UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Stock Price Analysis with Deep-Learning Models

Juan Javier Arosemena Cereceda

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 17 de diciembre de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Stock Price Analysis with Deep-Learning Models

Juan Javier Arosemena Cereceda

Nombre del profesor, Título académico	**Noel Pérez, PhD**

Quito, 17 de diciembre de año

© DERECHOS DE AUTOR

Nombres y apellidos:	Juan Javier Arosemena Cereceda
Código:	00129650
Cédula de identidad:	0923477004
Lugar y fecha:	Quito, 17 de diciembre de 2020

RESUMEN

La mayoría de los algoritmos novedosos de predicción de inteligencia artificial utilizan técnicas de aprendizaje profundo para predecir series temporales financieras, comúnmente utilizando redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN) como sus componentes básicos. Además, los codificadores automáticos han ganado notoriedad por su capacidad para extraer características del espacio latente de los datos y decodificarlas también para las predicciones. En este artículo comparamos arquitecturas de aprendizaje profundo con diferentes combinaciones de redes de memoria larga a corto plazo (LSTM) y CNN, así como autocodificadores implementados con estas redes para encontrar el modelo de mejor rendimiento para las tareas de pronóstico financiero. En este experimento, entrenamos cuatro arquitecturas diferentes con datos del mercado de valores de cuatro empresas de los años 2010-2020. El modelo de mejor rendimiento fue la arquitectura LSTM sin codificador automático para todas las empresas, que arrojó un error cuadrático medio de 0,004 para las acciones de AMD al aplicar una validación cruzada anidada de 10 veces. Los resultados muestran que los LSTM son muy adecuados para tareas de predicción mediante el uso de una arquitectura simple de aprendizaje profundo.

Palabras clave: LSTM, Conv2D, autoencoder, acciones, predicción, deep learning, series de tiempo

ABSTRACT

Most novel artificial intelligence prediction algorithms use deep learning techniques to predict financial time series, commonly using Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) as their building blocks. Also, autoencoders have gained notoriety for their ability to extract latent space features from data and decode them for predictions as well. In this paper we compare deep learning architectures with different combinations of Long short-term memory (LSTM) networks and CNNs, as well as autoencoders implemented with these networks to find the best performing model for financial forecasting tasks. In this experiment, we train four different architectures with stock market data of four companies from years 2010-2020. The best performing model was the LSTM architecture without autoencoder for all companies, which delivered a mean squared error of 0.004 for AMD stocks by applying 10-fold nested cross validation. The results show that LSTMs are very well suited for prediction tasks by using a simple deep-learning architecture.

Key words: LSTM, Conv2D, autoencoder, stocks, prediction, deep learning, time series

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCTION

Financial markets are one of the strongest driving factors in the world economy, and 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 (Bouattour, 2019). The stocks, also known as shares or securities, are a representation of 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, which translates to gains and losses of value for the owner of such stocks. This variability is what motivates some investors and scares away others from participating in the stock market because the profits of the investments depend on their 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 (Montgomery, 2013) (Picasso, 2019).

Through history, analysts have developed statistical models that can outperform gambles with the market by giving certain margins of risk and expecting certain price movements in the future. However, machine learning techniques for stock trend prediction have been on the rise since the early 2000s (Lee, 2001) 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 very traditional machine learning techniques used for market predictions are support vector machines (SVM) and artificial neural networks (ANNs), the latter proving itself as the better performing option in terms of accuracy. The promising potential of ANNs and back propagation algorithms has branched into different neural network architectures that have been applied to the field of stock market analysis, as well as many ways to implement these architectures with the available data. In this paper we will focus 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 deal with numerous amounts of inputs for the market data of several financial indicators using different periods for each one, and CNNs are capable of extracting and down sampling features from the local interactions between matrix-arranged inputs. (Hoseinzade, 2019). 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 (Rahman, 2019). Lastly, autoencoders are used to explore their ability to extract hidden features into latent space as it has proven useful for deep learning models to interpret hidden patterns more precisely (Chong, 2017).

