A Novel Hybrid Deep Learning Model For Stock Price Forecasting

Dhaifallah Alghamdi
Department of Industrial Engineering
University of Pittsburgh
Pittsburgh, USA
dha8@pitt.edu

Faris Alotaibi
Department of Informatics and Networked Systems
University of Pittsburgh
Pittsburgh, USA
f.alotibi@pitt.edu

Jayant Rajgopal Ph.D.
Department of Industrial Engineering
University of Pittsburgh
Pittsburgh, USA
gunner1@pitt.edu

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533553

***Abstract*—Stock price prediction is a challenging task due to its complexity and the dynamics associated with stock prices. In this paper, we propose a deep-learning based end-to-end framework with a novel architecture, for multi-step ahead stock closing price forecasts. The architecture exploits an encoder-decoder framework with variants of convolutions and recurrent neurons, in order to perform representation learning for the past behavior of the stock as well as associated exogenous factors.We incorporate an attention mechanism to capture the long term dependencies between inputs and outputs, and deploy Monte Carlo dropout layers in the architecture design to provide a stochastic setting for uncertainty estimation. We validate our model on two real datasets for AMZN and AAPL stocks.**

***Index Terms*—Stock Price Prediction, Predictive Modeling, Deep Learning, Attention, Uncertainty Estimation, Temporal Convolution Network (TCN), Feature Learning.**

I. Introduction

THE stock market is an essential component of any open economy and a major indicator for gauging the economic health of a country. Many factors affect the behavior of the stock market, including political & social conditions, the state of the global economy, and financial performance of enterprises. Generally speaking, in the finance community, investing strategies can be divided into three categories: fundamental, technical and quantitative. Investors who follow the fundamental strategy focus mainly on the performance of an individual company and its track record to value a stock, while technical analysis considers only historical time series data to discover patterns. The core of the quantitative strategy is to use mathematical and statistical modeling to map historical time-series, along with other external factors, to the future stock price. Forecasting stock prices is a challenging task in the finance world due to its complexity and the dynamics associated with stock prices. We propose an approach that is similar to technical analysis in that we use time-series data to make forecasts.

Most prior work in the area of time-series stock price forecasting has used classical models such as Auto-Regressive Moving Average (ARMA) [1], linear regression [2], and even simple moving averages [3] [4]. These models are based on parameterizing pre-defined equations to fit a mathematical model to a sequence of historical data. The major drawback of these models is that they lack the ability to capture the natural non-linear dynamics in the data. Also, these models cannot incorporate external factors (e.g., interactions between different companies) since they only consider univariate time-series. This limitation largely affects the ability to generalize one model based on historical time-series data of a specific stock, to other stocks. Apart from the classical models, traditional machine learning algorithms such as support vector machines (SVM), random forest (RF), and artificial neural networks (ANNs) have also been used for financial time-series forecasting and these generally yield higher forecast accuracy [5] [6] [7]. However, these models still assume a pre-determined non-linear mathematical form, which may not capture the true underlying nonlinear relationship. Another line of research uses recent advances in deep learning for stock price forecasting [8] [9] [10] [11] [12] [13] [14] [15] [16]. However, most of the work has focused only on deterministic next step prediction, or classification of the stock price direction.

In this study, we investigate the problem of obtaining reliable forecasts for multi-step-ahead stock prices for a target company by using deep learning. The technical challenges to overcome include complex and nonlinear interactions, combining time-series with exogenous factors in financial sequence modeling, achieving acceptable performance, and quantifying the uncertainty associated with the model's output in a probabilistic setting. We believe that forecasting multi-step ahead stock price for a particular company is essential in order to make better informed trading decisions. We also believe that in addition to the past performance record of the target company, the future stock price is dependent on both the impact of and the correlation with stock prices of other companies. Therefore, our framework will incorporate this along with other technical and macroeconomic indicators.

In order to address the challenges mentioned above, we propose an end-to-end feature learning framework with a novel architecture for multi-step stock price forecasting. Similar to the concept of converting a natural language sentence to a word vector embedding, we believe finding an appropriate embedding for the stock price history will enhance the learning process. Hence, we first encode a time-series sequence of historical records for the target company. This is then decoded to forecast the multi-step ahead closing prices. More specifically, in the encoder, the sequence of historical records are passed through a temporal convolutional network to extract the stock price's hidden characteristics. We then employ an attention mechanism to encapsulate the portions of the input sequence on which we should focus for each time-step in the forecast horizon. Concurrently, exogenous factors are mapped to a lower dimensional latent representation by passing them through an auto-encoder. For uncertainty estimation, we exploit Monte Carlo (MC) dropout layers and keep them activated during inference.

The reminder of this paper is organized as follows. Section 2 discusses background. Section 3 provides the problem definition and details of our proposed framework. Section 4 covers the experimental study, including the settings and results. Finally, in Section 5, we provide a summary and conclusions.

II. Background

A. Learning Representation

It is widely believed that learning representations is one of the main factors associated with the success of deep neural networks. In fact, the performance of machine learning algorithms can be greatly affected by the choice of data representation. Finding good representations for complex input data helps algorithms better understand the data and do the necessary processing. The computational aspect is also crucial because mapping large, complex data to a lower-dimensional space contributes to the algorithm's computational efficiency. The lower dimensional representation can also help avoid overfitting and yield better generalization to unseen data. Principal component analysis (PCA) is a fundamental technique in dimensionality reduction, where the raw data with p features is projected linearly to a q dimensional vector where $q <= p$ [17]. There are many equivalent mathematical ways for deriving the principal components. The first principal component is the direction which explains a maximal amount of variance in the data (the eigenvector corresponding to the largest eigenvalue of the covariance matrix for the data). The k^{th} component is the variance-maximizing direction orthogonal to the previous $k-1$ components. The orthogonality condition ensures this component is uncorrelated with the preceding components.

