Introduction

08162013_Twitter_Stocks_article-1.png

In recent years, Generative Adversarial Networks (GANs) have achieved good results in solving many complex problems (e.g., creating realistic images and videos, image-to-image and text-to-image conversion), but the effectiveness of using this type of network for stock price forecasting is still a matter of debate. This type of models was previously used mainly to generate new photos, videos, or texts, but not time series, especially as volatile as stock prices. The thing is that stock prices are usually a bit tougher to forecast due to market volatility and social influence on the trend of each stock.

Recently data scientists have started trying out GAN models for stock price prediction and some works show promising results. Moreover, news/social media analysis is becoming more and more used for this task, so why not try and combine those two approaches? :)

Therefore, in this notebook I create a model for forecasting Amazon (AMZN) stock prices which takes into account not only historical data and technical indicators, but also such external factors influencing the market as the mood of traders and brand reputation, which is represented in social media posts.

Table of contents

Import packages

```
import os
import numpy as np
import csv
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import math
import time
import tensorflow as tf
from tensorflow.keras.layers import GRU, LSTM, Bidirectional, Dense, Flatten, Conv1D
from tensorflow.keras import Sequential
from tensorflow.keras.utils import plot_model
from pickle import load
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
import statsmodels.api as sm
from math import sqrt
from datetime import datetime, timedelta
from sklearn.preprocessing import MinMaxScaler
from pickle import dump
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import unicodedata

import warnings
warnings.filterwarnings("ignore")
```

Get weekly sentiment for stock ticker

There is much more to the process of stock price formation than plain historical data. Over 1 day, one online post might be a turning point in the course of events, which may result in

AttributeError TypeError ValueError •••

You

Please explain the error:

```
ValueError: Argument(s) not recognized: {'lr': 0.0
```

Gemini

Hypothesis

The error message "ValueError: Argument(s) not recognized: {'lr': 0.0005}" indicates that the `Adam` optimizer in TensorFlow/Keras no longer accepts the `lr` argument for specifying the learning rate. Instead, it should be specified using the `learning_rate` argument.

Suggested Changes

```
learning_rate = 5e-4
epochs = 500

# Change 'lr' to 'learning_rate'
g_optimizer = tf.keras.optimizers.Adam(learning_
d_optimizer = tf.keras.optimizers.Adam(learning_

generator = make_generator_model(X_train.shape[1
discriminator = make_discriminator_model(X_train
```

Use code with caution

▶ Sources

Rate this answer

the market crash. Elon Musk tweets, coronavirus, start of russian full-scale invasion of Ukraine are the proof to that. Therefore, we will take into account another important external indicator, such as the mood of stock market participants. The most effective method in this task is the analysis of the tone (sentiment analysis) of the text, in this notewook we will be consider posts in the social network Twitter.

```
stock_name = 'AMZN'
```

```
all_tweets = pd.read_csv('/content/stock_tweets.csv')
```

```
print(all_tweets.shape)
all_tweets.head()
```

```
(80793, 4)
```

	Date	Tweet	Stock Name	Company Name
0	2022-09-29 23:41:16+00:00	Mainstream media has done an amazing job at br...	TSLA	Tesla, Inc.
1	2022-09-29 23:24:43+00:00	Tesla delivery estimates are at around 364k fr...	TSLA	Tesla, Inc.
2	2022-09-29 23:18:08+00:00	3/ Even if I include 63.0M unvested RSUs as of...	TSLA	Tesla, Inc.
3	2022-09-29 22:40:07+00:00	@RealDanODowd @WholeMarsBlog @Tesla Hahaha why...	TSLA	Tesla, Inc.
	2022-09-29	@RealDanODowd @Tesla Stop		

Next steps: Generate code with all_tweets | View recommended plots | New interactive sheet

```
df = all_tweets[all_tweets['Stock Name'] == stock_name]
print(df.shape)
df.head()
```

```
(4089, 4)
```