Several papers have studied the usefulness of building complex deep learning models to forecast the near future stock market. A common approach involves analysis of time series using feature extracting techniques like ARIMA or ARFIMA (Bukhari, 2020) or Empirical Mode Decomposition (Zhou, 2019), 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 on predicting prices based on sequential samples with diverse and numerous inputs (Eapen, 2019) (Liu, 2017) including non-financial time series such as electrical consumption (Khan, 2020), and it has proven to outperform traditional shallow learning architectures. A few papers have also implemented these models with Autoencoders (Essien, 2019) which is a type of deep learning architecture that transforms input data into latent space code to be then interpreted, and it can be used to remove noise and redundancies in the data. However, these papers tend to focus on the performance of a single architecture and do not compare the incremental improvement that yields by applying the mentioned network elements on the same data sets.

The main objective of this work is to explore the effectiveness of applying and combining different types of the aforementioned neural networks to stock price prediction. The effectiveness between tested models is measured based on error measurements of future price predictions for specific stocks. The usefulness of the application of autoencoders is also discussed based on the results for each combination of each type of neural network to report the best performing model with its respective set of hyperparameters.

MATERIALS AND METHODS

Stock database

This experiment uses stock data from several companies that are 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 are Dow Jones Industrial, Nasdaq Composite, S&P 500, NYSE and Russell 2000. All this information was downloaded using yfinance, which is a python library created by Ran Aroussi that retrieves historical stock and market index data from Yahoo Finance. Both stock and index data contain daily ticker information that includes open, close, high, and low prices, as well as trading volume and other factors that we are not considering for our data sets. The raw data contains market history from 2010 to 2020, and each year contains 252 daily ticker records.

Deep-learning models

Deep learning is one of the most successful approaches to solve time series prediction problems so far. There has been an increasing popularity in application of this methodology with financial time series using complex architectures and novel machine learning algorithms (Hiransha, 2018) (Nabipour, 2020). For this experiment we will study two deep learning architectures: a model without autoencoder and a model with autoencoder. Also, for each of these we will apply two different machine learning algorithms to process the data inputs.

Autoencoders.

Autoencoders are made 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 (Soleymani, 2020). Autoencoders can be implemented with almost any combination of machine learning algorithms, but RNNs are common candidates to use for the decoder implementation.

LSTM.

Long short-term memory networks are a popular type of RNN well suited to be used in time series predictions. LSTMs are made of a memory cell and hidden states which remember the previous elements in an input sequence sample, as well as 3 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. This is used to exploit temporal patterns in financial data because each time step involves matrix operations with trainable weights (Baek, 2018) between the memory cell, the inputs, and hidden states. LSTMs require the shape of the elements in input sequences to be one-dimensional.

Conv2D.

CNNs are a type of neural networks that are generally used along 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 in order to locally extract relevant features from the latent space of the input data (Gudelek, 2017). Conv2D networks can also be used for time series applications by using the two dimensions as feature and temporal axes, while the channel axis can be used to add related sources of the same data. This way, the input data must be arranged into two-dimension matrices, plus a channel axis.

Proposed method

The prediction task is carried out by four different deep learning architectures, which include two models without an RNN autoencoder and two models with autoencoders. The first group of models use each of the two aforementioned neural networks, followed by a fully-connected section of two hidden layers, and an output layer. We name these models LSTM, and Conv2D in reference to the main deep learning component in each of them. The first has a LSTM module that takes an input of 60 time steps by 240 features returning a singletime step output of size 200 to the fully-connected section, which we call Dense in this experiment. Before connecting to the Dense component, we apply a batch normalization layer to the LSTM outputs with a momentum of 0.5. The Dense component takes the LSTM output and passes it to a 400 neuron hidden layer, followed by another 100 neuron 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 also 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. Figure 1 shows a visual representation of the LSTM architecture without autoencoder.

