FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Utilization of Transformer architecture for predicting financial time series in the forex market
Student:	Radek Přibyl
Supervisor:	Ing. Stanislav Kuznetsov, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2025/2026

Instructions

The Transformer is a deep neural network archetype widely utilized in applications such as natural language processing, machine translation, and time series forecasting. This thesis aims to implement and evaluate various Transformer architectures designed for price forecasting in the foreign exchange market. The focus will be on detecting and predicting dynamic price fluctuations to enhance decision-making in trading.

1. Explore and summarize advancements in the field of transformers and in time series analysis. [1] [2] [3] [4]
2. Collect publicly available financial data from the forex market. Preprocess the data into a format suitable for machine learning.
3. Implement the original transformer model as described in [5] and choose and implement other models suitable for time series prediction, as discussed, e.g., in [1]. Select appropriate hyperparameters and train the models.
4. Verify the reliability of the models on historical data using relevant metrics. Compare, visualize, and discuss the results.
5. Publish the code — including implementations of Transformer architectures, data loading and preprocessing, and the training procedures — on a publicly available GitHub repository.

[1] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey", 2023 https://arxiv.org/abs/2202.07125

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 7 March 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

[2] A. Zeng, M. Chen, L. Zhang, Q. Xu, "Are transformers effective for time series forecasting?", 2022, https://arxiv.org/abs/2205.13504
[3] T. Muhammad, A. B. Aftab, Md. Ahsan, Md. M. Ahsan, M. M. Muhu, M. Ibrahim, S. I. Khan, M. S. Alam, "Transformer-based deep learning model for stock price prediction: A case study on Bangladesh stock market.", 2022, https://arxiv.org/abs/2208.08300
[4] Y. Li, S. Lv, X. Liu, Q. Zhang, "Incorporating Transformers and Attention Networks for Stock Movement Prediction", 2022, https://www.researchgate.net/publication/358897376_Incorporating_Transformers_and_Attention_Networks_for_Stock_Movement_Prediction
[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, I. Polosukhin, "Attention is all you need.", 2017, https://arxiv.org/abs/1706.03762

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 7 March 2024 in Prague.

Bachelor's thesis

UTILIZATION OF TRANSFORMER ARCHITECTURE FOR PREDICTING FINANCIAL TIME SERIES IN THE FOREX MARKET

Radek Přibyl

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Stanislav Kuznetsov, Ph.D.
October 24, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Přibyl Radek. *Utilization of Transformer architecture for predicting financial time series in the forex market.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

I would like to thank my supervisor Ing. Stanislav Kuznetsov Ph.D. for his guidance and invaluable expertise throughout the writing of this thesis. I am also deeply grateful to my family and friends for their unwavering encouragement and support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on October 24, 2024

Abstract

This thesis presents the implementation of two deep learning models–Transformer and Autoformer—for time series forecasting, specifically predicting foreign exchange (Forex) prices. Both models were developed from scratch using Python and TensorFlow. Part of the process was to utilize time series properties and signal processing techniques to improve the accuracy of the predictions. Data from the Forex market served as the input datasets, and technical indicators, were added to provide more information for the models to learn from. In addition, data smoothing techniques from the signal processing field were applied to reduce noise in the time series.

The models were trained to predict prices, and their outputs were transformed into the binary classification domain to determine how the price changed during the observed time period. The performance of the models was compared to each other and to two basic baseline models.

All code with the techniques used in this work is publicly available on GitHub at https://github.com/pribylr/bp/.

Keywords machine learning, neural networks, transformers, Python, Tensorflow, time series, signal processing

Abstrakt

Tato bakalářská práce se zabývá implementací dvou modelů hlubokého učení – Transformer a Autoformer – pro predikci časových řad, konkrétně předpovídání cen na trhu s devizami (Forex). Oba modely byly vyvinuty ručně v programovacím jazyku Python a knihovně TensorFlow. Součástí procesu bylo využití vlastností časových řad a zpracování signálu. Data z Forexového sloužila jako vstupní datasety, ke kterým byly přidány technické indikátory, aby modely získaly více informací, z nichž by se mohly učit. Navíc byly použity techniky vyhlazování z oblasti zpracování signálu, aby se v časových řadách snížil šum.

Modely byly trénovány na predikci cen, přičemž jejich výstupy byly transformovány do domény binární klasifikace, aby se určilo, jak se cena změnila v průběhu sledovaného časového období. Výkonnost modelů byla porovnána mezi sebou a také s dvěma základními modely.

Veškerý kód s technikami použitými v této práci je veřejně dostupný na GitHubu na adrese https://github.com/pribylr/bp/.

Klíčová slova strojové učení, neuronové sítě, transformery, Python, Tensorflow, časové řady, zpracování signálu

List of abbreviations

NLP	Natural Language Processing
LSTM	Long Short-Term Memory
NN	Neural Network
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
Adam	Adaptive Moment Estimation
Forex	Foreign Exchange Market
RSI	Relative Strength Index
DFT	Discrete Fourier Transform
IDFT	Inverse Discrete Fourier Transform
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform

Chapter 1
Introduction

Since its release in 2017, the paper "*Attention is All You Need*" has led to significant advancements in deep learning, especially in the field of natural language processing (NLP). The Transformer model, introduced in this paper, is a robust architecture designed to process sequential data. Sequential data can be a sentence in the case of NLP or time-ordered numerical values, like prices in financial markets, which is the focus of this thesis.

The original transformer has been widely used, and its performance has been documented mainly on NLP tasks. Since structural similarities exist between natural language and time series data, many transformer-based models have been proposed. These models include modifications and upgrades to the original architecture to make the models more suitable for time series forecasting.

Other models that are used for time series forecasting, such as AutoRegressive Integrated Moving Average (ARIMA) or long short-term memory (LSTM) networks, have difficulties capturing long-range dependencies and patterns in the data. The proposed mechanism in the transformer, the self-attention, is designed to capture these patterns more effectively, which is why it has gained attention in time series forecasting.

Time series forecasting presents challenges to which the models need to adapt, such as (non)-stationarity, seasonal or periodic patterns, or phenomenons occurring at irregular intervals.

This thesis presents the implementation of two models, the original transformer and *Autoformer*, a model explicitly designed for time series forecasting. Both models are built from scratch in Python with the use of the TensorFlow machine learning framework. The thesis also includes an explanation of the code, explaining how the models are structured, trained, and evaluated on time series data.

1.1 Goals

The theoretical chapter defines the research's fundamental concepts: machine learning, neural networks, transformers, time series, signal processing, and the foreign exchange market. It also includes research on related works in similar fields – price prediction in financial markets.

The experimental chapter describes the machine learning models used and how they are implemented. It explains where input data comes from and how it looks. It shows how the data is analyzed, understood, and processed and how it is used to make predictions. In the end, the results are discussed and compared to two simple baseline models, which are also described in this part of the thesis.

Chapter 2
Background and Theory

This chapter contains theoretical information and concepts on which the practical part of the work is based.

Section 2.1 provides the information necessary to comprehend the principles upon which this thesis is based. It includes information such as what neural networks are, how transformers work, and what time series are.

Section 2.2 describes the data on which the forecasts are made. The data used in this thesis comes from the foreign exchange market, which is a market where currencies are traded in pairs.

Autoformer, one of the models implemented in this thesis, is introduced in section 2.3.

Section 2.4 describes three different works with similar goals to this thesis's goal, which is price prediction. It contains information about the data used, the methods used for predictions, and the achieved results.

2.1 Technical background

Section 2.1.1 delivers a brief introduction to machine learning. The machine learning part contains information about binary classification, which is used to represent the final results in the experimental part of the thesis.

Sections 2.1.2 and 2.1.3 explain what neural networks and transformers are and how they work.

Section 2.1.4 summarizes theoretical information regarding time series, their characteristics, different types, and other features.

2.1.1 Machine learning

This section introduces and explains a few foundational machine-learning concepts used in this thesis, providing an overview of key algorithms and techniques and their relevance to solving specific problems within the research.

When dealing with a machine learning problem, a model is created to handle making predictions. To make accurate predictions, this model needs to be ”trained.” Training a model means supplying it with existing and harvested data, which helps the model learn patterns within the data. The goal of the training process is to help the model make predictions as accurate as possible on new and unseen data based on what it has learned from the existing data.

To ensure that a model makes accurate predictions on data that it has not seen yet, it is convenient not to train the model on all available data but rather to split it into two subsets—train and test sets.

Predictions can be made in different domains. For example, we can predict temperature, house price, or age. These examples have in common that the predicted variables, also called target variables, are continuous, and in this case, it is called a regression problem. On the other hand, classification problems map input data into a set of possible outcomes, such as recognizing traffic signs in image recognition or classifying diseases. In these cases, the target variable is one of the possible outcomes.

2.1.1.1 Binary classification

When a machine learning problem can result in only two possible outcomes, it is called binary classification. It is used in various applications, such as spam detection, medical diagnosis, sentiment analysis, and predicting whether the price of a financial asset will increase or decrease.

In binary classification, the two possible outcomes are denoted as positive (class 1) and negative (class 0). Several metrics exist to evaluate the quality of predictions. For example, commonly used metrics are:[1]

- *Accuracy*,
- *Precision*,
- *Recall*,
- *F1 score*,

which are the metrics used to represent results of the experimental part of this thesis. All four mentioned metric values range from zero to one, and the higher the value, the better the model's performance. After the model is trained and predictions are made, the following four values can be computed to calculate the mentioned metrics:[1]

1. true positive (TP) - number of correctly predicted data as positive,
2. false positive (FP) - number of incorrectly predicted data as positive,
3. true negative (TN) - number of correctly predicted data as negative,
4. false negative (FN) - number of incorrectly predicted data as negative.

Accuracy represents the ratio of all correctly predicted values to the total number of values. It is a pretty straightforward metric and convenient to use when the available samples are distributed somewhat evenly, meaning there are roughly 50 % samples of class 0 and roughly 50 % samples of class 1 in the case of binary classification. In situations when the data is not evenly distributed, a simple model constantly predicting 0 or 1 will achieve satisfactory results, even though it has not learned anything about the nature of the data.[1] Following is how to compute accuracy:[2]

$$\text{Accuracy} := \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.1}$$

Precision and *recall* are more convenient metrics to consider when *accuracy* is unsuitable. *Precision* is useful when an incorrectly predicted positive leads to significant consequences, and *recall* is useful when the cost of false negatives is high.[1]. The following are equations for *precision* and *recall*:[2]

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.2}$$

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.3}$$

Precision and *recall* can be both reach high values if data is easily separable and the right model is trained. However, sometimes, there is a trade-off between them, for example in spam detection classification problems. It is important not to identify a legitimate message as spam. In order to avoid doing this, the model will make few positive predictions, minimizing the number of false positive predictions, which increases *precision*. On the other hand, it will make more negative predictions, causing decrease in *recall*.[3]

The last metric considered in this thesis, in terms of binary classification, is the F1 score. It is a harmonic mean of precision and recall:[3]

$$\text{F1 score} := 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

A confusion matrix is a way to visualize the number of true positives, true negatives, false positives, and false negatives the model produced. Depending on how the confusion matrix is created, it often uses different shades of a given color to better illustrate the number of values. Below in figure 2.1 is an example of a confusion matrix[4], which is done in the *scikit-learn*[5] library.

In the provided example, a binary classifier was trained on images of cars. Its task was to predict whether a given image of a car was a sedan. After producing the predictions, numbers of TP, FP, TN, and FN emerged and were put into a confusion matrix.

The correctly predicted data, TN and TP, are in the upper left and bottom right corners of the matrix, respectively. False positives are in the upper right, and false negatives are in the bottom left.

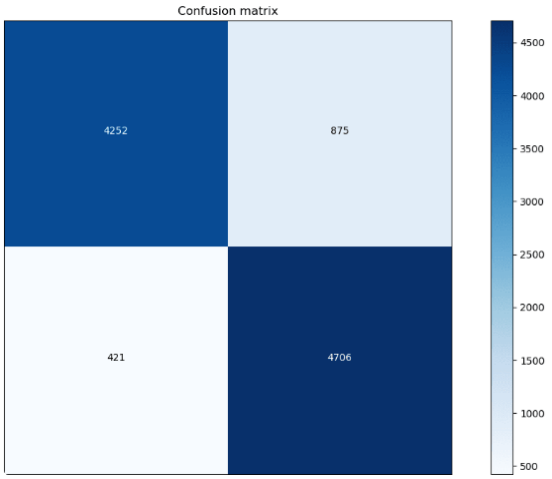


Figure 2.1 Example of confusion matrix

2.1.2 Neural networks

The main focus of this thesis is transformers, which are types of neural networks. Neural networks (NN) are a widely used concept in the machine learning field. The design of a neural network is inspired by the structure and functionality of the human brain, where neurons receive multiple signals from other neurons and produce one output. In (artificial) NN, neurons are organized into layers, with each layer producing output from input received from the previous layer.[6] Figure 2.2 shows NN with its neurons and layers:

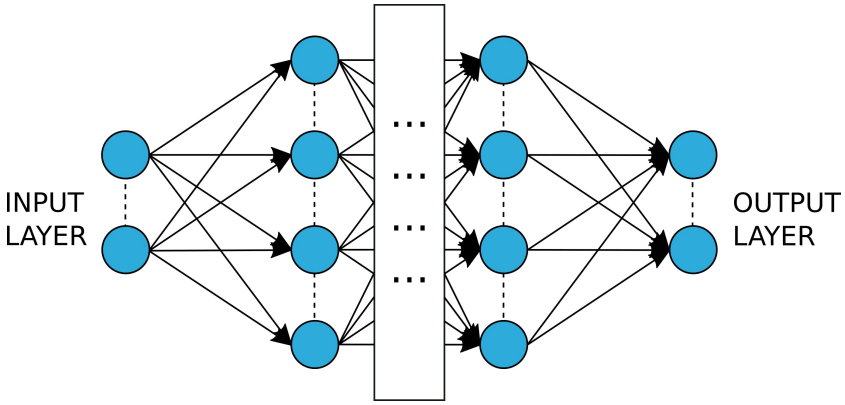


Figure 2.2 Neural network structure

Input to the neural network can be any data the network is designed to process. It may be images, sound, weather conditions, and more. Each input is transformed into a numerical representation before being processed by the network. The neural network's output is a prediction. It is either a number representing the actual domain the network is trained on in the case of a regression problem or a number representing a class in a classification problem.[7]

2.1.2.1 Perceptron

A perceptron is the first type of neural network introduced in 1958 by Frank Rosenblatt. It consists of one neuron in a single layer, and it was designed to solve binary classification tasks. It receives multiple inputs and each input is weighted with its own weight.[8]

The perceptron can correctly classify input data if its target variables are linearly separable. On the other hand, if the data is not linearly separable, the perceptron never finds a solution that would achieve 100 % accuracy. One of the problems the perceptron cannot solve is the *XOR* function which is shown in the table 2.1. X_1 and X_2 are input variables, and Z is the target variable:[6]

X_1	X_2	Z
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1 Xor function

The equation 2.5 below is an explanation of the mechanism by which a perceptron processes input data and how it calculates output.

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.5}$$

where $x_1, \ldots, x_n$ are input variables, $w_1, \ldots, w_n$ are weights and b is a bias, ϕ is an activation function, typically step function.[8] In equation 2.6, there is how the step function computes output for input x:[8]

$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.6}$$

Non-linearly separable data can be better handled by multi-layer perceptrons, where multiple perceptrons, or neurons, are put into various layers. How these layers work is explained in the following section.

2.1.2.2 Dense layer

The dense layer, also known as the fully connected layer, connects every neuron in a layer to every neuron in the next layer. The value of an *i-th* neuron in layer l is computed as:[9]

$$a_i^{(l)} := f\left(\sum_{j=1}^{n} W_{ij} a_j^{(l-1)} + b_i^{(l)}\right), \tag{2.7}$$

where $a_i^{(l)} \in \mathbb{R}$, m is a number of neurons in layer l, n is a number of neurons in layer previous to l, $\mathbf{W} \in \mathbb{R}^{m,n}$ is a matrix of weights, $\mathbf{a}^{(l-1)} = (a_0^{(l-1)}, \ldots, a_{n-1}^{(l-1)})^T \in \mathbb{R}^n$ is a vector of neurons in a layer previous to l, $\mathbf{b}^{(l)} = (b_0, \ldots, b_{m-1})^T \in \mathbb{R}^m$ is a vector of biases for layer l, and f is an activation function.

2.1.2.3 Activation function

The dense layer applies a linear transformation of its input, enabling the NN to learn linear patterns in the input data. However, the data can also contain non-linearities that the linear transformation may omit. An activation function is applied to help the NN learn non-linear patterns within the data. The activation function applies a non-linear transformation to the input value. One viable function is the Rectified Linear Unit (*ReLU*):[10][11]

$$\mathrm{ReLU}(x) := \max(0, x), \tag{2.8}$$

where x represents the sum of weighted outputs of the previous layer and biases, in other words, the output of one neuron, as detailed in equation 2.7.