	Date	Tweet	Stock Name	Company Name
48351	2022-09-29 22:40:47+00:00	A group of lawmakers led by Sen. Elizabeth War...	AMZN	Amazon.com, Inc.
48352	2022-09-29 22:23:54+00:00	$NIO just because I'm down money doesn't mean ...	AMZN	Amazon.com, Inc.
48353	2022-09-29 18:34:51+00:00	Today's drop in $SPX is a perfect example of w...	AMZN	Amazon.com, Inc.
48354	2022-09-29 15:57:59+00:00	Druckenmiller owned $CVNA this year \nMunger b...	AMZN	Amazon.com, Inc.
	2022-09-29	Top 10 $QQQ Holdings		Amazon.com.

```
sent_df = df.copy()
sent_df["sentiment_score"] = ''
sent_df["Negative"] = ''
sent_df["Neutral"] = ''
sent_df["Positive"] = ''
sent_df.head()
```

	Date	Tweet	Stock Name	Company Name	sentiment_score	Negativ
48351	2022-09-29 22:40:47+00:00	A group of lawmakers led by Sen. Elizabeth War...	AMZN	Amazon.com, Inc.		
48352	2022-09-29 22:23:54+00:00	$NIO just because I'm down money doesn't mean ...	AMZN	Amazon.com, Inc.		
48353	2022-09-29 18:34:51+00:00	Today's drop in $SPX is a perfect example of w...	AMZN	Amazon.com, Inc.		
48354	2022-09-29 15:57:59+00:00	Druckenmiller owned $CVNA this year \nMunger b...	AMZN	Amazon.com, Inc.		
48355	2022-09-29 15:10:30+00:00	Top 10 $QQQ Holdings \n\nAnd Credit Rating\n\n...	AMZN	Amazon.com, Inc.		

To get sentiment (polarity) scores, we use **VADER (Valence Aware Dictionary for Sentiment Reasoning)** model. VADER is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is available in the NLTK package and can be applied directly to unlabeled text data.

VADER sentimental analysis relies on a dictionary that maps lexical features to emotion intensities known as sentiment scores. The sentiment score of a text can be obtained by summing up the intensity of each word in the text.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import unicodedata

# Download the VADER lexicon if it's not already downloaded
nltk.download('vader_lexicon')

sentiment_analyzer = SentimentIntensityAnalyzer()

# Use items() instead of iteritems()
for indx, row in sent_df.T.items():
    try:
        sentence_i = unicodedata.normalize('NFKD', sent_df.loc[indx, 'Tweet'])
        sentence_sentiment = sentiment_analyzer.polarity_scores(sentence_i)
        sent_df.at[indx, 'sentiment_score'] = sentence_sentiment['compound']
        sent_df.at[indx, 'Negative'] = sentence_sentiment['neg']
        sent_df.at[indx, 'Neutral'] = sentence_sentiment['neu']
        sent_df.at[indx, 'Positive'] = sentence_sentiment['pos']
    except TypeError:
        print(sent_df.loc[indx, 'Tweet'])
        print(indx)
        break
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
sent_df.head()
```

	Date	Tweet	Stock Name	Company Name	sentiment_score	Negativ
48351	2022-09-29 22:40:47+00:00	A group of lawmakers led by Sen. Elizabeth War...	AMZN	Amazon.com, Inc.	-0.0772	0.08
48352	2022-09-29 22:23:54+00:00	$NIO just because I'm down money doesn't mean ...	AMZN	Amazon.com, Inc.	0.25	0.15
48353	2022-09-29 18:34:51+00:00	Today's drop in $SPX is a perfect example of w...	AMZN	Amazon.com, Inc.	-0.3182	0.16
48354	2022-09-29 15:57:59+00:00	Druckenmiller owned $CVNA this year \nMunger b...	AMZN	Amazon.com, Inc.	0.2382	0.06
48355	2022-09-29 15:10:30+00:00	Top 10 $QQQ Holdings \n\nAnd Credit Rating\n\n...	AMZN	Amazon.com, Inc.	0.7783	0.(

```
sent_df['Date'] = pd.to_datetime(sent_df['Date'])
sent_df['Date'] = sent_df['Date'].dt.date
sent_df = sent_df.drop(columns=['Negative', 'Positive', 'Neutral', 'Stock Name', 'Co
```

