2021 IEEE Colombian Conference on Applications of Computational Intelligence (ColCACI) | 978-1-6654-3534-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ColCACI52978.2021.9469554

Stock Price Analysis with Deep-Learning Models

Juan Arosemena, Noel Pérez, Diego Benítez, Daniel Riofrío and Ricardo Flores-Moyano

Colegio de Ciencias e Ingenierías "El Politécnico",
Universidad San Francisco de Quito USFQ, Quito 170157, Ecuador
jjarosemena@estud.usfq.edu.ec, {nperez, dbenitez, driofrioa, rflores}@usfq.edu.ec

***Abstract*—Novel artificial intelligence prediction algorithms use deep learning techniques, i.e., recurrent neural networks and convolutional neural networks, to predict financial time series. Also, autoencoders have gained notoriety to extract features from latent space data and decode them for predictions. This paper compares several deep learning architectures with different combinations of long short-term memory networks and convolutional neural networks. Autoencoders are implemented within these networks to find the best model performance for financial forecasting tasks. Four different architectures were trained with stock market data of four companies (AMD, ResMed, Nvidia, and Macy's) from 2010 to 2020. Without autoencoder, the long short-term memory network architecture achieved the best performance for all companies, obtaining a mean squared error of 0.004 for AMD stocks by applying 10-fold nested cross-validation. The results show that long short-term memory networks are very well suited for prediction tasks using a simple deep-learning architecture.**

***Keywords*—LSTM, Conv2D, autoencoder, stocks, prediction, deep learning, time-series**

I. Introduction

Financial markets are one of the most driving factors in today's world economy. The stock market is a subset of all financial markets that involves trading shares of public companies. These markets are characterized by their unpredictability, which for the inexperienced investor means a gamble with their money [1]. Stocks, also known as shares or securities, represent the ownership of a part of a company, and they can be bought and sold for currency. The price of stocks in markets tends to vary unpredictably along different periods, translating to gains and losses of value for such stocks' owners. This variability motivates some investors and scares away others from participating in the stock market since the profit from an investment depend on the correct risk management and being able to forecast when, by how much, and in what direction the stock price will move. With enough gained forecasting information, an investor can lower their probability of losing the value of their investments as well as optimize their gains [2] [3].

Throughout history, analysts have developed statistical models that can outperform gambles with the market by giving specific margins of risk and expecting certain price movements in the future. On the other hand, machine learning techniques for stock trend prediction have been on the rise since the early 2000s [4] by proving that they are capable of learning patterns on historical data to tell, with considerable precision, how the trend will move or even how the price of a stock will change. Two traditional machine learning techniques used for market predictions are support vector machines (SVM) and artificial neural networks (ANNs), the latter offering the best performance in terms of accuracy. However, in [5], several limitations of SVM approaches that cause overfitting in predicting financial data due to the higher dimensional, precariousness and noise associated were highlighted.

The promising potential of ANNs and backpropagation algorithms has branched into different neural network architectures applied to the field of stock market analysis and many ways to implement these architectures with the available data. Thus, this paper focuses on two of the most popular neural network architectures currently under research for the stock market: convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We dealt with numerous amounts of inputs for market data of several financial indicators using different periods for each one. CNNs are capable of extracting and down-sampling features from the local interactions between matrix-arranged inputs. [6]. RNNs are also a key feature in this study as they are designed to capture patterns in time series, which translates to analyzing the variation of these numerous inputs through time [7]. Lastly, autoencoders are used to explore their ability to extract hidden features in latent space. They have proven helpful in allowing deep learning models interpret hidden patterns more precisely [8].

