University of St.Gallen

Deep Neural Network Construction for Enhanced Multi-Asset Time Series and Cross-Sectional Momentum Strategies

Master Thesis

Supervisor:

Prof. Dr. David Preinerstorfer

Swiss Institute for Empirical Economic Research

Co-Supervisor:

Prof. Dr. Alexander Braun

Institute of Insurance Economics

Author:

Niklas Leander Kampe

Weiherweg 3, 8610 Uster

Master in Quantitative Economics & Finance (MiQE/F)

16-611-618

University of St. Gallen

November 20th 2023

Foreword

Most of the illustrations in this thesis (if not explicitly stated contrarily) have been designed by the author from the ground up based on existing and widely adapted generalized visualizations. This ensures, on the one hand, that the architectures can be visualized in a uniform format and design across different models as well as to allow direct visual comparability. In addition, the mathematical derivations of the underlying deep learning models might use different notations compared to the corresponding sources, which intends to ensure a unique uniformed derivation across different underlying model architectures as well as comparability across models and their components within the thesis. The corresponding proposed transformer-based CSMOM and TSMOM models have been implemented through TensorFlow (Abadi et al., 2015), whereby the model architectures have been implemented from the ground up according to the model breakdowns in section 3 with limited utilization of pre-defined TensforFlow components but rather custom layer-specific implementations in order to allow for flexibility and adjustments of the underlying model components. Due to limited computing resources, all experiments were conducted on a NVIDIA GeForce GTX 1050 GPU and 32GB of RAM memory, leading to model training times of multiple hours up to a few days.

Abstract

As one of the most applied portfolio strategies in practice, cross-sectional momentum (CSMOM) and time series momentum (TSMOM) serve as central concepts contradicting the efficient market hypothesis while obeying still unexplained market anomalies. While these strategies have been enhanced with deterministic model extensions with significant empirical out-performances over the last centuries, upcoming deep-learning approaches serve as a foundation for further optimizing the asset ranking for CSMOM and trend estimation as well as position sizing for TSMOM. This thesis focuses on deriving deep learning models from the ground up based on generalized CSMOM and TSMOM strategy frameworks as well as vanilla deep learning model architectures, with the goal of building momentum-specific deep learning models. Thereby, transformer neural networks serve as the foundation for the proposed *CSMOM Transformer Ranker* and *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* based on the concept of self-attentive (re-)ranking by Pobrotyn et al. (2021), as well as the proposed sharpe-optimizing *TSMOM Encoder-Decoder Transformer* and *TSMOM Decoder-Only Transformer* based on empirical approaches on time series applications and the suggestion by Li et al. (2020) that the decoder-part might be sufficient for such applications. In addition, other deep learning models such as MLPs, RNNs, LSTMs, and abstractions of those, are utilized next to the original CSMOM and TSMOM models as benchmarks in order to empirically back-test the proposed transformer-based strategies on a multi-asset data set with 50 continuous future contracts between 2000 and 2022. The empirical back-testing results based on a monthly re-balancing frequency show that in the course of the CSMOM back-testing, the learning-to-rank (LTR) MLP and the deterministic MACD-based benchmark strategy with sharpe ratios of 1.0248 and 1.0007, respectively, dominate, while the proposed transformer-based models outperform the other benchmark strategies with sharpe ratios of 0.7997 and 0.6076, respectively. In the course of the TSMOM strategies, there is a global out-performance of the encoder-decoder transformer with a sharpe ratio of 0.8633, while the decoder-only transformer with a sharpe ratio of 0.4497 even lags behind other benchmark strategies, which in turn underlines the inherent design of the transformer encoder-decoder architecture for time series applications. In the course of comparison to existing research findings by Poh et al. (2022) and Wood et al. (2021), a likely reason for the mixed performance of the proposed transformer-based CSMOM and TSMOM models are the problems related to data scarcity, whereby due to the monthly re-balance frequency and the relatively small universe of 50 continuous futures contracts, there is no sufficiently large training set, which means the in-sample training cannot generalize which leads to an overfitting bias and was already observed based on the training, validation and testing losses.

Table of Contents

List of Figures **VI**

List of Tables **VI**

List of Abbreviations **VIII**

1 Introduction **1**

2 Momentum Strategies **3**
2.1 Efficient Market Hypothesis & Momentum Anomalies . . . 3
2.2 Cross-Sectional Momentum (CSMOM) . . . 4
2.3 Time Series Momentum (TSMOM) . . . 8

3 Deep Learning Architectures **11**
3.1 Key Deep Learning Models . . . 12
3.1.1 Multi-Layer Perceptron (MLP) . . . 12
3.1.2 Recurrent Neural Network (RNN) . . . 22
3.1.3 Long Short-Term Memory Neural Network (LSTM) . . . 29
3.2 Transformer Neural Network . . . 38
3.2.1 Motivation . . . 38
3.2.2 General Model Architecture . . . 39
3.2.2.1 Input Embedding & Positional Encoding (PE) . . . 41
3.2.2.2 (Masked) Multi-Head Self-Attention ((M)MHA) . . . 42
3.2.2.3 Fully-Connected Feed-Forward Network (FFN) . . . 46
3.2.2.4 Residual Connection (Add) & Layer Normalization (Norm) . . . 47
3.2.2.5 Encoder Output . . . 48
3.2.2.6 Decoder Output . . . 48
3.2.3 Application to Numerical Use Cases . . . 49

4 OK Transformer-Based Multi-Asset CSMOM & TSMOM Architectures **51**
4.1 OK Generalized CSMOM Strategy Framework . . . 51
4.2 OK Deep Learning-Enhanced Multi-Asset CSMOM Model . . . 53
4.2.1 OK CSMOM Transformer Ranker . . . 53
4.2.2 OK CSMOM ListNet Pre-Ranker Transformer Re-Ranker . . . 57
4.2.3 OK CSMOM Strategy Assumptions & Requirements . . . 61
4.2.4 OK CSMOM Strategy Benchmark Models . . . 63
4.3 OK Generalized TSMOM Strategy Framework . . . 68
4.4 OK Deep Learning-Enhanced Multi-Asset TSMOM Model . . . 70
4.4.1 OK TSMOM Encoder-Decoder Transformer . . . 70
4.4.2 OK TSMOM Decoder-Only Transformer . . . 75
4.4.3 OK TSMOM Strategy Assumptions & Requirements . . . 78
4.4.4 OK TSMOM Strategy Benchmark Models . . . 79

5 OK Multi-Asset Data Set **83**
5.1 OK Data Generating Process (DGP) 83
5.2 OK Descriptive Statistics & Properties 84

6 OK Empirical Multi-Asset CSMOM & TSMOM Strategy Back-Testing **87**
6.1 OK Back-Testing Framework & Performance Measures 87
6.2 OK Model Features 89
6.3 OK Model Training, Validation & Testing 90
6.4 Model Back-Testing & Evaluation 99

7 Conclusion & Outlook **106**

List of References **VII**

List of Figures

1 Activation Functions . . . 14
2 MLP Architecture . . . 15
3 Activation Functions & Derivatives . . . 18
4 Optimal Model Complexity, Underfitting & Overfitting . . . 20
5 Recurrent Neural Network Architecture . . . 24
6 Recurrent Neural Network Types . . . 25
7 LSTM Memory Block . . . 30
8 LSTM Architecture . . . 33
9 Transformer Architecture . . . 40
10 Transformer Queries, Keys, Values . . . 43
11 Transformer Attention Scores . . . 44
12 Transformer Scaled Attention Scores . . . 44
13 Transformer Attention Weights . . . 44
14 Transformer Self-Attention . . . 44
15 Transformer Multi-Head Self-Attention . . . 45
16 Transformer MHA . . . 45
17 Transformer MMHA . . . 46
18 Transformer FFN . . . 47
19 CSMOM Transformer Ranker . . . 56
20 CSMOM ListNet Pre-Ranker Transformer Re-Ranker . . . 60
21 TSMOM Encoder-Decoder Transformer . . . 74
22 TSMOM Decoder-Only Transformer . . . 77
23 Multi-Asset Data Set - Cumulative Asset Returns . . . 87
24 Train-Validation-Test Split . . . 92
25 Model Back-Testing - CSMOM Strategy Cumulative (Log) Returns . . . 101
26 Model Back-Testing - TSMOM Strategy Cumulative (Log) Returns . . . 103

List of Tables

1 CSMOM Strategy Parameters - Portfolio Size . . . 61
2 CSMOM Strategy Parameters - Re-balance Frequency . . . 62
3 CSMOM Strategy Parameters - Target Volatility . . . 62
4 TSMOM Strategy Parameters - Realized Volatility . . . 63
5 TSMOM Strategy Parameters - Re-balance Frequency . . . 78
6 TSMOM Strategy Parameters - Look-Back Period . . . 78
7 TSMOM Strategy Parameters - Target Volatility . . . 79
8 TSMOM Strategy Parameters - Realized Volatility . . . 79
9 Multi-Asset Data Set - Overview . . . 85

10 Multi-Asset Data Set - Summary Statistics 86
11 Back-Testing Performance Measures 88
12 CSMOM Input Features 89
13 TSMOM Input Features 90
14 Model Input - CSMOM Transformer Ranker & ListNet Pre-Ranker Transformer Re-Ranker 92
15 Model Input - TSMOM Encoder-Decoder Transformer 93
16 Model Input - TSMOM Decoder-Only Transformer 94
17 Fixed & Hyperparameters - Benchmark CSMOM & TSMOM Models 96
18 Fixed & Hyperparameters - Proposed CSMOM & TSMOM Models 98
19 Model Back-Testing - CSMOM Model Evaluation 100
20 Model Back-Testing - TSMOM Model Evaluation 102

List of Abbreviations

BTT Back-Propagation Through Time

CEC Constant Error Carrousel

CSMOM Cross-Sectional Momentum

EMH Efficient Market Hypothesis

LSTM Long Short-Term Memory

MSE Mean Squared Error

RNN Recurrent Neural Network

RTRL Real-Time Recurrent Learning

TSMOM Time Series Momentum

1 Introduction

Since the initial examination of fundamental risk premia in the context of (multivariate) cross-sectional momentum (CSMOM) and (univariate) time series momentum (TSMOM) strategies by Jegadeesh & Titman (1993) and Moskowitz et al. (2012), those systematic portfolio strategies have seen a high degree of application in research and the industry. While the fundamental models as well as their further extensions of the last few years focus on the explicit definition of the asset ranking rule for CSMOM and the trend estimation as well as the position sizing rule for TSMOM, the latest research approaches in the field of deep learning offer further possibilities for the static definition components to be replaced by data-driven, complex model structures in order to break up any limitations of deterministic model architectures.

The main differences between cross-sectional momentum and time series momentum strategies are already apparent from the term definition: While cross-sectional momentum ranks two or more assets based on their relative performance to each other in the cross-section on top of which it infers corresponding portfolio weights, time series momentum strategies derive equivalent inference from the absolute performance of a single asset from its own past time series and past performance. Based on these theoretical frameworks, numerous scientific studies show that excess returns and thus significant profitability have been systematically achieved since the 1920s and continue to this day. Hence, there is a high density of literature that underlines the results for different time periods, geographic markets as well as asset classes and has always been promoted, among other things, by the contrary results to Fama's (1970) efficient market hypothesis (EMH). Hence, it doesn't seem surprising that Fama (1998) defined the momentum effects as one of the central, unexplained market anomalies and thus admitted the contrast to his own EMH.

In the course of optimizing the asset ranking for CSMOM strategies as well as the trend estimation and position sizing for TSMOM strategies, empirically valid and robust research approaches have emerged in recent years, ranging from enhanced deterministic up to machine learning-based approaches. Nevertheless, the recent research developments in the area of deep learning offer more advanced model architectures which can be adapted to the general CSMOM and TSMOM strategy frameworks such that they can potentially generate significant out-performance compared to benchmark strategies. In addition to multi-layer perceptrons (MLP), recurrent neural networks (RRN) and long short-term memory neural networks (LSTM), as well as adaptations of those models that have already been used empirically in different settings with regard to CSMOM and TSMOM applications, transformer neural networks are currently forming novel model components that achieve strong empirical results across various numerical applications. With regard to CSMOM and TSMOM strategies, there is central research work on deep learning-enhanced strategies by Poh et al. (2020) with reference to classic learning-to-rank CSMOM approaches, also by Poh et al. (2022) regarding self-attentive learning-to-rank CSMOM approaches, as well as Lim et al. (2020) with reference to deep learning-based TSMOM approaches and Wood et el. (2022) with reference to transformer-based TSMOM approaches. Nevertheless, it is shown, on the one hand, not only multi-asset approaches are tested empirically and, on another hand, very high re-balance frequencies such as daily re-balancing are usually used, which in turn calls into

question the practical applicability in terms of transaction costs and other limitations. Overall, in the course of research into momentum strategies and their empirical risk premium in various market environments, the focus is primarily on equity markets, although isolated fundamental research findings have shown that empirical momentum factors exist for other asset classes as well (Bird et al., 2017).

Based on the existing research work on deep learning-optimized CSMOM and TSMOM strategies under the construction and empirical application of deep neural networks, the aim of this thesis is to empirically back-test CSMOM- and TSMOM-specific neural network architectures, using existing vanilla models that are mathematically modified and designed from the ground up with regard to the specific strategy requirements. For this purpose, the first part describes the basic frameworks of CSMOM and TSMOM strategies according to Jegadeesh & Titman (1993) and Moskowitz et al. (2012), respectively, while the underlying strategy premia are identified. The second part introduces key deep learning models, their components and mathematical architectures, which then serve as the foundation for the third part regarding the construction of CSMOM and TSMOM-specific deep learning models with a specific focus on transformer neural networks, whereby further momentum-adapted deep learning models are built as benchmark models. Due to the key property of permutation-equivariance, transformers are generally applicable for both cross-sectional and time series use cases, whereby the vanilla architectures for both cases need to be adjusted accordingly. Hence, generalized CSMOM and TSMOM strategy frameworks are first set up, on top of which strategy-specific deep learning momentum models are derived and constructed from the ground up. In the following part, the multi-asset data set of future contracts is introduced and adapted with regard to the generation of a continuous price series, after which the descriptive statistics are examined. In the last part, the previously derived proposed and benchmark multi-asset CSMOM and TSMOM strategies are empirically examined as part of long-term empirical back-testing using the multi-asset data set of continuous future contracts on equities, fixed income, foreign exchange as well as commodities between 1999 and 2023. Within there, the entire train-validation-testing setup including the back-testing framework, the performance metrics and the underlying model (hyper) parameters are explained in detail.

The contribution of this thesis to existing CSMOM and TSMOM research lies in several areas. On the one hand, there are existing deep learning approaches for CSMOM and TSMOM strategies which are not necessarily applied to multi-asset portfolios nor under equivalent testing frameworks, with high re-balance frequencies in particular. In addition, state-of-the-art deep learning research developments lead to new possible approaches, especially with regard to transformer networks, which have not yet found wide empirical research applications in momentum strategies. Therefore, it is intended to represent a mathematically-driven research approach that develops and empirically back-tests existing and new deep learning approaches for CSMOM and TSMOM strategies in a multi-asset setting as well as lower re-balance frequencies. Overall, it also shows that the density of research work on momentum strategies is still relatively thin, while other research fields in the quantitative finance area are considered stronger. Lastly, momentum strategies still show great relevance in practice today, which can be optimized and validated primarily through research-based approaches.

2 Momentum Strategies

In the course of the model extensions of cross-sectional and time series momentum strategies through new deep learning approaches, it is first necessary to elaborate existing model definitions and setups. For this purpose, an analysis of the efficient market hypothesis and the associated momentum anomalies as well as the underlying momentum risk premium follows. In addition, the cross-sectional and time series momentum strategies are explained in detail which is used to define similarities, differences and central model limitations. Overall, the status quo of both models shall be derived and used as a foundation for the subsequent development of new model architectures.

2.1 Efficient Market Hypothesis & Momentum Anomalies

The empirical investigation of asset prices and their underlying dynamics is still one of the central questions in financial market research today. Due to the fact that the question does not seem to have a definitive answer, financial market models that have been developed over time serve as important generalizations and approachable concepts in order to use central assumptions to develop further conclusions. In this context, the Efficient Market Hypothesis (EMH) has established itself as one of the most important concepts for institutionalizing the incompletely explainable dynamics of the financial markets. Among other things, Fama (1970), as one of the main founders of the Efficient Market Hypothesis, shows that a distinction must be made between the weak, the semi-strong and the strong form, with those differing in terms of the degree of efficiency in terms of information access and thus in terms of the pricing of that information. Until the end of the 20^{th} century, however, there was a general view that markets have a high basic degree of efficiency and that new information is priced in by the market without delay (Malkiel, 2003, p.59). This implies that any market analysis does not generate any excess returns because there are no informational benefits. This assumption is based on the random walk assumption, which states that given prices reflect all available information and that any price changes are therefore subject to randomness, with independence across the return series (Fama, 1970, p.386). Formally, this can be defined as follows (Fama, 1970, p.387):

$$f(r_{j,t+1}|\mathcal{F}_t) = f(r_{j,t+1}) \tag{1}$$

where $\mathcal{F}$ represents the information set available at time t. Formula 1 thus defines that the conditional probability distribution of the future return as being identical to the marginal probability distribution of the future return and hence that the price change in $t+1$ is independent of the available information set in t (Fama, 1970, p.387). That strong definition implies one of the key results of the Efficient Market Hypothesis, which states that the expected return is independent of the information set at time t:

$$E[r_{j,t+1}|\mathcal{F}_t] = E[r_{j,t+1}] \tag{2}$$

where that definition is only limited to the expected value of the probability distribution and is accordingly implied by the formula 1. On the basis of these theoretical assumptions, the view

that the Efficient Market Hypothesis has weaknesses in the course of this definition and that asset prices are predictable to a certain extent increasingly manifested itself at the beginning of the 20^{th} century (Malkiel, 2003, p.60). In the course of this, behavioral and psychological theories in particular in connection with financial market dynamics received increasing attention, which probably represent a suitable basis for explaining certain dynamics (Malkiel, 2003, p.60). Thus, the focus shifted from the extended application of existing models to the exploration and explainability of asset pricing anomalies against the efficient market hypothesis.

One of the most important empirical asset pricing anomalies that cannot yet be fully explained are momentum effects, which generally define observable and significantly robust short-term serial correlations between successive returns and thus represent trend-like patterns in return series that persistent over certain periods of time (Malkiel, 2003, p.61). Fama (1998) already described momentum anomalies as one of the central challenges of existing rational asset pricing models, approximately 30 years after publishing its own theory on the efficient market hypothesis. The first fundamental research work goes back in particular to Jegadeesh and Titman (1993), who were able to achieve systematic, significant excess returns through trading strategies over a period of almost 25 years. Hence, based on the basic assumption that systematic excess returns can be achieved from the cross-section of asset returns, this finding represents a clear anomaly to the assumptions of the efficient market hypothesis from formula 2 and 1, since the conditional probability distribution and the conditional expected value of the future return in $t+1$ are no longer independent of the information set $\mathcal{F}$ at time t. In addition, the research findings by Moskowitz et al. (2012) represents another peculiarity that, in contrast to Jegadeesh and Titman, systematically prove excess returns in single asset price time series. In addition, the results show robust significance across several asset classes and an observation period of 25 years, in particular equity indices, currencies, commodities and bonds (Moskowitz et al., 2012, p.228). Thus, the dynamics surrounding time series momentum strategies also represent an empirical contrast to the random walk hypothesis from the formula 1 and hence the (strong-form) efficient market hypothesis, in that the available set of information about the asset price time series at time t can be exploited to achieve significant and robust excess returns based on return predictability (Moskowitz et al., 2012, p.229).

Schlussendlich gilt es zu prüfen,

2.2 Cross-Sectional Momentum (CSMOM)

The scientific research of momentum effects in the cross-section of asset returns goes back to the fundamental statements by Jegadeesh and Titman (1993). Based on the first findings of trend-based excess returns by De Bondt and Thaler (1987), who base their explanations on the over- and underreaction to the availability of new information and find significant higher returns for assets performing bad over the last three to five years compared to assets performing well over the same period, the theory was devoted to generalized cross-sectional momentum effects of shorter return reversal periods and further empirical investigations. While De Bondt's and Thaler's (1987) contrarian strategy suggests to buy relatively weak assets in the past and to sell relatively strong assets in the past under the observation period of three to five years, the general framework

of cross-sectional momentum strategies are based on the concept of buying relatively strong and selling relatively weak assets in the cross-section of asset prices under a shorter observation period of three to twelve months, while both leading to systematic excess returns (Jegadeesh and Titman, 1993, p.89). With regard to momentum effects in the cross-section, the underlying behavioral theory describes that initially private information only reaches the public and markets in granular steps, which causes an initial underreaction of asset prices (Hong and Stein, 1999, p.2146-2147). Hence, positive or negative return effects also occur granular over a certain period of time, which leads to persistent return trends, so-called momentum. In addition, there is the observation of post-event drifts, whereby after the reaction to directly publicly available information, further price movements occur that follow the same direction as the initial reaction (Hong and Stein, 1999, p.2146-2147). This underscores the theory of initial underreaction, which is the basis of sustained positive or negative returns over a period of time. With regard to reversals, it is helpful to refer back to the fundamental empirical insights of De Bondt and Thaler (1985) in context of contrarian strategies, who show a negative correlation of asset returns, with positive (negative) returns over three to five years leading to negative (positive) returns over the following three to five years. With reference to the classic theory of fundamental asset prices, and hence the underlying foundation of value investing strategies, which describes returns as cyclical fluctuations around fundamental values, the basis of return reversals emerges, according to which a reverse structure occurs after a certain temporal persistence of positive or negative returns (Hong and Stein, 1999, p.2147). Based on this underlying behavioural theory, Jegadeesh's and Titman's work intended to prove empirically that such behavioural market reactions to information exist in the short run and hence that trading strategies with systematically excess returns based on momentum effects arise in the markets.

The research findings of Jegadeesh and Titman (1993, p.89) show that relative strength portfolios, in which relatively strong assets are bought in the cross-section and relatively weak assets are sold, have predictable price developments over the first three to twelve months of the holding period. This explicitly implies that relatively strong assets maintain their performance relative to the other assets over that period of time, with the same applying to the relatively weak performance of weak assets. In addition, it can be seen that the predictability in the form of momentum decreases after about twelve months and negative abnormal returns can be recorded up to a period of 31 months after portfolio formation (Jegadeesh and Titman, 1993, p.66). This in turn is in line with the theory of contrarian strategies around De Bondt and Thaler, whereby the value of the assets returns to the fundamental value in the medium to long term (Jegadeesh and Titman, 1993, p.66). Furthermore, the specific analysis around the effects of earnings announcements shows that past relatively strong stocks around the earnings announcements and up to seven months later achieve higher returns, with an inverse effect compared to the previously relatively weak stocks in the following 13 months occurs (Jegadeesh and Titman, 1993, p.67).

In the course of the empirical evaluation of the hypotheses and results explained before, Jegadeesh and Titman (1993, p.68) examined different relative strength portfolio strategies which focus on buying recent relatively strong and selling recent relatively weak stocks with look-back periods of one to four quarters as well as holding periods of one to four quarters. In addition,

other strategies were examined that leave a week between the end of the observation period and the beginning of the holding period in order to prevent systematic biases caused by the effects of bid-ask spreads, price pressure or delayed reactions (Jegadeesh and Titman, 1993, p.68). With regard to the composition of the individual strategies, it is evident that several portfolios are included in each strategy at time t, in particular those that were formed in t and up to $k-1$ months before, with k being the holding period of the represents portfolios (Jegadeesh and Titman, 1993, p.68). Hence, every strategy includes the active portfolios at time t with the same strategic decomposition, in particular the same look-back and holding period (Jegadeesh and Titman, 1993, p.68). Finally, the resulting relative strength portfolio strategies thus relate to the look-back period of j months and the holding period of k months, while being defined by the following model (Jegadeesh and Titman, 1993, p.68):

Model 1 Cross-Sectional Momentum Strategy (Jegadeesh and Titman, 1993)

t = current month
j = look-back period (in months)
k = holding period (in months)
(j,k) = trading strategy
for each month t **do**
  $\{r_t^j\}$ = sequence of assets ranked in ascending order based on return over past j months
  $\{p_t^{decile,j}\}$ = sequence of 10 equally weighted sub-portfolios based on ranking deciles
  $s_t^{buy,j,h} = max\{p_t^{decile,j}\}$ ▷ buy positions in t
  $s_t^{sell,j,h} = min\{p_t^{decile,j}\}$ ▷ sell positions in t
  $p_t^{j,h} = (s_t^{buy,j,h}, s_t^{sell,j,h})$ ▷ portfolio with buy/sell positions held until $t+k$
end for
$r_t^{CSMOM,(j,k)}$ = monthly multi-asset, cross-sectional momentum (j,k)-strategy return

Based on the above model, a new portfolio is defined at each point in time t under the same strategy (j,k) and held for k months, while its positions initiated in $t-k$ are closed in t (Jegadeesh and Titman, 1993, p.68). Furthermore, Jegadeesh and Titman (1993, p.68) tested two alternative specifications of these strategies, within which on the one hand the positions were held between the formation date t and $t+k$ and on the other hand the positions are re-balanced for each month of the holding period between $t+1$ and $t+k$ such that the equal weights of the positions are maintained until the end of the holding period. As already explained above, the total monthly return $r_t^{(j,k)}$ of the strategy (j,k) includes not only those portfolios constructed in t, but also those constructed in constructed between t and $k-1$ and are therefore implicitly still active at time t. Under the condition $k > 1$, there are overlapping portfolios within the strategies (j,k), which positively contributes to the power of the test. Due to the overlapping of portfolios under the same strategy, the weights of the positions within each portfolio have to be adjusted every month t by a factor of $1/k$ (Jegadeesh and Titman, 1993, p.68). Finally, the monthly multi-asset, cross-sectional momentum (j,k)-strategy return can be defined as follows:

$$r_t^{CSMOM,(j,k)} = \sum_{i=0}^{t-k} r_i^{p,(j,k)} \tag{3}$$

where $r_i^{p,(j,k)}$ represents the monthly return of each active portfolio of the overall strategy (j,k) and thus defines the aggregated monthly return of the overall strategy over all months of the evaluation period. Regarding the evaluation of those strategies in the form of various combi-

nations of look-back j and holding periods k as well as data panels, one of the panels being the first one a week after measuring lagged returns in the course of ranking to prevent the previous explained biases carries out the portfolio formation (Jegadeesh and Titman, 1993, p.69). Across all strategy options $j = 3, 6, 9, 12$ and $k = 3, 6, 9, 12$ as well as across both panels, the average monthly performance for the evaluation period of 1965 to 1989 always yields positive monthly returns, with a large part of the measurements showing statistical significance (Jegadeesh and Titman, 1993, p.69). Nevertheless, the strategy ($j = 12, k = 3$) with direct formation of the portfolio shows the overall best, statistically significant performance with an average monthly yield of 1.31% after measuring the lagged returns as part of the relative asset ranking (Jegadeesh and Titman, 1993, p.69). These results underline the accompanying hypotheses and results: On the one hand, cross-sectional momentum represents an empirical anomaly, especially with regard to Fama's efficient market hypothesis. In addition, there are strategies, in particular with a look-back period of twelve months and a holding period of three months, to systematically benefit from the risk premia based on the model 1 and thus to achieve systematic returns over short and long time horizons.

Finally, based on the empirical results, the sources of those excess returns of the relative strength portfolios and thus the underlying return decomposition must be examined, especially with regard to the assessment whether the existence of relative strength profits imply market efficiencies (Jegadeesh and Titman, 1993, p.72). For this, Jegadeesh and Titman (1993, p.68) use two different return-generating models, which are evaluated under the representative strategy ($j = 6, k = 6$). The first model in the form of a simple one-factor model describes stock returns as a linear combination of the unconditional expected return (μ_i), the unconditional expected return on a mimicking portfolio (f_t) under relation on the sensitivity of the instrument (b_i) and the company-specific, idiosyncratic return component (e_{it}). This results in the following return-generating model and the underlying model assumptions:

$$\begin{aligned} r_{it} &= \mu_i + b_i f_t + e_{it} \\ E[f_t] &= 0 \\ E[e_{it}] &= 0 \\ Cov(e_{it}, f_t) &= 0 \ \ \forall i \\ Cov(e_{it}, e_{jt-1}) &= 0 \ \ \forall i \neq j \end{aligned} \tag{4}$$

Furthermore, implicitly from the previous empirical results of momentum effects, positive and negative returns are persistent over time, which can be expressed as follows:

$$\begin{aligned} E[r_{it} - \bar{r}_t | r_{it-1} - \bar{r}_{t-1} > 0] > 0 \\ E[r_{it} - \bar{r}_t | r_{it-1} - \bar{r}_{t-1} < 0] < 0 \end{aligned} \tag{5}$$

Based on the empirical findings of Jegadeesh and Titman (1993), new insights have emerged regarding the investigation of the causes of momentum effects as well as regarding the model specifications. Referring to the insights from behavioral finance, Lakonishok et al. (1994) that in addition to the overreaction or underreaction of market participants, another explanation for momentum effects lies in the extrapolation bias. Here, market participants extrapolate the (recent) past of the asset price or return cross section and infer the (short-term) future based on this (Lakonishok et al., 1994). Thus, relatively strong assets receive an overweight compared to

relatively weak assets, which underlines and reinforces the momentum effect and its existence.

2.3 Time Series Momentum (TSMOM)

As already examined in connection with the momentum anomalies, there also exist anomalies with regard to momentum effects in the asset price time series of individual assets. As the founders of the time series momentum strategy, Moskowitz et al. (2012, p.228) empirically show that robust, systematic excess returns across various asset classes, notably equity index, currencies, commodities and bonds futures, as well as across various markets can be achieved over a period of more than 25 years. In contrast to the cross-sectional momentum strategy, the difference is not only in terms of the data dimension, but also in terms of the observation period in the past, on the basis of which significant predictions can be made. Thus, in the course of the time series momentum strategies, it can be seen that the past 12-month excess returns of individual instruments serve as a positive indicator for predicting future returns (Moskowitz et al., 2012, p.228). That momentum persists consistently and significantly across the asset classes and instruments considered for about one year, after which a partial reversal structure occurs over longer-term horizons (Moskowitz et al., 2012, p.228). With regard to the explainability of the underlying market dynamics, Moskowitz et al. (2012, p.229) and other representatives of similar theories[1] outline that the initial underreaction of the market participants in the course of the granular availability of new information (Hong and Stein, 1999), conservativeness and anchoring biases (Barberis et al., 1998) or disposition effects due to selling profitable holdings too early and holding unprofitable holdings too long (de Long et al., 1990, Hong and Stein, 1999) might explain the timely persistent price and thus return trends. Such ambiguities primarily exist with regard to the higher correlation between time series momentum factors across asset classes compared to the correlation between the asset classes themselves, with regard to similar time series patterns across different asset classes caused by different investor types and ultimately with regard to the lack of empirical relations between time series momentum and existing sentiment analysis indicators (2012, p.229). Hence, a certain degree of unexplainability remains, which underlines the persistent existence of the time series momentum anomalies against existing empirical asset pricing and behavioural theories.

Based on the central findings regarding significant time series momentum effects, the underlying research method must be evaluated. As already explained above, the time series momentum strategy of Moskowitz et al. (2012, p.229) is applied to different asset classes and time periods. The first step is to construct the return series, calculating the daily excess return for each instrument and then creating a cumulative return index by compounding the daily excess returns (Moskowitz et al., 2012, p.233). In addition, the excess returns are scaled by the volatility such that the results become comparable across asset classes and instruments (Moskowitz et al., 2012, p.233). Finally, based on the excess return series, the risk-scaled returns can be regressed with

[1] Moskowitz et al. (2012, p.229) show that time series momentum dynamics directly match behavioural theories, mainly regarding the investor sentiment analysis of under- and overreaction to newly available information and biased self-attribution leading positive short-lag autocorrelations (see, e.g., Barberis et al., 1998, Hong and Stein, 1999, Daniel et al., 1998) as well as rational asset pricing theories covering momentum anomalies (see, e.g., Liu and Zhang, 2008, Johnson, 2002).

the own lagged excess returns according to the following setup (Moskowitz et al., 2012, p.233):

$$r_t^s/\sigma_{t-1}^s = \alpha + \beta_h r_{t-h}^s/\sigma_{t-h-1}^s + \epsilon_t^s \tag{6}$$

$$r_t^s/\sigma_{t-1}^s = \alpha + \beta_h sign(r_{t-h}^s) + \epsilon_t^s \tag{7}$$

where α represents the intercept, β the lag-dependent regression parameter of the lagged excess returns and ϵ the error term, s the indicator of the underlying instrument, t the evaluation time and h the number of lagged time steps. Specifically, setup 7 only focuses on the direction of the lagged excess return according to the general concept of time series momentum while 6 also considers its magnitude which implies a stronger assumption when evaluating the significance and robustness of time series momentum. Thus 6 and 7 basically describe the degree of variance of the volatility-scaled return r_t^s that is explained by the lagged, volatility-scaled excess return r_{t-h-1}^s or its direction, respectively. Furthermore, the annualized ex-ante volatility used in both setups is defined as follows (Moskowitz et al., 2012, p.233):

$$\sigma_t^2 = 261 \sum_{i=1}^{\infty} (1-\delta)\delta^i (r_{r-t-1} - \bar{r}_t)^2 \tag{8}$$

where the term uses the simplified volatility concept of exponentially weighted lagged squared returns, the scalar 261 is used for the annualization of the variance, the weights $(1-\delta)\delta^i$ add up to one and $\bar{r}_t$) defines the weighted average return (Moskowitz et al., 2012, p.233).

Both regression and testing setups by Moskowitz et al. (2012, p.233) exhibit significant results regarding the trend continuation for the first twelve months as well as the long-term reversal structure in the following four years based on a look-back period of twelve months, whereby also for nine months look-back period significant but weaker results are achieved. This represents the empirical basis for developing systematic strategies that are intended to achieve robust excess returns. Moskowitz et al. (2012, p.233) follow a direct approach, which determines the excess returns of the instrument s from its own time series via the deterministic look-back period k, and depending on the accumulated excess return enters a buy (positive accumulated excess return) or sell position (negative accumulated excess return), whereby the position size is set inversely proportional to the ex-ante volatility according to the formula 8 and is held for the deterministic holding period h. In summary, the following standard model for the systematic exploitation of time series momentum results:

Model 2 Time Series Momentum Strategy (Moskowitz et al., 2012)

s = instrument
t = current month
k = look-back period (in months)
h = holding period (in months)
(k, h) = trading strategy
for each instrument s **do**
  for each month t **do**
    $\widetilde{r}^s_{k,t} = r^s_{k,t} - r^{riskfree}_{k,t}$ ▷ excess return at time t over past k months at month t
    $I^s_t = \begin{cases} +1, & \text{if } \widetilde{r}^s_{k,t} > 0 \\ -1, & \text{if } \widetilde{r}^s_{k,t} < 0 \end{cases}$ ▷ momentum indicator based on excess return
    $p^s_t = I^s_t * 1/\sigma^s_{t-1}$ ▷ long/short position size inversely proportional to ex-ante volatility
  end for
end for
$r^{CSMOM,(k,h)}_t$ = monthly multi-asset, time series momentum (k, h)-strategy return

A single strategy return time series $r^{CSMOM,(k,h)}_t$ is derived for each strategy (k, h) in order to avoid overlapping observations for a holding period $h > 1$ (Moskowitz et al., 2012, p.233). Here, the strategy return at time t is calculated as the average return of all active portfolios within the strategy (k, h) at time t, specifically those portfolios within the same strategy that were constructed one, two and more months earlier and are still active in t (Moskowitz et al., 2012, p.233). While the basic framework appears relatively clear based on the empirical findings regarding the time series momentum effects, the relative position sizing inversely proportional to the ex-ante volatility serves further advantages. On the one hand, constant volatilities are achieved across all strategies, which means that there is comparability regardless of the underlying volatility. On the other hand, this prevents the performance of the strategy from being influenced by certain volatility regimes and thus remaining stable and evaluable (Moskowitz et al., 2012, p.233).

Ultimately, the strategy performance must be determined in the form of monthly overperformance. For this purpose, Moskowitz et al. (2012, p.235) calculated the alpha through a regression on the monthly strategy return $r^{CSMOM,(k,h)}_t$, with three central asset classes indices for equities, bonds and commodities and the Fama-French stock-market factors in terms of size, value and cross-sectional momentum premium are defined as covariates to extract passive (market) exposures:

$$\widetilde{r}^{CSMOM,(k,h)}_t = \alpha + \beta_1 MKT_t + \beta_2 BOND_t + \beta_3 GSCI_t + sSMBt + hHML_t + mUMD_t + \epsilon_t \quad (9)$$

where α represents the intercept and thus the abnormal return as well as $\beta_1, \beta_2, \beta_3, s, h, m$ the regression coefficients of the previously explained covariates. Based on the empirical investigation, it can be seen that the alphas show significance over different time horizons and asset classes, and show particular significance and robustness for look-back and holding periods $k, h \leq 12$ (Moskowitz et al., 2012, p.236). These results underline that not only do time series momentum effects exist, but that these can also be exploited through targeted systematic strategies and systemic abnormal returns against the market can be achieved.

According to the return decomposition in order to identify the underlying return properties leading to time series momentum patterns, Moskowitz et al. (2012, p.229) show that a positive

auto-covariance can be found in the asset return time series. This means that there is a significant, positive covariance between lagged pairs of returns and thus fundamentally describes a dependency within the return time series, which is further identified as the main contributor to time series momentum effects in accordance with Lewellen (2002). In addition, it is shown that time series momentum effects are the basis of individual stock momentum effects and thus also with regard to cross-sectional momentum effects, which is explained, among other things, by the UMD factor of Fama and French in the form of the premium between strong and weak stocks (Moskowitz et al., 2012, p.229). According to this empirical fact, there is a clear connection between time series and cross-sectional momentum effects, which is explained in more detail in the next chapter along with central similarities, differences as well as model limitations.