Another learning representation method that is widely used in deep learning is Restricted Boltzmann Machines (RBM). RBM is a shallow artificial neural network consists of two layers: visible and hidden. These two layers are connected by a fully bipartite graph, where no two nodes within the same layer are connected and every node in the visible layer is connected to every node in the hidden layer and vice versa. There are three primary steps in the learning process: 1) forward pass 2) backward pass 3) reconstruction error calculation. We pass the input to the hidden layer, and then reconstruct the input from the hidden layer. The loss function computes the distance between the generated output and the original input. Thus, RBM is an unsupervised learning algorithm that leverages back-propagation to find the best representation that can be used in order to later reconstruct the input. Auto-encoders have the same concept but with a deep network instead of a shallow one.

B. Dropout in deep learning

In deep learning, dropout layers are used for regularization in order to avoid overfitting [18]. Reduction in generalization error can be achieved by randomly dropping neurons of the network during training. This technique can be further expanded to quantify uncertainty [19]. The generated predictions of the network can be non-deterministic if Monte Carlo (MC) dropout is used. The main distinction is that MC dropout layers remain active during inference for multiple runs. Thus, for the same input, running the trained model multiple times will generate multiple predictions. The predictions generated can be viewed as samples from a probabilistic distribution. There are two main advantages of this framework: 1) it provides uncertainty estimation without any additional computational effort, and 2) it does not require any change to the network architecture.

III. Proposed Framework

In this section we introduce the notation used in this study, and then describe the details of our framework.

A. Notation and Problem Definition

Looking into the past for t days, we define $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_t\} \in \mathbb{R}^{tm}$ as the stock price time-series for the target company we are interested in, where $\mathbf{S}_t = \{H_t, L_t, O_t, C_t, V_t\}$ and m is the number of features in $\mathbf{S}_t$. Here, $\mathbf{S}_t$ represents the stock price vector for the target company at time t, and is composed of the highest (H_t), lowest (L_t), opening (O_t) and closing (C_t) prices as well as the volume (V_t) of traded stocks for the company. We also use $\mathbf{X}_t$ to represent the external factors at time t, which includes stock prices of other companies as well as some relevant macroeconomic indicators. The ground truth output is the closing price for the next F days $\mathbf{C} = C_{t+1}, ..., C_{t+F}$, while the predicted value is $\hat{\mathbf{C}} = \hat{C}_{t+1}, ..., \hat{C}_{t+F}$. The aim of our model is to learn a non-linear mapping σ to the multi-step ahead closing price:

$$\hat{\mathbf{C}} = \sigma(\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_t, \mathbf{X}_t)$$

B. Model

Our deep learning model has two sub-models that require sequential training. The aim of the first is to map the exogenous factors into a lower-dimensional space. Then, we learn a latent representation for the stock price history and concatenate it to the output of the first model to forecast future stock price. We now describe each sub-model in details.

1) Learning Embedding of Exogenous Factors: The main task of this portion of the model is to learn a suitable embedding for the exogenous factors. To achieve this, we deploy an auto-encoder architecture consisting of encoder and decoder stages. The auto-encoder is different from PCA since the orthogonality condition is relaxed in addition to stacking multiple linear layers, with a nonlinear activation function. In the encoder stage, we compress the exogenous factors vector into a lower-dimensional vector by passing it through three linear layers with a rectified linear units (ReLU) activation function. Then, in the decoder stage, we reconstruct the original input from the compressed representation by passing it through similar stacked layers in reverse order. Hence, the architecture presents a bottleneck in the middle, from which the reconstruction of the input data is implemented. Figure 1 shows the details of our auto-encoder. More formally, we split the network into two segments: the encoder ζ and the decoder ϑ.

$$\begin{aligned} \zeta &: \mathbf{X} \rightarrow \mathbf{Z} \\ \vartheta &: \mathbf{Z} \rightarrow \mathbf{X}' \end{aligned} \tag{1}$$

The encoder and decoder can be represented by a standard neural network, where $\mathbf{X}$, $\mathbf{X}'$, σ and $\mathbf{Z}$ are the exogenous factors vector, the reconstructed vector, the activation function and the bottleneck representation, respectively. $\mathbf{W}$, $\mathbf{b}$, $\mathbf{W}'$, and $\mathbf{b}'$ represent the encoder and decoder weights and biases.

$$\begin{aligned} \mathbf{Z} &= \sigma(\mathbf{W}\mathbf{X} + \mathbf{b}) \\ \mathbf{X}' &= \sigma'(\mathbf{W}'\mathbf{Z} + \mathbf{b}') \\ Objectivefunction &: \min_{\zeta,\vartheta} ||\mathbf{X} - (\zeta \circ \vartheta)\mathbf{X}|| \end{aligned} \tag{2}$$

The loss function of the auto-encoder is the squared distance between the original input and the constructed one:

$$\mathcal{L} = ||\mathbf{X} - \mathbf{X}'||^2 = ||\mathbf{X} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{X} + \mathbf{b})) + \mathbf{b}')||^2 \tag{3}$$

Algorithm 1 Auto-Encoder training algorithm

```
Input: Dataset $\mathbf{X}_1, ...., \mathbf{X}_n$
Output: Encoder $\zeta$, Decoder $\vartheta$
1: Initialize parameters $\zeta(.) = \mathbf{W}, \mathbf{b}$ & $\vartheta(.) = \mathbf{W}', \mathbf{b}'$
2: repeat
3:     for $i$ = 1, ..., $N$ do
4:         $\mathcal{L} = \sum_{i=1}^{N} ||\mathbf{X}_i - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{X}_i + \mathbf{b})) + \mathbf{b}')||^2$
           calculate sum of reconstruction errors
5:         $\zeta(.)\vartheta(.) \longleftarrow$ update encoder decoder parameters
6:     end for
7: until convergence of parameters $\zeta(.)$ & $\vartheta(.)$
```