```
sent_df.head()
```

	Date	Tweet	sentiment_score
48351	2022-09-29	A group of lawmakers led by Sen. Elizabeth War...	-0.0772
48352	2022-09-29	$NIO just because I'm down money doesn't mean ...	0.25
48353	2022-09-29	Today's drop in $SPX is a perfect example of w...	-0.3182

```
twitter_df = sent_df.groupby([sent_df['Date']])['sentiment_score'].mean()
print(twitter_df.shape)
```

```
(365,)
```

As the result of sentiment analysis we get average polarity scores of all tweets about a cartain stock ticker for each day:

```
twitter_df.head()
```

Date	sentiment_score
2021-09-30	0.24648
2021-10-01	0.359338
2021-10-02	-0.0007
2021-10-03	0.8344
2021-10-04	0.25865

Get final dataset for training

```
all_stocks = pd.read_csv('/content/stock_yfinance_data.csv')
print(all_stocks.shape)
all_stocks.head()
```

(6300, 8)

	Date	Open	High	Low	Close	Adj Close	Volume	St N
0	2021-09-30	260.333344	263.043335	258.333344	258.493347	258.493347	53868000	TS
1	2021-10-01	259.466675	260.260010	254.529999	258.406677	258.406677	51094200	TS
2	2021-10-04	265.500000	268.989990	258.706665	260.510010	260.510010	91449900	TS
3	2021-10-05	261.600006	265.769989	258.066681	260.196655	260.196655	55297800	TS
4	2021-10-06	258.733337	262.220001	257.739990	260.916656	260.916656	43898400	TS

Next steps: Generate code with all_stocks | View recommended plots | New interactive sheet

```
stock_df = all_stocks[all_stocks['Stock Name'] == stock_name]
stock_df['Date'] = pd.to_datetime(stock_df['Date'])
stock_df['Date'] = stock_df['Date'].dt.date
```

```
final_df = stock_df.join(twitter_df, how="left", on="Date")
final_df = final_df.drop(columns=['Stock Name'])
print(final_df.shape)
```

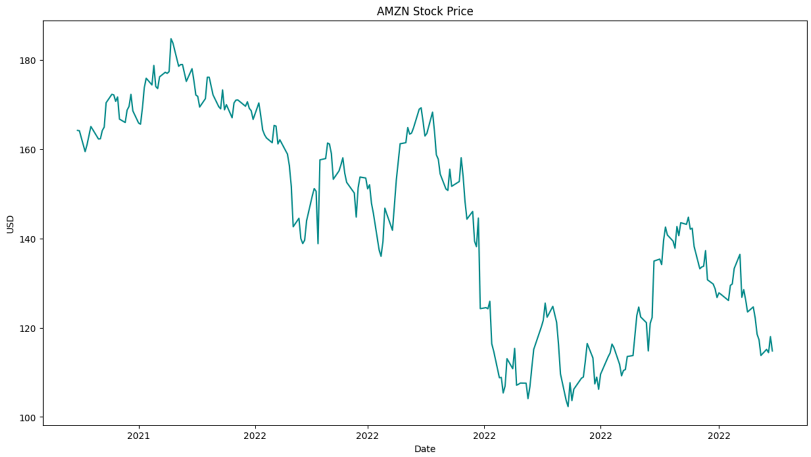
(252, 8)

```
final_df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
1008	2021-09-30	165.800003	166.392502	163.699493	164.251999	164.251999	56848000
1009	2021-10-01	164.450500	165.458496	162.796997	164.162994	164.162994	56712000
1010	2021-10-04	163.969498	163.999496	158.812500	159.488998	159.488998	90462000
1011	2021-10-05	160.225006	163.036499	160.123001	161.050003	161.050003	65384000
1012	2021-10-06	160.676498	163.216995	159.931000	163.100494	163.100494	50660000

Let's plot historical price data for the analyzed stock ticker:

```
fig, ax = plt.subplots(figsize=(15,8))
ax.plot(final_df['Date'], final_df['Close'], color='#008B8B')
ax.set(xlabel="Date", ylabel="USD", title=f"{stock_name} Stock Price")
ax.xaxis.set_major_formatter(DateFormatter("%Y"))
plt.show()
```



Adding technical indicators

To help the network understand the bigger picture of the market we add different technical indicators to the training data, such as moving averages, Bollinger bands etc., which describe the development of stock price not only for the current day, but for the past week or more.