In addition to the return decomposition and examination of return properties, it is also important to examine the underlying drivers of time series momentum effects. To this end, Moskowitz et al. (2012, p.229) primarily on examining the trading activities of speculators and hedgers and the associated return patterns. It shows that speculators take active positions based on time series momentum to generate positive excess returns over a period of up to 12 months and then granularly reduce the position again, which is consistent with the basic empirical evidence regarding significant holding periods (Moskowitz et al., 2012, p.229). This results in a structure in which speculators make profits at the expense of hedgers, since the latter take the opposite position in the case of return shocks (Moskowitz et al., 2012, p.229). With regard to the reversal effects in the longer term, Moskowitz et al. (2012, p.229-230) that these dynamics are mainly caused by price changes in the spot market, which agrees with the theory that the continuing return trends are to be classified as an overreaction of the market and are therefore long-term reverse again. However, this should not be confused with the under-reaction of the market, which, in connection with the granular pricing of new information, leads to the initial formation of persistent return momentum.

3 Deep Learning Architectures

The central goal of this thesis is given by the development and derivation of deep learning-enhanced multi-asset CSMOM and TSMOM strategies in order to achieve empirical out-performance against existing benchmark strategies. That derivation is based on vanilla deep learning architectures and the adaptation of those with regard to generalized CSMOM and TSMOM strategy frameworks. Hence, the first step is to examine existing deep learning architectures and derive them in terms of their mathematical functionality. This is done in the context of this section, with the first step evaluating empirically significant cross-sectional and time series-related deep learning models in the form of the *multi-layer perceptron* (MLP), the *recurrent neural network* (RNN) as well as the *long short-term memory neural network* (LSTM), which in turn serve in an adapted form as benchmark architectures. In the second step, the discussion and derivation of *transformer neural networks* follow, which achieve state-of-the-art performances, especially with regard to applications in text and natural language processing, while it can also be adapted with regard to numerical applications, and ultimately serve as the underlying vanilla architecture of

the proposed CSMOM and TSMOM strategies.

3.1 Key Deep Learning Models

The development of time series and cross-sectional momentum-specific deep learning architectures first requires the evaluation of the basic model basis. Artificial neural networks (hereinafter referred to as neural networks) basically represent mathematical models which, analogously to the naming convention, adapt the human brain's distinctive feature of learning and adapting and apply it to prediction as well as classification use cases across many research and industry application areas (Sazli, 2006, p.12). In the field of deep learning and hence neural networks in particular, the model architectures have evolved over the past few years in order to resolve shortcomings and limitations for certain application areas. In the following, the central deep learning architectures are therefore defined, especially with regard to the area of application of univariate and multivariate time series data, in order to then select the appropriate underlying architecture and then specifically expand it with regard to TSMOM and CSMOM.

3.1.1 Multi-Layer Perceptron (MLP)

Model Architecture

The architecture of the feed-forward neural network, so called multi-layer perceptrons, represents one of the central foundations on which state-of-the-art models are built upon. In general, neural networks are networks of individual *neurons* that represent the processing elements, while the main purpose consists of mapping a set of input values into a set of appropriate output values (Alsmadi et al., 2009, p.296). Those neurons are able to receive input signals, process them and then pass them on as an output signal. The neurons are located in different layers throughout the network, with *single-layer* neural networks consisting of an input and an output layer and *multi-layer* neural networks also consist of hidden layers between the input and output layer (Sazli, 2006, p.26). Those layers in a multi-layer perceptron can be defined as follows:

$$\begin{aligned} \text{Input:} \quad & \mathbf{x} \in \mathbb{R}^{1\times p} \\ \text{Hidden:} \quad & \mathbf{h}^{(l)} \in \mathbb{R}^{1\times k_l} \\ \text{Output:} \quad & \mathbf{o} \in \mathbb{R}^{1\times q} \end{aligned} \tag{10}$$

where l represents the hidden layer indicator since networks can consist of multiple hidden layers which gives the foundation for *deep* neural networks, p the number of covariates within the input data and q the number of output data. Corresponding to each layer, the input, hidden and output neurons, also referred to as input, hidden and output units, can be defined as follows:

$$\begin{aligned} \text{Input:} \quad & x_p \in \mathbf{x} \\ \text{Hidden:} \quad & h_i^{(l)} \in \mathbf{h}^{(l)} \\ \text{Output:} \quad & o_q \in \mathbf{o} \end{aligned} \tag{11}$$

Within and across the hidden and output layers, any neuron i is connected to at least one other neuron j within the previous layer, with each connection being characterized by the *weight coefficient*. In addition, each neuron's value consists of a *bias* term, often also referred to as a threshold term, which allows to offset the ith neuron's value (Svozil et al., 1997, p.45). Hence, the weights and biases can the formally defined as follows:

$$w_{ij}^{(l)} \in \mathbf{w}_i^{(l)} \in \mathbb{R}^{1 \times k_{l-1}} \quad , \quad \mathbf{w}_i^{(l)} \in \mathbf{W}^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \tag{12}$$

$$b_i^{(l)} \in \mathbf{b}^{(l)} \in \mathbb{R}^{1 \times k_l} \tag{13}$$

Thus each neuron receives "weighted" information from its predecessor neurons through the weight coefficient w_{ij}, while the weighted input can be offset by the bias/threshold term b_i. Hence, the unique weight coefficient determines the degree of information that flows between two neurons through their unique artificial connection. The processing and thus the generation of the output signal based on the input signal, which in turn is passed on to the next neuron by a further weight coefficient, is carried out by the *activation function* $\sigma(\cdot)$. Hence, the hidden neuron as well as each hidden layer in matrix notation can be formally described as follows (Grosse, 2019, p.3-4; Svozil et al., 1997, p.45):

$$\begin{aligned} h_i^{(l)} &= \begin{cases} \sigma\left(b_i^{(1)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(1)} x_j\right) & , \text{ if } l = 1 \\ \sigma\left(b_i^{(l)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(l)} h_j^{(l-1)}\right) & , \text{ if } l \geq 2 \end{cases} \\ \mathbf{h}^{(l)} &= \begin{cases} \sigma\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) & , \text{ if } l = 1 \\ \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right) & , \text{ if } l \geq 2 \end{cases} \end{aligned} \tag{14}$$

where the values of the hidden neurons $h_i^{(l)}$ are calculated as the activation function $\sigma(\cdot)$ of the weighted sum of the predecessor neurons, offset by the bias $b_i^{(l)}$. The value vector of the respective hidden layer $\mathbf{h}^{(l)}$, consisting of all hidden neurons $h_i^{(l)}$, is derived analogously based on the activation function $\sigma(\cdot)$, the weight matrix $\mathbf{W}^{(l)}$ and the bias vector $\mathbf{b}^{(l)}$. A distinction is made between the first and the remaining hidden layers, since the first hidden layer depends on the input layer and the following hidden layers depend on the previous one. Following the hidden layers, the derivation of the output neurons and output layer follows accordingly (Grosse, 2019, p.3-4; Svozil et al., 1997, p.45):

$$o_q = \sigma\left(b_i^{(O)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(O)} h_j^{(L)}\right) \quad , \quad \mathbf{o} = \sigma\left(\mathbf{W}^{(O)}\mathbf{h}^{(L)} + \mathbf{b}^{(O)}\right) \tag{15}$$

where the output neurons o_q and the entire output layer $\mathbf{o}$ are weighted sum of the hidden neurons $h_j^{(L)}$ in the last hidden layer $\mathbf{h}^{(L)}$, offset by the biases. The concrete form of the activation function $\sigma(\cdot)$ referred (15) and in 14 has to be determined in advance deterministically or in the context of hyperparameter tuning while typically assuming one of the following definitions

(Sharma et al., 2020, p.312-313):

$$\begin{aligned}
\text{Linear:} \quad & \sigma(\xi_i) = a\xi_i + c \\
\text{Sigmoid:} \quad & \sigma(\xi_i) = \frac{1}{1 + e^{-\xi_i}} \\
\text{Tanh:} \quad & \sigma(\xi_i) = \frac{2}{1 + e^{-2\xi_i}} - 1 \\
\text{ReLU:} \quad & \sigma(\xi_i) = \begin{cases} \xi_i, & \text{if } \xi_i \geq 0 \\ 0, & \text{if } \xi_i < 0 \end{cases} = \max(\xi_i, 0)
\end{aligned} \tag{16}$$

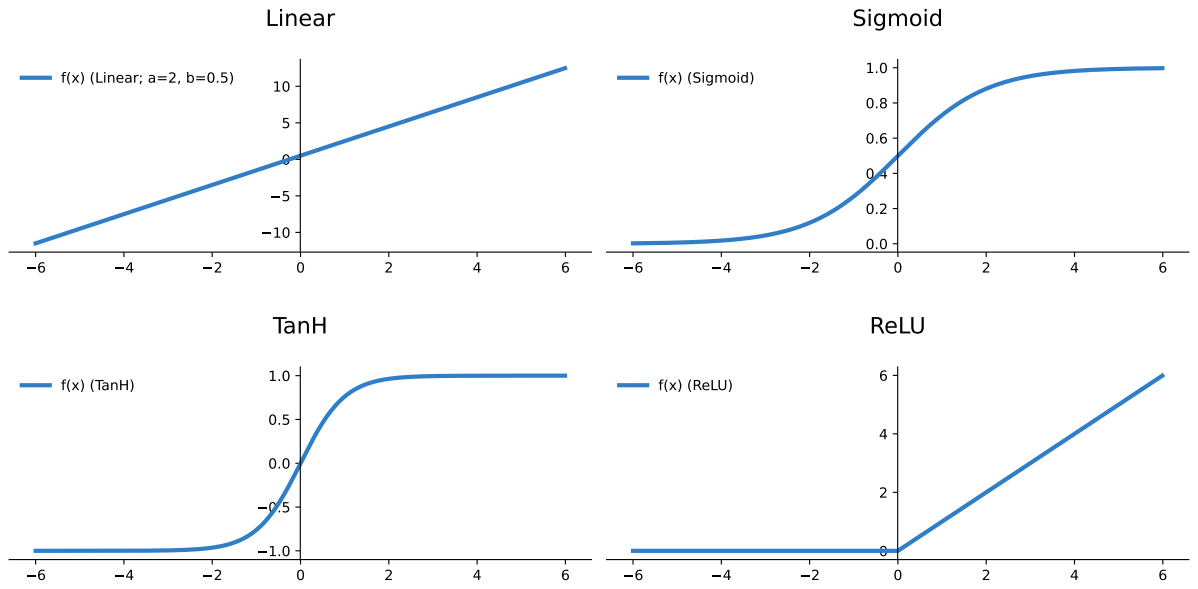


Figure 1: Activation Functions - Own Illustration

The central characteristics of the activation functions from (16) are as follows: The *binary step* function, which is not mentioned above in particular, has a threshold of 0 based on the input value, which determines whether the neuron is activated or not. The latter case means that the neuron does not pass on any value as input along the information path to the next layer. The definition already shows that the binary step function can only be used in the case of the binary classifier. In addition, the binary step function contains a gradient of zero, which can lead to issues in the context of back-propagation (elaborated in a later step). The *linear* activation function takes on the last problem of the binary step function and is defined as a linear transformation of the input value, whereby the gradient is well-defined and greater than zero, since the derivative corresponds to the scaling factor c. Nevertheless, the problem exists because the error cannot be minimized in the context of back-propagation, since the derivative is constant. Furthermore, the linear function does not allow non-linear, complex structures within the network, which is one of the main challenges to be solved in many use cases. The *sigmoid* function, on the other hand, solves this challenge by introducing non-linearity and represents one of the most widespread activation functions used in research and industry applications. The input values are transformed into a range of $[0, 1]$, whereby the function is continuously differentiable and exhibits a well-defined, non-constant derivative. Finally, the non-symmetry

about zero also means that all output values along the information paths have the same sign. A similar activation function is the *hyperbolic tangent (Tanh)* function, which however has symmetry around 0 and therefore does not have the same sign in each neuron. Furthermore, the function is defined on the interval $[-1, 1]$ and is also continuously differentiable, with the derivative being steeper. Finally, the *rectified liner unit (ReLU)* function is to be examined, which is also a non-linear function. The advantage here is efficiency, since only a certain number of neurons are activated at the same time. Similar to the binary step function, however, there is also the issue that individual derivatives are zero and therefore the relevant weights and biases cannot be updated as part of back-propagation (Sharma et al., 2020, p.312-313).

Based on the previous explanations regarding the basic structure in the form of the network layers, the neurons and their connection characteristics as well as the formal definition of the information flow within the network, the following structure can be elaborated as a basis for further model architectures:

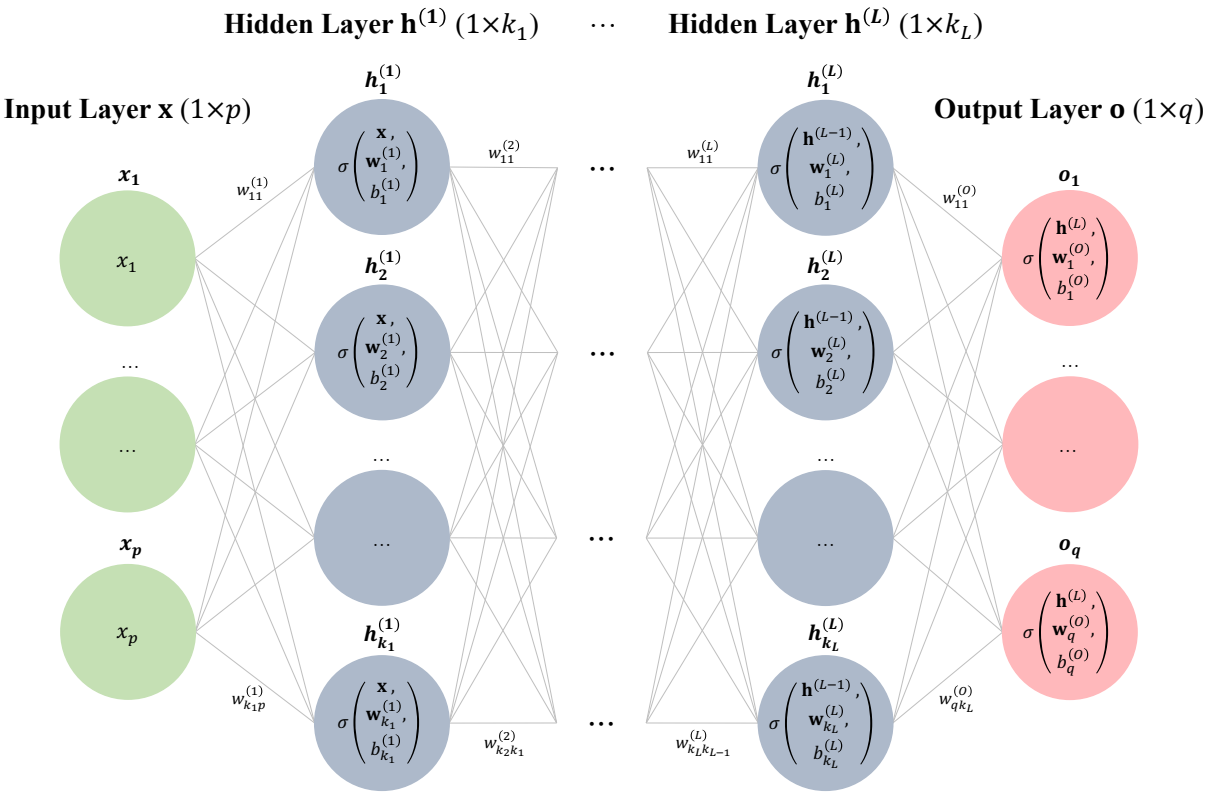


Figure 2: MLP Architecture - Own Illustration

$$
\begin{aligned}
\text{Input Layer:} \quad & \mathbf{x} \in \mathbb{R}^{1 \times p} && \\
\text{Hidden Layer(s):} \quad & \mathbf{h}^{(l)} \in \mathbb{R}^{1 \times k_l} && \forall\, l \in \{1, ..., L\} \\
\text{Output Layer:} \quad & \mathbf{o} \in \mathbb{R}^{1 \times q} && \\
\text{Input Neurons:} \quad & \mathbf{x} = \mathbf{x} && \\
\text{Hidden Neurons:} \quad & \mathbf{h}^{(l)} = \sigma\left(\xi^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
& \xi^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)} && \\
\text{Output Neurons:} \quad & \mathbf{o} = \sigma\left(\xi^{(O)}\right) && \\
& \xi^{(O)} = \mathbf{W}^{(O)}\mathbf{h}^{(L)} + \mathbf{b}^{(O)} && \forall\, l \in \{1, ..., L\} \\
\text{Weight:} \quad & \mathbf{w}_i^{(l)} \in \mathbf{W}^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} && \forall\, i \in \{1, ..., k_l\} \\
\text{Bias:} \quad & \mathbf{b}^{(l)} \in \mathbb{R}^{1 \times k_l} && \forall\, i \in \{1, ..., k_l\}
\end{aligned}
\tag{17}
$$

where $\mathbf{z}^{(l-1)} \in \left\{\mathbf{x}, \mathbf{h}^{(1)}, ..., \mathbf{h}^{(L-1)}\right\}$ refers to the previous layer and either consists of the input layer or the previous hidden layer, depending on the location of the hidden layer l. This general overview of a "vanilla" multi-layer perceptron summarizes the sufficient model components. It shows to what extent the core elements of the weighted information flows between the neurons are embedded in the overall architecture under the concept of neuron activation. On top of that, the overview serves a foundation for decomposing the model components which can be optimized within the framework of model training with regard to the general concept of learning and adapting. This optimization in the context of back-propagation is examined in more detail in the following section.

Back-Propagation & Gradient Descent

In the course of training multi-layer perceptrons, back-propagation is one of the most fundamental algorithms as part of supervised learning techniques and contributes significantly to the popularity and performance of neural networks. The "vanilla" multi-layer perceptron architecture from formula 17 shows that both the connection weights $w_{ij}^{(n)}$ between the neurons of different layers as well as the neuron-specific bias terms $b_i^{(n)}$ serve as central optimization parameters since they can be adjusted in a data-driven way. Nevertheless, due to the feed-forward dynamics, the network depth and the inter-dependencies between the neurons, the parameters cannot be optimized locally only but must be optimized along the entire information paths. Hence, while the information flow from the input up until the output layer propagates forward through all layers with $\hat{Y}_{(1 \times Q)}$ as the final vector of predictions, the minimization of the prediction error with respect to the vectors of biases and weights happens under back-propagation from the output layer back until the input layer. In essence, back-propagation is defined as the updating of network synaptic weights by back-propagating a gradient vector with each vector-element representing the derivative of an error measure with respect to a parameter (Sazli, 2006, p.14). Hence, based on this general definition, the first step is to define a *cost function*, also

referred to as an error or objective function, that quantifies the deviation between the true and predicted values while serving as the target for minimization:

$$C(\mathbf{o}, \mathbf{y}) = \sum_{i=1}^{q} C(o_i, y_i) \tag{18}$$

where o_i and y_i represent the predicted outputs and true values, respectively. One of the most trivial cost functions is the *mean squared error* (MSE) and is used in many applications as the fundamental error structure, since outlying predictions are heavily penalized by the squared distance. The MSE as a cost function in the context of multi-layer perceptrons is defined as follows (Svozil et al., 1997, p.45):

$$C(\mathbf{o}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{q} (y_i - o_i)^2 \tag{19}$$

where, analogously to formula (18), y_i and o_i represent the true output and predicted values, respectively, from the ith output neuron. Since the weight coefficients $\mathbf{W}^{(l)}$ and the biases $\mathbf{b}^{(l)}$ represent the adjustable parameter within the multi-layer perceptron, the cost function has to be minimized with respect to these parameter. The *gradient descent* method is suitable for this, whereby in each iteration of the back-propagation a step is taken opposite to the gradient ∇, also referred to as steepest descent, of the cost function, resulting in the following direction (Ernst, 2014, p.1):

$$-\nabla C(\mathbf{o}, \mathbf{y}) \tag{20}$$

Based on the definition of the gradient descent method within the framework of back-propagation as well as the *forward pass* fully derived in formula (17), the updating of the weights matrix $\mathbf{W}^{(l)}$ and the bias vector $\mathbf{b}^{(l)}$ on the *backward pass* is derived from the following learning rules in accordance with the gradient descent method (Svozil et al., 1997, p.45; Ernst, 2014, p.1):

$$\begin{aligned} \mathbf{W}^{(l)} &= \mathbf{W}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{W}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\} \\ \mathbf{b}^{(l)} &= \mathbf{W}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{b}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\} \end{aligned} \tag{21}$$

where $\mathbf{W}^{(l)} \in \left\{ \mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)}, \mathbf{W}^{(O)} \right\}$ and $\mathbf{b}^{(l)} \in \left\{ \mathbf{b}^{(1)}, ..., \mathbf{b}^{(L)}, \mathbf{b}^{(O)} \right\}$. In addition, $\lambda > 0$ represents the deterministic *learning rate* which defines an additional model hyperparameter and can be optimized in a data-driven way during the hyperparameter optimization process (Svozil et al., 1997, p.45). λ controls the payoff between the efficiency of the calculation and the achievement of a local or global minimum of the cost function $C(\cdot)$.

Formula (21) shows that the first order derivatives of the cost function with respect to $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ represent the central challenges. Based on the application of the *chain rule*, which defines a central methodology in deriving the first order derivatives in back-propagation algorithms, the following notation arises (Svozil et al., 1997, p.46):

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{W}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} \overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l)}} \frac{\partial \sigma\left(\xi^{(l)}\right)}{\partial \xi^{(l)}} \frac{\partial \xi^{(l)}}{\partial \mathbf{W}^{(l)}} = \frac{\partial C}{\partial \mathbf{z}^{(l)}} \sigma'\left(\xi^{(l)}\right) \mathbf{z}^{(l-1)\mathrm{T}} \\
\frac{\partial C}{\partial \mathbf{b}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l)}} \frac{\partial \sigma\left(\xi^{(l)}\right)}{\partial \xi^{(l)}} \frac{\partial \xi^{(l)}}{\partial \mathbf{b}^{(l)}} = \frac{\partial C}{\partial \mathbf{z}^{(l)}} \sigma'\left(\xi^{(l)}\right)
\end{aligned} \tag{22}$$

where $\mathbf{z}^{(l)} \in \left\{\mathbf{h}^{(1)}, ..., \mathbf{h}^{(L)}, \mathbf{o}\right\}$ and $\xi^{(l)} \in \left\{\xi^{(1)}, ..., \xi^{(L)}, \xi^{(O)}\right\}$ define the vector of output values in the hidden layers and the output layer. In addition, $\sigma'\left(\xi^{(l)}\right)$ represents the vector of the first order derivatives of the activation function. Based on the common activation functions introduced in formula (16), the following first order derivatives can be derived formally as well as visually:

$$\begin{aligned}
\text{Linear:} \quad & \frac{\partial}{\partial \xi_i}\sigma(\xi_i) = \sigma'(\xi_i) = c \\
\text{Sigmoid:} \quad & \frac{\partial}{\partial \xi_i}\sigma(\xi_i) = \sigma'(\xi_i) = \sigma(\xi_i)[1 - \sigma(\xi_i)] \\
\text{Tanh:} \quad & \frac{\partial}{\partial \xi_i}\sigma(\xi_i) = \sigma'(\xi_i) = 1 - \sigma(\xi_i)^2 \\
\text{ReLU:} \quad & \frac{\partial}{\partial \xi_i}\sigma(\xi_i) = \sigma'(\xi_i) = \begin{cases} 1, & \text{if } \xi_i \geq 0 \\ 0, & \text{if } \xi_i < 0 \end{cases}
\end{aligned} \tag{23}$$

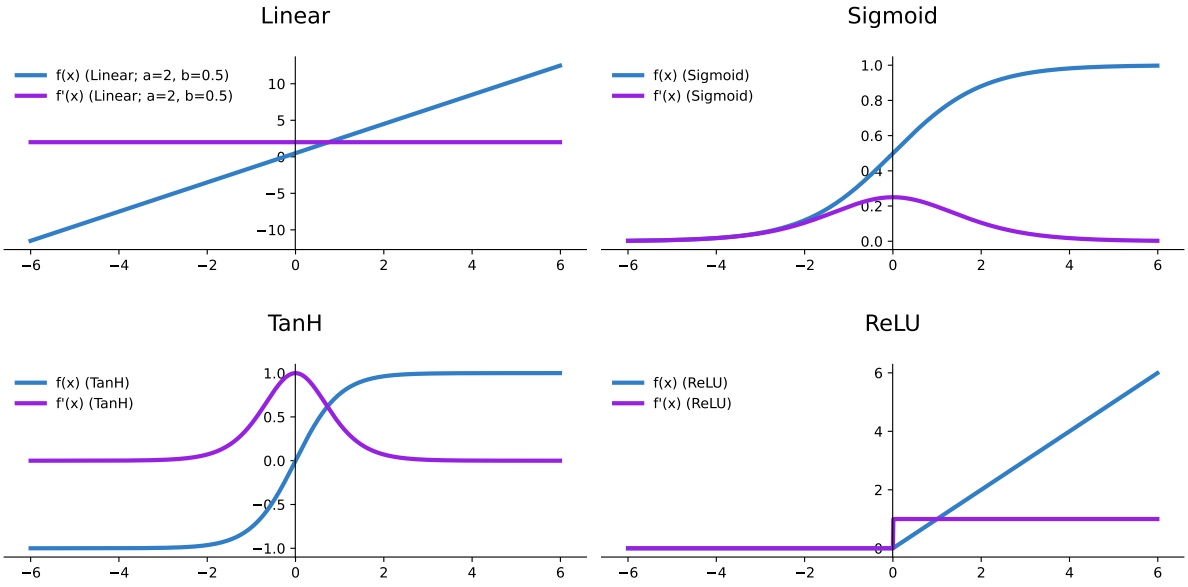


Figure 3: Activation Functions & Derivatives - Own Illustration

In addition to the introduction to of the first order derivatives of the activation functions, formula (22) results in a central property between the first order derivatives of the cost function with respect to the weights matrix $\mathbf{W}^{(l)}$ and biases/thresholds $\mathbf{b}^{(l)}$, respectively (Svozil et al., 1997, p.46):

$$\frac{\partial C}{\partial \mathbf{W}^{(l)}} = \frac{\partial C}{\partial \mathbf{b}^{(l)}} \mathbf{z}^{(l-1)\mathrm{T}} \qquad \forall\, l \in \{1, ..., L, O\} \tag{24}$$

The last step consists of defining the first derivatives of the cost function with respect to the ith neuron output values x_i, from which the foundation of the back-propagation algorithm emerges (Svozil et al., 1997, p.46):

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{z}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l+1)}} \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{z}^{(l)}} \overset{\text{chain rule}}{=} \frac{\partial C}{\partial \mathbf{z}^{(l+1)}} \frac{\partial \sigma\left(\xi^{(l+1)}\right)}{\partial \xi^{(l+1)}} \frac{\partial \xi^{(l+1)}}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial C}{\partial \mathbf{z}^{(l+1)}} \sigma'\left(\xi^{(l+1)}\right) \mathbf{W}^{(l+1)} \overset{(22)}{=} \frac{\partial C}{\partial \mathbf{b}^{(l+1)}} \mathbf{W}^{(l+1)} \qquad \forall\, l \in \{1, ..., L, O\} \end{aligned} \tag{25}$$

where $\mathbf{z}^{(l+1)} \in$ and $\xi^{(l+1)}$ define the vectors output and input values of the one-step ahead layers $l = 1...L$. In addition, a special case arises for the output layer $l = O$ based on the property in formula (22) (Svozil et al., 1997, p.46):

$$\frac{\partial C}{\partial \mathbf{z}^{(O)}} = \mathbf{o} - \mathbf{y} \tag{26}$$

Based on the derivations of formula (21) to (26), the updating of the weight and bias parameters based on the gradient descent can be back-propagated through the entire network from the output layer O through the following hidden layers $l = 1...L$, which finally delivers the optimized multi-layer perceptron parameters (Svozil et al., 1997, p.46).

Model Training & Testing

In principle, it is important to differentiate between in-sample training and out-of-sample testing when evaluating multi-layer perceptrons and model performance in general. The model training serves to calibrate the variable parameters in order to finally evaluate the accuracy and the performance within the framework of testing, whereby "new" and training-independent data are used as input in the testing, which are not yet known for the model (Svozil et al., 1997, p.46). Thus, the underlying data set has to be divided into training and testing subsets, namely the vectors of covariates X^{train} and X^{test} as well as the vectors of dependent variables Y^{train} and Y^{test}.

Based on the previous explanations, the back-propagation algorithm represents the central method for the training and thus the learning of the multi-layer perceptron. At the beginning of the model training under X^{train} and Y^{train} defines arbitrary values for the connection weights in the entire network, which are either chosen completely randomly or under specific rules (Svozil et al., 1997, p.46). On the basis of this follows an iteration of back-propagation across the whole network, a so-called *epoch*. The number of epochs and thus the number of back-propagation iterations under the arbitrary start values or subsequently under the previously optimized weights finally leads to the achievement of the local minima of the cost function and hence the locally minimizing weights, assuming a sufficient learning rate $\lambda > 0$, and also represents a corresponding model hyperparameter (Svozil et al., 1997, p.46). Nevertheless, too many epochs can lead to the generalization of the data structure being too close to the actually in-sample data structure and therefore delivering poor performances in the following testing.

This characteristic is called *overfitting* and is characterized by good in-sample but weak out-of-sample performances (Svozil et al., 1997, p.47). However, the same applies vice versa, whereby the training data structure is generalized too much, resulting in *underfitting*, which can be caused by too few epochs (Svozil et al., 1997, p.46). This underlines the characteristics of the epochs as a hyperparameter, which, as a deterministic parameter, must also be optimized in the course of training.

After completing the model training, the model optimized for the in-sample training set and its optimized parameters must be tested in the course of the out-of-sample testing sets X^{test} and Y^{test}. The out-of-sample covariates flow into the new network and propagate from the input layer up until the output layer which generates the final predictions, independent of the use case being of discrete (classification) or continuous (regression) prediction. The goal is therefore the optimal generalization of the training data structure, so that this generalization leads to optimal out-of-sample predictions in the testing sample. As already mentioned, the prerequisites for good generalizations are the number of selected epochs, as well as the sufficient size of in-sample training data and the representability of the subset (Svozil et al., 1997, p.47).

Optimal Model Architecture & Hyperparameter Tuning

As can already be seen from the previous section, the optimal generalization and thus the prevention of overfitting of the training sample is one of the central challenges in order to ensure good out-of-sample performance which ultimately defines the main goal of multi-layer perceptrons and other models in general. The general connection between the in-sample and out-of-sample model error as well as the model complexity in the course of under- and overfitting can be visualized in the following plot (Smith, 2018, p.3):

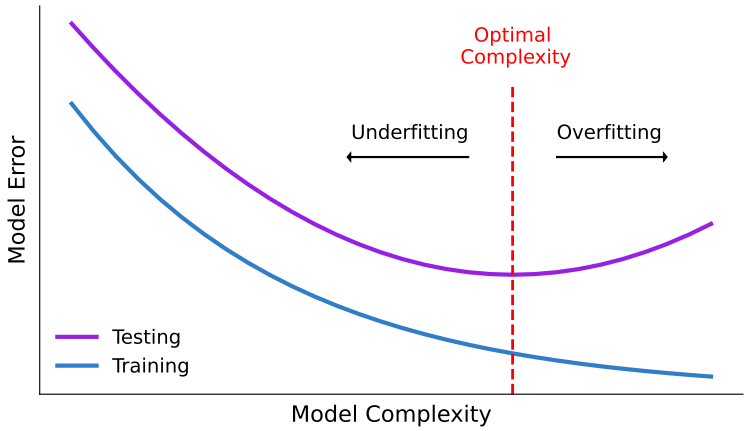


Figure 4: Optimal Model Complexity, Underfitting & Overfitting - Own Illustration

Figure 4 shows the basic structure of deep learning models in the context of optimal model complexity and thus the prevention of under- and overfitting biases. While in-sample training error is (strictly) decreasing with respect to the model complexity, there is a different structure for the out-of-sample testing error, since a generalization that is too strongly aligned with the training sample leads to poorer performance out-of-sample, which is written by the phenomenon

of *overfitting*. The other way around, it also applies that too strong a generalization also leads to poorer performance out-of-sample, which is described as *underfitting*. In addition to the aforementioned sufficient size of the training sample, there are other central approaches to avoid overfitting and thus to converge to an optimal degree of generalization, which are explained in the following section.

With regard to the *model selection* in the context of multi-layer perceptrons with a look back at the "vanilla" structure defined in figure 2, the choice of the number N of hidden layers $H_{n,(1\times k_n)}$ as well as the number of neurons k_n within each layer n results as two of the central control components (Svozil et al., 1997, p.47). However, there is no deterministic rule that specifies the optimal number of neurons in the hidden layers. On the one hand, the optimum depends on the complexity of the actual data-based prediction problem, but on the other hand there are also dependencies on other components of the model architecture, such as early stopping and regularization components (Svozil et al., 1997, p.48). With regard to the number of hidden layers, reference should first be made to the *universal approximation property*, which is represents the foundation for using neural networks for prediction problems. This states that a continuous function on a bounded domain, which represents the mapping between the covariates and the dependent variable, can be approximated by a sufficiently large two or more layer neural network (Lu and Lu, 2020, p.1). Thus, even with few or even no hidden layers and hence low network depth, it can be assured that there is a good approximation, given the sufficient network size (Svozil et al., 1997, p.48). In addition, hidden layers contribute to the fact that only local minima and not the global minimum of the cost function might be reached in the course of back-propagation, and the training process loses efficiency (Svozil et al., 1997, p.48). Nevertheless, hidden layers also increase the complexity of the model to the extent that more complex functions can be approximated (Svozil et al., 1997, p.48). Thus, the number of neurons in the hidden layers as well as the number of hidden layers must also be evaluated in a data-driven way by estimating models under different deterministic settings, while it is recommended to start with one hidden layer and many hidden neurons and to gradually increase the layers under different number of hidden neurons (Svozil et al., 1997, p.48).

As the previous section elaborates, deep learning models consist of pre-defined architectural parameters that need to be optimized in a data-driven approach, which are defined as *hyper-parameters*. Those hyperparameters are not independently adapted by a universal learning algorithm $f(\mathbf{x};\theta)$ in the course of model training, but are determined a priori (Goodfellow et al., 2016, p.120-121). Here, given hyperparameter spaces, and thus the sets of possible values of each hyperparameter, different learning algorithms are tested with regard to model performance. In order to prevent in-sample bias, a further data split can be defined in the form of the *validation set* in addition to the existing training and testing set, whereby the previous in-sample training set is again split into a training set and validation set (Goodfellow et al., 2016, p.121). Thus, the model training of each learning algorithm takes place under the different hyperparameters on the new training set, after which the performance evaluation of those learning algorithms takes place on the validation set (Goodfellow et al., 2016, p.121). Consequently, the best-performing learning algorithm with regard to the underlying hyperparameters based on the train-validation performance is chosen, which is then used as the final model for out-of-sample prediction based

on the test set (Goodfellow et al., 2016, p. 121). In addition to model selection, methods under the cloak of *regularization* are used to additionally prevent overfitting biases. In the course of a general learning algorithm $f(\mathbf{x};\theta)$, the cost function is adjusted so that a penalty term is added, which creates a preference with regard to the hypothesis space (Goodfellow et al., 2016, p.118-119). The aim of this regularization is to reduce the generalization error as effectively as possible without negatively influencing the training error in order to ultimately achieve an optimal balance of the model fit with reference to figure 4 (Goodfellow et al., 2016, p.120).

3.1.2 Recurrent Neural Network (RNN)

Model Architecture

Recurrent neural networks (RNN) represent a fundamental further development and at the same time a modification of multi-layer perceptrons or feed-forward neural networks, whose architecture is primarily designed for the detection of patterns in sequences of data (Schmidt, 2019, p.1). This basic definition already shows that recurrent neural networks found large areas of application, especially in the area of time series analysis and in particular financial market analysis (Schmidt, 2019, p.1), which is precisely the reference to time series and cross-sectional momentum strategies. The most fundamental difference to multi-layer perceptrons is in the area of information flow: while MLPs have a strict feed-forward information flow, RNNs have "recurrent" structures, whereby information components are back-transmitted through cycles in the network (Schmidt, 2019, p.1). This allows to fall back and reference on several and, above all, differently indexed input values, which makes it possible to explicitly model serial (auto-)correlations and/or dependencies, which is not possible or only indirectly possible in the context of MLPs. In general, research often distinguishes between RNNs and Deep RNNs, with the latter being constructed by stacking ordinary RNNs of any type (Schmidt, 2019, p.4). In the context of this work, Deep RNNs are used as the basis of the derivation of the architecture in order to generalize those derivations independent of the depth of the networks.