Several previous works have studied the usefulness of building complex deep learning models to forecast the near future stock market. A common approach involves analyzing time series using feature extracting techniques like ARIMA or ARFIMA [9] or Empirical Mode Decomposition [10], which are then fed into neural network architectures for stock price prediction. Another methodology incorporates both CNNs and RNNs into deep learning models to train models to predict prices based on sequential samples with diverse and numerous inputs [11] [12] including non-financial time series such as electrical consumption [13], and it has proven to outperform traditional shallow learning architectures. A few papers have also implemented these models with autoencoders [14], a type of deep learning architecture that transforms input data into latent space code to be then interpreted to remove noise and redundancies in the data. However, these papers tend to focus on the performance of a single architecture. They do not compare the incremental improvement that yields by applying the mentioned network elements on the same data sets.

This work aims to explore the effectiveness of applying and combining different types of the aforementioned neural networks to stock price prediction. The significance between tested models is measured based on error measurements of future price predictions for specific stocks. The utility of using automatic encoders is also analyzed based on the results obtained for each combination, reporting the best performance model with its respective set of hyperparameters.

II. Materials and Methods

A. Stock database

This experiment used publicly available stock data from several companies considered volatile in the market, namely AMD, ResMed, Macy's, and Nvidia. Market index data was used as well as part of the input vectors to feed the models, and the indices used were Dow Jones Industrial, Nasdaq Composite, S&P 500, NYSE, and Russell 2000. All this information was downloaded using yfinance [15], which is a python library created by Ran Aroussi that retrieves historical stock and market index data from Yahoo Finance [16]. Both stock and index data contain daily ticker information that includes open, close, high, low prices, and trading volume. They also include other factors not considered for this study. Raw data contains market history from 2010 to 2020, and each year provides 252 daily ticker records.

B. Deep-learning models

Deep learning is one of the most successful approaches to solve time series prediction problems. There has been increasing popularity in the application of this methodology with financial time series using complex architectures and novel machine learning algorithms [17] [18]. Two deep learning architectures are used for this experiment: a model with and without an autoencoder and two different machine learning algorithms to process the data inputs for each model.

1) Autoencoders: Autoencoders consist of two parts: an encoder and a decoder. Encoders take the inputs and transform them into latent space by reducing its dimensionality. The decoder takes this encoded information and reconstructs it, which allows it to detect nonlinear correlations and reduce redundancies in the data more easily [19]. Autoencoders can be implemented with almost any machine learning algorithm, but RNNs are typical candidates for decoder implementation.

2) LSTM: Long short-term memory (LSTM) networks are a popular type of RNN well suited for time-series predictions. LSTMs are made of a memory cell and hidden states that remember the previous elements in an input sequence sample and three gates that regulate the flow of information in the network. The memory cells of the LSTM allow it to make better predictions than traditional feed-forward networks by including a temporal dimension to the data, which is used to exploit temporal patterns in financial data because each time step involves matrix operations with trainable weights [20] between the memory cell, the inputs, and hidden states. LSTM networks require the shape of the elements in input sequences to be one-dimensional.

3) Conv2D: CNNs are a type of neural network generally used along with input data with numerous features arranged in matrices (e.g., digital images). Conv2D is a type of CNN that applies convolution operations to two-dimensional inputs and also allows the inputs to have an additional channel axis to use for the depth of convolutions, such as in RGB images. These networks also use max-pooling layers after the convolutions to locally extract relevant features from the latent space of the input data [21]. Conv2D networks can also be used for time series applications by using the two dimensions as a feature and temporal axes, while the channel axis can be used to add related sources of the same data. The input data must be arranged into two-dimensional matrices, plus a channel axis.