MA(7) stans for Moving Average for past 7 days, whereas **MA(20)** means Moving Average for past 20 days.

EMA is Exponential Moving average and we can calculate it as:

- *EMA_t = Pclose + (EMA_t-1 * (100 - P))*

Bollinger Bands are calculated as:

- middle line: *stdev(MA(20))*
- upper bound: *MA(20) + 2stdev(MA(20))*
- lower bound: *MA(20) - 2stdev(MA(20))*

```
def get_tech_ind(data):
    data['MA7'] = data.iloc[:,4].rolling(window=7).mean() #Close column
    data['MA20'] = data.iloc[:,4].rolling(window=20).mean() #Close Column

    data['MACD'] = data.iloc[:,4].ewm(span=26).mean() - data.iloc[:,1].ewm(span=12,a
    #This is the difference of Closing price and Opening Price

    # Create Bollinger Bands
    data['20SD'] = data.iloc[:, 4].rolling(20).std()
    data['upper_band'] = data['MA20'] + (data['20SD'] * 2)
    data['lower_band'] = data['MA20'] - (data['20SD'] * 2)

    # Create Exponential moving average
    data['EMA'] = data.iloc[:,4].ewm(com=0.5).mean()

    # Create LogMomentum
    data['logmomentum'] = np.log(data.iloc[:,4] - 1)

    return data


tech_df = get_tech_ind(final_df)
dataset = tech_df.iloc[20:,:].reset_index(drop=True)
dataset.head()
```

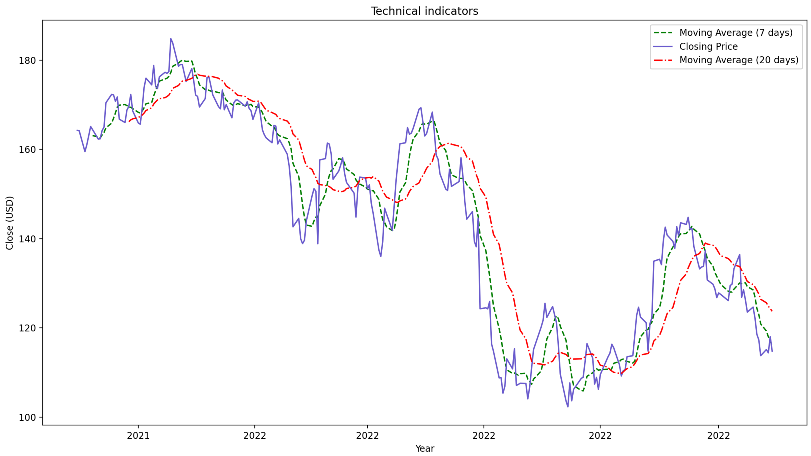
	Date	Open	High	Low	Close	Adj Close	Volume	s
0	2021-10-28	170.104996	173.949997	169.300003	172.328506	172.328506	114174000	
1	2021-10-29	165.001007	168.740997	163.666000	168.621506	168.621506	129722000	
2	2021-11-01	168.089996	168.792999	164.600998	165.905502	165.905502	72178000	
3	2021-11-02	165.750504	166.556000	164.177505	165.637497	165.637497	52552000	
4	2021-11-03	165.449997	169.746002	164.876007	169.199997	169.199997	67944000	

Next steps: Generate code with dataset | View recommended plots | New interactive sheet

```
def tech_ind(dataset):
    fig,ax = plt.subplots(figsize=(15, 8), dpi = 200)
    x_ = range(3, dataset.shape[0])
    x_ = list(dataset.index)

    ax.plot(dataset['Date'], dataset['MA7'], label='Moving Average (7 days)', color=
    ax.plot(dataset['Date'], dataset['Close'], label='Closing Price', color='#6A5ACD
    ax.plot(dataset['Date'], dataset['MA20'], label='Moving Average (20 days)', colo
    ax.xaxis.set_major_formatter(DateFormatter("%Y"))
    plt.title('Technical indicators')
    plt.ylabel('Close (USD)')
    plt.xlabel("Year")
    plt.legend()

    plt.show()
```