The foundation for deriving the concrete architecture of recurrent neural networks is defined by the "vanilla" multi-layer perceptron architecture from Figure 2 and formula 17. Thus, there are analogous structures with regard to the central model components of the input, hidden and output layers as well as the neurons within the layers, while the layers being indexed in time. This results in the following definition of the inputs, hidden and the output layers:

$$\begin{aligned} \text{Input:} \quad & x_{t,p} \in \mathbf{x}_t \in \mathbb{R}^{1\times p} \\ \text{Hidden:} \quad & h_{t,i}^{(l)} \in \mathbf{h}_t^{(l)} \in \mathbb{R}^{1\times k_l} \\ \text{Output:} \quad & o_{t,q} \in \mathbf{o}_t \in \mathbb{R}^{1\times q} \end{aligned} \tag{27}$$

where t represents the time or the sequential index in general, p the number of inputs, so called covariates in the context of the "vanilla" multi-layer perceptron, and k_l the number of hidden neurons in each layer l (Schmidt, 2019, p.1). It should be noted here that the number of hidden layers L and the number of neurons k_l in each hidden layer l are identical across the indices t (Schmidt, 2019, p.1-2). While the input layer is completely characterized by the input

data and deterministic model parameters analogous to MLPs, the hidden layers are defined by the predecessors as well as the same hidden layer at the previous index $t-1$, while the output layer only depends on the last hidden layer at each time step. Hence, the recurrent neural network consists of additional weight coefficients which can be formally defined as follows:

$$\begin{aligned} w_{ij}^{(l)} \in \mathbf{w}_i^{(l)} \in \mathbf{W}^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \\ v_{ij}(l) \in \mathbf{v}_i^{(l)} \in \mathbf{V}^{(l)} \in \mathbb{R}^{k_l \times k_l} \end{aligned} \tag{28}$$

where the weight matrix $\mathbf{W}^{(l)}$ contains the weight coefficients of the hidden layer l to the previous hidden layer $l-1$ or input layer $\mathbf{x}$ and the weight matrix $\mathbf{V}^{(l)}$ the weight coefficients of hidden layer l with the same hidden layer l with previous index $t-1$. It should be noted here that any weight parameters are the same across the indices.

$$b_i^{(l)} \in \mathbf{b}^{(l)} \in \mathbb{R}^{1 \times k_l} \tag{29}$$

Thus, the hidden first hidden layer as well any following hidden layer can be derived as follows:

$$\begin{aligned} h_{t,i}^{(l)} &= \begin{cases} \sigma\left(b_i^{(1)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(1)} x_{t,j} + v_{ij}^{(1)} h_{t-1,j}^{(1)}\right) & , \text{ if } l = 1 \\ \sigma\left(b_i^{(l)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(l)} h_{t,j}^{(l-1)} + v_{ij}^{(l)} h_{t-1,j}^{(l)}\right) & , \text{ if } l \geq 2 \end{cases} \\ \mathbf{h}_t^{(l)} &= \begin{cases} \sigma\left(\mathbf{W}^{(1)} \mathbf{x}_t + \mathbf{V}^{(1)} \mathbf{h}_{t-1}^{(1)} + \mathbf{b}^{(1)}\right) & , \text{ if } l = 1 \\ \sigma\left(\mathbf{W}^{(l)} \mathbf{h}_t^{(l)} + \mathbf{V}^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)}\right) & , \text{ if } l \geq 2 \end{cases} \end{aligned} \tag{30}$$

where $\mathbf{h}_t^{(l)}$ represents the vector of the hidden neurons in the hidden layer l at time t. Those hidden neurons basically depend on the previous layer $l-1$ with the same index t and the same layer l at the previous time $t-1$. In addition, the weight matrix $\mathbf{W}^{(l)}$ consists of the weight coefficients of the individual neurons for the respective neurons from the previous layer and the weight weight $\mathbf{V}^{(l)}$ of the weight coefficients to the same layer of the previous index. As explained before, within the framework of recurrent neural networks, the same parameters exist across the indices t, in particular the weight coefficients $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$ as well as the depth of the network in the form of the number of hidden layers L and the number of neurons k_l in each hidden layer l.

$$o_{t,i} = \sigma\left(b_i^{(O)} + \sum_{j \in \Gamma_i^{-1}} w_{ij}^{(O)} h_{t,j}^{(L)}\right) \quad , \quad \mathbf{o}_t = \sigma\left(\mathbf{W}^{(O)} \mathbf{h}_t^{(L)} + \mathbf{b}^{(O)}\right) \tag{31}$$

where $\sigma(\cdot)$ represents the activation function analogously to the formulas 16 and 23, $\xi_t^{(i)}$ represents the net input of the hidden layers, respectively the output layer , as well as W_{ho} and B_o the vectors of the weight coefficients and biases/thresholds. In comparison to the "vanilla" multi-layer perceptron from the form 17, it shows that the basic structure is the same, except that there is a recurrent structure within the hidden layers, whereby not only the output values

of the predecessors layers serve as input , but also the output values of the "same" hidden layer at index $t-1$. This reflects exactly the peculiarity of RNNs compared to MLPs, whereby information from the hidden layers of the prior index $t-1$ flows into the activation of the hidden units in the hidden layers with index t, resulting in sequential input data and their characteristics can be processed better compared to classic MLPs. With a look back at the formula 30, we now have to define the weight coefficient vector and the bias/thresholds vectors:

In the course of the previous derivations and the basis of the "vanilla" multi-layer perceptron, which only differs in the recurrent structure within the framework of the net input, the following general structure of (deep) recurrent neural networks can be visualized and formalized:

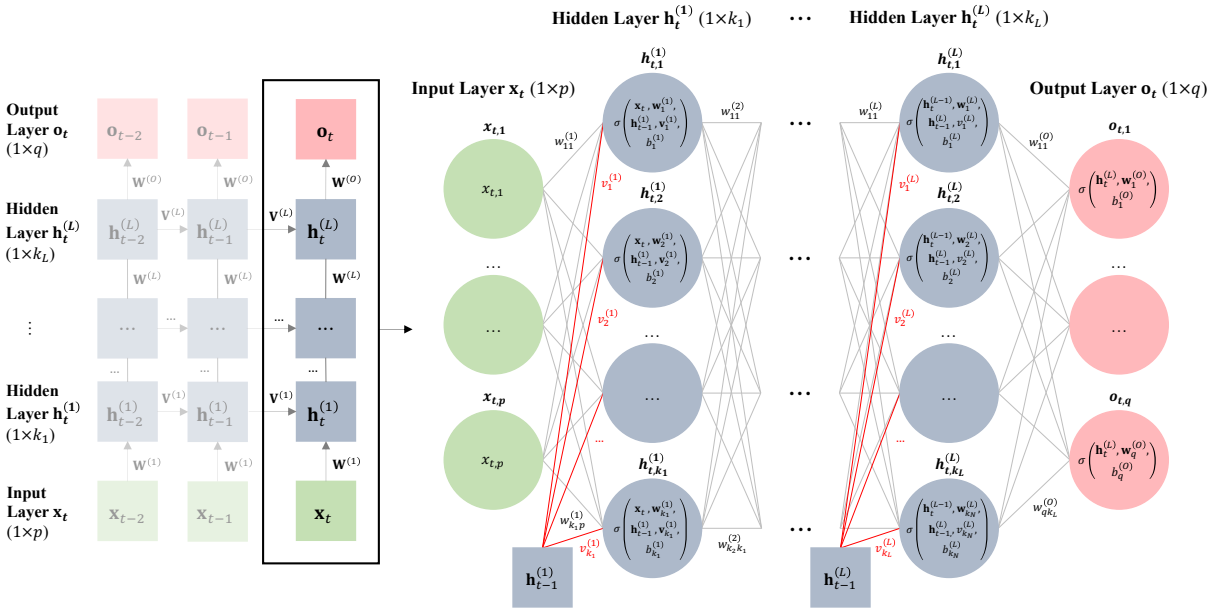


Figure 5: Recurrent Neural Network Architecture - Own Illustration

$$
\begin{aligned}
\text{Input Layer:} \quad & \mathbf{x}_t \in \mathbb{R}^{1\times p} && \\
\text{Hidden Layer(s):} \quad & \mathbf{h}_t^{(l)} \in \mathbb{R}^{1\times k_l} && \forall\, l \in \{1,...,L\} \\
\text{Output Layer:} \quad & \mathbf{o}_t \in \mathbb{R}^{1\times q} && \\
\text{Input Neurons:} \quad & \mathbf{x}_t = \mathbf{x}_t && \\
\text{Hidden Neurons:} \quad & \mathbf{h}_t^{(l)} = \sigma\left(\xi_t^{(l)}\right) && \forall\, l \in \{1,...,L\} \\
& \xi_t^{(l)} = \mathbf{W}^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)} && \forall\, l \in \{1,...,L\} \\
\text{Output Neurons:} \quad & \mathbf{o}_t = \sigma\left(\xi_t^{(O)}\right) && \\
& \xi_t^{(O)} = \mathbf{W}^{(O)}\mathbf{h}_t^{(L)} + \mathbf{b}^{(O)} && \forall\, l \in \{1,...,L\} \\
\text{Weight:} \quad & \mathbf{w}_i^{(l)} \in \mathbf{W}^{(l)} \in \mathbb{R}^{k_{l-1}\times k_l} && \forall\, i \in \{1,...,k_l\} \\
& \mathbf{v}_i^{(l)} \in \mathbf{V}^{(l)} \in \mathbb{R}^{k_{l-1}\times k_l} && \forall\, i \in \{1,...,k_l\} \\
\text{Bias:} \quad & \mathbf{b}^{(l)} \in \mathbb{R}^{1\times k_l} && \forall\, i \in \{1,...,k_l\}
\end{aligned}
\tag{32}
$$

where $\mathbf{z}_t^{(l-1)} \in \left\{\mathbf{x}_t, \mathbf{h}_t^{(1)}, ..., \mathbf{h}_t^{(L-1)}\right\}$ refers to the previous layer at the same time step and either consists of the input layer or the previous hidden layer, depending on the location of the hidden layer l at time. While the general model architecture in Figure 5 and formula (32 applies to arbitrary models with regard to the input, hidden and output layers, there are different types of recurrent neural networks that have evolved over the course of the existence of the input and output layers for the various time steps t and can be visualized as follows:

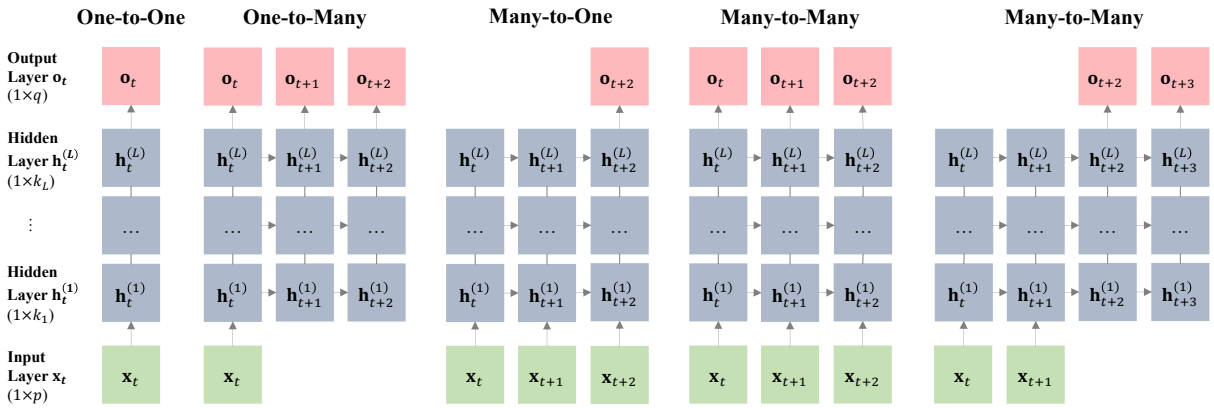


Figure 6: Recurrent Neural Network Types - Own Illustration (based on IBM, 2023)

This general overview of a recurrent neural network summarizes the sufficient model components. Compared to the definition of a multi-layer perceptron, it shows to what extent the recurrent element of the additional information flow between hidden layers of different time steps are embedded into the overall model architecture. On top of that, due to the recurrence, the optimization does not follow the same back-propagation is examined in the multi-layer perceptron, but consists of an adapted optimization algorithm which will be derived in the following section.

Back-Propagation Through Time (BPTT)

The *Back-Propagation Through Time* (BTT) algorithm represents the adaptation of the back-propagation from classic multi-layer perceptrons to recurrent neural networks. As already elaborated before, the central challenge lies in the recurrent mechanisms, where the hidden layers at time t no longer only depend on the previous layer, but also on the same-level hidden layer of the previous time step $t-1$. In order to solve for recurrence, an *unfolding* in time of the recurrent neural network takes place, which is consistent with the visualization in Figure 5 and 6 while formalizing a MLP-like network for each time step t under the identical weight matrices $\mathbf{W}^{(l)}$ and $\mathbf{V}^{(l)}$ (Gruslys et al., 2016, p.2). Thus the network is broken up for the individual time steps t and can then be treated by back-propagation as for the multi-layer perceptron with tied weights (Gruslys et al., 2016, p.2). As a starting point, analogous to MLPs, there is a deterministic loss function that quantifies the loss through the estimation on the basis of the predicted outputs and the true values:

$$C(\mathbf{O}, \mathbf{Y}) = \sum_{t=1}^{T} C_t(\mathbf{o}_t, \mathbf{y}_t) \tag{33}$$

where $\mathbf{o}_t$ and $\mathbf{y}_t$ represent the vectors of predicted output and true values, respectively, which are summed up over all time steps t up to T to form the total cost/loss of the network. As in the context of the derived back-propagation under gradient descent for multi-layer perceptrons, it is necessary to derive the gradient descent of the cost function, taking into account the time indices, analogously to the following formalization:

$$-\nabla C_t(\mathbf{o}_t, \mathbf{y}_t) \tag{34}$$

Based on the definition of the deterministic cost function and the gradient descent as well as the *forward pass* derived in (32), the *backward pass* in context of the BPTT can be derived. In the first step, based on the gradient descent definition in (34), the optimal weights and biases are formally defined as follows:

$$\begin{aligned}
\mathbf{W}^{(l)} &= \mathbf{W}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{W}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\} \\
\mathbf{V}^{(l)} &= \mathbf{V}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{V}^{(l)}} \qquad \forall\, l \in \{1, ..., L\} \\
\mathbf{b}^{(l)} &= \mathbf{b}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{b}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\}
\end{aligned} \tag{35}$$

where $\mathbf{W}^{(l)} \in \left\{\mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)}, \mathbf{W}^{(O)}\right\}$ represents the weight matrix at any layer l, $\mathbf{V}^{(l)} \in \left\{\mathbf{V}^{(1)}, ..., \mathbf{V}^{(L)}\right\}$ the weight matrix between same level hidden layers between any consecutive time steps t and $t-1$ and $\mathbf{b}^{(l)} \in \left\{\mathbf{b}^{(1)}, ..., \mathbf{b}^{(L)}, \mathbf{b}^{(O)}\right\}$ the vector of biases in each layer l. Similar to the derivation in context of the MLP, the central challenge is given by deriving the first order derivatives with respect to these parameters. Hence, in a first step, the first order derivative of

the cost function with respect to any hidden state $h_t^{(l)}$ at time t and layer l as well as the output is defined as follows:

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{W}^{(l)}} &= \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{W}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\} \\
\frac{\partial C}{\partial \mathbf{V}^{(l)}} &= \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{V}^{(l)}} \qquad \forall\, l \in \{1, ..., L\} \\
\frac{\partial C}{\partial \mathbf{b}^{(l)}} &= \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{b}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\}
\end{aligned} \tag{36}$$

where the first order derivative of the overall cost function represents the sum of the individual first order derivatives over all time steps t, in accordance with formula (33). As a next step, one can apply the *chain rule* which, analogously to the MLP derivations, represents a central component in deriving the gradients of the cost function with respect to the model parameters. Thus, the formulas in (36) can further be broken up as follows:

$$\begin{aligned}
\frac{\partial C_t}{\partial \mathbf{W}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{z}_t^{(l)}} \frac{\partial \mathbf{z}_t^{(l)}}{\partial \mathbf{W}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\} \\
\frac{\partial C_t}{\partial \mathbf{V}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{V}^{(l)}} \qquad \forall\, l \in \{1, ..., L\} \\
\frac{\partial C_t}{\partial \mathbf{b}^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{z}_t^{(l)}} \frac{\partial \mathbf{z}_t^{(l)}}{\partial \mathbf{b}^{(l)}} \qquad \forall\, l \in \{1, ..., L, O\}
\end{aligned} \tag{37}$$

where $\mathbf{z}_t^{(l)} \in \left\{ \mathbf{h}_t^{(1)}, ..., \mathbf{h}_t^{(L)}, \mathbf{o}_t \right\}$ defines the vector of output values in the hidden layers and the output layer. Based on these results, the second term in form of the first order derivative of the hidden layer $\mathbf{h}_t^{(l)}$ with respect to the weight and bias parameters can be further elaborated under the chain rule as follows:

$$\begin{aligned}
\frac{\partial \mathbf{z}_t^{(l)}}{\partial \mathbf{W}^{(l)}} &\overset{\text{chain rule}}{=} \sigma' \left(\xi_t^{(l)} \right) \mathbf{z}_t^{(l-1)} \qquad \forall\, l \in \{1, ..., L, O\} \\
\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{V}^{(l)}} &\overset{\text{chain rule}}{=} \sigma' \left(\xi_t^{(l)} \right) \mathbf{h}_{t-1}^{(l)} \qquad \forall\, l \in \{1, ..., L\} \\
\frac{\partial \mathbf{z}_t^{(l)}}{\partial \mathbf{b}^{(l)}} &\overset{\text{chain rule}}{=} \sigma' \left(\xi_t^{(l)} \right) \qquad \forall\, l \in \{1, ..., L, O\}
\end{aligned} \tag{38}$$

where $\xi_t^{(l)} \in \left\{ \xi_t^{(1)}, ..., \xi_t^{(L)}, \xi_t^{(O)} \right\}$ and the first order derivatives of the activation function are in accordance with formula (23). In a next step, the first term of (37) as the first order derivative of the cost function $C_t(\cdot)$ with respect to the hidden layer $\mathbf{h}_t^{(l)}$ can be derived as follows:

$$\begin{aligned}
\frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l+1)}} \frac{\partial \mathbf{h}_t^{(l+1)}}{\partial \mathbf{h}_t^{(l)}} + \frac{\partial C_t}{\partial \mathbf{h}_{t+1}^{(l)}} \frac{\partial \mathbf{h}_{t+1}^{(l)}}{\partial \mathbf{h}_t^{(l)}} = \frac{\partial C_t}{\partial \mathbf{h}_t^{(l+1)}} \mathbf{W}^{(l+1)\mathrm{T}} + \frac{\partial C_t}{\partial \mathbf{h}_{t+1}^{(l)}} \mathbf{V}^{(l)\mathrm{T}} \\
\frac{\partial C_t}{\partial \mathbf{z}_t^{(l)}} &\overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{z}_t^{(l+1)}} \frac{\partial \mathbf{z}_t^{(l+1)}}{\partial \mathbf{z}_t^{(l)}} + \frac{\partial C_t}{\partial \mathbf{z}_{t+1}^{(l)}} \frac{\partial \mathbf{z}_{t+1}^{(l)}}{\partial \mathbf{z}_t^{(l)}} = \frac{\partial C_t}{\partial \mathbf{z}_t^{(l+1)}} \mathbf{W}^{(l+1)\mathrm{T}} + \frac{\partial C_t}{\partial \mathbf{z}_{t+1}^{(l)}} \mathbf{V}^{(l)\mathrm{T}}
\end{aligned} \tag{39}$$

where $\mathbf{h}_{t+1}^{(l-1)}$ represents the hidden layer $l-1$ at time $t+1$ and $\mathbf{h}_{t+1}^{(l)}$ the hidden layer l at time $t+1$. Referring back to formula (37), the second terms can further be elaborated under the general notations for the forward pass of neural networls according to formula (32):

Since both terms of the underlying formula (37) are properly defined, it allows propagate back the error through the entire network, starting from the output layer through the hidden layers back to the input layer. The central methodology of back-propagating the error through the network is equivalent to the back-propagation derived under multi-layer perceptrons, whereby the the additional weight matrix $\mathbf{V}^{(l)}$ as well as the addition of costs across the different time steps t according to formula (33)represent additional algorithmic components in the course of back-propagation through time for recurrent neural networks.

Vanishing/Exploding Gradient Problem

In the context of recurrent neural networks, vanishing or exploding gradients represent a key problem in training the network through BPTT, while the same issues applies to the alternative optimization algorithm of real-time recurrent learning (RTRL) (Schmidt, 2019, p.3). As the names suggest, on the one hand, the *vanishing gradient* problem describes the fact that the gradients of the cost function $C(\cdot)$ with respect to the model parameters $\mathbf{W}^{(l)}$, $\mathbf{V}^{(l)}$ and/or $\mathbf{b}^{(l)}$ become too small and thus vanish. Hence, long term components go very fast to norm 0, which prevents the model to learn correlation between temporally distant events (Pascanu et al., 2013, p.2-3). On the other hand, the *exploding gradient* problem expresses the opposite problem of gradients becoming too big while experiencing a large increase in the norm of the gradient (Hochreiter and Schmidhuber, 1997, p.1; Pascanu et al., 2013, p.2-3). In the case of exploding gradients, it turns out in the training process that the parameters to be optimized oscillate between extreme values and do not converge along the iterations, while in the case of vanishing gradients the training process takes a very long time or is not even feasible (Hochreiter and Schmidhuber, 1997, p.1).

In the course of deriving the vanishing/exploding gradients, the central components of the BPTT algorithm in the form of the gradient descent learning rule and the hidden layer definition must first be recapitulated:

$$\begin{aligned}
\mathbf{h}_t^{(l)} &= \sigma\left(\mathbf{W}^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}^{(l)}\right) \\
\mathbf{V}^{(l)} &= \mathbf{V}^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{V}^{(l)}}
\end{aligned} \tag{40}$$

where $\mathbf{z}_t^{(l-1)} \in \left\{\mathbf{x}_t, \mathbf{h}_t^{(1)}, ..., \mathbf{h}_t^{(L-1)}\right\}$. In addition, the gradient component $\frac{\partial C}{\partial \mathbf{V}^{(l)}}$ has to be further derived in order to be able to examine the dynamics within the gradient descent within the

framework of the BPTT, with the following Notations consistent with the previous derivation of the general BPTT and the proofs by Pascanu et al. (2013, p.2):

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{V}^{(l)}} &\overset{\text{chain rule}}{=} \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{h}_k^{(l)}} \frac{\partial \mathbf{h}_k^{(l)}}{\partial \mathbf{V}^{(l)}} \\
\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{h}_k^{(l)}} &\overset{\text{chain rule}}{=} \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j^{(l)}}{\partial \mathbf{h}_{j-1}^{(l)}} = \prod_{j=k+1}^{t} \mathbf{V}^{(l)\mathrm{T}} \cdot \operatorname{diag}\left[\sigma'\left(\mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{j-1}^{(l)} + \mathbf{b}^{(l)}\right)\right] \qquad (41) \\
\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{V}^{(l)}} &\overset{\text{chain rule}}{=} \sigma'\left(\mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{j-1}^{(l)} + \mathbf{b}^{(l)}\right) \cdot \mathbf{h}_k^{\mathrm{T}}
\end{aligned}$$

where the first equation follows from the chain rule, the second equation also from the chain rule and the transformation into the product from Jacobian matrices (sum-of-products form) and finally the third equation from the chain rule using the definition of the hidden layer from equation (40), in accordance with Pascanu et al. (2013, p.2). The last step consists of inserting the derived terms from equation (41) into the gradient descent rule from equation (40) in order to examine the effects on the resulting learning rule with regard to vanishing and exploding gradients:

$$\begin{aligned}
\frac{\partial C_t}{\partial \mathbf{V}^{(l)}} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \left(\prod_{j=k+1}^{t} \mathbf{V}^{(l)\mathrm{T}} \cdot \operatorname{diag}\left[\sigma'\left(\mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{j-1}^{(l)} + \mathbf{b}^{(l)}\right)\right]\right) \cdot \\
\sigma'\left(\mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{j-1}^{(l)} + \mathbf{b}^{(l)}\right) \cdot \mathbf{h}_k^{\mathrm{T}}
\end{aligned} \qquad (42)$$

As already evident from the above equation in the context of the BPTT, the component $\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{h}_k^{(l)}}$ is the central object for investigation. The reason is that it takes the form of the product of $t-k$ Jacobian matrices (see equation (41)), where that product equivalent to the product of $t-k$ real numbers shrinks to zero or explodes to infinity (Pascanu et al., 2013, p.2). Pascanu et al. (2013, p.2) show under the assumption $||\operatorname{diag}(\sigma'(\odot))|| \leq \gamma$ that the sufficient condition for *vanishing gradients* is given by $\lambda_1^{(l)} < \frac{1}{\gamma}$, where $\lambda_1^{(l)}$ represents the largest absolute eigenvalue of the recurrent weight coefficient matrix $\mathbf{V}^{(l)}$. Analogously, it follows that $\lambda_1^{(l)} < \frac{1}{\gamma}$ serves as a sufficient condition for *exploding gradients*. Thus, vanishing/exploding gradients represent a fundamental problem of RNNs, which makes the optimization of the parameters by BPTT (as well as RTRL) fundamentally more difficult or even impossible while learning long-term dependencies across the network becomes infeasible at certain distances.

3.1.3 Long Short-Term Memory Neural Network (LSTM)

Model Architecture

Long Short-Term Memory (LSTM) neural networks are an adapted architecture of the recurrent neural network, which are focused on the processing of sequential data while mainly being based on the problem of vanishing or exploding gradients and thus the enablement of learning

of long-range dependencies (Hochreiter and Schmidhuber, 1997, p.1). The central component and difference to RNNs are described by a *constant error carousel* (CEC), which allows a constant error flow between special, self-connected units and thus prevent vanishing or exploding gradients (Hochreiter and Schmidhuber, 1997, p.6; Staudemeyer and Morris, 2019, p.19). Such CEC is embodied by a self-connected, linear unit j (Hochreiter and Schmidhuber, 1997, p.6). Additionally, on the one hand, an *input gate unit* is introduced which controls the amount of information flowing through from the same time step's previous layer as well as the previous time step's same layer and hence protects the memory contents in j from perturbation by irrelevant inputs (Hochreiter and Schmidhuber, 1997, p.6). On the other hand, a multiplicative *output gate unit* protects other units, in particular the same time step's next layer as well as the next time step's same layer, from perturbation by irrelevant memory contents stored in j (Hochreiter and Schmidhuber, 1997, p.6). In addition, state-of-the-art LSTMs include an additional *forget gate*, which was not part of the initial structure proposed by Hochreiter & Schmidhuber (1997), while it has been introduced by Gers et al. (2000) and is included in today's vanilla LSTM architectures (Van Houdt et al., 2020, p.5931) since it allows the unit to reset itself at appropriate times. Thus, the basic architecture allows each LSTM unit to remember values over arbitrary time intervals and to regulate the flow of information as well as its own state through the input, output and forget states (Van Houdt et al., 2020, p.5931). The resulting unit under these components is called *memory block* and represents the basic LSTM unit between input and hidden, hidden and output as well as between hidden layers themselves (Hochreiter and Schmidhuber, 1997, p.6; Gers et al., 2000, p.2). The following figure visualizes that memory block and its relevant information gates:

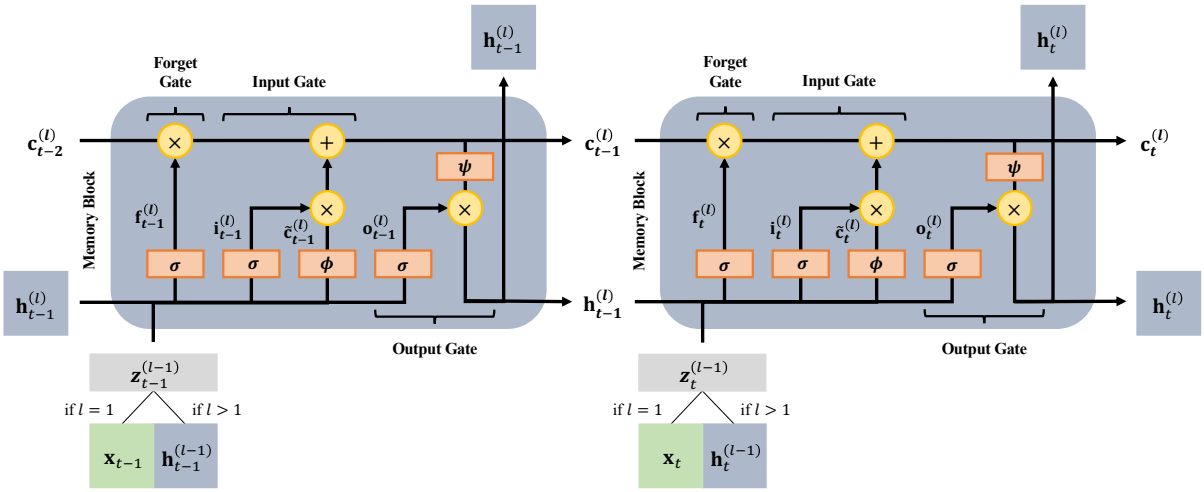


Figure 7: LSTM Memory Block - Own Illustration (based on Gers et al., 2000, p.5)

$$
\begin{aligned}
\text{Forget Gate:} \quad & \mathbf{f}_t^{(l)} = \sigma\left(\mathbf{W}_f^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
\text{Input Gate:} \quad & \mathbf{i}_t^{(l)} = \sigma\left(\mathbf{W}_i^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_i^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_i^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
\text{Output Gate:} \quad & \tilde{\mathbf{o}}_t^{(l)} = \sigma\left(\mathbf{W}_o^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_o^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_o^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
\text{Block Input:} \quad & \tilde{\mathbf{c}}_t^{(l)} = \phi\left(\mathbf{W}_c^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
\text{Memory Cell:} \quad & \mathbf{c}_t^{(l)} = \mathbf{f}_t^{(l)} \odot \mathbf{c}_{t-1}^{(l)} + \mathbf{i}_t^{(l)} \odot \tilde{\mathbf{c}}_t^{(l)} && \forall\, l \in \{1, ..., L\} \\
\text{Block Output:} \quad & \mathbf{h}_t^{(l)} = \tilde{\mathbf{o}}_t^{(l)} \odot \psi\left(\mathbf{c}_t^{(l)}\right) && \forall\, l \in \{1, ..., L\}
\end{aligned}
\tag{43}
$$

where $\mathbf{z}_t^{(l-1)} \in \left\{\mathbf{x}_t, \mathbf{h}_t^{(1)}, ..., \mathbf{h}_t^{(L-1)}\right\}$ refers to the previous layer at the same time step and either consists of the input layer or the previous hidden layer, depending on the location of the memory block. According to Figure 7 and equations (43), $\mathbf{f}_t^{(l)}$, $\mathbf{i}_t^{(l)}$ and $\tilde{\mathbf{o}}_t^{(l)}$ represent the activated outputs of the forget, input and output gate, respectively. They themselves depend on the outputs of the same time step's previous layer $\mathbf{z}_t^{(l-1)}$ weighted by $\mathbf{W}_f^{(l)}$, $\mathbf{W}_i^{(l)}$, and $\mathbf{W}_o^{(l)}$ as well as the previous time step's same layer $\mathbf{h}_{t-1}^{(l)}$ weighted by $\mathbf{V}_f^{(l)}$, $\mathbf{V}_i^{(l)}$ and $\mathbf{V}_o$ which allows the direct notation comparison with the architecture of RNNs. In addition, the bias parameters $\mathbf{b}_f^{(l)}$, $\mathbf{b}_i^{(l)}$ and $\mathbf{b}_o^{(l)}$ allow for linearly offsetting the input of each gate. Overall, the net input is activated by the activation function $\sigma(\cdot)$ which usually implies the *sigmoid* or *tanh* function according to equation (16) (Van Houdt et al., 2020, p.5932). Hence, through this architecture, the forget, input and output gate can directly control the degree of information from outside of the memory block, in particular the previous network layers. On top of that, the memory block consists of a block input, the value of the memory cell and the block output. Similar to the gates, the block input $\tilde{\mathbf{c}}_t^{(l)}$ is also calculated from the previous layers weighted by $\mathbf{W}_c^{(l)}$ and $\mathbf{V}_c^{(l)}$ and offset by $\mathbf{b}_c^{(l)}$ while being activated by ϕ which usually represents the *tanh* function. The value of the memory, which defines the core of the memory block, is defined based on the outer product of the forget gate $\mathbf{f}_t^{(l)}$ and the memory cell $\mathbf{c}_{t-1}^{(l)}$ provided by the same layer of the previous time $t-1$ as well as the outer product of input gate $\mathbf{i}_t^{(l)}$ and the block input $\tilde{\mathbf{c}}_t^{(l)}$. Lastly, the block output, which derives the value vector of the layer that the memory block refers to, is derived from the outer product of the output gate $\mathbf{o}_t^{(l)}$ and the value of the memory cell $\mathbf{c}_t^{(l)}$ activated by the activation function $\psi(\cdot)$, which indeed often refers to the *tanh* function according to figure 1 and equation (16) (Van Houdt et al., 2020, p.5932).

As can already be seen from the derivation of the memory blocks, LSTMs represent a special architecture of RNNs, with the basic structure of the feed-forward information flows between layers and the recurrent information flows between the sequential time steps being identical. Thus, the following network layers can be defined analogously:

$$
\begin{aligned}
\text{Input:} \quad & x_{t,p} \in \mathbf{x}_t \in \mathbb{R}^{1\times p} \\
\text{Hidden:} \quad & h_{t,i}^{(l)} \in \mathbf{h}_t^{(l)} \in \mathbb{R}^{1\times k_l} \\
\text{Output:} \quad & o_{t,q} \in \mathbf{o}_t \in \mathbb{R}^{1\times q}
\end{aligned}
\tag{44}
$$

with t as the time index, p the number of covariates and k_l the number of hidden neurons in each layer l (Schmidt, 2019, p.1). Similar to the vanilla RNN architecture, the number of layer-specific hidden neurons is constant across all time steps. In addition, however, there are fundamental differences with regard to the weights, since there are no longer just two weight matrices between the previous layer and the same layer of the previous time step, but also the weights of the memory blocks that determine the degree of information in the forget gate, input gate and the output gate as well as the block input itself (Van Houdt et al., 2020, p.5932-5933). With reference to the derivation of the memory block in equation (43), the following formalization results:

$$\begin{aligned}
\text{Forget Gate:} \quad & w_{ij,f}^{(l)} \in \mathbf{w}_{i,f}^{(l)} \in \mathbf{W}_f^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \\
& v_{ij,f}^{(l)} \in \mathbf{v}_{i,f}^{(l)} \in \mathbf{v}_f^{(l)} \in \mathbb{R}^{k_l \times k_l} \\
\text{Input Gate:} \quad & w_{ij,i}^{(l)} \in \mathbf{w}_{i,i}^{(l)} \in \mathbf{W}_i^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \\
& v_{ij,i}^{(l)} \in \mathbf{v}_{i,i}^{(l)} \in \mathbf{v}_i^{(l)} \in \mathbb{R}^{k_l \times k_l} \\
\text{Output Gate:} \quad & w_{ij,o}^{(l)} \in \mathbf{w}_{i,o}^{(l)} \in \mathbf{W}_o^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \\
& v_{ij,o}^{(l)} \in \mathbf{v}_{i,o}^{(l)} \in \mathbf{v}_o^{(l)} \in \mathbb{R}^{k_l \times k_l} \\
\text{Input Block:} \quad & w_{ij,c}^{(l)} \in \mathbf{w}_{i,c}^{(l)} \in \mathbf{W}_c^{(l)} \in \mathbb{R}^{k_{l-1} \times k_l} \\
& v_{ij,c}^{(l)} \in \mathbf{v}_{i,c}^{(l)} \in \mathbf{v}_c^{(l)} \in \mathbb{R}^{k_l \times k_l} \\
\text{Output Layer:} \quad & w_{ij}^{(O)} \in \mathbf{w}_i^{(O)} \in \mathbf{W}^{(O)} \in \mathbb{R}^{k_{l-1} \times q}
\end{aligned} \tag{45}$$

where the weight matrices $\mathbf{W}_f^{(l)}$, $\mathbf{W}_i^{(l)}$, $\mathbf{W}_o^{(l)}$ and $\mathbf{W}_c^{(l)}$ contain the weight coefficients of the hidden layer l to the previous hidden layer $l-1$ or input layer $\mathbf{x}$, depending on the location of the memory block, while the weight matrices $\mathbf{V}_f^{(l)}$, $\mathbf{V}_i^{(l)}$, $\mathbf{V}_o^{(l)}$ and $\mathbf{V}_c^{(l)}$ contain the weight coefficients of hidden layer l with the same hidden layer l at the previous index $t-1$. It should be noted here that any weight parameters are the same across the indices. In addition, there are also differences to the vanilla RNN in the bias vectors with regard to the architecture of the memory block, whereby these can be summarized as follows in the context of the LSTM:

$$\begin{aligned}
\text{Forget Gate:} \quad & b_{i,f}^{(l)} \in \mathbf{b}_f^{(l)} \in \mathbb{R}^{1 \times k_l} \\
\text{Input Gate:} \quad & b_{i,i}^{(l)} \in \mathbf{b}_i^{(l)} \in \mathbb{R}^{1 \times k_l} \\
\text{Output Gate:} \quad & b_{i,o}^{(l)} \in \mathbf{b}_o^{(l)} \in \mathbb{R}^{1 \times k_l} \\
\text{Input Block:} \quad & b_{i,c}^{(l)} \in \mathbf{b}_c^{(l)} \in \mathbb{R}^{1 \times k_l} \\
\text{Output Layer:} \quad & b_i^{(O)} \in \mathbf{b}^{(O)} \in \mathbb{R}^{1 \times y}
\end{aligned} \tag{46}$$

Since the neuron value in the form of the output block and the underlying value of the memory cell as well as the block input and thus the hidden layer have already been fully defined in

equation (43), the last step is to define the output layer. This is done equivalently to the vanilla RNN architecture, where the output layer is derived from the last hidden layer only and can be formalized as follows:

$$\mathbf{o}_t = \sigma\left(\mathbf{W}^{(O)}\mathbf{h}_t^{(L)} + \mathbf{b}^{(O)}\right) \tag{47}$$

After the core components of the LSTM have been derived analogously to the MLP and the RNN, in particular in the form of memory as a central component of the special variant of the vanilla RNN, the general overall structure of the vanilla LSTM can be defined both visually and formally as follows:

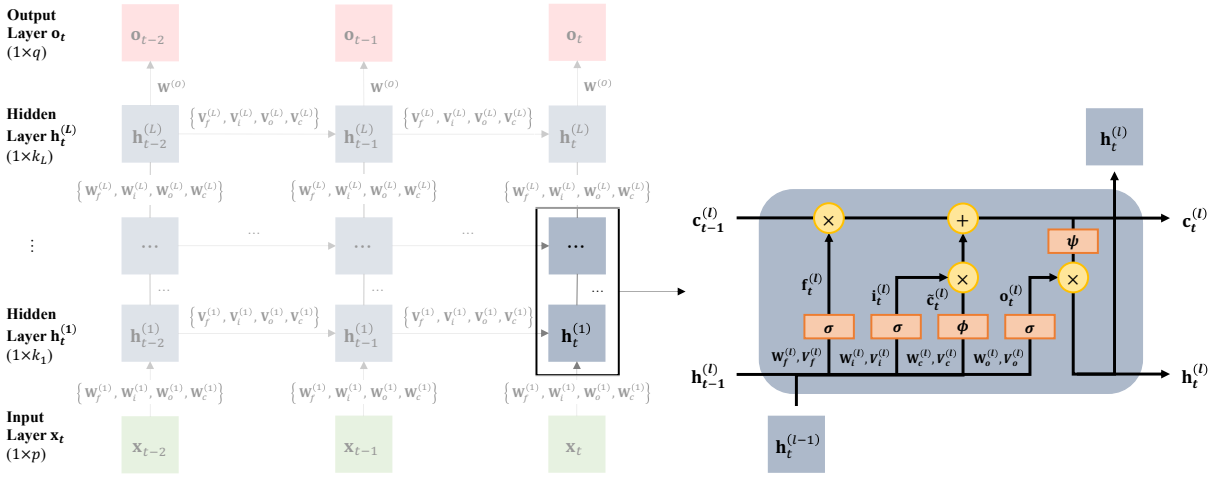


Figure 8: LSTM Architecture - Own Illustration

$$
\begin{aligned}
\text{Input Layer:} \quad & \mathbf{x}_t \in \mathbb{R}^{1 \times p} \\
\text{Hidden Layer(s):} \quad & \mathbf{h}_t^{(l)} \in \mathbb{R}^{1 \times k_l} && \forall\, l \in \{1, ..., L\} \\
\text{Output Layer:} \quad & \mathbf{o}_t \in \mathbb{R}^{1 \times q} \\
\text{Input Neurons:} \quad & \mathbf{x}_t = \mathbf{x}_t \\
\text{Memory Block:} \quad & \mathbf{f}_t^{(l)} = \sigma\left(\mathbf{W}_f^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
& \mathbf{i}_t^{(l)} = \sigma\left(\mathbf{W}_i^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_i^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_i^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
& \tilde{\mathbf{o}}_t^{(l)} = \sigma\left(\mathbf{W}_o^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_o^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_o^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
& \tilde{\mathbf{c}}_t^{(l)} = \phi\left(\mathbf{W}_c^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
& \mathbf{c}_t^{(l)} = \mathbf{f}_t^{(l)} \odot \mathbf{c}_{t-1}^{(l)} + \mathbf{i}_t^{(l)} \odot \tilde{\mathbf{c}}_t^{(l)} && \forall\, l \in \{1, ..., L\} \\
& \mathbf{h}_t^{(l)} = \tilde{\mathbf{o}}_t^{(l)} \odot \psi\left(\mathbf{c}_t^{(l)}\right) && \forall\, l \in \{1, ..., L\} \\
\text{Output Neurons:} \quad & \mathbf{o}_t = \sigma\left(\mathbf{W}^{(O)} \mathbf{h}_t^{(L)} + \mathbf{b}^{(O)}\right) \\
\text{Weight:} \quad & \text{see (45)} \\
\text{Bias:} \quad & \text{see (46)}
\end{aligned}
\tag{48}
$$

After the overall architecture of a vanilla LSTM based on the initial derivations by Hochreiter and Schmidhuber (Hochreiter and Schmidhuber, 1997) and the additional extensions by Gers et al. (2000), the functionality of the constant error carousel with regard to the solution of the vanishing/exploding gradient problem and the network training in the context of back-propagation in time must be evaluated in the following sections.