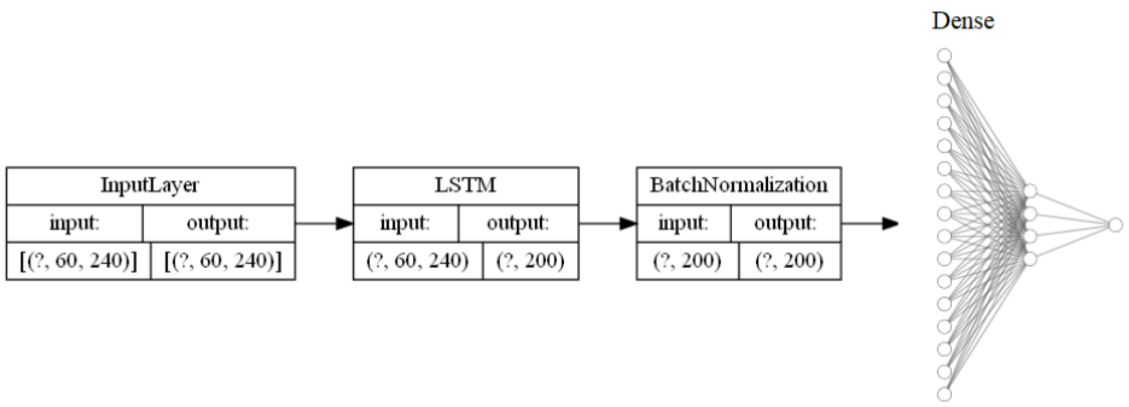

Figure 1: Architecture of the proposed LSTM model without autoencoder. The ’?’ values in the input and output shapes refer to the batch size, which is a training hyperparameter.

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 6 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 include an additional RNN module to serve as a decoder for the autoencoder architecture, specifically a LSTM decoder. For these, we use the same deep learning architecture as the models in the first group, except that we use the main component as an encoder to the LSTM decoder that connects to a Dense component, and we name the models LSTM-LSTM and Conv2D-LSTM. For the Conv2DLSTM model we apply a flattening layer after the encoder outputs, which are two dimensional vectors, in order to match the dimensions of the decoder inputs. 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 model we initialize the hidden states of the decoder with the last state returned by the LSTM encoder. The decoders in both autoencoder models are set to return a single time step vector instead of sequences to predict the price of the stock for the next time step of each sample. Finally, the decoder outputs pass through the same Dense section as described in the first group of models to deliver a single value prediction. Figure 2 shows a visual representation of the LSTM-LSTM autoencoder architecture.

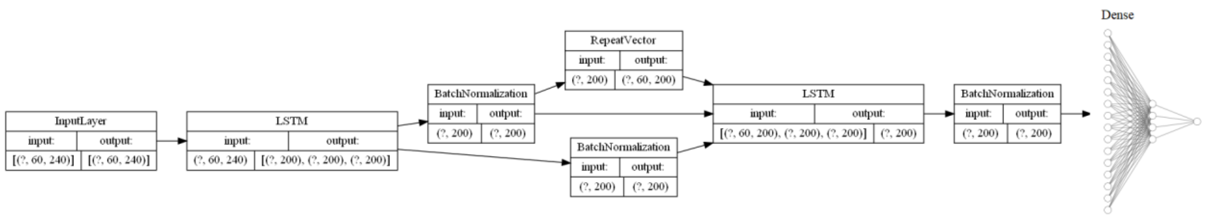

Figure 2: Architecture of the proposed LSTM-LSTM model with autoencoder. The '?' values in the input and output shapes refer to the batch size, which is a training hyperparameter.

Experimental setup

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 for the past 59 days, which makes 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 by using 5 different periods each: 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*5 = 40 indicators are calculated for the stock data we try to predict concatenated with the same indicators for the 5 market indices we selected; this means each vector contains the same information for 6 financial sources, which yields the 8*5*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 entire time series of the sample (i.e.: the price on day 61 for a given sample). We construct our experiment data set by considering two topologies for the same data: one-dimensional (1D) and twodimensional (2D), which is used to train and test the LSTM and Conv2D respectively, as well as their autoencoder variations. This approach allows us to apply the same data sets to different deep learning models with different input topologies. The dimensions refer to the shape of each daily feature vector in each sample sequence, while the temporal 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. Figure 3 shows a visualization of these data arrangements.

