UDC 004.415.2
DOI https://doi.org/10.32782/2663-5941/2024.3.1/24

Oleshchenko L.M.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Zheng Jinsong
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

LSTM NEURAL NETWORK IMPLEMENTATION FOR THE STOCK MARKET OPERATIONS PREDICTING

The stock market is an essential part of the modern financial system. Analyzing and forecasting trends in the stock market can effectively help investors avoid investment risks and increase profits. The stock index reflects changes in the stock market and can indicate market trends. Time series forecasting in the stock market is a relevant research direction among investors and researchers. Traditional statistical models are widely used in time series forecasting tasks of stock market data. Due to the nonlinear characteristics of time series data, traditional statistical models have certain limitations and forecasting problems. Machine learning models can effectively address nonlinear problems in time series data. With the development of machine learning models, deep learning models such as artificial neural networks have begun to be widely used in forecasting time series for stock market data.

The article analyzes the use of recurrent neural networks for forecasting operations in the stock market. The main advantages and disadvantages of existing solutions are provided. The proposed software architecture and key technologies for forecasting operations in the stock market are described, including the description of the developed software, its main modules, and components. The functional capabilities of the developed software product are outlined. The effectiveness of the proposed software method using LSTM neural network for forecasting operations in the stock market is analyzed.

The proposed software implements the LSTM neural network training method to improve the accuracy of forecasting financial operations in the stock market. According to the conducted research, the use of LSTM can increase the accuracy of forecasting operations in the stock market by up in average to 15 % compared to using the ARMA model. The developed software in the Python programming language, along with machine learning libraries and modules, allows obtaining relatively accurate forecasting results compared to using known statistical methods.

Key words: *software, neural networks, ARMA, ANN, RNN, LSTM, Python, TensorFlow, stock market operations forecasting.*

Problem statement. Stocks, originating from the Dutch East India Company in 1611, symbolize ownership in companies and are key assets for investors, offering both high risks and potential returns. Stock markets worldwide generate vast data daily, crucial for investors' decision-making. However, predicting stock prices is complex due to factors like noise, non-linearity, and unpredictable investor behavior.

Challenges in forecasting include noise in data, nonlinearity, overfitting, dynamic market conditions, and the need for advanced algorithms. Adapting to evolving market dynamics, limited historical data availability, model complexity, and external factors like unethical practices or global events further complicate predictions. Overcoming these challenges necessitates advanced modeling, rigorous data preprocessing, continuous refinement, and deep market understanding.

The main goal of the article is creating software to predict stock market prices and obtain more accurate results.

Related research. The article [1] explores the use of LSTM networks to predict future stock price trends based on historical prices and technical indicators, achieving promising results in predicting stock price movements. The paper [2] proposes an LSTM-based RNN for forecasting future values of GOOGL and NKE assets, showing promising results in tracing opening prices, with future work aimed at optimizing data length and training epochs to improve prediction accuracy. The article [3] concludes that the CEEMDAN-LSTM mixture model optimally forecasts stock index prices for both the S&P 500 and CSI 300 indices, outperforming other models such as SVM, BP, Elman network, and WAV combined with CEEMDAN. Stock market prediction is a complex task in financial

research, tackled through various methodologies like fundamental and technical analysis. Fundamental analysis examines economic, political, company-specific, and psychological factors influencing stock prices. Economic indicators and political stability impact stock markets significantly, while company profitability and industry development also play crucial roles. Technical analysis relies on quantitative indicators like price and volume, emphasizing market rules and short-term trends. Investor behavior, influenced by herd mentality and reflexivity theory, further complicates predictions. Manipulation by market makers, bubbles, and panic selling add layers of complexity. Historical data analysis is vital for understanding market trends and patterns, guiding accurate predictions [4–7].

Existing software solutions. Stock market technical methods are broadly categorized into traditional statistical and machine learning approaches. Time series data, organized in chronological order, is fundamental for time series forecasting, which aims to predict future trends based on historical data. Traditional statistical methods like ARMA and ARIMA are well-established, while machine learning methods like support vector machines (SVM) and neural networks are increasingly popular due to their effectiveness and generalization capabilities. Statistical prediction models encompass various techniques tailored for different types of variables, such as categorical, ordinal, interval, and ratio variables. Regression analysis, moving average, cyclical change analysis, and seasonal change analysis are common statistical methods used for stock market prediction. These methods aim to model and forecast stock price trends, but each has its limitations and challenges, such as the need for data preprocessing and the suitability for non-linear time series like stock prices [8].

Machine learning models excel in handling non-linear stock data compared to traditional statistical methods. As a rapidly growing branch of artificial intelligence, machine learning has evolved significantly from symbolic learning to statistical learning. It focuses on enhancing system performance across various fields like medicine, biology, and economics. Different machine learning algorithms are employed based on the research objectives, such as decision trees and Bayesian classifiers for classification tasks and linear regression and neural networks for regression tasks. Additionally, machine learning algorithms are categorized into supervised learning, which uses labeled training data, and unsupervised learning, which doesn't require labeled information, with examples including classification and regression for supervised learning and clustering for unsupervised learning [9].

