ENHANCING HIGH-FREQUENCY CURRENCY EXCHANGE RATE FORECASTING: A COMPARATIVE STUDY OF DEEP LEARNING AND THE AUTOREGRESSIVE INTEGRATED MOVING AVERAGE MODEL

A Thesis By

NIMA NIKOPOUR
ORCID iD: 0009-0008-5355-8207

California State University, Fullerton
Fall, 2023

In partial fulfillment of the degree:
Master of Arts in Economics

Department:
Department of Economics

Approval Committee:
Gabriela Best, Department of Economics, Committee Chair
Pedro Amaral, Department of Economics
Thomas May, Department of Economics

DOI:
10.5281/zenodo.10421119

Keywords:
time-series, forecasting, currency, neural networks, deep learning, econometrics

Abstract:
This research presents a comprehensive evaluation of Deep Learning models for high-frequency univariate financial time series forecasting. I compare several architectures including Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN), an Attention-LSTM hybrid, and a novel Attention-CNN hybrid. These models are benchmarked against the conventional Autoregressive Integrated Moving Average (ARIMA) model. Using 15-minute interval data for the AUD/USD exchange rate, I meticulously assess these models on both one-step ahead and multistep ahead predictions while maintaining consistent hyperparameters. My findings demonstrate that all Deep Learning models outperform the ARIMA benchmark, with the novel Attention-CNN hybrid emerging as the best-performing model. This hybrid offers retail investors a more accurate alternative that requires minimal data preparation, effectively addressing the complexities of predicting currency exchange rates in dynamic financial markets.

TABLE OF CONTENTS

LIST OF TABLES iii

LIST OF FIGURES iv

ACKNOWLEDGMENTS v

1. INTRODUCTION AND LITERATURE REVIEW 1

    Introduction 1
    Literature Review 3

2. MODEL DESCRIPTIONS 8

    Autoregressive Integrated Moving Average 8
    Artificial Neural Networks and Optimization 9
    Long Short-Term Memory 15
    Convolutional Neural Networks 18
    Multi-head Attention 20

3. DATA AND METHODOLOGY 23

    Data 23
    Methodology 25
    Benchmark ARIMA 26

4. RESULTS AND CONCLUSION 28
    Results 28
    Conclusion 31

APPENDIX: ONE-STEP MODEL SUMMARIES AND LOSS PLOTS 32

REFERENCES 40

LIST OF TABLES

Table Page

1. Results 29

LIST OF FIGURES

Figure		Page
1.	Visualization of a simple ANN	10
2.	Three-dimensional gradient descent representation	12
3.	Two-dimensional gradient descent representation	13
4.	Visualization of stacked LSTM cells	16
5.	Convolution of a Gaussian curve and a rectangular pulse	19
6.	Plot of the AUD/USD currency pair from 1999 to 2022	24
7.	First difference of the input time series	27
8.	Plot of multistep ahead Attention-CNN and ARIMA forecasts	30

ACKNOWLEDGMENTS

I would like to express my gratitude to the chair of my committee, Dr. Gabriela Best. Her guidance and support have been instrumental in enhancing my understanding of time series forecasting. I also extend my thanks to Dr. Pedro Amaral for his thorough and constructive feedback on my written drafts, which significantly contributed to the refinement of my research. I also want to thank Dr. Thomas May for his assistance with the coding aspects of this research; his expertise in neural networks has greatly deepened my comprehension of the subject. Additionally, thank you to Dr. Radha Bhattacharya, the chair of the Economics Department, for her support and encouragement throughout this process.

Finally, I acknowledge the entire California State University Economics Department for creating an exceptional academic environment. The knowledge and skills I have acquired throughout the program have been pivotal to my success.

CHAPTER 1

INTRODUCTION

The Foreign Exchange market is the deepest and most liquid in the world. The Bank of International Settlements 2022 Triennial Central Bank Survey of Foreign Exchange reported that currency exchange markets were trading an average of 7.5 trillion per day as of April 2022 (BIS, 2022). This market is highly stratified, with much of the activity emanating from large firms and central banks. Until the late 1990s, they dominated the market and used currency exchange primarily for hedging purposes. This changed with the advent of online trading platforms, which have allowed retail investors to enter the market (Sobol & Szemelter, 2020). In 2016, trading conducted by non-institutional investors grew to constitute 5.5% of the currency exchange market (BIS, 2016). Unlike institutional traders, retail traders almost exclusively engage in speculative trading (Forman III, 2016). Despite its popularity, most traders lose money in day trading (Barber et al., 2014; Mahani & Bernhardt, 2007). The difficulty in making accurate predictions is due to the complex web of relationships currency prices have with the macroeconomy and psychological factors (Yao & Tan, 2000). Forecasting on long time horizons requires careful input selection, which can include technical indicators and news and sentiment analysis, which do not fluctuate at a high frequency within one day (Singh et al., 2022). Input selection is, therefore, difficult for intraday forecasts. Furthermore, retail investors will not typically have easy access to real-time high-frequency data, nor will they usually have the computational resources to train datasets with more variables. Thus, there is a need for accurate univariate models.

The Autoregressive Integrated Moving Average model (ARIMA), used in the Box-Jenkins methodology, is a conventional method of predicting univariate financial time series. The ARIMA predicts future values based on lags of the series itself and its error term. While it is a simple and popular model, it has two major drawbacks. One is that it requires sufficient preprocessing to make the data stationary; this is the "Integrated" part of the model. Additionally, it can only produce linear outputs, failing to capture the non-linear movements that currency exchange data often exhibits.

Deep Learning (DL), a subset of machine learning, has gained popularity for its robust predictive capabilities, utilizing artificial neural networks (ANN) inspired by the intricate architectures of the human brain. These networks contain layers of interconnected nodes, or neurons, designed to process and transform data. Neural networks typically consist of an input layer that receives the raw data, one or more hidden layers that progressively abstract features from the input data, and an output layer that yields the final prediction or decision based on the process. Each connection between neurons has an associated weight, and the neurons apply mathematical transformations to the input data as it flows through the network. This depth enables Deep Learning models to discover and represent intricate patterns and hierarchical features in data. Whereas machine learning relies on manually specified features, Deep Learning excels at automatically extracting relevant features as part of its learning process. In contrast to traditional time series forecasting models like ARIMA, Deep Learning can effectively handle non-stationary data and produce non-linear outputs.

Notable research findings suggest the superiority of Deep Learning models in certain cases, but there are instances where ARIMA models have outperformed DL models, especially in univariate settings (Siami-Namini & Siami-Namin, 2018; Kobiela et al., 2022; Shah & Shroff, 2021; Yamak et al., 2019; Zhang et al., 2022). This variability in results is due to several factors. Firstly, the characteristics of the time series data significantly influence model performance. DL models tend to excel when dealing with complex, non-linear, and high-dimensional data, whereas ARIMA models may perform well when the data exhibits clear temporal patterns and stationarity. Secondly, the choice and fine-tuning of hyperparameters, such as the number of layers, units, and learning rates, play a pivotal role in DL model performance since variations in such can lead to different outcomes. Lastly, different studies employ different forecasting regimes and model architectures, rendering direct comparisons challenging.

This research aims to evaluate multiple Deep Learning models against the conventional ARIMA model in the context of univariate currency exchange time series. Additionally, this research offers investors a more accurate alternative to the ARIMA model that also requires less data preparation. To

achieve this, I evaluate four DL models against an ARIMA benchmark on both one-step ahead forecasts and multistep ahead forecasts. I employ three prominent types of Deep Learning architectures: Long Short-Term Memory (LSTM), which employs gating mechanisms to retain pertinent information while discarding extraneous data selectively; Convolutional Neural Networks (CNN), which use convolution operations to map input sequences to output sequences; and multi-head attention, which assigns weighted importance to values within the sequence to capture temporal dependencies across varying time horizons. Additionally, I construct hybridizations of these models. I leverage the AUD/USD currency exchange rate data recorded at 15-minute intervals for this analysis. All Deep Learning models outperform the ARIMA benchmark, with a novel Attention-CNN hybrid performing best on both one-step ahead and multistep ahead forecasts.

Literature Review

Generally, Deep learning models have outperformed standard ARIMA models in time series exchange rate forecasting applications, with Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN) based models such as Long Short-Term Memory (LSTM) as the most successful and recommended architectures (Nemavhola et al., 2021). Deep Learning refers to stacking neural network layers to increase predictive capabilities. The hierarchical structure of multiple layers allows the model to learn the underlying patterns of the data through multiple layers of abstraction (LeCun et al., 2015). The strengths of DL in time series forecasting are that there is no need to define specific model parameters since they can learn based on the features of the data. Additionally, DL methods can model non-linear dependencies, while traditional models such as ARIMA can only produce linear outputs. Moreover, the estimation of Neural Networks does not hinge on stationarity assumptions since the number of trainable parameters allows neural networks to learn time-variant processes, unlike the ARIMA, which requires stationarity (Preeti et al., 2019). However, the disadvantage is that the inner mechanisms are less intuitive to understand than simpler models. This is why many refer to the inner mechanism of deep neural networks as a "black box." (Pouyanfar et al., 2018)

Hu et al. (2021) surveyed Deep Learning methods in currency and stock market forecasting. They found that 44% of papers were LSTM based, and 20% were CNN based. Additionally, in a review of currency forecasting methods, Islam et al. (2020) found that 48% of surveyed papers were Neural Network-based and 5% were ARIMA-based. Of the LSTM research, close price was the most frequent variable of interest, with many papers focusing on it exclusively. The most cited work in this area is by Siami-Namini and Siami Namin (2018), who compare an LSTM to an ARIMA on a dataset containing univariate financial time series, finding that the LSTM outperforms the ARIMA by 85% in rolling forecasts. This form of forecasting entails making one-step ahead predictions and adding the last predicted value to the next prediction's input series. This can be characterized as an iterative approach. However, this method does not fully leverage the capability of Deep Learning models, which can predict complete sequences at once (direct forecasting). Hamzaçebi et al. (2009) compare the performance of ANNs under both forecasting regimes, finding that direct forecasts of the entire series yield favorable results to iterative forecasts.

The success of LSTMs in time series forecasting lies in their ability to model long-term sequential dependencies. However, CNNs can perform similarly to LSTMs in time series problems, with the added benefit of significantly lower train time (Mehtab & Sen, 2020; Weyjtjens & De Weerdt, 2021). Dwivedi et al. (2021) evaluated CNN, LSTM, and ARIMA on univariate financial time series. They find that the CNN and LSTM perform similarly, although the LSTM slightly outperforms the CNN on the basis of mean squared error and that both DL models outperform the ARIMA, with the CNN exhibiting the quickest train time.

Attention mechanisms, first proposed by Bahdenau et al. (2014), introduced the idea of Bahdenau attention in the literature. It followed the work of Cho et al. (2014) and Sutskever et al. (2014), who used encoder-decoder RNN models in Neural Machine translation. Whereas RNNs take the whole input vector to predict the output, attention mechanisms learn to select which features of the input are most important in determining the output. This is most successfully demonstrated in natural

language processing (NLP), where the placements of words affect the context of the whole sentence. Vaswani et al. (2017) further developed this concept in their seminal paper "Attention is All You Need" which proposed the Transformer architecture, a widely used model in NLP based on Bahdanau style attention. Researchers have since applied attention-based architectures to other sequential problems, such as the financial time series. Lara-Benítez et al., 2021, constructed a Transformer that outperformed CNNs and LSTMs on over 50,000 univariate time series forecasts. They attribute the success of Transformers to their ability to capture long-term temporal dependencies but also find that parameterizing these models is more difficult since different parameter choices yield a higher variability of results than LSTM and CNN models. Transformers and other attention-based models are also shown to be insensitive to local dependencies in time series forecasting (Li et al., 2019).

