Stock Price Prediction with ARIMA and Deep Learning Models

Zihao Gao
MCCALLIE SCHOOL
Tennessee, US
bobgao22@mccallie.org

2021 IEEE 6th International Conference on Big Data Analytics (ICBDA) | 978-1-6654-1513-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICBDA51983.2021.9403037

***Abstract*—Financial markets are vital to the capitalist economies and often volatile and hard to predict. This work compares the performance of different time-series models in predicting close price movement for 30 listed stocks from the Dow Johns Industrial Average (DIJA). The mechanisms of autoregressive moving average (ARIMA), artificial neural network (ANN), recurrent neural network (RNN), and long short-term memory (LSTM) are briefly explained in the essay. Comparison results suggest that LSTM and sequence to sequence (Seq2Seq) model with attention estimates fit the price movement pattern well with relatively low latency. Built upon LSTM, Seq2Seq models exhibit good performance in forecasting. The vanilla Seq2Seq model is compared with Seq2Seq with attention in forecasting price several days in the future. Seq2Seq with attention outperforms other models and is capable of generating sequential predictions.**

Keywords—stock, time-series forecasting, machine learning, long short-term memory, sequence to sequence

I. BACKGROUND OVERVIEW

A. Introduction

Financial markets are creative inventions of mankind for the great impact they have on in societies across history. Researchers and investors have been establishing and backtesting models of stock price forecasting, which is an incredibly difficult task due to the dynamic, fluid, and unpredictable characteristics of the market. This paper focuses on technical aspect of stock price prediction with machine-learning and statistical models.

Technical analysis, according to Hu et al. (2015) [1]can be classified into the trend, momentum, raw data, volatility, volume, flow-of-funds, and sentiment. Raw data is the simple rearrangement of original data, for example, the candlestick charts. Price-based indicators, such as various types of moving average, relative strength index, and average directional index, are momentum and trend indicators, which are used to track the velocity of price movement and identify trend reversals. Volatility sheds light on the magnitude of fluctuation in prices and suggests potential risks of a market.

According to [2], pattern recognition, statistics, machine learning, and sentiment analysis are four popular approaches to stock analysis and forecasting. This paper focuses on models using statistical techniques and supervised machine learning.

The majority of statistical approaches require a stationary input series. A few of the traditional time-series models would ßbe discussed.

For supervised machine learning models in this paper, the training-set and test-set data are labeled and accessible. Supervised learning aims to train a model that minimizes the loss between the predicted value and observed value with the processed input data. Artificial Neural Network (ANN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Sequence to Sequence models(Seq2Seq), given data with sufficient information and properly trained, exhibit excellent performance.

Models used in this study have been extensively tested in previous researches except for Seq2Seq models, which are conventionally applied in machine translation tasks. The input sequence is the past price information instead of the sentence to be translated, and the output sequence is predicted the next few day's close prices. Seq2Seq with attention, outperforms any other model in most of the stocks it predicted.

B. Literature Review

De Faria et al. (2010) explored and compared the performance of the neural network and exponential smoothing models [3]. The study suggests both approaches have similar performance forecasting future values, but neural networks behave better in predicting the sign of market return.

Ariyo et al. (2014) investigated the construction of the ARIMA model [4] extensively. Several benchmarks, like adjusted R-square, select the optimal sets of parameters. The result indicates that ARIMA is powerful enough to compete with current forecasting techniques in short-term prediction.

Di Persio and Honchar (2017) compares three variants of the recurrent neural network: simple recurrent neural network, long short-term memory (LSTM), and gated recurrent unit(GRU) [5]. LSTM outperforms other RNNs with 72% prediction accuracy, making it applicable for real trading. The paper also clarifies the reason for the excellent performance of LSTM.

Dev et al. (2018) evaluate the forecasting performance of LSTM and deep neural network (DNN) [6]. Both networks behave well in daily prediction, but LSTM performs better than DNN at weekly prediction, rendering it more suitable for long-term prediction.

Rebane et al. (2019) conducted a comparative study of ARIMA and Seq2Seq models in cryptocurrency forecasting [7], a market known for its extreme volatility. Seq2Seq exhibits excellent effectiveness over ARIMA for long-term (30 days) predictions, and additional data (information of altcoins) improves the accuracy of Seq2Seq during less volatile periods.