2) Time-series encoder decoder: We leverage the encoder decoder framework to understand time-series history and forecast future stock prices. The main assumption here is that an appropriate embedding for the stock price history can result in more accurate forecasts. The advantage of this approach is that it removes noise from the original input and captures only important local features. Hence, the main objective of the encoder in our framework is to map the history of stock price $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_{t-1}\}$ to a latent representation of a fixed dimension vector. The vector is then passed to the decoder to generate the predictions. Our model is shown in Fig. 2. The encoder usually consists of stacked Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers; however, we

Algorithm 2 Training of CAED-TCN

```
Input: training data $\{\mathbf{S}_1, ...., \mathbf{S}_N\}$, $\{\mathbf{X_1}, ..., \mathbf{X_N}\}$, lookback
    window size $k$, Forecast window $F$, trained auto-encoder
    $\zeta(.)$ & $\vartheta(.)$
Output: CAED-TCN model $\mathcal{M}(.)$
1: Initialize the encoder $\mathcal{G}(.)$, and decoder $\mathcal{F}(.)$ in Fig. 2
2: while stopping criteria not met do
3:     for $t$ = k, ...,N do
4:         $\mathbf{k}_t, \mathbf{v}_t = \mathcal{G}(S_{t-k}, ..., S_t)$ encoder output
5:         Decoder:
6:         for $\tau = t, ..., t + F$ do
7:
               $C_\tau = \begin{cases} \hat{C}_\tau & \text{w.p. } p \\ C_\tau & \text{otherwise} \end{cases}$   teacher forcing case
8:             $\nu_\tau = \phi(C_\tau)$, where $\phi(.)$ is a linear layer
9:             For each LSTM cell and hidden state(h and r are
               the two outputs of the LSTM)
               $\{d_\tau, h_\tau\} = \begin{cases} lstm(\nu_\tau, \mathbf{h}^l_{\tau-1}, \mathbf{r}_{\tau-1}), \text{ if } l = 1 \\ lstm(\mathbf{d}^{l-1}_\tau, \mathbf{h}^l_{\text{initialized}}, \mathbf{0}), \text{ if } \tau = 1 \\ lstm(\mathbf{d}^{l-1}_\tau, \mathbf{h}^l_{\tau-1}, \mathbf{r}_{\tau-1}), \text{ otherwise} \end{cases}$
10:            use $\mathbf{k}_t, \mathbf{v}_t$ to compute the attention vector $a_\tau$
11:            $Z_\tau = \zeta(X_{\tau-1})$
12:            $i_\tau = concatenate[a_\tau, \mathbf{r}_\tau, Z_\tau]$
13:            $\hat{C}_\tau = MLP(\{i\}_\tau)$
14:        end for
15:        perform backward passes to update parameters of
           $\mathcal{G}(.)$ and $\mathcal{F}(.)$ by minimizing the loss function
           $\mathcal{L}(\hat{\mathbf{C}}_t, \mathbf{C}_t)$
16:    end for
17: end while
```

use a Temporal Convolution Network (TCN) architecture in our encoder. The encoder receives the financial sequence $\mathbf{S}_t$ and passes it through the TCN to extract local characteristics of the time series. The TCN reduces the total number of trainable parameters through the concept of parameter sharing, and by leveraging local connectivity of convolution layers. The architecture of TCN can only handle input-output sequences of the same length. Another important characteristic of TCN architecture is the causality condition for the convolutions, i.e. an output y_t at time t is convolved only with elements y_τ in the previous layer, where $\tau \leq t$ [20].

Simple Causal convolutions are less preferred to dilated convolutions due to the ability of the latter to allow the receptive field to grow exponentially as we increase the number of layers. Mathematically, given an input sequence, $\mathbf{x} \in \mathbb{R}^k$ and a kernel function $\psi(.) : \{0, \dots, k-1\} \longrightarrow \mathbb{R}$, the dilated convolution $\mathbf{C}(.)$ on element s of the sequence is defined as

$$\mathbf{C}(s) = (\psi \circledast_\delta \mathbf{x})(s) = \sum_{i=0}^{k-1} \psi(i) \, . \, \mathbf{x}_{s-\delta i}, \tag{4}$$