Back-Propagation Through Time (BPTT)

LSTMs represent a special architecture of RNNs, with which the *back-propagation through time* (BPTT) learning algorithm can be used analogously to RNNs. It is only important to derive the back-propagation for the memory block, as this represents the central difference to vanilla RNN. Thus, the general *cost function* and the basic learning rule under the *gradient descent* method must be recapitulated and defined for the additional parameters of the LSTM. While the learning rule for the parameters in the output layer is equivalent to the derivations within the framework of the RNN according to equation (35)-(39), the following explanations focus on the memory blocks in the hidden layers:

$$
C(\mathbf{O}, \mathbf{Y}) = \sum_{t=1}^{T} C_t(\mathbf{o}_t, \mathbf{y}_t) \tag{49}
$$

$$\begin{aligned}
\mathbf{W}_j^{(l)} &= \mathbf{W}_j^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{W}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\} \\
\mathbf{V}_j^{(l)} &= \mathbf{V}_j^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{V}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\} \\
\mathbf{b}_j^{(l)} &= \mathbf{b}_j^{(l)} - \lambda \frac{\partial C}{\partial \mathbf{b}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\}
\end{aligned} \tag{50}$$

where j represents the index for the (recurrent) weight matrices and bias vectors for the forget gate, input gate, output gate and the input block, while the learning rules from equation (50) follow from the general definition $-\nabla C_t(\mathbf{o}_t, \mathbf{y}_t)$ of the gradient descent. Based on the statements in equation (49) and (50), the first order derivatives result with respect to the weight matrices and the bias vectors can be expressed as the following sum over the indexed cost functions:

$$\frac{\partial C}{\partial \mathbf{W}_j^{(l)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{W}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\} \tag{51}$$

$$\frac{\partial C}{\partial \mathbf{V}_j^{(l)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{V}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\} \tag{52}$$

$$\frac{\partial C}{\partial \mathbf{b}_j^{(l)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial \mathbf{b}_j^{(l)}} \qquad \forall\, j \in \{f, i, o, c\} \quad \forall\, l \in \{1, ..., L\} \tag{53}$$

where the first order derivative of the overall cost function represents the sum of the individual first order derivatives over all time steps t, in accordance with equation (49). Analogously to the derivations in the context of RNNs, the first order derivatives can be defined as follows:

$$\frac{\partial C_t}{\partial \mathbf{W}_j^{(l)}} \overset{\text{chain rule}}{=} \begin{cases} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{f}_t^{(l)}} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{W}_f^{(l)}}, & \text{if } j = f \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{i}_t^{(l)}} \frac{\partial \mathbf{i}_t^{(l)}}{\partial \mathbf{W}_i^{(l)}}, & \text{if } j = i \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{o}}_t^{(l)}} \frac{\partial \tilde{\mathbf{o}}_t^{(l)}}{\partial \mathbf{W}_o^{(l)}}, & \text{if } j = o \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{c}}_t^{(l)}} \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{W}_c^{(l)}}, & \text{if } j = c \end{cases} \qquad \forall\, l \in \{1, ..., L\} \tag{54}$$

$$\frac{\partial C_t}{\partial \mathbf{V}_j^{(l)}} \overset{\text{chain rule}}{=} \begin{cases} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{f}_t^{(l)}} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{V}_f^{(l)}}, & \text{if } j = f \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{i}_t^{(l)}} \frac{\partial \mathbf{i}_t^{(l)}}{\partial \mathbf{V}_i^{(l)}}, & \text{if } j = i \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{o}}_t^{(l)}} \frac{\partial \tilde{\mathbf{o}}_t^{(l)}}{\partial \mathbf{V}_o^{(l)}}, & \text{if } j = o \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{c}}_t^{(l)}} \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{V}_c^{(l)}}, & \text{if } j = c \end{cases} \qquad \forall\, l \in \{1, ..., L\} \tag{55}$$

$$\frac{\partial C_t}{\partial \mathbf{b}_j^{(l)}} \overset{\text{chain rule}}{=} \begin{cases} \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{f}_t^{(l)}} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{b}_f^{(l)}}, & \text{if } j = f \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{i}_t^{(l)}} \frac{\partial \mathbf{i}_t^{(l)}}{\partial \mathbf{b}_i^{(l)}}, & \text{if } j = i \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{o}}_t^{(l)}} \frac{\partial \tilde{\mathbf{o}}_t^{(l)}}{\partial \mathbf{b}_o^{(l)}}, & \text{if } j = o \\ \frac{\partial C_t}{\partial \mathbf{h}_t^{(l)}} \frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{c}}_t^{(l)}} \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{b}_c^{(l)}}, & \text{if } j = c \end{cases} \quad \forall\, l \in \{1, ..., L\} \tag{56}$$

Now it is necessary to derive the corresponding terms for equations (54)-(56) in order to finally derive the first order derivative term for the gradient descent updating rules from equation (50). The last term in the form of the first order derivatives of the various gate values and the block input values with respect to the according (recurrent) weight matrices and bias vectors are calculated as follows:

$$\begin{aligned} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{W}_f^{(l)}} &= \sigma' \left(\mathbf{W}_f^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)} \right) \mathbf{z}_t^{(l-1)\mathrm{T}} \quad \forall\, l \in \{1, ..., L\} \\ &\vdots \\ \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{W}_c^{(l)}} &= \phi' \left(\mathbf{W}_c^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)} \right) \mathbf{z}_t^{(l-1)\mathrm{T}} \quad \forall\, l \in \{1, ..., L\} \end{aligned} \tag{57}$$

$$\begin{aligned} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{V}_f^{(l)}} &= \sigma' \left(\mathbf{W}_f^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)} \right) \mathbf{h}_{t-1}^{(l)\mathrm{T}} \quad \forall\, l \in \{1, ..., L\} \\ &\vdots \\ \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{V}_c^{(l)}} &= \phi' \left(\mathbf{W}_c^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)} \right) \mathbf{h}_{t-1}^{(l)\mathrm{T}} \quad \forall\, l \in \{1, ..., L\} \end{aligned} \tag{58}$$

$$\begin{aligned} \frac{\partial \mathbf{f}_t^{(l)}}{\partial \mathbf{b}_f^{(l)}} &= \sigma' \left(\mathbf{W}_f^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)} \right) \quad \forall\, l \in \{1, ..., L\} \\ &\vdots \\ \frac{\partial \tilde{\mathbf{c}}_t^{(l)}}{\partial \mathbf{b}_c^{(l)}} &= \phi' \left(\mathbf{W}_c^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)} \right) \quad \forall\, l \in \{1, ..., L\} \end{aligned} \tag{59}$$

In addition, the second term now applies in the form of the first order derivatives of the hidden states or the input states $\mathbf{z}_t^{(l)}$ with respect to the various gate values and the block input values to derive:

$$\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{f}_t^{(l)}} = \left(\tilde{\mathbf{o}}_t^{(l)} \odot \psi \left(\mathbf{c}_t^{(l)} \right) \right) \odot \mathbf{c}_t^{(l-1)} \odot \sigma' \left(\mathbf{W}_f^{(l)} \mathbf{z}_t^{(l-1)} + \mathbf{V}_f^{(l)} \mathbf{h}_{t-1}^{(l)} + \mathbf{b}_f^{(l)} \right) \quad \forall\, l \in \{1, ..., L\} \tag{60}$$

$$\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{i}_t^{(l)}} = \left(\tilde{\mathbf{o}}_t^{(l)} \odot \psi\left(\mathbf{c}_t^{(l)}\right)\right) \odot \tilde{\mathbf{c}}_t^{(l)} \odot \sigma'\left(\mathbf{W}_i^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_i^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_i^{(l)}\right) \quad \forall\, l \in \{1, ..., L\} \tag{61}$$

$$\frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{o}}_t^{(l)}} = \psi\left(\mathbf{c}_t^{(l)}\right) \odot \sigma'\left(\mathbf{W}_o^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_o^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_o^{(l)}\right) \quad \forall\, l \in \{1, ..., L\} \tag{62}$$

$$\frac{\partial \mathbf{h}_t^{(l)}}{\partial \tilde{\mathbf{c}}_t^{(l)}} = \left(\tilde{\mathbf{o}}_t^{(l)} \odot \psi\left(\mathbf{c}_t^{(l)}\right)\right) \odot \mathbf{i}_t^{(l)} \odot \phi'\left(\mathbf{W}_c^{(l)}\mathbf{z}_t^{(l-1)} + \mathbf{V}_c^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{b}_c^{(l)}\right) \quad \forall\, l \in \{1, ..., L\} \tag{63}$$

The last step is to derive the first and thus the final of the three terms of equation (54):

$$\frac{\partial \mathbf{C}_t^{(l)}}{\partial \mathbf{h}_t^{(l)}} \overset{\text{chain rule}}{=} \frac{\partial C_t}{\partial \mathbf{h}_{t+1}^{(l-1)}} \frac{\partial \mathbf{h}_{t+1}^{(l-1)}}{\partial \mathbf{h}_t^{(l+1)}} + \frac{\partial C_t}{\partial \mathbf{h}_t^{(l+1)}} \frac{\partial \mathbf{z}_{t+1}^{(l)}}{\partial \mathbf{z}_t^{(l)}} \quad \forall\, l \in \{1, ..., L\} \tag{64}$$

Since all three terms of the underlying equation (54) are properly defined, it allows to propagate the error back through the entire LSTM network, starting from the output layer through the hidden layers back to the input layer. While the fundamental methodology of back-propagating the error through the network is equivalent to the back-propagation derived under RNNs, the additional (recurrent) weight matrices as well as the bias vectors from the memory block represent additional algorithmic components in the course of back-propagation through time for long short-term memory neural networks.

Constant Error Carousel (CEC)

As can already be seen from the accompanying explanations, the special architecture of the LSTM in the form of the memory block represents the basis of the constant error carousel and thus the vanishing/exploding gradient problem of recurrent neural networks, with which the LSTM architecture allows long- to learn term dependencies. As already shown in the explanations in the context of the vanishing/exploding gradient problem, the central problem with regard to the gradients results from the following BPTT component in the form of the product of Jacobian matrices:

$$\frac{\partial \mathbf{h}_t^{(l)}}{\partial \mathbf{h}_k^{(l)}} \overset{\text{chain rule}}{=} \prod_{j=k+1}^{t} \frac{\partial \mathbf{h}_j^{(l)}}{\partial \mathbf{h}_{j-1}^{(l)}} = \prod_{j=k+1}^{t} \mathbf{V}^{(l)\mathrm{T}} \cdot \operatorname{diag}\left[\sigma'\left(\mathbf{W}^{(l)}\mathbf{z}_j^{(l-1)} + \mathbf{V}^{(l)}\mathbf{h}_{j-1}^{(l)} + \mathbf{b}^{(l)}\right)\right] \tag{65}$$

where the product of $t-k$ Jacobian matrices shrinks or explodes equivalently to the product of $t-k$ real numbers whereby the direction depends on the value of the largest absolute eigenvalue of the recurrent weight matrix (Pascanu et al., 2013, p.2). In contrast, the LSTM architecture has the additional component of the memory block and in particular the memory cell state $\mathbf{c}_t^{(l)}$, with the following first order derivative gives:

$$\frac{\partial \mathbf{c}_t^{(l)}}{\partial \mathbf{c}_{t-1}^{(l)}} = \mathbf{f}_t^{(l)} \quad , \quad \frac{\partial \mathbf{c}_t^{(l)}}{\partial \mathbf{c}_k^{(l)}} = \prod_{j=k+1}^{t} \mathbf{f}_j^{(l)} \tag{66}$$

where the first order derivative of $\mathbf{c}_t^{(l)}$ with respect to any arbitrary, previous cell state at time $k < t$ is defined by the product of the forget gates between $k+1$ and t. Since the definition of the forget gate implies that the forget values are bounded on $[0, 1]$, the gradient with respect to the cell state gets propagated through time (across the time steps t) without majorly vanishing or exploding. This leads to the ability of LSTMs to maintain a relatively constant gradient across time and hence to learn long-term dependencies more effectively compared to vanilla RNNs.

3.2 Transformer Neural Network

As can already be seen from the previous section, recurrent neural networks (RNN) and long short-term memory neural networks (LSTM) have become central models in machine learning applications in the context of sequence modelling exposed in financial markets research. In addition, in recent years, based on the initial publication by Vaswani et al. (2017) established another class of neural networks in the field of deep learning, namely *Transformers*. The architecture of transformer neural networks discussed below also originates from language modelling and machine translation and to date represents one of the most advanced innovations in this regard and is finding ever broader applications in research and industries. Hence, the following chapter provides an overview of the general model approach as well as a detailed examination of its architecture as a foundation for optimization regarding time series and cross-sectional momentum applications.

3.2.1 Motivation

For a long time, long short-term memory (LSTM) networks and gated recurrent unit (GRU) networks represented the state-of-the-art approaches in terms of application to sequence modelling tasks, especially in the areas of language modelling and machine translation, whereby these were also of great importance for financial applications in terms of time series processing (Vaswani et al., 2017, p.1). Nevertheless, fundamental challenges and limitations result from the basic architectures in the family of recurrent neural networks: On the one hand, the *sequential architecture* allows hidden references to the previous layers at time t as well as hidden recurrent references to the same-level layers at time $t-1$ that the network can also only be trained sequentially and excludes the parallelization of the training processes (Vaswani et al., 2017, p.2). Depending on the sequence length, network depth and batch sizes (size of training samples), this ensures that the training process becomes inefficient and requires long training times (Vaswani et al., 2017, p.1). In the course of the development of increasing computational parallelizations, the goal arises that this is also used in the context of deep sequence models. In this context, Vaswani et al. (Vaswani et al., 2017, p.6-7) show that the general recurrent structure leads to increased complexity per layer, sequential operations as well as a higher maximum path length

compared to the self-attention architecture in transformer networks. On the other hand, in the course of research and further development of sequence modelling networks of various RNNs, it is found that the architecture is also reaching its *limits* in terms of performance and accuracy and that LSTMs also do not fully enable learning of long-term dependencies (Vaswani et al., 2017, p.2). Ultimately, *attention mechanisms* have also been established in recent years, which enable sequence modelling and the learning of dependencies independently of the respective (temporal) distance, whereby those were mainly used in RNNs and thus entail the previous, equivalent limitations (Vaswani et al., 2017, p.2). All those shortcomings and limitations of recurrently structured neural networks represent the basic motivation that emerged from the development of transformer neural networks and revolutionized the possibilities in the field of sequence modelling.

3.2.2 General Model Architecture

The general "vanilla" transformer architecture has initially been introduced by Vaswani et al. (2017) with the focus on improving the aforementioned shortcomings of recurrent neural networks in the context of sequence-to-sequence modelling tasks. Even though the vanilla structure is not sufficient for the application on CSMOM and TSMOM tasks, the fundamental understanding of the general architecture represents a crucial step in order to adjust the model components. Generally, the *encoder* and *decoder* components fundamentally incorporate the basic architecture of transformers, whereby the encoder maps an input sequence of tokens $\mathbf{x} = (x_1, ..., x_n)$ to a continuous representation $\mathbf{Z} = (\mathbf{z}_1, ..., \mathbf{z}_n)$, after which the decoder takes $\mathbf{Z}$ in order to generate the targeted output sequence $\mathbf{o} = (o_1, ..., o_m)$ (Vaswani et al., 2017, p.2). The central reason for this encoder-decoder architecture is the model's original focus on sequence-to-sequence tasks in the area of natural language processing (NLP) and machine translation, with the input being given as a sequence of characters or words (= tokens), which require to be transformed to continuous vectors by the encoder and to be transformed back into a sequence of characters by the decoder based on that continuous representation in order to allow for information processing.

A look at the encoder and decoder components shows that there is a model-internal preprocessing of the input in the form of *input embedding* and *output embedding*, respectively, and *positional encoding*. However, the difference here is that the encoder takes the input sequence itself as input and the decoder sequentially takes the true outcomes in case of model training and the predicted outputs in case of model testing one element at a time, whereby in both cases the sequences are shifted by one to the are right (Vaswani et al., 2017, p.2). Hence, the model and in particular the decoder incorporates an auto-regressive structure by consuming previously generated outputs as additional input when generating the next output (Vaswani et al., 2017, p.2). This also means that the encoder can process all inputs at once and the decoder can only generate the model outputs sequentially. Those processing steps of embedding and positional encoding pursue the purpose of the continuous representation of the tokenized input sequence, whereby each token is transformed into a continuous vector. The continuous representations then flow into the actual *encoder stack* and *decoder stack*, respectively, whereby those components are stacked N times on top of each other. In the case of the encoder stack, this results

in the *Multi-Head Self-Attention* (MHA), a *Fully Connected Feed-Forward Network* (FNN) and a subsequent *Add & Norm* component for residual connections or layer normalizations. The output of the last encoder stack N finally represents the output in the form of the processed continuous representation $\mathbf{Z} = (\mathbf{z}_1, ..., \mathbf{z}_n)$, which consequently serves as an (additional) input for the decoder. In contrast, the decoder stack consists of a *Masked Multi-Head Self-Attention* (MMHA) at the beginning, followed by an MHA and an FNN equivalent to the encoder stack, whereby the MHA layer, however, uses the encoder output $\mathbf{Z}$ takes up. In addition, the decoder stacks are followed by an additional *linear transformation* and an activation component in the form of the *softmax* function, which ultimately represents the *output probabilities* on the interval $[0, 1]$. Based on those output probabilities over the total set of possible outputs, the decoder finally generates the output sequence $\mathbf{o}$ in terms of the sequential tokens o_t.

Based on the high-level explanations of the encoder and decoder as well as their central components, the following architecture of "vanilla" transformers results based on the initial statements by Vaswani et al. (2017):

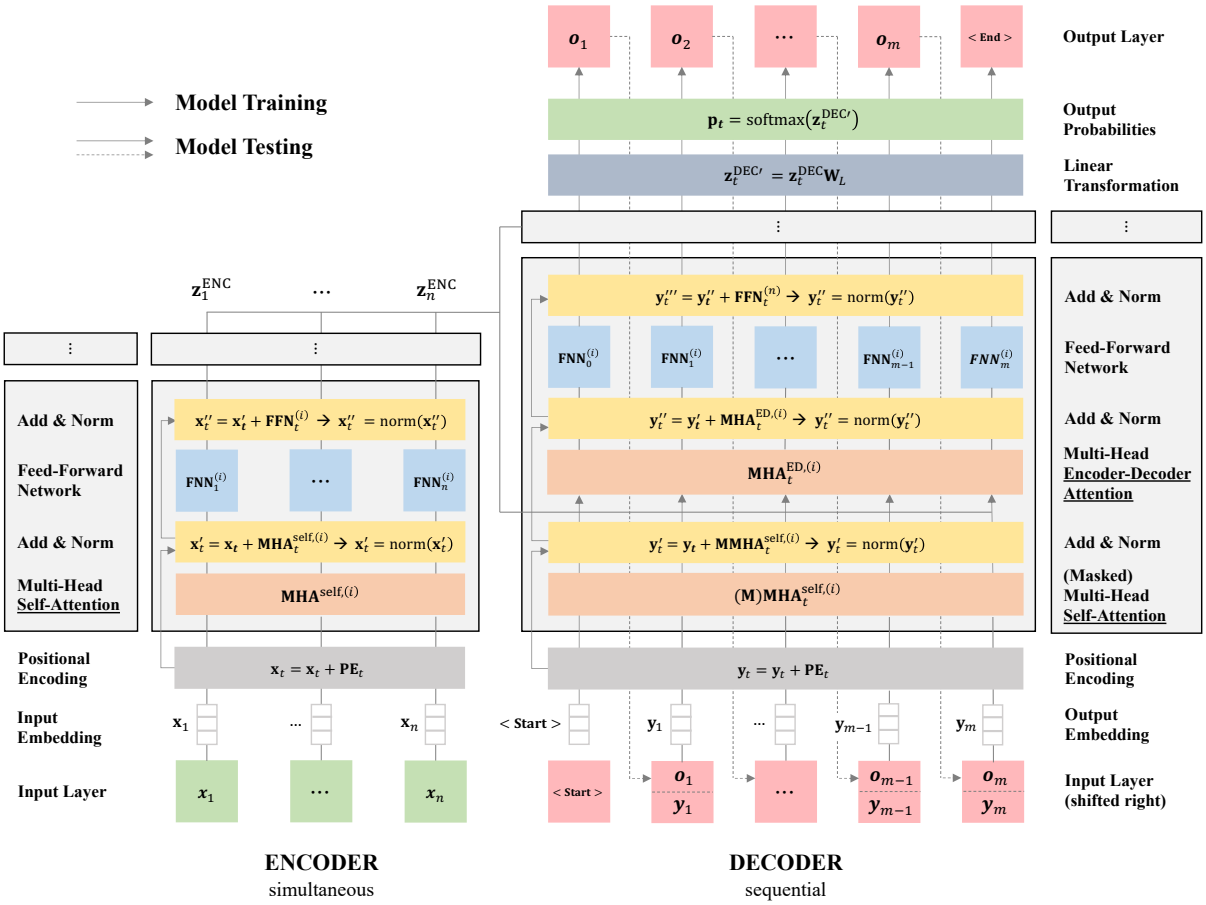


Figure 9: Transformer Architecture - Own Illustration (based on Vaswani et al., 2017, p.3)

After the high-level components of the transformer network have been introduced, it is necessary to derive the individual components fundamentally in order to complement the overall understanding of the transformers. This is done by looking at the encoder and decoder components in the following sections.

3.2.2.1 Input Embedding & Positional Encoding (PE)

Analogously to the explanations in the context of MLPs, RNNs and LSTMs, the *input layer* represents the starting point of the model in the context of the encoder, with transformer networks taking up an arbitrary input sequence $\mathbf{x} = (x_1, ..., x_n)$. That input sequence can accommodate arbitrary lengths $n \leq 2$, whereby the elements can take different forms. While the input sequence consists of characters or words in the main areas of application of the transformer, there are equivalent areas of application with continuous or discrete input sequences. Especially with regard to the processing of text inputs, the first step is to tokenize the input sequence, through which the input sequence is transformed into smaller units, so-called *tokens*. Hence, an arbitrary *input tokenization* can be formalized as follows:

$$\text{Encoder Input:} \quad \mathbf{x} \longrightarrow (x_t)_{t=1}^{n} \tag{67}$$

where $\mathbf{x}$ represents the original input sequence and $(x_t)_{t=1}^{n}$ the tokenized sequence with corresponding tokens x_t at position $1 \leq t \leq n$. After the input sequence has been tokenized, the elements of that sequence $(x_t)_{t=1}^{n}$ each need to be transformed into vectors of continuous representations. This is necessary because transformer networks cannot efficiently handle and process character-based sequences and are dependent on numeric, continuous representations. The *embedding matrix* $\mathbf{E}$ serves as the basis of that input embedding, whereby that matrix can take on different forms and is learned during the model training, while fundamental embedding models are given by Word2Vec (Mikolov et al., 2013) or Time2Vec (Kazemi et al., 2019). The formalization of that embedding matrix appears as follows:

$$\mathbf{E} = \begin{bmatrix} \mathbf{E}_1 \\ \vdots \\ \mathbf{E}_v \end{bmatrix} = \begin{bmatrix} E_{1,1} & \cdots & E_{1,d_{model}} \\ \vdots & \ddots & \vdots \\ E_{v,1} & \cdots & E_{v,d_{model}} \end{bmatrix} \in \mathbb{R}^{v \times d_{model}} \tag{68}$$

where d_{model} represents a hyperparameter that handles the trade-off between the ability of learning complex information structures and computational efficiency, while each vector $\mathbf{E}_i$ represents a vector of continuous representation. In order to map the tokenized inputs to the input embedding matrix, the individual tokens must first be mapped to their unique IDs $i \in \{1, ..., v\}$, whereupon the vector $\mathbf{E}_i$ of the embedding matrix serves as a continuous representation of the token x_t. That mapping in the context of *input embedding* can finally be formalized as follows:

$$\text{Input Embedding:} \quad x_t \longrightarrow \mathbf{x}_t = \mathbf{E}_i \in \mathbb{R}^{1 \times d_{model}} \tag{69}$$

where $\mathbf{x}_t$ defines the continuous representation vector of the token x_t derived from the embedding matrix $\mathbf{E}$. Since the transformer network does not contain any recurrence or convolution structures, a method must be introduced that provides information about the absolute or relative position of the tokens x_t in the form of the continuous representation $\mathbf{x}_t$ with reference to the input sequence $\mathbf{x}$ (Vaswani et al., 2017, p.5-6). This is done using the *positional encoding*,

which can be defined as a fixed or learnable rule (2017, p.6). Hereby, Vaswani et al. (2017, p.6) proposes a positional encoding under the sine and cosine function for different frequencies, which can be formalized as follows:

$$\text{Positional Encoding:} \quad \mathbf{x}_t = \mathbf{x}_t + \mathbf{PE}_t = \mathbf{x}_t + \begin{cases} \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), & \text{if } 2i = \text{even} \\ \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), & \text{if } 2i+1 = \text{odd} \end{cases} \tag{70}$$

where $0 \leq pos \leq n-1$ represents the index t (0-indexed) of token x_t in relation to the original input sequence $\mathbf{x}$, $0 \leq i \leq \frac{d}{2} - 1$ represents the dimension index of the positional encoding vector and d_{model} the embedding dimension hyperparameter as defined before. Hence, the result positional encoding vector of dimension d_{model} takes the following form:

$$\mathbf{PE}_t = \begin{bmatrix} \sin\left(\frac{pos}{10000^{0/d_{model}}}\right) \\ \cos\left(\frac{pos}{10000^{0/d_{model}}}\right) \\ \sin\left(\frac{pos}{10000^{2/d_{model}}}\right) \\ \cos\left(\frac{pos}{10000^{2/d_{model}}}\right) \\ \vdots \\ \sin\left(\frac{pos}{10000^{(d_{model}-2)/d_{model}}}\right) \\ \cos\left(\frac{pos}{10000^{(d_{model}-2)/d_{model}}}\right) \end{bmatrix}^{\mathrm{T}} \tag{71}$$

This method of positional encoding ensures that every continuous representation and thus indirectly every token of the input sequence receives a position "labeling", whereby the relative position of the token within the transformer network can be determined.

3.2.2.2 (Masked) Multi-Head Self-Attention ((M)MHA)

The central method for recording inter-dependencies within the input sequence and thus between the tokens of the input sequence in the context of transformer networks is countered by the concept of *self-attention*. As already emerged from the motivation, the central advantages of self-attention compared to recurrence or convolution mechanisms consist on the one hand in computational efficiency and, especially with regard to RNNs and LSTMs, in the ability of capturing long-term dependencies without any loss of information. The foundation of the calculation of the self-attention and thus the attention between the tokens are represented by three central matrices:

i The *query* represents a matrix in order to determine the relevance of each token with respect to each other token from the input sequence

ii The *keys* represents a matrix with the characteristics of each token in order to match those against the query matrix.

iii The *values* represent a matrix that contains the current value/content of each token.

All matrices consist of the vectors with respect to each token, namely $\mathbf{q}_t$, $\mathbf{k}_t$ as well as $\mathbf{v}_t$, which are multiplied by the learnable weight matrices $\mathbf{W}^Q$, $\mathbf{W}^K$ and $\mathbf{W}^V$, while, analogously, the matrices are derived by multiplying the stacked vectors of continuous input representations $\mathbf{x}_t$ to the input matrix $\mathbf{X}$ with the corresponding weight matrices. Hence, the queries, keys and values components can be generally formalized and visualized as follows:

$$\begin{aligned} \text{Query:} \quad & \mathbf{q}_t = \mathbf{x}_t \mathbf{W}^Q \in \mathbb{R}^{1 \times d_k} \quad , \quad \mathbf{Q} = \mathbf{X}\mathbf{W}^Q \in \mathbb{R}^{n \times d_k} \\ \text{Keys:} \quad & \mathbf{k}_t = \mathbf{x}_t \mathbf{W}^K \in \mathbb{R}^{1 \times d_k} \quad , \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K \in \mathbb{R}^{n \times d_k} \\ \text{Values:} \quad & \mathbf{v}_t = \mathbf{x}_t \mathbf{W}^V \in \mathbb{R}^{1 \times d_v} \quad , \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V \in \mathbb{R}^{n \times d_v} \end{aligned} \tag{72}$$

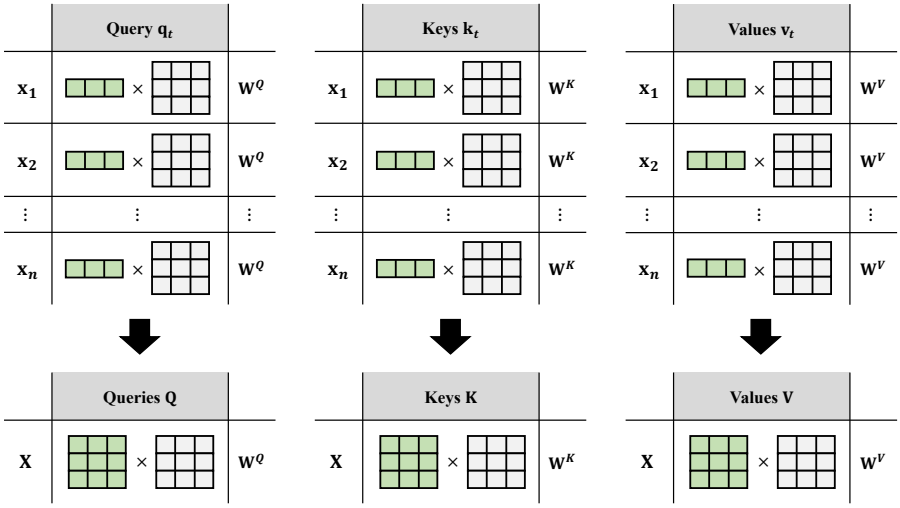


Figure 10: Transformer Queries, Keys, Values - Own Illustration

The calculation of the self-attention based on the queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$ is based on the *attention function*, where the *additive attention* and *dot-product attention* serve as state-of-the-art functions. Vaswani et al. (2017, p.4) supplement the dot-product addition with a scaling factor, which is considered a kind of regularization term in the course of optimizing computational efficiency. Self-attention based on the scaled dot-product attention function is formalized as follows:

$$\text{Self-Attention:} \quad \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\mathbf{V} \tag{73}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ represent the matrices of queries, keys and values with dimensions d_k and d_v, respectively. When decomposing the structure of the self-attention, the following key components can be derived:

i The multiplication between queries $\mathbf{Q}$ and keys $\mathbf{K}$, given by $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$, returns a matrix of *attention/alignment scores* for each query-key pair and hence implicitly for each token pair

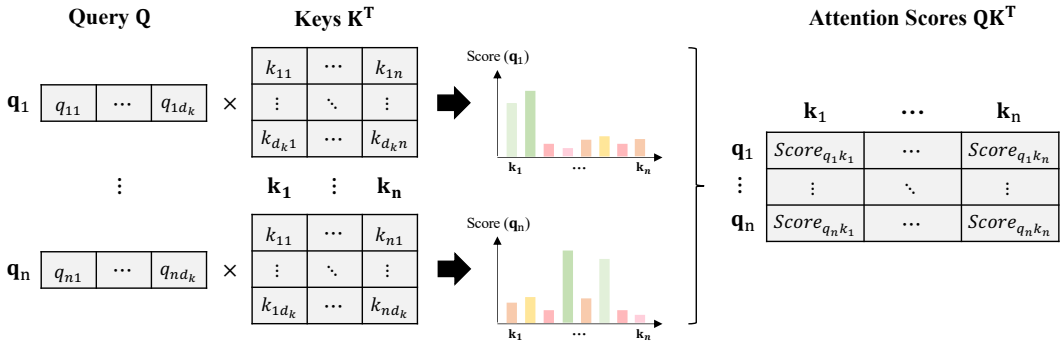


Figure 11: Transformer Attention Scores - Own Illustration

ii The scaling by the square root of the key dimension d_k, given by $\frac{1}{\sqrt{d_k}}$, fulfills the purpose of a *regularization* component in order to increase the computational efficiency of especially large and hence complex transformers

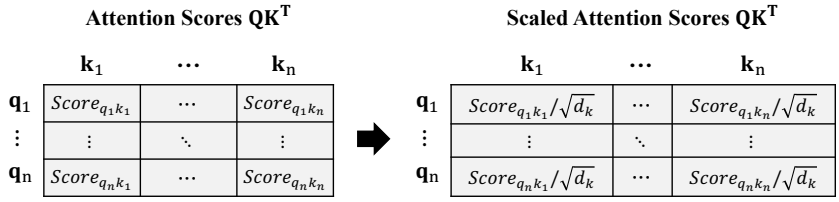


Figure 12: Transformer Scaled Attention Scores - Own Illustration

iii The application by the softmax$(\cdot)$ function to the scaled alignment scores transforms them into *attention/alignment weights* on the interval $[0, 1]$ such that the weights for each query sum up to 1

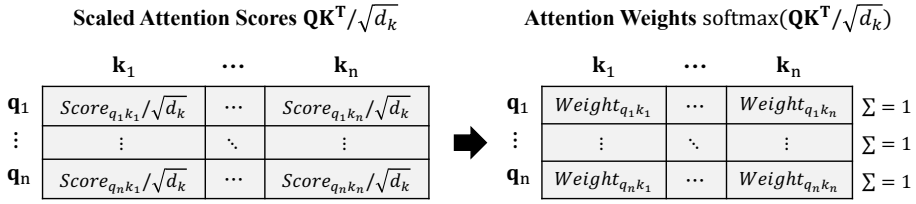


Figure 13: Transformer Attention Weights - Own Illustration

iv The product of the value matrix $\mathbf{V}$ and the attention weights, given by softmax$(\cdot)\mathbf{V}$, returns the *self-attention* vector for each query as a linear combination of the initial continuous representation