C. Data Preprocessing

Data used in this study is available on yahoo finance. For time-series models, the input data set is from January 2016 to December 2017. For machine learning models, the training set is from January 2006 to December 2015, and the test set is from January 2016 to December 2017.

For Seq2Seq models, 12 companies' stock price are taken as inputs. Unlike traditional time-series models, machine learning models can take in many parameters, which is beneficial if more variables containing trend information could be taken as input. In this study, three indicators are used: rolling standard deviation, moving average convergence divergence, and average directional index. Volume is removed as a feature, for it seems too random to indicate patterns in daily data, and its inclusion decreases prediction accuracy.

To account for inflation, dividends, splits and other factors, all the training and back tests proceed with adjusted open, close, high, and low prices. The following includes some features included in the data.

Rolling Standard Deviation: The rolling standard deviation is obtained from the recent 6 adjusted close price.

Moving Average Convergence Divergence (MACD): The moving average convergence divergence, calculated from the index-weighted moving average of the 6 and 13 trading day adjusted closing prices, is modified to minimize the delay in trading signals.

EWMA:

$$x_t = \beta x_{t-1} + (1-\beta)\theta_t \tag{1}$$

β is the parameter to be adjusted. $(\beta)^p = \frac{1}{e}$ indicates that x_t represents the exponential moving average in p days.

Average Directional Index (ADX): Average Directional Index estimates the strength of a trend, regardless of its direction.

$$ADX = MA[\frac{((+DI)-(-DI))}{((+DI)+(-DI))}] \times 100 \tag{2}$$

+DI: Plus directional index; -DI: Plus directional index.

II. Time-Series Models

A. Definition

The main purpose of the time series model is to establish a relationship between historical information and predict the data of a certain period in the future. As shown in Figure 1, Time Series data could be decomposed into three features: trend, seasonality and residual.

B. Stationarity

Time series data used in all three models are required to be stationary, indicating mean, variance, and auto-correlation of input time series have to stay constant, in other words, time-invariant. Constantly fluctuating mean and variance reduces correlation among different properties in the series. Stock prices are unlikely to be time-invariant due to their extreme unpredictability. Certain mathematical conversions could render non-stationary series approximately stationary. ACF is a good indicator to examine whether a dataset is stationary, which will be introduced later.

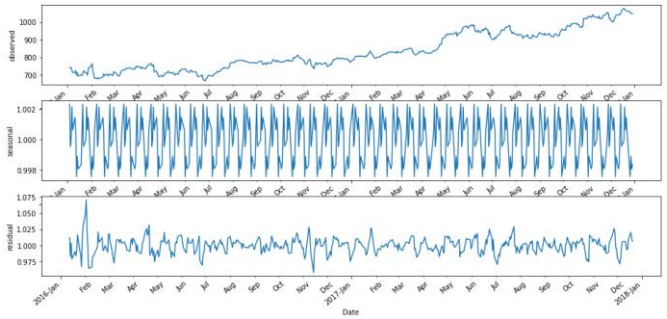

Fig. 1. Google 2016-2018 Adjusted Close Seasonal Decomposition

C. Forecasting Models

Auto Regressive (AR): AR forecasts future data point using the linear combination data of several past time-stamps and a white noise term, which is a random variable with mean 0 and a constant variance. $x_{t-1}, x_{t-2} \dots x_{t-s}$ is the rolling set of past data used to predict x_t. $u_t$is the white-noise term at time-stamp t.

$$x_t = \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_s x_{t-s} + u_t \tag{3}$$

Moving Average (MA): MA considers the current data point as the linear combination of a rolling set of previous random white-noise terms.

Facebook Prophet (FBProphet): Facebook Prophet forecasts future values by fitting linear or non-linear models to the time-series.

$$y(t) = g(t) + s(t) + h(t) + e(t) \tag{4}$$

It is an additive model with many options to customize. $g(t)$ predicts trend of the series, $s(t)$ indicates the periodic changes, $h(t)$ introduces the holiday effect if needed and $e(t)$ is the unique white noise unexplained by the previous factors. In general, Prophet estimates future value's movement through fitting a curve to historical information.