C. Proposed method

The prediction task is carried out by four different deep learning architectures, including two models without an RNN autoencoder and two models with autoencoders. The first group of models uses each of the aforementioned neural networks, followed by a fully connected section of two hidden layers and an output layer. These models were named as LSTM and Conv2D in reference to the leading deep learning component in each of them. The first has an LSTM module that takes an input of 60-time steps by 240 features returning a single-time step output of size 200 to the fully-connected section, which we call dense in this experiment. Before connecting to the dense component, a batch normalization layer to the LSTM outputs with a momentum of 0.5 was used. The dense component takes the LSTM output and passes it to a 400 neuron hidden layer, followed by another 100 neurons hidden layer, and finally to a single neuron output layer that returns the predicted price. The first and second layers of the dense section apply a dropout of 0.3 to reduce overfitting and apply a *tanh* activation each. The output layer of the dense component uses a sigmoid activation to guarantee that the values fall into the same range as the normalized prediction labels. Fig. 1 shows a visual representation of the LSTM architecture without an autoencoder.

The Conv2D model implements the same architecture as LSTM, using batch normalization after the main component and the same dense section. For Conv2D, the inputs are 60-time steps by 40 indicators and periods by six markets, using this last axis as the channel dimension of the convolution operations, followed by a max-pooling layer of size 2.

The second group of models includes an additional RNN module to serve as a decoder for the autoencoder architecture, specifically an LSTM decoder. For these, the same deep learning architecture as the models in the first group was used, except the principal component as an encoder to the LSTM decoder that connects to a dense component, and the models were named LSTM-LSTM and Conv2D-LSTM. For the Conv2D-LSTM model, a flattening layer after the encoder outputs was applied, and two-dimensional vectors, to match the decoder inputs' dimensions. Before passing the inputs to the RNN decoder, each model uses a repeat vector layer to pass the encoded vectors as sequences. For the LSTM-LSTM

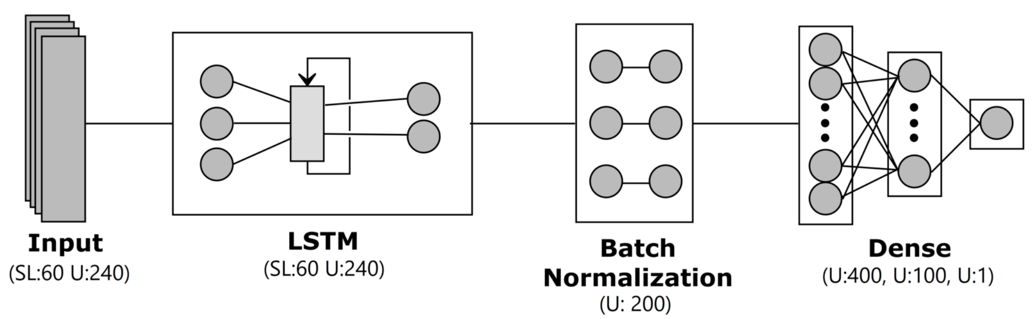

Fig. 1: Architecture of the proposed LSTM model without autoencoder. Batch size is not depicted because it is a training hyperparameter; SL - sequence length; U - units.