LSTM networks excel in predicting stock market operations due to their unique ability to capture long-term dependencies in sequential data. Unlike traditional neural networks, LSTMs can effectively model complex and non-linear patterns over extended periods, making them well-suited for financial time series analysis. Analyzing stock market transactions provides valuable insights into market trends and investor sentiment, with the increasing volume of electronic trades generating substantial big data. LSTM networks offer promising prospects for accurately predicting stock market movements by leveraging their superior performance in handling temporal dynamics and incorporating historical information. As financial markets evolve, LSTM networks continue to demonstrate their effectiveness in forecasting stock market behavior, making them a preferred choice for researchers and practitioners seeking robust predictions. Ongoing advancements in deep learning further enhance the capabilities of LSTM networks, ensuring their relevance in stock market forecasting [10].

Decision support methods in stock market operations aid investors, traders, and financial professionals in making informed decisions. They encompass fundamental analysis, technical analysis, algorithmic trading, and sentiment analysis.

Fundamental analysis evaluates a company's financial health and intrinsic value, while technical analysis forecasts price movements using historical data and statistical tools. Algorithmic trading automates trading decisions using computer algorithms based on various strategies. Sentiment analysis assesses market sentiment by analyzing news, social media, and other sources, employing NLP and machine learning. Implementing NLP-driven insights into trading systems enhances decision-making, providing actionable information for investors.

Machine learning models, like neural networks, SVM, and decision trees, are increasingly used for predicting stock prices. These methods, often used together, depend on investor preferences and market conditions, with technology advancements continuously enhancing decision support tools.

Data sources and data preprocessing. In empirical research, data quality significantly impacts results. Data collection methods include statistical surveys and scientific empirical methods. Authenticity is ensured by sourcing data from Yahoo Finance.

This research uses five years of real historical time series data from NASDAQ Index ETF, S&P 500 Index ETF, and DOW JONES Index ETF, from September 1st, 2018, to September 1st, 2023. Data preprocessing involves reviewing data for completeness and accuracy, dealing with missing values, and standardizing data.

If too many missing values exist, a threshold can be set, and attributes exceeding it may be deleted. Alternatively, missing numeric values can be filled using averages, while non-numeric ones can be filled with the mode. Linear interpolation can fill missing values based on fitted linear relationships, but it may not be suitable for data at the beginning or end of a sample, where alternative methods like mean filling may be considered.

Data standardization aims to clarify variable relationships and enhance comparability by scaling data within a specific interval. Common methods include min-max, *log* function, *atan* function, and *z*-score standardization. Each method affects model results differently, and selection depends on data and model conditions. Standardization benefits model convergence speed and accuracy.

This research adjusts data to the range of 0 to 1 using standardization due to the sensitivity of the deep learning model framework to input data. The most common method is min-max standardization, which linearly transforms data to fit within [0,1], retaining original data correlations but with some limitations.

If new original data needs to be added in the experiment, this method has a certain probability of causing its maximum and minimum values to change, so it needs to be re-perform definitions and calculations:

$$y_i = \frac{x_i - \min\limits_{1\le j\le n}\{x_j\}}{\max\limits_{1\le j\le n}\{x_i\} - \min\limits_{1\le j\le n}\{x_j\}}, \quad (1)$$

where max is the maximum value in the sample data, and min is the minimum value in the sample data.

Log function conversion method uses the log function with base 10 to transform data, with the goal of achieving normalization. This method is one of the most commonly used methods in linear regression and data regression research. The specific formula is as shown below:

$$X = \log_{10}(x). \quad (2)$$

In the *z*-score standardization method, if all data do not need to be mapped to the interval [0,1], then the most common method is *z*-score standardization, also called standard deviation standardization. The formula is shown below.

$$y_i = \frac{x_i - \bar{x}}{s}, \quad (3)$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad (4)$$

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}. \quad (5)$$

This method normalizes the original data to new data using *z*-score. After converting the original sequence, the new sequence $y_1, y_2, \ldots, y_n$ formed after $x_1, x_2, \ldots, x_n$ changes has a mean of 0, a variance of 1, and no dimension. This is because $x-\bar{x}$ only changes the mean of the data, while the standard deviation remains the same. Logisitic transformation uses the logistic function to transform data. The function of the logistic function is to make the data tend to 0 in the interval from negative infinity to 0, and tend to 1 in the interval from 0 to positive infinity. The logisitic function is as shown in the following formula.

$$y_i = \frac{1}{\left(1 + e^{-x_i}\right)}. \quad (6)$$

In this research, the data is standardized by using the "min-max data standardization method" in the data preprocessing stage.

The LSTM deep learning model framework in this study requires standardized data for accurate predictions. Python 3.11.5 is utilized, employing the *MinMaxScaler* preprocessing class from the Scikit-learn library for data normalization. Python offers versatile tools for data handling, including Pandas for efficient data manipulation and Scikit-learn for scaling and preprocessing tasks. These libraries empower data scientists to seamlessly explore and prepare datasets, facilitating subsequent analysis and machine learning tasks.

Proposed software method. In the research we build LSTM model to predict stock price of sample data. The LSTM model is mainly designed to solve the problems of gradient disappearance and gradient explosion during long sequence training. It is so far the best model for time series prediction among deep learning methods (Fig. 1).

We build the LSTM model framework based on the LSTM layer, Dropout layer and Dense layer in the neural network. In the experiment of this research, the Keras deep learning framework is used to build the framework.