Trend:

$$g(t) = (-k + a(t)^T\delta)t + (m + a(t)^T\gamma) \tag{5}$$

δ is the vector of rate adjustments, and $\gamma_i = t_i\sigma_i$. k is growth rate, and m the offset parameter. Growth rate k receives adjustment at each time stamp.

$$Growthrate: -k + a(t)^T\delta \tag{6}$$

When growth rate is adjusted, the offset parameter m is adjusted accordingly, represented by vector γ: is the i^{th} entry and the change made to m at time t_i.

Seasonality: Seasonality considers periodic changes on the daily, weekly, or yearly basis. Fourier series helps obtain a robust model of periodic effect. Let P denotes the regular period of the time series. For instance, $P = 365.25$ for yearly data and $P = 7$ for daily data.

$$s(t) = \sum_{n=1}^{n=N} a_n cos\frac{2\pi nt}{P} + b_n sin\frac{2\pi nt}{P} \quad (7)$$

$$s(t) = \beta X(t) \quad (8)$$

III. Artificial Neural Network

A. Forward-Propagation

Neural network is a mathematical model that takes certain amount of data as inputs, and finds the internal pattern between the features and the actual label by adjusting weight parameters to minimize a predefined loss function, then yields a predicted value.

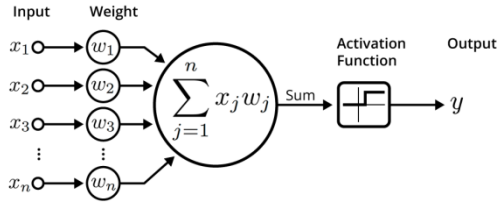

Fig. 2. Structure of a neuron

B. Gradient Descent and Back-Propagation

Only the weight matrices of each layer need to be adjusted. To find the gradient at each layer so that the modified weights can reduce the loss function, another calculation process is introduced: back-propagation.

$$a_j^l = g(z_i^l) \quad (9)$$

$$L = \left\| \hat{y} - y^2 \right\|^2 \quad (10)$$

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{\partial L}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = a_j^{(l-1)} g'(z^{(l)}) \frac{\partial L}{\partial a_i^{(l)}} \quad (11)$$

$$\frac{\partial L}{\partial a_i^{(l)}} = \sum_{k=0}^{n_{l+1}-1} \frac{\partial L}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_i^{(l)}} = \sum_{k=0}^{n_{l+1}-1} w_{ki} g'(z^{(l+1)}) \frac{\partial L}{\partial a_k^{(l+1)}} \quad (12)$$

$$\nabla L = [\frac{\partial L}{\partial w_1} \frac{\partial L}{\partial w_2} \frac{\partial L}{\partial w_3} \dots \frac{\partial L}{\partial w_{n_{weights}}}]^T \quad (13)$$

Each weight is adjusted as follow (α is the learning rate, a hyper parameter):

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \quad (14)$$

IV. Recurrent Neural Network and Sequence to Sequence Model

A. Recurrent Neural Network

While using the vanilla neural network, it is assumed that different features are independent, which is not the case for many situations, such as machine translation and stock price prediction. The recurrent neural network utilizes sequential data, which, intuitively, is a neural network unfolded through time. One input is received at each time step, and the hidden state incorporates the information from the input previous sequence by combining both X_t (the current input) and S_{t-1} (the previous hidden state). This memory preserving mechanism makes RNN born to process sequential data, as is shown in Figure 4.

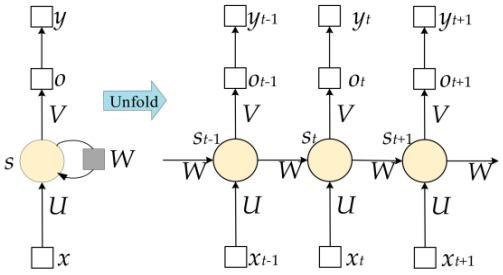

Fig. 3. The forward propagation through time of an unfolded RNN

$$s_t = Wg(s_{t-1}) + Ux_t \quad (15)$$

$$a_t = g(s_t) \quad (16)$$

$$z_t = V \cdot a_t \quad y_t = g(z_t) \quad (17)$$

The backward-propagation is much more complex for RNN than it is for the vanilla feed-forward neural network.