Attention is widely used in various hybridized models that combine the long-term capabilities of attention with the specialties of other types of layers. Abbasimehr and Paki (2021) construct a univariate time series model where input data is passed through separate LSTM and attention layers and then concatenated to be further processed through dense layers. This performed the best against both a regular attention model and an ARIMA model on 16 different datasets. attention hybridizations have been constructed with CNN layers as well. Pourdaryaei et al. (2023) developed a model that fed the outputs of CNN layers into an attention layer, achieving better mean absolute percentage error results than other Deep Learning models on the same electricity consumption dataset. Aoud et al. (2022) constructed a similar hybridized CNN-Attention model, which outperforms regular CNN models on short-term univariate electricity consumption. Lastly, Li et al. (2019), as noted before, find that regular attention-based transformer models are insensitive to local dependencies and thus modify the canonical Transformer model with CNN layers to capture long and short-term relationships. This model yielded favorable results for both regular Transformer and ARIMA benchmarks on electricity consumption data. Neither of these three CNN and attention hybrid models employ a similar architecture as the Abbasimehr and Paki paper which fed inputs separately to an attention layer and LSTM layer before

further processing into the output. Instead, they all directly connected the CNN and attention-based layers. Furthermore, they consider only electricity consumption data, which has a less stochastic data generation process than financial time series.

While most literature has found superior performance from DL models against ARIMA, there are some varied results. Kobiela et al. (2022) found that an ARIMA outperforms LSTM through multiple time horizons on their dataset. The ARIMA model beat LSTM and a Gated Recurrent Unit model (a DL model similar in design to LSTMs) models in financial time series forecasting (Yamak et al., 2019). Similarly, Zhang et al. (2022) found that ARIMA models outperform LSTM models on monthly and weekly models, while the LSTM outperformed the ARIMA on rolling daily forecasts. Shah and Shroff (2021) compare an ARIMA to multiple DL architectures, including Transformer and LSTM in the context of univariate stock price data. Their explanation of the superior performance of the ARIMA is that financial data is inherently chaotic, making it difficult for Deep Learning models to find relevant regularities to make accurate predictions. Gers et al. (2001) evaluated LSTMs against traditional statistical models in simple univariate forecasting tasks of short input lengths. They argued through their findings that since LSTMs' successes are usually attributed to their ability to capture long-range dependencies, they are less useful in simple tasks. Thus, they recommend only using LSTMs in simple forecasts when traditional models fail.

However, these findings do not necessarily mean that LSTMs, or any other DL model, are incapable of outperforming statistical models in certain instances. ARIMA, for instance, only uses a handful of parameters, while DL models have thousands or even millions of parameters. With proper architecture design, the DL models should, at the very least, be able to learn the parameters that ARIMA would learn. According to the Universal Approximation Theorem, there exists a network that can approximate any function to another with any non-zero amount of error (Hornik et al., 1989; Cybenko, 1989). While this proves the possibility, it does not guarantee a formula that can reliably approximate any function onto another. For complex problems, this may entail a model with an infeasible number of

layers. Additionally, the ability to approximate one function to another well does not guarantee that those same parameters would allow the model to generalize well to new functions (Goodfellow et al., 2016, p. 194). Therefore, the question is not whether DL models can outperform ARIMA. Instead, research should focus on which model can most efficiently outperform the ARIMA and best generalize to new data.

CHAPTER 2

MODEL DESCRIPTIONS

Autoregressive Integrated Moving Average (ARIMA)

The ARIMA model is a linear model representing the relationship the current observation of a time series has with a specified order of past values and error terms. It consists of three components: the Autoregressive component, the Moving Average Component, and the Integrated Component (Hamilton, 1994, p. 43-58).

The Autoregressive (AR) term captures the relationship between the variable of interest and a number of lagged observations of itself:

$$Y_t = \phi_1 Y_{\{t-1\}} + \phi_2 Y_{\{t-2\}} + \cdots + \phi_p Y_{\{t-p\}} + \epsilon_t \quad (1)$$

Where Y_t is the value of the series at time t; $\phi_1, \phi_2, \ldots, \phi_p$ are the parameters of the AR part of the model; and ϵ_t is the error term. The coefficients of the AR component represent the influence or weight of the respective lagged values on the current value of Y_t. For instance, ϕ_1 represents the impact of the preceding value of the series on the current value, an ϕ_2 represents the effect the value from two periods ago has on the current value.

The Moving Average (MA) term captures the relationship between an observation and the lags of the error term. The MA(q) component is represented as:

$$Y_t = \epsilon_t + \theta_1 \epsilon_{\{t-1\}} + \theta_2 \epsilon_{\{t-2\}} \quad (2)$$

$\theta_1, \theta_2, .. \theta_q$ are the parameters of the MA part of the model. The error term, ϵ_t is assumed to be white noise. This means that the probability distribution of the term has a zero mean and constant variance. Additionally, it means that the observations are statistically independent from each other.

The Integrated component refers to the level of differencing that was applied to the data to achieve stationarity. A fundamental assumption of the ARIMA model is that the time series of interest is a stationary process. This means that the series has a mean of zero and constant variance. When data is non-stationary, it often exhibits trends, seasonality, or other systematic patterns that change over time,

making it challenging to identify meaningful relationships or make accurate forecasts (Stock & Watson, 2018, p. 519-520). If the series does not exhibit stationarity, differencing can be applied until stationarity is achieved. This is represented by the following:

$$y' = y_t - y_{t-1} \quad (3)$$

This process removes trend and seasonality in the series which stabilizes the mean (Wang et al., 2019). If one order of differencing is insufficient in achieving stationary, additional levels of differencing can be performed. Augmented Dickey-Fuller (ADF) Tests are a standard stationarity testing method. There exists a unit root in the series defined by Equation 4 if $\rho = 1$, this is what gives the series its trend. The ADF test tests the null hypothesis that the time series has a unit root (Hansen, 2022, p.566-570).

$$Y_t = \rho Y_{t-1} + \epsilon_t \quad (4)$$

Artificial Neural Networks (ANN) and Optimization

Artificial Neural Networks (ANN) constitute a direction of machine learning developed to mirror the organization of biological neural networks through simple yet large-scale computations. Given a vector of inputs, ANNs aim to find a set of weights that can ultimately produce a desired output when applied to the vector. While a simple process, doing such by hand is inefficient. Therefore, ANN developers aim to construct algorithms to find the optimal weights.

The simple ANN in Figure 1 has an input layer, hidden layer, and output layer, each consisting of two neurons. The input here is a vector of two values x_1 and x_2 which are transformed in the hidden layers to produce a vector of output values o_1, and o_2. Hidden layers are any layer in between the input and output layers. In Figure 1, I use a fully connected (also known as a 'dense') layer, which is named as such since each of its neurons is connected to all preceding and succeeding neurons. The weights are first initialized to random values following a Gaussian distribution, and the biases are all set to an equal value, typically 1. The weights represent the strengths of connections between neurons, while the biases provide neurons with an offset value that influences when they activate, ensuring that they can produce meaningful outputs even when inputs are near zero. The weights and biases (parameters) transform the

inputs into their respective hidden states. These parameters, θ, are what the model attempts to optimize during training to minimize loss, in other words, the difference between predicted and actual values. A typical loss function is Mean Squared Error (MSE).

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_i \ (\hat{y} - y)^2 \tag{5}$$

At each layer, the values are passed through an activation function, which introduces non-linearity to the model's outputs. The sigmoid function and the hyperbolic tangent function Are commonly used activation functions

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{6}$$

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{7}$$

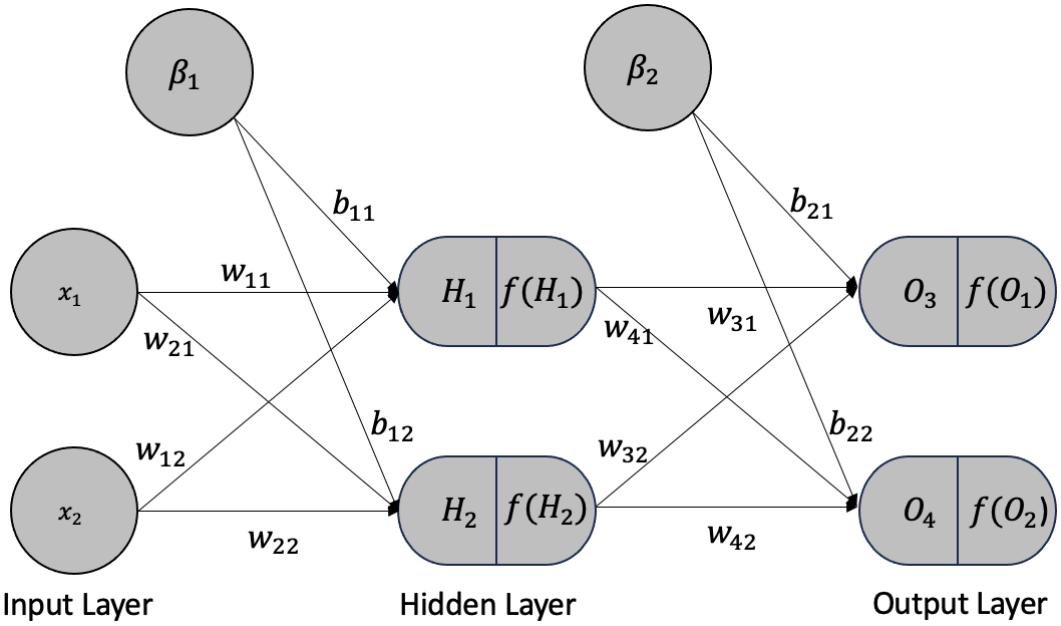


Figure 1. Visualization of a simple ANN

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \tag{8}$$

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{9}$$

$$h_1 = x_1 w_{11} + x_2 w_1 2 + \beta_1 b_{11} \tag{10}$$

$$h_2 = x_1 w_{21} + x_2 w_{22} + \beta_1 w12 \tag{11}$$

$$o_1 = f(h_1) w_{31} + f(h_2) w_{32} + \beta_1 b_{21} \tag{12}$$

$$o_2 = f(h_1) w_{41} + f(h_2) w_{42} + \beta_2 b_{22} \tag{13}$$

This transformation is called the forward pass and the outputs from the first forward pass will almost always differ greatly from the actual values. It is through backpropagation that the model can optimize the weights. During the backpropagation process, the network uses the Leibniz chain rule to compute partial derivatives through the model. In other words, backpropagation calculates the effect all preceding weights have on each output node using partial derivatives. For instance, the model calculates the effect that w_{11} has on total loss (considering only weights for now) by calculating:

$$\frac{\partial \mathcal{L}(W)}{\partial w_{11}} = \left(\frac{\partial \mathcal{L}(W)}{\partial f(o_1)} \; \frac{\partial f(o_1)}{\partial o_1} \; \frac{\partial o_1}{\partial f(h_1)} \; \frac{\partial f(h_1)}{\partial h_1}\right) + \left(\frac{\partial \mathcal{L}(W)}{\partial f(o_2)} \; \frac{\partial f(o_2)}{\partial o_2} \; \frac{\partial o_2}{\partial f(h_2)} \; \frac{\partial f(h_2)}{\partial h_2}\right) \tag{14}$$

Upon determining the network's weights' partial derivates (gradients), the model uses an optimization algorithm to update all weights at once. The most popular algorithm is gradient descent, which can, in this case, be summarized as:

$$\begin{bmatrix} w_{11}^{t+1} \\ w_{12}^{t+1} \\ \vdots \\ w_n^{t+1} \end{bmatrix} = \begin{bmatrix} w_{11}^{t} \\ w_{12}^{t} \\ \vdots \\ w_n^{t} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial \mathcal{L}(W)}{\partial w_{11}^t} \\ \frac{\partial \mathcal{L}(W)}{\partial w_{12}^t} \\ \vdots \\ \frac{\partial \mathcal{L}(W)}{\partial w_n^t} \end{bmatrix} \tag{15}$$

This can be generalized for all the model's parameters, $\theta,$ as:

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t) \tag{16}$$

The derived gradients tune the weights towards the direction that minimizes the objective function. In other words, gradient descent calculates the opposite direction of the gradient at each iteration. The weight parameters are updated to move in the negative direction of the gradient by a step of size η, also known as the learning rate (Drori, 2022, p. 39). Figure 2 and Figure 3 show a simple simulation of this process where the model learns to minimize the objective function by finding the gradient of its two weight parameters at each iteration through backpropagation. The gradient descent

algorithm, in this case, multiplies the gradient at each point for each weight with the learning rate. This is subtracted from the previous value of each weight to determine the updated parameters. To increase both efficiency and precision, the learning rate here exponentially decays so that the model initially takes larger steps to leave the maxima rapidly but will eventually take smaller steps as it approaches the minima, so it is less likely to overshoot the optimum weights.