There are several reasons for not using non-linear activation functions. When the task is simple or patterns in the input data are prevalently of a linear nature, it can be better to omit using non-linear functions. Another reason is that when the goal is to predict a continuous variable, the non-linear function in the outer layer of an NN should not be used.[11] In such cases, the identity function is a viable option:

$$\mathrm{Id}(x) := x. \tag{2.9}$$

The output of an activation function serves as an input to every neuron in the next layer. *ReLU* and identity are the activation functions used in models implemented in this thesis.

2.1.2.4 Loss function

The models in this thesis are trained on past prices of forex currency pairs, and the prices are handled as a continuous variable. This means the models are designed to solve a regression problem.

In a regression problem, loss functions measure the quality of a model's predictions. One such function is the mean squared error (*MSE*):

$$\mathrm{MSE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.10}$$

where $\mathbf{Y} = (y_0, \ldots, y_{n-1})$ is a vector of real values and $\hat{\mathbf{Y}} = (\hat{y}_0, \ldots, \hat{y}_{n-1})$ is a vector of predicted values.

MSE is the loss function the models use in this thesis. There are many different loss functions besides MSE, which include:

- *mean absolute error*,
- *root mean squared error*,
- *mean squared logarithmic error*,
- *mean absolute percentage error*,

and many others. Each is calculated differently and reveals the model's error from a slightly different perspective. So, it matters what loss function is used for what problem. Various loss functions yield distinct evaluations regarding the discrepancy between predicted values and actual data in certain instances.[12]

The loss function is regularly calculated during the model's training, and the goal is to decrease its value.

2.1.2.5 Optimization method

Training a model aims to make its predictions as close to reality as possible. One way to do that is to minimize a loss function during the training process. Optimization methods are algorithms designed to adjust the NN's weights by determining how much and in which direction each weight should be updated to minimize the loss function.

Multiple optimization algorithms exist, such as:

- *stochastic gradient descent*,
- *adaptive gradient algorithm*,
- *root mean squared propagation*,
- *adaptive moment estimation* (*Adam*)

Adam is one of the most common methods, and it is the method used in this thesis.[13] The following is an overview of how *Adam* works.

Compute gradient g of a loss function J for timestep t and model parameters w: $g_t = \nabla_w J(w)$. Information on gradients from past timesteps is retained like this:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \tag{2.11}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2, \tag{2.12}$$

where $\beta_1, \beta_2 \in [0, 1)$ are decay factors, m_t, v_t are first and second moments. At timestep 0 they are initialized as $m_0 = \theta, v_0 = \theta$. Because of this initialization they need to be corrected like this:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.13}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.14}$$

Finally weights of a model are updated:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t, \tag{2.15}$$

where η is a learning rate and ϵ is a small value preventing division by zero.[7, 14, 13]

2.1.3 Transformer

A transformer is a type of neural network commonly used to solve *sequence-to-sequence* (*seq2seq*) tasks. A *seq2seq* task is one where input and output are sequential data. It is common, for example, in translating sentences from one language to another or converting speech to text. Models designed to handle *seq2seq* tasks can generally process sequences of variable length. Several models are commonly used for these tasks, such as recurrent neural networks (RNNs), LSTMs, gated recurrent units (GRUs), or models utilizing the encoder-decoder framework. [15, 16]

The transformer's architecture is displayed in figure 2.3. It utilizes self-attention mechanism, which is a type of attention framework. The transformer model is built from the following components, which will be described in more detail further: [17]

- *Input embedding*,
- *Positional encoding*,
- *Multi-head attention*,
- *Masked multi-head attention*,
- *Add&Norm*,
- *Feed forward*.

This thesis is about time series forecasting, but to clarify the concept, some of the essential parts of the transformer are presented using examples from natural language processing. To keep the terminology general and consistent, *token* and *sequence* are used instead of *word* and *sentence*.

2.1.3.1 Encoder

The transformer is based on the encoder-decoder architecture. The encoder processes input data, finds its essential features, explores data patterns, and converts them into a meaningful representation. It can optionally consist of several layers, with each layer's output being an input to the next layer. The last layer's output is then passed to the decoder. Each encoder's layer consists of the following sublayers: *Multi-head attention*, *Add&Norm*, *Feed forward*, followed by another *Add&Norm*. [17, 18]

2.1.3.2 Decoder

The decoder functions in an autoregressive manner. It generates the output sequence in steps, each step predicting one token based on the tokens it has generated up to this point. It receives two different inputs: data received from the encoder and its own generated part of the output sequence. The decoder can also contain multiple layers. It consists of the following sublayers: *Masked multi-head attention*, *Multi-head attention*, *Feed forward*, all of them followed by *Add&Norm*. [15, 17]

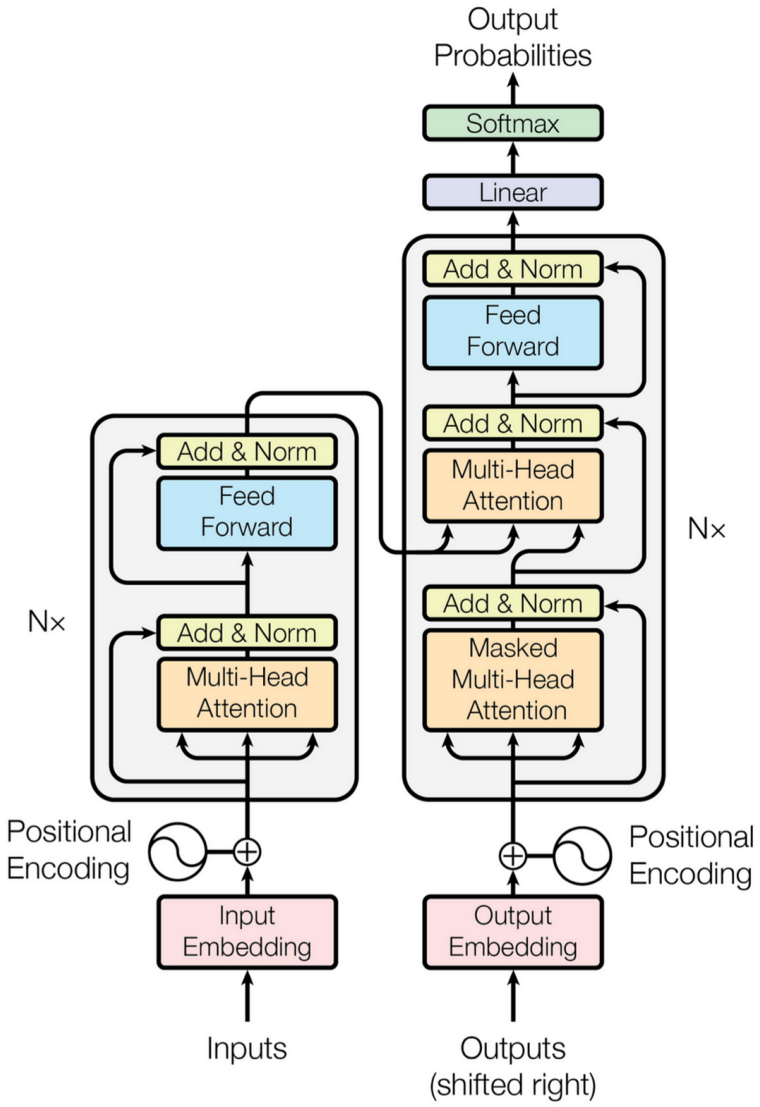


Figure 2.3 Transformer architecture

2.1.3.3 Self-attention mechanism

The attention mechanism was first introduced in paper [19]. It was created to increase encoder-decoder model performance in *seq2seq* tasks. Before this mechanism, the problem in *seq2seq* tasks had been how the decoder used the encoder's representation of the input sequence. The encoder compressed the input sequence into a fixed-length vector, which remained the same size regardless of the length of the input sequence. This caused a problem, especially with longer sequences, because the information in a long sequence was more easily lost because the model could not represent all critical information.

The attention mechanism allows the encoder to create a sequence of vectors, each representing different parts of the input sequence. The decoder uses its learned weights and focuses on the most relevant parts of the input sequence for each token it generates. The decoder's partial output, combined with the context it received from the encoder, helps it predict the next token. [19]

Self-attention is a specific type of attention mechanism that allows each position in the input sequence to be attended to over every other position in the sequence. The transformer has a self-attention mechanism called *scaled dot product attention*. Firstly, it transforms the input sequence vector into three vectors: *query*, *key*, and *value*. The transformation is done by linearly projecting the input sequence (*seq*) three times, using matrices with learned weights: W_q, W_k, and W_v, and it works like this: [17]

$$Q = seq \cdot W_Q, \tag{2.16}$$
$$Q = seq \cdot W_K, \tag{2.17}$$
$$Q = seq \cdot W_V. \tag{2.18}$$

Linear transformation is effectively equivalent to passing data through a dense layer without an activation function. Thus, every of the three transformations results in vectors of a predefined dimension determined by the number of neurons in the layer. The dimensions of Q, K, and V are d_Q, d_K, and d_V respectively.

The operation qk^T constructs attention weights between each pair of tokens from the sequence. The product creates a 2D matrix where each value displays the attention weight between two tokens. The attention weights measure how much focus should each token give to every other token. [20]

How self-attention works is in this section shown on a specific sequence: *"fear of a name increases fear of the thing itself"* The computation of attention weights and visualizations on this example sequence is performed using *BertViz*[21], an interactive tool for visualizing attention in transformer language models. The model used to process the sequence is a pre-trained transformer, *DistilBERT*[22], a simplified version of *BERT*[23] (Bidirectional Encoder Representations from Transformers), which is a transformer-based model designed for natural language processing tasks. The following Table 2.2 shows what the attention weights can look like:

	[CLS]	fear	of	a	name	increases	fear	of	the	thing	itself	[SEP]
[CLS]	.046	.05	.11	.075	.074	.037	.063	.12	.1	.043	.08	.2
fear	.14	.058	.042	.012	.098	.099	.067	.047	.015	.17	.14	.1
of	.21	.066	.056	.039	.076	.075	.067	.057	.053	.12	.094	.086
a	.21	.074	.021	.063	.094	.11	.074	.021	.065	.091	.1	.075
name	.26	.09	.029	.028	.04	.19	.083	.026	.035	.1	.075	.04
increases	.15	.094	.09	.037	.066	.051	.09	.078	.052	.13	.11	.046
fear	.17	.071	.054	.014	.072	.091	.067	.051	.017	.18	.12	.085
of	.24	.072	.061	.05	.068	.067	.063	.054	.063	.11	.095	.06
the	.3	.054	.059	.068	.052	.058	.05	.055	.1	.065	.079	.065
thing	.17	.13	.039	.033	.077	.064	.014	.037	.036	.074	.12	.08
itself	.086	.093	.069	.079	.059	.072	.095	.075	.09	.079	.047	.16
[SEP]	.15	.049	.049	.11	.058	.031	.054	.057	.11	.042	.063	.22

Table 2.2 Attention weights matrix

After that, the product QK^T is scaled by $\frac{1}{\sqrt{d_k}}$ and passed through the *softmax* function. The scaling is important because for large values of d_k, the product's values can become large, which leads to very small gradients in the *softmax* function. The *softmax* function is a function that takes a vector of numbers as its input and returns a probability distribution. The resulting values range from 0 to 1 and sum to 1. It is a function often used in classification tasks to predict probabilities that a data point belongs to a category. Following is the mathematical definition of the function:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \tag{2.19}$$

where $z_i \in \mathbf{z} = (z_1, \ldots, z_n)$.[17, 20]

The result of the *softmax* function applied on the scaled QK^T signals the attention weights of each token relative to one another.

Each of the three vectors—*query*, *key*, and *value*—serves a different purpose. The transformer uses multiple sets of these three vectors to analyze the sequence simultaneously, where each *query*, *key*, and *value* focuses on a different aspect of the input. This is further explained in one of the following section 2.1.3.6.

The *query* represents the information a token seeks from other tokens. The *keys* indicate how relevant each token is to a given *query*; based on how well a *key* matches the *query*, the mechanism determines how much focus to give to a token. The value vector contains each token's actual information, weighted by the attention scores from the *queries* and *keys*. [24, 17, 20]

In conclusion, the self-attention mechanism in transformer functions like this:

$$\text{ScaledDotProductAttention}(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.20}$$

2.1.3.4 Input embedding

Input embedding is applied to the input tokens in both the encoder and the decoder. Each token is mapped into a desired dimensional space through an embedding layer. The dimensionality of the embedding is in the paper [17] determined by the parameter d_{model}, which is set to 512.

2.1.3.5 Positional encoding

The self-attention mechanism does not provide the tokens with information about their position. As was mentioned previously, it processes information using three vectors: *keys*, *queries*, and *values*. It lets the vectors communicate, but it needs to be given information about each token's position. Unlike recurrence-based models (RNN), which process the information in order, thus having information about the token's position, the transformer needs to utilize a *Positional encoding* component, which provides information about the relative position of tokens within the input.[24, 20]

Positional encoding is applied after the *Input embedding*, both in the encoder and in the decoder. It adds a vector of length d_{model} to each embedded token's vector. The value added is determined by the token's position in the sequence *pos* and by the position of elements in the embedding vector i. The computation of the *Positional encoding* differs for even- and odd-indexed positions in the embedding vector and is described by the following equations [17]:

$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \tag{2.21}$$

$$\text{PE}(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \tag{2.22}$$

Figure 2.4 shows what the output of *positional encoding* looks like. The plots illustrate the behavior of the *positional encoding* vectors across different positions in the embedding vector. The left plot represents sine values for the even-indexed embedding dimensions. Each line belongs to a particular embedding dimension, called *feature* in the plot. It is clear that the higher the dimension index, the bigger the period of the sine function. The right plot shows cosine values for the odd-indexed dimensions.

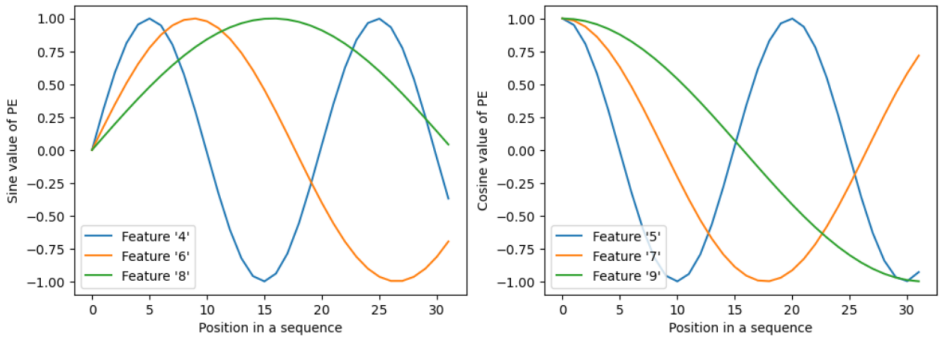


Figure 2.4 Visualization of *Positional encoding* values

The output of the *Positional encoding* has the same dimension as the output of the *Input embedding* so that it can be summed and passed on to the successive layers of the transformer.

2.1.3.6 Multi-head attention

Multi-head attention (*MHA*) utilizes the *self-attention* mechanism several times independently. Each self-attention mechanism is executed in one head; the number of heads in the paper [17] was set to eight. Each head allows the transformer to focus on different aspects and features of the sequence. The vectors W_q, W_k, and W_v are different for each head. The calculation of i-th head works as follows:

$$\text{head}_i(Q, K, V) = \text{ScaledDotProductAttention}(Q, K, V) \tag{2.23}$$

After each head is computed, every output is concatenated and passed to the following layers in the transformer:

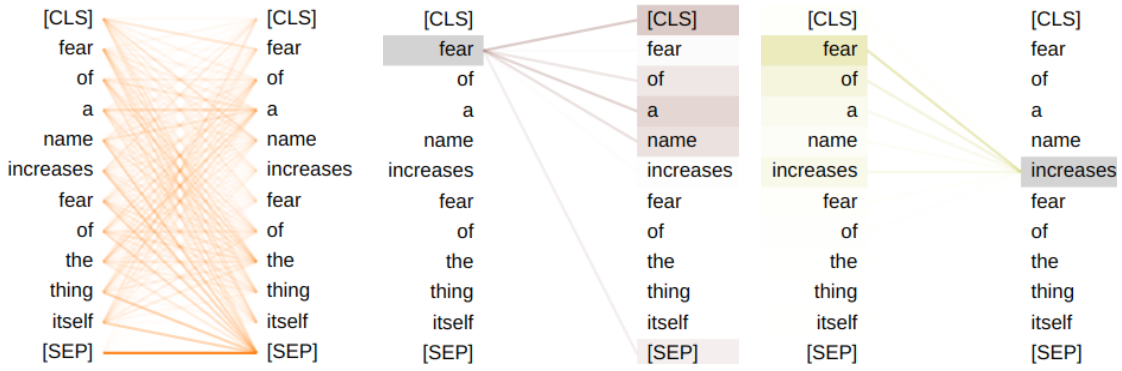
$$\text{MHA}(seq) = \text{Concat}(\text{head}_1(Q_1, K_1, V_1), \ldots, \text{head}_h(Q_h, K_h, V_h)), \tag{2.24}$$

$$Q_i = seqW_i^Q, \tag{2.25}$$

$$K_i = seqW_i^K, \tag{2.26}$$

$$V_i = seqW_i^V. \tag{2.27}$$

Figure 2.5 shows three visualizations of the self-attention mechanism in the example sequence. Subfigures 2.5a, 2.5b, and 2.5c show how tokens attend to other tokens in a sequence. This is represented on the *BertViz* visualization. The first Subfigure, 2.5a, shows the attention distribution of a specific attention head; it represents how each token in the sequence attends to every other token in the sequence. The Subfigure in the middle, 2.5b, represents what tokens the token *fear* attends to. The last Subfigure at the right, 2.5c, displays what tokens attend to the token *increases*. The *DistilBERT* model consists of 12 attention heads. Since each attention head focuses on different features of the input sequence, the visualization looks different for each head, and the same goes for each encoder or decoder layer. The visualization displayed on every one of the Figures comes from the third *DistilBERT* layer, the middle graph comes from the 6th attention head, and the right graph comes from the 9th attention head.