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} = \sum_{t=1}^{T} \sum_{i=1}^{t} \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} (\prod_{j=i+1}^{t} W^T diag(g'(s_{j-1}))) \frac{\partial s_i}{\partial W} \quad (18)$$

$$\frac{\partial s_t}{\partial s_i} = \prod_{t=1}^{t} \frac{\partial s_j}{\partial s_{j-1}} = \prod_{j=i+1}^{t} W^T diag(g'(s_{j-1})) \quad (19)$$

B. Long Short-Term Memory

RNN exhibits good performance dealing with not very long sequence inputs, but to be able to remember previous context and internal relations, the model needs to selectively forget and retain only relevant information.

For each new input at next timestamp, the model modifies existing information through matrix multiplication, suggesting that it does not distinguish "significant" information from "insignificant" ones.

By contrast, LSTM makes smaller modifications to "relevant" information through gates that control information flow

at each timestamp. The model can selectively forget "irrelevant" information, which is learnt through back-propagation.

Let X_t denotes the vector obtained by stacking x_t on top of s_{t-1}. $\sigma(x)$ represents the sigmoid function. When $\sigma(x)$ approaches 0, the previous state s_{t-1} does not flow through the forget gate, and s_{t-1} flows through when $\sigma(x)$ approaches 1.

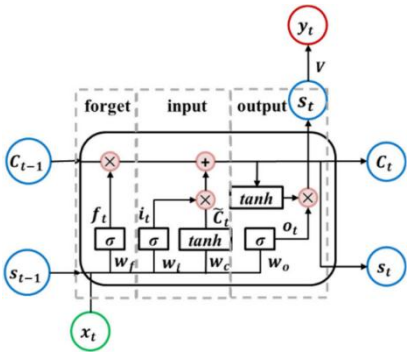

Fig. 4. Structure of an LSTM unit

Forget gate determines the part of information to be discarded: $f_t = \sigma(W^f X_t)$

Input gate determines information to be stored in the inner state c_t : $i_t = \sigma(W^i X_t)$

Output gate controls the output at time-stamp t: $o_t = \sigma(W^o X_t)$, shown in Figure 6.

$$f_t = \sigma(w_f)X_t \tag{20}$$

$$\tilde{C}_t = tanh(w_c X_t) \tag{21}$$

$$c_t = f_t \odot c^{t-1} + i_t \odot \tilde{C}_t \tag{22}$$

$$s_t = o_t \odot tanh(c^t) \tag{23}$$

$$y_t = \sigma(V s_t) \tag{24}$$

```
def seq2seq(after_day=5, input_shape=(20, 1)):
    '''
    Encoder:
    X = Input sequence; C = LSTM(X); The context vector
    Decoder:
    y(t) = LSTM(s(t-1), y(t-1)); where s is the hidden state of the LSTM(h and c)
    y(0) = LSTM(s0, C); C is the context vector from the encoder.
    '''

   encoder_inputs=Input(shape=input_shape)
    # (timesteps,   feature)
    encoder_lstm=LSTM(units=100, return_state=True, return_sequences=True)(encoder_inputs)
    encoder_lstm=LSTM(units=100,  return_state=True)(encoder_lstm)
    encoder_outputs, state_h, state_c=encoder_lstm
    states = [state_h, state_c]

    reshapor = Reshape((1, 100), name='reshapor')
    decoder = LSTM(units=100, return_sequences=True, return_state=True, name='decoder')
    densor_output = Dense(units=secshape, activation='linear', name='output')

    inputs = reshapor(encoder_outputs)
    all_outputs = []

    for _ in range(after_day):
        outputs, h, c = decoder(inputs, initial_state=states)
        inputs = outputs
        states = [state_h, state_c]

        outputs = densor_output(outputs)
        all_outputs.append(outputs)

    decoder_outputs = Lambda(lambda x: K.concatenate(x, axis=1))(all_outputs)
    model = Model(inputs=encoder_inputs, outputs=decoder_outputs)

    return model
```