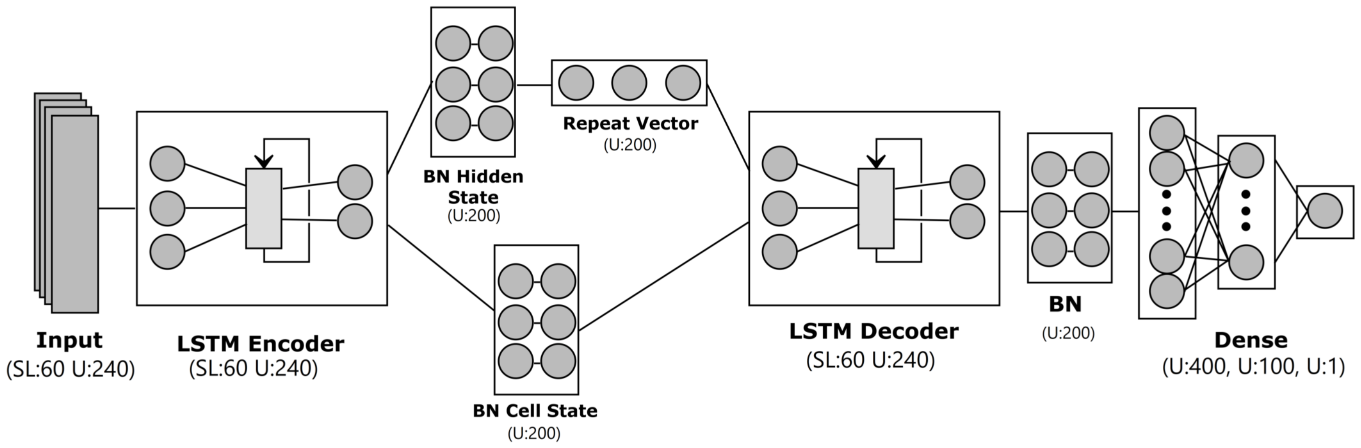

Fig. 2: Architecture of the proposed LSTM-LSTM model with autoencoder. Batch size is not depicted because it is a training hyperparameter; SL - sequence length; U - units; BN - batch normalization.

model, the hidden states of the decoder were initialized with the last state returned by the LSTM encoder. The decoders in both autoencoder models are set to produce a single time step vector instead of sequences to predict the stock price for the next time step of each sample. Finally, the decoder outputs pass through the same dense section described in the first group of models to deliver a single value prediction. Fig. 2 shows a visual representation of the LSTM-LSTM autoencoder architecture.

D. *Experimental setup*

1) Feature vectors and labels: One preprocessed data set is built for each of the analyzed companies. One feature vector for a date consists of 240 features calculated for that date and the past 59 days, making it a total of $240 * 60 = 14400$ values for each daily stock vector. The 240 daily features are made up of 8 different financial indicators calculated using five different periods: 1, 5, 10, 20, and 90 market days in the past as information windows. The list of financial indicators is as follows: Rate of Change (ROC), Relative Strength Index (RSI), Money Flow Index (MFI), Exponential Moving Average (EMA), Stochastic Oscillator (SO), Aroon Indicator (Aroon), Detrended Price Oscillator (DPO), and Average True Range (ATR). These $8 \times 5 = 40$ indicators are calculated for the stock data we try to predict concatenated with the same indicators for the five market indices selected; this means each vector contains the same information for six financial sources, which yields the $8 \times 5 \times 6 = 240$ total of daily features.

The features are normalized using Min-Max scaling across the entire data set for each company. Each input vector is labeled with the normalized closing price of the day following the whole time series of the sample (i.e., the price on day 61 for a given sample).

The experiment data set was constructed by considering two topologies for the same data: one-dimensional (1D) and two-dimensional (2D), which is used to train and test the LSTM and Conv2D well as their autoencoder variations. This approach allows us to apply the same data sets to different deep learning models with varying input topologies. The dimensions refer to the shape of each daily feature vector in each sample sequence. Simultaneously, the time axis represents an additional dimension of length 60 in each of the feature arrangements. The samples in 1D data sets are arranged in chronologically sequential, one-dimensional vectors of 240 features. The samples in 2D data sets are arranged in sequences of two-dimensional vectors of 40×6 features corresponding to the 40 indicators and periods used to calculate them across the stock and index data.

2) Training and test partitions: A special kind of cross-validation to the data sets called Time Series Nested Cross-Validation [22] was applied, which involves performing a k number of train/test splits into increasing portions of the chronologically ordered data set. $k = 10$ train/test splits were performed, where $split_i$ contains all the data from the beginning of the whole series up to i/k of its length for each $i \in [1, k]$. Each of these splits was then separated into train and test sets with a 70/30 ratio. Classical k-fold cross-validation cannot be used for time series data sets as it would not be practical to train a machine learning algorithm with future data to predict past data.

For each of the four stocks to analyze, the two aforementioned dimensional arrangements of the data sets were generated, and then split into ten training and testing sets of increasing sizes, yielding a total of 120 separated data sets.