(a) *BertViz* attention distribution **(b)** *BertViz* token *fear* **(c)** *BertViz* token *increases*

Figure 2.5 *BertViz* visuzalizations

2.1.3.7 Masked multi-head attention

The transformer is specialized to solve sequence generation tasks. The goal in sequence generation is to correctly predict the next token based on the tokens the model has generated so far. As mentioned previously, the decoder is the part of the transformer that functions in steps and generates the output sequence. During training, the transformer has access to the whole target sequence and it allows the model to learn its weights based on the correct output.[20]

The self-attention mechanism, explained in section 2.1.3.3, creates a risk in the decoder because it could have access to tokens that should be generated in the future. Being able to see future tokens can mean that the decoder would not learn meaningful patterns in the target sequence but only what tokens will be generated. Alternatively, it would learn patterns that occur in the distant part of the sequence, which would skew its current outputs the wrong way. [20, 18]

The previous Table 2.2 demonstrates the problem where there are attention weight values between a token and other tokens that come after it. The self-attention mechanism is upgraded to a masked self-attention mechanism to prohibit the decoder from seeing the weights of "future" tokens. Table 2.3 presents how the attention weights look in the masked version of the mechanism. The negative infinities will result in zeros after applying the *softmax* function, resulting in tokens not paying attention to future tokens. The dot product attention value from equation 2.20 is transformed by summing the QK^T with a triangular matrix with zeros on and below the diagonal and negative infinity above the diagonal. With the triangular matrix M the equation functions as follows: [17]

$$\text{DotProductAttention}(Q, K, V) = Softmax(\frac{QK^T + M}{\sqrt{d_k}})V \tag{2.28}$$

	[CLS]	fear	of	a	name	increases	fear	of	the	thing	itself	[SEP]
[CLS]	.046	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
fear	.14	.058	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
of	.21	.066	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
a	.21	.074	.021	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
name	.26	.09	.029	.028	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
increases	.15	.094	.09	.037	.066	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
fear	.17	.071	.054	.014	.072	.091	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
of	.24	.072	.061	.05	.068	.067	.063	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
the	.3	.054	.059	.068	.052	.058	.05	.055	$-\infty$	$-\infty$	$-\infty$	$-\infty$
thing	.17	.13	.039	.033	.077	.064	.014	.037	.036	.074	$-\infty$	$-\infty$
itself	.086	.093	.069	.079	.059	.072	.095	.075	.09	.079	.047	$-\infty$
[SEP]	.15	.049	.049	.11	.058	.031	.054	.057	.11	.042	.063	.22

Table 2.3 Masked atttention weights matrix

2.1.3.8 Add&Norm

In sublayer *Add&Norm*, layer normalization is applied to the data. It stabilizes and accelerates training. Unlike batch normalization, which normalizes input for every feature across the batch dimension, layer normalization normalizes the data across the features for each data point. The normalization is performed independently for each data point and is more appropriate for sequence tasks. [25, 26, 17]

The following equations, 2.29, 2.30, 2.31, 2.31, show how layer normalization works. Firstly, the mean and variance of the input data point $\mathbf{x} = (x_1, \ldots, x_d)$ is computed:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i \tag{2.29}$$

$$\sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2 \tag{2.30}$$

Then, every element of the data point is normalized:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}. \tag{2.31}$$

ϵ is a constant preventing division by zero. Finally, each element is transformed with parameter γ and β, which are parameters that will be adjusted during training the model.

$$y_i = \gamma\hat{x}_i + \beta, \tag{2.32}$$

After layer normalization is done, its output is summed with the original input to *Add&Norm*, the operation is called residual connection and can function as follows:

$$y = f(x) + x, \tag{2.33}$$

where f represents layer normalization.

2.1.3.9 Feed forward

In this sublayer, the data is passed through two dense layers. The first one had 2048 neurons in paper [17]. The information about number of neurons was stored in the variable *d_ff*.

It also applies the *ReLU* activation function. The second dense layer has *d_model* neurons and no activation function. How this sublayer works can be summarized using the following equation:

$$\text{FeedForward}(x) = ReLU(xW_1 + b_1)W_2 + b_2, \tag{2.34}$$

where W_1, W_2 are learned weight matrices and b_1, b_2 are learned biases.

2.1.3.10 Additional information and summary

The output of each sublayer, as well as the output of each layer in the *Feed forward* sublayer, is followed by dropout. Dropout is a regularization technique that prevents overfitting. During training, every NN neuron has some probability of being "dropped," meaning it will not participate in the current training step. This technique forces the network not to rely on specific neurons but instead use all its neurons for learning. [27]

The Figure 2.6 shows how a NN looks after dropout disables a set of neurons.

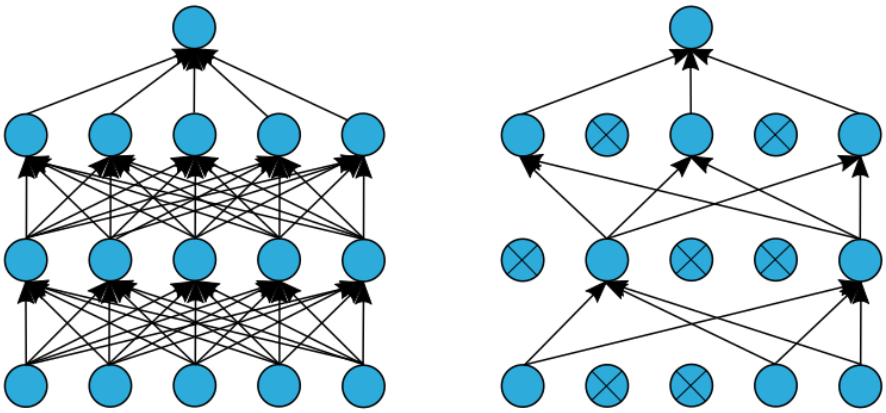


Figure 2.6 Dropout: before application (left) and after (right)

The following Table 2.4 summarizes the key hyperparameters of the transformer model. It details the values of the hyperparameters used in the paper [17].

Hyperparameter	Value
encoder layers	6
decoder layers	6
d_model	512
d_ff	2048
attention heads	8
d_q	64
d_k	64
d_v	64
dropout rate	0.1

Table 2.4 Transformer hyperparameters

2.1.4 Time series

"*A time series is a sequence of observations taken sequentially in time. Many sets of data appear as time series: a monthly sequence of the quantity of goods shipped from a factory, a weekly*

series of the number of road accidents, daily rainfall amounts, hourly observations made on the yield of a chemical process, and so on."[28] In this thesis, time series can be thought of as a finite n-tuple:

$$(f(t_i) \mid i = 1, 2, \ldots, n)\,,\ t_i \in T, \tag{2.35}$$

where T is a set of timestamps, for example "24.10.2017 14:30" $\in T$, and $f : T \to \mathbb{R}$. In addition, function f is not explicitly known, making forecasting challenging.

2.1.4.1 Types of time series

Each point in time can contain any number of variables depending on what is measured. From this point of view, there are two types of time series:

1. *univariate* - each data point contains only one variable,
2. *multivariate* - each data point contains two or more variables.

For multivariate time series, the definition from equation 2.35 can be upgraded to:

$$(f_1(t_i), f_2(t_i), \ldots, f_m(t_i) \mid i = 1, 2, \ldots, n)\,,\ t \in T, \tag{2.36}$$

where $f_j(t)$ represents the j-th variable of the series which has m variables.

Figure 2.7 shows two line plots created in the *matplotlib* library[29]: the left with a *univariate* time series and the right with a *multivariate* time series. The data for both plots is not real and is generated by ChatGPT[30], but they capture the meaning of *univariate* and *multivariate* series well. The *univariate* time series represents a temperature measured in January 2020. The plot with the *multivariate* time series captures temperature, humidity, and wind speed from the same time frame.

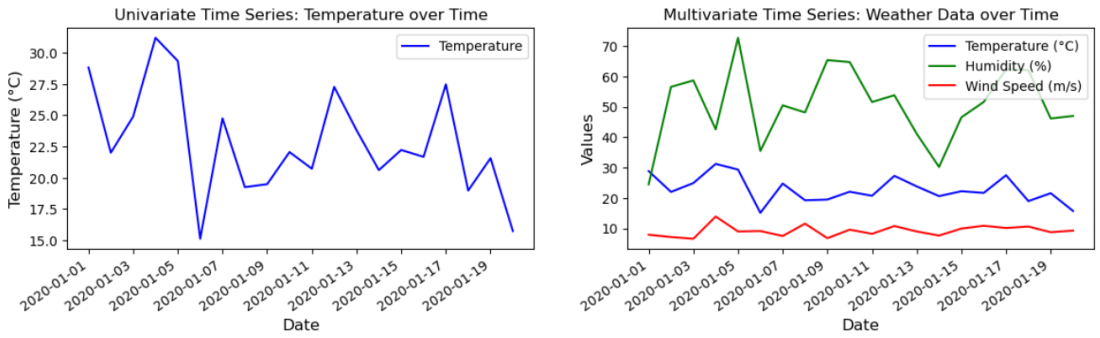


Figure 2.7 Examples of univariate and multivariate time series

2.1.4.2 Components of time series

Many factors can influence the data that make up a time series, determining how a time series looks and what its properties are. Each of these factors behaves differently and can be categorized into specific groups. While not every factor is present in every single time series, the components are:[31]

- *level* - baseline value around which the data oscillates,
- *trend* - long-term direction of the data,
- *seasonality* - periodic fluctuations occurring at regular intervals,
- *noise* - irregular, unpredictable and random property of the data.

Figure 2.8 shows a plot of a known dataset, *Air passengers*, showing the number of passengers of US airlines each month in the years 1949-1960. It is a suitable example of a time series to showcase some of its components, especially *trend* and *seasonality*. Upon examining the plot, an upward *trend* in the number of passengers can be seen over time. Besides the *trend*, the plot goes up and down over time, with peaks in summer months when the number of passengers was higher and a decline in passengers in winter months.

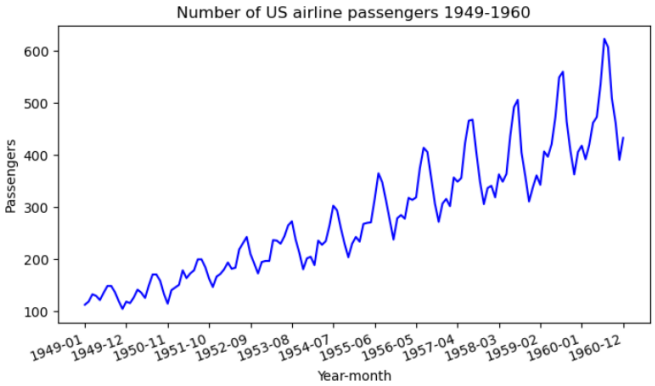


Figure 2.8 Visualization of US airline passengers dataset

There are approaches to decomposing a time series into components mentioned before. The approaches are: *additive model* and *multiplicative model*. In the *additive model*, any data point at a specific time is assumed to be a linear combination of its components. Thus, it can be computed as a sum of the components at that time. In the *multiplicative model*, the components are combined multiplicatively, and the relationship between the components of the time series is expected to be non-linear. The following equations 2.37 and 2.38 show how a data point at a time t is computed:

$$Y_t = L_t + T_t + S_t + N_t, \tag{2.37}$$

$$Y_t = L_t \cdot T_t \cdot S_t \cdot N_t, \tag{2.38}$$

where Y_t is an element of a time series at time t, L_t is a *level* value at time t, T_t is a *trend* value at time t, S_t is a *seasonality* value at time t and N_t is a *noise* value at time t.[31]

2.1.4.3 Stationarity

Data sorted in time provides information about the series. When considering the statistical properties of time series over time, there are two types of series:

- *stationary*,
- *non-stationary*.

A time series $(f(t_i) \mid i = 1, 2, \ldots, n)$ is *stationary* if:

1. $E(f(t_i)^2) < \infty$,
2. $E(f(t_i))$ is a constant, independent of t,
3. $Cov(f(t_i), f(t_{i+k}))$ is independent of t for each k, [32]

where $E(X) = \sum_x x \cdot P(X = x)$ is expected value of a discrete random variable, $P(X = x)$ is the probability that random variable X is equal to x, and $Cov(X, Y) = E((X - EX)(Y - EY))$

is covariance. *Stationary* time series do not have *trends* or *seasonality*. Statistical tests can be performed to check *stationarity*, for example *Dickey-Fuller test*.[31]

Figure 2.9 shows an example of *stationary* and *non-stationary* time series. The data comes from the foreign exchange market (Forex) and it contains information about the *EUR/USD currency pair*. Both plots show 96 data points from the same time frame, from 2024-07-10, 00:00 to 2024-07-11, 00:00. The prices were recorded every 15 minutes. The left plot displays price of the currency pair, while the right plot shows how the price changed from one time step to another.

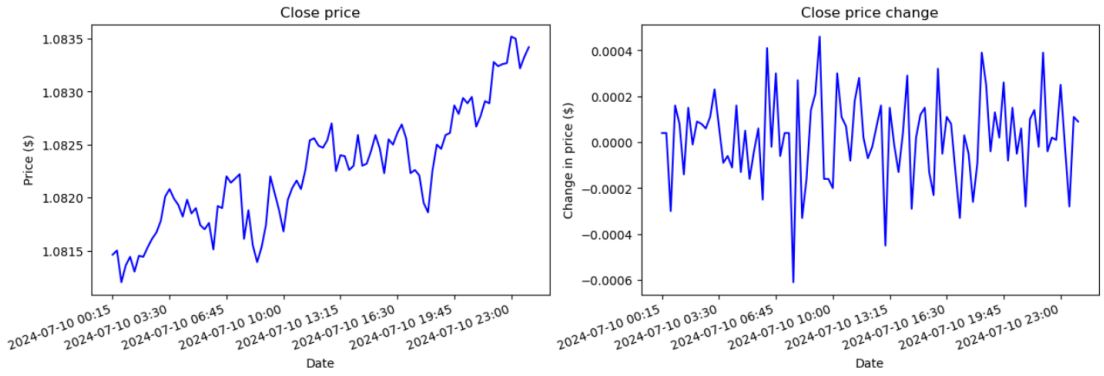


Figure 2.9 Example of stationary and non-stationary time series

2.1.4.4 Time series forecasting

Information in time series can provide context for how the series will behave in the future. Predicting the series' next steps can be relatively simple if there is a clear *trend* or *seasonality*. Even when it is not simple, forecasting methods can be utilized, and predictions can be made, such as training a neural network, or any other model, on historical financial data and measuring its performance, which will be done in the experiments in this thesis. Different fields where time series forecasting makes sense include:

- finance,
- weather,
- supply chain management,
- hotel occupancy rate,
- spread of an epidemic.

Forecasting can be categorized into two types based on the number of steps that need to be predicted:

1. *one-step forecasting*,
2. *multi-step forecasting*: depending on how the predicted variables are constructed, there are types:
   a. *iterated forecasting*: essentially an *one-step forecasting* with the predicted values used as additional input until the desired number of forecasts is reached,
   b. *multi-output forecasting*: the whole output sequence is produced directly.

Generally, there is a sequence of n elements sorted in time, $(x_1, x_2, \ldots, x_n)$, and the goal is to predict the following p elements, $(x_{n+1}, \ldots, x_{n+p})$. If p is equal to one, then it is *one-step forecasting*.[31]

The authors of the paper "*Transformers in Time Series: A Survey*"[33] write about how transformers are well-built for natural language processing tasks or computer vision and that transformer-based models have been increasingly used for time series forecasting tasks due to the ability to capture dependencies in sequential data. They also write about the limitations of these models, such as problems with capturing seasonal patterns or computational complexity. They give an overview of some transformer-based models: Autoformer[34], FEDformer[35], or LogTrans[36], each with their adaptations and improvements.