Let's plot Moving Averages for our data:

```
tech_ind(tech_df)
```




```
dataset.iloc[:, 1:] = pd.concat([dataset.iloc[:, 1:].ffill()])
```

```
datetime_series = pd.to_datetime(dataset['Date'])
datetime_index = pd.DatetimeIndex(datetime_series.values)
dataset = dataset.set_index(datetime_index)
dataset = dataset.sort_values(by='Date')
dataset = dataset.drop(columns='Date')
```

```
def normalize_data(df, range, target_column):

    '''
    df: dataframe object
    range: type tuple -> (lower_bound, upper_bound)
        lower_bound: int
        upper_bound: int
    target_column: type str -> should reflect closing price of stock
    '''
```

```
    target_df_series = pd.DataFrame(df[target_column])
    data = pd.DataFrame(df.iloc[:, :])

    X_scaler = MinMaxScaler(feature_range=range)
    y_scaler = MinMaxScaler(feature_range=range)
    X_scaler.fit(data)
    y_scaler.fit(target_df_series)

    X_scale_dataset = X_scaler.fit_transform(data)
    y_scale_dataset = y_scaler.fit_transform(target_df_series)

    dump(X_scaler, open('X_scaler.pkl', 'wb'))
    dump(y_scaler, open('y_scaler.pkl', 'wb'))

    return (X_scale_dataset,y_scale_dataset)


def batch_data(x_data,y_data, batch_size, predict_period):
    X_batched, y_batched, yc = list(), list(), list()

    for i in range(0,len(x_data),1):
        x_value = x_data[i: i + batch_size][:, :]
        y_value = y_data[i + batch_size: i + batch_size + predict_period][:, 0]
        yc_value = y_data[i: i + batch_size][:, :]
        if len(x_value) == batch_size and len(y_value) == predict_period:
            X_batched.append(x_value)
            y_batched.append(y_value)
            yc.append(yc_value)

    return np.array(X_batched), np.array(y_batched), np.array(yc)


def split_train_test(data):
    train_size = len(data) - 20
    data_train = data[0:train_size]
    data_test = data[train_size:]
    return data_train, data_test


def predict_index(dataset, X_train, batch_size, prediction_period):

    # get the predict data (remove the in_steps days)
    train_predict_index = dataset.iloc[batch_size: X_train.shape[0] + batch_size + p
    test_predict_index = dataset.iloc[X_train.shape[0] + batch_size:, :].index

    return train_predict_index, test_predict_index


X_scale_dataset,y_scale_dataset = normalize_data(dataset, (-1,1), "Close")
X_batched, y_batched, yc = batch_data(X_scale_dataset, y_scale_dataset, batch_size =
print("X shape:", X_batched.shape)
print("y shape:", y_batched.shape)
print("yc shape:", yc.shape)

X_train, X_test, = split_train_test(X_batched)
y_train, y_test, = split_train_test(y_batched)
yc_train, yc_test, = split_train_test(yc)
index_train, index_test, = predict_index(dataset, X_train, 5, 1)
```

```
X shape: (227, 5, 15)
y shape: (227, 1)
yc shape: (227, 5, 1)
```

```
input_dim = X_train.shape[1]
feature_size = X_train.shape[2]
output_dim = y_train.shape[1]
```

Build GAN model

In this notebook we build a GAN model architecture, where the generator has 5 LSTM blocks and the discriminator has 5 convolutional and 3 dense layers with sigmoid activation function.