Listing 1. Code for seq2seq applied to close price prediction

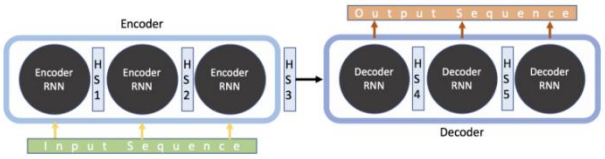

Fig. 5. Encoder-decoder model for Seq2Seq

C. *Seq2Seq Models*

Seq2Seq: In a simple RNN, output at timestamp t is determined by the hidden state of the timestamp $t-1$ and the input at timestamp t, which contributes to the excellent handling of sequence data of RNN. This type of architecture is known as "one-to-one" (i.e. one input corresponds with one output). In natural language processing problems, a sequence of output needs to be generated given a sequence of input("many-to-many"), such as translation and chat bot. One modification of RNN for machine translation is encoder-decoder model for seq2seq modeling.

The input sequence is fed to the encoder part. Each hidden state at time stamp t is calculated by:

$$h_t = g(W_e^{hh} h_{t-1} + W_e^{hx} x_t) \tag{25}$$

is the weight matrix between h_t and x_t (essentially matrix in the unfolded RNN). Subscript e indicates two matrices belong to the encoder section of the model. HS_3denotes the context vector produced by the encoder. Decoder takes context vector as input, and each hidden state at time stamp t in decoder is calculated by:

$$h_t = g(W_d^{hh} h_{t-1}) \tag{26}$$

The input of one timestamp is the output of the previous timestamp. When a sequence of information is condensed to vector, the significant loss of accuracy is inevitable if encoder

does not capture the information well, which could be alleviated by adding additional RNN layers. In this study, the encoder section has two LSTM layers. Additional features (open, high, low, rolling std, macd, adx) are taken to reduce the lag of forecasting.

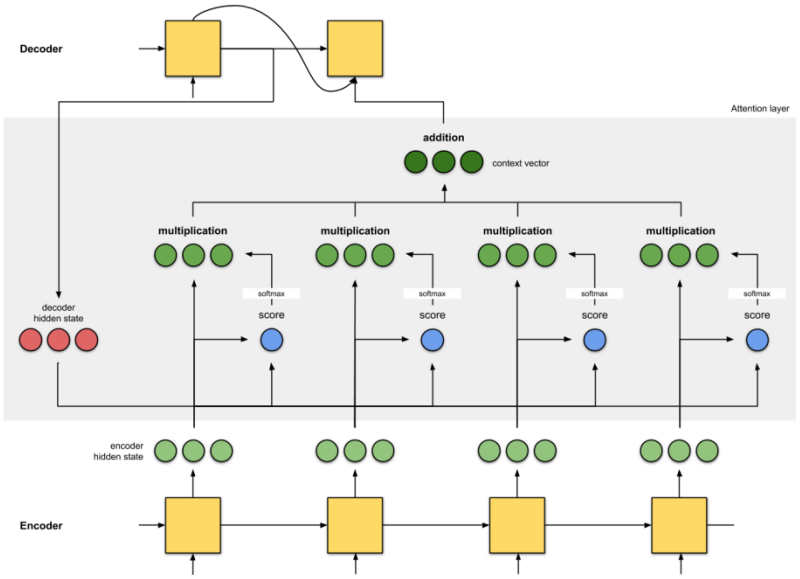

Fig. 6. Attention layer

Attention: The problem with seq2seq model is that all input the decoder receives is the context vector. When the length of the input sequence increases, the translation accuracy decreases as the model loses focus on the essential information. The model applies global attention, taking every hidden state as part of the input of the attention layer. T is the total time stamp. Labeling the first row of green hidden states as h_1, h_2, … , h_T. h_t and the hidden states of decoder at timestamp $t-1$ as s_{t-1}. Let S_{t-1} denote the matrix whose T columns are s_{t-1}, s_{t-1}, $\cdots$, s_{t-1}. H is a matrix whose columns are h_1, h_2, … , h_T. h_t. $[H_t, S_{t-1}]$ denotes stacking H_t on S_{t-1}. Score vector is in R^T.

$$score = ReLU(W_1[H_t, S_{t-1}]) \tag{27}$$

The input context vector for decoder at timestamp t, denoted by c_t is the attention weight vector α_t times H. α_t denotes the attention weight assigned to each vector.

$$\alpha_t = softmax(score) \tag{28}$$

$$c_t = \alpha_t H \tag{29}$$

Figure 8 illustrates the general structure. A large score increases the weight of a hidden state in the context vector, therefore receiving more attention as an input for the decoder. c_t, along with the last hidden state and the cell state of the decoder LSTM, would be the input for the decoder at time stamp t.