Autoformer was the second model chosen to be implemented besides the original Transformer for its ability to model short-term and long-term dependencies through its time series decomposition mechanism and auto-correlation attention, which also reduces the computational complexity of the self-attention mechanism from the Transformer. The model will be introduced further in section 2.3.

2.2 Data

This section offers more insight into the data and transformation techniques utilized in the experiments. Section 2.2.1 explains the foreign exchange market (Forex). Then, it shows what the data from this market looks like and how it can be further analyzed to extract more information. Section 2.2.2 operates with time series as a signal and introduces the field of signal processing. It shows how to work with signals and how they can be modified or filtered, and it presents and explains one specific filter used in the experiments.

2.2.1 Forex

Forex may be the largest financial market in the world. There are multiple reasons to support this statement:

- This market trades a massive amount of value. Daily trading volume reached $7.5 trillion in April 2022. In comparison, the National Association of Securities Dealers Automated Quotations (NASDAQ, one of the largest stock exchanges) was at $260,867,444,019 on 2022-12-04, about 28 times less.[37, 38]
- It is a market with many users worldwide: governments, central banks, corporations, hedge funds, and individuals.[39]
- It is open 24 hours a day except for weekends, as opposed to, for example, the New York Stock Exchange (NYSE), the largest stock exchange by market capitalization of listed companies, which is open from 9:30 a.m. to 4:00 p.m. (eastern time) from Monday to Friday.[40, 41]

The asset traded in this market is called a currency pair. It consists of two different currencies and it represents their values relative to each other. The currencies in a currency pair are called:

- *base currency*,
- *quote currency*.

In the most traded *EUR/USD* currency pair, the euro is the *base currency*, and the US dollar is the *quote currency*. The pair's value indicates how much of the *quote currency* it takes to buy one unit of the *base currency*. If the pair's value increases, the *base currency* strengthens relative to the *quote currency*.[42]

Buying the pair means purchasing the *base currency* and selling the *quote currency*. Buying the pair might be worth it when the value of the *base currency* is assumed to rise relative to the *quote currency*. This process of assuming and buying the pair is called a *long position*. On the other hand, expecting the value of the *base currency* to fall relative to the *quote currency* means selling the currency pair, which means taking a *short position*.

Forex data is available in various forms depending on the time intervals in which it is collected. There are, for example, 1-minute, 5-minute, daily, monthly data. Each data point can be represented as a candlestick, which illustrates four currency pair prices at an exact time. The prices are:[43]

- opening price - the price of a currency pair at the first trade carried out in the trading period.
- high price - the maximum price of all executed trades during the period.
- low price - the minimum price of all executed trades during the period.
- closing price - the price of a currency pair at the last trade carried out in the period.

Forex data often comes with an additional feature, volume, which is the total amount of an asset traded during the period.

Knowing how a price changes from one time step to another is handy when working with prices. This is deducted from subtracting the prices at the two time stamps and transforming the subtracted value into *pips*, or price interest points. A *pip* is the slightest possible price movement, and for most currency pairs, including the *EUR/USD*, it is set to 0.0001. For example, if a price changes by $0.0001 in the *EUR/USD* currency pair, it is said to have changed by one *pip*.

2.2.1.1 Technical indicators

Technical indicators are mathematical calculations based on historical prices or volumes used to analyze and predict possible future market movements. The following explains two technical indicators that are used in the experiments, the *relative strength index* (*RSI*) and *Williams percent range*. *RSI* measures the speed and change of price movements. It ranges from 0 to 100 and defines *overbought* and *oversold* assets. When it is above 70, it suggests an asset is *overbought*, while below 30, it indicates *oversold*. The *RSI* can help predict potential price changes in near future by signaling whether an asset is gaining or losing strength. *RSI* is calculated as:

$$\begin{aligned} RSI(n) &= 100 - \frac{100}{1 + RS(n)}, \\ RS(n) &= \frac{\text{average\_gain}(n)}{\text{average\_loss}(n)}, \\ \text{average\_gain}(n) &= \frac{1}{n}\sum_{i=1}^{n} \max(C_i - C_{i-1}, 0), \\ \text{average\_loss}(n) &= \frac{1}{n}\sum_{i=1}^{n} \max(C_{i-1} - C_i, 0), \end{aligned} \tag{2.39}$$

where n is the number of previous time steps to be considered, C_i is the close price at time step i, *average_gain* is a sum of price increases timestep-to-timestep, *average_loss* is a sum of price declines timestep-to-timestep.[44]

The *Williams percent range* index also uses *overbought* and *oversold* terms to describe whether the price will likely rise or fall. It ranges from -100 to 0. Values above -20 indicate an *overbought* condition, and values below -80 indicate an *oversold* condition:[44]

$$WILL\%R(n) = \frac{\max(H_i, H_{i-1}, \ldots, H_{i-(n-1)} - C_i}{\max(H_i, H_{i-1}, \ldots, H_{i-(n-1)}) - \min(L_i, L_{i-1}, \ldots, L_{i-(n-1)})} \cdot (-100). \tag{2.40}$$

There are many other technical indicators; using them together is convenient for getting more information from past prices and volumes.

2.2.2 Signal processing

When trading on the financial market, a financial instrument's price can be considered a signal.

When a person speaks, their vocal cords vibrate and produce sound waves. These sound waves are a type of signal. They travel through the air into another person's ears, where they are processed and delivered into the brain, where they are interpreted. When a person looks at a (2D) image, they see a signal - in this case, as a function of two variables, one for each dimension.[45]

A signal is a piece of information represented quantitatively across various domains. The domain can be time, space, frequency, or other parameters. For the previous examples, the quantitative manner can be the loudness of the sound or color. The domain can be time or space. In the context of this thesis, the quantitative manner is the price of a currency pair, and the domain is time. Two basic types of signals are continuous and discrete. Continuous signals are defined for every instant of time, and discrete signals are defined at specific points in time. Although a sequence of prices is often visualized as a plot line so it appears continuous, in reality, the prices are measured at specified intervals. Therefore, the signal will be considered discrete in this thesis.

The price of any financial asset fluctuates over time. Price changes happen daily, and yet, in many cases, there is a clear long-term trend. Figure 2.10 shows the closing price of the *EUR/USD* currency pair during May 2022. The line shows a gradual increase in price from roughly half the month to its end. At the same time, many fluctuations occur hourly, daily, and weekly. This behavior highlights the concept of frequency in a signal. Low frequencies are represented in the long-term trend in the price; in this example, it would be the price increase during the second half of the month. On the other hand, the daily fluctuations reflect higher frequencies present in the signal. A signal's frequencies can be analyzed through techniques like *Fourier analysis*, which is explained in the following section 2.2.2.1.

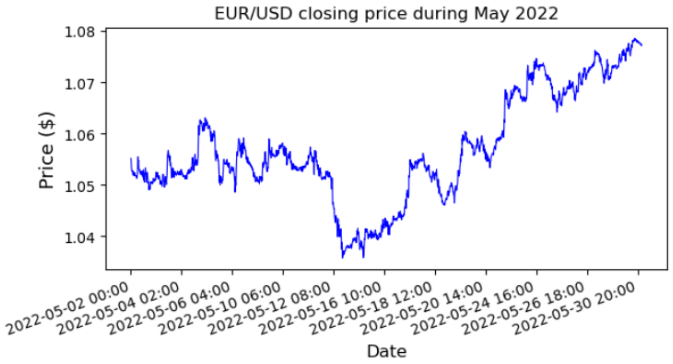


Figure 2.10 Low and high frequencies in a signal

Signals can be received, analyzed, and transformed to extract useful information. For example, sound can be made to be better quality or to lose unwanted noise. Transformation applied to an image can find sharp edges or locations with high and low brightness. These transformations can be performed using signal filters. There are four types of filters, categorized by whether they retain or strengthen specific frequencies or attenuate them:

- *low-pass*,
- *high-pass*,

- *band-pass*,
- *band-stop*.

A *low-pass* filter has a specified cutoff frequency, which indicates that frequencies below this threshold should be kept and higher frequencies should be attenuated. A *high-pass* filter performs the opposite of a *low-pass* filter. A *band-pass* filter needs a specified range of frequencies it passes through, while a *band-stop* filter keeps the frequencies outside this range.[45] Figure 2.11 shows how the filters work. On the x-axis, there are frequencies; on the y-axis, there is a *frequency response*, or *gain*, which signals how much information from a frequency is allowed to pass through the filter.[46]

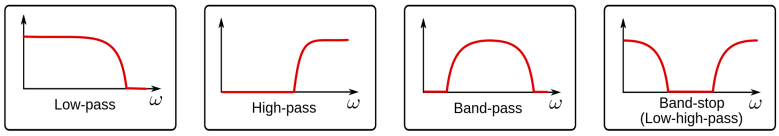


Figure 2.11 Types of filters based on what frequencies they allow to pass through

In the experiments in this thesis, a *low-pass* filter will be used to retain longer-term information while attenuating unwanted noise in the time series.

2.2.2.1 Fourier Transform

Generally, *Fourier transform* is a mathematical technique to transform data from one domain to another. It is used in numerical analysis or signal processing. When working with signals, specifically in the time domain, the information in the signal can be represented in two ways: as a function of time or frequency. The *Fourier transform* allows a signal to be converted from the time domain to the frequency domain, and the *inverse Fourier transform* converts the frequency function back to the function of time. How to switch between the two representations is depicted in the following equations: 2.41, 2.42:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-2\pi i f t}\, dt, \tag{2.41}$$

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{2\pi i f t}\, df, \tag{2.42}$$

where f is frequency, t is time, i is an imaginary unit, h is the function of time and H is the function of frequency. When t is measured in seconds, the frequency is measured in cycles per second. Besides frequency, angular frequency is often used, for which the units are radians per second[47]:

$$\omega = 2\pi f. \tag{2.43}$$

In the case of discrete signals, *discrete Fourier transform* (*DFT*) and *inverse discrete Fourier transform* (*IDFT*) are utilized:

$$H_n = \sum_{k=0}^{N-1} h_k \cdot e^{-2\pi i k n / N}, \tag{2.44}$$

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n \cdot e^{2\pi i k n / N}, \tag{2.45}$$

where h_k represents the discrete signal at time k, H_n is the discrete signal in the frequency domain, N is the number of values in the discrete signal.[47]

2.2.2.2 Fast Fourier transform

The *Fast Fourier transform* (*FFT*) is a set of algorithms for faster computing the *DFT*. The *DFT* has time complexity N^2, while the *FFT* can be calculated in $N \cdot logN$ operations. One of the better-known algorithms for computing *FFT* is the *Cooley-Tukey* algorithm, which uses the *divide-and-conquer* algorithm that breaks the problem into several smaller subproblems and solves each subproblem recursively. The idea behind the *Cooley-Tukey* algorithm is as follows: equation 2.44 can be rewritten as a sum of two *DFT*s, each consisting of only even and odd elements of the original *DFT* and each of length $\frac{N}{2}$:

$$H_n = \sum_{k=0}^{\frac{N}{2}-1} (h_{2k} \cdot e^{-2\pi ink/(\frac{N}{2})}) + e^{\frac{-2\pi in}{N}} \cdot \sum_{k=0}^{\frac{N}{2}-1} (h_{2k+1} \cdot e^{-2\pi ink/(\frac{N}{2})}) = H_n^e + e^{\frac{-2\pi in}{N}} \cdot H_n^o \quad (2.46)$$

This method can be used recursively on each H_n^e and H_n^o, which creates H_n^{ee}, H_n^{eo}, H_n^{oe}, H_n^{oo} and so on. It is convenient to have N equal to the power of two because, after several operations, each sum will contain only one element.[47, 48]

After the smaller *DFT* parts are calculated, they are combined in a structure called a *butterfly diagram*, where the smaller parts are combined in pairs and weighted by *twiddle factors*, computed during the dividing part of the algorithm, $e^{\frac{-2\pi ik}{N}}$.[47, 48]

The *inverse Fast Fourier transform* (*IFFT*) shown in equation 2.45 can be calculated similarly since the equation is very similar to the *FFT*, apart from the exponent and multiplication by $\frac{1}{N}$. The *FFT* and *IFFT* are used in one of the models implemented in this thesis, the *Autoformer*.

2.2.2.3 Butterworth filter

The *Butterworth filter* will be used to filter signals in experiments in this thesis. Depending on its configuration, it can function as a *low-pass*, *high-pass*, *band-pass*, or *band-stop* filter. The following text and the experiments will work with the filter in its *low-pass* version. It has two input parameters:

- *order*,
- *cutoff frequency*.

The *cutoff frequency* separates the lower frequencies that are to be retained from the higher frequencies that are to be filtered out. The *order* is an integer that controls how sharply the frequencies above the *cutoff frequency* are attenuated.[45]

Figure 2.12 shows a plot similar to the *low-pass* filter in Figure 2.11. It illustrates how different values of the *order* influence the filter's *frequency response*. The higher the *order*, the less the high frequencies appear in the output.

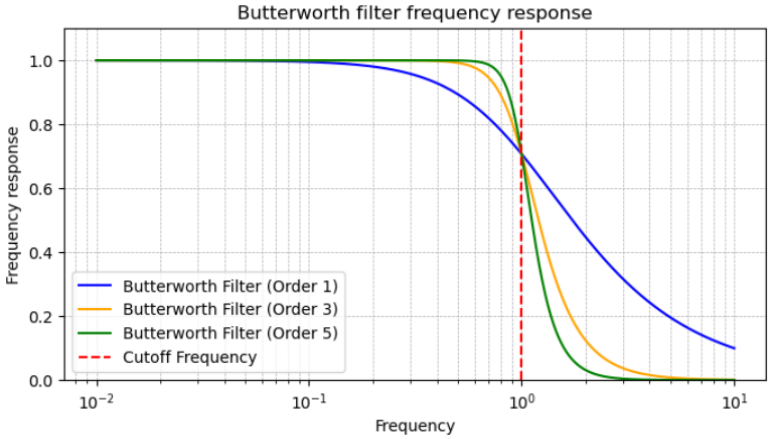


Figure 2.12 *Butterworth*'s *frequency response* on different *order* values

The *Butterworth filter* is designed to have a maximally flat *frequency response* in the pass-band, meaning the filter maintains a *frequency response* close to one for frequencies below the cutoff. Another feature of this filter is a *smooth transition*, which means the response curve gradually decreases as the frequencies pass through the *cutoff frequency*. The following equation 2.47 explains how the *order* and *cutoff frequency* influence what frequencies are allowed to pass through the filter:

$$|H(\omega)| = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}}. \tag{2.47}$$

ω is the angular frequency of a signal, the $|H(\omega)|$ is the *frequency response*, ω_c is the *cutoff frequency*, and N is the *order*.[45]

In the case of discrete signals, the *Butterworth filter* transforms each signal element, as shown in the following equation:

$$y[t] = \sum_{i=0}^{N} b_i x[t-i] - \sum_{j=1}^{N} a_j y[t-j]. \tag{2.48}$$

$y[t]$ is the filtered output at time step t, $x[t]$ is the input signal at time step t, N is the *order*, and b_i and a_j are filter coefficients. Initial conditions for this equation can be handled in multiple ways, such as setting zeros, or *padding*. The *scipy* library, which will be used in the experiments, uses the *padding* method.[49]

2.3 Autoformer

The Autoformer was introduced in paper *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting* [34]. It is a type of deep learning model designed specifically for time series forecasting. The researchers who introduced him created it with the following upgrades from the transformer model:

- The self-attention mechanism's computational complexity is quadratic with respect to the input sequence's length, making it more challenging to work with longer time series. The Autoformer model does not utilize the self-attention mechanism but the *Auto-Correlation* mechanism, which has computational complexity O(L*logL), where L is the sequence length.[34]
- The self-attention mechanism has difficulty analyzing long-term time series because it tends to focus on local interactions rather than the overall structure of the time series. To check

the global properties of the series, the Autoformer uses series decomposition, which finds the trend and seasonality components of the series. [34]

Like the transformer, the Autoformer uses an encoder-decoder architecture. Unlike the transformer, which does the *iterated forecast*, the Autoformer performs the *multi-output forecast* and yields the output sequence at once. In the following Figure 2.13 is an overall structure of the model: [34]

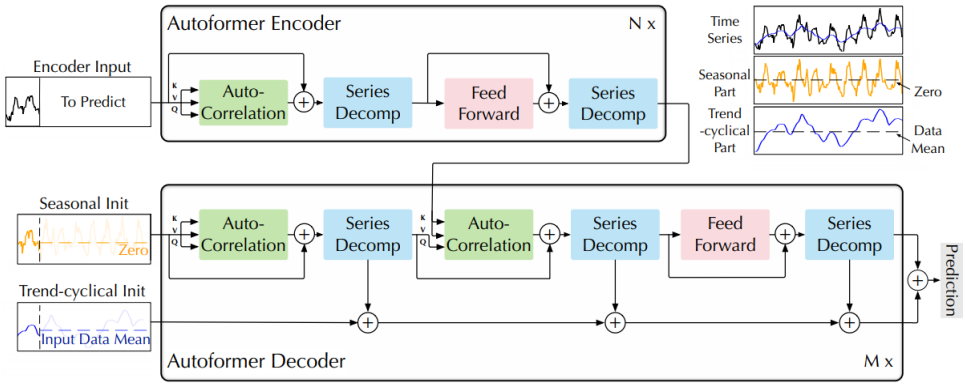


Figure 2.13 Autoformer architecture

The encoder and decoder layers contain the following sublayers:

- Auto-Correlation,
- Series Decomp,
- Feed Forward.