First, we use the Keras deep learning framework to build a Sequential model; secondly, add the LSTM layer, Dropout layer and Dense layer in the neural network to the model in sequence, and set

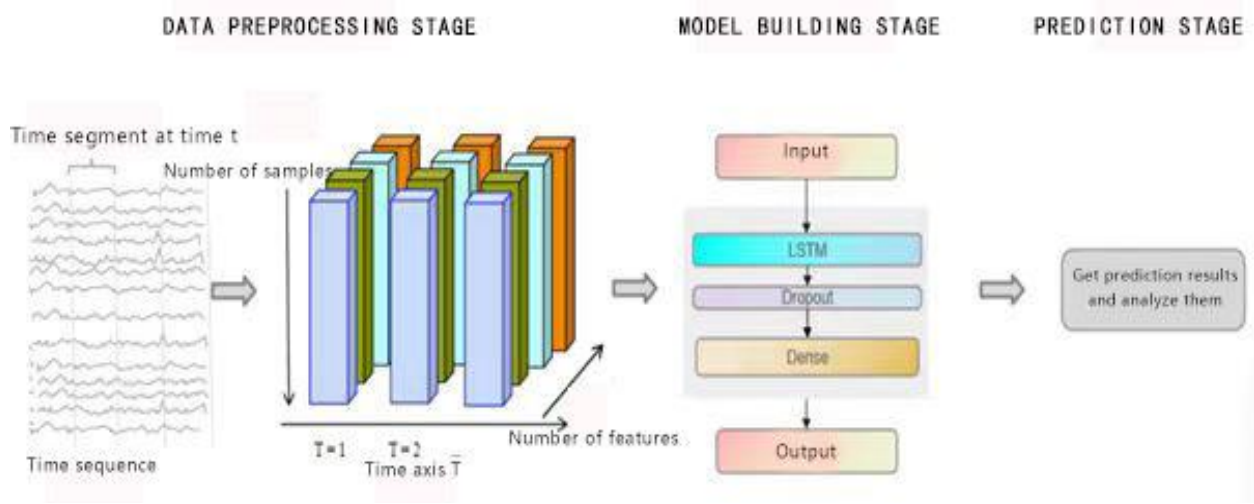

Fig. 1. LSTM model usage to predict stock price of sample data

the parameters of each layer. Among them, the activation function uses Relu function to optimize the framework. The algorithm adopts Adam algorithm.

The Keras sequential model allows us to specify a neural network, precisely, sequential: from input to output, passing through a series of neural layers, one after the other. TensorFlow also allows to use the functional API for building deep learning models. A Sequential model is appropriate for a plain stack of layers where each layer has one input tensor and one output tensor.

This research aims to build a deep learning model LSTM framework to predict time series data of Exchange-Traded Fund stock index (ETF). In order to effectively verify the accuracy of the framework established in this research in the index return prediction problem, this research selects three different stock index ETF sample data in the same time period to test the model, namely, the NASDAQ Index ETF, the S&P 500 Index ETF, and the Dow Jones Index ETF.

In order to better evaluate the prediction accuracy of each model, this research uses two evaluation indicators: MAE (mean absolute error) and RMSE (root mean square error) . The formulas of each evaluation index are as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|, \tag{7}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}, \tag{8}$$

where y_i represents the real value of the index ETF in the i-th period, $\hat{y}_i$ represents the predicted value of the index ETF in the i-th period.

RMSE is the operation of squaring and accumulating errors and then taking the square root. Therefore, RMSE actually magnifies the gap between errors, while MAE can reflect a more realistic error situation.

The results of the two evaluation indicators RMSE and MAE are that, the smaller the number, the better the prediction effect of the model and the higher the accuracy of the model.

This research selects five years NASDAQ Index ETF (ticker symbol: QQQ), S&P 500 Index ETF (ticker symbol: SPY), and DOW JONES Index ETF (ticker symbol: DIA) stock market closing prices time series data as sample data.

We divide the three sample data into a training set and a test set. The training set is used to learn the potential rules, the test set is to test the model trained on the training set in the new sample of the test set to judge the model's ability to discriminate new samples. The method of dividing the training set and the test set of the exponential sample in this research keeps consistent. The first 80 % of the time series data is used as the training set, and the last 20 % of the time series data is used as the test set.

In the model building stage, this research uses the Keras deep learning framework to build the LSTM model. The specific methods are as follows.

First, we build a Sequential model in the Keras deep learning framework.

Second, we add the necessary neural network layers to the Sequential model in sequence. These include LSTM model layer, Dropout layer and Dense layer.

In the hidden layer, there are a total of 64 neurons, the output layer is set to 1 neuron, and the input variables are features of one time step (t-1).

The default parameter settings are as follows: dropout_ratio in the Dropout layer is 0.5, the output space dimension of the Dense layer is 2, the input

space dimension of the LSTM layer is 100, the optimization function is Adam, and the loss function is Mean Absolute Error (MAE).

We import all the required libraries. yFinance is an open-source Python library that allows to acquire stock data from Yahoo Finance.

```
import math
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from tensorflow.keras import layers
```

NASDAQ Index ETFs replicate the NASDAQ stock market's performance, known for its focus on tech and growth companies. They offer diversified exposure to NASDAQ's stocks without buying individual ones. These ETFs, like Invesco QQQ Trust (ticker: QQQ), track NASDAQ indices and are traded on stock exchanges.