Each iteration of this process is referred to as an epoch. In each epoch, the neural network undergoes a cycle of forward propagation (the forward pass), where input data is passed through the network to make predictions, followed by backpropagation, where gradients are computed for all weights. This is ultimately used for gradient descent. The number of epochs can either be a predetermined value or a stopping criterion can be set. For instance, the model can be configured to terminate training if loss has not improved after a certain number of epochs.

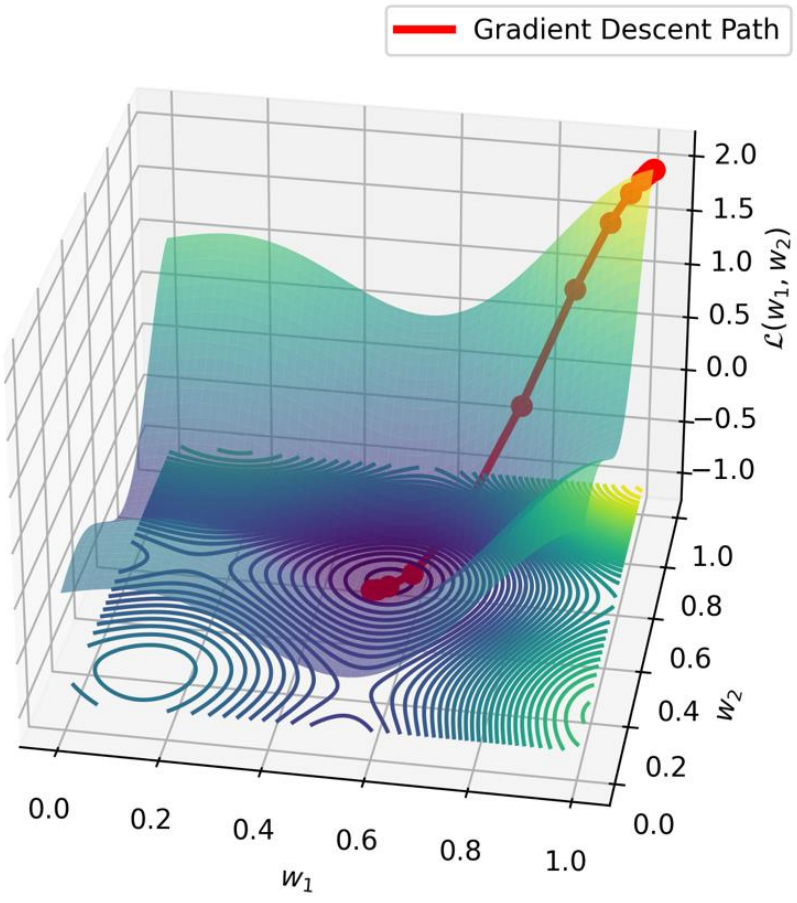


Figure 2. Three-dimensional gradient descent representation

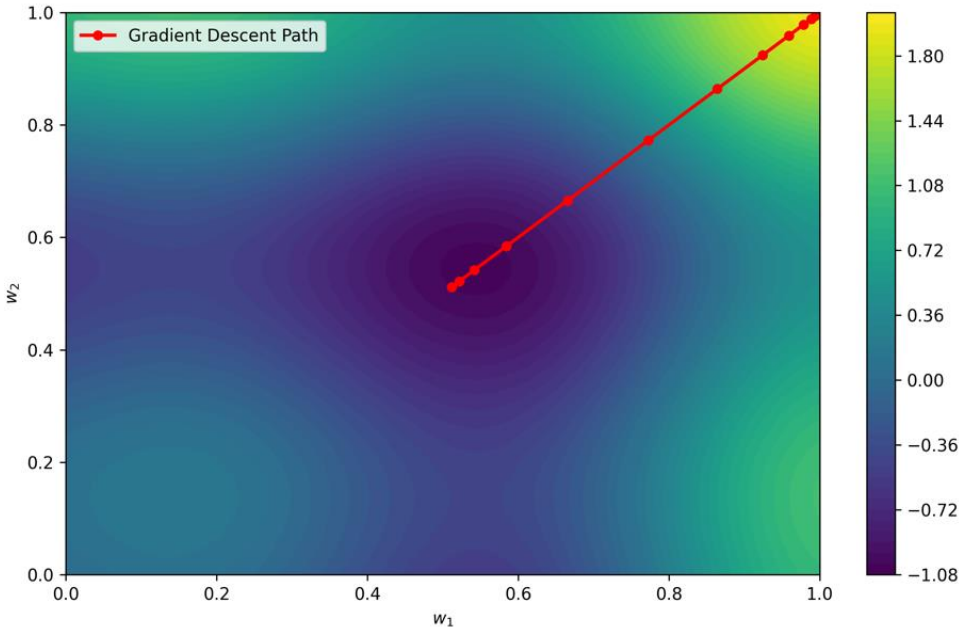


Figure 3. Two-dimensional gradient descent representation

While the preceding examples consider simple network designs with relatively small inputs, as models become deeper (more hidden layers) and input sizes increase, the model runs the risk of experiencing the vanishing gradient problem. This occurs because, with an increasing number of layers, the gradients need to traverse a longer path during backpropagation. As a result, the gradients can become increasingly small, making it difficult for the model to update each layer's weights and biases effectively.

Traditional gradient descent is computationally inefficient because it considers the whole dataset each time, so new methods have been introduced to induce faster performance. The most well-known of which is stochastic gradient descent (SGD):

$$\theta_{t+1} = \theta - \eta \nabla \mathcal{L}_i(\theta_t) \tag{17}$$

The SGD algorithm randomly shuffles the train set $x^1, \dots, x^n$ for $i = 1$, thereby computing the gradient at each iteration from a random sample rather than the entire data set. This is useful in Deep Learning applications where high dimensionality renders traditional gradient descent too time-consuming (Goodfellow et al., 2016, p. 292).

The learning rate, η, is the size of the step that parameters make at each iteration. Values too small can result in the model stagnating within local minima, and a value too large can cause the model to keep overshooting the convergence point. One solution is to use a learning rate scheduler, whereby the learning rate starts large and exponentially shrinks as it approaches the global minimum, ideally preventing overshooting as well as stagnation in local minima. In Figure 2 and Figure 3, I set the learning rate to 0.04 and specified that it should exponentially decay to make smaller steps as it progressed through training. However, the most popular modern method is the use of Adaptive Momentum Estimation (Adam), which is computationally efficient and well-suited for non-stationary and noisy data. Adam maintains an adaptive learning rate for each parameter in the model. By dynamically adjusting the learning rate based on the history of gradients, Adam strikes a balance between learning efficiently (making large updates when gradients are large) and maintaining stability (making smaller updates as the optimization progresses). This algorithm also does not require a stationary objective function (Kingma and Ba, 2014). The Adam algorithm maintains two key moving averages within the model. The first moving average is:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{18}$$

This represents the exponentially decaying average of past gradients where $g_t = \nabla \mathcal{L}(\theta)$ is the gradient w.r.t stochastic objective at timestep t. It effectively incorporates the momentum effect, aiding the algorithm in navigating through flat regions and escaping local minima. The second moving average is:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{19}$$

This represents the exponentially decaying average of the root of the sum of past squared gradients, which corresponds to the $L2$ norm. This moving average plays a role in adjusting learning rates for each parameter. By considering the $L2$ norm of gradients, Adam can differentiate between parameters associated with large gradients (requiring smaller learning rates for stability) and those with small gradients (allowing larger learning rates for faster convergence). Both these moving averages are

corrected for initialization bias to become $\hat{m}$ and $\hat{v}$ and used to calculate to update the parameter update by way of

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_\theta}{\sqrt{\theta} + \epsilon} \quad (20)$$

where ϵ is a small scalar preventing division by 0. Kingma and Ba also detailed AdaMax, a variant of Adam in the same 2014 paper. While Adam utilizes the L2 norm (Euclidean norm) of past squared gradients to adapt learning rates, AdaMax employs the infinity norm (maximum norm) to track the maximum absolute value of past gradients for each parameter,

$$u_t = max(\beta_2 \cdot u_{t-1}, |g_t|) \quad (21)$$

This maximum norm signifies the largest magnitude of a gradient component observed during training. Scaling each parameter's learning rate based on the maximum gradient magnitude,

$$\theta_{t+1} = \theta_t - \frac{\eta}{1-\beta_1} \cdot \frac{m_t}{u_t} \quad (22)$$

ensures that parameters with larger maximum gradient values receive lower learning rates to maintain stability, while parameters with smaller maximum gradients can have larger learning rates to facilitate rapid convergence. This utilization of the infinity norm makes AdaMax particularly robust in situations where gradients exhibit high variability compared to Adam.

Long Short-Term Memory (LSTM)

Recurrent Neural Networks, when faced with sequential data of longer lengths, are subject to the vanishing or exploding gradient problems. LSTMs, which are a type of Recurrent Neural Network first proposed by Hochreiter and Schmidhuber (1997), are the most successful means of addressing the vanishing and exploding gradients problem posed by traditional RNNs. The backbone of an LSTM's cell structure is the cell state. Where C_{t-1} represents the previous cell's state and C_t represents the updated cell state at the end of the sequence. Throughout the process, illustrated in Figure 4, the cell state is altered through regulated gates, whose activations are regulated by $sigmoid$ or $tanh$ functions. X_t is the vector of inputs at the time t. The vector formulas are:

$$C_t = f_t \odot C_{t-1} + i \odot \tilde{C} \quad (23)$$

$$\tilde{\mathrm{C}} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (24)$$

$$\mathrm{f_t} = \sigma(\mathrm{W_f} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_f}) \quad (25)$$

$$\mathrm{i_t} = \sigma(\mathrm{W_i} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_i}) \quad (26)$$

$$\mathrm{o_t} = \sigma(\mathrm{W_o} \cdot [\mathrm{h_{t-1}}, \mathrm{x_t}] + \mathrm{b_o}) \quad (27)$$

$$\mathrm{h_t} = \mathrm{o_t} \odot \tanh(\mathrm{C_t}) \quad (28)$$

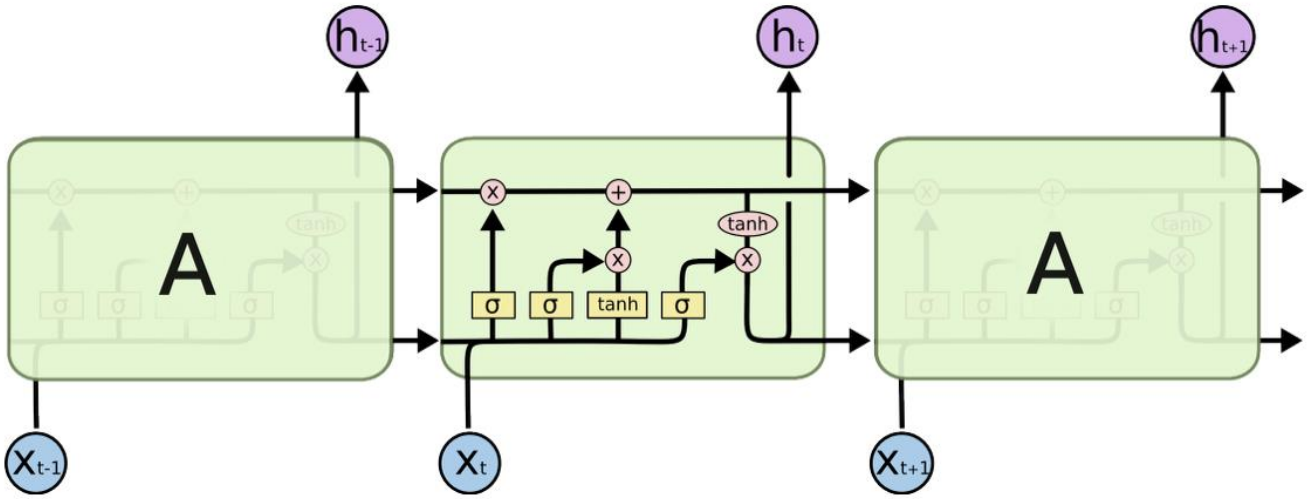


Figure 4. Visualization of stacked LSTM cells (Olah, 2015)

The previous cell state, $\mathrm{C_{t-1}}$, and the previous hidden state, $\mathrm{h_{t-1}}$, are set to initial values, usually zeros or small random values at first since there is no previous context to inform them at the start of training. The parameters, which are crucial in affecting the cell's behavior, are also randomly initialized. The first component of the LSTM cell is the Forget Gate. This gate regulates the flow of information from the previous cell state, $\mathrm{C_{t-1}}$, to the current cell state, $\mathrm{C_t}$. Mathematically, the forget gate computes its output, $\mathrm{f_t}$, by applying a sigmoid activation function to a weighted sum of the previous hidden state, $\mathrm{h_{t-1}}$, and the current input, $\mathrm{x_t}$. The forget gate output determines which information from the previous cell state should be retained and what should be discarded. This process can be expressed by Equation 25. $\mathrm{W_f}$ represents the forget gate's weight matrix, and $\mathrm{b_f}$ is its bias. The sigmoid activation function

squeezes the values into a range of 0 to 1, with large positive values approaching one and large negative values approaching zero.