The *Auto-correlation* mechanism finds periodic relationships and dependencies across different data points in a time series. It finds sub-series with similar patterns by calculating autocorrelation between different time steps in the series. When it finds them, it aggregates them using *time delay aggregation*, which combines the information from them. [34, 31]

The autocorrelation is computed using the *convolution theorem*[50], which states that the Fourier transform of the convolution of two functions is the product of their Fourier transform. In other words, convolution in the time domain corresponds to multiplication in the frequency domain. The *convolution theorem* can be applied since autocorrelation is closely related to convolution. A significant advantage of calculating the autocorrelation using the Fourier transform is the reduced time complexity of the mechanism.[51, 52]

The *Series Decomp* sublayer separates time series into *trend-cyclical* and *seasonal* parts. It is performed using "*moving average to smooth out periodic fluctuations and highlight the long-term trends*".[34].

The *Feed Forward* works similarly to the one from the Transformer, introduced in section 2.1.3.9.

2.4 Related work

This section summarizes three works related to this thesis's topic, price prediction. The following works predicted prices of either forex assets or stocks. Each subsection contains information about the exact goal of each work, the methods and models used for predictions, the data used, how it was processed, and the results achieved.

2.4.1 Transformer-Based Deep Learning Model for Stock Price Prediction: A Case Study on Bangladesh Stock Market

The researchers of paper [53] used the transformer-based model to predict stock prices in the Bangladesh Stock Market, specifically on the Dhaka Stock Exchange.

The work was done with daily and weekly data. For the daily data, the goal was to predict the following day's closing price based on the previous day's values. For weekly data, the goal was to predict the following week's closing price based on the values from previous weeks. Data contained these seven features:

1. the trading code (a unique identifier assigned to each company listed on the Dhaka Stock Exchange),
2. date,
3. opening price,
4. high price,
5. low price,
6. closing price,
7. volume.

Data from eight companies were considered, each with its own trained model, so 16 models were created, eight for daily data and eight for weekly data.

During data processing, the authors deemed it crucial to make the data stationary. This was done by extracting each feature's previous time step value from the current time step value. Then, the values were replaced with the mean average of window size ten. Then, values were normalized with min-max normalization. Ultimately, the data was split into train, validation, and test sets.

The model used was a transformer-based model consisting of just an encoder. It received sequences of length eight, with each time step containing five features:

1. open price,
2. high price,
3. low price,
4. close price,
5. volume.

The data is augmented with *Time2Vec* encoding[54] to encode the information about time. The model's output is a single continuous variable (a price return).

Since the model's output is a continuous variable, the authors solved a regression problem and chose *mean absolute error* (*MAE*) and *root mean squared error (RMSE)*, which is the square root of the *MSE* value, to estimate the models' performance. In table 2.5 and table 2.6, there are *MAE* and *RMSE* for both the daily and weekly data for training, validation, and testing sets for each of the eight companies, respectively.

	Daily					
	Training		Validation		Testing	
Trading Code	RMSE	MAE	RMSE	MAE	RMSE	MAE
1JANATAMF	3.94E-02	3.07E-02	4.68E-02	3.77E-02	6.59E-02	4.76E-02
AAMRANET	6.69E-02	5.45E-02	7.71E-02	6.45E-02	1.02E-01	8.43E-02
ABBANK	4.90E-02	3.94E-02	4.84E-02	3.90E-02	6.96E-02	5.20E-02
ACI	1.62E-02	1.25E-02	1.95E-02	1.55E-02	1.93E-02	1.55E-02
ACIFORMULA	3.18E-02	2.54E-02	3.48E-02	2.79E-02	4.06E-02	3.23E-02
AGRANINS	3.39E-02	2.66E-02	6.09E-02	4.67E-02	8.39E-02	6.25E-02
ALLTEX	2.31E-02	1.71E-02	2.11E-02	1.62E-02	2.23E-02	1.74E-02
DELTALIFE	2.05E-02	1.61E-02	1.33E-02	1.08E-02	2.54E-02	1.94E-02

Table 2.5 Dhaka Stock Exchange: results on daily data

	Weekly					
	Training		Validation		Testing	
Trading Code	RMSE	MAE	RMSE	MAE	RMSE	MAE
1JANATAMF	5.78E-02	4.33E-02	7.54E-02	5.49E-02	7.70E-02	5.53E-02
AAMRANET	1.30E-01	1.06E-01	3.19E-01	3.00E-01	1.72E-01	1.42E-01
ABBANK	8.35E-02	6.32E-02	5.90E-02	4.65E-02	1.98E-01	1.59E-01
ACI	5.83E-02	4.73E-02	5.90E-02	3.98E-02	7.79E-02	6.55E-02
ACIFORMULA	6.82E-02	5.39E-02	9.13E-02	8.24E-02	9.31E-02	7.64E-02
AGRANINS	5.78E-02	4.54E-02	1.07E-01	8.79E-02	1.21E-01	9.09E-02
ALLTEX	5.05E-02	3.52E-02	2.38E-02	2.00E-02	3.87E-02	2.98E-02
DELTALIFE	5.46E-02	4.05E-02	2.27E-02	1.79E-02	6.95E-02	4.80E-02

Table 2.6 Dhaka Stock Exchange: results on weekly data

2.4.2 Predicting Forex Rates using Sentiment Analysis on Financial Articles

The researchers of this paper[55] focused on mood extracted from financial articles in combination with historical prices and technical indicators. The goal was to predict the closing prices of the EUR/USD currency pair in the forex market. They performed the task with regard to the *Efficient Market Hypothesis*, which was influenced by Eugene Fama and his research, *Efficient Capital Markets: A Review of Theory and Empirical Work*[56], from 1970. The *Efficient Market Hypothesis* says that financial markets process information efficiently, making prices of financial assets reflect all available information about themselves. Thus, it is very difficult to time or predict the market consistently.

They worked with two kinds of data, one containing financial data and the other texts. The first data set contains historical financial forex data. It is from September 2018 to May 2021 for the EUR/USD currency pair. The historical data were measured every hour, with roughly 16,000 data points. Each data point contains the following features:

- opening price,
- high price,
- low price,

- closing price,
- volume.

During data processing, they discarded features low, high and open and added 10 technical indicators, having 14 features in total.

The second kind of data is financial texts. The training data was acquired from the *financial phrase bank*[57] corpus, consisting of 5000 financial texts. These texts provide an opportunity to influence the price of the EUR/USD currency pair based on their sentiment. Thus, the data presents a classification problem with categories:

- positive,
- neutral,
- negative,

where neutral sentiment occurs roughly 60 % of the time. This created a challenge that their model needed to achieve higher than 60 % accuracy in this classification problem to accomplish improvement.

The financial articles they used for predictions come from the financial news websites *fxstreet.com* and *investing.com*, from the same time frame as the historical forex prices. Together, there are 7413 articles.

Their model is created to predict the closing price of the EUR/USD currency pair one hour into the future. The basis of the model is a *FinBERT*, which is a pre-trained natural language processing model to analyze the sentiment of the financial text. There is an upgrade of *FinBERT*, *FinBERT-SIMF*, which can analyze the sentiment of financial articles' titles, as well as forex or cryptocurrency data. The researchers created their own model, building another upgrade to *FinBERT-SIMF*, *FinBERT-LSIMF*, named *Financial Bidirectional Encoder Representations from Transformers based Long Sentiment and Informative Market Feature*. To analyze not only the articles' titles but also the bodies, they built a *Longformer*-based model[58], to analyze long text sequences.

As for data, the results are of two types. One for the text classification and one for the price prediction. Table 2.7 displays the *cross-entropy loss* and *accuracy* for the dataset the *Longformer*-based model was trained on to classify sentiment in financial articles. The dummy classifier is a model that always predicts neutral. The table 2.8 shows the performance in predicting the closing price. The researchers measured performance on four models:

- *FinBERT-LSIMF* - able to analyze titles and bodies of articles,
- *FinBERT-SIMF* (researchers' version) - able to analyze titles of articles,
- *FinBERT-SIMF*[59] (original) - able to analyze titles of articles,
- *FinBERT-IMF*[59] - not able to analyze text, only historical forex data

	Cross-entropy loss	**Accuracy**
Training set	0.72	0.69
Validation set	0.67	0.72
Test set	0.67	0.72
Dummy classifier	-	0.61

Table 2.7 Text classifier results on the *financial phrase bank* corpus

	Validation MAPE	Test MAPE
FinBERT-LSIMF	0.102	0.161
FinBERT-SIMF	0.124	0.399
FinBERT-SIMF (original)	0.121	0.291
FinBERT-IMF	3.623	6.695

Table 2.8 Closing price prediction *MAPE* values

2.4.3 Predicting Stock Closing Prices in Emerging Markets with Transformer Neural Networks: The Saudi Stock Exchange Case

The goal of paper[60] is to predict the next day's closing price. The researchers discussed the usability of deep learning in several fields, such as computer vision, natural language processing, and medicine. They aimed to use it for financial data, specifically on stocks on the Saudi Stock Exchange. They aspired to use the self-attention mechanism to learn nonlinear patterns in (highly volatile) time-series data while discussing the limitations of other approaches, such as multi-layer perceptrons, long short-term memory networks, and recurrent neural networks.

The Saudi Stock Exchange, also known as Tadawul, divides its listed companies into categories based on their sector. The researchers used data from the Banks Index, Materials Index, Telecommunication Services Index, and Tadawul All Share Index, which they downloaded from 1993-01-02 to 2021-06-17. The data was treated as time series with each data point containing features:

- opening price,
- high price,
- low price,
- closing price,
- volume,
- stock name.

The stock name feature was removed, the data was normalized using *MinMaxScaler* of the *scikit-learn* library[5], and missing values were removed. Table 2.9 represents what hyperparameters were considered and their values.

Hyperparameter	Value
Learning rate	0.001
Optimizer	AdamW
Batch size	256
Epochs	500
Early stopping (epochs)	70
Early stopping (monitoring parameter)	validation loss
Loss Function	MSE

Table 2.9 Hyperparameters overview

The model they used was a *vision transformer*[61], the main elements of which are a linear layer for embedding, a stack of transformer blocks with multi-head self-attention, feed-forward layers, and an output linear layer.

The model was evaluated on multiple metrics and for different batch sizes, as demonstrated in the tabular structures below:

Loss function	MAE	MSE	RMSE	MAPE

Batch size	2	4	8	16	32	64	128	256	512

In table 2.10, each row represents a data category, and each column is a loss function. For each loss function, there is a batch size that resulted in the lowest value of the loss function, along with the value of the loss function for that batch size.

	Loss metric							
	MAE		**MSE**		**RMSE**		**MAPE**	
Sector	b_size	value	b_size	value	b_size	value	b_size	value
TASI	8	0.0001	8	0.0001	8	0.154	8	1.681
TBNI	2	0.0012	2	0.0013	2	0.1697	2	0.696
TTSI	4	0.0020	4	0.0101	4	0.1885	4	1.712
TMTI	4	0.0198	4	0.0021	4	0.2361	4	1.727

Table 2.10 Saudi Stock Exchange: loss functions values and batch sizes

2.4.4 Summary

The first paper[53] reviewed was the *Transformer-Based Deep Learning Model for Stock Price Prediction: A Case Study on Bangladesh Stock Market* in subsection 2.4.1. Advanced Chemical Industries Limited, with the trading code ACI, achieved the best daily data results regarding both *RMSE* and *MAE*. It got $1.93 \cdot 10^{-2}$ *RMSE* and $1.55 \cdot 10^{-2}$ *MAE*. The company Alltex Industries Limited, with the trading code ALLTEX, achieved the best results on weekly data and both metrics. It got $3.87 \cdot 10^{-2}$ *RMSE* and $2.98 \cdot 10^{-2}$ *MAE*. The results presented in this paragraph are from the testing set of the data. The authors of this paper also mentioned the option to solve a classification problem instead of a regression problem: "*Although the proposed model addresses the price prediction problem as a regression problem, it can be easily modified to deal with classification problems. For example, a model can be designed to predict whether the price will rise or fall in the upcoming days (thereby dealing with a binary classification problem).*"

The paper[55], *Predicting Forex Rates using Sentiment Analysis on Financial Articles* focused on predicting a forex EUR/USD currency pair one hour into the future based on historical data. The historical data included prices and financial articles from the same time. Taking the sentiment of the articles into account on the specific data they worked with helped the predictions. They also warned that there is no guarantee that it will help in general and suggested ways of improvement.

Chapter 3
Experiments

This chapter explains the implemented models and the data they were trained on. Section 3.1 introduces the source financial data used to train and evaluate the models. Section 3.2 shows how the implemented models work, using pseudocode to explain the algorithms they are based on better. Section 3.3 describes in detail the fundamental approach to the experiments. Section 3.4 presents the available code. Finally, section 3.5 provides the results of the experiments.

All the code is written in Python, and the models are built, trained, and evaluated using the TensorFlow framework. For a comfortable working environment, some of the code is written into Jupyter notebooks, and the initial operations with the source data are performed using the pandas library. The visualizations inside the notebooks and in the following sections are done using the matplotlib library. The code is available at https://github.com/pribylr/bp.

3.1 Data

This part presents the datasets that served as input for the experiments. Section 3.1.1 shows how the original data looks, where it was downloaded, and why it was selected. The following section, 3.1.2, explains and visualizes the data to give more insight into its structure. It also explains how the input data was transformed, what information was derived from the data, and what information was omitted to better understand the processes executed in the experiments. Finally, it explains how the target variable looks like.

3.1.1 Input datasets

The focus was placed on the more popular currency pairs during the experiments in this thesis. Table 3.1 shows the five most traded pairs in the world[42]. The experiments were built on data from three currency pairs, all in the top 6. The following pairs were used:

- *EUR/USD*,
- *GBP/USD*,
- *USD/CAD*.

The data intervals differ slightly to include a little variety. The data with the EUR/USD pair was measured every 15 minutes; the GBP/USD pair had five-minute intervals, and the USD/CAD pair had one-minute intervals.

The experimental part of the thesis was built with the assumption that higher-interval data would be harder to predict because the more extended period would contain more information

Currency pair	Volume (%)
EUR/USD	27.95
USD/JPY	13.34
GBP/USD	11.27
AUD/USD	6.37
USD/CAD	5.22
USD/CHF	4.63

Table 3.1 Top 6 most traded currency pairs

that would be harder to learn and make predictions from. To be more specific, in the case of daily intervals, there are around 260 data points in one year, taking into account the market's opening hours. That is a relatively small amount of data points distributed across a wide time interval. During this one year, a lot can happen, for example, a shift in politics, interest rate change, the start of a pandemic, or war. Model trained on such data will learn patterns that these events created. Such a model may become a liability for future data, where such events will not occur or influence the price in different ways.

On the other hand, the experiments in this thesis were performed over shorter intervals, focusing only on short-term price fluctuations. The datasets were downloaded from [62] as *csv* files.

3.1.2 Data observation

After downloading the datasets, they were loaded into Jupyter notebooks as a Pandas DataFrame. Each dataset contains the following features:

- date – date including year, month, day and exact time,
- open – the price of the first trade during period,
- high – the highest price of all trades during period,
- low – the lowest price of all trades during period,
- close – the price of the last trade during period,
- volume – the amount of asset traded.

The first step of data observation was looking for missing values in the datasets. Since no missing values were found, no approach to dealing with them needed to be applied.