Similarly, S&P 500 Index ETFs, such as SPY, mirror the S&P 500's performance, representing 500 large-cap US companies. They provide diversified market exposure without buying individual stocks. DOW JONES INDEX ETFs, like DIA, mimic the Dow Jones Industrial Average's performance, consisting of 30 large US companies. They offer exposure to the DJIA's performance without buying its individual stocks.

In the research, we explore the relationship between length of time period and accuracy of prediction, and the relationship between data volatility and accuracy of prediction. Therefore, we use 6 months, 1 year, 3 years and 5 years four different time periods to test the relationship (Table 1).

The relationship between the length of time and prediction accuracy is nuanced and depends on various factors. Successful prediction strategies often involve a combination of models tailored to different time horizons, considering the specific characteristics and challenges associated with each. The choice of features, model architecture, and the incorporation of external factors play crucial roles in determining the effectiveness of predictions across different timeframes.

Research results. For the research we select five years (from September 1, 2018 to September 1, 2023) NASDAQ Index ETF (ticker symbol: QQQ), S&P 500 Index ETF (ticker symbol: SPY), and DOW JONES Index ETF (ticker symbol: DIA) stock market closing prices time series data as sample data to implement LSTM model. We use the yFinance method to acquire the stock data starting from September 1, 2018 to September 1, 2023 and then preview the data:

```
stock_data = yf.download('QQQ', start='2018-09-01', end='2023-09-01')
stock_data.head( )
```

Table 1

Research time periods to test the relationship between data volatility and accuracy of prediction

Stock symbol	Time length	Time range
NASDAQ INDEX ETF (QQQ)	6 months	01/03/2023-01/09/2023
	1 year	01/09/2022-01/09/2023
	3 years	01/09/2020-01/09/2023
	5 years	01/09/2018-01/09/2023
S&P 500 INDEX ETF (SPY)	6 months	01/03/2023-01/09/2023
	1 year	01/09/2022-01/09/2023
	3 years	01/09/2020-01/09/2023
	5 years	01/09/2018-01/09/2023
DOW JONES INDEX ETF (DIA)	6 months	01/03/2023-01/09/2023
	1 year	01/09/2022-01/09/2023
	3 years	01/09/2020-01/09/2023
	5 years	01/09/2018-01/09/2023

Code running result as below:

Date	Open	High	Low	Close	Adj Close	Volume
2018-09-04	186.080002	186.399994	184.850006	185.850006	179.448425	29063500
2018-09-05	185.520004	185.550003	182.820007	183.449997	177.131104	42623300
2018-09-06	183.529999	183.750000	180.580002	181.809998	175.547607	46091400
2018-09-07	180.500000	182.669998	180.440002	181.110001	174.871719	46629500
2018-09-10	182.149994	182.250000	180.729996	181.720001	175.460678	26132000

Date	Open	High	Low	Close	Adj Close	Volume
2018-09-04	186.080002	186.399994	184.850006	185.850006	179.448425	29063500
2018-09-05	185.520004	185.550003	182.820007	183.449997	177.131104	42623300
2018-09-06	183.529999	183.750000	180.580002	181.809998	175.547607	46091400
2018-09-07	180.500000	182.669998	180.440002	181.110001	174.871719	46629500
2018-09-10	182.149994	182.250000	180.729996	181.720001	175.460678	26132000

We set the plot figure size and title, use the Matplotlib plot method to create a line chart for historical close prices of QQQ, in the code lines 15–16 we set the x-axis and y-axis labels:

```
12  plt.figure(figsize=(15, 8))
13  plt.title('Stock Prices History')
14  plt.plot(stock_data['Close'])
15  plt.xlabel('Date')
16  plt.ylabel('Prices ($)')
```

Code running result as below (Fig. 2):