3) Model configuration: Each of the four deep neural networks was trained on each stock train data set for 10,000 epochs with early stopping and using the test data set as the validation set. The early stop is triggered when no decrease of loss has been detected for the last 100 epochs. For every model, the mean squared error (MSE) was used as the loss function to optimize. The models were trained with batch sizes of 120, 60, and 30 samples. The models were also trained using learning rates of 0.01, 0.005, 0.001, and 0.0005 with the Adam optimizer [23]. Adam is an algorithm for first-order gradient-based optimization of objective functions, which is the main goal of machine learning algorithms and applies well to our training architecture. The Adam optimizer was used because it dynamically adjusts the learning rate during training and is known for being memory efficient.

4) Experimental environment: The deep learning models were implemented using the Keras Functional API. The experiments were carried out in a distributed GPU environment in an Nvidia DGX Station for optimal training time.

5) Assessment metrics: For each trained instance, the models were evaluated on test data to predict the future price of each sample and measure the mean squared error, root mean squared error (RMSE), and mean absolute error (MAE). MSE can be interpreted as how well fitted is the prediction line against the actual values, giving higher errors for outliers. RMSE and MAE can be interpreted similarly as how far our predictions are from the actual values with the same measurement units as the target value.

6) Selection criteria: For each company, the best performing model (with its hyperparameters) was selected in terms of the mean MSE for the four architectures to visualize the predictions.

III. Results and Discussion

For the 1,920 trained deep learning instances, the results were evaluated for each company. The trained models were evaluated on the respective testing set of each fold, and their MSE, RMSE, and MAE were calculated to obtain the mean and standard deviation of each metric across the folds. Table I shows these summarized evaluations from which we can argue that:

1) Best and worst overall architectures: The LSTM without autoencoder appears to be the best performing model among the rest in all companies. The best model performance was achieved on AMD data with a batch size of 30 and a learning rate of 0.001, with a mean MSE of 0.004. The learning rates of all four models are among the lower ones tested, but there is no specific optimal batch size for the best performing instances. Also, it seems that the Conv2D and Conv2D-LSTM models mainly performed the worst among all companies compared to the LSTM-based architectures.

2) Best fitted company: It is pretty noticeable that all architectures performed better with the AMD data set than with other companies. This may be due to differences in the volatility of prices between the companies, which account for more noisy training data. However, the difference between the performance of LSTM overall companies is within the same order of magnitude, which means there is no exaggerated preference for this model.

3) Autoencoder performance: We can conclude that the addition of autoencoders for this time series prediction task did not show any improvement in performance under the same training conditions as non-autoencoders. The extra complexity of the models may require more training time to achieve lower errors. However, this may hint that the models are overfitting on the training data and missing on the testing data.

A. Performance evaluation

For each company, the best performing model trained on the last fold of each cross-validation was evaluated. The predicted price time series were plotted over the actual prices to visualize how the models could simulate trading algorithms, as shown in Fig. 3. As it can be seen, the prediction plot fits very close to the realistic prices for AMD and Macy's. However, the sudden rising and fall in trading prices are not always predicted correctly, which accounts for the higher errors obtained for ResMed, NVidia, and Macy's stock prices. This could be because the model quickly learns to fix the forecast at past prices and then pushes it slightly up or down based on the other characteristics to predict in the right direction. Although this may seem like a good strategy when the variations in the trading price are within a small margin between the limits of the training data, however, when the trading price movement is considerably out of these limits, the model cannot adjust to the changes and therefore denote a significant error with the actual values. In the case of AMD, the predicted curve fits considerably close to the actual values. However, all four models tend to replicate the overall trend movement, even for Fig. 3b, but with disproportionate prices.