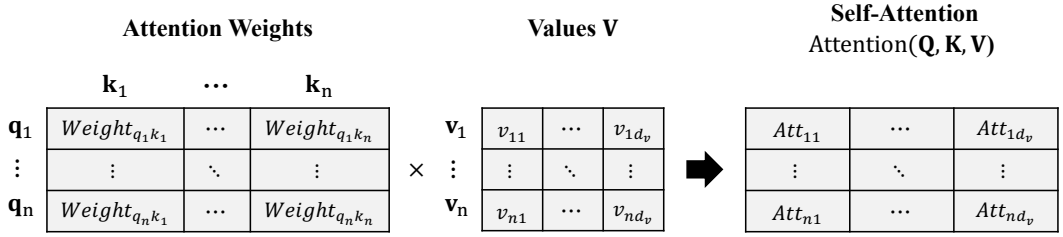


Figure 14: Transformer Self-Attention - Own Illustration

Based on the self-attention mechanism, Vaswani et al. (2017, p.4-5) additionally propose an

extension, namely *multi-head attention*, which allows for multivariate attention and dependencies of each token with respect to the other tokens. Thus, this allows the transformer network to learn a context-like understanding of the token within the input sequence, where more complex relationships can be identified. Especially with regard to the processing of text sequences, where the language contains complex comprehension connections, this is of central importance, whereby multivariate relationships can also be assumed in other areas of application. Thus, in accordance with Vaswani et al. (2017, p.5), *multi-head self-attention* can be formalized as follows:

$$\begin{aligned} \text{Multi-Head Self-Attention:} \quad \mathbf{MHA}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \text{concat}\left(\text{head}_1,...,\text{head}_h\right)\mathbf{W}^O \quad , \\ \text{head}_i &= \text{Attention}\left(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V\right) \end{aligned} \tag{74}$$

where the weight matrices imply the dimensions $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $\mathbf{W}_i^O \in \mathbb{R}^{hd_v \times d_{model}}$ with h being the number of parallel attention layers ($= heads$) and $d_k = d_v = d_{model}/h$. Hence, the consolidation of single attention heads to the final multi-head self-attention as the output of the MHA layer consists of the following logic:

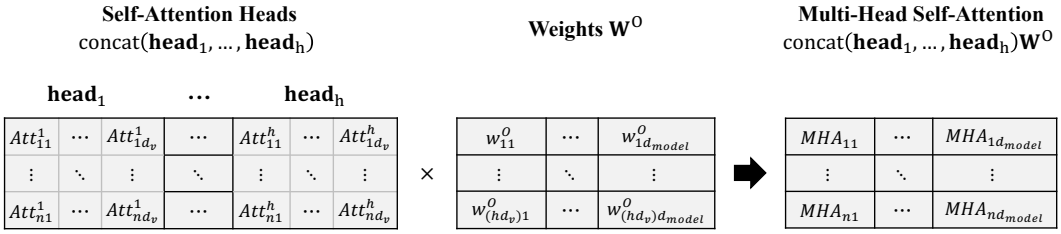


Figure 15: Transformer Multi-Head Self-Attention - Own Illustration

Following the derivations of the multi-head self-attention and the underlying attention mechanism, the output matrix $\mathbf{MHA}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right) \in \mathbb{R}^{n \times d_{model}}$ contains the multivariate dependency structure of the input tokens which can be considered as a context structure across the input sequence. Finally, in addition to Figure 9, the full structure can be summarized as follows:

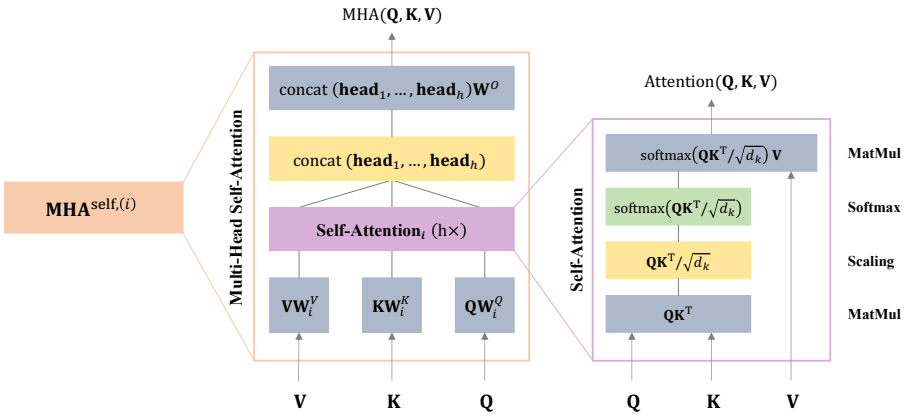


Figure 16: Transformer MHA - Own Illustration (based on Vaswani et al., 2017)

While the concept of general multi-head self-attention was previously described, which is used in both the encoder and decoder, looking back at the general model architecture, there is a further

definition in the form of *masked multi-head self-attention* . Here, an additional mask M is added to the previous multi-head self-attention definition from (73) as an additive component within the softmax function. Such mask ensures that self-attention layers in the decoder allow each position in the decoder to only attend to all positions in the decoder up to and including that position, which prevents leftward information flow in the decoder to preserve the auto-regressive property (cite[p.5]Vaswani2017). Implicitly, given the ordered input sequences in the decoder, prevents a look-ahead bias during the output generation. Hence, the masked self-attention can be formalized with reference to (73) as follows:

$$\text{Masked Self-Attention:} \quad \text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V} \tag{75}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ again represent the matrices of queries, keys and values with dimensions d_k and d_v, respectively with the same key components as the self-attention and the mask M in addition to preserve the autoregressive structure. Accordingly to the derivations in (74), the masked multi-head self-attention for context-like relationship interpretation can thus be defined as follows:

$$\begin{aligned} \text{Masked Multi-Head Self-Attention:} \quad \text{MMHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{concat}\left(\text{head}_1, ..., \text{head}_h\right)\mathbf{W}^O \quad , \\ \text{head}_i &= \text{MaskedAttention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right) \end{aligned} \tag{76}$$

where $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $\mathbf{W}_i^O \in \mathbb{R}^{hd_v \times d_{model}}$ with h as the number of parallel attention layers ($= heads$) and $d_k = d_v = d_{model}/h$, while the look-ahead mask is implicitly provided through the masked attention component derived in (75). Based on this definition, the masked multi-head self-attention can be visualized equivalently to the (unmaksed) multi-head self-attention from figure (16) as follows:

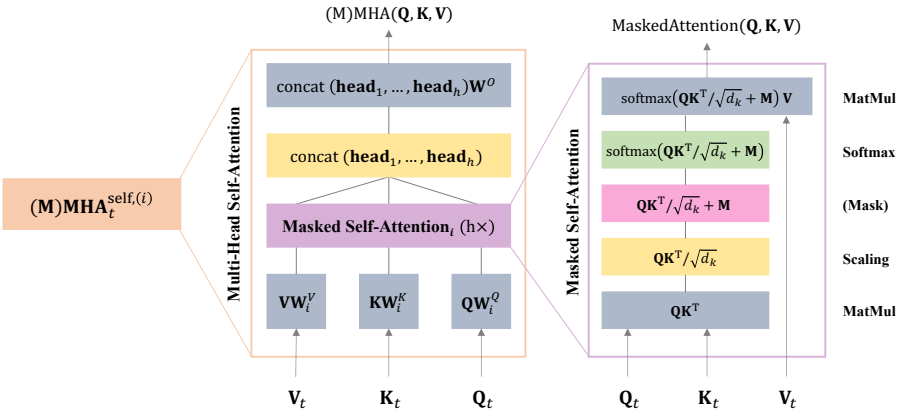


Figure 17: Transformer MMHA - Own Illustration (based on Vaswani et al., 2017)

3.2.2.3 Fully-Connected Feed-Forward Network (FFN)

The second sub-layer of the encoder stack according to Figure 9 represents the *feed-forward network* (FNN), which adds another non-linear layer and thus increased complexity to the transformer network. The architecture of that fully-connected FNN follows the equivalent structure to the explanations on multi-layer perceptrons in Chapter 3.2.1. The parameters of the FNN each amount to an input $(1 \times d_{model})$, hidden $(d_{ff} \times d_{model})$ and output layer $(1 \times d_{model})$ fully-connected by the weight matrices $\mathbf{W}_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ between input and hidden and $\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ between hidden and output layer. The input consists of the output of the multi-head self-attention and add & norm layer (the latter is dealt with in the next paragraph), whereby each row vector $\mathbf{x}_t'$ are fed into separate and identical FNNs simultaneously. Thus, the fully-connected feed-forward network sub layer of the encoder stack can be formalized as follows:

$$\text{Feed-Forward Network:} \quad \text{FFN}(\mathbf{x}) = \mathbf{h}_1^{\text{ReLU}} \mathbf{W}_2 + \mathbf{b}_2 = \max\left(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1\right) \mathbf{W}_2 + \mathbf{b}_2 \tag{77}$$

where the hidden units are activated by the ReLu activation function following the definition in equation (1) and (23). Similarly, the encoder stack component in Figure 9 can be visually broken down as follows:

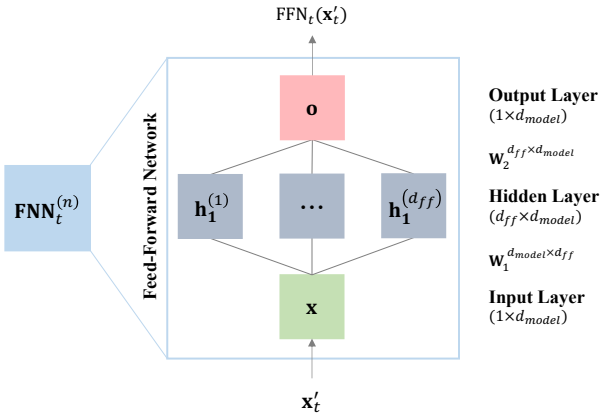


Figure 18: Transformer FFN - Own Illustration (based on Vaswani et al., 2017)

3.2.2.4 Residual Connection (Add) & Layer Normalization (Norm)

Another central component within the encoder stack is given by the *add & norm* layer, which is responsible for the residual connections and layer normalizations. Each layer is introduced separately downstream for multi-head self-attention and the feed-forward network and, on the one hand, combines the MHA and FNN outputs with the upstream continuous input representations and normalizes them on the other hand. The central tasks of those layers are that the outputs of the MHA and FNN are stabilized and, in particular, it represents a facilitation component of the gradient flow as part of the back-propagation in model training, since the back-propagation is only applied to the components of the add & norm layer without directly propagating the errors back within the underlying and more complex MHA and FNN structure. Formally, the residual connections (add) and layer normalizations (norm) can be defined as follows:

$$\begin{aligned} & & \text{Residual Connection (Add)} & & \text{Layer Normalization (Norm)} \\ \text{post-MHA:} & & \mathbf{x}_t^{'} = \mathbf{x}_t + \mathbf{MHA}_t(\mathbf{Q}, \mathbf{K}, \mathbf{V}) & \longrightarrow & \mathbf{x}_t^{'} = \frac{\mathbf{x}_t^{'} + \mu(\mathbf{x}_t^{'})}{\sqrt{\sigma^2(\mathbf{x}_t^{'}) + \epsilon}} \odot \gamma + \beta \\ \text{post-FNN:} & & \mathbf{x}_t^{''} = \mathbf{x}_t^{'} + \mathbf{FNN}_t(\mathbf{x}_t^{'}) & \longrightarrow & \mathbf{x}_t^{''} = \frac{\mathbf{x}_t^{''} + \mu(\mathbf{x}_t^{''})}{\sqrt{\sigma^2(\mathbf{x}_t^{''}) + \epsilon}} \odot \gamma + \beta \end{aligned} \tag{78}$$

where γ and β in the layer normalization represent additional scaling and shifting parameters compared to the traditional normalization to zero mean to standard deviation of one. Both parameters are also learned during the model training and lead to suitable scale and mean for certain underlying transformer tasks.

3.2.2.5 Encoder Output

The fundamental components of the encoder are the encoder stacks, each of which contains the multi-head self-attention and feed-forward network mechanisms with downstream add & norm layers, which have been derived in the previous sections 3.2.2.2, 3.2.2.3 and 3.2.2.4, respectively. Within the framework of the transformer network, those encoder stacks are stacked N times on top of each other, so that the actual continuous input representations after the input embedding and the position encoding from section 3.2.2.1 serve as input in the first encoder stack and the outputs of the upstream stack serve as input in the following encoder stacks. After completing the last encoder stack, where add & norm represents the last sub-layer, the output follows in the form of the following vectors or aggregated matrix:

$$\text{Encoder Output:} \quad \mathbf{z}_t^{\text{ENC}} \in \mathbf{Z}^{\text{ENC}} \in \mathbb{R}^{n^{\text{ENC}} \times d_{model}} \tag{79}$$

where n^{ENC} is defined as the length of the tokenized encoder input sequence and d_{model} as the input embedding dimension. As already seen in the architecture in Figure 9, the output of the encoder is fed into the decoder, whose output is explained and derived in the following section.

3.2.2.6 Decoder Output

Equivalent to the encoder part of the transformer network, the decoder input sequence representations are first generated through an input embedding and a positional encoding layer according to section 3.2.2.1. These representations serve as the input for the decoder stacks, which, analogously to the encoder stack, are stacked N times such that the following decoder stacks receive the input from the output from the previous decoder stacks. Within those stacks, a (masked) multi-head self-attention layer, an add & norm layer, another multi-head self-attention layer, followed by another add & norm layer as well as a fully-connected feed-forward layer and a very final add & norm layer are applied downstream. Whether the first multi-head self-attention layer includes a mask depends on the use case and hence whether the auto-regressive information

structure and prevention of look-ahead bias shall be preserved or not. Another special component is given by the second multi-head self-attention layer, which receives additional input in the form of the encoder output. With the reference to the multi-head self-attention definition from (73) and (74), the keys $\mathbf{K}$ and values $\mathbf{V}$ are derived from the encoder output, while the queries $\mathbf{Q}$ follow downstream from the previous input representation or decoder component after the first decoder stack. Thereby, each vector of the encoder output serves as both a key and a value. Through this architecture, the model allows the output-generating decoder to access relevant information from the processed encoder input. Finally, a linear transformation layer as well as a softmax function follow after the decoder stacks. Due to the property of the softmax function of generating scalars in $[0, 1]$, these outputs are treated as output probabilities. Hence, the decoder output and hence the overall transformer network output can be formalized as follows:

$$\text{Decoder Output:} \quad \mathbf{z}_t^{\text{DEC}} \in \mathbf{Z}^{\text{DEC}} \in \mathbb{R}^{n^{\text{DEC}} \times n^{\text{VOC}}} \tag{80}$$

where n^{DEC} is defined as the length of the tokenized decoder input sequence, n^{VOC} as the size of the corresponding vocabulary and d_{model} as the input embedding dimension. Hence, the output probabilities assign probabilities to each element of the underlying vocabulary, where the token with the highest probability is returned. While transformer networks are often referred to as "next word prediction", this has now been derived conceptually and mathematically from the ground up by assigning output probabilities to each token of the underlying vocabulary.

3.2.3 Application to Numerical Use Cases

While the previously introduced model architecture of the transformer neural network was originally designed for text and language processing tasks, many other areas of application have evolved, especially in the context of numerical use cases. The following explanations provide an overview of the application of transformer networks to numerical inputs with reference to the previously introduced vanilla model architecture from section 3.2 and derive to what extent an adaptation of the model architecture is necessary. The application of transformer neural networks to numerical applications is thereby derived based on the research work of Kisiel & Gorse (2022) as well as Pobrotyn et al. (2021), whereby the statements are generally applicable to other numerical applications under vanilla transformers.

In the course of numerical applications of transformer networks, the input data and shapes must first be defined. In contrast to text and language processing tasks, previously defined *tokens* no longer serve as input elements of each input node, but rather numerical *scalar values* or *vectors*, which means that both the encoder and the decoder obey input vectors or input matrices aggregated all input nodes. This results in the following input notations and shapes (2022, p.5):

$$\begin{aligned} \text{Encoder:} \quad & \mathbf{X}_i^{\text{Enc}} \in \mathbb{R}^{n^{\text{Enc}} \times p} \\ \text{Decoder:} \quad & \mathbf{X}_i^{\text{Dec}} \in \mathbb{R}^{n^{\text{Dec}} \times p} \end{aligned} \tag{81}$$

where p is the number of features and n^{Enc} and n^{Dec} are the number of input nodes of the encoder and decoder, respectively. The subsequent component of the *input embedding* can then be understood as a conversion into a higher-dimensional space, whereby the input scalar or vector is mapped into an embedding vector. Here too, various embedding mechanisms are available, ranging from simple linear projections in the form of linear dense layers up to sophisticated embedding methods like time2vec, whereby both deterministic or learnable embeddings can be applied (2022, p.4). After input embedding, the equivalent setup compared to natural language applications already exists as the numerical embedded vector. With regard to *positional encoding*, however, it should be noted that this step primarily serves to enrich the embedded vector with additional information about the absolute and relative position of the input node. With regard to numerical applications, positional encoding is only necessary if the order of the input nodes is relevant. Because of the *permutation-equivariance* property, which states that the order of the inputs within the self-attention has no influence on the attention calculation itself, which is one of the key motivations for positional encoding, positional encoding is therefore only necessary if the order of the input nodes obeys a meaning (Pobrotyn et al., 2021, p.3). The subsequent *encoder stacks*, however, remain equivalent to text- and language-based applications.

As part of the decoder component of the vanilla transformer network, there are further adjustments regarding the application of numerical inputs. As can be seen from Figure 9, in the course of text- and language-based processing there is both a start-of-sequence token and an end-of-sequence token, which is not used in numerical use cases. This means that all input nodes serve as explicit inputs based on the decoder input scalar or vector $\mathbf{x}_i^{\text{Dec}}$. Furthermore, the input does not take place in the form of the output of the previous step within the framework of the sequential decoder architecture, but rather based on the predefined input vector or the input matrix $\mathbf{X}_i^{\text{Dec}}$. Consequently, after the input layer, the equivalent statements regarding *input embedding* apply, whereby a conversion to a higher-dimensional space is applied. However, with regard to *positional encoding*, there is a difference in that the decoder is designed as a sequential architecture, while the encoder has a simultaneous architecture. Therefore, it is only necessary to design numerical applications including the decoder, which should have sequential outputs, after which the positional encoding is the central component for passing on information about the absolute and relative position within the decoder. Equivalent to text- and language-based, within the context of the *decoder stacks* it is only necessary to decide whether a look-ahead bias should be prevented by the auto-regressive mask as part of the *masked multi-head self-attention*, whereby the rest of the decoder stack for numerical applications does not require any changes. Ultimately, the *softmax* results in an output structure with probability properties. The extent to which this is desirable in the context of numerical applications must be decided on a use case-specific basis, whereby the output layer can be replaced by other functions.

4 OK Transformer-Based Multi-Asset CSMOM & TSMOM Architectures

In the context of the main empirical part of this thesis, the key goal is given the derivation and construction of new deep learning-enhanced CSMOM and TSMOM model architectures. In order to ensure the alignment with existing research and the capturing of the empirical strategy-specific return drivers, generalized CSMOM and TSMOM frameworks are first derived, on top of which the deep learning-enhanced multi-asset CSMOM and TSMOM models are derived and constructed from the ground up based on the vanilla models architectures from section 3. On top of that, state-of-the-art benchmark models are derived for performance comparison, whereby deterministic models as well as deep learning-based models are used.

4.1 OK Generalized CSMOM Strategy Framework

The basic goal of developing deep learning-enhanced multi-asset CSMOM strategies is to incorporate the fundamental concepts and theoretical approaches of cross-sectional strategies and to expand them within the framework of deep learning models so that out-performance in terms of risk-adjusted returns can be achieved robustly over several evaluation periods. The underlying motivation here lies in the extended complexity that can be captured by deep learning models compared to existing benchmark models. The derivation of the proposed deep learning-based model is analogous to the derivation of the benchmark models, which is based on the basic cross-sectional momentum framework in the form of the basic return definition and four framework components that build on each other (2020, p.3).

The arbitrarily applicable definition of a cross-sectional momentum strategy, which is characterized by a portfolio composition based on the relative ranking of assets to one another, can be defined as follows (2020, p.3):

$$r_{t,t+1}^{CSMOM} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{CSMOM}} r_{t,t+1}^{(i)} \tag{82}$$

where $r_{t,t+1}^{CSMOM}$ represents the strategy return from period t to period $t+1$ under n_t assets at time t and $z_t^{(i),\text{CSMOM}} \in [-1, 1]$ represents the asset-specific cross-sectional momentum signal, with a maximum short position of -1, a maximum long position of 1 and no position with size 0, creating an implied leverage restriction. Thus, the return is given as the average return of the underlying portfolio assets using the cross-sectional momentum signal for each asset i. In addition, Poh at al. (2020, p.3) developed a more extensive concept of volatility-scaled returns based on the findings of Kim et al. (2016) through which the return of each asset is scaled by its own volatility in relation to the annual portfolio target volatility. This ensures that each asset has a similar contribution to the overall portfolio return (2020, p.3). The corresponding volatility-normalized CSMOM strategy return can be formalized as follows (2020, p.3):

$$r_{t,t+1}^{CSMOM} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{CSMOM}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{83}$$

where each asset return $r_{t,t+1}^{(i)}$ is factored by the volatility scaling component $\frac{\sigma_{target}}{\sigma_t^{(1)}}$. The latter definition also represents the foundation of the proposed deep-learning-enhanced multi-asset CSMOM model in order to ensure comparability with existing research work. In addition to the return definition, the actual cross-sectional momentum framework for the model derivation, which is dedicated to the definition of the momentum signal $z_t^{(i),\text{CSMOM}}$, is now to be examined. For this purpose, the four central components are defined as follows:

1. Momentum Score Calculation

The first component of the CSMOM framework is the *momentum score calculation*. The momentum score of each asset is based on the input feature vector $\mathbf{u}_t^{(i)}$ and any arbitrary prediction model f to determine the score $s_t^{(i)}$, according to the following equation (2020, p.3):

$$s_t^{(i)} = f(\mathbf{u}_t^{(i)}) \tag{84}$$

where $s_t^{(i)} \in \mathbf{s}_t = s_t^{(1)}, ..., s_t^{(n_t)}$ represents the momentum score of an asset as a member of the momentum score vector of all assets at time t. It is already evident that the definition of the prediction model f is a fundamental variable in the context of momentum strategies and can be defined as a central component of the strategy performance, which can be optimized through state-of-the-art deterministic or machine learning-based models.

2. Momentum Score Ranking

The *momentum score ranking* is the second component of the CSMOM framework, within which the previously calculated momentum score of each asset must be ranked relative to the other assets in the cross-section at time t. To do this, the operator R has to be defined, which sorts the elements $s_t^{(i)}$ in any logical order based on the momentum score vector $\mathbf{s}_t$ (2020, p.3). Hence, the ranking mechanism can be formalized as follows (2020, p.3):

$$\tilde{s}_t^{(i)} = \text{R}(s_t)^{(i)} \tag{85}$$

where $\tilde{s}_t^{(i)} \in \{1, ..., n_t\}$ is the rank of an asset i in the cross-section with the other assets at time t based on the previously calculated momentum scores.

3. Security Selection

The third component of the CSMOM framework consists of the *security selection*, which takes an important position in the context of portfolio re-balancing. To this day, the decile portfolio approach represents a state-of-the-art method, with a long position in the top decile and a short position in the bottom decile of the momentum score ranking (2020, p.3). This is in line with the original remarks by Jegadeesh & Titman (1993). That step of security selection ultimately

represents the momentum signal of each security with reference to the CSMOM return definition from equation (83) and can be formalized as follows (2020, p.3):

$$z_t^{(i),\text{CSMOM}} = \begin{cases} -1, & \text{if } \tilde{s}_t^{(i)} \leq [0.1n_t] \\ 1, & \text{if } \tilde{s}_t^{(i)} \geq [0.9n_t] \\ 0, & \text{otherwise} \end{cases} \tag{86}$$

where $[0.1n_t]$ represents the bottom decile and $[0.9n_i]$ the top decile, while the momentum indicator $z_t^{(i),\text{CSMOM}}$ is defined based on the allocated decile of asset i in the cross-section.

4. Portfolio Construction

The last component of the CSMOM framework represents the actual *portfolio construction* based on the momentum indicator of the underlying assets. For this purpose, equation (83) serves as the foundation to form the bottom and top decile portfolios with maximum short and maximum long positions under the concept of volatility scaling. This results in an equally weighted long-short portfolio, which is accordingly scaled to the target volatility.

4.2 OK Deep Learning-Enhanced Multi-Asset CSMOM Model

One of the key parts of this thesis is the empirical derivation and evaluation of deep learning-enhanced multi-asset CSMOM strategies. This derivation and architectural structure is carried out, on the one hand, using the vanilla deep learning architectures from section 3 and, on the other hand, by adapting those vanilla architectures according to the fundamental CSMOM strategy framework from section 4.1. In the context of this thesis, two different model architectures based on transformer networks are proposed based on the concept of self-attentive ranking by Pobrotyn et al. (2021): In the first model, the vanilla transformer architecture is chosen in the form of an *CSMOM Transformer Ranker*, with the cross-section asset features for a time step as inputs used for an enhanced momentum score calculation. In the second model, a *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* is applied, which first generates LTR-based pre-rankings resulting in the top and bottom quintile portfolios, which are then separately re-ranked through an encoder-only transformer in order to generate final top and bottom decile portfolios. Both models are based on the general concept of the self-attentive (re-)ranking by Pobrotyn et al. (Pobrotyn et al., 2021), which has already been validated in other research areas with learning-to-rank applications.

4.2.1 OK CSMOM Transformer Ranker

The architecture of the proposed CSMOM Transformer Ranker combines the central advantages of general transformer models from section 3.2.1 with the general architectural requirements of cross-sectional momentum strategies from section 4.1, under the umbrella of self-attentive ranking by Pobrotyn et al. (2021). It can be shown that the time series functionality of the vanilla transformer from section 3.2 in the form of the encoder-decoder architecture is not

required in the context of cross-sectional applications, which justifies the use of a *encoder-only transformer* ranker. The following section provides an overview as well as the detailed derivation of the proposed encoder-only transformer architecture.

Looking back at the general CSMOM strategy framework, it can be seen that the application of deep learning-based components is particularly suitable in the context of the *momentum score calculation*. In addition, the original definitions of security selection and portfolio construction according to Jegadeesh & Titman (1993) can still be applied on the basis of the momentum score, which ensures fundamental comparability with existing benchmark models. In addition, the key property that makes the encoder-only transformer suitable for numerical ranking use cases and hence CSMOM applications is given by the *permutation-equivariance* property. This property says, that given a numeric, real-valued vector $\mathbf{u} \in \mathbb{R}^n$ as well as an arbitrary permutation $\pi \in S_n$ of n elements, a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is called permutation-equivariant iff $f(\pi(\mathbf{u})) = \pi(f(\mathbf{u}))$, which means that the score of the items does not depend on the initial ordering of the items in the input matrix and builds the foundation for ranking applications of the encoder-only transformer-based architecture (Pobrotyn et al., 2021, p.3). These observations represent an important foundation for formulating the concept of CSMOM strategies as a learning-to-rank problem. In conclusion, it underlines that CSMOM strategies can be expanded to include the complexity advantages of transformer neural networks, especially with regard to self-attention mechanisms, in order to optimize the capturing of underlying cross-sectional structures and patterns.

According to the vanilla transformer architecture from section 3.2, there are dedicated architectural adjustments needed in order to make the transformer applicable for learning-to-rank applications. As it was already explained in the context of the encoder-decoder architecture, the encoder output flows into the multi-head attention component of the decoder, while, in the context of an encoder-only architecture, there needs to be a dedicated encoder output for the subsequent *loss* calculation. In accordance with the self-attentive ranking framework proposed by Probotyn et al. 2021, p.3-4, the general components of *continuous embedding* as well as the *encoder blocks* are applied according to the vanilla transformer encoder by Vaswani et al. (2017), while an encoder output layer as well as a learning-to-rank-adapted loss function is required in addition. As part of the loss function, existing learning-to-rank approaches for CSMOM strategies can be used. There are two main loss function approaches available: *pairwse* or *listwise* loss. The empirical research results by Poh et al. (2020; 2022) show that both pairwise loss-based models such as *RankNet* as well as listwise loss-based models such as *ListNet* perform well in the course of CSMOM strategies and can significantly out-perform benchmark models. In comparison, a key advantage of listwise loss-based models is given by the computational efficiency, since listwise loss-based models calculate the loss metrics based on individual pairs, while listwise loss-based models do this based on entire lists (Poh et al., 2020, p.5). This results in the following loss complexities (Poh et al., 2020, p.5):

$$\textbf{Listwise: } O\left(N_t\right) \quad , \quad \textbf{Pairwise: } O\left(N_t^2\right) \tag{87}$$

where N_t represents the number of items at time t. Therefore, based on the computational

efficiency as well as the empirical performance in the course of CSMOM strategies, it is argued that the listwise loss function *ListNet* is used as part of the CSMOM Transformer Ranker. Based on the fundamental learning-to-rank framework, the ListNet model was initially developed by Cao et al. (2007) which implies an arbitrary, neural network-based model for the prediction function f. The corresponding loss function follows the concept of top one probabilities $P_s(j)$, which represents the probability of getting the highest ranking given the ranking scores of all other elements in the same query i, and can be formalized as follows (Cao et al., 2007, p.4):

$$P_s(j) = \sum_{\pi(1)=j, \pi \in \Omega_n} P_s(\pi) \tag{88}$$

where $P_s(\pi)$ represents the permutation probability of π given s and the top one probability of element j is given as the sum of the permutation probabilities of permutations in which the element j is ranked highest (Cao et al., 2007, p.4). Furthermore, equation (88) shows that the predicted score s_j for object j has to be transformed into a probability object. That transformation provides the softmax function, which transforms the elements of a vector into probabilities while satisfying the probability axioms, and can be formalized as follows (Cao et al., 2007, p.4):

$$P_s(j) = \frac{\phi(s_j)}{\sum_{k=1}^{n} \phi(s_k)} = \frac{\exp(s_j)}{\sum_{k=1}^{n} \exp(s_k)} \tag{89}$$

where $\phi(.) = \exp .$ represents the softmax definition. The objective of minimizing the deviation of the estimated probability distributions and thus implicitly optimizing the ranking of the elements given the ground-truth ranking labels results from the concept of the probability distribution from the top one probabilities (Cao et al., 2007, p.4). Finally, the following ListNet loss function can be defined as a metric based on the predictions model $f(.)$ and the cross-entropy (Cao et al., 2007, p.4):

$$L(\mathbf{y}, f(\mathbf{u})) = -\sum_{i} \text{softmax}(\mathbf{y})_i * \log\left(\text{softmax}\left(f(\mathbf{u})\right)_i\right) \tag{90}$$

where $\mathbf{y}$ represents the ground truth labels and $f(\mathbf{u})$ represents the predicted rankings based on the model inputs $\mathbf{u}$, while the softmax functions are derived based on the permutation probability and top-one probabilities.

After the corresponding loss function of the proposed CSMOM Transformer Ranker has been derived, the final component is to define the output functionality between the loss function and the encoder blocks by adapting the general logic of predicting object-specific rankings. This results in an additional *linear dense layer* as a dedicated encoder output layer in accordance with the original self-attentive ranking concept by Pobrotyn et al. (Pobrotyn et al., 2021, p.3-4), whereby the outputs of the encoder blocks in turn serve as inputs of that output layer. Hence, the final dense layer maps the encoder output vectors of size d_{model} to corresponding scalars. This can be further justified because the outputs of the encoder-only transformers are to be understood as rankings and alternative activation functions in the dense function fulfil other use cases, for example, softmax for output probabilities between 0 and 1 or tanh for weights

between -1 and 1. Overall, this results in the final proposed formal and visual architecture of the CSMOM Transformer Ranker:

$$
\begin{aligned}
\mathbf{z}_t = f\left(\mathbf{U}_t\right) &= \text{Linear}\left(\text{Enc}_N\left(\cdots\left(\text{Enc}_1\left(\text{PE}\left(\mathbf{U}_t\right) + \text{Emb}\left(\mathbf{U}_t\right)\right)\right)\right)\right) \\
\text{Enc}_j\left(\mathbf{H}_t\right) &= \text{Norm}\left(\mathbf{H}_t + \text{Dropout}\left(\text{FF}\left(\mathbf{H}_t\right)\right)\right) \\
\mathbf{H}_t &= \text{Norm}\left(\mathbf{X}_t + \text{Dropout}\left(\text{MHA}\left(\mathbf{X}_t\right)\right)\right) \\
\mathbf{X}_t &= \begin{cases} \text{PE}\left(\mathbf{U}_t\right) + \text{Emb}\left(\mathbf{U}_t\right), & \text{if } j = 1 \\ \text{Enc}_{j-1}\left(\cdot\right), & \text{if } j \geq 1 \end{cases}
\end{aligned} \tag{91}
$$

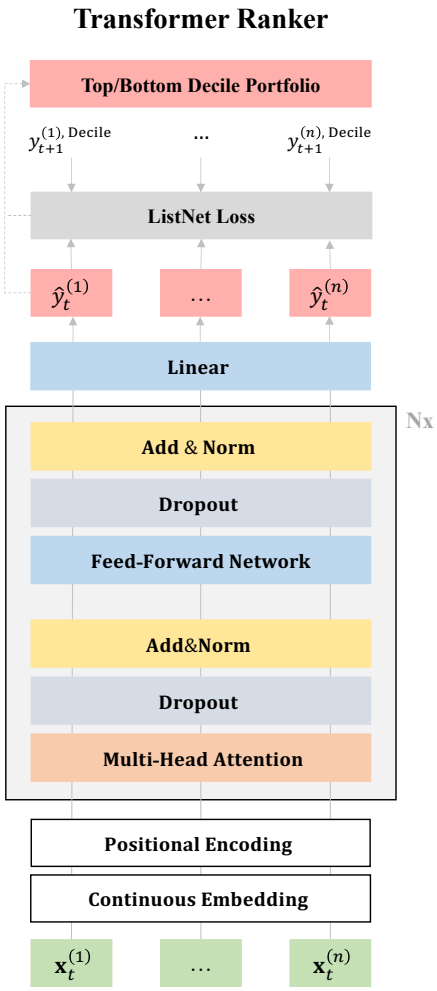


Figure 19: CSMOM Transformer Ranker - Own Illustration

In summary, based on figure 19, the data flow and generation of asset rankings through the CSMOM Transformer Ranker model can be described as follows: All assets i and their corresponding feature vectors $\mathbf{u}_t^{(i)} \in \mathbf{U}_t \in \mathbb{R}^{n_t \times p}$ for each time t separately are fed into the transformer encoder and are transformed by the continuous embedding into $\mathbf{U}_t \in \mathbb{R}^{(2n_t)/5 \times d_{model}}$, after which the transformed input data are fed into the encoder blocks with each having a multi-head attention layer, dropout layer, add&norm layer, feed-forward neural network layer, again a dropout layer and another final add&norm layer. The output of the encoder is then finally fed into

the linear dense output layer, which re-transforms the data into $\mathbf{U}_t \in \mathbb{R}^{(2n_t)/5\times 1}$ and generates the final model ranking predictions as scalars for each asset i. Finally, the output rankings $z_t^{(i)} \in \mathbf{z}_t \in \mathbb{R}^{1\times n_t}$ together with the ground-truth decile labels derived from the volatility-scaled returns between t and $t+1$ are used for the listwise loss calculation. Based on the final outputs, the portfolio of top and bottom decile assets is built according to the predicted rankings, with corresponding maximum long and maximum short positions, while the remaining assets are weighted at zero.

4.2.2 OK CSMOM ListNet Pre-Ranker Transformer Re-Ranker

The architecture of the proposed *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* builds on existing research results, while the key components can be separated into two main subcomponents: i) *ListNet Pre-Ranking* based on multi-layer perceptrons (MLP) and a listwise ranking objective according to Cao et al. (2007) and Poh at al. (2020) as well as ii) *Transformer Re-Ranking* based on the self-attentive ranking architecture according to Pobrotyn et al. (2021) and Poh at al. (2022). Hence, motivated and driven by the state-of-the-art CSMOM and LTR research findings, the prediction of momentum scores is split into the two parts of pre- and re-ranking while both being based on the deep learning architectures derived in section 3, in particular the MLP and the transformer enural network.

ListNet Pre-Ranking

Looking back at the CSMOM strategy framework, the *momentum score calculation* represents the central variable for optimization, while the score ranking and the security selection as well as the resulting portfolio construction can be treated as deterministic components due to its robust performance based on Jegadeesh & Titman (1993). With regard to the score calculation, Pobrotyn et al. (2021) provide an adaptable and empirically robust architecture for re-ranking models that serves as the foundation for the proposed momentum score calculation. The concept of self-attentive ranking offers the possibility to apply the ranking from scratch (as seen in the CSMOM Transformer Ranker from section 4.2.1) or based on a pre-ranking context. Extensive research in the field of CSMOM validates the possibility of using a well-founded and robust model for pre-ranking in order to then integrate the self-attentive ranker as a re-ranking component. Based on the results of the application of LTR to CSMOM by Poh et al. (2020), as well as the previously derived computational efficiency of listwise loss-models against pairwise loss functions, the *ListNet* model is used as the pre-ranker, on top of which a dedicated encoder-only transformer is applied for re-ranking.

As previously derived during the encoder-only transformer ranking model, the LTR applications results from queries $Q = \{q_1, ..., q_m\}$, query-associated documents $d_i = \{d_i^{(1)}, ..., d_i^{(n_i)}\}$ and query and documents-associated ground-truth labels $y_i = \{y_i^{(1)}, ..., y_i^{(n_i)}\}$. This results in the feature vector $u_i^{(j)} = \phi(q_i, d_i^{(j)})$ for each query-document pair i for each query i and each element j under a feature function $\phi(.)$ (Poh et al., 2020, p.11). In the case of CSMOM use cases, the framework can be adapted with a query q_i being defined as a portfolio re-balancing event or point in time, with the universe within the scope of each event available assets $e_i = \{e_i^{(1)}, ..., e_i^{(n_i)}\}$ the

documents and the cumulative returns (or alternative performance labels) represent the ground-truth labels over the holding period until the next re-balancing event (Poh et al., 2020, p.11). Finally, the prediction function f learns the in-sample generalization based on the training set $\{\mathbf{u}_i, \mathbf{y}_i\}_{i=1}^m$ and predicts the out-of-sample score or rank associated with query i and element i based on the out-of-sample testing set $u_{m+1}^{(i)}$.