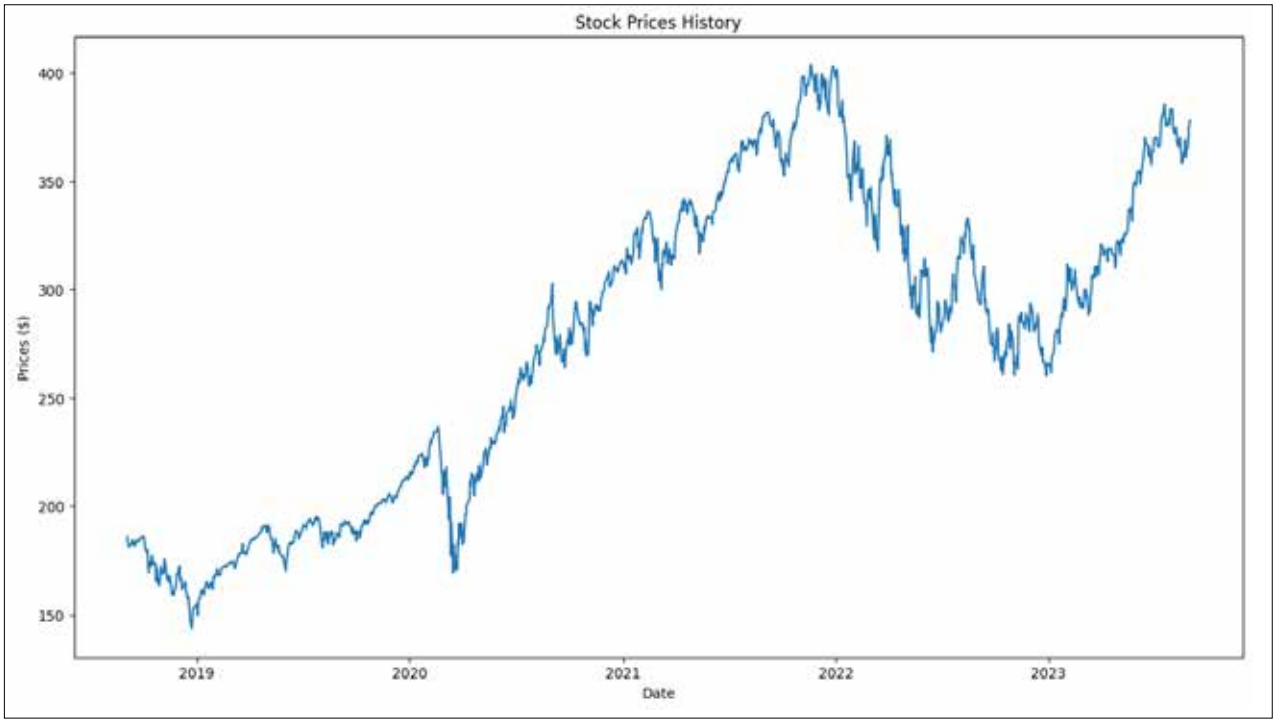

Fig. 2. The line chart for historical close prices of QQQ

We extract the closing prices from the acquired stock data and convert it to a number series, calculate the data size for 80 % of the dataset. The *math.ceil* method is to ensure the data size is rounded up to an integer. We use the Scikit-Learn MinMaxScaler to normalize all stock data ranging from 0 to 1. We also reshape our normalized data into a two-dimensional array:

```
17  close_prices = stock_data['Close']
18  values = close_prices.values
19  training_data_len = math.ceil(len(values)* 0.8)

20  scaler = MinMaxScaler(feature_range=(0,1))
21  scaled_data = scaler.fit_transform(values.reshape(-1,1))
22  train_data = scaled_data[0: training_data_len, :]

23  x_train = []
24  y_train = []

25  for i in range(60, len(train_data)):
26      x_train.append(train_data[i-60:i, 0])
27      y_train.append(train_data[i, 0])

28  x_train, y_train = np.array(x_train), np.array(y_train)
29  x_train = np.reshape(x_train, (x_train.shape[0],
      x_train.shape[1], 1))
```

In the code line 22 we set apart the first 80 % of the stock data as the training set. Code lines 23–24 create an empty list for a sequence of feature data (x_train) and a sequence of label data (y_train).

Code lines 25–27 create a 60-days window of historical prices (i-60) as our feature data (x_train) and the following 60-days window as label data (y_train).

Code lines 28–29 convert the feature data (x_train) and label data (y_train) into Numpy array as it is the data format accepted by the Tensorflow when training a neural network model. Reshape again the x_train and y_train into a three-dimensional array as part of the requirement to train a LSTM model.

Then we extract the closing prices from our normalized dataset (the last 20 % of the dataset):

```
30  test_data = scaled_data[training_data_len-60: , : ]
31  x_test = []
32  y_test = values[training_data_len:]

33  for i in range(60, len(test_data)):
34    x_test.append(test_data[i-60:i, 0])

35  x_test = np.array(x_test)
36  x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1],
1))
```

Code lines 31–34: similar to the training set, we will have to create feature data (x_test) and label data (y_test) from our test set.

Code lines 35–36 convert the feature data (x_test) and label data (y_test) into Numpy array. Reshape again the x_test and y_test into a three-dimensional array.

We define a Sequential model which consists of a linear stack of layers and add a LSTM layer by giving it 100 network units, set the return_sequence to true so that the output of the layer will be another sequence of the same length (code lines 37–38):

```
37  model = keras.Sequential()
38  model.add(layers.LSTM(100, return_sequences=True,
                  input_shape=(x_train.shape[1], 1)))
```

We add another LSTM layer with also 100 network units. But we set the return_sequence to false for this time to only return the last output in the output sequence length (code line 39):

```
39  model.add(layers.LSTM(100, return_sequences=False))
40  model.add(layers.Dense(25))
41  model.add(layers.Dense(1))
42  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 100)           40800

 lstm_1 (LSTM)               (None, 100)               80400

 dense (Dense)               (None, 25)                2525

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 123751 (483.40 KB)
Trainable params: 123751 (483.40 KB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 3. The summary of LSTM neural network

Code line 40 adds a densely connected neural network layer with 25 network units. Code line 41 at last, adds a densely connected layer that specifies the output of 1 network unit. Code line 42 shows the summary of our LSTM neural network. Code running result as below (Fig. 3).

Then we adopt Adam optimizer and set the mean square error as loss function and train the model by fitting it with the training set. We try with batch_size of 1 and run the training for 3 epochs (code lines 43–44):

```
43  model.compile(optimizer='adam', loss='mean_squared_error')
44  model.fit(x_train, y_train, batch_size= 1, epochs=3)
```

Code running result as below (Fig. 4).