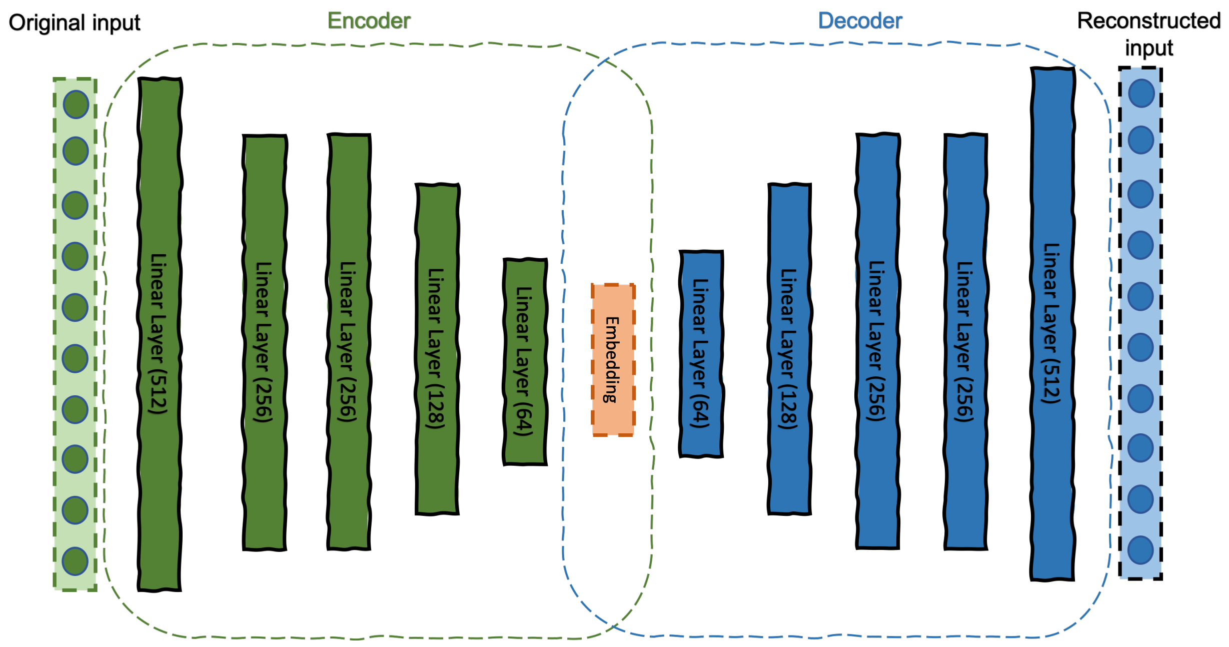

Fig. 1: Learning exogenous factors embedding

where $\delta = 2^\eta$ is the dilation factor, η is the depth of the network, N is the kernel size, and $\mathbf{s} - \delta i$ accounts for the direction of the past [20].

The encoder consists of two stacked residual blocks each consisting of two dilated convolution layers. The kernels of both layers are normalized and passed through ReLU activation function, followed by a dropout layer for regularization purposes. The problem of exploding/vanishing gradients is handled by these blocks. The encoder produces the latent representation in the form of two parallel linear layers, `keys` and `values`, as shown in Figure 2. This specific form of output is necessary in the implementation of the attention mechanism in our model.

The most recent ground truth stock price vector (with probability p) or the most recent prediction generated (with probability $(1-p)$) is used to initialize the decoder. The decoder consists of two linear layers, followed by two LSTM layers, an attention model, a CNN layer (Convolutional Neural Network), and a two-layer MLP (multi-layer perceptron). The decoder forecasts the future stock closing price $\hat{\mathbf{C}}$ for the target company. For random instances in the forecast horizon, with predetermined probability p, we feed the decoder with the forecast generated at the previous step instead of the associated ground truth. This technique helps with training the model to avoid the propagation of any inaccurate prediction to subsequent forecasts in the forecast horizon.

We also integrate the general attention mechanism in our model, which is a component of our network's architecture, and is in charge of quantifying the interdependence between the input and the output [21]. It helps the model focus on the most important segments of the input sequence at each time step in the forecast horizon. More precisely, for each time step in the forecast horizon, we pass the current output of the last LSTM layer along with the output of the encoder, `keys` and `values`, to the attention function. Then, through batch matrix multiplication, we obtain the attention context, which identifies the significant input segment at the current time step.

Algorithm 3 Inference of CAED-TCN

```
Input: testing data $\{\mathbf{S}_t\}$; trained CAED-TCN model $\mathcal{M}(.)$,
    exogenous factors vector $\mathbf{X_t}$, dropout probability $p$, number of iterations $N$
Output: prediction mean $\mu_{\mathbf{J}_t}$ and uncertainty $\xi_{\mathbf{J}_t}$
1: for $i = 1, \ldots, N$ do
2:     $\hat{\mathbf{C}}_t^i = Dropout(\mathcal{M}(\mathbf{C}_t, X_t), p)$
3: end for
4: $\mu_{\mathbf{C}_t} = \frac{1}{N} \sum_{i=1}^{N} \hat{\mathbf{C}}_t^i$
5: $\xi^2_{\mathbf{C}_t} = \frac{1}{N} \sum_{i=1}^{N} (\hat{\mathbf{C}}_t^i - \mu_{\mathbf{C}_t})^2$
6: return $\mu_{\mathbf{C}_t}$, $\xi_{\mathbf{C}_t}$
```

IV. Experimental Study

In this section, we evaluate our model and compare it to baseline models from the literature. We start by describing the dataset used in our study along with the experimental settings and then provide an analysis of the results.

A. Dataset

We consider two datasets to train and evaluate our framework. Amazon is the target company in the first dataset, while we use Apple in the second.The selection is based on the large trading volume of these enterprises, which affects the entire market and the SP 500 companies in particular, since