Following the text on *(non)stationarity* in time series in section 2.1.4.3, the original data was copied, and the second dataset was used to represent *stationary* time series. To help achieve *stationarity*, each data point was subtracted from the preceding one. Table 3.2 shows the original data. Table 3.3 shows the subtracted data. It shows changes in the price from one time step to the next. Values in both tables are depicted on one of the three datasets used in the experiments, the currency pair *EUR/USD* at a 15-minute interval, as is every observation demonstrated in this section.

date	open	high	low	close	volume
2020-10-08 12:45	1.17593	1.17625	1.17542	1.17551	16392
2020-10-08 13:00	1.17553	1.17562	1.17326	1.17388	28294
2020-10-08 13:15	1.17388	1.17447	1.17377	1.17442	21323
2020-10-08 13:30	1.17444	1.17446	1.17364	1.17407	20293
2020-10-08 13:45	1.17408	1.17471	1.17386	1.17446	23214

Table 3.2 EUR/USD price data summary

date	open	high	low	close	volume
2020-10-08 12:45	-0.00055	-0.00038	-0.00023	-0.00040	-2660
2020-10-08 13:00	-0.00040	-0.00063	-0.00216	-0.00163	11902
2020-10-08 13:15	-0.00165	-0.00115	0.00051	0.00054	-6971
2020-10-08 13:30	0.00056	-0.00001	-0.00013	-0.00035	-1030
2020-10-08 13:45	-0.00036	0.00025	0.00022	0.00039	2921

Table 3.3 EUR/USD price data summary, subtracted prices

Values in columns open, high, low, and close seem very similar to each other. This raises the question of what relationships exist between these features. Figure 3.1 contains correlation matrices between all five features on the original and the subtracted data. The correlation coefficient is of the type Pearson, which measures the linear relationship between data.

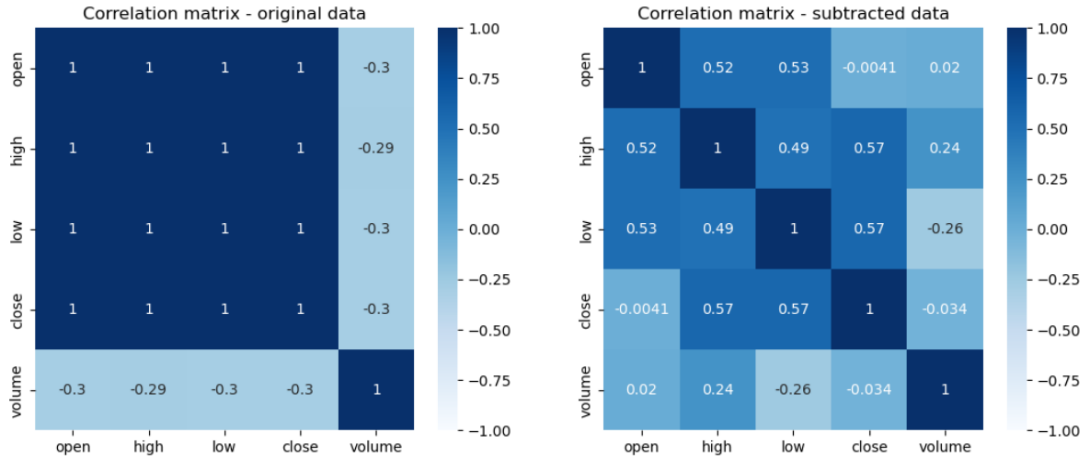


Figure 3.1 Correlation matrices: stationary (left) and non-stationary (right) data

The original and subtracted data contain essentially the same information. One exact price value from any time step is enough to reconstruct the original data from the subtracted data. And yet, there is a big difference in the correlation coefficients. In the case of the original data, where there are highly correlated features, there is a possibility of discarding all of these features, except the closing price, since the closing price is the target variable. This possibility may come with a trade-off:

Keeping open, high, low:

+ provides more context.

− contains redundant information.

The experiments are performed while keeping the opening, high, and low prices. There may also be a risk of overfitting, but thanks to dropout, neural networks are generally more resistant to it. In addition, the extra context provided by all of the prices can be valuable in short-term scenarios.

Next, the *Butterworth* filter is applied to each feature open, high, low, and close price. Figure 3.2 a plot of 50 data points of the original and subtracted closing price and its filtered version from 2020-10-08, 15:15 to 2020-10-09, 03:30. Models can be trained on the actual or the filtered prices. Both options came with another set of advantages and disadvantages. From observing and comparing the plot, following conclusion could be made:

Replacing price with its filtered version:

\+ removes noise, thus makes learning easier.

− deletes the actual information from the dataset.

Since noise can be attributed to any source and it is very challenging to learn its pattern, the filtering was done to extract relevant information from the data. So, the filtered open, high, low, and close prices replaced the original prices.

Figure 3.2 Closing price and Butterworth filter

Then, the *Butterworth* filter's parameters were chosen: the *order* and the *cutoff frequency*. Figure 3.3 shows how different values of the cutoff frequency influence the filter's output. Four plots display the same 50 data points from 2020-10-08, 15:15 to 2020-10-09, 03:30, but each with a different version of the filtered signal. The examples show a problem similar to the previous one, where there was a choice between retaining the original data and simplifying the signal. In the experiments, the *order*'s value was chosen to be 5 for each case, while the *cutoff frequency* varied depending on the currency pair and its time interval. Table 3.4 shows the values of the *order* and *cutoff frequency* used on the input datasets.

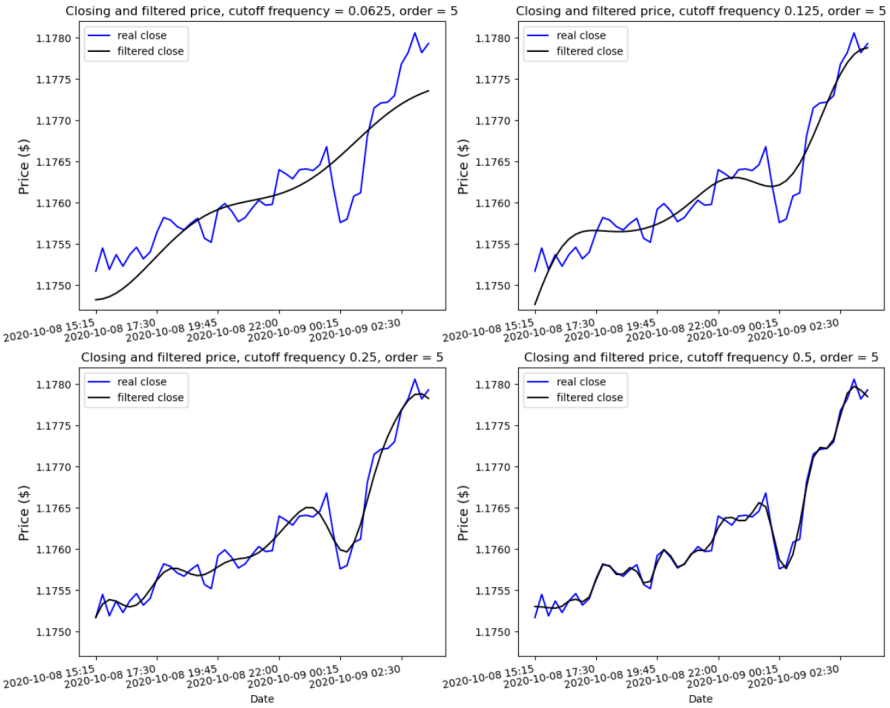


Figure 3.3 Different frequencies of *Butterworth* filter

Currency pair	Time interval	Order	Cutoff frequency
EUR/USD	15 minutes	5	$\frac{1}{8}$
USD/CAD	1 minute	5	$\frac{1}{12}$
GBP/USD	5 minutes	5	$\frac{1}{6}$

Table 3.4 Parameters of *Butterworth* signal on input datasets

3.2 Models

Two models were implemented in this thesis, and the following sections explain how they work. The models are the Transformer, introduced in paper [17], and the Autoformer [34].

3.2.1 Transformer

The transformer model implementation can be found in the file bp/src/vannila_transformer.py. The whole model consists of the following classes:

- *Transformer*,
- *Decoder*,
- *DecoderLayer*,
- *Encoder*,
- *EncoderLayer*,

- *MultiHeadAttention*,
- *ScaledDotProductAttention*,
- *FeedForward*,

with an additional function *PositionalEncoding*. All the mentioned classes are subclasses of the tensorflow.keras.Layer[63] class since they contain learnable weights.

The model is trained using data that is split into batches. Each batch has a shape of (B, L, F), where B represents the batch size, L is the sequence length, and F is the number of features for each token in the sequence. During training, each batch is passed through the model. Firstly, it goes into the encoder. Before passing the data into encoder layers, *Input embedding* and *Positional encoding* are applied. The *Input embedding* transforms the data into a shape of (B, L, d_model), where *d_model* is the dimensionality of the embedding space; it is a fixed size representing each token. This process is shown in the following algorithm 1:

Algorithm 1 Encoder algorithm

Input: x
x ← *InputEmbedding*(x)
x ← *PositionalEncoding*(x)
for each i in $1, \ldots,$ encoder_layers_num **do**
  x ← *EncoderLayer*(x)
end for
Output: x

Inside each encoder layer, the data is passed through its sublayers: *Multi-head attention*, *Feed forward*, and *Add&Norm*. The layer normalization is performed directly using a library function [64] and the residual connections inside the *Add&Norm* are implemented as in the following algorithm:

Algorithm 2 Encoder layer algorithm

Input: x, x, x
y ← *MultiHeadAttention*(x, x, x)
y ← *LayerNormalization*(x+y)
z ← *FeedForward*(y)
z ← *LayerNormalization*(y+z)
Output: z

The following Figure 3.4 shows how the MHA is implemented. Firstly, the MHA sublayer consists of multiple heads with each head performing the attention mechanism and the computation happens independently for each head. Each head creates three matrices, *query*, *key*, and *value*, as is explained by equations 2.18. The matrices are used to calculate the *ScaledDotProductAttention*. At the end of the MHA mechanism, the output of each head is concatenated back to the original size of the data so the subsequent sublayers can operate with the data.

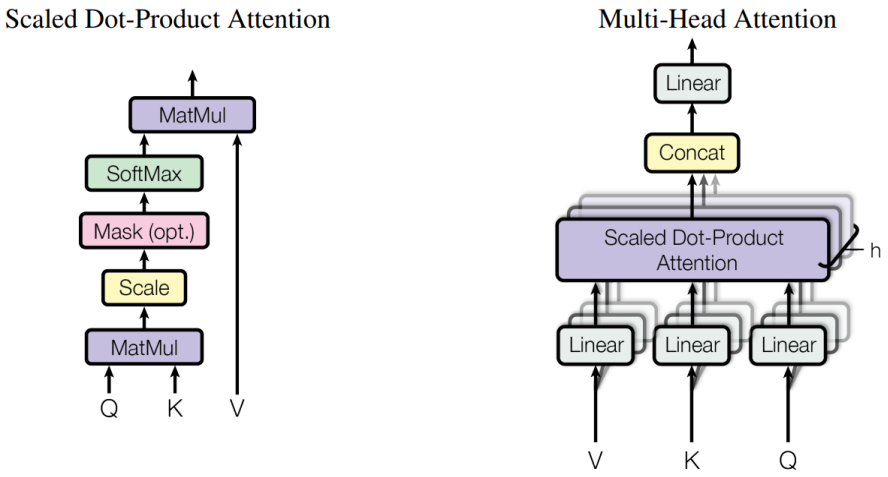


Figure 3.4 Transformer MHA mechanism

The output of the last encoder's layer is passed to the decoder. The behavior of the decoder depends on whether it is being trained or makes predictions.

Algorithm 3 Decoder algorithm

Input: last_token, encoder_output, target_sequence
if training == **True then**
  x ← *Decoder.train*(encoder_output, target_sequence)
else
  x ← *Decoder.infer*(last_token, encoder_output)
end if
Output: x

3.2.1.1 Training phase

When the model is trained, the decoder works with the encoder output and the whole target sequence, manipulating its weights to minimize training loss. The decoder in the training phase works similarly to the encoder from algorithm 1, apart from one dense layer at the end, which projects the data into a desired dimension, one for all models in this thesis since the target variable is always only the closing price:

Algorithm 4 Decoder train algorithm

Input: last_token, encoder_output, target_sequence
x ← *InputEmbedding*(target_sequence)
x ← *PositionalEncoding*(x)
for each i in $1, \ldots,$ decoder_layers_num **do**
  x ← *DecoderLayer*(x, encoder_output)
end for
x ← linear_out(x)
Output: x

3.2.1.2 Inference phase

When it makes predictions, it works in the inference mode. In the inference mode, the decoder works in steps, and during each step, it works with the encoder output and its generated sequence. There are several methods [65] for determining the input the decoder should use at its first step, such as a zeros-filled token or the last several tokens from the input sequence. In this experiment, the decoder received one token, the most recent one, the last token in the input sequence. Since the last token of the input sequence is part of the output sequence during this phase, it is discarded at the end:

Algorithm 5 Decoder inference algorithm

Input: last_token, encoder_output, target_sequence
generated_sequence ← last_token
while generating output **do**
  x ← generated_sequence
  x ← *InputEmbedding*(x)
  x ← *PositionalEncoding*(x)
  for each i in $1, \ldots,$ decoder_layers_num **do**
    x ← *DecoderLayer*(x, encoder_output)
  end for
  x ← linear_out(x)
  generated_sequence.append(x)
end while
Output: generated_sequence[1:]

3.2.1.3 Multi-head attention in decoder

The following algorithm 6 for the decoder layer works similarly in the training and inference phases. The first difference is the sequence length during the inference phase, which gets progressively longer as the decoder generates the output step by step. The decoder layer executes two MHA sublayers. The first one is the masked MHA mechanism, and the second is an MHA mechanism, which receives the output of the encoder in the form of the *key* and *value* and the output of the previous masked MHA as *query*.

Algorithm 6 Decoder layer algorithm

Input: encoder_output, decoder_input
x ← *MaskedMultiHeadAttention*(decoder_input, decoder_input, decoder_input)
x ← *LayerNormalization*(x+decoder_input)
y ← *MultiHeadAttention*(x, encoder_output, encoder_output)
y ← *LayerNormalization*(x+y)
z ← *FeedForward*(y)
z ← *LayerNormalization*(y+z)
Output: z

3.2.2 Autoformer

The Autoformer model is implemented in the file https://github.com/pribylr/bp/blob/master/src/autoformer.py and it consists of the following classes:

- *Autoformer*,
- *Encoder*,

- *EncoderLayer*,
- *Decoder*,
- *DecoderLayer*,
- *Autocorrelation*,
- *FeedForward*,
- *Series_decomp*.

In the algorithm 7 is the *Series Decomp*. The smoothing method used in the experiments is the *TensorFlow's AvgPool1D*.[66] The output of this sublayer is the seasonal and trend extracted information from the input data:

Algorithm 7 *Series Decomp* algorithm

Input: x
$x_t \leftarrow$ AvgPool(x)
$x_s \leftarrow$ x - x_t
Output: x_s, x_t

3.2.2.1 Encoder

The following algorithms, 8 and 9, explain how the encoder works. The encoder is responsible for handling the *seasonal* component of the input sequence; it discards the *trend* output of the *Series Decomp* sublayer. Each *Series Decomp* sublayer utilizes the residual connections:

Algorithm 8 Autoformer encoder algorithm

Input: x
for each i in $1, \ldots,$ encoder_layers_num **do**
  x $\leftarrow$ *EncoderLayer*(x)
end for
Output: x

Algorithm 9 Autoformer encoder layer algorithm

Input: x
y $\leftarrow$ *Auto-Correlation*(x,x,x)
y, _ $\leftarrow$ *Series Decomp*(x+y)
z $\leftarrow$ *Feed Forward*(y)
z, _ $\leftarrow$ *Series Decomp*(y+z)
Output: z

3.2.2.2 Decoder

The encoder's output contains past *seasonal* information. This information is used as cross-information to help the decoder, which also processes the *trend* property of the input data and model prediction results.

Algorithm 10 Autoformer decoder algorithm

Input: x_s, x_t, encoder_output
S ← x_s
T ← x_t
for each i in $1, \ldots,$ decoder_layers_num **do**
  S, T ← *DecoderLayer*(S, T, encoder_output)
end for
out ← linear_out(S+T)
Output: out

Algorithm 11 Autoformer decoder layer algorithm

Input: x_s, x_t, encoder_output
x ← *Auto-Correlation*(x_s, x_s, x_s)
S, T1 ← *Series Decomp*(x+x_s)
x ← *Auto-Correlation*(S, encoder_output, encoder_output)
S, T2 ← *Series Decomp*(x+S)
x ← *FeedForward*(S)
S, T3 ← *Series Decomp*(x+S)
T ← x_t+T1+T2+T3
Output: S, T

3.2.2.3 Auto-correlation mechanism

The following algorithm 12 shows how the *Auto-correlation* mechanism works in the *Autoformer*. As in the MHA sublayer in the transformer, the *Auto-correlation* mechanism uses *queries*, *keys*, and *values*. The FFT is applied to the *query* and the *key*, converting the data from the time domain to the frequency domain. Then, the key is conjugated. Since the values in the frequency domain are complex numbers, the complex conjugate of a complex number $a+bi$ is $a-bi$, where a is the real part and b is the imaginary part of the number, and i is the imaginary unit. Then, the result of the multiplication of the *query* and *key* is processed by the IFFT to transform the data back to the time domain.