The input gate and the candidate cell state, $\tilde{C}$, operate in tandem. The input gate controls what new information should be added to the cell state, while $\tilde{C}$ calculates a candidate value that could be incorporated. Similar to the forget gate, the input gate's output, i_t, is computed by applying a sigmoid activation function to a weighted sum of the previous h_{t-1} and the current input x_t, as shown in Equation 26. $\tilde{C}$ is determined by applying the hyperbolic tangent (tanh) function to another weighted sum of the previous hidden state and the current input, as shown in Equation 24. These two components work together to decide which values should be added to the cell state.

The next step involves updating the current cell state, C_t, by combining the previous cell state C_{t-1} with the information from the Forget Gate f_t and the Input Gate C_{t-1}. This dynamic update is expressed by Equation 23. Values from the forget gate are element-wise multiplied by corresponding values from the previous state. Values closer to one retain information from the previous cell state while values closer to zero diminish their corresponding information in the previous cell state. A similar interaction occurs between the input gate's outputs and candidate values which selects which new information should be added to the current cell state.

This updated cell state can be considered the long-term memory of the LSTM layer up until this point, it will be used to create the output of the given cell and will be sent to the next cell state to repeat the same process.

The final architectural elements of the LSTM cell are the output gate, o_t, and the hidden state, h_t. The output gate, akin to the forget and input gates, calculates its output by applying a sigmoid activation function to a weighted sum of the previous hidden state h_{t-1} and the current input x_t shown in Equation 27. The hidden state h_t, which is the LSTM cell's output at time step t, is computed by applying the hyperbolic tangent activation function to the updated cell state C_t and element-wise multiplying it by the output gate's output, as shown in Equation 28. The hidden state represents the information captured by

any given cell and is the information that will be sent to further layers in the model which can be used to influence the final output (Appleyard, 2016; Goodfellow et al., 2016, p. 404-407; Olah, 2015; Yan, 2016).

Convolution Neural Networks (CNN)

Convolutional Neural Networks (LeCun, 1989) have been most popular in their 2-dimensional applications for image data and their 1-dimensional applications in time series. CNNs are generally characterized as neural networks that use convolution, rather than regular matrix multiplication, in at least one layer (Goodfellow, 2016, p.326).

Convolution is a fundamental operation in signal processing and Deep Learning, widely used for analyzing and processing one-dimensional time series data. Mathematically, it involves sliding a small window, known as the kernel or filter, along the input time series. At each position of the window, element-wise multiplication is performed between the kernel and the corresponding elements of the input. The results of these multiplications are then summed up to produce a single output value. This process is repeated for each position of the window, generating a new time series called the output feature map.

$$((f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau)\, d\tau) \quad (29)$$

Mathematically, the process can be described by Equation 29. Here, $f(t)$ represents the input time series data, and $g(t)$ is the kernel or filter that the CNN uses to extract features from the input data. The convolution operation, $(f * g)(t)$, involves sliding the kernel over the input data and computing the integral of their element-wise product at each position t. This integral effectively summarizes how well the kernel matches a particular segment of the input data at each time step. τ is a dummy variable used for integration. A simple example is illustrated in Figure 5, where the rectangular pulse input series is element-wise multiplied by the values of a Gaussian curve (the kernel built up of a series of learnable weights), which creates a new series.

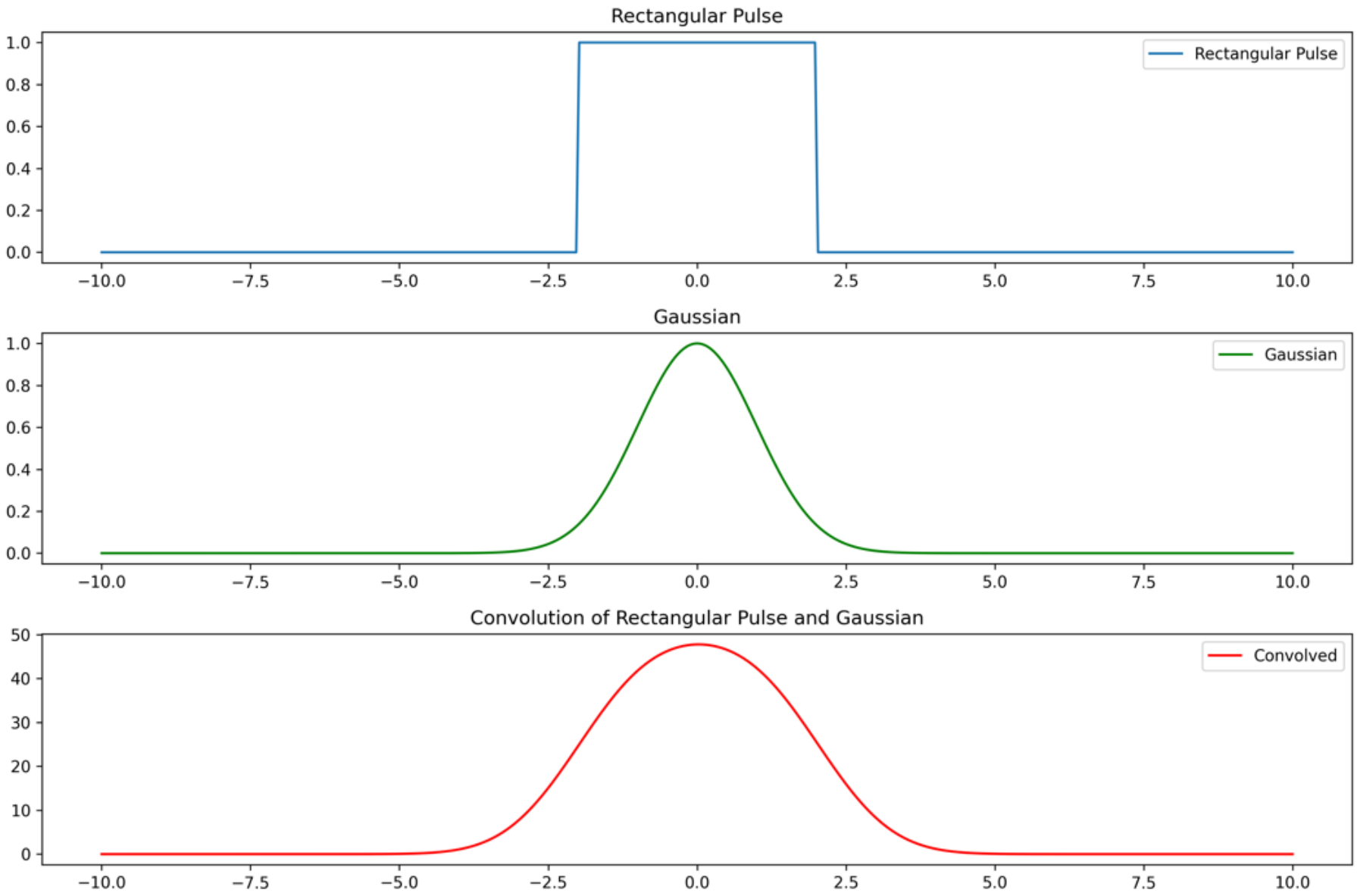


Figure 5. Convolution of a Gaussian curve and a rectangular pulse.

In complex time-series forecasting tasks, the ultimate goal of the convolution is not to directly predict what weights can approximate the input into the output. Rather, by performing convolution with various kernels of different shapes and sizes, a 1D CNN can learn to identify and recognize critical patterns and features that are pertinent to forecasting. These features might encompass trends, seasonality, or intricate dependencies within the data. Subsequently, the CNN leverages these extracted features to make forecasts about future values in the financial time series.

The architecture of a deep CNN begins with the input layer feeding into the convolution layer, in which the aforementioned convolution is applied. The resulting output is fed through an activation layer, typically a Rectified Linear Unit (ReLu). This replaces negative numbers with zeros. The next layer is the max pooling layer, which results in a reduction in the size of the previous output; this leaves only the most relevant information by keeping, for each given sized group of previous outputs, the maximum value (Zhou & Chellappa, 1988). There can be multiple cycles of convolution to max-pooling layers, usually decreasing in dimensionality until the model connects to the fully connected layers. CNNs have shown strong capabilities in understanding temporal dependencies and have been widely used in problems of time-series classification and forecasting (Goodfellow et al., 2016).

Multi-head Attention

The attention mechanism can be highly effective for extracting a range of features from a single data stream. This mechanism is constructed using the query, key, and value matrices. The query matrix is associated with the current state or the specific point in the series that the model aims to predict or analyze. The key and value pairs, derived from the historical data in the time series, play crucial roles in this process. The key aids the model in identifying which segments of past data are relevant to the current query. In contrast, the value provides the actual historical information that the model leverages for its predictions or analysis.

In a univariate time series context, both the query and the keys are derived from the same single variable but at different time steps. For instance, if the model is predicting today's exchange rate, the

query would be the representation of the current state, and the keys would be representations of the exchange rates at previous time steps.

Multi-head attention enhances this process by allowing the model to simultaneously focus on different aspects or patterns within the univariate time series. Each 'head' in a multi-head attention mechanism can potentially focus on different types of patterns. This ability to concurrently process and integrate different types of information from the same time series is particularly advantageous in univariate analysis. It enables the model to develop a more nuanced understanding of the data, capturing both immediate and long-term influences on the variable of interest. This comprehensive analysis is a reason why multi-head attention mechanisms have become a preferred choice over traditional models like LSTMs, especially in complex and rapidly changing domains like financial markets.

The input sequence of length n is transformed into three matrices: the key matrix K, the query matrix Q , and the value matrix V.

$$X = [x_1, x_2, \dots, \mathrm{x_n}] \tag{30}$$

$$K = X \cdot W^K \tag{31}$$

$$Q = X \cdot W^Q \tag{32}$$

$$V = X \cdot W^V \tag{33}$$

Where W^K, W^Q, and W^V are learnable weight matrices. Next, the attention mechanism calculates the attention scores by taking the dot product between the query and key vectors,

$$A = Q \cdot K^T \tag{34}$$

These scores reflect the strength of relationships between different positions in the sequence. To stabilize the attention competition, the attention scores are scaled by the square root of the key dimension

$$S = \frac{A}{\sqrt{d_k}} \tag{35}$$

The SoftMax function is then applied to the scaled attention scores to obtain the attention weights.

This function normalizes the scaled logits derived from Equation 35 by exponentiating each s_i to ensure that they are positive and then divide by them the sum of all exponentiated values across the sequence.

$$Softmax(S)_i = \frac{e^{s_i}}{\sum_{j=1}^{N} e^{s_i}} \tag{36}$$

This normalization step ensures that the resulting values are in the range [0,1] and sum up to 1, effectively creating a probability distribution over the elements in the sequence. These computed values from the SoftMax function represent the attention weights, in other words, they represent the importance of different positions in the sequence for each position. Finally, the attention weights are used to compute the weighted sum of value vectors. This aggregation captures the relationship between the positions in the sequence. The whole process can be captured by:

$$Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \tag{37}$$

Each set of (W_Q, W_K, W_V) matrices is an attention head. multi-head attention models can be constructed to cast attention to different parts of the sequence in parallel, increasing processing power and speed. This is achieved by, as before, calculating attention through each attention head indexed by i then concatenating the attention head outputs and linearly transforming them with W^O which is a learned weight matrix that is trained alongside the rest of the model's parameters. Since the weights of each attention head are randomly initialized, and thus begin at different starting points during training, they will ultimately learn different features through the optimization process.

$$MultiheadAttention(Q, K, V) + Concat_{i \in [\#heads]}\left(Attention\left(XW_i^Q, XW_i^K, XW_i^V\right)\right) W^O \tag{38}$$

This attention algorithm was introduced by Vaswani et al. (2017).