A significant factor affecting the way the deep learning models interpret the target labels is how they are normalized. For this experiment, the Min-Max scaling was used, which essentially bounds the data sets with their maximum and minimum values, which means that the model may struggle

TABLE I: Results of the 10-Fold time-series nested cross-validation of evaluated models per company. For each architecture, the best performing set of hyperparameters was select. Each metric shown in teh table is the mean of the cross-validation plus/minus a standard deviation.

Stock	Model	Learning Rate	Batch Size	MSE $\pm\sigma$	RMSE $\pm\sigma$	MAE $\pm\sigma$
AMD	LSTM	0.0010	30	0.004 $\pm$0.007	0.049 $\pm$0.046	0.038 $\pm$0.036
	Conv2D	0.0100	120	0.007 $\pm$0.009	0.069 $\pm$0.054	0.055 $\pm$0.042
	LSTM-LSTM	0.0010	30	0.009 $\pm$0.019	0.062 $\pm$0.073	0.050 $\pm$0.055
	Conv2D-LSTM	0.0050	30	0.010 $\pm$0.021	0.066 $\pm$0.080	0.053 $\pm$0.061
ResMed	LSTM	0.0050	120	0.006 $\pm$0.014	0.053 $\pm$0.060	0.043 $\pm$0.048
	Conv2D	0.0050	30	0.018 $\pm$0.033	0.096 $\pm$0.097	0.083 $\pm$0.085
	LSTM-LSTM	0.0050	60	0.013 $\pm$0.020	0.085 $\pm$0.081	0.065 $\pm$0.070
	Conv2D-LSTM	0.0005	30	0.015 $\pm$0.025	0.095 $\pm$0.079	0.078 $\pm$0.069
Nvidia	LSTM	0.0050	60	0.008 $\pm$0.012	0.059 $\pm$0.068	0.049 $\pm$0.058
	Conv2D	0.0100	30	0.017 $\pm$0.024	0.090 $\pm$0.101	0.076 $\pm$0.088
	LSTM-LSTM	0.0010	30	0.015 $\pm$0.021	0.085 $\pm$0.093	0.072 $\pm$0.079
	Conv2D-LSTM	0.0050	30	0.014 $\pm$0.021	0.080 $\pm$0.092	0.066 $\pm$0.078
Macy's	LSTM	0.0010	120	0.008 $\pm$0.008	0.078 $\pm$0.041	0.065 $\pm$0.038
	Conv2D	0.0100	30	0.023 $\pm$0.020	0.139 $\pm$0.064	0.121 $\pm$0.059
	LSTM-LSTM	0.0005	30	0.019 $\pm$0.016	0.127 $\pm$0.054	0.110 $\pm$0.051
	Conv2D-LSTM	0.0050	120	0.036 $\pm$0.036	0.167 $\pm$0.094	0.143 $\pm$0.083