Intuitively, each hidden state of the encoder could be viewed as a semi-processed sequence of words, and the attention layer assigns a weight to each hidden state to determine how much influence it should have on a specific time stamp of the decoder, optimized by back-propagation.

For stock price prediction, seq2seq models replace the words with close prices as the inputs, which are sliced into 60 time-steps and 1 feature, to predict the next 6 close prices. Below is the code for the attention layer.

```
def one_step_attention (a , s_prev , repeator , concatenator , de
nsor ,activator , dotor ) :
    s_prev = repeator ( s_prev ) # RepeatVector ( time_step )
    concat = concatenator ([ s_prev , a ])
    e = densor ( concat ) # densor = Dense (1 , activation = " re
lu ")
    al-
phas = activator ( e ) # activator = Activation ( softmax , name
= 'attention_weights ')
    context = dotor ([ alphas , a ]) #Dot ( axes = 1)
    return context
def seq2seq_attention(feature_len=1, after_day=1, input_shape=(20,
1), time_step=20):
    # Define the inputs of your model with a shape (Tx, feature)
    X = Input(shape=input_shape)

    all_outputs = []
    encoder = LSTM(units=100, return_state=True, re-
turn_sequences=True, name='encoder')
    decoder = LSTM(units=100, return_state=True, name='decoder')
    decoder_output = Dense(units=feature_len, activation='linear',
name='output')
    model_output = Reshape((1, feature_len))

    # Attention
    repeator = RepeatVector(time_step)
```

```
    concatenator = Concatenate(axis=-1)
    densor = Dense(secshape, activation = "relu")
    activator = Activation(softmax, name='attention_weights')
    dotor =  Dot(axes = 1)

    encoder_outputs, s, c = encoder(X)

for t in range(after_day):
        context = one_step_attention(encoder_outputs, s, repeator,
concatenator, densor, activator, dotor)
        a,s,c = decoder(context, initial_state=[s, c])

        outputs = decoder_output(a)
        outputs = model_output(outputs)
        all_outputs.append(outputs)

    all_outputs = Lambda(lambda x: K.concatenate(x, ax-
is=1))(all_outputs) #keras backend imported as K
    model = Model(inputs=X, outputs=all_outputs)

return model
```

Listing 2. Code for the Attention Layer. LSTM, Dense, Activation, TimeDistributed, Dropout, Lambda, RepeatVector, Input, Reshape, Concatenate, Dot are all imported from keras.layers, and Model, Sequential, load_model from keras.model

V. Results and Discussion

Each company has 3018 recorded daily open, high, low, and close prices from January 2006 to December 2017. The date of these data from 2006 to 2016, 83% of the data set; the test data is all price information from the last two years. RNN, LSTM take the adjusted prices and three indicators described in chapter 1 as inputs, and Seq2Seq models takes only the last 60 days adjusted close as inputs.

The measurement metrics is mean-squared error.

The visualized prediction result of all models tested: https://drive.google.com/file/d/1PN03sKckC2Cu_g3mfTxZlx1I5vy8T_TA/view?usp=sharing

In table I, the majority of low MSE belongs to RNN models for their excellent sequence data prediction. LSTM has a higher accuracy than RNN as expected. Despite occasional lower MSE of the auto ARIMA, statistical models do not capture price movement pattern nearly as well as deep learning models do. Seq2Seq with attention has better performance in most of the stocks it predicted than other models and adding attention mechanism to the vanilla Seq2Seq reduces the error.