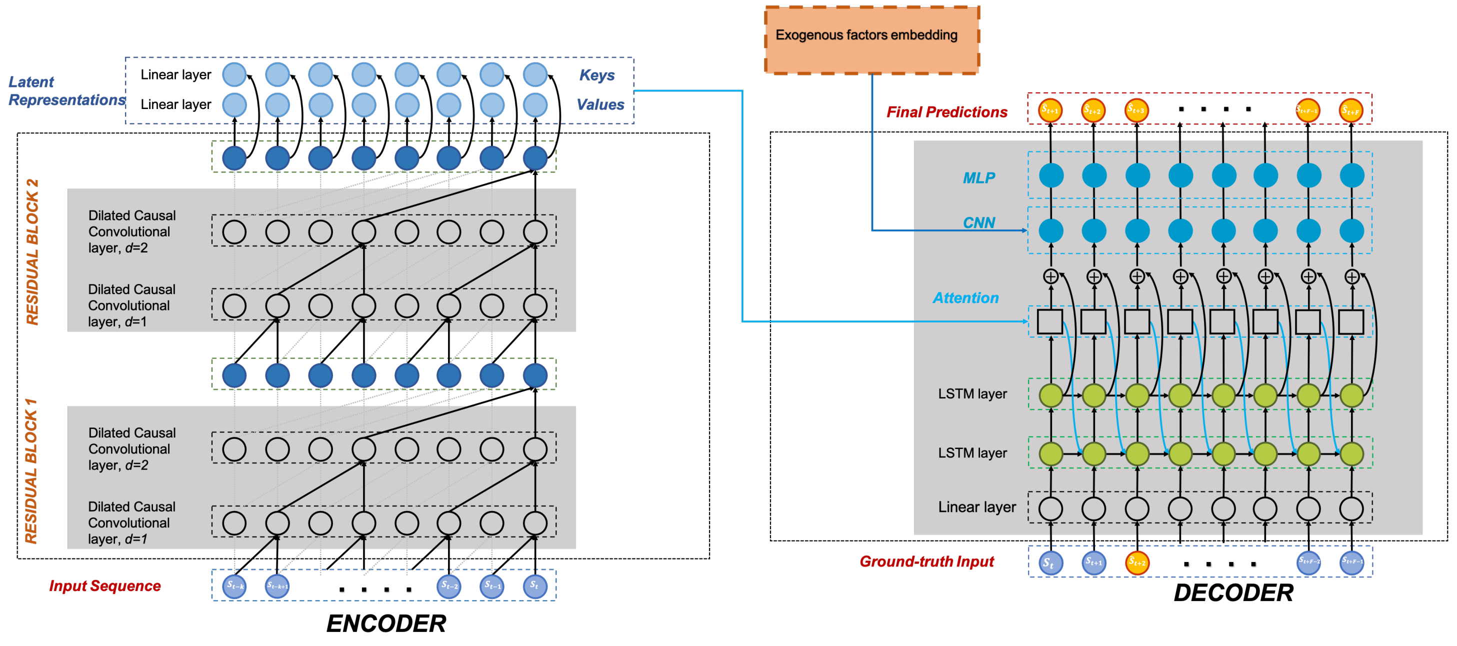

Fig. 2: Model

we aim to study the effect of market indices (e.g. SP 500) on individual stocks. We fetched the historical multivariate time-series data from the *Yahoo Finance* website for the last 10 years. The time-series includes the daily open, close, low, and high prices in addition to the traded volume. For the exogenous factors, we obtained the closing price for the rest of the S&P 500 companies, as well as the NASDAQ, S&P 500, and Dow Jones indices. We also used VIX, *a.k.a* the Chicago Board Options Exchange (CBOE) index, which is a real-time market index that represents the market's expectation of 30-day forward-looking volatility. In addition, we performed feature engineering to obtain technical indicators such as the moving average based on different rolling values, the difference in traded volumes over the previous two days, as well as the time-series seasonality, trend and residual for the target company. Furthermore, we normalized all inputs to ensure learning stability and efficiency [22].

B. Experimental Setup

We evaluate the performance of our confident, attentive, encoder-decoder model with TCN architecture (CAED-TCN), and compare it to the performance of the following baselines, where each is implemented with with the necessary fine-tuning.

- MLP: A multi layer perceptron composed of stacked linear layers
- Vanila LSTM: A single LSTM layers followed by an output layer
- Multi-LSTM: Three LSTM layers followed by an output layer.
- Encoder-Decoder LSTM: An encoder and a decoder, each with two LSTM layers.
- Modified LAS (Chan, William 2015 [21]): Listen attend and spell (LAS) is an encoder-decoder sequence model with attention. The structure of the encoder consists of 3 pyramidal bidirectional LSTM (pBLSTM) layers and the decoder is composed of 2 BLSTM layers.

The sliding window size for our input sequence is 50 days, which we used to forecast over three different forecast horizon lengths (one, three and five steps ahead). We also evaluated the impact of exogenous factors and the correlation with other companies on the performance of the model. The evaluation metrics used in our analysis are: Root Mean Square Error ($RMSE$), Symmetric Mean Absolute Percentage Error ($sMAPE$), and Smooth L1 ($sL1$). Although learner sensitivity to outliers is not the focus of our work and can be further investigated in future studies, the effect of anomalies are better captured by RMSE and sMAPE. The last measure, also called Huber loss, is less sensitive to outliers and uses a squared term only if the absolute error is under 1.

$$RMSE = \frac{1}{T}\sqrt{\sum_{t=1}^{T}(C_t - \hat{C}_t)^2}$$

$$sMAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|C_t - \hat{C}_t|}{(|C_t| - |\hat{C}_t|)/2 + 1}$$

$$sL1 = \frac{1}{T}\sum_{t=1}^{T} z_t$$

where z_t is given by:

$$z_t = \begin{cases} 0.5 * (C_t - \hat{C}_t)^2, & \text{if } |C_t - \hat{C}_t| < 1 \\ |C_t - \hat{C}_t| - 0.5, & \text{otherwise} \end{cases}$$