Generator model sructure looks like this: Generator model

```
def make_generator_model(input_dim, output_dim, feature_size):
    model = tf.keras.Sequential([LSTM(units = 1024, return_sequences = True,
                                input_shape=(input_dim, feature_size),recurrent_
                            LSTM(units = 512, return_sequences = True, recurrent_
                            LSTM(units = 256, return_sequences = True, recurrent_
                            LSTM(units = 128, return_sequences = True, recurrent_
                            LSTM(units = 64, recurrent_dropout = 0.3),
                            Dense(32),
                            Dense(16),
                            Dense(8),
                            Dense(units=output_dim)])
    return model
```

Discriminator model sructure looks like this: Discriminator model

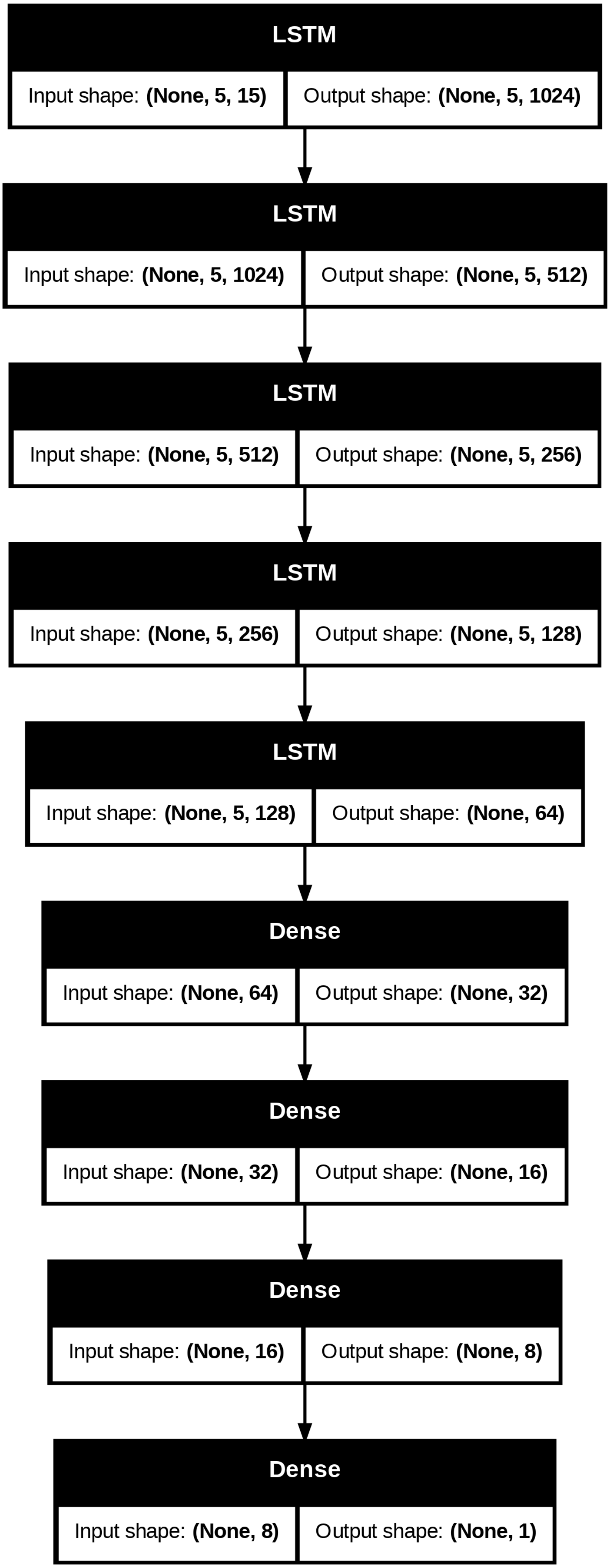
```
def make_discriminator_model(input_dim):
    cnn_net = tf.keras.Sequential()
    cnn_net.add(Conv1D(8, input_shape=(input_dim+1, 1), kernel_size=3, strides=2, pa
    cnn_net.add(Conv1D(16, kernel_size=3, strides=2, padding='same', activation=Leak
    cnn_net.add(Conv1D(32, kernel_size=3, strides=2, padding='same', activation=Leak
    cnn_net.add(Conv1D(64, kernel_size=3, strides=2, padding='same', activation=Leak
    cnn_net.add(Conv1D(128, kernel_size=1, strides=2, padding='same', activation=Lea
    #cnn_net.add(Flatten())
    cnn_net.add(LeakyReLU())
    cnn_net.add(Dense(220, use_bias=False))
    cnn_net.add(LeakyReLU())
    cnn_net.add(Dense(220, use_bias=False, activation='relu'))
    cnn_net.add(Dense(1, activation='sigmoid'))
    return cnn_net
```