Attention model is exhibiting the best performance evident from its low latency. One of the key reasons is the mechanism that take every hidden state into account for calculating each node in the decoder layer. Its superiority over LSTM arises from the feature of directly and sufficiently utilize all hidden states instead of storing only a few hidden states at a given time step. Although the ability to reserve "significant" information through training the three gates enables LSTM handle sequential data, this is still a cumulative process. By contrast, attention model is trained to assign weight to each hidden state to calculate every node within the decoder. This more direct interaction with previous hidden layers may contribute to less information loss, crucial to the forecasting trend. The writer believes attention will be even more powerful when trained to predict stocks in a longer time horizon.

Established upon LSTM, Seq2Seq models "translate" the input matrices to predicted future stock prices. The performance of the vanilla Seq2Seq model depends on how well the context vector captures the key information of the input, but with an attention layer, each hidden state of the encoder is considered at every time step of the decoder, therefore lower information loss and higher prediction accuracy.

TABLE I. Mean Squared Error for ARIMA, FBProphet, ANN, RNN, and LSTM. Bolded values are the smallest or are close to the smallest MSE.

	ARIMA mse	FB mse	ANN mse	RNN mse	LSTM mse
AABA	3.25	6.30	22.94	9.46	**2.26**
AAPL	23.37	77.64	112.44	86.20	**10.59**
AMZN	**1319.54**	2619.45	5378.36	8667.23	10545.15
AXP	6.92	18.92	7.58	**1.41**	1.79
BA	49.35	124.09	90.06	53.28	**19.33**
CAT	14.83	30.81	21.97	**3.14**	7.18
CSGO	1.23	2.89	0.52	0.75	**0.19**
CVX	11.28	19.72	17.93	**2.92**	5.03
GE	0.86	2.12	2.00	0.55	**0.25**
GOOGL	**806.73**	1616.31	6860.83	11860.43	1769.62
GS	77.21	193.41	144.79	35.21	**16.60**
HD	19.01	47.60	76.01	59.13	**8.26**
IBM	23.46	62.94	42.66	10.94	**5.78**
INTC	2.27	5.40	2.43	2.39	**1.60**
JNJ	8.99	20.65	26.99	92.93	**28.42**
JPM	**8.91**	20.10	19.11	53.84	13.72
KO	**0.72**	1.18	1.53	2.23	2.06
MCD	**11.29**	22.55	84.85	86.23	36.85
MMM	26.77	50.80	377.88	73.35	**10.91**
MRK	3.92	7.82	2.20	1.53	**1.05**
MSFT	**3.98**	7.61	12.79	44.38	11.50
NKE	5.45	10.75	4.95	3.20	**1.88**
PFE	0.90	1.85	1.17	1.19	**0.29**
PG	3.37	7.39	2.48	0.78	**0.74**
TRV	11.04	28.81	21.29	19.96	**3.26**

UNH	**21.14**	36.83	305.91	377.57	94.70
UTX	**10.02**	25.19	23.91	4.99	**11.25**
VZ	3.50	10.10	0.98	0.56	0.94
WMT	4.79	11.41	1.85	1.59	**1.00**
XOM	4.33	9.91	3.37	1.21	**1.00**