```
Epoch 1/3
946/946 [==============================] - 40s 37ms/step - loss: 0.0035
Epoch 2/3
946/946 [==============================] - 36s 39ms/step - loss: 0.0015
Epoch 3/3
946/946 [==============================] - 35s 37ms/step - loss: 0.0012
<keras.src.callbacks.History at 0x781fac6ffeb0>
```

Fig. 4. The training for 3 epochs

We apply the model to predict the stock prices based on the test set and use the inverse_transform method to denormalize the predicted stock prices (code lines 45–46):

```
45  predictions = model.predict(x_test)
46  predictions = scaler.inverse_transform(predictions)
47  rmse = np.sqrt(np.mean(predictions - y_test)**2)
48  print("RMSE=", rmse)
49  mae = np.mean(np.abs(predictions - y_test))
50  print("MAE=", mae)
51  daily_volatility = np.std(predictions)
52  print("daily_volatility=", daily_volatility)
53  count = np.size(predictions)
54 print("count=", count)
55 volatility = daily_volatility * np.sqrt(count)
56 print("Volatility=", volatility)
```

Code lines 47–48 apply the RMSE formula to calculate the degree of discrepancy between the predicted prices and real prices (y_test) and display the result. Code lines 49–50 apply the MAE formula to calculate the degree of discrepancy between the predicted prices and real prices (y_test) and display the result. Code lines 51–56 apply the volatility formula to calculate the volatility of stock prices based on the test set and display the result.

Code running result as below (Fig. 5).

```
8/8 [==============================================] - 1s 32ms/step
RMSE= 9.1085936060445718
MAE= 31.6988327615506
daily_volatility= 33.66907
count= 251
Volatility= 533.4184053639773
```

Fig. 5. The volatility of stock prices based on the test set

We use the *filter* method to only retain the closing price column in the dataframe and split our stock data into three plotting regions: training, validation and prediction (code lines 57–60):