CHAPTER 3

DATA

I use close prices of the Australian Dollar to U.S. Dollar exchange pair from January 27, 1999, to November 29, 2019. This was sourced from a publicly available dataset on Kaggle.com (Jiménez, 2019). The data observes the close price in 15-minute intervals, totaling 515,200 total observations. Figure 6 shows a plot of this data.

Deep Learning training requires the data to be divided into input and target variables. To convert my time series arrays into a usable format, I utilized time-delay embedding to separate sequences of inputs and outputs. Given a time series array of $x_n = [x_1, x_2, x_3, \dots, x_n]$, I iteratively looped over the series to create two matrices, one being the input arrays and the other being the target arrays of the sequences that immediately follow their respective inputs. By setting the stride length to 1, it creates a series of overlapping windows that can be visualized as:

$$Inputs: \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_2 & x_3 & x_4 & x_5 \\ x_3 & x_4 & x_5 & x_6 \\ \vdots & \vdots & \vdots & \vdots \\ x_{n-7} & x_{n-6} & x_{n-5} & x_{n-4} \end{bmatrix} \tag{39}$$

$$Targets: \begin{bmatrix} x_5 & x_6 & x_7 & x_8 \\ x_6 & x_7 & x_8 & x_9 \\ x_7 & x_8 & x_9 & x_{10} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n-3} & x_{n-2} & x_{n-1} & x_n \end{bmatrix} \tag{40}$$

This example uses input and output lengths of size four as an example. In my models, I use inputs of length 96 and either length one or 96 for outputs (there are 96 15-minute intervals in a day).

Figure 6. Plot of the AUD/USD currency pair from 1999 to 2022

In the example above, the model at each point would attempt to predict the output sequence based on the corresponding row of inputs. Time delay embedding with overlapping windows has been shown to increase the precision of parameter estimates as opposed to sampling independent windows (von Oertzen and Boker, 2010). This is because it maximizes the amount of training examples available to the model. When making multistep ARIMA predictions, I use an iterative approach since ARIMA models can only produce one output at a time. This entails making the prediction for the first time step, appending it to the end of the input series, and removing the first element of the series to make the subsequent prediction. This continues for a predetermined number of steps until the entire sequence is predicted. Deep Learning can be trained to make direct multistep predictions where the model outputs the complete forecast at once rather than by the iterative approach.

Methodology

I use close price data at the end of each 15-minute interval. Close price best reflects the movement of the data in each period (Ariyo et al., 2014). This is the most common target variable, according to Islam et al. (2020).

First, I evaluate all the models on a one-step-ahead forecasting strategy, where an input row of values from one day predicts the next single value. Second, I employ a direct forecasting strategy in which the model learns to output a one-day length of values in one shot (with the exception of the ARIMA which, as noted before, still has to use the iterative approach).

The outputs are either of length 1 for one-step forecasts or of length 96 for the direct forecasting strategy. These corresponding matrices of inputs and their respective outputs are divided into training, validation, and testing sets, with 80% of the data being used for training, 20% used for testing, and 20% of the training set being used for validation during training. Conventional wisdom holds that in time series problems, the validation set should be the tail end of the training set to prevent leakage. This is called holdout validation. However, Bergmeir and Benitez (2012) tested this assumption on a variety of methods and determined that in stationary time series data, cross-validation offers more robust model

selection, specifically blocked cross-validation, which ensures no overlap between the train and validation folds. This finding was reiterated by Bergmeir et al. (2014), which found that blocked cross-validation provided more precision in error estimates than standard holdout validation. Conversely, Cerquiera et al. (2020) find that while blocked cross-validation works well for stationary time series, holdout validation works best for non-stationary series. Since my series is non-stationary, I decided to use holdout validation. I run my Python 3 code on a Google Colab workbook, utilizing their T4 GPUs to reduce train time. The Deep Learning models are all developed through the TensorFlow library, using the Keras sequential API. I chose Keras because of its modularity and ease of experimentation.

I use the AdaMax optimizer and batch sizes of 32 to train the data on 50 epochs. The models are set to stop early if validation loss does not improve after 50 epochs and to remember the best model parameters. The outputs of each model are evaluated against the actual values on the basis of MSE, RMSE, MAE, MAPE, and a custom directional accuracy metric that calculates the ratio of correct directional movements of each output.

Benchmark ARIMA

For a robust determination of ARIMA parameters, I construct code to search for the ideal AR(p), I(d), and MA(q) values. This search finds the lowest Akaike information criterion score by testing all (p, d, q) combinations within the range of 0-2 using the training dataset. The results find that an ARIMA(1,1,2) is the ideal set of parameters. This means that the next step of the time series is, on average, most accurately determined by considering the current value of the series and two lags of its error term. Additionally, the data needs to be differenced once to achieve stationarity. The differenced series is shown in Figure 7. The full equation is:

$$Y_t = \phi_1(Y_{t-1} - Y_{t-2}) - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2} + \epsilon_t \quad (41)$$

To then derive my baseline results, I construct code that loops through each row of the test set, fits the ARIMA(1,1,2) model, and outputs a forecast. This generates a new data frame, shown in Figure 7, of forecasts that I evaluate against each of the corresponding actual values.

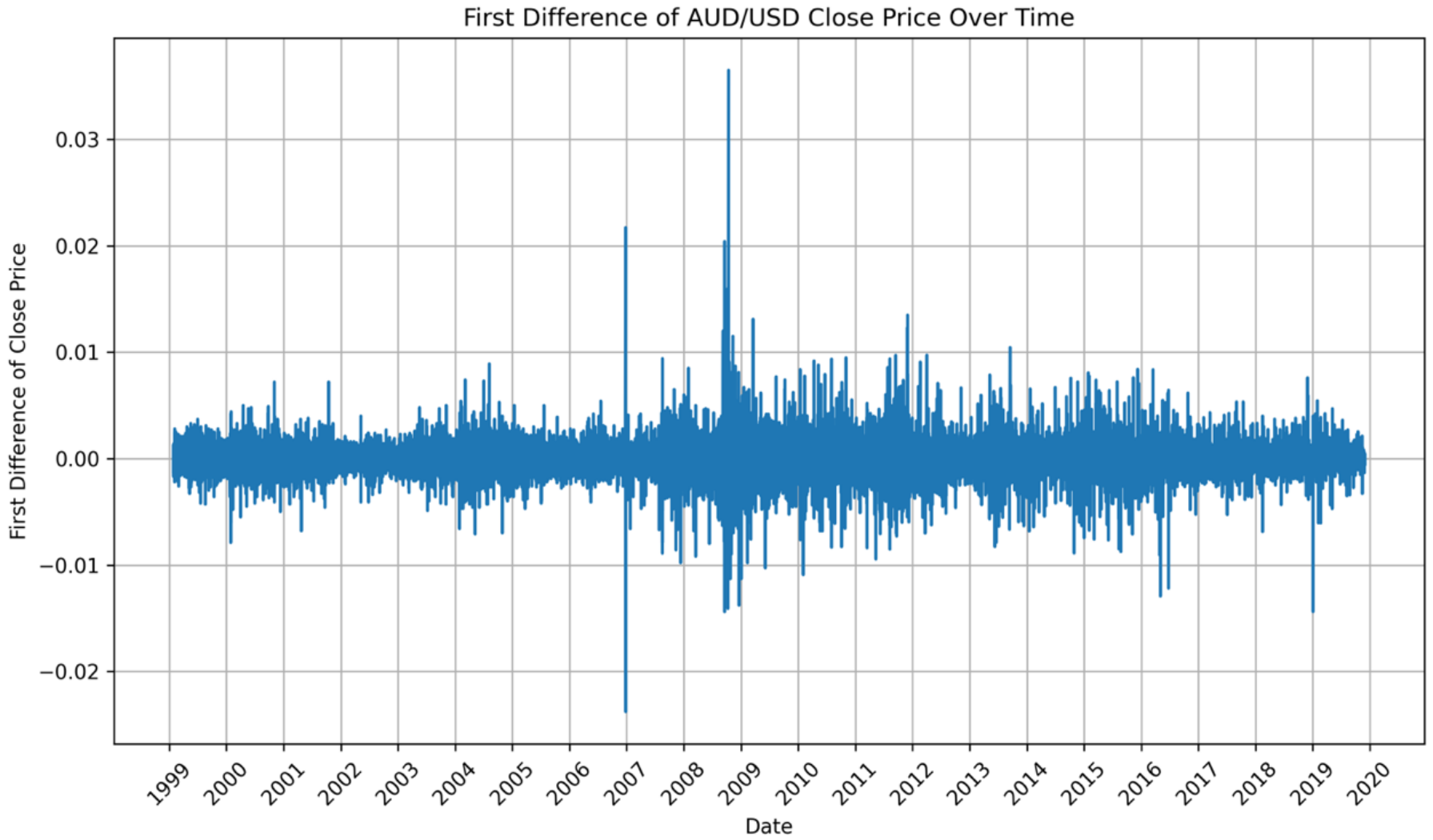


Figure 7. First difference of the input time series

CHAPTER 4

RESULTS AND CONCLUSIONS

Results

According to the results shown in Table 1, all Deep Learning models introduced in this study consistently outperform the benchmark ARIMA method, and they do so without necessitating any preliminary data preparation steps such as stationarity testing or differencing. Notably, the incorporation of attention mechanisms yielded a significant performance boost for CNN models, whereas its impact on LSTM models was comparatively limited. This divergence in performance can be attributed to the inherent complexity of both LSTM and attention models, which excel in handling sequential data. Amalgamating these two intricate models may introduce unnecessary complexity, potentially impeding their capacity to generalize effectively on new data. Conversely, CNNs, known for their simplicity, stand to gain more from the integration of a complex sequential component like an attention mechanism. Another potential factor contributing to this performance difference could be the redundancy in the capabilities of both LSTM and attention mechanisms, which are both proficient at capturing long-range dependencies. In contrast, CNN excels at handling short-range dependencies. Therefore, the hybridization of CNN and attention mechanisms proved more advantageous, especially considering the relatively small input lengths present in the dataset. The fusion of multi-head attention's long-term and contextual capabilities with CNN's adeptness in identifying patterns and local dependencies ultimately yielded the most accurate model in this study.

While this research has demonstrated the superiority of Deep Learning over the benchmark ARIMA, it is essential to recognize the inherent limitations of univariate forecasting approaches, particularly in the context of financial time series data. Figure 8 shows that while the Attention-CNN model captures non-linearities better than the ARIMA model in multistep forecasts, its forecasts rarely match the movement of actual values. In financial settings with highly stochastic data-generating processes, it becomes challenging to identify meaningful patterns solely from the univariate time series. Therefore, even though it has outperformed the benchmark, the model remains inherently limited in its ability to provide forecasts that are truly valuable or useful for investment purposes.