The *convolution theorem* states that convolution in the time domain can be performed as multiplication in the frequency domain. That is why the query and the conjugated key are multiplicated, and the result is passed to the IFFT. The result of the IFFT operation stores the autocorrelation values between the query and the key for different lags (time shifts). For example, suppose the result shows a strong autocorrelation at lag 5. In that case, it means that the behavior of the *query* strongly correlates with the behavior of the *key* shifted by 5 time steps. This fact will be reflected in the *time domain aggregation* mechanism, which will use the behavior of the series at lag 5 to model predictions. Then, the *Top_k*[67] function is applied to select the most relevant time lags based on the autocorrelation values. Only these top k most relevant lags are considered in the time delay aggregation mechanism, so the predictions are based only on the parts of the series that correlate the most. The *Top_k* function returns the highest values from the input and their indices.

Algorithm 12 *Autoformer Auto-correlation* mechanism

Input: x, x, x
Q, K, V ← createQKV(x, x, x)
Q, K ← FFT(Q), FFT(K)
Corr ← IFFT(Q · Conjugate(K))
values, indices ← Topk_k(Corr)
out ← time_delay_aggregation(values, indices, V)
Output: out

3.2.3 Baseline models

The models predict a sequence of outputs, which are combined to determine whether the price increased or decreased during the output sequence compared to the price at the end of the input sequence. Therefore, the models were evaluated using binary classification. To be aware of the quality of the predictions, two simple baseline models are presented in the following sections, and the results from the Transformer and the Autoformer are compared with these baseline models.

3.2.3.1 Random 50 % model

Since the price can change in two ways, one of the baseline models is a random model predicting each category with 50 % probability. This model's expected value of accuracy is 0.5. The models should achieve more than 0.5 accuracy when dealing with binary classification tasks. If they do not, the models are insufficient.

3.2.3.2 Baseline model based on past value

The second simple model works with the assumption that if the price rose or declined from the start to the end of the input sequence, it is likely to increase or decrease from the beginning to the end of the output sequence.

The following Figure 3.5 shows two plots: the left one with correct prediction based on this model and the right one with incorrect prediction. Both plots contain three vertical lines: the first line represents the start of the input sequence, the middle one represents the end of the input sequence and the start of the output sequence, and the last line represents the end of the output sequence. On the right plot, it is clear that the price is lower at the end of the input sequence than at the beginning, which indicates further price decline based on this model, which did not happen. Thus, this prediction is considered wrong. The performance of this model is available at https://github.com/pribylr/bp/blob/master/simple_baseline.ipynb

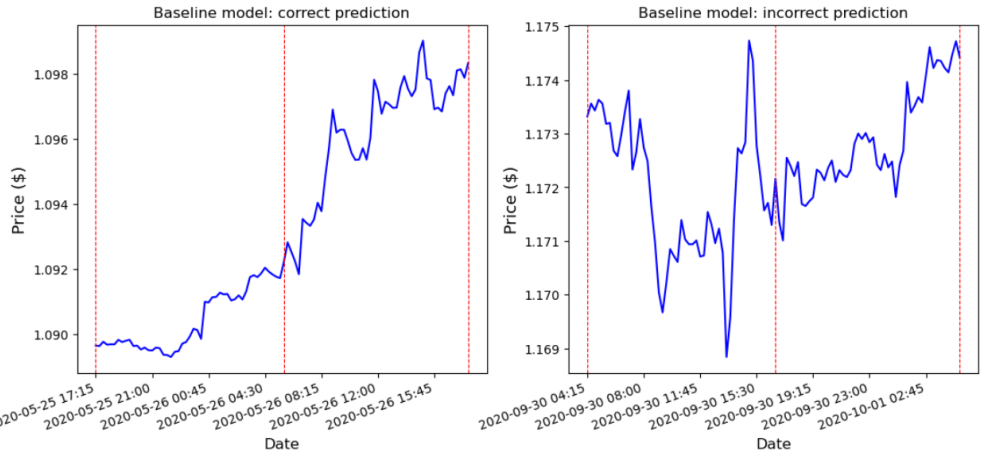


Figure 3.5 Baseline model: correct (left) and incorrect (right) prediction

3.3 Experiments overview

After the data is preprocessed, including filtering its signals and computing technical indicators, it is split into sequences. Each data point consists of an input sequence and a target sequence, which follows right after the input. The lengths of the sequences are pre-defined and differ for each of the experiments in the project. Creating the input and output sequences is called a sliding window method. A fixed-size window slides over the data and takes the time steps inside the window. Then, the window shifts by one step, repeating the process. After this method is finished, there are two 3D objects, in the notebooks called *Xdata* and *ydata*. The shape of these objects is (N, L_{in}, F) and $(N, L_{out}, 1)$, where N is the number of sequences, L_{in} is the length of the input sequence, L_{out} is the length of the output sequence, and F is number of features.

After the input and target sequences were created, they were split into training, validation, and test sets in a ratio of 60:20:20. The following Table 3.5 show how much data contains each dataset:

Currency pair	rows in dataset	amount: training set	amount: test set
EUR/USD	99,999	59,931	19,978
USD/CAD	80,000	47,889	15,963
GBP/USD	60,000	35,969	11,990

Table 3.5 Amount of data in input datasets

Essentially, every model in the experiments is trained to predict a sequence of numerical values. There are two types of these sequences:

- Price prediction – in this case, the models the output of the models is treated as a raw price of a currency pair. Table 3.2 presents how the prices on historical data look, and Figure 3.2 (on the left) presents a plot of the prices.
- Change in price prediction – examples of how data representing changes in a price are in the Table 3.3 and Figure 3.2 (on the right).

The next step is to clarify what the models were trained to predict. Figure 3.6 shows the input and target sequence, separated by a red vertical line. The input data lacks other features to simplify the image, and only the filtered closing price is displayed.

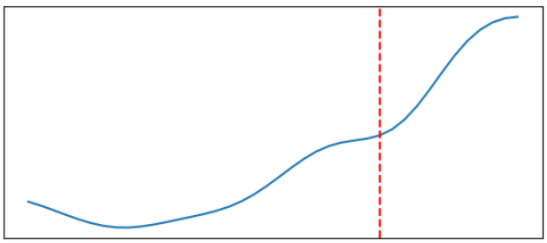


Figure 3.6 Input and target sequence

The models are trained to predict several future time steps. Depending on whether the data was non-stationary, a currency pair's closing prices are predicted. If it was stationary, a series of currency pair price changes is predicted. From the predicted sequence, it is derived whether the price increased during the predicted time steps or decreased, turning the task into a binary classification problem. The model evaluation results, which contain the classification metrics *accuracy*, *precision*, *recall*, and *f1 score*, are presented in the following section 3.5.

Table 3.6 shows the values of Transformer hyperparameters used in notebooks [1], [2], and [3]. Table 3.7 shows hyperparameter values of Transformer model in notebook [4]

Hyperparameter	Value
encoder layers	1
decoder layers	1
d_model	32
d_ff	32
attention heads	4
d_q	32
d_k	32
d_v	32
dropout rate	0.1
batch size	32

Table 3.6 Transformer hyperparameters in experiments

Hyperparameter	Value
encoder layers	1
decoder layers	1
d_model	32
d_ff	32
attention heads	8
d_q	32
d_k	32
d_v	32
dropout rate	0.1
batch size	32

Table 3.7 Transformer hyperparameters in experiments

Table 3.8 displays hyperparameters of the Autoformer model used in notebook [5]. Table 3.9 shows the values from notebook [6].

[1] https://github.com/pribylr/bp/blob/master/transformer_in_seq.ipynb
[2] https://github.com/pribylr/bp/blob/master/transformer_out_seq.ipynb
[3] https://github.com/pribylr/bp/blob/master/thesis_showcase.ipynb
[4] https://github.com/pribylr/bp/blob/master/transformer_predict_price.ipynb
[5] https://github.com/pribylr/bp/blob/master/autoformer_predict_price.ipynb
[6] https://github.com/pribylr/bp/blob/master/autoformer_predict_price_move.ipynb

Hyperparameter	Value
pool size	64
autocorrelation_heads	4
c	4
d_model	32
d_ff	32
encoder_layers	1
decoder_layers	1
dropout rate	0.1
batch size	32

Table 3.8 Autoformer hyperparameters in experiments

Hyperparameter	Value
pool size	64
autocorrelation_heads	4
c	4
d_model	32
d_ff	32
encoder_layers	2
decoder_layers	1
dropout rate	0.1
batch size	32

Table 3.9 Autoformer hyperparameters in experiments

3.4 Code overview

The experiments consist of several Jupyter notebooks. Each notebook contains data loading, pre-processing, model training, and evaluation. The notebooks predicting prices are:

- autoformer_predict_price.ipynb – The file contains an Autoformer model trained on data regarding the USD/CAD currency pair.
- transformer_predict_price.ipynb – Transformer model, trained on USD/CAD currency pair.

The following notebook include models trained to predict changes in currency pair's price:

- autoformer_predict_price_move.ipynb – In this file, an Autoformer model was trained on data on the EUR/USD currency pair.
- transformer_in_seq.ipynb – This file contains three Transformer models, all trained on the same data, but each worked with a different input sequence length. The currency pair on which this experiment was conducted was the GBP/USD pair.
- transformer_out_seq.ipynb – Similar to the previous notebook, this one contains three Transformer models. They were trained to output varying-length sequences. The pair used was the EUR/USD pair.

3.5 Results

In the case of working with non-stationary data, both models achieved overall worse results. The Transformer and Autoformer model used the USD/CAD currency pair with a 1-minute interval for this task. The prediction period was set to 30 steps, or half an hour. The following table 3.10 displays the metrics. The first observation that can be derived from this is that the Transformer achieved worse than 50 % on all four metrics, making it worse than a random model:

	Accuracy	Precision	Recall	F1 score
Autoformer	0.513	0.513	0.570	0.540
Transformer	0.482	0.477	0.349	0.403
base model 1	0.500	0.500	0.500	0.500
base model 2	0.494	0.486	0.482	0.484

Table 3.10 Results: Price

The models trained to predict closing price changes did so on the currency pair EUR/USD with 15-minute intervals. On this data, the Transformer achieved significantly better results than in the previous task, with all four metrics above 70 %:

	Accuracy	Precision	Recall	F1 score
Autoformer	0.507	0.516	0.533	0.524
Transformer	0.756	0.786	0.716	0.749
base model 1	0.500	0.500	0.500	0.500
base model 2	0.489	0.498	0.495	0.496

Table 3.11 Results: price change

The following two tables, 3.12 and 3.13, present the results of experiments that focused on comparing different lengths of input and output sequences. In the first case, the aim was to discover if a longer input sequence could provide more context for how the series will look in the feature. In the second case,

Input sequence length	Accuracy	Precision	Recall	F1 score
32 (160 minutes)	0.783	0.791	0.781	0.786
16 (80 minutes)	0.761	0.778	0.737	0.757
8 (40 minutes)	0.764	0.736	0.835	0.782

Table 3.12 Results: varying input sequence length

Output sequence length	Accuracy	Precision	Recall	F1 score
4 (60 minutes)	0.713	0.732	0.695	0.713
8 (120 minutes)	0.756	0.786	0.716	0.749
16 (240 minutes)	0.576	0.728	0.274	0.398

Table 3.13 Results: varying output sequence length

3.5.0.1 The threshold method

Since the experiments are performed on historical data, the price of each currency pair is known. Even when predicting the change of the pair's price, the predicted changes can be cumulatively summed, and the final price at the end of the output sequence can be constructed and compared to the actual value.

This part of the experiment aims to get better results by only predicting an increase or decrease if it predicts that the price at the end of the output sequence is significantly higher than its value at the beginning. With this method, the task transforms into a ternary classification with the following classes:

- **1** – the price is predicted to increase significantly,
- **0** – the price is predicted to change by very little, so it is risky to predict an increase or decrease,
- **-1** – the price is predicted to decrease significantly.

The code uses the parameter *threshold* to measure a significant change in a price. If the price changes above the *threshold*, the classification belongs to category **1** or **-1**; otherwise, it is classified as **0**.

This method was used on the Transformer model, which was trained on data regarding the EUR/USD pair. Table 3.14 shows the model's performance in a standard way. The results from the following two tables come from the file https://github.com/pribylr/bp/blob/master/ thesis_showcase.ipynb.

	Accuracy	Precision	Recall	F1 score
Transformer	0.75	0.78	0.72	0.75
base model 1	0.50	0.50	0.50	0.50
base model 2	0.49	0.50	0.50	0.50

Table 3.14 Results: varying output sequence length

The following two Figures, 3.7 and 3.8, show how the values of the four metrics change depending on the value of the threshold parameter. Firstly, the original task was transformed to solve a ternary classification with the three categories mentioned at the start of this section. The left plot in Figure 2.1 shows that the accuracy approaches 100 % as the value of the threshold increases. This is because the higher the threshold the price change has to overcome, the fewer data points will overcome it, so most of the data points will be classified as **0**. At the same time, the higher the threshold, the more the actual data points will be of category **0** because the market is not so volatile that the price would change by tens or hundreds of percent every several minutes or hours. With most of the actual data points belonging to the "neutral" category and most of the predictions belonging to the same category, it is clear why accuracy would approach value **1**.

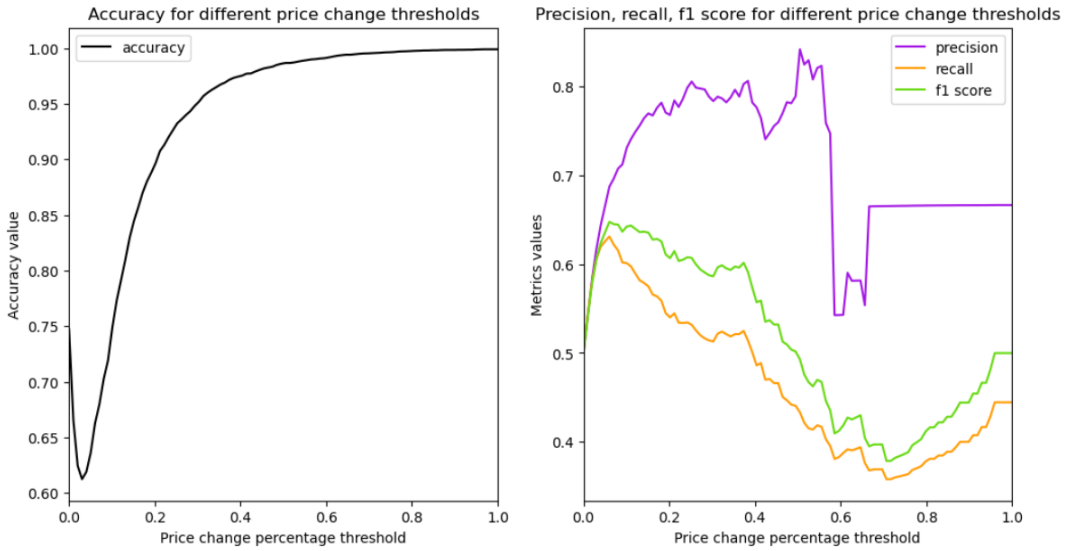


Figure 3.7 Ternary classification; accuracy (left) and precision, recall, f1 score (right)

To compare results using this threshold method with the task where this method was not used, the problem was transferred back to binary classification. This was done by discarding the data points the model classified as category **0**. With only categories **1** and **-1** left, Figure 3.8 shows the values of the metrics based on different *threshold* values. The plots also contain the results from the same task that did not utilize this method, which results are in the previous Table 3.14, to compare if this method helps achieve any improvement.

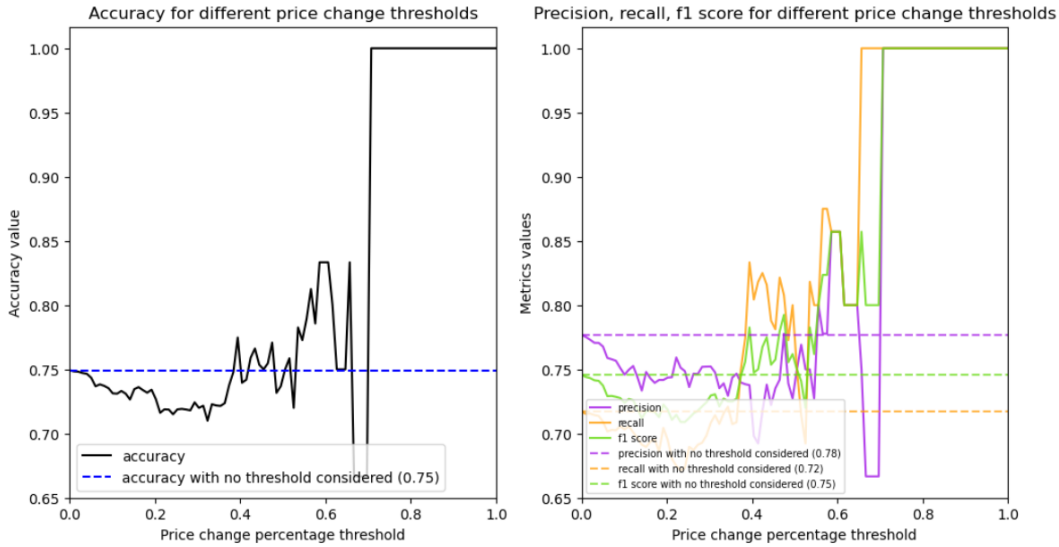


Figure 3.8 Binary classification; accuracy (left) and precision, recall, f1 score (right)

Even though the metrics approach 100 % as the *threshold* value increases, it is not wise to consider the highest *threshold* values. This essentially means that no action should be taken because the prediction falls into the category **0**. Since the ultimate goal of the task is to help decide whether to buy or sell currencies, the models should be able to signal a clear increase or decrease prediction.