In accordance with the derivations of the CSMOM Transformer Ranker from section 4.2.1, the listwise loss function defines a central component of the ListNet model and follows the concept of top one probabilities $P_s(j)$, which represents the probability of getting the highest ranking given the ranking scores of all other elements in the same query i and can be formalized as follows (Cao et al., 2007, p.4):

$$P_s(j) = \sum_{\pi(1)=j, \pi \in \Omega_n} P_s(\pi) \tag{92}$$

where $P_s(\pi)$ represents the permutation probability of π given s and the top one probability of element j as the sum of the permutation probabilities of permutations in which the element j am is ranked highest (Cao et al., 2007, p.4). Furthermore, the predicted score s_j for object j gets transformed into a probability object through the softmax function as follows (Cao et al., 2007, p.4):

$$P_s(j) = \frac{\phi(s_j)}{\sum_{k=1}^n \phi(s_k)} = \frac{\exp(s_j)}{\sum_{k=1}^n \exp(s_k)} \tag{93}$$

where $\phi(.) = \exp.$ represents the softmax definition. The objective of minimizing the deviation of the estimated probability distributions and thus implicitly optimizing the ranking of the elements given the ground-truth ranking labels results from the concept of the probability distribution from the top one probabilities. Finally, the following ListNet loss function can be defined as a metric based on the predictions model $f(.)$ and the cross-entropy (Cao et al., 2007, p.4):

$$L(\mathbf{y}, f(\mathbf{u})) = -\sum_i \text{softmax}(\mathbf{y})_i * \log\left(\text{softmax}\left(f(\mathbf{u})\right)_i\right) \tag{94}$$

After the basic ListNet framework for CSMOM applications has been defined, it is finally necessary to define the underlying, neural network-based prediction model. Here, Poh et al. (2020, p.6-8) show that vanilla multi-layer perceptrons (MLP) combined with the ListNet loss function using volatility scaling and monthly re-balancing in US equity markets leads to significant and robust out-performances compared to common benchmark models. With reference to the architecture proposed here for the pre-ranking as part of the multi-asset CSMOM portfolio construction, a *multi-layer perceptron* is also used as a prediction model, which in connection with the ListNet loss function can be formalized as follows:

$$\begin{aligned} \mathbf{z}_t^{\text{Pre-Rank}} &= f\left(\mathbf{U}_t^{\text{Pre-Rank}}\right) = \text{Output}\left(\text{Hidden}_N\left(\cdots\left(\text{Hidden}_1\left(\mathbf{U}_t^{\text{Pre-Rank}}\right)\right)\right)\right) \\ &\mathbf{U}_t^{\text{Pre-Rank}} \in \mathbb{R}^{n_t \times p} \end{aligned} \tag{95}$$

Self-Attentive Re-Ranking

The self-attentive re-ranker according to Probotyn et al. (2021) provides an encoder-only transformer-based architecture for real-valued input matrices $\mathbf{U}_t$ in the form input feature vectors $\mathbf{u}_t \in \mathbf{U}_t$, where each vector represents the features of an asset i and the output vector $f(\mathbf{U}_t)$ represents the relative scalar scores $f(\mathbf{u}_t)$ of the assets (Pobrotyn et al., 2021, p.3-4). The actual architecture of the CSMOM Transformer Re-Ranker model is equivalent to the CSMOM Transformer Ranker from the previous section: A *feed-forward layer* is used for the continuous embedding, which implicitly represents a learnable input embedding layer for numerical, real-valued input vectors $\mathbf{u}_t$ (Pobrotyn et al., 2021, p.3-4), after which the *encoder blocks* are applied and the encoder block outputs are transformed into a scalar output $f(\mathbf{u}_t) \in \mathbb{R}$ by a *linear dense layer* which finally generates the predicted output scores (Pobrotyn et al., 2021, p.3-4). Since the input of the CSMOM Transformer Re-Ranker will already be a sorted list based on the top and bottom quintiles of the CSMOM ListNet Pre-Ranker, no positional encoding is needed. In addition, the property of *permutation-equivariance*, which means that the score of the items does not depend on the initial ordering of the items in the input matrix, builds the foundation for re-ranking applications of the encoder-only transformer-based architecture (Pobrotyn et al., 2021, p.3).

The central link between the CSMOM ListNet Pre-Ranker and the Transformer Re-Ranker is given by the input of the transformer $\mathbf{U}_t$, which in turn represents the output of the pre-ranker. The top and bottom quintile portfolios are formed as part of the ListNet pre-ranking, after which only the top and bottom quintile assets are included as separate input into the transformer. The task of the transformer is therefore to carry out a re-ranking of the already pre-ranked assets, whereby all other assets outside the top and bottom quintiles are not included in the re-ranker. Based on those top and bottom quintiles, the re-ranker constructs top and bottom decile portfolios, whereby the top deciles are weighted with maximum long and the bottom deciles with maximum short position, and the remaining assets are weighted with zero. Thus, the output of the entire pre-ranking-re-ranking model consists of the same number of assets as the existing benchmark strategies, which are further explained in section 4.2.4. Finally, the full CSMOM ListNet Pre-Ranker Transformer Re-Ranker model can be formalized and visualized as follows:

$$
\begin{aligned}
\mathbf{z}_t^{\text{Re-Rank}} = f\left(\mathbf{U}_t^{\text{Re-Rank}}\right) &= \text{Linear}\left(\text{Enc}_N\left(\cdots\left(\text{Enc}_1\left(\text{Emb}\left(\mathbf{U}_t\right)\right)\right)\right)\right) \\
\text{Enc}_j\left(\mathbf{H}_t\right) &= \text{Norm}\left(\mathbf{H}_t + \text{Dropout}\left(\text{FF}\left(\mathbf{H}_t\right)\right)\right) \\
\mathbf{H}_t &= \text{Norm}\left(\mathbf{X}_t + \text{Dropout}\left(\text{MHA}\left(\mathbf{X}_t\right)\right)\right) \\
\mathbf{X}_t &= \begin{cases} \text{Emb}\left(\mathbf{U}_t\right), & \text{if } j = 1 \\ \text{Enc}_{j-1}\left(\cdot\right), & \text{if } j \geq 1 \end{cases} \\
\mathbf{U}_t^{\text{Re-Rank}} \in \mathbb{R}^{(n_t/5)\times p} &\in \mathbf{U}_t^{\text{Pre-Rank}} \in \mathbb{R}^{n_t \times p}
\end{aligned}
\tag{96}
$$

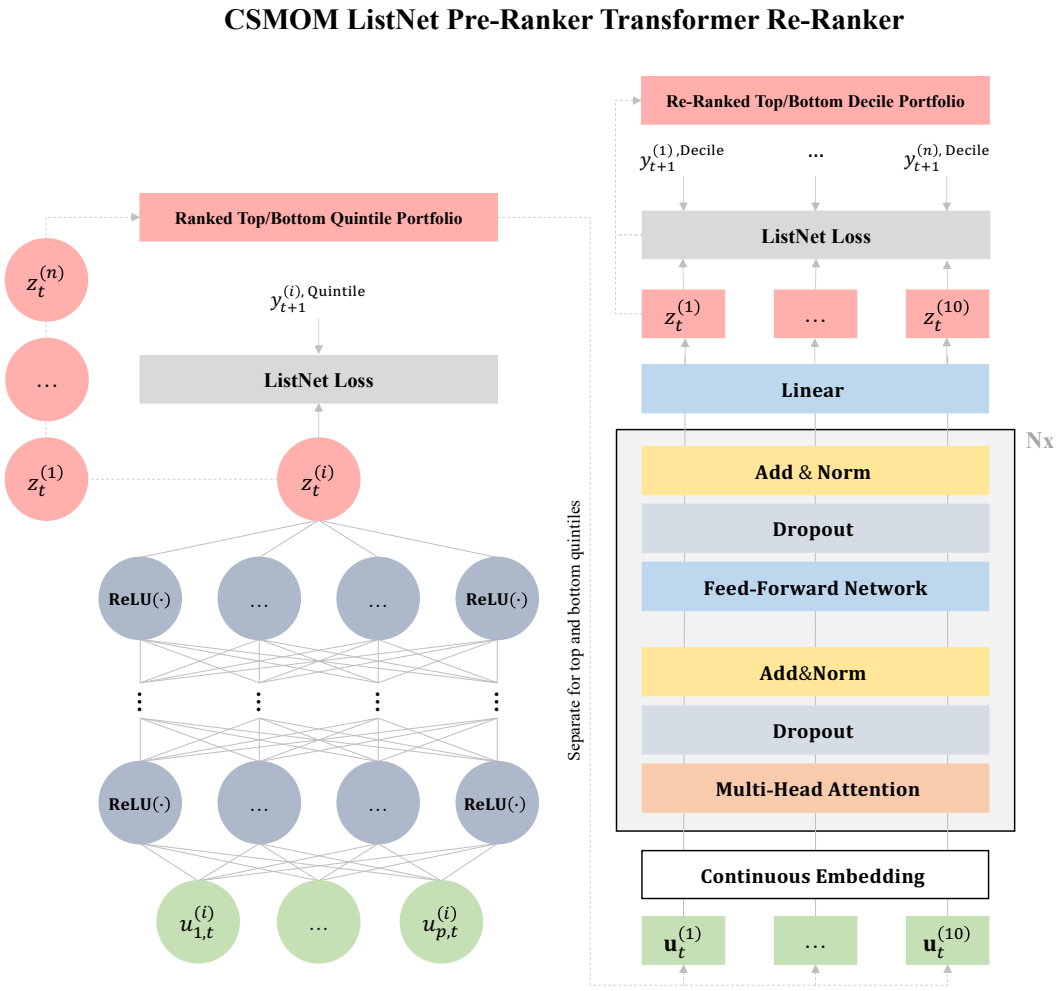


Figure 20: CSMOM ListNet Pre-Ranker Transformer Re-Ranker - Own Illustration

Based on figure 20, the CSMOM ListNet Pre-Ranker Transformer Re-Ranker model can be summarized as follows: All assets i and their corresponding feature vector $\mathbf{u}_t^{(i),\text{PreRanker}} \in \mathbf{U}_t^{\text{PreRanker}} \in \mathbb{R}^{n_t \times p}$ for each time t separately are fed into the fully-connected MLP with multiple hidden layers, from which the pre-ranker output rankings $z_t^{(i),\text{PreRanker}} \in \mathbf{z}_t^{\text{PreRanker}} \in \mathbb{R}^{1 \times n_t}$ together with the ground-truth decile labels derived from the volatility-scaled returns between t and $t+1$ are used for the listwise loss calculation. From there, the top and bottom quintile assets are further separately fed into the re-ranker. In there, the input features $\mathbf{u}_t^{(i),\text{ReRanker}} \in \mathbf{U}_t^{\text{ReRanker}} \in \mathbb{R}^{(2n_t)/5 \times p}$ are transformed by the continuous embedding and by the positional encoding layer into $\mathbf{U}_t^{\text{ReRanker}} \in \mathbb{R}^{(2 \times n_t)/5 \times d_{model}}$, after which the transformed input data are fed into the encoder blocks with each having a multi-head attention layer, dropout layer, add&norm layer, feed-forward neural network layer, again a dropout layer and another final add&norm layer. The output of the encoder is then finally be fed into the last linear dense layer, which re-transforms the data into $\mathbf{U}_t^{\text{ReRanker}} \in \mathbb{R}^{(2n_t)/5 \times 1}$. Finally, the re-ranker output rankings $z_t^{(i),\text{ReRanker}} \in \mathbf{y}_t^{\text{ReRanker}} \in \mathbb{R}^{1 \times n_t}$ together with the ground-truth decile labels derived from the volatility-scaled returns between t and $t+1$ are used for the listwise loss calculation. Based on the final outputs, the portfolio of top and bottom decile assets is build according to the predicted rankings, with corresponding maximum long and maximum short positions, while the

remaining assets are weighted at zero.

4.2.3 OK CSMOM Strategy Assumptions & Requirements

After the proposed CSMOM models have been defined as the CSMOM Transformer Ranker as well as the CSMOM ListNet Pre-Ranker Transformer Re-Ranker, the deterministic strategy parameters need to be defined, while the hyperparameters are explained in more detail during the strategy back-testing in section 6.

Portfolio Size

As can already be seen from the general definitions of the proposed CSMOM models, the portfolio construction is not carried out based on all available assets, which is in line with existing CSMOM benchmark strategies such as the original research by Jegadeesh et al. (1993). It is therefore important to determine the portfolio size with regard to the long and short positions. Here too, existing research approaches are used to ensure comparability during the evaluation. In accordance with the security selection defined as part of the general CSMOM strategy framework in equation (86), the portfolio size is defined as follows:

CSMOM Transformer Ranker	**CSMOM ListNet Pre-Ranker Transformer Re-Ranker**
$z_t^{(i)} = \begin{cases} -1, & \text{if } \tilde{s}_t^{(i)} \leq [0.1n_t] \\ 1, & \text{if } \tilde{s}^{(i)} \geq [0.9n_t] \\ 0, & \text{otherwise} \end{cases}$	$z_t^{(i),\text{Pre-Ranker}} = \begin{cases} -1, & \text{if } \tilde{s}_t^{(i),\text{Pre-Ranker}} \leq [0.2n_t] \\ 1, & \text{if } \tilde{s}_t^{(i),\text{Pre-Ranker}} \geq [0.8n_t] \\ 0, & \text{otherwise} \end{cases}$ $z_t^{(i),\text{Re-Ranker}} = \begin{cases} -1, & \text{if } \tilde{s}_t^{(i),\text{Re-Ranker}} \leq [0.1n_t] \\ 1, & \text{if } \tilde{s}_t^{(i),\text{Re-Ranker}} \geq [0.9n_t] \\ 0, & \text{otherwise} \end{cases}$

Table 1: CSMOM Strategy Parameters - Portfolio Size

where the CSMOM Transformer Ranker directly generates a top and bottom decile portfolio according to the ranking outputs with maximum long positions $z_t^{(i)} = 1$ and maximum short positions $z_t^{(i)} = -1$. As part of the CSMOM Transformer Re-Ranker, top and bottom quintile portfolios are generated in the first step by the CSMOM ListNet Pre-Ranker, which in turn serve with the original features as input for the re-ranker, which then, equivalent to the CSMOM Transformer Ranker, creates a top and bottom decile portfolio with position sizes $z_t^{(i)} = 1$ and $z_t^{(i)} = -1$. These procedures are in turn in accordance with existing research approaches by Poh et al. (2020; 2022).

Re-balance Frequency

With regard to transaction cost effects, strategy responsiveness and the computational performance of automated trading strategies, the deterministic re-balance frequency takes a central

role. The CSMOM strategies proposed here are set to be re-balanced monthly in accordance with existing benchmark strategies and research work, in particular to prevent excessive transaction cost effects that occur in the context of short re-balancing frequencies (Poh et al., 2020, p.3). Hence, the re-balance frequency can be defined as follows:

CSMOM Transformer Ranker	**CSMOM ListNet Pre-Ranker Transformer Re-Ranker**
$n_{rebalance} = 21$	$n_{rebalance} = 21$

Table 2: CSMOM Strategy Parameters - Re-balance Frequency

where 21 days are derived as an approximation of the trading days within a month over approximately 252 trading days per year. Implicitly, this means that the position sizes $\mathbf{z}^{(i)} = \left(z_t^{(i)}\right)_{t=1}^{T}$ remain constant over the days within the following trading month.

Target & Realized Volatility

Based on the proposed CSMOM models, the target volatility and the realized volatility in the course of volatility-scaled returns ultimately serve as a key component when deriving the realized strategy return over time. In line with benchmark CSMOM strategies by Poh et. al (2022, p.3), the target volatility is defined as follows:

CSMOM Transformer Ranker	**CSMOM ListNet Pre-Ranker Transformer Re-Ranker**
$\sigma_{target} = 15\%$	$\sigma_{target} = 15\%$

Table 3: CSMOM Strategy Parameters - Target Volatility

The second component for deriving volatility-scaled returns is given by the definition of the realized volatility. In the course of this work, the key goal for defining the strategy parameters is to establish comparability with benchmark models and existing research work, which means that ex-ante volatility is used in the form of the exponentially weighted moving standard deviation with a predefined volatility look-back period, according to Poh et al. (2022, p.3). However, other and more sophisticated methods such as econometric GARCH or machine learning approaches can be utilized as well (Poh et al., 2022, p.3). It is important to note that this approach is suitable for portfolios with an approximate diagonal covariance matrix, which means that the realized portfolio volatility is approximated from the asset-specific volatilities and the covariance can be neglected due to low correlation between the assets (Wood et al., 2022, p.5). As will be explained in more detail later in section 5, the underlying data set consists of continuous future contracts, which have significantly less covariance structure compared to other asset classes, which can justify the proposed approach (Wood et al., 2022, p.5). This results in the following definition of the realized volatility:

CSMOM Transformer Ranker	**CSMOM ListNet Pre-Ranker Transformer Re-Ranker**
$\sigma_t^{(i)} = \text{EWMSD}\left(\left(r_t^{(i)}\right)_{t=1}^{T}\right)$	$\sigma_t^{(i)} = \text{EWMSD}\left(\left(r_t^{(i)}\right)_{t=1}^{T}\right)$

Table 4: TSMOM Strategy Parameters - Realized Volatility

4.2.4 OK CSMOM Strategy Benchmark Models

In the course of the evaluation of the proposed deep learning-enhanced CSMOM strategies, state-of-the-art benchmark strategies need to be defined in order to back-test the strategy performance. As part of cross-sectional momentum, three types of strategies are used: First, *Random* portfolio weights are derived, with random weights for each asset i are generated every month t. This strategy serves as the absolute baseline strategy. Furthermore, the second class of benchmark models consists of fundamental CSMOM strategies, in particular the *Original CSMOM* strategy by Jegadeesh et al. (1993), which to this day represents a central standard of CSMOM back-testing, and the *MACD Strategy* by Baz et al. (2015), which extends the conceptual idea of original CSMOM strategies by including convergence-divergence dynamics. The third and last class of benchmark models consists of machine learning models, in particular *Multi-Layer Perceptrons* (MLP) as well as *ListNet*, while the latter builds upon the concept of MLPs while incorporating different loss function dynamics. In the following sections, those strategies are explained and defined in more detail.

Random

Within the framework of the random benchmark models as the absolute baseline model, in accordance with the general CSMOM strategy parameters in section 4.2.3, the top and bottom decile portfolios are randomly defined at each re-balance period t, while the remaining assets are not included in the portfolio. This results in the following derivation of the portfolio weights under implied leverage restriction:

$$\textbf{Security Selection:} \quad z_t^{(i),\text{Random}} = \begin{cases} -1, & \text{if } \text{R}(i) = -1 \\ 1, & \text{if } \text{R}(i) = 1 \\ 0, & \text{otherwise} \end{cases} \tag{97}$$

where $\text{R}(i)$ assigns 1 to 10% of the assets, -1 to 10% of the assets and 0 to all other assets, whereupon top and bottom decile portfolios are generated, each with maximum long positions $z_t^{(i)} = 1$ and maximum short positions $z_t^{(i)} = -1$. Implicitly, according to the general CSMOM strategy framework, the realized strategy return from period t to the next period $t+1$ can be derived from (82) as follows:

$$r_{t,t+1}^{\text{Random}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{Random}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{98}$$

where the position sizes are defined as given in (97), while all other strategy components are defined as given in table 3 and 4.

Original CSMOM

Within the second class of benchmark strategies, the original CSMOM strategies according to Jegadeesh et al. (1993) are utilized. In comparison to the random strategy, the momentum score calculation as well as the momentum score ranking are explicitly derived, which is then incorporated into the specific security selection according to the general CSMOM framework in section 4.2. Here, the momentum score calculation is defined as the cumulative return over a predefined look-back period, whereupon the momentum score ranking ranks all assets in ascending order according to the cumulative returns, after which the security selection takes the ranking as the indicator for the top and bottom decile portfolios with a maximum long or maximum short positions, respectively (1993). This results in the following overall strategy definition (1993):

$$\textbf{Momentum Score Calculation:} \quad s_t^{(i)} = r_{t-m,t}^{(i)} \;\; , \;\; \mathbf{s}_t = \left\{ s_t^{(1)}, ..., s_t^{(n_t)} \right\} \tag{99}$$

where m is given as a central strategy parameter as the look-back period for deriving cumulative asset returns. The subsequent momentum score ranking is based on the original version according to Jegadeesh & Titman (1993), whereby the momentum scores are sorted in descending order, with the subsequent security selection as the top and bottom decile portfolios and corresponding maximum long and short positions. Based on this definition, it is necessary to define the strategy parameter of the look-back period for deriving the cumulative returns, for which the following deterministic values are defined in line with the empirical results of Jegadeesh et al. (1993) and further research work by Poh et al. (2020):

$$m \in \{3, 6, 12\} \tag{100}$$

whereby the momentum score calculation in accordance with (99) is derived based on the three-month, six-month and twelve-month cumulative asset returns. Finally, the realized return of the original CSMOM strategies can be defined as follows:

$$r_{t,t+1}^{\text{Original}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{Original}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{101}$$

where all strategy components are defined as given in table 3 and 4.

Moving Average Convergence Divergence (MACD)

Based on the empirical results of Jegadeesh et al. (1993), there is an extension in the form of the Moving Average Convergence Divergence (MACD) strategy. As the name suggests, the MACD strategy is a method of signal construction that uses the convergence or divergence of moving averages of asset returns (Baz et al., 2015, p.10-11). Upon that, the strategy combines differ-

ent intermediate volatility-scaled MACD signals over different time scales through a dedicated response function (Poh et al., 2020, p.3-4). Hence, the MACD strategy in course of the general CSMOM strategy framework can be defined as follows, in line with the formalization of Poh et al. (2020):

$$\textbf{Momentum Score Calculation:} \quad \begin{aligned} s_t^{(i)} &= \sum_{k=1}^{3} \phi\left(\tilde{s}_t^{(i)}\left(S_k, L_k\right)\right) \\ \tilde{s}_t^{(i)} &= \xi_t^{(i)} / \text{std}\left(z_{t-252:t}^{(i)}\right) \\ \xi_t^{(i)} &= \text{MACD}\left(i, t, S, L\right) / \text{std}\left(p_{t-63:t}^{(i)}\right) \\ \text{MACD}\left(i, t, S, L\right) &= m\left(i, S\right) - m\left(i, L\right) \end{aligned} \tag{102}$$

where $\text{std}\left(p_{t-63:t}^{(i)}\right)$ defines the rolling standard deviation of asset i, $m(i, S)$ represents the exponentially weighted moving average of prices for asset i, while S is the half-life decay factor formalized as $HL = \log(0.5)/\log(1 - 1/S)$ (Poh et al., 2020, p.3-4). The final momentum score $s_t^{(i)}$ combines different volatility-scaled MACD scores over different time scales through the response function $\phi(\dot{})$ and a set of short and long time scales S_k and L_k (Poh et al., 2020, pp. 3–4). Based on the generalized CSMOM strategy framework, the MACD-based momentum scores are accordingly ranked in descending order, after which the security selection evolves from top and bottom decile portfolios with maximum long and maximum short positions. In accordance with existing benchmark approaches, the sets of short and long time scales are defined as follows:

$$S_k \in 8, 16, 32 \quad , \quad L_k \in 24, 48, 96 \tag{103}$$

From (102) it can be seen that, after the MACD-specific momentum score calculation, the momentum score ranking and the security selection are carried out analogously to the original CSMOM strategies, whereby the MACD-based score is sorted in descending order and then the top and bottom decile portfolios with maximum long and maximum short positions are formed. Finally, analogous to the general CSMOM strategy framework, the volatility-scaled realized strategy return from time t to time $t+1$ can be defined as follows:

$$r_{t,t+1}^{\text{MACD}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{MACD}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{104}$$

where the position sizes $z_t^{(i),\text{MACD}}$ correspond to (102) and all strategy components are defined as given in table 3 and 4.

Multi-layer Perceptron (MLP)

In the context of ranking problems, machine learning approaches represent state-of-the-art methods, whereby the CSMOM can be approached as a classic regress-the-rank case (Poh et al., 2020, p.4). As already explained in section 3.1.1, the *multi-layer perceptron* represents a fundamental model architecture that, in the context of feed-forward neural networks, takes a predefined

feature set as input and uses sequentially linear operations to generate an output set. Applying this model architecture to CSMOM strategies shows that all assets and their features at time t can serve as inputs, whereupon the momentum score can be generated for each asset in the cross-section. This results in the following CSMOM strategy definition utilizing MLPs:

$$\begin{aligned} \textbf{Momentum Score} \\ \textbf{Calculation:} \quad s_t^{(i)} &= f\left(u_t^{(i)}\right) \\ &= \text{Output}\left(\text{Hidden}_N\left(...\left(\text{Hidden}_1\left(\text{Input}\left(u_t^{(i)}\right)\right)\right)\right)\right) \end{aligned} \tag{105}$$

where $f\left(u_t^{(i)}\right)$ is defined as an MLP prediction model according to an input layer, N hidden layers and an output layer. According to the model introduction in section 3.1.1, the model parameters are validated as hyperparameters in the course of model training in section 6, whereby only the number of hidden layers according to the benchmark strategies of Poh et al. (2020) is set to $N = 2$. The momentum score ranking based on the MLP-based rankings is done in descending order, after which top and bottom decile portfolios are constructed under maximum long and maximum short positions according to Jegadeesh et al. (1993). As a last component during the momentum score calculation, deep neural networks such as the MLP require a dedicated loss function to be defined in order to calibrate the model parameters. Since the problem setting corresponds to a regress-then-rank case, the loss function is defined as the mean squared error (MSE) according to the following formalization:

$$L(\mathbf{r}_{t,t+1}, \mathbf{s}_t) = \frac{1}{n}\sum_{t=1}^{n}\left(s_t^{(i)} - \frac{r_{t,t+1}^{(i)}}{\sigma_t^{(i)}}\right)^2 \tag{106}$$

where $\frac{r_{t,t+1}^{(i)}}{\sigma_t^{(i)}}$ are used as volatility-scaled returns as true labels and $s_t^{(i)}$ represent the asset i-specific MLP ranking outputs at time t. This means that the goal of the MLP is to predict the volatility-scaled returns, which are then used as a momentum score and are then sorted in descending order and incorporated into the portfolio construction of the generalized CSMOM strategy framework. Finally, the volatility-scaled realized strategy return from time t to time $t+1$ can be defined as follows:

$$r_{t,t+1}^{\text{MLP}} = \frac{1}{n_t}\sum_{i=1}^{n_t} z_t^{(i),\text{MLP}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{107}$$

where all strategy components are defined as given in table 3 and 4 based on the predicted momentum scores from (105).

ListNet

One of the current state-of-the-art learning-to-rank algorithms in the field of machine learning is *ListNet*, which has already been utilized within the pre-ranking mechanism of the proposed CSMOM strategy. In contrast to existing methods based on a pairwise approach, where two objects are viewed as instances and ranked in pairs with each other, ListNet follows a listwise

ranking approach (Cao et al., 2007, p.1). Here, lists of objects are defined as instances in the learning process, with the subsequent ranking taking place on the basis of a loss function, which is derived from the permutation probability and as well as the top one probability (Cao et al., 2007, p.1). Since that approach has already been explained in the derivation of one of the proposed CSMOM strategy models (see section 4.2.2), and only differs from standard MLPs in terms of the listwise loss function, only the ListNet loss function is further elaborated. The loss input in the form of the ground truth labels is used as the ordinal decile ranking in the asset cross-section. For this purpose, at the prediction time t, the returns in $t+1$ of all assets are classified as deciles of the cross-section, whereby the ground truth labels of the top 10% of the assets have the value 10 based on their returns and the ground truth labels of the bottom 10% gets the value 1. Assets in the deciles in between receive the ordinal ranking according to the assigned decile. In addition, the outputs of the MLP are used as predicted labels, on the basis of which the listwise loss between the ground truth labels and the predicted rankings is then calculated. The general model architecture can therefore be defined as follows, analogous to the MLP:

$$\begin{aligned} \textbf{Momentum Score Calculation:} \quad s_t^{(i)} &= f\left(u_t^{(i)}\right) \\ &= \text{Output}\left(\text{Hidden}_N\left(...\left(\text{Hidden}_1\left(\text{Input}\left(u_t^{(i)}\right)\right)\right)\right)\right) \end{aligned} \tag{108}$$

whereupon, analogous to the MLP benchmark model, the momentum score ranking in descending order and the security selection takes place based on the top and bottom decile portfolio as well as corresponding maximum long and maximum short positions, according to the general CSMOM strategy framework as well as Jegadeesh et al. (1993). In contrast to the MLP architecture from the previous section, the loss function within the momentum score calculation can be defined according to the listwise loss as follows (Cao et al., 2007, p.4):

$$L(\mathbf{y}_t, \mathbf{s}_t) = -\sum_i \text{softmax}(\mathbf{y})_i * \log\left(\text{softmax}\left(\mathbf{s}_t\right)_i\right) \tag{109}$$

where $\mathbf{y}_{t+1}$ represents the ground truth decile labels based on the asset returns in $t+1$ and $f(\mathbf{s}_t)$ represents the predicted asset rankings, while the softmax functions are derived based on the permutation probability and top-one probabilities, according to the derivations in section 4.2. This loss function ensures that the ordinal ranking in the asset cross-section is optimized. Based on this definition, the following realized strategy return from t to $t+1$ results:

$$r_{t,t+1}^{\text{ListNet}} = \frac{1}{n_t}\sum_{i=1}^{n_t} z_t^{(i),\text{ListNet}} \frac{\sigma_{target}}{\sigma_t^{(1)}} r_{t,t+1}^{(i)} \tag{110}$$

where all strategy components are defined as given in table 3 and 4 based on the initial momentum scores from (108)

4.3 OK Generalized TSMOM Strategy Framework

Equivalent to the CSMOM strategy framework, the derivation of a generalized TSMOM strategy framework builds the foundation for deriving and constructing deep learning-enhanced model extensions in order to ensure the required model architectures as well as the comparability with existing research approaches. The key underlying framework component is given by the realized TSMOM strategy return according to Lim et al. (2020, p.3-4), which is based on the original work by Moskowitz et al. (2012) and is fundamentally used across many research projects:

$$r_{t,t+1}^{TSMOM} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{TSMOM}} r_{t,t+1}^{(i)} \tag{111}$$

where $r_{t,t+1}^{TSMOM}$ represents the strategy return from period t to $t+1$ under a portfolio of n_t assets and the asset-specific time series momentum position size $z_t^{(i),\text{TSMOM}} \in [-1, 1]$ with an implied leverage restriction. In addition, equivalent to the generalized CSMOM strategy framework from section 4.1, the concept of volatility-scaled returns based on the findings of Kim et al. (2016) can also be applied to the TSMOM strategy returns which ensures that each asset has a similar contribution to the overall portfolio return. Thus, the TSMOM strategy return can be further defined as follows:

$$r_{t,t+1}^{TSMOM} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{TSMOM}} \frac{\sigma_{target}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)} \tag{112}$$

where each asset return $r_{t,t+1}^{(i)}$ is factored by the volatility scaling factor $\frac{\sigma_{target}}{\sigma_t^{(i)}}$. In addition to the return definition, the actual time series momentum framework as the foundation for the model derivation, which is dedicated to the definition of the momentum signal $z_t^{(i),\text{TSMOM}}$, is now to be examined. For this purpose, the aforementioned two central framework components are defined and elaborated in the following paragraphs.

1. Trend Estimation

The first component of the general TSMOM framework is the *trend estimation*. As previously explained, the trend estimation is carried out for each asset i separately based on the vector of input features $\mathbf{u}_t^{(i)}$, which only contains asset i-related features. Trend estimation is therefore treated as a classic regression task. Accordingly, it is necessary to define a prediction model f, which calculates the specific trend of the asset i based on the feature vector as follows (Lim, Zohren, et al., 2020, p.3-4):

$$s_t^{(i)} = f(\mathbf{u}_t^{(i)}) \tag{113}$$

where $s_t^{(i)} \in \mathbf{s}_t = \left\{ s_t^{(1)}, ..., s_t^{(n_t)} \right\}$ represents the trend estimation of an asset as an element of the trend estimation vector of all assets at time t. It is already evident that the definition of the prediction model f is a fundamental variable of choice in the context of time series momentum strategies and can be defined as a central component of the strategy performance.

2. Position Sizing

After the trend estimation has been defined for each asset i at each time t, the next step requires to convert it into an actionable position size. Thus, it is necessary to determine a function ϕ which takes the trend estimation $y_t^{(i)}$ as an input and transforms it into an asset-specific position size at time t. This can be formalized as follows (Lim, Zohren, et al., 2020, p.3-4):

$$z_t^{(i),\text{TSMOM}} = \phi(s_t^{(i)}) \tag{114}$$

where $z_t^{(i),\text{TSMOM}} \in \mathbf{z}_t^{\text{TSMOM}} = \{z_t^{(1),\text{TSMOM}}, ..., z_t^{(n_t,\text{TSMOM})}\}$ represents the position size of an asset as an element of the position size vector of all assets at time t, according to (111). State-of-the-art TSMOM approaches often refer to a concrete form of the prediction function $\phi(.)$, especially in the form of the sign function, which takes a maximum long position $z_t^{(i),\text{TSMOM}} = 1$ for positive trends and a maximum short position $z_t^{(i),\text{TSMOM}} = -1$ for negative trends. This specification results in the following position sizing rule:

$$z_t^{(i),\text{TSMOM}} = \phi(s_t^{(i)}) = \text{sign}(s_t^{(i)}) \tag{115}$$

where $z_t^{(i),\text{TSMOM}}$ implies the same properties as derived in (114). While trend estimation and position sizing can be understood as sequential components of TSMOM strategies, research approaches exist to derive these components simultaneously.

1+2. Simultaneous Trend Estimation & Position Sizing

In addition to the sequential definition of trend estimation and the position sizing, new research approaches pursue the simultaneous learning of these strategy components. Here, the explicit step of trend estimation is skipped, whereby the position sizing rule is learned directly based on the asset-specific feature input vector $\mathbf{u}_t^{(i)}$ in a data-driven approach. This can be formalized as follows:

$$z_t^{(i),\text{TSMOM}} = f(\mathbf{u}_t^{(i)}) \tag{116}$$

where $z_t^{(i),\text{TSMOM}} \in \mathbf{z}_t = \{z_t^{(1)}, ..., z_t^{(n_t)}\}$ represents the position size of an asset as an element of the position size vector of all assets at time t, with a maximum short position of $z_t^{(i),\text{TSMOM}} = -1$ and a maximum long position of $z_t^{(i),\text{TSMOM}} = 1$. In comparison to the sequential approach, simultaneously learning trend estimations and position sizing comprises the lack of direct information on the optimal positions to hold at each time step t (Lim, Zohren, et al., 2020, p.5). Since this information is required to produce labels for standard regression models, the model calibration must be performed by directly optimising strategy performance metrics, such as the sharpe ratio (Lim, Zohren, et al., 2020, p.5). A common approach of including the performance directly into the regression model is given by specifying a loss function for machine learning models which provides the performance given the predicted position size and the actual asset returns.

3. Portfolio Construction

After the trend estimation and the position sizing have been derived in the course of sequential and simultaneous approaches for each asset i at each point in time t, the final step compromises the transformation into a concrete framework for portfolio construction. A comparison with the CSMOM framework shows that not only parts of the available assets are included in the portfolio in the form of top and bottom deciles, but all assets are traded according to the position size $z_t^{(i),\text{TSMOM}}$ and therefore no security selection step exists. According to (112), the volatility-scaled TSMOM strategy return is ultimately based on the asset-specific position sizes generated over time.

4.4 OK Deep Learning-Enhanced Multi-Asset TSMOM Model

Equivalent to the development of the deep learning-enhanced CSMOM strategies, the generalized TSMOM strategy framework from the previous section is used within the framework of TSMOM to expand deep learning models from chapter 3. In the context of this thesis, two different model architectures based on transformer neural networks are proposed: The **TSMOM Encoder-Decoder Transformer**, with the asset-specific asset features for each timestamp as inputs with regard to the predefined look-back period are used for simultaneous trend estimation and position sizing, whereupon a single position size of the next timestamp is predicted. In contrast, in the second model a **Decoder-Only Transformer** is applied, which uses the asset-specific asset features for each timestamp with regard to the predefined look-back period are used for simultaneous trend estimation and position sizing and the position size is predicted for each timestamp. While the research findings of Li et al. (2020) suggest that transformer architectures for time series forecasting only utilizing the decoder part might be sufficient, while Zhou et al. (2021) argue that transformers are inherently designed as an encoder-decoder models, as further outlined with respect to time series momentum strategy by Wood et al., 2022. Hence, both setups are proposed and particularly designed and back-tested for TSMOM strategies, while the following sections explain the proposed models in detail.

4.4.1 OK TSMOM Encoder-Decoder Transformer

As already shown in the vanilla transformer architecture from section 3.2 based on the original research findings by Vaswani et al. (2017), there is already a native structure for capturing time series dependencies within the framework of encoder-decoder transformer structures. In addition, one of the main motivations is to optimize the shortcomings of RNNs and LSTMs with regard to the complex capturing of time dependencies through the novel self-attention mechanisms. Especially with regard to the numerical application of encoder-decoder transformer, the motivation arises to apply the basic architecture of transformer neural networks to simultaneous trend estimation and position sizing within the generalized framework of TSMOM strategies. In addition, there is already a lot of research work dedicated to general time series applications

based on encoder-decoder transformers and achieving robust results, see Lim et al. (2020). Nevertheless, with a view to the vanilla transformer architecture, it is evident that fundamental adjustments must be made with regard to the application to TSMOM strategies.