```
57  data = stock_data.filter(['Close'])
58  train = data[:training_data_len]
59  validation = data[training_data_len:]
60  validation['Predictions'] = predictions
61  plt.figure(figsize=(16,8))
62  plt.title('Model')
63  plt.xlabel('Date')
64  plt.ylabel('Close Price USD ($)')
65  plt.plot(train)
66  plt.plot(validation[['Close', 'Predictions']])
67  plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
68  plt.show()
```

Code lines 61–68 configure the chart figure size, title, x-axis & y-axis label. Code running result as below (Fig. 6).

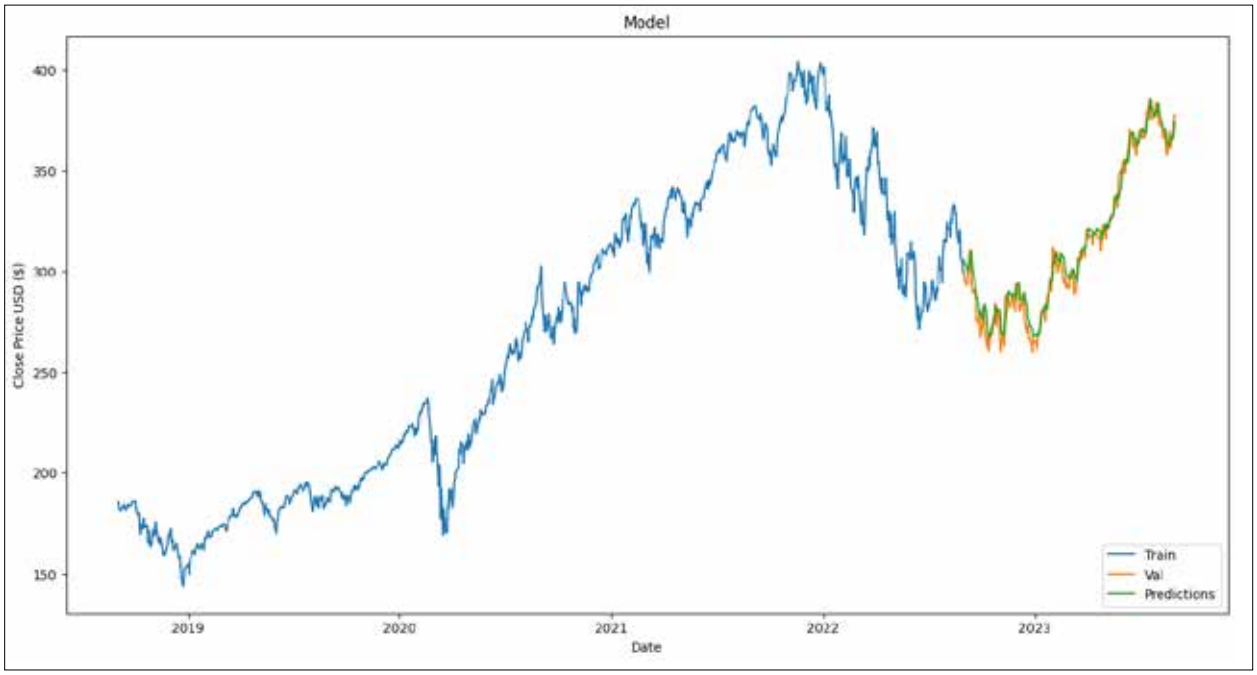

Fig. 6. Training, validation and prediction for closing price

In the research we compare results between a classic ARMA (Auto Regressive Integrated Moving Average) model and LSTM model.

The difference in performance between a classic ARMA model and an LSTM model for prediction can vary significantly depending on the specific dataset, the characteristics of the time series, and the quality of the features used (Table 2).

To compare the ARMA and LSTM models, we split the dataset into training and testing sets, train both models, and evaluate their performance using Mean Squared Error and Mean Absolute Error metrics. The choice between models depends on the data's characteristics, with ARMA suitable for linear, stationary series and LSTM for more complex patterns. Empirical testing is crucial for a reliable comparison. Using LSTM can enhance stock market forecasting accuracy by up in average to 15 % due to its ability to capture complex patterns and handle non-stationary data. The time period factor, reflecting training and testing duration, influences prediction accuracy. Longer training periods capture long-term trends, crucial for gradual changes. ARIMA requires sufficient training for stationarity, while LSTM benefits from longer periods. Adapting models to changing patterns through periodic updates enhances accuracy. Finding the optimal time period balances historical information capture and model adaptability to change.

We use 6 months, 1 year, 3 years and 5 years four different time periods to test the relationship between length of time horizon and accuracy of prediction. It performs three trials for each period. The test results are as below (Table 3).

Table 2

The difference between a classic ARMA model and an LSTM model

Model	Strengths	Weaknesses
ARMA	ARMA models are effective for capturing linear dependencies in time series data and are relatively interpretable.	ARMA may struggle with non-linear relationships, complex patterns, and long-term dependencies.
LSTM	LSTMs excel at capturing non-linear dependencies, handling long sequences, and learning complex patterns. They are well-suited for time series data with intricate dynamics.	LSTMs can be computationally intensive, may require a larger amount of data to train effectively, and their black-box nature makes interpretation more challenging.

Table 3

The test the relationship between length of time horizon and accuracy

Stock index symbol	Time period	Trial	MAE	Average MAE	RMSE	Average RMSE
1	2	3	4	5	6	7
NASDAQ INDEX ETF (QQQ)	6 months	1st trial	6.7135	8.7709	1.7893	1.8691
	6 months	2nd trial	9.6697		1.9716	
	6 months	3rd trial	9.9296		1.8464	
	1 year	1st trial	10.7406	14.7756	6.1734	6.3860
	1 year	2nd trial	15.3006		5.0005	
	1 year	3rd trial	18.2858		7.9841	
	3 years	1st trial	24.3482	24.1006	7.9908	8.1220
	3 years	2nd trial	24.3163		7.6801	
	3 years	3rd trial	23.6374		8.6952	
	5 years	1st trial	31.4609	31.5846	7.5907	9.0781
	5 years	2nd trial	31.5941		10.5352	
	5 years	3rd trial	31.6988		9.1086	
S&P 500 INDEX ETF (SPY)	6 months	1st trial	5.8295	5.9424	1.0637	1.3618
	6 months	2nd trial	6.1867		1.6571	
	6 months	3rd trial	5.8112		1.3648	
	1 year	1st trial	8.3912	8.4622	3.1726	2.6822
	1 year	2nd trial	9.1074		3.2899	
	1 year	3rd trial	7.8880		1.5840	
	3 years	1st trial	11.7970	12.1536	3.7479	4.3685
	3 years	2nd trial	11.5517		2.6988	
	3 years	3rd trial	13.1121		6.6587	
	5 years	1st trial	17.8998	18.7945	4.4478	8.3420
	5 years	2nd trial	21.1163		15.1932	
	5 years	3rd trial	17.3673		5.3850	

Table 3 (ending)

1	2	3	4	5	6	7
DOW JONES INDEX ETF(DIA)	6 months	1st trial	4.2899	4.4544	1.1860	1.2047
	6 months	2nd trial	5.0016		1.5155	
	6 months	3rd trial	4.0717		0.9126	
	1 year	1st trial	9.3840	7.9804	1.7926	1.8547
	1 year	2nd trial	7.8737		1.8309	
	1 year	3rd trial	6.6833		1.9406	
	3 years	1st trial	9.4862	9.5353	2.6493	2.5654
	3 years	2nd trial	9.5737		2.4493	
	3 years	3rd trial	9.5460		2.5977	
	5 years	1st trial	15.4066	15.6000	2.8401	3.2497
	5 years	2nd trial	15.2846		1.4647	
	5 years	3rd trial	16.1087		5.4442	

Volatility measures the degree of price fluctuation in financial instruments over time and is crucial in assessing risk. In stock market prediction, volatility significantly impacts model performance. Traditional models like ARIMA may struggle with high volatility due to their stationary assumptions.

Advanced models like LSTM networks are more adaptable to volatility, capturing long-term patterns effectively. Models incorporating volatility indicators can better manage uncertainty. High volatility poses challenges for prediction accuracy, but adaptive models can offer robust forecasts.

In the research, we test volatility's relationship with prediction accuracy over different time periods using MAE and RMSE evaluation metrics. LSTM models demonstrate high accuracy, especially in shorter time frames and high-volatility scenarios.