Chapter 4
Conclusion

This thesis aimed to implement and evaluate the performance of Transformer architectures for predicting short-term financial time series in the Forex market. Specifically, two models were tested: the original Transformer and the Autoformer. Both models were built from scratch in Python using the TensorFlow framework and evaluated on historical Forex data.

The goal of creating a custom implementation of transformer models was achieved; the models were successfully implemented, trained, and evaluated.

After the training, the models were evaluated. The Transformer model yielded better overall results than the Autoformer on stationary data but did not beat the two baseline models when it worked with non-stationary data.

The performed experiment shows that the Autoformer is better suited for non-stationary data, possibly due to its Auto-correlation mechanism and its series decomposition method.

The transformer model achieved the best results when it predicted changes in currency pair prices. It achieved an accuracy of over 75 %, suggesting that the Transformer, or other transformer-based models, can solve time series forecasting tasks.

Additional experiments with the predictions performed at the end of the experiments suggest that more creative methods can be tried to achieve better results, which recommends further research. Additionally, different pre-processing approaches can be utilized to improve the models' performance, such as smoothing techniques or more combinations of technical indicators, to provide the models with more information to work with.

The next steps could also lead to enhanced experimentation with models' hyperparameters to make them more or less robust, balancing a trade-off between computational needs and the quality of the predictions. Another new approach could also involve using language models on news from the financial world, capturing sentiment from them, thus adding new information to the models.

Bibliography

1. KLU.AI. *Glossary: Accuracy, Precision, Recall, F1* [https://klu.ai/glossary/accuracy-precision-recall-f1]. 2024. Accessed: 2024-09-29.
2. PROJECTPRO. *Explain Accuracy, Precision, Recall, and F-beta Score*. 2022. Available also from: https://www.projectpro.io/recipes/explain-accuracy-precision-recall-and-f-beta-score. Accessed: 2024-09-29.
3. KUNDU, Rohit. *F1 Score in Machine Learning: Intro & Calculation*. 2022. Available also from: https://www.v7labs.com/blog/f1-score-guide. Accessed: 2024-09-29.
4. WISDOM, Banso D. *Understanding the Confusion Matrix*. 2019. Available also from: https://dev.to/overrideveloper/understanding-the-confusion-matrix-2dk8. Accessed: 2024-10-02.
5. *scikit-learn: Machine Learning in Python* [https://scikit-learn.org/stable/]. 2024. Accessed: 2024-09-30.
6. VAŠATA, Daniel. *BI-ML2.21 přednáška 7,* [lecture]. Prague, 2023-05-04; accessed 2024-10-04. Available also from: https://online.fit.cvut.cz/zaznam/B222/bi-ml2.21_pre_2023-04-05.html.
7. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
8. DU, Ke-Lin; LEUNG, Chi-Sing; MOW, Wai Ho; SWAMY, M. N. S. Perceptron: Learning, Generalization, Model Selection, Fault Tolerance, and Role in the Deep Learning Era. *Mathematics*. 2022, vol. 10, no. 24. ISSN 2227-7390. Available from DOI: 10.3390/math10244730.
9. NIELSEN, Michael A. *Neural Networks and Deep Learning,* [online]. Determination Press, 2018. Available also from: http://neuralnetworksanddeeplearning.com/.
10. LIANG, XingLong; XU, Jun. Biased ReLU neural networks. *Neurocomputing*. 2021, vol. 423, pp. 71–79. ISSN 0925-2312. Available from DOI: https://doi.org/10.1016/j.neucom.2020.09.050.
11. TESMA. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology (IJEAST)*. 2020, pp. 310–316. Available also from: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf. Accessed: 2024-09-30.
12. RINK, Towards Data Science Konstantin. *Time Series Forecast Error Metrics You Should Know* [https://towardsdatascience.com/time-series-forecast-error-metrics-you-should-know-cc88b8c67f27]. 2021. Accessed: 2024-09-30.
13. VAŠATA, Daniel. *BI-ML2.21 přednáška 9*. 2023. Available also from: https://courses.fit.cvut.cz/BI-ML2/@B222/lectures/files/BI-ML2-09-cs-slides.pdf.

14. KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Available from arXiv: 1412.6980 [cs.LG].
15. GRAVES, Alex. *Generating Sequences With Recurrent Neural Networks*. 2014. Available from arXiv: 1308.0850 [cs.NE].
16. WU, Neo; GREEN, Bradley; BEN, Xue; O'BANION, Shawn. *Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case*. 2020. Available from arXiv: 2001.08317 [cs.LG].
17. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. 2023. Available from arXiv: 1706.03762 [cs.CL].
18. ALAMMAR, Jay. *The Illustrated Transformer*. 2018. Available also from: https://jalammar.github.io/illustrated-transformer/. Accessed: 2024-10-20.
19. BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. Available from arXiv: 1409.0473 [cs.CL].
20. WANG, Zian (Andy). Visualizing and Explaining Transformer Models From the Ground Up. *Deepgram*. 2023. Available also from: https://deepgram.com/learn/visualizing-and-explaining-transformer-models-from-the-ground-up. Published January 19, 2023, Updated June 13, 2024.
21. VIG, Jesse. *BertViz: Visualize Attention in NLP Models (BERT, GPT2, BART, etc.)* 2024. Available also from: https://github.com/jessevig/bertviz. Accessed: October 19, 2024.
22. SANH, Victor; DEBUT, Lysandre; CHAUMOND, Julien; WOLF, Thomas. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2020. Available from arXiv: 1910.01108 [cs.CL].
23. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. Available from arXiv: 1810.04805 [cs.CL].
24. KARPATHY, Andrej. *Let's build GPT: from scratch, in code, spelled out.* [video]. Youtube [online], 2023 [accessed 2024-10-04]. Available also from: https://www.youtube.com/watch?v=kCc8FmEb1nY.
25. BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E. *Layer Normalization*. 2016. Available from arXiv: 1607.06450 [stat.ML].
26. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Available from arXiv: 1502.03167 [cs.LG].
27. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, no. 56, pp. 1929–1958. Available also from: http://jmlr.org/papers/v15/srivastava14a.html.
28. BOX, George E. P.; JENKINS, Gwilym M.; REINSEL, Gregory C. *Time series analysis: forecasting and control*. 4th. Englewood Cliffs: Prentice Hall, 2008. ISBN 9780470272848;0470272848;
29. TEAM, The Matplotlib Development. *Matplotlib: Visualization with Python* [https://matplotlib.org/]. 2024. Accessed: 2024-10-02.
30. OPENAI. *ChatGPT Official Website* [https://chatgpt.com/]. 2024. Accessed: 2024-10-02.
31. BROWNLEE, Jason. *Introduction to time series forecasting with python: how to prepare data and develop models to predict the future*. Machine Learning Mastery, 2017.

32. FAN, Jianqing.; YAO, Qiwei. *Nonlinear time series : nonparametric and parametric methods*. Nonlinear time series : nonparametric and parametric methods. New York: Springer, 2003. Springer series in statistics. ISBN 0387951709.

33. WEN, Qingsong; ZHOU, Tian; ZHANG, Chaoli; CHEN, Weiqi; MA, Ziqing; YAN, Junchi; SUN, Liang. *Transformers in Time Series: A Survey*. 2023. Available from arXiv: 2202.07125 [cs.LG].

34. WU, Haixu; XU, Jiehui; WANG, Jianmin; LONG, Mingsheng. *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*. 2022. Available from arXiv: 2106.13008 [cs.LG].

35. ZHOU, Tian; MA, Ziqing; WEN, Qingsong; WANG, Xue; SUN, Liang; JIN, Rong. *FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting*. 2022. Available from arXiv: 2201.12740 [cs.LG].

36. NIE, Xingqing; XIAOGEN, Zhou; LI, Zhiqiang; WANG, Luoyan; LIN, Xingtao; TONG, Tong. *LogTrans: Providing Efficient Local-Global Fusion with Transformer and CNN Parallel Network for Biomedical Image Segmentation*. 2022. Available from DOI: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00128.

37. *Daily Market Summary* [https://www.nasdaqtrader.com/Trader.aspx?id=DailyMarketSummary]. 2024. Accessed: 2024-10-14.

38. INTERNATIONAL SETTLEMENTS, Bank for. *OTC foreign exchange turnover in April 2022* [https://www.bis.org/statistics/rpfx22_fx.htm]. 2022. Accessed: 2024-10-14.

39. GANTI, Akhilesh. *Foreign Exchange Market: How It Works, History, and Pros and Cons*. 2024. Available also from: https://www.investopedia.com/terms/forex/f/foreign-exchange-markets.asp. Accessed: 2024-10-17.

40. SCOTT, Gordon. *Forex Market Hours: Can You Trade 7 Days a Week?* [https://www.investopedia.com/terms/forex/f/forex-market-trading-hours.asp]. 2024. Accessed: 2024-04-15.

41. SCOTT, Gordon. *Largest stock exchange operators worldwide as of December 2023, by market capitalization of listed companies* [https://www.statista.com/statistics/270126/largest-stock-exchange-operators-by-market-capitalization-of-listed-companies/]. 2024. Accessed: 2024-10-14.

42. SLAVA LOZA. *The Most Traded Currency Pairs in Forex (2024 Edition)*. 2024. Available also from: https://fxssi.com/the-most-traded-currency-pairs. Accessed: 2024-10-10.

43. SECURITIES, Ventura. *OHLC Meaning: How Does It Help You Trade?* 2024. Available also from: https://www.venturasecurities.com/blog/ohlc-meaning-how-does-it-help-you-trade/. Accessed: 2024-10-17.

44. KAUFMAN, Perry J. *Trading Systems and Methods*. 6;6th;Sixth; Newark: Wiley, 2019. ISBN 9781119605393;1119605393;1119605350;9781119605355;

45. OPPENHEIM, A.V.; WILLSKY, A.S.; NAWAB, S.H. *Signals & Systems*. 2nd ed. Prentice-Hall International, 1997. ISBN 9780136511755.

46. AASVIK, Mads. *Arduino Tutorial: Simple High-pass, Band-pass and Band-stop Filtering*. 2016. Available also from: https://www.norwegiancreations.com/2016/03/arduino-tutorial-simple-high-pass-band-pass-and-band-stop-filtering/. Accessed: 2024-10-13.

47. PRESS, William H.; TEUKOLSKY, Saul A.; VETTERLING, William T.; FLANNERY, Brian P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. Cambridge University Press, 2007. ISBN 0521880688. Available also from: http://www.amazon.com/Numerical-Recipes-3rd-Scientific-Computing/dp/0521880688/ref=sr_1_1?ie=UTF8&s=books&qid=1280322496&sr=8-1.

48. COOLEY, James; TUKEY, John. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*. 1965, vol. 19, no. 90, pp. 297–301.

49. SCIPY DEVELOPERS. *scipy.signal.filtfilt* [https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html]. 2024. Available also from: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html. Accessed: 2024-10-17.

50. GWINN, C. R.; JOHNSON, M. D. Noise and Signal for Spectra of Intermittent Noiselike Emission. *The Astrophysical journal*. 2011, vol. 733, no. 1, 51–jQuery1323908476286='48'. ISBN 0004-637X.

51. JENSEN, G. *Part 2: Convolution and Cross-Correlation* [YouTube video]. 2023. Available also from: https://www.youtube.com/watch?v=MQm6ZP1F6ms. Accessed: 2024-10-23.

52. SMITH, Steven W. *The scientist and engineer's guide to digital signal processing*. USA: California Technical Publishing, 1997. ISBN 0966017633.

53. MUHAMMAD, Tashreef; AFTAB, Anika Bintee; IBRAHIM, Muhammad; AHSAN, Md. Mainul; MUHU, Maishameem Meherin; KHAN, Shahidul Islam; ALAM, Mohammad Shafiul. Transformer-Based Deep Learning Model for Stock Price Prediction: A Case Study on Bangladesh Stock Market. *International Journal of Computational Intelligence and Applications*. 2023, vol. 22, no. 03. ISSN 1757-5885. Available from DOI: 10.1142/s146902682350013x.

54. KAZEMI, Seyed Mehran; GOEL, Rishab; EGHBALI, Sepehr; RAMANAN, Janahan; SAHOTA, Jaspreet; THAKUR, Sanjay; WU, Stella; SMYTH, Cathal; POUPART, Pascal; BRUBAKER, Marcus. *Time2Vec: Learning a Vector Representation of Time*. 2019. Available from arXiv: 1907.05321 [cs.LG].

55. GRAMER, ARVID AND DANIELSSON, SIMON. *Predicting Forex Rates using Sentiment Analysis on Financial Articles*. 2023. Student Paper.

56. FAMA, Eugene F. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*. 1970, vol. 25, no. 2, pp. 383–417. Available from DOI: 10.2307/2325486.

57. MALO, Pekka; SINHA, Ankur; TAKALA, Pyry; KORHONEN, Pekka; WALLENIUS, Jyrki. *Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts*. 2013. Available from arXiv: 1307.5336 [cs.CL].

58. BELTAGY, Iz; PETERS, Matthew E.; COHAN, Arman. *Longformer: The Long-Document Transformer*. 2020. Available from arXiv: 2004.05150 [cs.CL].

59. ANBAEE FARIMANI, Saeede; VAFAEI JAHAN, Majid; MILANI FARD, Amin; TABBAKH, Seyed Reza Kamel. Investigating the informativeness of technical indicators and news sentiment in financial market price prediction. *Knowledge-Based Systems*. 2022, vol. 247, p. 108742. ISSN 0950-7051. Available from DOI: https://doi.org/10.1016/j.knosys.2022.108742.

60. MALIBARI, Nadeem; KATIB, Iyad; MEHMOOD, Rashid. Predicting Stock Closing Prices in Emerging Markets with Transformer Neural Networks: The Saudi Stock Exchange Case. *International Journal of Advanced Computer Science and Applications*. 2021, vol. 12, no. 12. Available from DOI: 10.14569/IJACSA.2021.01212106.

61. DOSOVITSKIY, Alexey; BEYER, Lucas; KOLESNIKOV, Alexander; WEISSENBORN, Dirk; ZHAI, Xiaohua; UNTERTHINER, Thomas; DEHGHANI, Mostafa; MINDERER, Matthias; HEIGOLD, Georg; GELLY, Sylvain; USZKOREIT, Jakob; HOULSBY, Neil. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. Available from arXiv: 2010.11929 [cs.CV].

62. LTD., Forex Software. *Historical Forex Data*. 2024. Available also from: https://forexsb.com/historical-forex-data. Accessed: October 19, 2024.

63. TENSORFLOW. *tf.keras.Layer*. 2024. Available also from: https://www.tensorflow.org/api_docs/python/tf/keras/Layer. Accessed: 2024-10-21.

64. TENSORFLOW. *tf.keras.layers.LayerNormalization*. 2024. Available also from: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LayerNormalization. Accessed: 2024-10-21.

65. ALDOSARI, Mohammed; MILLER, John. On Transformer Autoregressive Decoding for Multivariate Time Series Forecasting. In: 2023, pp. 423–428. Available from DOI: 10.14428/esann/2023.ES2023-171.

66. TENSORFLOW. *tf.keras.layers.AveragePooling1D*. 2024. Available also from: https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D. Accessed: 2024-10-22.

67. TENSORFLOW. *tf.math.top_k*. 2023. Available also from: https://www.tensorflow.org/api_docs/python/tf/math/top_k. Accessed: 2024-10-23.

Attachments

- `readme.md` — medium description
- `thesis_showcase.ipynb` — commented solution procedure
- `autoformer_predict_price_move.ipynb` — model training
- `autoformer_predict_price.ipynb` — model training
- `transformer_in_seq.ipynb` — model training
- `transformer_out_seq.ipynb` — model training
- `transformer_predict_price.ipynb` — model training
- `simple_baseline.ipynb` — baseline model implementation
- `src`
  - `autoformer.py` — autoformer implementation
  - `vanilla_transformer.py` — transformer implementation
  - `load_data.py` — class for loading data
  - `train_model.py` — class for training models
  - `visualize_data.py` — class for data visualization
- `imports.py` — imported libraries