With regard to the application of encoder-decoder transformers to numerical TSMOM problems, it is fundamentally important to develop an encoder-decoder structure that generates the position size in the decoder as an output based on the implicit trend estimation and uses a time series look-back period for the encoder input. Hence, there is a data input as the encoder input feature vectors $\mathbf{u}_t^{(i),\text{Encoder}} \in \mathbf{U}_{T-\tau+1:T-1}^{(i),\text{Encoder}}$ and the decoder input feature vector $\mathbf{u}_T^{(i),\text{Decoder}}$ separately for each asset i. Furthermore, the scalar output can be defined as $z_T^{(i)}$, which represents the position size at the re-balancing time T. Based on this, the first central change in contrast to vanilla transformer arises: It is constantly assumed that the decoder input and output only comprise one time step and that the the condition $z_T^{(i)}\ in[-1,1]$ has to be fulfilled.

After the basic requirements for the input and output data have been described, the encoder component of the TSMOM Encoder-Decoder Transformer needs to be derived. The architecture is carried out according to the vanilla transformer architecture proposed by Vaswani et al. (2017). Accordingly, an input transformation takes place using a *continuous embedding* in the form of a fully-connected linear layer and a *positional encoding*, whereby the original input according to $\mathbf{U}_{T-\tau+1:T-1}^{(i),\text{Encoder}} \in \mathbb{R}^{(\tau-1)\times d_{model}}$ gets reshaped. In addition, the fully-connected layer allows learnable, data-driven embedding of the input vectors. The transformed input data is then passed to the encoder block consisting of a *multi-head attention* layer, a *dropout* layer, a *add&norm* layer as well as a *feed-forward networks* and another subsequent *dropout* and *add&norm* layers. As already explained in section 3.2, the encoder output is then fed into the decoder part of the transformer. After a transformation in the form of an equivalent *continuous embedding* as well as *positional encoding* within the decoder input to $\mathbf{u}_T^{(i),\text{Decoder}} \in \mathbb{R}^{1\times d_{model}}$, the decoder block then occurs, with the encoder outputs in the first component of the *multi-head attention* as additional input data according to the attention mechanism from section 3.2.2. Due to the one-dimensional structure of the decoder, the masked multi-head attention and the add&norm layer based on it are eliminated. This results in a decoder structure in which, analogous to the vanilla transformer, this is followed by a *dropout* layer, a *add&norm* layer as well as a *feed-forward network* and then another *dropout* and *add&norm* layers. With regard to the requirements of the decoder output in the form of the asset-specific TSMOM position sizes, further changes need to be made in the output layer compared to the vanilla transformer, whereby the last softmax function generates probability-like instead of weight-like outputs. Therefore, a *tanh layer* is used as the output layer, which by definition generates outputs on the interval $[-1,1]$ due to its properties. There is also a *linear dense* layer upstream, which transforms the encoder outputs of dimension $\mathbb{R}^{1\times d_{model}}$ into a scalar value, which is then passed to the tanh layer for the position size transformation.

After the generation of the position sizes in the form of the decoder output has been explained, the final step is to define the loss function, which is of key importance in terms of the model performance. In contrast to the previously derived proposed CSMOM transformer models, it is possible to directly optimize a portfolio performance metric since the transformer output are direct position sizes. In line with existing research such as Wood et al. (2022), the sharpe ratio

is used, which results in the following sharpe loss function:

$$L\left(\mathbf{y}_{T+1}^{(i)}, f\left(\mathbf{u}_{T}^{(i)}\right)\right) = -\frac{\sqrt{\frac{n_{trading}}{n_{rebalance}}}R_{T+1}^{(i)}}{\sqrt{Var\left(R_{T+1}^{(i)}\right)}} \tag{117}$$

where $n_{trading}$ is defined as the number of trading days per year and $n_{rebalance}$ the portfolio re-balance frequency in days, while R_{T+1}^{i} is considered the asset-specific realized strategy return from time T to time $T+1$, derived from the position size $f\left(\mathbf{u}_{T}^{(i)}\right)$ and the volatility-scaled cumulative asset return $\mathbf{y}_{T+1}^{(i)}$. As can already be seen from (117), for reasons of simplification, the additional consideration of the abnormal return in comparison to the market return is omitted, which means that the addition of the market return or risk-free return is no longer necessary, which is in line with existing research work (Wood et al., 2022, p.5). Furthermore, the realized strategy return can still be formalized more explicitly. In the course of this work, as in the context of the CSMOM strategies, volatility-scaled returns are used, whereby the realized strategy return takes the following form in accordance with the definition by Wood et al. (2022, p.5):

$$r_{t+1}^{(i)} = z_{t}^{(i)}\frac{\sigma_{target}}{\sigma_{t}^{(i)}}r_{t+1}^{(i)} \tag{118}$$

where $\frac{\sigma_{target}}{\sigma_{t}^{(i)}}$ represents the volatility scaling component. This volatility scaling ensures that, despite the different volatility of i at times t, each asset has a similar contribution to the overall portfolio return in the portfolio construction and that the overall portfolio performance is not only driven by individual out-performing assets (Wood et al., 2022, p.5) which ensures the evaluation of a holistic TSMOM strategy. This approach is also in line with existing research results in the TSMOM literature (Wood et al., 2021, Kim et al., 2016, Campbell et al., 2018). In summary, the entire TSMOM Encoder-Decoder Transformer can be formalized and schematically visualized as follows:

$$
\begin{aligned}
z_T^{(i)} = f\left(\mathbf{u}_T^{(i)}\right) &= \text{Tanh}\left(\text{Linear}\left(\text{Dec}_N\left(\cdots\left(\text{Dec}_1\left(\text{PE}\left(\mathbf{u}_T^{(i)}\right) + \text{Emb}\left(\mathbf{u}_T^{(i)}\right)\right)\right)\right)\right)\right) \\
\text{Dec}_j\left(\mathbf{h}_t^{(i)}\right) &= \text{Norm}\left(\mathbf{h}_T^{(i)} + \text{Dropout}\left(\text{FF}\left(\mathbf{h}_T^{(i)}\right)\right)\right) \\
\mathbf{h}_T^{(i)} &= \text{Norm}\left(\mathbf{x}_T^{(i)} + \text{Dropout}\left(\text{MHA}\left(\mathbf{z}_{T-\tau+1:T-1}^{(i)}; \mathbf{x}_T^{(i)}\right)\right)\right) \\
\mathbf{x}_T^{(i)} &= \begin{cases} \text{PE}\left(\mathbf{u}_T^{(i)}\right) + \text{Emb}\left(\mathbf{u}_T^{(i)}\right), & \text{if } j = 1 \\ \text{Dec}_{j-1}\left(\cdot\right), & \text{if } j \geq 1 \end{cases} \\
\mathbf{z}_{T-\tau+1:T-1}^{(i)} &= \text{Enc}_N\left(\cdots\left(\text{Enc}_1\left(\text{PE}\left(\mathbf{U}_{T-\tau+1:T-1}^{(i)}\right) + \text{Emb}\left(\mathbf{U}_{T-\tau+1:T-1}^{(i)}\right)\right)\right)\right) \\
\text{Enc}_j\left(\mathbf{H}_{T-\tau+1:T-1}\right) &= \text{Norm}\left(\mathbf{H}_{T-\tau+1:T-1}^{(i)} + \text{Dropout}\left(\text{FF}\left(\mathbf{H}_{T-\tau+1:T-1}^{(i)}\right)\right)\right) \\
\mathbf{H}_{T-\tau+1:T-1}^{(i)} &= \text{Norm}\left(\mathbf{X}_{T-\tau+1:T-1}^{(i)} + \text{Dropout}\left(\text{MHA}\left(\mathbf{X}_{T-\tau+1:T-1}^{(i)}\right)\right)\right) \\
\mathbf{X}_{T-\tau+1:T-1}^{(i)} &= \begin{cases} \text{PE}\left(\mathbf{U}_{T-\tau+1:T-1}^{(i)}\right) + \text{Emb}\left(\mathbf{U}_{T-\tau+1:T-1}^{(i)}\right), & \text{if } j = 1 \\ \text{Enc}_{j-1}\left(\cdot\right), & \text{if } j \geq 1 \end{cases} \\
\mathbf{U}_T^{(i)} &\in \mathbb{R}^{1 \times p} \\
\mathbf{U}_{T-\tau+1:T-1}^{(i)} &\in \mathbb{R}^{(\tau-1) \times p}
\end{aligned}
\tag{119}
$$

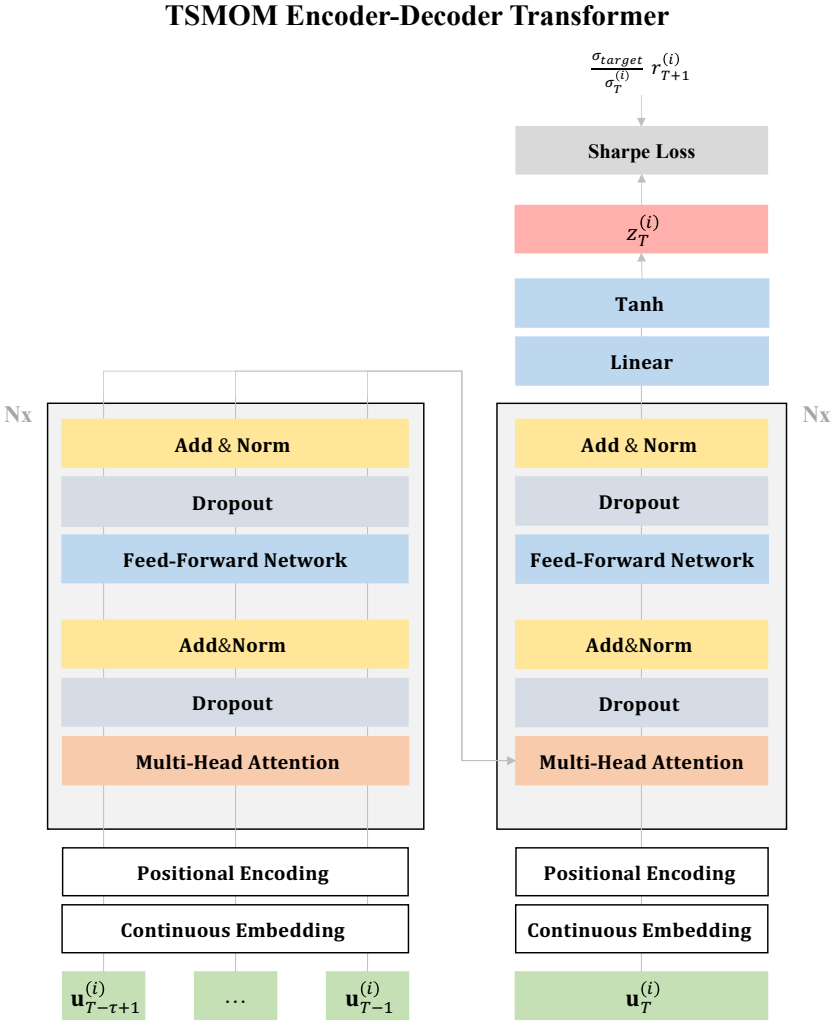


Figure 21: TSMOM Encoder-Decoder Transformer - Own Illustration

Based on figure 21, the TSMOM Encoder-Decoder Transformer model can be summarized as follows: For each asset i separately, their corresponding feature vectors $\mathbf{u}_t^{(i),\text{Encoder}} \in \mathbf{U}_{T-\tau+1:T-1}^{(i),\text{Encoder}} \in \mathbb{R}^{n_t \times p}$ for $t \in [T-\tau, T-1]$ are fed into the transformer encoder and are transformed by the continuous embedding and by the positional encoding layer into $\mathbf{U}_{T-\tau+1:T-1}^{(i),\text{Encoder}} \in \mathbb{R}^{n_t \times d_{model}}$, after which the transformed input data are fed into the encoder blocks with each having a multi-head attention layer, a dropout layer, an add&norm layer, a feed-forward neural network layer, again a dropout layer and another final add&norm layer. The output of the encoder is then finally fed into the decoder, in particular into the multi-head attention layer. The input of the decoder is given by the input vector $\mathbf{u}_T^{(i),\text{Decoder}}$, which is equivalently transformed by the continuous embedding as well as the positional encoding layer into $\mathbf{u}_T^{(i),\text{Decoder}} \in \mathbb{R}^{1 \times d_{model}}$. Afterwards, the decoder input data is fed into the multi-head attention layer together with the encoder output, after which a dropout layer, an add&norm layer, a feed-forward neural network layer, again a dropout layer and another final add&norm layer follow. In order to transform the decoder output back into a scalar value, a linear dense layer is applied, after which the final output dense layer with tanh activation follows in order to transform the scalar into a position size as $z_T^{(i)} \in [-1, 1]$. Corresponding to the sharpe loss definition, the volatility-scaled return between T and $T+1$ are used as additional inputs for the loss calculation. Lastly, since only one position

size for one asset i at time T is predicted, the loss calculation happens across multiple batches of at least all assets once and the portfolio construction for time T requires the additional position size predictions from the other asset iterations for the same time.

4.4.2 OK TSMOM Decoder-Only Transformer

Based on the research findings by Li et al. (2020) and other empirical results, it can be seen that using the decoder part of the vanilla transformer architecture without information ingestion from the encoder part through the additional multi-head self-attention layer can be sufficient within the framework of time series forecasting. This represents the underlying motivation to design a *Decoder-Only Transformer* with regard to simultaneous trend estimation and position sizing for TSMOM strategies and to compare the performance against the encoder-decoder equivalent as well as existing benchmark strategies.

As part of the TSMOM Decoder-Only Transformer, the architecture for the proposed TSMOM strategy is closely related to the vanilla transformer from section 3.2. The input layer is centrally defined by the deterministic look-back period τ, which in turn defines the time interval that is used to simultaneously derive the trend estimation and the final position sizing. This results in an input matrix $\mathbf{U}^{(i)}_{T-\tau+1:T}$ separately for each asset i, which in turn consists of real-valued, asset-specific input feature vectors $\mathbf{u}^{(i)}_t \in \mathbf{U}^{(i)}_{T-\tau+1:T}$, hence leading to $\mathbf{u}^{(i)}_t \in \{\mathbf{u}^{(i)}_{T-\tau+1}, ..., \mathbf{u}^{(i)}_T\}$. It results that, for each asset i, the position sizes $\mathbf{z} = (z_t)^T_{t=T-\tau+1}$ and implicitly the trend estimation for each time step from $T-\tau+1$ to T are predicted and thus a delta is created in the starting period of the training iterations of τ.

After the structure of the input layer has been described in detail, the next step contains the definition of the decoder components within the transformer. The first step is given by *continuous embedding*, which is achieved by a linear transformation of the input feature vectors $\mathbf{z}^{(i)}_t$ from dimension $1 \times p$ to $1 \times d_{model}$ as well as a *positional encoding* applied to it, which corresponds to the original transformer architecture proposed by Vaswani et al. (2017, p.6). The positional transformation is then fed into the *decoder block*, which differs from the vanilla transformer in section 3.3 only in terms of the lack of multi-head attention with input from the encoder block. Thus, within the framework of this proposed architecture, the decoder block only consists of a *masked multi-head attention*, in which the mask prevents a look-ahead bias across the look-back time series, as well as a *dropout layer* and a *add & norm layer*. This is followed by a *feed-forward network*, which is again followed by a *dropout layer* and *add & norm layer*. The dropout layer primarily serves to prevent overfitting, as was already explained in the introduction to deep learning models. That decoder block can then be stacked N-times. In the last step, the decoder block output serves as input into a *linear dense layer*, which in turn transforms the feature vectors of the dimension $1 \times d_{model}$ element-wise into scalar values, which are then finally transformed element-wise by a *tanh layer* according to the activation function definition in (16). This last step ensures, through the properties of the tanh function, that the final position sizes have the properties of leverage-limited portfolio weights $z^{(i)}_t \in [-1, 1]$.

Following the prediction of the position sizes, the final component of the TSMOM Decoder-Only

Transformer needs to be defined in the form of the loss function. Equivalent to the proposed TSMOM Encoder-Decoder Transformer from the previous section, the *sharpe ratio* is used as the loss-underlying performance metric in line with existing TSMOM benchmark strategies Wood et al., 2022, p.5. Thus, the following sharpe loss function can be defined:

$$L\left(\mathbf{y}^{(i)}_{T-\tau:T+1}, f\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right)\right) = -\frac{\sqrt{\frac{n_{trading}}{n_{rebalance}}}\mathbf{R}^{(i)}_{T-\tau:T+1}}{\sqrt{Var\left(\mathbf{R}^{(i)}_{T-\tau:T+1}\right)}} \tag{120}$$

where $n_{trading}$ is the number of trading days per year and $n_{rebalance}$ the portfolio re-balance frequency in days. Furthermore, R^i_t is considered the asset-specific realized strategy return from time $t-1$ to time t, calculated from the position sizes $f\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right)$ and the volatility-scaled cumulative asset returns $\mathbf{y}^{(i)}_{T-\tau:T+1}$. In line with the sharpe loss definition for the TSMOM Encoder-Decoder Transformer, the additional consideration of the abnormal return is omitted in line with existing research work (Wood et al., 2022, p.5). Finally, based on the general TSMOM strategy framework, the realized strategy return takes the following form in accordance with the definition by Wood et al. (2022, p.5):

$$r^{(i)}_{t+1} = z^{(i)}_t \frac{\sigma_{target}}{\sigma^{(i)}_t} r^{(i)}_{t+1} \tag{121}$$

where $\frac{\sigma_{target}}{\sigma^{(i)}_t}$ represents the volatility scaling component. As elaborated before, the volatility scaling ensures that each asset has a similar contribution to the overall portfolio return (Wood et al., 2022, p.5). After all components of the TSMOM Decoder-Only Transformer have been defined, the entire model architecture can be formalized and schematically visualized as follows:

$$\begin{aligned}
\mathbf{z}^{(i)}_{T-\tau+1:T} &= f\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right) \\
&= \text{Tanh}\left(\text{Linear}\left(\text{Dec}_N\left(\cdots\left(\text{Dec}_1\left(\text{PE}\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right) + \text{Emb}\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right)\right)\right)\right)\right)\right) \\
\text{Dec}_j\left(\mathbf{H}^{(i)}_{T-\tau+1:T}\right) &= \text{Norm}\left(\mathbf{H}^{(i)}_{T-\tau+1:T} + \text{Dropout}\left(\text{FF}\left(\mathbf{H}^{(i)}_{T-\tau+1:T}\right)\right)\right) \\
\mathbf{H}^{(i)}_{T-\tau+1:T} &= \text{Norm}\left(\mathbf{X}^{(i)}_{T-\tau+1:T} + \text{Dropout}\left(\text{MHA}\left(\mathbf{X}^{(i)}_{T-\tau+1:T}\right)\right)\right) \\
\mathbf{X}^{(i)}_{T-\tau+1:T} &= \begin{cases} \text{PE}\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right) + \text{Emb}\left(\mathbf{U}^{(i)}_{T-\tau+1:T}\right), & \text{if } j = 1 \\ \text{Dec}_{j-1}\left(\cdot\right), & \text{if } j \geq 1 \end{cases} \\
\mathbf{U}^{(i)}_{T-\tau+1:T} &\in \mathbb{R}^{\tau \times p}
\end{aligned} \tag{122}$$

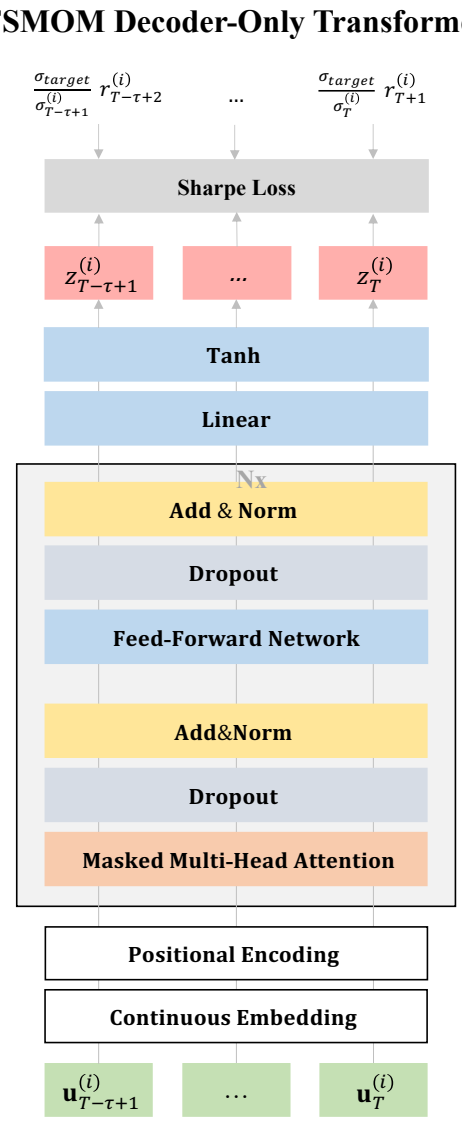


Figure 22: TSMOM Decoder-Only Transformer - Own Illustration

Based on figure 22, the TSMOM Decoder-Only Transformer model can be summarized as follows: For each asset i separately, their corresponding feature vectors $\mathbf{u}_t^{(i)} \in \mathbf{U}_{T-\tau+1:T}^{(i)} \in \mathbb{R}^{\tau \times p}$ are fed into the transformer decoder and are transformed by the continuous embedding and by the positional encoding layer into $\mathbf{U}_{T-\tau+1:1}^{(i)} \in \mathbb{R}^{\tau \times d_{model}}$, after which the transformed input data are fed into the decoder blocks with each having a multi-head attention layer, a dropout layer, an add&norm layer, a feed-forward neural network layer, again a dropout layer and another final add&norm layer. In order to transform the decoder output back into a scalar value, a linear dense layer is applied, after which the final output dense layer with tanh activation follows in order to transform the scalar into a position size as $z_t^{(i)} \in [-1, 1] \in \mathbf{z}_{T-\tau+1:T}^{(i)}$. Corresponding to the sharpe loss definition, the volatility-scaled return between t and $t+1$ are used as additional inputs for the loss calculation. Lastly, since multiple position sizes are predicted at once, the training start interval is given by the look-back period τ.

4.4.3 OK TSMOM Strategy Assumptions & Requirements

After the proposed TSMOM models have been defined as the TSMOM Encoder-Decoder Transformer and the TSMOM Decoder-Only Transformer, the deterministic strategy parameters to be defined, while the hyperparameters are explained in more detail during the strategy back-testing.

Re-balance Frequency

In the course of general portfolio strategies, the re-balance frequency represents a central variable that has a fundamental influence, especially with regard to transaction cost effects, strategy responsiveness and the computational performance of automated trading strategies. The TSMOM strategies proposed here are re-balanced monthly in accordance with the previously derived CSMOM strategies as well as in accordance with existing benchmark strategies, in particular to prevent excessive transaction cost effects that occur in the context of short rebellion frequencies (Poh et al., 2020, p.3). The re-balance frequency can therefore be defined as follows:

TSMOM Encoder-Decoder Transformer	**TSMOM Decoder-Only Transformer**
$n_{rebalance} = 21$	$n_{rebalance} = 21$

Table 5: TSMOM Strategy Parameters - Re-balance Frequency

where 21 days are derived as an approximation of the trading days within a month over approximately 252 trading days per year. Implicitly, it means that each position size $z_t^{(i)}$ in the form the output of the encoder-decoder or the decoder-only transformer at time T remains constant over the days within the following trading month. Furthermore, it follows that the sharpe ratio is calculated in the course of the sharpe loss function by adding the cumulative volatility-adjusted monthly returns as well as the predicted position sizes for each asset i and each time step t, resulting in the following volatility-adjusted returns as inputs for the sharpe loss:

$$r^{(i)}_{\tau_m,\tau_{m+1}} = z_t^{(i)} \frac{\sigma_{target}}{\sigma^{(i)}_{\tau_m}} r^{(i)}_{\tau_m,\tau_{m+1}} \tag{123}$$

Look-Back Period

With reference to the proposed TSMOM model architectures, the look-back period for defining the time series interval represents a central modelling variable. In accordance with existing research work and benchmark models, the deterministic look-back period is defined as follows:

TSMOM Encoder-Decoder Transformer	**TSMOM Decoder-Only Transformer**
$\tau = \frac{252}{n_{rebalance}} = \frac{252}{21} = 12$	$\tau = \frac{252}{n_{rebalance}} = \frac{252}{21} = 12$

Table 6: TSMOM Strategy Parameters - Look-Back Period

where $n_{rebalance}$ is defined according to table 5 and 252 days is used as an approximation of the trading days per year. This ultimately results in the look-back period being twelve months and

monthly time intervals being set in accordance with the re-balance frequency for each model iteration for each asset i and each time step t, whereby the delta in the model training, validation and prediction is implicitly given by twelve months as well and will be further outlined in section 6.

Target & Realized Volatility

Based on the proposed TSMOM models, the target volatility and the realized volatility in the course of the volatility-scaled returns ultimately serve as the external input of the sharpe loss calculation. In line with benchmark TSMOM strategies by Wood et. al (2022, p.5) and as well as other research work (Moskowitz et al., 2012; Lim, Zohren, et al., 2020; Wood et al., 2021), the target volatility is defined as follows:

TSMOM Encoder-Decoder Transformer	**TSMOM Decoder-Only Transformer**
$\sigma_{target} = 15\%$	$\sigma_{target} = 15\%$

Table 7: TSMOM Strategy Parameters - Target Volatility

After the target volatility has been fixed as the numerator of the volatility-scaled returns according to (123), the derivation of the realized volatility has to be derived. There are dozens of model specifications, ranging from econometric to machine learning approaches. In the course of this work, however, it is important to establish comparability with benchmark models and existing research work, which means that ex-ante volatility is used in the form of the exponentially weighted moving standard deviation with a predefined volatility look-back period, in accordance with the CSMOM strategy setup. As previously elaborated, this approach is suitable for portfolios with an approximate diagonal covariance matrix (Wood et al., 2022, p.5). This means that the realized portfolio volatility is approximated from the asset-specific volatilities and the covariance can be neglected due to the negligible correlation between the assets. This results in the following definition of the realized volatility:

TSMOM Encoder-Decoder Transformer	**TSMOM Decoder-Only Transformer**
$\sigma_t^{(i)} = EWMSD\left(\left(r_t^{(i)}\right)_{t=1}^{T}\right)$	$\sigma_t^{(i)} = EWMSD\left(\left(r_t^{(i)}\right)_{t=1}^{T}\right)$

Table 8: TSMOM Strategy Parameters - Realized Volatility

4.4.4 OK TSMOM Strategy Benchmark Models

In the course of evaluating the proposed TSMOM strategies, state-of-the-art benchmark strategies need to be defined in order to test the performance. As part of time series momentum, three central strategy types are used: First, a portfolio is defined with *Random* portfolio weights as the very baseline model equivalent to the CSMOM benchmark strategy. Second, the *Original TSMOM* strategy by Moskowitz et al. (2012), which to this day represent a central standard of TSMOM back-testing. Lastly, two machine learning-based models in the form of the *Recur-*

rent Neural Network (RNN) as well as its adaption *Long Short-Term Memory Neural Network* (LSTM), are applied, which are defined by definition for time series use cases. In the following sections, those strategies are explained and defined in more detail.

Random

As can already be seen from the terminology, within the framework of the baseline model, the portfolio weights are *randomly* defined for each asset i at each re-balance point on the interval $[-1, 1]$ in line with the underlying weight axioms. This results in the following derivation of the portfolio weights:

$$\textbf{Position Sizing Rule:} \quad z_t^{(i),\text{Random}} = \text{Rand}[-1, 1] \tag{124}$$

where $\text{Rand}[-1, 1]$ represents an expression for a random number on the closed interval $[-1, 1]$. Furthermore, concerning the re-balance frequency of 21 days for TSMOM strategies according to table 5, the realized strategy return equivalent to the general volatility-scaled TSMOM strategy return according to (123) with the addition of the position size definition can be defined as follows:

$$r_{t,t+1}^{\text{Random}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{Random}} \frac{\sigma_{target}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)} \tag{125}$$

where the position sizes are defined as given in (124), while all other components are defined as given in table 7 and 8.

Original TSMOM

The second class of benchmark strategies in the course of the TSMOM strategy evaluation are given by the *original TSMOM* strategies according to Moskowitz et al. (2012). In comparison to the random strategy, a trend estimation is explicitly derived, which is then incorporated into the specific position sizing rule, according to the generalized TSMOM framework in section 4.4. Here, the trend estimation is defined as the cumulative return over a predefined look-back period, whereupon the position sizing rule takes the sign of the trend estimation as an indicator for a maximum long or maximum short position (Lim, Zohren, et al., 2020, p.4). This results in the following trend estimation and position sizing rule (Lim, Zohren, et al., 2020, p.4):

$$\begin{aligned} \textbf{Trend Estimation:} \quad & s_t^{(i)} = r_{t-m,t}^{(i)} \\ \textbf{Position Sizing Rule:} \quad & z_t^{(i),\text{Original}} = \text{sign}\left(s_t^{(i)}\right) \in \{-1, 1\} \end{aligned} \tag{126}$$

where m is the central strategy parameter as the look-back period. According to Moskowitz et al. (2012), the strategy based on daily re-balancing is tested with look-back periods of three, six and twelve months. Analogously, the same look-back periods are used in this work, with monthly re-balancing being carried out, which can be formalized as follows:

$$m \in \{3, 6, 12\} \tag{127}$$

whereby the trend estimation in accordance with (126) is derived based on the three-month, six-month and twelve-month cumulative asset returns and the position sizing results in a maximum long or maximum short position according to the sign of those returns (Lim, Zohren, et al., 2020, p.4). Finally, the realized return of the original TSMOM strategies can be defined as follows:

$$r_{t,t+1}^{\text{Original}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{Original}} \frac{\sigma_{target}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)} \tag{128}$$

where all strategy components are defined as given in table 7 and 8.

Recurrent Neural Network (RNN)

The third class of benchmark strategies in the course of the TSMOM strategy evaluation is given by sharpe-optimizing *recurrent neural networks* (RNN). According to the model derivation in section 3.1.2, the recurrent mechanism allows for capturing temporal dynamics which builds the foundation of time series momentum strategies. In comparison to the previous benchmark strategies, the trend estimation is derived implicitly and incorporated into the specific position sizing rule, according to the generalized TSMOM framework in section 4.4. In addition, in order to make the vanilla RNN architecture from section 3.1.2 applicable to TSMOM strategies, specific architectural adjustments are required which lead to the following position size rule:

$$\begin{aligned} \textbf{Position Sizing Rule:} \quad z_{T-\tau+1:T}^{(i),\text{RNN}} &= f\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right) \\ &= \text{Tanh}\left(\text{RNN}\left(\text{Input}\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right)\right)\right) \in [-1,1] \end{aligned} \tag{129}$$

where an additional *tanh* layer is the output layer in order to ensure that the model outputs are considered as position sizes according to the generalized TSMOM strategy framework, while the RNN component is defined according to section 3.1.2. In addition, the model is set up as a many-to-many architecture which generates position sizes for every input time step, equivalently to the TSMOM Decoder-Only Transformer. As the final model component, the RNN requires a dedicated loss function for model training, whereby the sharpe loss function from the proposed TSMOM models is utilized:

$$L\left(\mathbf{y}_{T-\tau+1:T}^{(i)}, f\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right)\right) = -\frac{\sqrt{\frac{n_{trading}}{n_{rebalance}}}\mathbf{R}_{T-\tau:T+1}^{(i)}}{\sqrt{Var\left(\mathbf{R}_{T-\tau:T+1}^{(i)}\right)}} \tag{130}$$

where the sharpe loss function and its components are defined according to sections 4.4.1 and 4.4.2. Finally, the realized RNN strategy return in accordance with the generalized TSMOM strategy framework can be formalized as follows:

$$r_{t,t+1}^{\text{RNN}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{RNN}} \frac{\sigma_{target}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)} \tag{131}$$

where the position sizes are defined as given in (129), while all other components are defined as given in table 7 and 8.

Long Short-Term Memory Neural Network (LSTM)

The last class of benchmark strategies is given by sharpe-optimizing *long short-term memory neural networks* (LSTM), which define a further adaption of underlying RNN architectures. According to the model derivation in section 3.1.3, LSTMs allow for optimized capturing of long-term temporal dependencies, hence empirical out-performing against RNNs on large input sequences. Equivalent to the RNN benchmark strategy, a trend estimation is implicitly derived and incorporated into the specific position sizing rule. In order to make the vanilla LSTM architecture applicable to TSMOM strategies, specific architectural adjustments are necessary, resulting in the following position size rule:

$$\begin{aligned} \textbf{Position Sizing Rule:} \quad z_{T-\tau+1:T}^{(i),\text{LSTM}} &= f\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right) \\ &= \text{Tanh}\left(\text{LSTM}\left(\text{Input}\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right)\right)\right) \in [-1,1] \end{aligned} \tag{132}$$

where the LSTM block is defined according to (43) and the underlying LSTM memory cells according to figure 7. In addition, there is a direct position size output through the tanh output layer on the interval $[-1,1]$ for each time-step input, leading to a many-to-many LSTM architecture. In addition, equivalent to the architecture of the proposed transformer-based TSMOM models, the LSTM is directly trained on optimizing the sharpe ratio through the following sharpe loss function:

$$L\left(\mathbf{y}_{T-\tau+1:T}^{(i)}, f\left(\mathbf{U}_{T-\tau+1:T}^{(i)}\right)\right) = -\frac{\sqrt{\frac{n_{trading}}{n_{rebalance}}}\mathbf{R}_{T-\tau:T+1}^{(i)}}{\sqrt{Var\left(\mathbf{R}_{T-\tau:T+1}^{(i)}\right)}} \tag{133}$$

where the sharpe loss function and its components are defined according to sections 4.4.1 and 4.4.2. Finally, the realized LSTM strategy return in accordance with the generalized TSMOM strategy framework can be formalized as follows:

$$r_{t,t+1}^{\text{LSTM}} = \frac{1}{n_t} \sum_{i=1}^{n_t} z_t^{(i),\text{LSTM}} \frac{\sigma_{target}}{\sigma_t^{(i)}} r_{t,t+1}^{(i)} \tag{134}$$

where the position sizes are defined as given in (132), while all other components are defined as given in table 7 and 8.

5 OK Multi-Asset Data Set

The underlying data set for back-testing the proposed multi-asset CSMOM and TSMOM strategies against the corresponding benchmark strategies is given by 50 future contracts from the Pinnacle Data Corp CLC database (Corp, 2023). That data set represents a frequently used data foundation, especially in the context of back-testing CSMOM and TSMOM strategies, see Poh et al. (2022), Lim et al. (2020) or Tan et al. (2023). The underlying asset classes within the data set enable a well-balanced multi-asset portfolio of *equities*, *bonds*, *currencies* and *commodities* as well as different geographical focuses (Corp, 2023). Nevertheless, the issue of the expiration of futures contracts leads to a non-continuous price series with jumps, which requires dedicated data adjustments. Hence, the data-generating process is first elaborated, after which the data set is further investigated in terms of its descriptive statistics and properties.

5.1 OK Data Generating Process (DGP)

A central challenge of working with futures contracts is given by the fact that those contracts expire over time and a rollover has to be carried out by investors so that the existing contract is closed and a new contract is opened directly, resulting in a non-continuous time series with jumps, whereby those jumps exactly arise at the rollover dates (Vojtko and Padysak, 2020, p.2). However, in order to be able to use the data set in the course of strategy back-testing and thus transform it into a continuous price series, it needs to be transformed into continuous futures contracts, which can be done using various empirical methods (Vojtko and Padysak, 2020). In the context of this work, in accordance with existing approaches from CSMOM and TSMOM research, this is done using *backwards ratio adjustments*. Given an existing future contract price and the new future contract price at the rollover date t, a time-specific adjustment ratio is established, whereupon each historical future contract price before the rollover date back to the previous rollover date $t-1$ is adjusted with the ratio to smooth the series at the rollover dates (Vojtko and Padysak, 2020, p.3-4). This can thus be formalized as follows:

$$R_t = \frac{P_t^{\text{old}}}{P_t^{\text{new}}} \quad , \quad P_t^{\text{adjusted}} = P_t^{\text{historical}} \prod_{j=1}^{t} R_j \tag{135}$$

where $\prod_{j=1}^{t}\dot{()}$ represents the backward interval from rollover date t to the previous rollover date $t-1$ and thus all prices within this interval are adjusted with the ratio (Vojtko and Padysak, 2020, p.3-4). After the continuous future price series has been ensured, the handling of missing and outlier values must be defined. In this context, only futures contracts that have less than 10% of missing data are used, which is in accordance with Lim et al. (2020, p.16). This condition can be formalized as follows:

$$\delta(x) = \begin{cases} 1, & \text{if } x \text{ is missing} \\ 0, & \text{otherwise} \end{cases} \quad , \quad \sum_{i=1}^{n} \delta \mathbf{M}_{ij} \leq 0.1n \tag{136}$$

where $\mathbf{M}$ represents the matrix of continuous futures contract prices with the dates on the

rows and assets on the columns. This means that only assets that have less than 10% missing data are retained. In addition, missing values within the assets that meet the condition are replaced with a forward-filling method, whereby a missing value at time t is replaced with the value at time $t-1$. After the handling of missing values has been defined, the final step is to derive a deterministic rule for outlier detection to account for data issues. This rule is based on existing research approaches, where data is limited to be within 5 times the exponentially weighted moving (EWM) standard deviations from the EWM average, using a 252-day half-life, in accordance with Wood et al. (2022, p.15), which can formalized as follows:

$$\mathbf{M}_{ij} = \begin{cases} \mu_{t,j} + 5\sigma_{t,j}, & \text{if } \mathbf{M}_{ij} > \mu_{t,j} + 5\sigma_{t,j} \\ \mu_{t,j} - 5\sigma_{t,j}, & \text{if } \mathbf{M}_{ij} < \mu_{t,j} - 5\sigma_{t,j} \\ \mathbf{M}_{ij}, & \text{otherwise} \end{cases} \tag{137}$$

where $\mu_{t,j}$ is defined as the exponentially weighted moving average and $\sigma_{t,j}$ as the exponentially weighted moving standard deviation, each with a half-life of 252 days. After applying the rules for handling missing data and outliers, the resulting data set represents the foundation for back-testing the proposed and benchmark multi-asset CSMOM and TSMOM strategies.