Table 1. Results

	Metrics					
Model	MSE	RMSE	MAPE	MAE	Directional Accuracy	Train Time
One Step Ahead						
ARIMA	$2.30e^{-7}$	0.00048	0.042	0.00031	47.93%	
LSTM	$2.13e^{-7}$	0.00046	0.042%	0.00031	47.78%	01:31:10
CNN	$2.18e^{-7}$	0.00047	0.042%	0.00031	47.77%	00:26:47
Attention-LSTM	$2.21e^{-7}$	0.00047	0.043%	0.00032	47.81%	03:43:14
Attention-CNN	$2.13e^{-7}$	0.00046	0.041%	0.00031	47.63%	00:46:19
Multistep Ahead						
ARIMA	$9.36e^{-6}$	0.00306	0.286%	0.00212	50.53%	
LSTM	$9.09e^{-6}$	0.00301	0.299%	0.00218	49.01%	01:31:26
CNN	$8.86e^{-6}$	0.00298	0.285%	0.00211	49.10%	00:28:19
Attention-LSTM	$9.12e^{-6}$	0.00302	0.296%	0.00218	49.11%	02:46:30
Attention-CNN	$8.83e^{-6}$	0.00297	0.284%	0.00210	49.10%	00:46:58

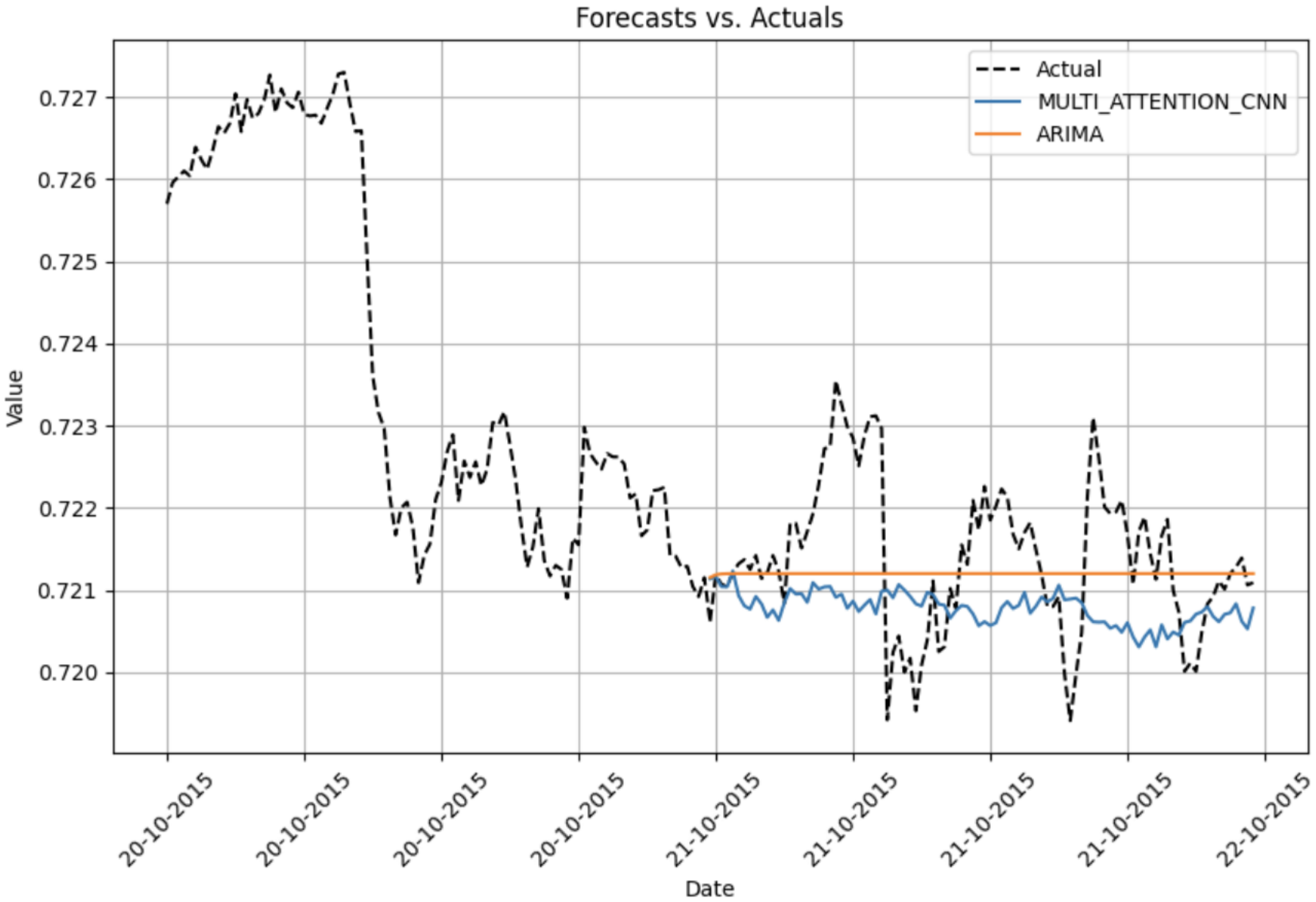


Figure 8. Plot of multistep ahead Attention-CNN and ARIMA forecasts

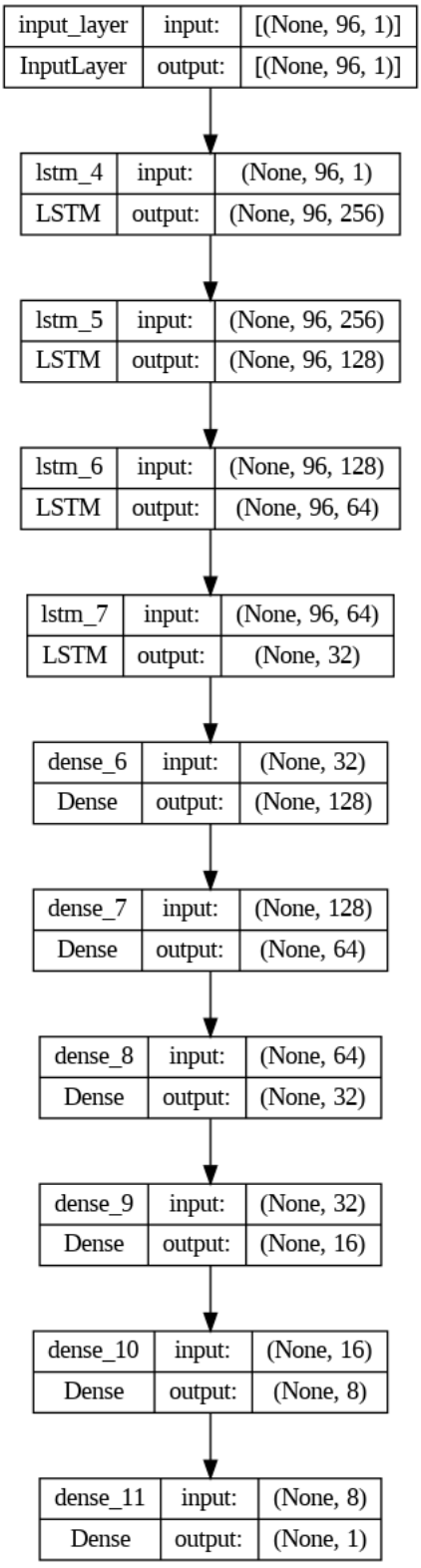
Conclusion

In summary, this research offers valuable insights into high-frequency univariate financial time series predictions, demonstrating the superior performance of LSTM, CNN, and attention-based hybrid models compared to the ARIMA benchmark, underscoring the potential of advanced Deep Learning techniques in financial forecasting. Furthermore, the study contributes significantly by conducting a thorough comparison of these models, considering both one-step ahead and multistep ahead forecasts while meticulously controlling for crucial hyperparameters, providing a nuanced understanding of their effectiveness in capturing complex financial data patterns. A notable aspect of this research is the introduction of a novel Attention-CNN hybrid, an innovative approach not previously explored in financial time series forecasting. This hybridization method, involving input feeding to both the CNN and attention layers with subsequent concatenation of their outputs before forwarding to dense layers, holds promise for improving forecasting accuracy in high-frequency financial time series, representing a promising avenue for further research and application in the field.