Now we define loss functions for our models. We will use BinaryCrossEntropy loss for both models:

```
def discriminator_loss(real_output, fake_output):
    loss_f = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss = loss_f(tf.ones_like(real_output), real_output)
    fake_loss = loss_f(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    loss_f = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    loss = loss_f(tf.ones_like(fake_output), fake_output)
    return loss


@tf.function

def train_step(real_x, real_y, yc, generator, discriminator, g_optimizer, d_optimize
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_data = generator(real_x, training=True)
        generated_data_reshape = tf.reshape(generated_data, [generated_data.shape[0]
        d_fake_input = tf.concat([tf.cast(generated_data_reshape, tf.float64), yc],
        real_y_reshape = tf.reshape(real_y, [real_y.shape[0], real_y.shape[1], 1])
        d_real_input = tf.concat([real_y_reshape, yc], axis=1)

        real_output = discriminator(d_real_input, training=True)
        fake_output = discriminator(d_fake_input, training=True)

        g_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(g_loss, generator.trainable_variables
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainab

    g_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_vari
    d_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.traina

    return real_y, generated_data, {'d_loss': disc_loss, 'g_loss': g_loss}


def train(real_x, real_y, yc, Epochs, generator, discriminator, g_optimizer, d_optim
    train_info = {}
    train_info["discriminator_loss"] = []
    train_info["generator_loss"] = []

    for epoch in tqdm(range(Epochs)):
        real_price, fake_price, loss = train_step(real_x, real_y, yc, generator, dis
        G_losses = []
```

```
        D_losses = []
        Real_price = []
        Predicted_price = []
        D_losses.append(loss['d_loss'].numpy())
        G_losses.append(loss['g_loss'].numpy())
        Predicted_price.append(fake_price.numpy())
        Real_price.append(real_price.numpy())

        #Save model every X checkpoints
        if (epoch + 1) % checkpoint == 0:
            tf.keras.models.save_model(generator, f'./models_gan/{stock_name}/genera
            tf.keras.models.save_model(discriminator, f'./models_gan/{stock_name}/di
            print('epoch', epoch + 1, 'discriminator_loss', loss['d_loss'].numpy(),

        train_info["discriminator_loss"].append(D_losses)
        train_info["generator_loss"].append(G_losses)

    Predicted_price = np.array(Predicted_price)
    Predicted_price = Predicted_price.reshape(Predicted_price.shape[1], Predicted_pr
    Real_price = np.array(Real_price)
    Real_price = Real_price.reshape(Real_price.shape[1], Real_price.shape[2])

    plt.subplot(2,1,1)
    plt.plot(train_info["discriminator_loss"], label='Disc_loss', color='#000000')
    plt.xlabel('Epoch')
    plt.ylabel('Discriminator Loss')
    plt.legend()

    plt.subplot(2,1,2)
    plt.plot(train_info["generator_loss"], label='Gen_loss', color='#000000')
    plt.xlabel('Epoch')
    plt.ylabel('Generator Loss')
    plt.legend()

    plt.show()

    return Predicted_price, Real_price, np.sqrt(mean_squared_error(Real_price, Predi


def plot_results(Real_price, Predicted_price, index_train):
    X_scaler = load(open('/content/X_scaler.pkl', 'rb'))
    y_scaler = load(open('/content/y_scaler.pkl', 'rb'))
    train_predict_index = index_train

    rescaled_Real_price = y_scaler.inverse_transform(Real_price)
    rescaled_Predicted_price = y_scaler.inverse_transform(Predicted_price)

    predict_result = pd.DataFrame()
    for i in range(rescaled_Predicted_price.shape[0]):
        y_predict = pd.DataFrame(rescaled_Predicted_price[i], columns=["predicted_pr