TABLE II. MEAN SQUARED ERROR COMPARISON: VANILLA SEQ2SEQ MODEL AND SEQ2SEQ WITH ATTENTION.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6
AAPL Attention	8.86	13.24	21.27	30.90	40.19	48.06
AAPL Seq2Seq	19.25	20.80	23.13	26.07	29.39	32.83
AMZN Attention	571.94	919.46	1092.55	1253.26	1472.38	1791.66
AMZN Seq2Seq	592.82	746.75	966.62	1143.61	1295.26	1423.96
BA Attention	9.30	13.93	19.89	27.73	36.62	45.23
BA Seq2Seq	17.22	19.91	24.66	30.72	37.74	44.57
CSGO Attention	0.51	0.70	1.10	1.56	1.98	2.31
CSGO Seq2Seq	0.75	1.05	1.26	1.43	1.59	1.76
GE Attention	0.21	0.31	0.38	0.45	0.54	0.62
GE Seq2Seq	0.38	0.49	0.62	0.74	0.85	0.97
IBM Attention	6.00	8.54	10.83	13.09	15.30	17.61
IBM Seq2Seq	9.58	13.39	16.03	18.76	21.44	24.03
JPM Attention	2.87	4.23	5.05	5.77	6.54	7.36
JPM Seq2Seq	2.87	3.82	4.75	5.61	6.41	7.17
KO Attention	0.19	0.29	0.44	0.63	0.80	0.93
KO Seq2Seq	0.71	0.81	0.90	0.99	1.08	1.18
MRK Attention	1.59	2.29	3.37	4.52	5.61	6.66
MRK Seq2Seq	1.03	1.44	1.77	2.13	2.48	2.83
MSFT Attention	1.67	2.18	3.63	5.30	6.61	7.31
MSFT Seq2Seq	2.23	2.55	2.97	3.40	3.84	4.22
WMT Attention	1.10	1.59	2.05	2.50	2.99	3.48
WMT Seq2Seq	1.47	2.14	2.14	3.23	3.74	4.17
XOM Attention	2.04	2.48	2.48	5.22	6.82	8.36
XOM Seq2Seq	2.90	3.86	3.86	5.29	5.93	6.49

VI. CONCLUSION AND FUTURE WORK

In this work, statistical models and deep learning algorithms have been tested for their performance in stock price forecasting. Conventional time-series models, such as ARIMA, tend to be more explainable but don't capture the trend well. Long short-term memory (LSTM) proves itself to be reliable than vanilla ANN and RNN. Built upon the LSTM model, Seq2Seq models are excellent predictors for stock prices of several days in the future with even lower mean squared error. Seq2Seq model with attention has the least MSE in forecasting the next day's close price while having a nice fit to the trend. Both the comparison among ANN, RNN, and LSTM and the comparison between two Seq2Seq models indicate that the increased complexity of neural network does generate better performance.

The only error measurement used is the mean squared error. More demonstrative metrics such as MAPE and AUC could be included in the comparison of performance. Additional features could be incorporated and selected before training, and these models could be applied for volatility prediction. More research is needed to determine if any of the models compared in this study could be transformed into trading strategies. Furthermore, Seq2Seq models could be trained with input data of larger total time steps to forecast more future price information, i.e., weeks or months in the future, which could potentially assist short-term investors.

ACKNOWLEDGMENT

This project would not be possible without the help from the listed individuals.

I wish to express my sincere gratitude to Professor Zhao Feng for his guidance and provision of valuable advice for my paper.

I also wish to extend my thanks to Andrew Ng and Hung-Yi Lee, whose excellent machine learning lectures introduced me to the fascinating field of artificial intelligence.

Special thanks to Yuan-yuan Du, my college counselor, for her encouragement and support.

REFERENCES

[1] Yong Hu et al. “Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review”. In: Applied Soft Computing 36 (2015), pp. 534–551.

[2] Dev Shah et al. “Stock Market Analysis: A Review and Taxonomy of Prediction Techniques”. In: International Journal of Finance Studies (2019). doi: 10.3390/ijfs7020026.

[3] De Faria E. L. et al. “Predicting the Brazilian Stock Market through Neural Networks and Adaptive Exponential Smoothing Methods”. In: Expert Systems with Applications 36 (2010), pp. 12506–12509.

[4] Ariyo Adebiyi A., Adewumi O. Adewumi, and Charles K. Ayo. “Stock Price Prediction Using the Arima Model”. In: Paper presented at the 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation (UKSim), Cambridge, UK ().

[5] Luca Di Persio and Oleksandr Honchar. “Recurrent Neural Networks Approach to the Financial Forecast of Google Assets”. In: International Journal of Mathematics and Computers in simulation 11 (2017), pp. 7–13.

[6] Dev Shah, Campbell Wesley, and Zulkernine Farhana. “A Comparative Study of LSTM and DNN for Stock Market Forecasting”. In: Paper presented at the 2018 IEEE International Conference on Big Data, Seattle, WA, USA ().

[7] I. Karlsson J. Rebane and P. Papapetrou S. Denic. “Seq2Seq RNNs and ARIMA models for Cryptocurrency Prediction: A Comparative Study”. In: SIGKDD Fintec 18 (2018). J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.