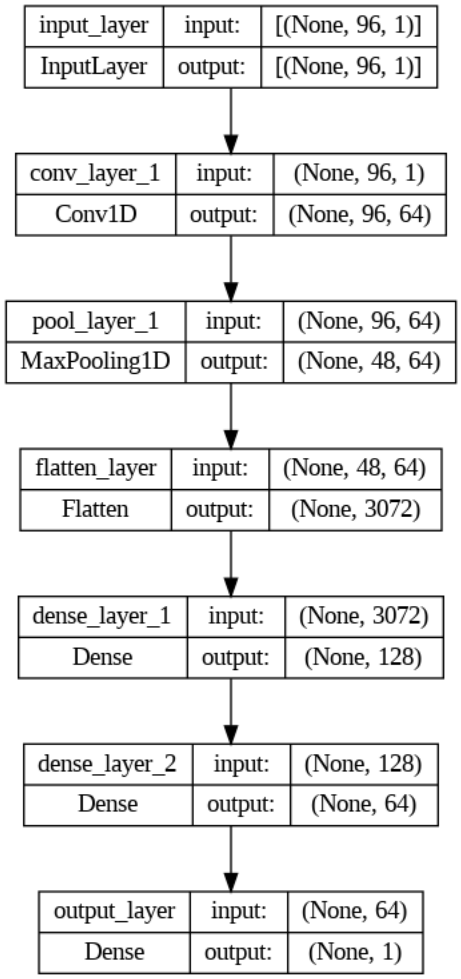
APPENDIX

ONE-STEP MODEL SUMMARIES AND LOSS PLOTS

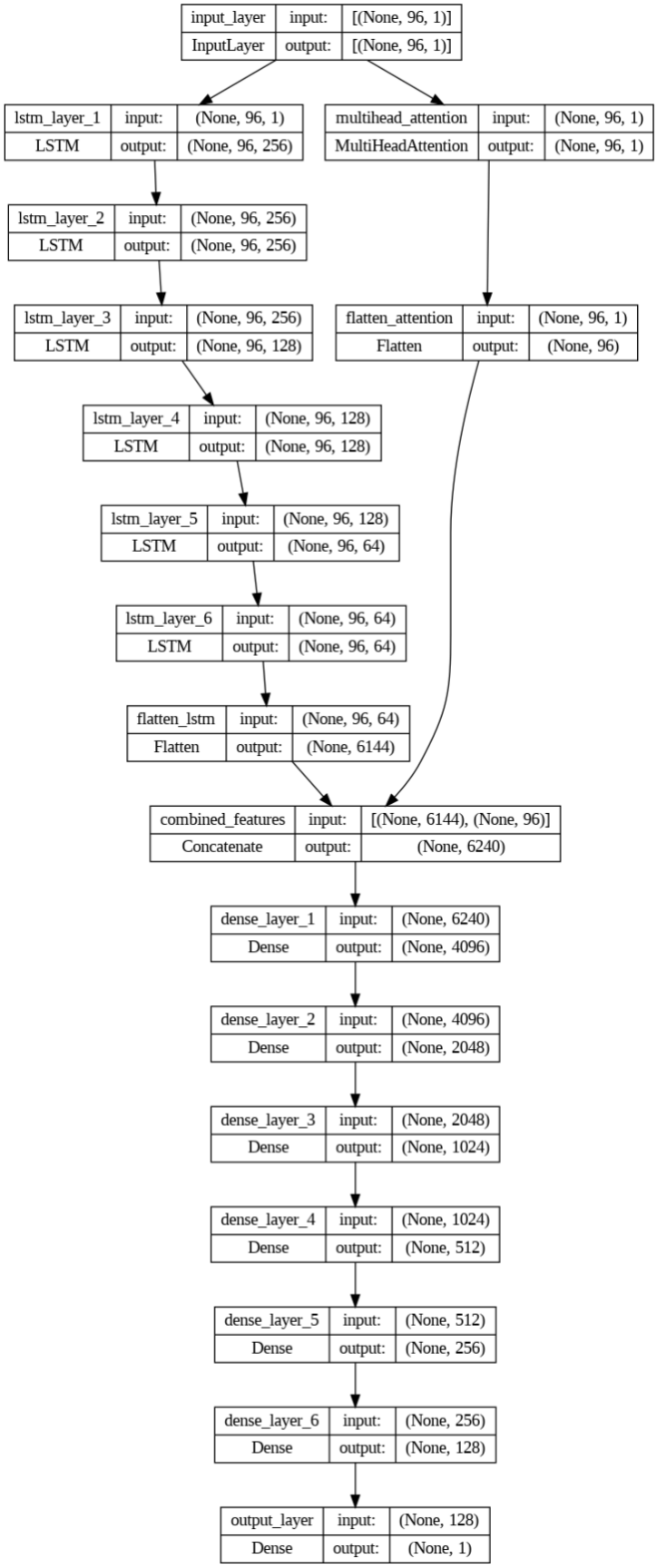
One-Step LSTM



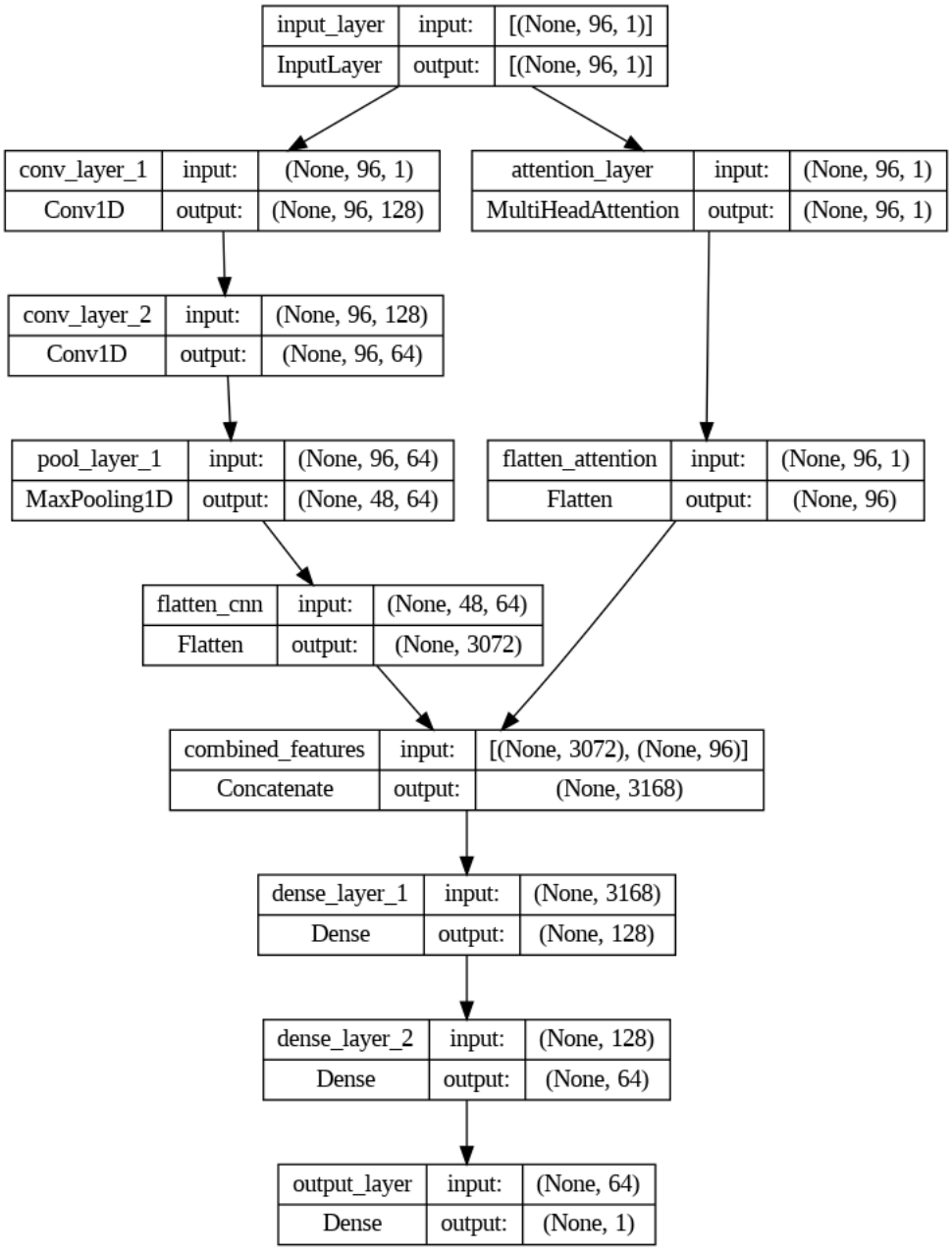
One-Step CNN


input_layer | input: | [(None, 96, 1)]
InputLayer | output: | [(None, 96, 1)]
conv_layer_1 | input: | (None, 96, 1)
Conv1D | output: | (None, 96, 64)
pool_layer_1 | input: | (None, 96, 64)
MaxPooling1D | output: | (None, 48, 64)
flatten_layer | input: | (None, 48, 64)
Flatten | output: | (None, 3072)
dense_layer_1 | input: | (None, 3072)
Dense | output: | (None, 128)
dense_layer_2 | input: | (None, 128)
Dense | output: | (None, 64)
output_layer | input: | (None, 64)
Dense | output: | (None, 1)