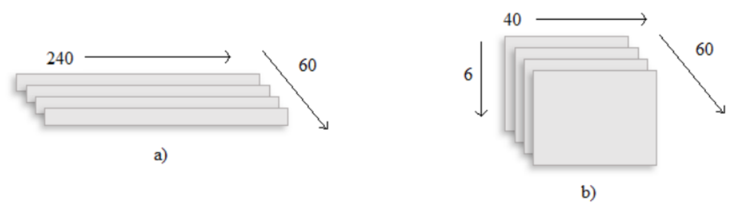

Figure 3: Visualization of a training sample of the two data topologies. a) 1D: 60 time steps of 240 features. b) 2D: 60 time steps of 40*6 features.

Training and test partitions.

We applied a special kind of cross-validation to the data sets called Time Series Nested Cross-Validation (Cochrane, 2018), which involves performing a k number of train/test splits on increasing portions of the chronologically ordered data set. We perform k = 10 train/test splits where spliti 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 is 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 in order to predict past data. For each of the four stocks to analyze we generate the two aforementioned dimensional arrangements of the data sets, which are then split into ten training and testing sets of increasing sizes. This yields a total of 120 separated data sets.

Model configuration.

Each of the four deep neural networks are trained on each stock train data set for 10,000 epochs with early stopping and using the test data set as validation set. The early stop is triggered when no decrease of loss has been detected for the last 100 epochs. For every model we use mean squared error (MSE) as the loss function to optimize. We trained the models with batch sizes of 120, 60, and 30 samples. Also, we trained the models using learning rates of 0.01, 0.005, 0.001, and 0.0005 with the Adam optimizer (Kingma, 2014). Adam is an algorithm for first-order gradientbased optimization of objective functions, which is the main goal of machine learning algorithms and applies well to our training architecture. We choose to use the Adam optimizer because it dynamically adjusts the learning rate during training and is known for being memory efficient.

Experimental environment.

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

Assessment metrics.

For each trained instance we evaluate the models 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.

Selection criteria.

We show the best performing model with its set of hyperparameters with respect to the mean MSE for each proposed company and for each architecture. For each company we select the best performing of the four architectures to visualize its predictions.

RESULTS AND DISCUSSION

Table 1: 10-Fold time series nested cross validation of evaluated models per company. For each architecture, we select the best performing set of hyperparameters. Each metric shown is the mean of each 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
RMD	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
NVDA	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
M	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

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

1) Best and worst overall architecture: The LSTM without autoencoder appears to be the best performing model among the rest in all companies. Specifically, the best model performed 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 performed mostly the worst among all companies in comparison to the LSTM-based architectures.

2) Best fitted company: It is fairly noticeable that all architectures performed better with the AMD data set than with other companies. This may be due to differences in volatility of prices

between the companies, which account to more noisy training data. However, the difference between the performance of LSTM over all companies is within the same order of magnitude, which means there is no exaggerated preference of this model over the companies.

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. It is possible that the extra complexity of the models require more training time to be able to achieve lower errors. However, this may hint that the models are actually overfitting on the training data and missing on the testing data.

Performance evaluation

For each company, we evaluate the best performing model trained on the last fold of each cross validation. We plot the predicted price time series over the actual prices to visualize how the models could simulate trading algorithms, as it can be observed in figure 4. We can see that the prediction plot fits mostly close to the real prices for AMD and M. However, the sudden jumps and falls in price are not always predicted correctly, which account to the higher errors for RMD, NVDA, and M. In the case of AMD, the predicted curve fits considerably close to the real values. However, all four models have a tendency to replicate the overall trend movement, even for figure 4b, but with disproportionate prices. A very important factor that can affect the way the deep learning models interpret the target labels is the way they are normalized. For this experiment we used Min-Max scaling, which essentially bounds the data sets with their own maximum and minimum values. This means that the model may struggle to predict prices that are outside of the range of prices found in the training data set, and therefore the model will struggle with companies whose prices eventually reach new highs or new lows.

