems like predicting the stock market price or trends in further future.