Conclusions and future work. Stock market data are affected by many external factors, such as national policy, macroeconomics, company fundamentals and human psychological factors. Therefore, stock price patterns are difficult to predict, which in turn affects stock indexes. Traditional statistical analysis prediction models have certain deficiencies in predicting complex stock market time series data, while the deep learning model LSTM neural network has greater advantages in predicting long-term series data, and the model has stronger generalization capabilities.

By building a deep learning LSTM neural network model, the influence of certain human irrational factors can be eliminated, allowing the computer to continuously learn from historical data to derive corresponding data change patterns. The use of LSTM can improve the accuracy of forecasting operations in the stock market by up in average to 15 % compared to using the ARMA model according to the research. LSTM neural network demonstrates promising accuracy in predicting stock market operations, highlighting its potential for effective forecasting, while future research could explore its application across diverse market conditions and the integration of real-time data for enhanced performance.

Bibliography:

1. David M. Q. Nelson, Adriano C. M. Pereira, Renato A. de Oliveira. Stock Market's Price Movement Prediction with LSTM Neural Networks. *International Joint Conference on Neural Networks (IJCNN)*. 2017. P. 1419–1426. DOI: 10.1109/IJCNN.2017.7966019.

2. Adil Moghar, Mhamed Hamiche. Stock Market Prediction Using LSTM Recurrent Neural Network. *International Workshop on Statistical Methods and Artificial Intelligence (IWSMAI).* 2020. Warsaw, Poland. Procedia Computer Science. Vol.170. P. 1168–1173. DOI: 10.1016/j.procs.2020.03.049.

3. Lin Y., Yan Y., Xu J., et al. Forecasting stock index price using the CEEMDAN-LSTM model. *The North American Journal of Economics and Finance*. 2021. Vol. 57. P. 101421.

4. Fischer T., Krauss C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*. 2018. Vol. 270. P. 654–669.

5. Yang Q., Wang C. Research on global stock index prediction based on deep learning LSTM neural network. *Statistical Research*. 2019. Vol. 36 (03). P. 65–77.

6. Han Shanjie, Tan Shizhe. Design and implementation of deep learning model for stock prediction based on TensorFlow. *Computer Applications and Software*. 2018. Vol. 35 (6). P. 267–271.

7. Bao Zhenshan, Guo Junnan, Xie Yuan, Zhang Wenbo. Stock price fluctuation prediction model based on LSTM-GA. *Computer Science*. 2020. Vol. 47. P. 467–473.

8. G. Peter Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*. 2003. Vol. 50. P. 159–175.

9. Liu S., Zhang C., Ma J. CNN-LSTM neural network model for quantitative strategy analysis in stock markets. *International Conference on Neural Information Processing*. Springer, Cham. 2017. P. 198–206.

10. Ouyang Hong-Bing, Huang Kang, Yan Hong-Ju. Prediction of financial time series based on LSTM neural network. *Chinese Journal of Management Science*. 2020. Vol. 28 (4). P. 27–35.

Олещенко Л.М., Чжен Ц. ЗАСТОСУВАННЯ НЕЙРОННОЇ МЕРЕЖІ LSTM ДЛЯ ПРОГНОЗУВАННЯ ОПЕРАЦІЙ НА ФОНДОВОМУ РИНКУ

Фондовий ринок є важливою частиною сучасної фінансової системи. Аналіз і прогнозування тенденцій фондового ринку може ефективно допомогти інвесторам уникнути інвестиційних ризиків і збільшити прибуток. Фондовий індекс відображає зміни на фондовому ринку та може відображати тенденції ринку. Прогнозування часових рядів на фондовому ринку є актуальним напрямком досліджень серед інвесторів і науковців. Традиційні статистичні моделі широко використовуються в задачах прогнозування часових рядів даних фондового ринку. Через нелінійні характеристики даних часових рядів традиційні статистичні моделі мають певні недоліки та проблеми прогнозування. Моделі машинного навчання можуть ефективно вирішувати нелінійні проблеми в даних часових рядів. З розвитком моделей машинного навчання, моделі глибокого навчання, такі як штучні нейронні мережі, почали широко використовуватися в прогнозуванні часових рядів для даних фондового ринку.

У статті проаналізовано використання рекурентних нейронних мереж для прогнозування операцій на фондовому ринку. Наведено основні переваги та недоліки існуючих рішень. Описано запропоновану архітектуру програмного забезпечення та основні технології для прогнозування операцій на фондовому ринку, наведено опис розробленого програмного забезпечення, його основних модулів та компонентів. Наведено функціональні можливості розробленого програмного продукту. Проаналізовано ефективність запропонованого програмного методу з використанням нейронної мережі LSTM для прогнозування операцій на фондовому ринку.

Запропоноване програмне забезпечення реалізує метод навчання нейронної мережі LSTM для підвищення точності прогнозування фінансових операцій на фондовому ринку. Згідно досліджень, використання LSTM може підвищити точність прогнозування операцій на фондовому ринку в середньому до 15 % у порівнянні з використанням моделі ARMA. Розроблене програмне забезпечення на мові програмування Python разом із бібліотеками та модулями машинного навчання дозволяє отримувати відносно точні результати прогнозування, аніж з використанням відомих статистичних методів.

Ключові слова: *програмне забезпечення, нейронні мережі, ARMA, ANN, RNN, LSTM, Python, TensorFlow, прогнозування операцій фондового ринку.*