        predict_result = pd.concat([predict_result, y_predict], axis=1, sort=False)

    real_price = pd.DataFrame()
    for i in range(rescaled_Real_price.shape[0]):
        y_train = pd.DataFrame(rescaled_Real_price[i], columns=["real_price"], index
        real_price = pd.concat([real_price, y_train], axis=1, sort=False)

    predict_result['predicted_mean'] = predict_result.mean(axis=1)
    real_price['real_mean'] = real_price.mean(axis=1)

    plt.figure(figsize=(16, 8))
    plt.plot(real_price["real_mean"])
    plt.plot(predict_result["predicted_mean"], color = 'r')
    plt.xlabel("Date")
    plt.ylabel("Stock price")
    plt.legend(("Real price", "Predicted price"), loc="upper left", fontsize=16)
    plt.title("The result of Training", fontsize=20)
    plt.show()

    predicted = predict_result["predicted_mean"]
    real = real_price["real_mean"]
    For_MSE = pd.concat([predicted, real], axis = 1)
    RMSE = np.sqrt(mean_squared_error(predicted, real))
    print('-- Train RMSE -- ', RMSE)


## Test Code

@tf.function

def eval_op(generator, real_x):
    generated_data = generator(real_x, training = False)
```

```
    return generated_data


def plot_test_data(Real_test_price, Predicted_test_price, index_test):
    X_scaler = load(open('X_scaler.pkl', 'rb'))
    y_scaler = load(open('y_scaler.pkl', 'rb'))
    test_predict_index = index_test

    rescaled_Real_price = y_scaler.inverse_transform(Real_test_price)
    rescaled_Predicted_price = y_scaler.inverse_transform(Predicted_test_price)

    predict_result = pd.DataFrame()
    for i in range(rescaled_Predicted_price.shape[0]):
        y_predict = pd.DataFrame(rescaled_Predicted_price[i], columns=["predicted_pr
        predict_result = pd.concat([predict_result, y_predict], axis=1, sort=False)

    real_price = pd.DataFrame()
    for i in range(rescaled_Real_price.shape[0]):
        y_train = pd.DataFrame(rescaled_Real_price[i], columns=["real_price"], index
        real_price = pd.concat([real_price, y_train], axis=1, sort=False)

    predict_result['predicted_mean'] = predict_result.mean(axis=1)
    real_price['real_mean'] = real_price.mean(axis=1)

    predicted = predict_result["predicted_mean"]
    real = real_price["real_mean"]
    For_MSE = pd.concat([predicted, real], axis = 1)
    RMSE = np.sqrt(mean_squared_error(predicted, real))
    print('Test RMSE: ', RMSE)

    plt.figure(figsize=(16, 8))
    plt.plot(real_price["real_mean"], color='#00008B')
    plt.plot(predict_result["predicted_mean"], color = '#8B0000', linestyle='--')
    plt.xlabel("Date")
    plt.ylabel("Stock price")
    plt.legend(("Real price", "Predicted price"), loc="upper left", fontsize=16)
    plt.title(f"Prediction on test data for {stock_name}", fontsize=20)
    plt.show()



learning_rate = 5e-4
epochs = 500

# Change 'lr' to 'learning_rate'
g_optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
d_optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

generator = make_generator_model(X_train.shape[1], output_dim, X_train.shape[2])
discriminator = make_discriminator_model(X_train.shape[1])


plot_model(generator, to_file='generator_keras_model.png', show_shapes=True)
```


LSTM
Input shape: (None, 5, 15)
Output shape: (None, 5, 1024)
LSTM
Input shape: (None, 5, 1024)
Output shape: (None, 5, 512)
LSTM
Input shape: (None, 5, 512)
Output shape: (None, 5, 256)
LSTM
Input shape: (None, 5, 256)
Output shape: (None, 5, 128)
LSTM
Input shape: (None, 5, 128)
Output shape: (None, 64)
Dense
Input shape: (None, 64)
Output shape: (None, 32)
Dense
Input shape: (None, 32)
Output shape: (None, 16)
Dense
Input shape: (None, 16)
Output shape: (None, 8)
Dense
Input shape: (None, 8)
Output shape: (None, 1)

```
tf.keras.utils.plot_model(discriminator, to_file='discriminator_keras_model.png', sh
```