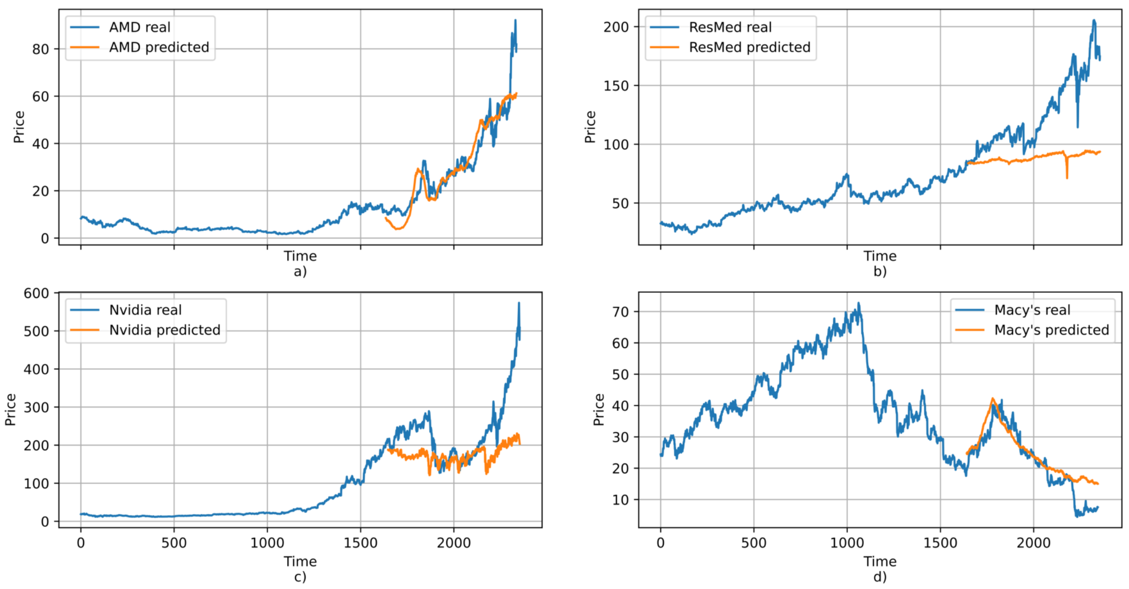

Fig. 3: Results evaluation of the best performing models for each company: a) AMD, b) ResMed, c) Nvidia, and d) Macy's. The time axis refers to the daily samples from 2011 up to 2020. Both the training and test sets are shown. However, only the predictions of the 10th fold model are plotted.

to predict prices outside of the range of prices found in the training data set. Therefore the model will struggle with companies whose prices eventually reach new highs or new lows. A possible explanation for the inconsistency between the predictions closeness to the actual values, such as Nvidia and ResMed stock prices versus AMD and Macy's stock prices, is that data normalization for each stock was performed for the entire dataset previous to train and test partitioning.

IV. Conclusions and Future Work

This paper has implemented a data processing pipeline to retrieve historical stock and market indices information and extract financial indicators from the data to use as features to our models. A pipeline to train and evaluate the proposed models against individual companies whose stock prices are considered volatile was also developed. Lastly, the best models and hyperparameters were selected for each model company to predict and plot the similarities between actual and predicted stock prices.

The results obtained indicate that LSTM-based deep-learning architectures can predict the behavior of volatile companies' stock prices over data sets with numerous features. It also shows that even though autoencoders have proven successful in other time-series tasks, the model configurations and architectures in this experiment may not apply uniformly well to all algorithms. Additionally, CNNs do not outperform

LSTMs in prediction tasks even if the data sets contain numerous features, which tends to be favorable for CNNs.

This work can be improved and expanded in two aspects. Firstly, data normalization to a Gaussian distribution can help remove the shard upper and lower bounds created by the Min-Max scaling in data [24]. And secondly, the data sets could be further enriched with non-technical information of the market like sentiment analysis, which would quantify the human reactions to financial news that could suddenly affect the market.

Acknowledgment

The authors thank the Applied Signal Processing and Machine Learning Research Group USFQ for providing the computing infrastructure (NVidia DGX workstation) to implement and execute the developed source code.

References

[1] M. Bouattour and I. Martinez, “Efficient market hypothesis: an experimental study with uncertainty and asymmetric information,” *Finance Contrôle Stratégie*, no. 22-4, 2019.

[2] T. A. Montgomery, P. M. Stieg, M. J. Cavaretta, and P. E. Moraal, “Experience from hosting a corporate prediction market: benefits beyond the forecasts,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1384–1392.

[3] A. Picasso, S. Merello, Y. Ma, L. Oneto, and E. Cambria, “Technical analysis and sentiment embeddings for market trend prediction,” *Expert Systems with Applications*, vol. 135, pp. 60–70, 2019.

[4] J. W. Lee, “Stock price prediction using reinforcement learning,” in *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, vol. 1. IEEE, 2001, pp. 690–695.