TABLE I: Performance of the Models

Forecast window		One-step			Three-step			Five-step		
Dataset	Model	$RMSE$	$sMAPE$	$sL1$	$RMSE$	$sMAPE$	$sL1$	$RMSE$	$sMAPE$	sL1
	MLP	0.068	0.0033	0.057	0.073	0.0032	0.057	0.074	0.0036	0.059
	Vanilla LSTM	0.069	0.0034	0.065	0.073	0.0031	0.058	0.074	0.0037	0.059
AMZN	multi-LSTM	0.067	0.0033	0.055	0.067	0.0032	0.054	0.068	0.0033	0.054
	Enc-Dec LSTM	0.064	0.0030	0.052	0.067	0.0031	0.054	0.067	0.0032	0.053
	modified LAS	0.048	0.0011	0.043	0.048	0.0014	0.037	0.049	0.0015	0.037
	CAED-TCN	**0.026**	**0.0004**	**0.021**	**0.035**	**0.0010**	**0.034**	**0.042**	**0.007**	**0.028**
	MLP	0.044	0.0010	0.039	0.058	0.0017	0.051	0.064	0.0021	0.059
	Vanilla LSTM	0.043	0.0012	0.039	0.047	0.0011	0.039	0.050	0.0012	0.041
AAPL	multi-LSTM	0.039	0.0007	0.033	0.040	0.0009	0.034	0.046	0.0011	0.041
	Enc-Dec LSTM	0.062	0.002	0.058	0.064	0.0021	0.060	0.074	0.0029	0.070
	modified LAS	0.025	0.0003	0.023	0.040	0.0008	0.033	0.044	0.0009	0.038
	CAED-TCN	**0.017**	**0.0001**	**0.0142**	**0.035**	**0.0006**	**0.031**	**0.035**	**0.0008**	**0.029**

During inference, for uncertainty estimation in our forecasts, we focus on the one-step-ahead ($o-s-a$) forecasts and run the model for 1000 epochs while keeping the MC dropout activated. For the same input, this process generates 1000 different $o-s-a$ forecasts for each period, which in turn allows us to build a confidence interval for the forecast for each period. We constructed both 90% & 95% confidence intervals, and we then used the coverage probability (CP metric to evaluate the uncertainty estimation. Here CP is the proportion of the confidence intervals constructed that contain the ground truth being forecast.

C. Results

TABLE II: Incorporation of Additional Information & Attention

Dataset	Model	$RMSE$	$sMAPE$	$sL1$
	univariate	0.048	0.0014	0.039
AMZN	multivariate	0.042	0.0012	0.034
	multivariate & exogenous factors	**0.035**	**0.0007**	**0.028**
	multivariate & exogenous factors w/o attention	0.041	0.0010	0.034
	univariate	0.071	0.0029	0.064
AAPL	multivariate	0.051	0.0013	0.045
	multivariate & exogenous factors	**0.035**	**0.0008**	**0.029**
	multivariate & exogenous factors w/o attention	0.047	**0.0011**	0.042

TABLE III: Coverage Probability

Dataset	Model	CP at 99%	CP at 95%
AMZN	Modified LAS	0.71	0.57
	CAED-TCN	**0.89**	**0.83**
APPL	Modified LAS	0.87	0.71
	CAED-TCN	**0.98**	**0.87**

Table I shows the performance, based on our three metrics, of our model and the other five models, evaluated over the entire test dataset for both the AMZN and the AAPL datasets for three different forecast horizon lengths. As expected, the performance of all models is inversely proportional to the length of the forecast window (except in a couple of cases, with respect to sL1, which treats errors selectively). This observation agrees with the fundamental forecasting concept where accuracy degrades as we look further into the future.

In looking at the baseline models, although Vanilla LSTM is designed to capture long-term relationships, MLP actually performs as well or slightly better, which might reflect the fact that the depth of MLP can provide a more meaningful representation. However, both MLP and vanilla LSTM have the lowest performance metrics relative to the other models due, indicating that simple models do not efficiently capture stock market dynamics. Modifying the vanilla LSTM model by stacking multiple LSTM layers (multi-LSTM) improves the performance since the model now is deep and with LSTM components. The model performs even better with the adoption of the encoder-decoder framework. The key to this performance enhancement is the latent representation learning performed within this framework. Finally, the modified LAS outperforms all previous models with the adoption of an attention mechanism. The main advantage of adopting the attention mechanism is to exploit the long-term associations between inputs and outputs of the dataset.

As Table I illustrates, our model, CAED-TCN, is superior to the baseline models with respect to all metrics and all forecast horizons. In fact, even the five-steps-ahead forecasts from our model outperform the one-step-ahead forecasts from all of the baseline models. There are two major reasons for this: 1) the adoption of the TCN architecture for the encoder, and 2) the incorporation of the learned representation for exogenous factors. In addition to the enhancement provided by the attention mechanism, teacher forcing and the representation learning of the exogenous factors, the TCN encoder exploits the local connectivity and parameter-sharing to provide more efficient and stable learning. Although Modified LAS provides the closest performance to our model, it requires learning $13,680,500$ parameters while our model only has $1,362,688$ learnable parameters. This is an essential contribution of our work, where we reduced the learning computational complexity by approximately 90%.

Next, we focus on the five-steps ahead forecasts and illustrate the importance of incorporating the exogenous factors as well as the multivariate time-series data in our model. The univariate time-series input contains only closing price history, while the multivariate time-series includes all values in $\mathbf{S}_t$. Clearly, learning the latent representation for the multivariate time-series feeds the decoder with more information that better explains the past behavior of the target company stock. For AMZN this modification to the stock history encoding