One-Step Attention-LSTM

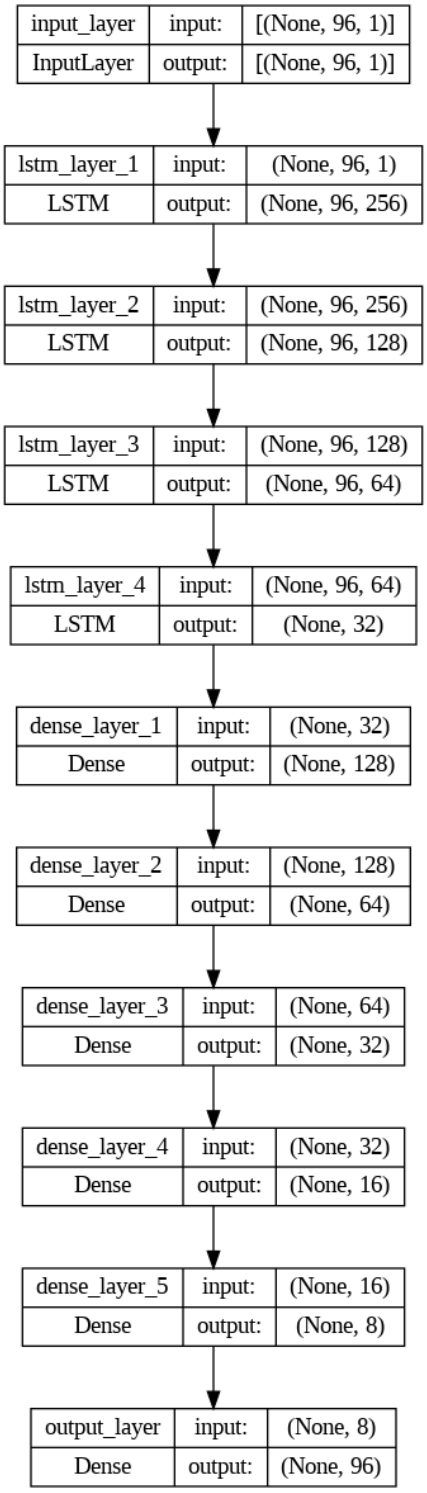


One-Step Attention-CNN

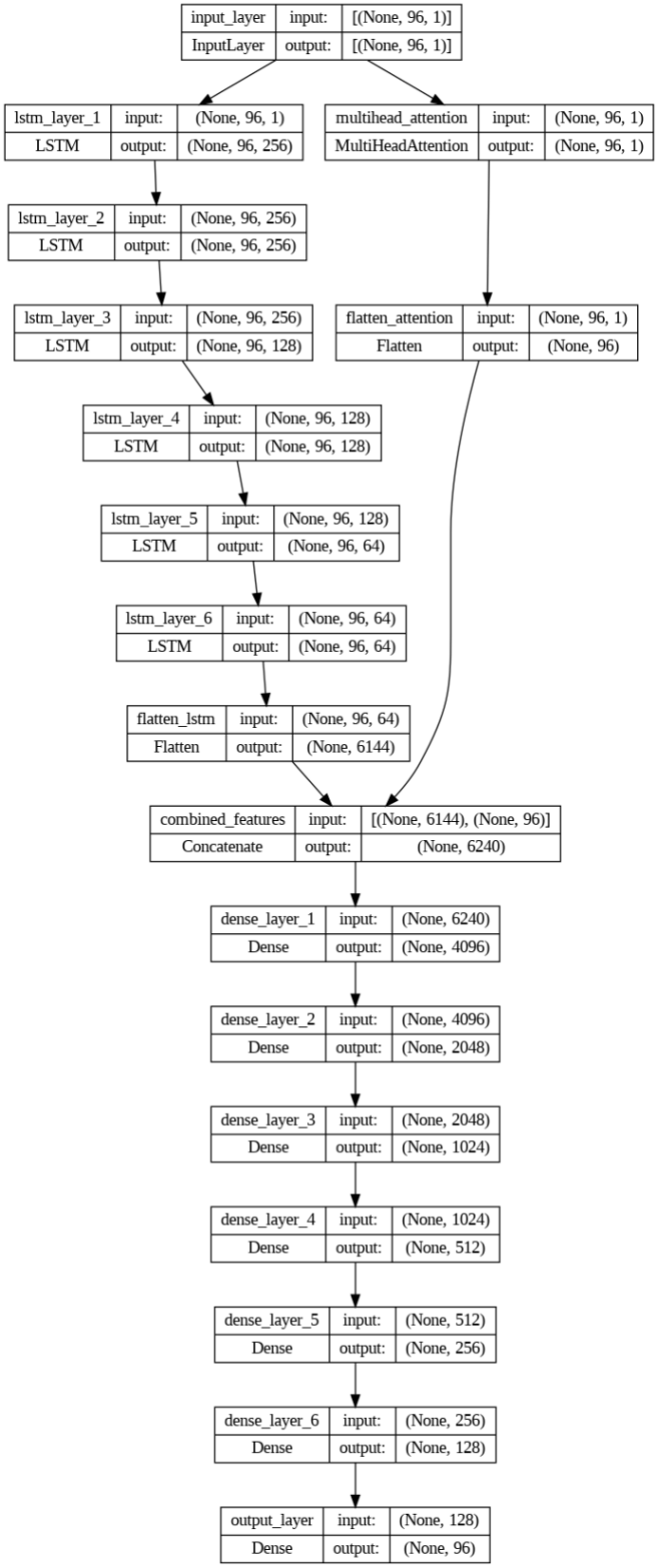


Multistep Model Summaries and Loss Plots

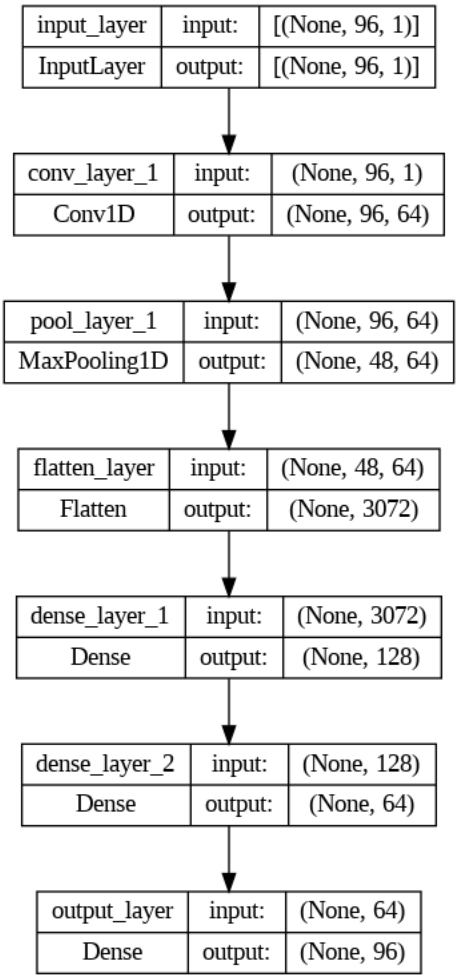
Multistep LSTM



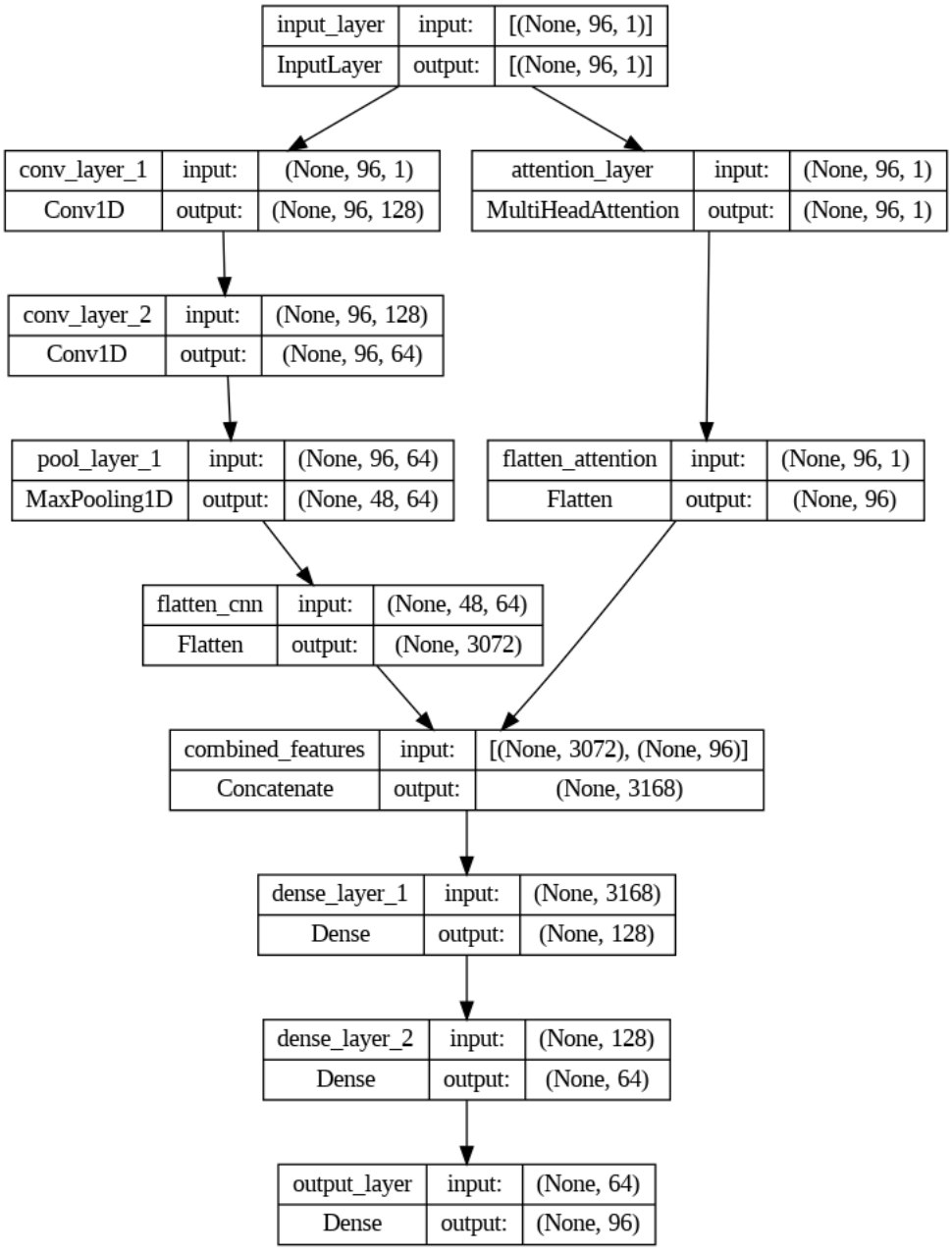
Multistep Attention-LSTM



Multistep CNN



Multistep Attention-CNN



REFERENCES

Abbasimehr, H., & Paki, R. (2022). Improving time series forecasting using LSTM and attention models. *Journal of Ambient Intelligence and Humanized Computing*, *13*, 1-19. https://doi.org/10.1007/s12652-020-02761-x

Aouad, M., Hajj, H., Shaban, K., Jabr, R. A., & El-Hajj, W. (2022). A CNN-Sequence-to-Sequence network with attention for residential short-term load forecasting. *Electric Power Systems Research*, *211*, 108152. https://doi.org/10.1016/j.epsr.2022.108152

Appleyard, J. (2016, April 6). *Optimizing recurrent neural networks in Cudnn 5*. NVIDIA Technical Blog. https://developer.nvidia.com/blog/optimizing-recurrent-neural-networks-cudnn-5/

Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014, March). Stock price prediction using the ARIMA model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation* (pp. 106-112). IEEE. https://doi.org/10.1109/UKSim.2014.67

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate*. arXiv. https://doi.org/10.48550/arXiv.1409.0473

BIS. (2016). *Triennial central bank survey of foreign exchange turnover in 2016*. Bank of International Settlements. https://www.bis.org/publ/rpfx16fx.pdf

BIS. (2022). T*riennial central bank survey of foreign exchange and over-the-counter (OTC) derivatives markets in 2022*. Bank of International Settlements.https://www.bis.org/statistics/rpfx22.html

Barber, B. M., Lee, Y.-T., Liu, Y.-J., & Odean, T. (2014). The cross-section of speculator skill: Evidence from day trading. *Journal of Financial Markets*, *18*, 1–24. https://doi.org/10.1016/j.finmar.2013.05.006

Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, *191*, 192-213. https://doi.org/10.1016/j.ins.2011.12.028

Bergmeir, C., Costantini, M., & Benítez, J. M. (2014). On the usefulness of cross-validation for directional forecast evaluation. *Computational Statistics & Data Analysis*, *76*, 132-143. https://doi.org/10.1016/j.csda.2014.02.001

Cerqueira, V., Torgo, L., & Mozetič, I. (2020). Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, *109*, 1997-2028. https://doi.org/10.1007/s10994-020-05910-7

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation*. arXiv. https://doi.org/10.48550/arXiv.1406.1078

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303-314. https://doi.org/10.1007/BF02551274

Drori, I. (2022). *The science of deep learning*. Cambridge University Pres

Dwivedi, S. A., Attry, A., Parekh, D., & Singla, K. (2021, February). Analysis and forecasting of time-series data using S-ARIMA, CNN and LSTM. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (pp. 131-136). IEEE. https://doi.org/10.1109/icccis51004.2021.9397134.

Forman, J. (2016). The retail spot foreign exchange market structure and participants. *SSRN Electronic Journal.* https://dx.doi.org/10.2139/ssrn.2753823

Gers, F. A., Eck, D., & Schmidhuber, J. (2001, August). Applying LSTM to time series predictable through time-window approaches. In *Artificial Neural Networks - ICANN 2001* (pp. 669-676). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-44668-0_93

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press

Hamilton, J. D. (1994). *Time series analysis.* Princeton University Press

Hamzaçebi, C., Akay, D., & Kutay, F. (2009). Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting. *Expert Systems with Applications, 36*(2), 3839–3844. https://doi.org/10.1016/j.eswa.2008.02.042

Hansen, B. (2022). *Econometrics*. Princeton University Press

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*(5), 359-366. https://doi.org/10.1016/0893-6080(89)90020-8

Hu, Z., Zhao, Y., & Khushi, M. (2021). A survey of forex and stock price prediction using deep learning. *Applied System Innovation*, *4*(1), 9. https://doi.org/10.3390/asi4010009

Islam, M. S., Hossain, E., Rahman, A., Hossain, M. S., & Andersson, K. (2020). A review on recent advancements in forex currency prediction. *Algorithms, 13*(8), 186. https://doi.org/10.3390/a13080186

Jiménez, D. F. (2019). *Forex currencies* [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/820558

Kim, S., & Kang, M. (2019). *Financial series prediction using attention LSTM*. arXiv. https://doi.org/10.48550/arXiv.1902.10877

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv. https://doi.org/10.48550/arXiv.1412.6980

Kobiela, D., Krefta, D., Król, W., & Weichbroth, P. (2022). Arima vs LSTM on NASDAQ stock exchange data. *Procedia Computer Science, 207,* 3836–3845. https://doi.org/10.1016/j.procs.2022.09.445

Lara-Benítez, P., Gallego-Ledesma, L., Carranza-García, M., & Luna-Romera, J. M. (2021). Evaluation of the transformer architecture for univariate time series forecasting. In E. Alba, G. Luque, F. Chicano, C. Cotta, D. Camacho, M. Ojeda-Aciego, S. Montes, A. Troncoso, J. Riquelme, & R. Gil-Merino (Eds.), *Advances in Artificial Intelligence: 19th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2020/2021* (pp. 106-115). Springer International Publishing. https://doi.org/10.1007/978-3-030-85713-4_11

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten ZIP code recognition. *Neural Computation, 1*(4), 541-551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y. X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, *32*. https://doi.org/10.48550/arXiv.1907.00235

Mahani, R. S., & Bernhardt, D. (2004). Financial speculators' underperformance: Learning, self-selection, and provision of liquidity. *The Journal of Finance*, 62, 1313-1340. https://doi.org/10.1111/j.1540-6261.2007.01237.x

Mehtab, S., & Sen, J. (2020). Stock price prediction using CNN and LSTM-based deep learning models. In *2020 International Conference on Decision Aid Sciences and Application (DASA)* (pp. 447-453). https://doi.org/10.1109/dasa51403.2020.9317207

Nemavhola, A., Chibaya, C., & Ochara, N. M. (2021). Application of the LSTM - deep neural networks - in forecasting foreign currency exchange rates. *2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. https://doi.org/10.1109/imitec52926.2021.9714685

Olah, C. (2015, August 27). *Understanding LSTM networks*. Colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pourdaryaei, A., Mohammadi, M., Mubarak, H., Abdellatif, A., Karimi, M., Gryazina, E., & Terzija, V. (2024). A new framework for electricity price forecasting via multi-head self-attention and CNN-based techniques in the competitive electricity market. *Expert Systems with Applications, 235,* 121207. https://doi.org/10.1016/j.eswa.2023.121207

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M. L., Chen, S. C., Iyengar, S. S. (2018). A survey on Deep Learning. *ACM Computing Surveys*, *51*(5), 1–36. https://doi.org/10.1145/3234150

Preeti, Bala, R., & Singh, R. P. (2019). Financial and non-stationary time series forecasting using LSTM recurrent neural network for short and long horizon. *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. https://doi.org/10.1109/icccnt45670.2019.8944624

Shah, V., & Shroff, G. (2021). *Forecasting Market Prices using DL with Data Augmentation and Meta-learning: ARIMA still wins!*. arXiv. https://doi.org/10.48550/arXiv.2110.10233

Siami-Namini, S., Tavakoli, N., & Siami Namin, A. (2018). A comparison of Arima and LSTM in forecasting time series. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. https://doi.org/10.1109/icmla.2018.00227

Singh, T., Kalra, R., Mishra, S., Satakshi, & Kumar, M. (2022). An efficient real-time stock prediction exploiting incremental learning and deep learning. *Evolving Systems*, *14*(6), 919–937. https://doi.org/10.1007/s12530-022-09481-x

Sobol, I., & Szmelter, M. (2020). Retail investors in the foreign exchange market. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, *64*(6), 168-181. https://doi.org/10.15611/pn.2020.6.13

Stock, J. H., & Watson, M. W. (2003). *Introduction to econometrics*. Pearson

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *27*. https://doi.org/10.48550/arXiv.1409.3215

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*. https://doi.org/10.48550/arXiv.1706.03762

Von Oertzen, T., & Boker, S. M. (2010). Time delay embedding increases estimation precision of models of intraindividual variability. *Psychometrika*, *75*, 158-175. https://doi.org/10.1007/s11336-009-9137-9

Wang, S., Li, C., & Lim, A. (2019). *Why are the ARIMA and SARIMA not sufficient*. arXiv. https://doi.org/10.48550/arXiv.1904.07632

Weytjens, H., & De Weerdt, J. (2020). Process outcome prediction: CNN vs. LSTM (with attention). In: A. Del Río Ortega, H. Leopold, & F. M. Santoro (Eds.), *Business process management workshops* (pp. 321–333). https://doi.org/10.1007/978-3-030-66498-5_24

Yamak, P. T., Yujian, L., & Gadosey, P. K. (2019). A comparison between Arima, LSTM, and GRU for time series forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence* (pp. 49-55). https://doi.org/10.1145/3377713.3377722

Yan, S. (2017, November 15). *Understanding LSTM and its diagrams*. Medium. https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

Yao, J., & Tan, C. L. (2000). A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, *34*(1–4), 79–98 https://doi.org/10.1016/s0925-2312(00)00300-3

Zhang, R., Song, H., Chen, Q., Wang, Y., Wang, S., & Li, Y. (2022). Comparison of Arima and LSTM for prediction of hemorrhagic fever at different time scales in China. *PLoS One*, *17*(1). https://doi.org/10.1371/journal.pone.0262009

Zhou, & Chellappa. (1988, July). Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks* (pp. 71-78). IEEE. https://doi.org/10.1109/ICNN.1988.23914