[5] I. K. Nti, A. F. Adekoya, and B. A. Weyori, “Efficient stock-market prediction using ensemble support vector machine,” *Open Computer Science*, vol. 10, no. 1, pp. 153–163, 2020.

[6] E. Hoseinzade and S. Haratizadeh, “Cnnpred: Cnn-based stock market prediction using a diverse set of variables,” *Expert Systems with Applications*, vol. 129, pp. 273–285, 2019.

[7] M. O. Rahman, M. S. Hossain, T.-S. Junaid, M. S. A. Forhad, and M. K. Hossen, “Predicting prices of stock market using gated recurrent units (grus) neural networks,” *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, vol. 19, no. 1, pp. 213–222, 2019.

[8] E. Chong, C. Han, and F. C. Park, “Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies,” *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017.

[9] A. H. Bukhari, M. A. Z. Raja, M. Sulaiman, S. Islam, M. Shoaib, and P. Kumam, “Fractional neuro-sequential arfima-lstm for financial market forecasting,” *IEEE Access*, vol. 8, pp. 71 326–71 338, 2020.

[10] F. Zhou, H.-m. Zhou, Z. Yang, and L. Yang, “Emd2fnn: A strategy combining empirical mode decomposition and factorization machine based neural network for stock market trend prediction,” *Expert Systems with Applications*, vol. 115, pp. 136–151, 2019.

[11] J. Eapen, D. Bein, and A. Verma, “Novel deep learning model with cnn and bi-directional lstm for improved stock market index prediction,” in *2019 IEEE 9th annual computing and communication workshop and conference (CCWC)*. IEEE, 2019, pp. 0264–0270.

[12] S. Liu, C. Zhang, and J. Ma, “Cnn-lstm neural network model for quantitative strategy analysis in stock markets,” in *International Conference on Neural Information Processing*. Springer, 2017, pp. 198–206.

[13] Z. A. Khan, T. Hussain, A. Ullah, S. Rho, M. Lee, and S. W. Baik, “Towards efficient electricity forecasting in residential and commercial buildings: A novel hybrid cnn with a lstm-ae based framework,” *Sensors*, vol. 20, no. 5, p. 1399, 2020.

[14] A. Essien and C. Giannetti, “A deep learning framework for univariate time series prediction using convolutional lstm stacked autoencoders,” in *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*. IEEE, 2019, pp. 1–6.

[15] R. Aroussi, “yfinance,” https://pypi.org/project/yfinance/, 2020.

[16] “Yahoo finance - stock market live, quotes, business amp; finance news.” [Online]. Available: https://finance.yahoo.com/

[17] M. Hiransha, E. A. Gopalakrishnan, V. K. Menon, and K. Soman, “Nse stock market prediction using deep-learning models,” *Procedia computer science*, vol. 132, pp. 1351–1362, 2018.

[18] M. Nabipour, P. Nayyeri, H. Jabani, S. Shahab, and A. Mosavi, “Predicting stock market trends using machine learning and deep learning algorithms via continuous and binary data; a comparative analysis,” *IEEE Access*, vol. 8, pp. 150 199–150 212, 2020.

[19] F. Soleymani and E. Paquet, “Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath,” *Expert Systems with Applications*, p. 113456, 2020.

[20] Y. Baek and H. Y. Kim, “Modaugnet: A new forecasting framework for stock market index value with an overfitting prevention lstm module and a prediction lstm module,” *Expert Systems with Applications*, vol. 113, pp. 457–480, 2018.

[21] M. U. Gudelek, S. A. Boluk, and A. M. Ozbayoglu, “A deep learning based stock trading model with 2-d cnn trend detection,” in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–8.

[22] C. Cochrane, “Time series nested cross-validation,” May 2018. [Online]. Available: https://towardsdatascience.com/time-series-nested-cross-validation-76adba623eb9

[23] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[24] G. L. Squires, *The variance of s2 for a Gaussian distribution*, 4th ed. Cambridge University Press, 2001, p. 164–165.