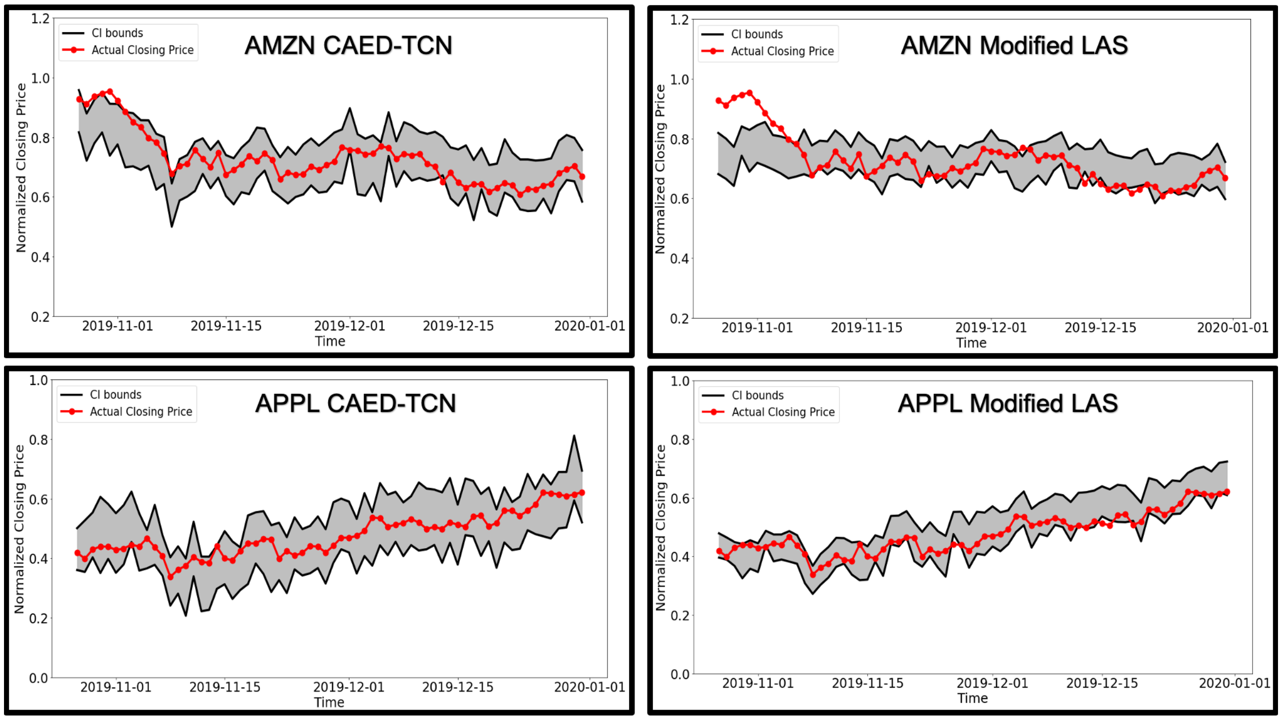

Fig. 3: 99% Confidence intervals

yields a reduction of around 12%, 14% and 13% for the $RMSE$, $sMAPE$ and $sL1$ respectively, as shown in Table II. Furthermore, the representation learning of the external factors discussed in Section IV.A advanced the performance by 16% for the $RMSE$, 41% for $sMAPE$ and 17% for the $sL1$. Similarly, for AAPL stock, the incorporation of multi-variate time-series reduced the RMSE by 28%, while the incorporation of exogenous factors led to a further 16% reduction. The same general trend can be observed for the other evaluation metrics. Therefore, the analysis here suggests that including exogenous factors and the more comprehensive encoding for the stock price history enhances the learning of our model. Furthermore, the table illustrates the importance of the attention related aspect in our architecture for the learning enhancement process.

Finally, we end our discussion by examining the uncertainty estimation associated with our model to provide some measure of confidence in our forecasts. The scope considered here is one-step ahead prediction, and the comparison is limited to our CAED-TCN model and the most competitive baseline model (modified LAS). Table III shows the coverage probability (CP) for both models for both datasets. The CP of our CAED-TCN is higher than the CP of modified LAS for both datasets across different confidence levels. More precisely, at a 95% confidence level, the CP for CAED-TCN exceeds that of the modified LAS by 26% and 16% for AMZN and AAPL respectively. The same pattern is observed at a 99% confidence level with a 27% advantage for the AMZN dataset and 11% for AAPL. Our model was able to capture the movement of the closing price with narrower confidence intervals for both stocks 3& 4. These figures provide more support to the conclusion that CAED-TCN is superior to the modified LAS model.

V. Conclusion

In this paper we propose a confident, attentive, encoder-decoder with TCN (CAED-TCN) model, a novel deep-learning based approach for stock closing price predictions. We address several technical challenges such as representation learning for exogenous factors, latent representation for multivariate time-series, model robustness, and forecast uncertainty estimation. We first design the auto-encoder to learn how to represent exogenous factors. Then, we leverage the encoder-decoder framework to map the historical records of the target stock to a latent representation. Our design for the encoder consists of a temporal convolution network (TCN) structure, while the decoder has two stacked LSTM layers followed by a convolution layer and an MLP. A general attention mechanism is deployed, and a teacher-forcing policy helps the model to learn how to recover from early mistakes in the forecast horizon.

We learn that representation learning for exogenous factors, and additional historical time-series data incorporation enhance the performance of our model. We quantify uncertainty estimation by designing the architecture with Monte Carlo dropout layers. This step is done during inference by running the model multiple times to build a confidence interval instead of relying on a single forecast. Our experimental study on AMZN and AAPL stocks demonstrates that our model outperforms other advanced models.