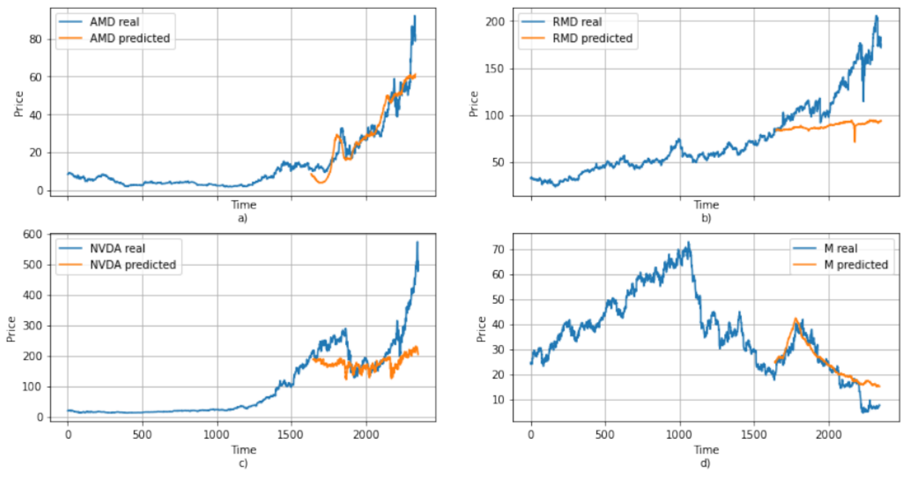

Figure 4: Evaluation of the best performing model for each company: a) AMD, b) ResMed, c) Nvidia, and d) Macy's. The time axis refers to the daily samples from 2011 up to 2020 present in both training and test sets. Only the predictions of the 10th fold model are plotted.

CONCLUSIONS

This experiment demonstrates how to implement data preprocessing and deep-learning architectures to make decently accurate predictions of financial time series. We have 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. We have also developed a pipeline to train and evaluate the proposed models against certain companies whose stock prices are considered volatile. Lastly, we selected the best models and hyperparameters for each company to predict and plot the similarities between real and predicted stock prices.

This paper shows that LSTM-based deep-learning architectures are capable of predicting considerably well on stock prices of volatile companies 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.

There are some aspects in this work that can be improved and expanded upon. First, the data processing step is crucial to the whole experiment, and an experimentation with more financial indicators could be useful to decrease errors. Also, the data sets could be further enriched with non-technical information of the market like sentiment analysis, which would give a quantification of the human reactions to financial news that could suddenly affect the market. Additionally, the way that the data is normalized can affect the limits of performance for the models, as with Min-Max scaling we are constraining the maximum and minimum predictable prices to the upper and lower bounds found in the training data. We can normalize the features and labels to a Gaussian distribution, which would remove the hard bounds on possible values for predicted prices (Squires, 2001). This experiment can also be applied to other deep learning models such as ConvLSTM2D networks, which combine the convolutional operations of a Conv2D with the sequential analysis of an LSTM. Other interesting models include Bidirectional LSTMs and Conv3D to apply for both encoders and decoders, as well as non-autoencoder variations. Finally, the experiment could be further expanded with autoencoders with attention mechanisms, which are capable of learning from time series in a more complex order than non-attention models.

REFERENCES

M. Bouattour and I. Martinez, “Efficient market hypothesis: an experimental study with uncertainty and asymmetric information,” Finance Controle Strat ˆ egie ´ , no. 22-4, 2019.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

H. Li, Y. Shen, and Y. Zhu, “Stock price prediction using attention-based multi-input lstm,” in Asian Conference on Machine Learning, 2018, pp. 454–469. [15] R. Aroussi, “yfinance,” https://pypi.org/project/yfinance/, 2020.

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

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.

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.

F. Soleymani and E. Paquet, “Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoderdeepbreath,” Expert Systems with Applications, p. 113456, 2020.

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.

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.

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

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

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