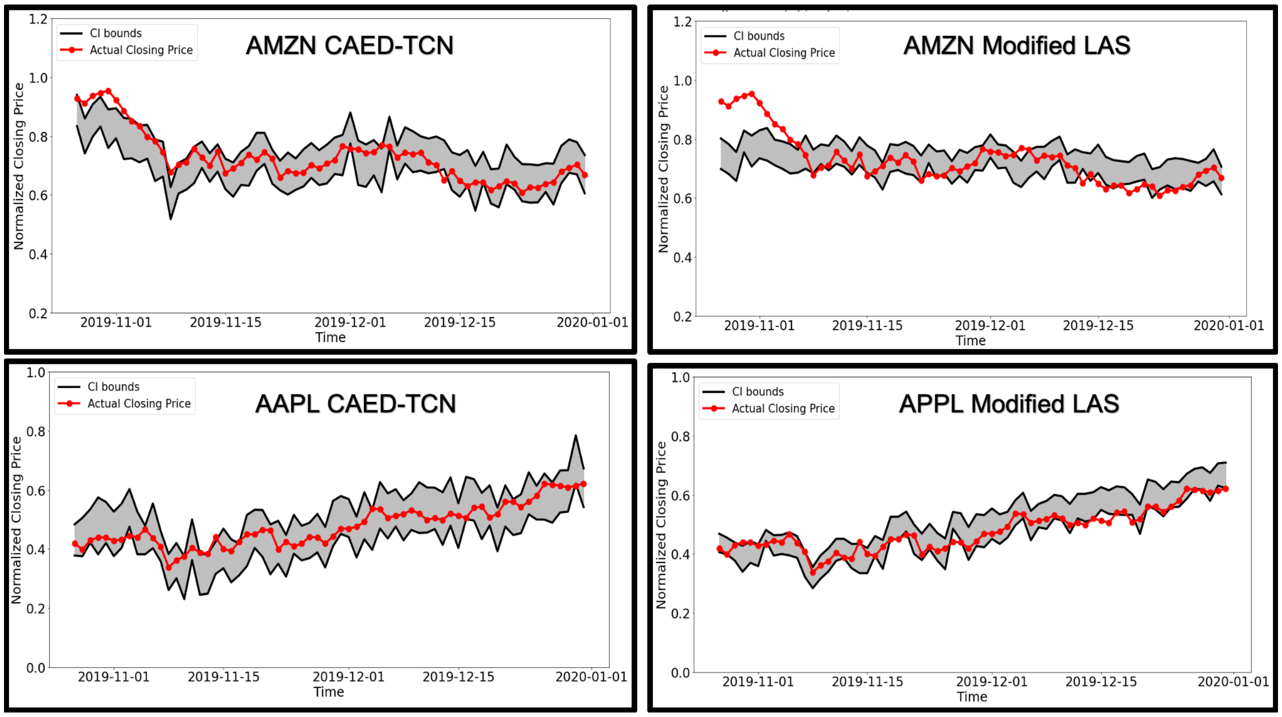

Fig. 4: 95% Confidence intervals

The next step would be to extend the model to perform online learning and predict stock prices in real time. It can also be deployed as an input model to design a more comprehensive automated trading system. Another promising direction for this work, is to tailor this model to fit other application domains with relatively similar underlying structure.

REFERENCES

[1] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.
[2] Y. E. Cakra and B. D. Trisedya, "Stock price prediction using linear regression based on sentiment analysis," in *2015 international conference on advanced computer science and information systems (ICACSIS)*. IEEE, 2015, pp. 147–154.
[3] S. Fifield, D. Power, and D. Knipe, "The performance of moving average rules in emerging stock markets," *Applied Financial Economics*, vol. 18, no. 19, pp. 1515–1532, 2008.
[4] T. C. Mills, "Technical analysis and the london stock exchange: Testing trading rules using the ft30," *International Journal of Finance & Economics*, vol. 2, no. 4, pp. 319–331, 1997.
[5] M. Ballings, D. Van den Poel, N. Hespeels, and R. Gryp, "Evaluating multiple classifiers for stock price direction prediction," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7046–7056, 2015.
[6] M. Kumar and M. Thenmozhi, "Forecasting stock index movement: A comparison of support vector machines and random forest," in *Indian institute of capital markets 9th capital markets conference paper*, 2006.
[7] S. P. Das and S. Padhy, "Support vector machines for prediction of futures prices in indian stock market," *International Journal of Computer Applications*, vol. 41, no. 3, 2012.
[8] W. Long, Z. Lu, and L. Cui, "Deep learning-based feature engineering for stock price movement prediction," *Knowledge-Based Systems*, vol. 164, pp. 163–173, 2019.
[9] A. Arévalo, J. Niño, G. Hernández, and J. Sandoval, "High-frequency trading strategy based on deep neural networks," in *International conference on intelligent computing*. Springer, 2016, pp. 424–436.
[10] 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.
[11] K. Khare, O. Darekar, P. Gupta, and V. Attar, "Short term stock price prediction using deep learning," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2017, pp. 482–486.
[12] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using lstm, rnn and cnn-sliding window model," in *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE, 2017, pp. 1643–1647.
[13] M. A. I. Sunny, M. M. S. Maswood, and A. G. Alharbi, "Deep learning-based stock price prediction using lstm and bi-directional lstm model," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE, 2020, pp. 87–92.
[14] S. Mehtab and J. Sen, "A time series analysis-based stock price prediction using machine learning and deep learning models," *arXiv preprint arXiv:2004.11697*, 2020.
[15] P. Yu and X. Yan, "Stock price prediction based on deep neural networks," *Neural Computing and Applications*, vol. 32, no. 6, pp. 1609–1628, 2020.
[16] X. Wang, Y. Wang, B. Weng, and A. Vinel, "Stock2vec: A hybrid deep learning framework for stock market prediction with representation learning and temporal convolutional network," *arXiv preprint arXiv:2010.01197*, 2020.
[17] M. Ringnér, "What is principal component analysis?" *Nature biotechnology*, vol. 26, no. 3, pp. 303–304, 2008.
[18] N. Srivastava, "Improving neural networks with dropout," *University of Toronto*, vol. 182, no. 566, p. 7, 2013.
[19] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.
[20] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.
[21] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4960–4964.
[22] M. Puheim and L. Madarász, "Normalization of inputs and outputs of neural network based robotic arm controller in role of inverse kinematic model," in *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2014, pp. 85–89.