5.2 OK Descriptive Statistics & Properties

After the underlying data set of continuous futures contracts was derived, a further data investigation needs to be carried out, with the purpose of better understanding the underlying structure and patterns of the data set, uncovering any remaining data anomalies and forming expectations and hypotheses regarding the back-testing. The first step is to provide an overview of the continuous futures contracts in the table below in terms of the description of the underlying assets, the starting date of the available continuous price series, the associated asset class and the geographical focus:

	Asset	Asset Class	Start Date	Geography
HS	Hang Seng, Index	EQUITIES (EQ)	1997-10-22	Hong Kong
AX	DAX 40, Index	EQUITIES (EQ)	1997-11-05	Germany
LX	FTSE 100, Index	EQUITIES (EQ)	1990-01-02	Great Britian
ES	S&P 500, Mini	EQUITIES (EQ)	1997-09-11	United States of America
MD	S&P 400, Index	EQUITIES (EQ)	1992-02-13	United States of America
NK	Nikkei, Index	EQUITIES (EQ)	1991-01-02	Japan
FB	US T-Note (5y) (Comp)	FIXED INCOME (FI)	1989-01-03	United States of America
EC	Eurodollar (Comp)	FIXED INCOME (FI)	1982-02-01	Euro Zone
DT	Euro Bond (Bund)	FIXED INCOME (FI)	1989-11-23	Euro Zone
TU	US T-Note (2y) (Comp)	FIXED INCOME (FI)	1990-09-20	United States of America
TY	US T-Note (10y) (Comp)	FIXED INCOME (FI)	1983-01-03	United States of America
CB	Canada Govt Bond (10yr)	FIXED INCOME (FI)	1994-06-01	Canada
US	US T-Bonds (Comp)	FIXED INCOME (FI)	1978-01-03	United States of America
GS	Long Gilt	FIXED INCOME (FI)	1989-11-23	Great Britian
SN	Swiss Franc	FOREIGN EXCHANGE (FX)	1976-01-02	Switzerland
JN	Japanese Yen	FOREIGN EXCHANGE (FX)	1978-01-03	Japan
AN	Australian Dollar (Comp)	FOREIGN EXCHANGE (FX)	1988-01-04	Australia
MP	Mexican Peso	FOREIGN EXCHANGE (FX)	1995-05-03	Mexico
BN	British Pound	FOREIGN EXCHANGE (FX)	1976-01-02	Great Britian
FN	Euro (Comp)	FOREIGN EXCHANGE (FX)	1975-02-13	Euro Zone
DX	US Dollar Index	FOREIGN EXCHANGE (FX)	1986-01-02	United States of America
CN	Canadian Dollar	FOREIGN EXCHANGE (FX)	1978-01-03	Canada
JO	Orange Juice	COMMODITIES (CM)	1973-01-02	Global
ZK	Copper	COMMODITIES (CM)	1989-01-03	Global
ZL	Soyoil	COMMODITIES (CM)	1969-01-02	Global
ZM	Soymeal	COMMODITIES (CM)	1969-01-02	Global
ZN	Natural Gas	COMMODITIES (CM)	1991-01-02	Global
ZO	Oats	COMMODITIES (CM)	1975-01-02	Global
ZP	Platinum	COMMODITIES (CM)	1973-01-02	Global
ZR	Rough Rice	COMMODITIES (CM)	1987-01-02	Global
ZS	Soybeans	COMMODITIES (CM)	1969-01-02	Global
ZT	Live Cattle	COMMODITIES (CM)	1971-01-04	Global
ZU	Crude Oil	COMMODITIES (CM)	1984-01-03	Global
ZI	Silver	COMMODITIES (CM)	1973-01-02	Global
ZG	Gold	COMMODITIES (CM)	1975-01-02	Global
ZB	Gasoline	COMMODITIES (CM)	1985-01-02	Global
ZC	Corn	COMMODITIES (CM)	1970-01-05	Global
GI	Goldmnan Saks C.I.	COMMODITIES (CM)	1993-01-04	Global
ZA	Paladium	COMMODITIES (CM)	1983-01-03	Global
CC	Cocoa	COMMODITIES (CM)	1981-01-05	Global
CT	Cotton	COMMODITIES (CM)	1973-01-02	Global
SB	Sugar	COMMODITIES (CM)	1975-01-02	Global
DA	Milk	COMMODITIES (CM)	1997-09-02	Global
MW	Wheat (Mini)	COMMODITIES (CM)	1981-01-02	Global
W	Wheat	COMMODITIES (CM)	1969-01-02	Global
LB	Lumber	COMMODITIES (CM)	1974-01-02	Global
KW	Wheat (KC)	COMMODITIES (CM)	1977-01-03	Global
KC	Coffee	COMMODITIES (CM)	1974-01-02	Global
ZF	Feeder Cattle	COMMODITIES (CM)	1978-01-03	Global
ZZ	Lean Hogs	COMMODITIES (CM)	1970-01-05	Global

Table 9: Multi-Asset Data Set - Overview

From table 9, it can be seen that there is sufficient diversification in terms of asset classes as well as geographical focuses. The available future contracts cover the asset classes of equities, bonds, currencies and commodities, which is in line with benchmark multi-asset strategies. Based on the observations of the start dates, a fixed observation period is chosen, which is set from 1999-01-01 to 2022-12-31. Based on the time window for back-testing, in the second step, the descriptive statistics are investigated, whereby, for the continuous price series of each available future contract, the number of observations, the number of unique values, as well as the mean, median, standard deviation, min and max value, as well as the lower and upper quantiles, are described:

	Start Date	End Date	Obs	Unique	Mean	Median	Std	Min	Max	Q1	Q3
AN	1999-01-05	2022-12-30	6212	5906	0.0000	0.0001	0.0078	-0.0951	0.0622	-0.0038	0.0043
AX	1999-01-05	2022-12-30	6212	6066	0.0000	0.0006	0.0149	-0.1275	0.1705	-0.0064	0.0072
BN	1999-01-05	2022-12-30	6212	5981	-0.0000	0.0000	0.0058	-0.0778	0.0341	-0.0032	0.0032
CB	1999-01-05	2022-12-30	6212	5853	0.0000	0.0000	0.0036	-0.0199	0.0229	-0.0020	0.0022
CC	1999-01-05	2022-12-30	6212	5815	0.0000	0.0000	0.0182	-0.0951	0.1048	-0.0100	0.0103
CN	1999-01-05	2022-12-30	6212	5942	0.0000	0.0000	0.0053	-0.0371	0.0525	-0.0028	0.0030
CT	1999-01-05	2022-12-30	6212	5975	-0.0000	0.0000	0.0166	-0.0713	0.0742	-0.0088	0.0088
DA	1999-01-05	2022-12-30	6212	5368	0.0000	0.0000	0.0138	-0.0996	0.0692	-0.0070	0.0072
DT	1999-01-05	2022-12-30	6212	5970	0.0000	0.0001	0.0036	-0.0204	0.0201	-0.0018	0.0021
DX	1999-01-05	2022-12-30	6212	6045	0.0000	0.0000	0.0050	-0.0271	0.0239	-0.0028	0.0028
EC	1999-01-05	2022-12-30	6212	4234	0.0000	0.0000	0.0004	-0.0036	0.0047	-0.0001	0.0001
ES	1999-01-05	2022-12-30	6212	5972	0.0000	0.0004	0.0123	-0.1039	0.1411	-0.0047	0.0057
FB	1999-01-05	2022-12-30	6212	5870	0.0000	0.0000	0.0024	-0.0190	0.0197	-0.0012	0.0013
FN	1999-01-05	2022-12-30	6212	5814	-0.0000	0.0000	0.0059	-0.0301	0.0334	-0.0033	0.0032
GI	1999-01-05	2022-12-30	6212	5965	0.0000	0.0001	0.0148	-0.1198	0.0769	-0.0076	0.0080
GS	1999-01-05	2022-12-30	6212	5979	0.0000	0.0000	0.0041	-0.0305	0.0423	-0.0021	0.0023
HS	1999-01-05	2022-12-30	6212	5893	0.0000	0.0000	0.0154	-0.1082	0.1198	-0.0070	0.0077
JN	1999-01-05	2022-12-30	6212	5941	-0.0000	0.0000	0.0062	-0.0392	0.0533	-0.0035	0.0030
JO	1999-01-05	2022-12-30	6212	5925	0.0000	0.0000	0.0192	-0.1277	0.2010	-0.0096	0.0100
KC	1999-01-05	2022-12-30	6212	5937	-0.0000	0.0000	0.0212	-0.1253	0.2361	-0.0120	0.0102
KW	1999-01-05	2022-12-30	6212	5931	-0.0000	0.0000	0.0176	-0.0860	0.0843	-0.0107	0.0097
LB	1999-01-05	2022-12-30	6212	5877	-0.0000	0.0000	0.0204	-0.0883	0.1093	-0.0128	0.0120
LX	1999-01-05	2022-12-30	6212	6025	0.0000	0.0001	0.0116	-0.0959	0.1005	-0.0052	0.0058
MD	1999-01-05	2022-12-30	6212	5982	0.0000	0.0005	0.0140	-0.1630	0.1187	-0.0059	0.0071
MP	1999-01-05	2022-12-30	6212	5863	0.0000	0.0003	0.0071	-0.0740	0.0670	-0.0032	0.0038
MW	1999-01-05	2022-12-30	6212	5927	0.0000	0.0000	0.0152	-0.0766	0.0821	-0.0084	0.0082
NK	1999-01-05	2022-12-30	6212	6055	0.0000	0.0004	0.0148	-0.1075	0.1407	-0.0072	0.0081
SB	1999-01-05	2022-12-30	6212	5770	0.0000	0.0000	0.0196	-0.1163	0.0896	-0.0110	0.0113
SN	1999-01-05	2022-12-30	6212	5927	0.0000	0.0000	0.0067	-0.0850	0.1698	-0.0034	0.0034
TU	1999-01-05	2022-12-30	6212	5549	0.0000	0.0000	0.0009	-0.0092	0.0087	-0.0004	0.0004
TY	1999-01-05	2022-12-30	6212	5885	0.0000	0.0000	0.0037	-0.0240	0.0363	-0.0019	0.0022
US	1999-01-05	2022-12-30	6212	5895	0.0000	0.0000	0.0063	-0.0411	0.0406	-0.0034	0.0039
ZA	1999-01-05	2022-12-30	6212	5985	0.0000	0.0002	0.0216	-0.2119	0.2578	-0.0102	0.0117
ZB	1999-01-05	2022-12-30	6212	6003	0.0000	0.0006	0.0234	-0.2319	0.2126	-0.0113	0.0133
ZC	1999-01-05	2022-12-30	6212	5855	-0.0000	0.0000	0.0164	-0.0780	0.0905	-0.0090	0.0086
ZF	1999-01-05	2022-12-30	6212	5942	0.0000	0.0000	0.0096	-0.0583	0.0550	-0.0049	0.0053
ZG	1999-01-05	2022-12-30	6212	5964	0.0000	0.0000	0.0108	-0.0935	0.0922	-0.0047	0.0056
ZI	1999-01-05	2022-12-30	6212	5978	0.0000	0.0002	0.0190	-0.1771	0.1328	-0.0078	0.0094
ZK	1999-01-05	2022-12-30	6212	5964	0.0000	0.0000	0.0165	-0.1105	0.1235	-0.0079	0.0086
ZL	1999-01-05	2022-12-30	6212	5879	0.0000	0.0000	0.0151	-0.0885	0.0843	-0.0086	0.0086
ZM	1999-01-05	2022-12-30	6212	5942	0.0000	0.0000	0.0160	-0.0779	0.0782	-0.0085	0.0095
ZN	1999-01-05	2022-12-30	6212	5996	-0.0000	0.0000	0.0319	-0.1653	0.2064	-0.0184	0.0166
ZO	1999-01-05	2022-12-30	6212	5718	0.0000	0.0000	0.0191	-0.1125	0.1174	-0.0103	0.0108
ZP	1999-01-05	2022-12-30	6212	5990	0.0000	0.0003	0.0153	-0.1159	0.1184	-0.0076	0.0085
ZR	1999-01-05	2022-12-30	6212	5888	-0.0000	0.0000	0.0147	-0.0934	0.1030	-0.0083	0.0076
ZS	1999-01-05	2022-12-30	6212	5951	0.0000	0.0000	0.0143	-0.0820	0.0693	-0.0075	0.0082
ZT	1999-01-05	2022-12-30	6212	5928	0.0000	0.0000	0.0097	-0.0616	0.0560	-0.0051	0.0053
ZU	1999-01-05	2022-12-30	6212	6001	0.0000	0.0003	0.0254	-0.4337	0.2515	-0.0118	0.0126
ZW	1999-01-05	2022-12-30	6212	5943	-0.0000	0.0000	0.0190	-0.0953	0.0919	-0.0117	0.0104
ZZ	1999-01-05	2022-12-30	6212	5974	0.0000	0.0000	0.0169	-0.0852	0.0834	-0.0093	0.0094

Table 10: Multi-Asset Data Set - Summary Statistics

From the above table 10, one gets fundamental insights about the data set especially with regard to the sample distribution. As can be seen from the overview, all future contracts have the same number of daily returns, while the number of unique values lies within reasonable ranges. In addition, the mean and median show a typical picture in connection with daily return series, whereby these only deviate slightly from zero and the mean tends to be more strongly influenced by positive or negative extreme values. There are no anomalies with regard to the minimum and maximum values, while it should be noted that returns are not below -1 and do not have too strong outlier values upwards. It can therefore be concluded that there are no anomalies within the continuous future contract returns and that there are typical patterns with regard to asset returns. In addition, the cumulative returns of the individual continuous future contracts can be visualized as follows:

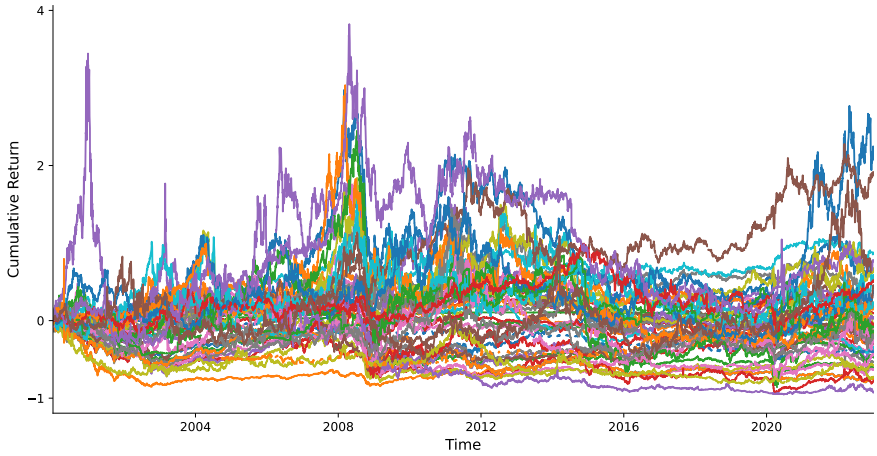


Figure 23: Multi-Asset Data Set - Cumulative Asset Returns

Figure 23 of the cumulative daily continuous future contract returns confirms, on the one hand, that there are no significant anomalies with regard to the daily asset returns. On the other hand, it can be seen that only individual assets have strongly positive cumulative returns over the observation period, which can be equivalently observed on the negative side. With regard to the back-testing, it is therefore important to note that the foundation for the evaluation should not consist exclusively of the individual strategy performances, but rather primarily of the mutual comparison between the benchmark strategies and the proposed strategies.

6 OK Empirical Multi-Asset CSMOM & TSMOM Strategy Back-Testing

In addition to the derivation of the proposed multi-asset CSMOM and TSMOM strategies, the back-testing of those strategies against the previously defined benchmark strategies represents the central empirical part of this thesis. In the first step, the general framework of the model evaluation and the underlying performance measures are described. In the third step, the CSMOM- and TSMOM-specific model features are derived in accordance with existing research work. In the third step, the training, validation and testing process is explained, which is of fundamental importance with regard to deep learning architectures. The last step includes the out-of-sample evaluation of the individual strategies and their comparison using the predefined evaluation framework.

6.1 OK Back-Testing Framework & Performance Measures

In the course of the back-testing, a general framework must be defined in accordance with existing research work, which ensures a standardized procedure for the model evaluation. With reference to the proposed and benchmark multi-asset CSMOM and TSMOM strategies from

the sections 4.2 and 4.4, the following pillars and procedures must be determined within such a framework: The 4.4 *number of assets*, the *evaluation period* and the *performance measures*.

Number of Assets

As can already be seen from section 5, the underlying data set consists of continuous futures contracts, with the scope having already been set to 50 assets. Hence, with reference to the proposed as well as the benchmark strategies, the parameter of the number of assets can be formalized as follows:

$$n_t = n = 50 \tag{138}$$

whereby the number of assets remains constant across the evaluation period.

Performance Measures

In the course of evaluating portfolio strategies, it is important to define the central performance metrics, which are calculated from the out-of-sample realized strategy returns. Here, existing research approaches for momentum strategies as well as for portfolio strategy back-testing in general are used, in accordance with Wood et al. (Wood et al., 2021, p.7) and Poh et al. (Poh et al., 2022, p.5-6). Within these performance metrics, three different performance clusters arise by *profit*-related, *risk*-related and *performance*-related metrics. The following table provides an overview of the corresponding metrics and their description based on the statements by Poh et al. (Poh et al., 2022, p.5-6):

	Metric	Description
Profit	**Annualized Returns**	Average rate of the portfolio return per fiscal year
Risk	**Annualized Volatility**	Average rate of portfolio volatility per fiscal year
	Downside Risk	Dispersion of the portfolio return less than a pre-specified return target (set to zero)
	Max Drawdown	Largest portfolio value percentage drop over a pre-specified time frame, hence illustrating the worst-case portfolio scenario
Performance	**Sharpe Ratio**	Risk-adjusted return in excess of the risk-free return (set to zero) over the portfolio holding period
	Calmar Ratio	Risk-adjusted portfolio performance measure by adjusting the annualized portfolio return with the portfolio's maximum drawdown
	Sortino Ratio	Risk-adjusted portfolio performance measure by adjusting the annualized portfolio return by the risk of negative returns
	% Positive Returns	Percentage of positive portfolio returns compared to all portfolio returns
	Profit-Loss Ratio	Average profit from positive portfolio returns against the average loss from negative portfolio returns

Table 11: Back-Testing Performance Measures

As can be seen from table 11, these metrics holistically cover the various perspectives of performance evaluation and thus allow a direct comparison across all strategies, with the risk-adjusted performance metric in particular as part of the sharpe ratio as the key metric is being used.

6.2 OK Model Features

As can already be seen from the explanations of the proposed multi-asset CSMOM and TSMOM strategies in sections 4.2 and 4.4, there is a central strategy component regarding the choice of input features. As part of CSMOM strategies, both for the *CSMOM Transformer Ranker* and the *CSMOM ListNet Pre-Ranker Transformer Re-Ranker*, those features are used for all 50 continuous future contracts at the time of training, validation and testing. In contrast, this is done separately for individual assets at the time of training, validation and testing the cross-section reference to the predefined look-back period within the class of the TSMOM strategies, both for the *TSMOM Encoder-Decoder Transformer* and the *TSMOM Decoder-Only Transformer*. Overall, the choice of the number of features as well as the respective features themselves can make a significant contribution to the success or failure of deep learning-based models. In the context of this thesis, both the number and the choice of features rely on existing research work in the area of CSMOM and TSMOM strategies.

CSMOM Model Features

The choice of input features for the multi-asset CSMOM models proposed in section 4.2 is based primarily on heuristics as well as on existing empirical research results. The research work by Poh et al. particularly serves as a reference (2022), as well as previous works by the same authors (Poh et al., 2020; Poh et al., 2023) and subsequent works by Tan et al. (2023). It is important to note that, despite the monthly re-balancing period, the input features are constructed from daily asset returns of the continuous future contracts. The state-of-the-art approaches regarding the application of deep learning methods to CSMOM strategies show that there are three central feature groups: *raw cumulative returns* across different time scales, *volatility -normalized returns* across different time scales as well as *MACD indicators* across different pairs of short and long time scales as well as different past times (Poh et al., 2020, p.5). In the context of this work, due to the empirical significance and the purpose of comparability, the feature groups of volatility-normalized as well as MACD-based features are used, whereby the definitions are based on the previously introduced concepts in section 4.2. This means that the following input features can ultimately be defined for the back-testing of the proposed multi-asset CSMOM strategies:

Normalized Returns	**MACD Indicators**
$r^{(i)}_{t-d,t} / \left(\sigma^{(i)}_t \sqrt{d}\right)$, $d \in \{1, 5, 21, 63, 126, 252\}$	$M^{(i)}_{t,t-m}(S,L) \mid (S,L) \in \mathcal{T}$, $\mathcal{T} \in \{(8,24),(16,28),(32,96)\}$ $\Updownarrow \in \{0, 21, 63, 126, 252\}$

Table 12: CSMOM Input Features

where the ex-ante volatility $\sigma^{(i)}_t$ is derived from section 4.2 according to the exponentially weighted moving standard deviation with look-back of 63 days. In addition, the MACD indicators are calculated according to the original definition by Baz et al. (2015), which was also already shown in section 4.2. In addition, not only the MACD as of the time step t is included as a feature, but also previous MACD indicators 21, 63, 126 and 252 days before time t, in line

with Poh et al. (2020, p.5). In total, there are six volatility-normalized return features, which are supplemented by a total of 15 features from the MACD-based group. This ultimately results in a total of 21 input features for the proposed multi-asset CSMOM models.

TSMOM Model Features

With reference to the state-of-the-art empirical results regarding deep learning-enhanced TSMOM strategies by Tan et al. (2023), as well as the underlying research results of Wood et al. (2022) and Lim et al. (2020), it shows that a uniform framework of input features has been established over time. Equivalently to the CSMOM features, the input features are constructed from daily asset returns of the continuous future contracts. This contributes to the fact that, despite a comparatively long portfolio holding phase between the re-balance dates, more recent information is included in the model in-sample training and validation as well as out-of-sample testing and thus enables the capturing of short-term trends within the asset-specific time series. In the context of existing literature, there are two main groups of input features with regard to TSMOM strategies: *volatility-normalized returns* at different time scales and *MACD indicators* at different time scale pairs. With regard to the parametrization of both feature groups, the definition is made with reference to equivalently used features in the benchmark strategies from section 4.4.4 as well as existing TSMOM literature as follows:

Normalized Returns	**MACD Indicators**
$r^{(i)}_{t-d,t} / \left(\sigma^{(i)}_t \sqrt{d}\right)$, $d \in \{1, 5, 21, 63, 126, 252\}$	$M^{(i)}_t (S, L) \,\vert\, (S, L) \in \mathcal{T}$, $\mathcal{T} \in \{(8, 24), (16, 28), (32, 96)\}$

Table 13: TSMOM Input Features

where, in the course of the volatility-normalized returns, the ex-ante volatility $\sigma^{(i)}_t$ is defined as the exponentially weighted moving standard deviation of the daily asset returns with look-back period 63 and d is the look-back period in days is set in the course of calculating the cumulative returns. With regard to the MACD indicators $M^{(i)}_t (S, L)$, the calculation is carried out analogously to the MACD benchmark model from section 4.4.4. This results in a total of six different volatility-normalized and three different MACD indicators, meaning a total of nine TSMOM input features are used.

6.3 OK Model Training, Validation & Testing

The back-testing of the proposed multi-asset CSMOM and TSMOM strategies is carried out using the traditional setup for machine learning-based and, above all, supervised models by training, validation and testing cycles. Supervised refers to the setup in which both training input data and pre-processed output labels are available, whereby the underlying data dynamics and patterns can be learned in-sample based on those input-output pairs (Goodfellow et al., 2016, p.105). The selection of the optimal model for the final out-of-sample prediction follows a clear scheme: In the first step, a set of hyperparameters is selected from all available hyperparameters,

after which the model and the model-specific parameters are trained. In the second step, the model performance is evaluated using the validation data set. Based on all train-validation iterations, the final hyperparameters are chosen based on the best validation performance, after which the model and the underlying parameters are finally optimized based on the train data set. In the last step, the out-of-sample prediction follows from the finally calibrated model based on the out-of-sample inputs from the test data set. It should be noted that the training, validation and testing data sets do not contain any overlapping observations in order to ensure the strategy or model performance for rolling future evaluations. The choice of the train-validation-test split based on the available data is of central importance here, as a balance must be created between over- and underfitting problems in order to learn the in-sample data structures and patterns sufficiently and at the same time to generalize sufficiently to achieve an optimal out-of-sample based on unseen data.

According to existing research in the area of CSMOM and TSMOM strategies, there is a key difference between the two approaches regarding the definition of the training, validation and testing interval, see Poh et al. (2020, p.5), Poh et al. (2022, p.6) and Wood et al. (2022, p.6). In order to ensure comparability with existing benchmarks, this difference is also adopted in this work, whereby in the course of CSMOM strategies a *rolling window* approach (Poh et al., 2022, p.6) and in the course of TSMOM strategies an *expanding window* approach (Wood et al., 2022, p.6) is applies. This implies that CSMOM models are trained and validated at five-year intervals and the following five years serve as the testing interval (Poh et al., 2022, p.6). This happens on a rolling basis, so that the next training-validation interval uses the previous testing interval and the following five years again serve as a testing interval. In contrast, the training-validation interval for TSMOM strategies expands as intervals progress, keeping the training starting point constant across all intervals but expanding the training endpoint by five years per interval (Wood et al., 2022, p.6). Equivalent to CSMOM strategies, the testing interval consists of the following five years and is rolling, which means there is no overlap between training, validation and testing data sets. So it can be summarized that CSMOM strategies have constant train-test splits of $50\%-50\%$ and TSMOM strategies start with train-test splits of $50\%-50\%$, but, across iterations, a stronger weighting on the training samples is developed, with approximately $67\% - 33\%$ in the second iteration, $75\% - 25\%$ in the third iteration, etc. Within the train split itself, both strategies have a train-validation split in the ratio of approximately $80\% - 20\%$, whereby 80% of the training samples are used to train the model parameters and 20% are used to validate the hyperparameters. This is also in line with existing research, see Poh et al. (2020, p.5), Poh et al. (2022, p.6) and Wood et al. (2022, p.6). As a summary of the train-validation-test split just derived, the following figure serves as a simplified graphical overview of the generalized CSMOM and TSMOM framework:

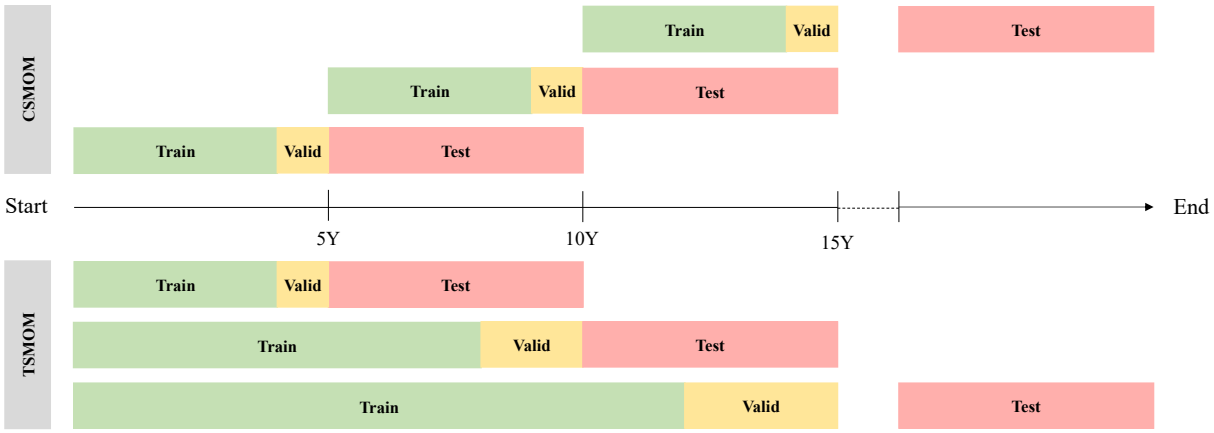


Figure 24: Train-Validation-Test Split - Own Illustration

With reference to the strategy assumptions defined in section 4.2.3 and 4.4.3, the specific train-validation-test split can be derived from the previously introduced train-validation-testing framework. When the framework gets applied to the underlying data set of continuous futures contracts, the following splits result for the proposed transformer-based multi-asset CSMOM models of the *CSMOM Transformer Ranker* and the *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* in the form of the date intervals and corresponding input data shapes:

		CSMOM Transformer Ranker & CSMOM ListNet Pre-Ranker Transformer Re-Ranker			
		Batch #1	**Batch #2**	**Batch #3**	**Batch #4**
Train	**Dates**	01/2000 - 06/2004	12/2004 - 04/2009	10/2009 - 03/2014	09/2014 - 01/2019
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (55, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (55, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (55, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (55, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (55, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (55, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (55, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (55, 50, 1)
Valid	**Dates**	07/2004 - 10/2004	05/2009 - 08/2009	04/2014 - 07/2014	02/2019 - 05/2019
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (4, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (4, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (4, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (4, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (4, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (4, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (4, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (4, 50, 1)
Test	**Dates**	12/2004 - 08/2009	10/2009 - 07/2014	09/2014 - 05/2019	07/2019 - 11/2022
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (59, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (59, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (59, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (59, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (59, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (59, 50, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (42, 50, 21) $\mathbf{y}^{\mathbf{Enc}}$: (42, 50, 1)

Table 14: Model Input - CSMOM Transformer Ranker & ListNet Pre-Ranker Transformer Re-Ranker

From table 14, it can be seen that both the input samples $\mathbf{U}^{\text{Enc}}$ and the output samples $\mathbf{y}^{\text{Enc}}$ have a three-dimensional structure across the train, validation and test splits. This is consistent with the necessary input dimensions from section 4.2 and 4.4, where the dimensionality is as (number of samples, number of assets, number of features), which is the central encoder-only abstraction of the original transformer architecture for numerical applications. Thus, each sample within the train, validation and test samples represents the cross-section of all available assets and their input features at the same time step. Furthermore, it shows that there are no overlaps between

the train, validation and test samples within the respective batches.

In the course of applying the training-validation-testing framework to the proposed transformer-based TSMOM strategies, in addition to the previously explained strategy assumptions of the re-balance frequency of 21 days (one month), the look-back period of 12 months is utilized as well. In addition, compared to the proposed CSMOM strategies, there are different training, validation and testing sample structures due to the different model architectures of the TSMOM Encoder-Decoder Transformer and the TSMOM Decoder-Only Transformer, which is why the TSMOM model inputs as the date intervals and the input shapes can be derived separately for both proposed TSMOM strategies:

		TSMOM Encoder-Decoder Transformer		
		Batch #1	**Batch #2**	**Batch #3**
Train	**Dates**	06/2003 - 04/2007	06/2003 - 03/2011	06/2003 - 02/2015
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (2400, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (2400, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (2400, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (4800, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (4800, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (4800, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (7200, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (7200, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (7200, 1, 1)
Valid	**Dates**	05/2007 - 04/2008	04/2011 - 02/2013	03/2015 - 01/2018
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (600, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (600, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (600, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (1200, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (1200, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (1200, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (1800, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (1800, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (1800, 1, 1)
Test	**Dates**	05/2008 - 02/2013	03/2013 - 01/2018	01/2018 - 11/2022
	Shapes	$\mathbf{U}^{\mathbf{Enc}}$: (3000, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (3000, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (3000, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (3000, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (3000, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (3000, 1, 1)	$\mathbf{U}^{\mathbf{Enc}}$: (3000, 11, 9) $\mathbf{U}^{\mathbf{Dec}}$: (3000, 1, 9) $\mathbf{y}^{\mathbf{Dec}}$: (1800, 1, 1)

Table 15: Model Input - TSMOM Encoder-Decoder Transformer

Table 15 shows the extent to which the fundamental difference exists between cross-sectional and time series momentum frameworks. While in the course of the CSMOM strategies each sample represented all available assets and their features at the same step, it shows that each sample represents the asset-specific time series and the time-specific input features. This also explains the larger number of samples on the first dimension, with the general notation of the dimensionality of the input samples in the form of (number of samples, number of time steps, number of features). In addition, during each sample, only one position size is predicted as the output of the transformer decoder based on the implicit trend estimation over the look-back period. The following reshaping is then carried out after the predictions: First, the outputs of the respective batches are squeezed, resulting in a matrix $\mathbf{y}^{\text{Dec}} \in \mathbb{R}^{\text{ Number of Samples} \times \text{Number of Time Steps}}$. The matrix is then reshaped such that the time steps are arranged in descending order on the first dimension and the assets on the second dimensionality, resulting in a final position size matrix $\mathbf{y}^{\text{Dec}}\ in \mathbb{R}^{\text{Number of Time Steps} \times \text{Number of Assets}}$.

As already explained before, a separate data structure needs to be created for the TSMOM

Decoder-Only Transformer, since the input samples are now only fed into the decoder. This leads to the following model input intervals and shapes:

		TSMOM Decoder-Only Transformer		
		Batch #1	**Batch #2**	**Batch #3**
Train	**Dates**	06/2003 - 04/2007	06/2003 - 03/2011	06/2003 - 02/2015
	Shapes	$\mathbf{U}^{\mathbf{Dec}}$: (200, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (200, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (400, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (400, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (600, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (600, 12, 1)
Valid	**Dates**	05/2007 - 04/2008	04/2011 - 02/2013	03/2015 - 01/2018
	Shapes	$\mathbf{U}^{\mathbf{Dec}}$: (50, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (50, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (100, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (100, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (150, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (150, 12, 1)
Test	**Dates**	05/2008 - 02/2013	03/2013 - 01/2018	01/2018 - 11/2022
	Shapes	$\mathbf{U}^{\mathbf{Dec}}$: (250, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (250, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (250, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (250, 12, 1)	$\mathbf{U}^{\mathbf{Dec}}$: (250, 12, 9) $\mathbf{y}^{\mathbf{Dec}}$: (250, 12, 1)

Table 16: Model Input - TSMOM Decoder-Only Transformer

Table 16 includes the difference between the TSMOM Encoder-Decoder Transformer and the TSMOM Decoder-Only Transformer input structure. It turns out that all time steps within the look-back period only flow into the decoder, which means that not just one, but $12(=\tau)$ position sizes are predicted for each sample. Building on this, the sample size in the first dimension is also reduced by the factor of the look-back period compared to the TSMOM Encoder-Decoder Transformer. Nevertheless, the general dimensionality remains as (number of samples, number of time steps, number of features). As a direct result, there is also a larger delta in the train start. While the samples in the TSMOM Encoder-Decoder Transformer represent successive time steps, there is a delta equal to the look-back period of 12 days in the TSMOM Decoder-Only Transformer, since in each sample a position size is already predicted for each time step. Analogous to the encoder-decoder architecture, the outputs are also reshaped so that they can serve as a sharpe loss input. While the specific steps of reshaping are more comprehensive, the same output as for the TSMOM Encoder-Decoder Transformer also results for the TSMOM Decoder-Only Transformer as $\mathbf{y}^{\text{Dec}} \in \mathbb{R}^{\text{Number of Time Steps} \times \text{Number of Assets}}$.

After the basic framework for the definition of training, validation and testing intervals has been defined and the input data structures for all proposed CSMOM and TSMOM transformer-based models have been derived, the corresponding model (hyper)parameters need to be derived for the benchmark as well as the proposed transformer-based models. In the first step, the parameters defined in the CSMOM and TSMOM strategy assumptions from the sections 4.2.3 and 4.4.3 as the *portfolio size*, the *re-balance frequency*, the *target volatility* and, specifically for TSMOM strategies, the *look-back period* are reconciled. In addition, there are further parameters that need to be defined in the course of back-testing, in the form of fixed parameters and value sets of hyperparameters, which are validated as part of the validation iterations.

As it was already derived in section 4.2.4, benchmark models consist of the *Random* approach,

the *Original CSMOM* approach according to Jegadeesh & Titman (1993) and the *MACD* approaches according to Baz et al. (2015). In these models, all components are defined deterministically, which means that only the look-back periods of the original CSMOM strategies have to be defined as fixed parameters as well as the short and long spans. In line with the original statement by Jegadeesh & Titman (1993) as well as key benchmark research work by Poh et al. (2020), Poh et al. (2022) and Tan et al. (2023), the original CSMOM strategy is tested as three versions with three, six and twelve months and the short and long span periods with $\{8, 16, 12\}$ and $\{24, 48, 96\}$, respectively. On the other hand, there are two machine learning-based models by *Multi-Layer Perceptrons* (MLP) and *ListNet*. Those models obey parameters that need to be fixed as well as model hyperparameters that need to be validated during the train-validation iteration. A similar picture exists in the course of the benchmark models for TSMOM strategies according to section 4.4.4. On the one hand, there are benchmark models by the *Random* approach as a baseline benchmark and the *Original TSMOM* approach according to Moskowitz et al. (2012). In this context, it is only necessary to define the look-back period for the original TSMOM strategy, which is in line with existing research work by Woods et al. (2021) and Tan et al. (2023) as well as the original setup by Moskowitz et al. (2012) and set to 12 months. On the other hand, machine learning-based models also exist within the framework of TSMOM in the form of the *Recurrent Neural Network* (RNN) and *Long Short-Term Memory Neural Network* (LSTM), whose fixed parameters are to be set and the model hyperparameters are to be validated as part of the train validation iterations. The following table shows both the choice of fixed parameters and the predefined value sets of the corresponding hyperparameters:

		CSMOM		TSMOM	
		MLP	**ListNet**	**RNN**	**LSTM**
Fixed Parameters	**Portfolio Size**	10	10	50	50
	Re-balance Frequency	21	21	21	21
	Target Volatility	15%	15%	15%	15%
	Δ Train Start	1	1	12	12
	Train Interval	5 (rolling)	5 (rolling)	5, 10, 15 (expanding)	5, 10, 15 (expanding)
	Train-Validation	80%-20%	80%-20%	80%-20%	80%-20%
	Test Interval	5	5	5	5
	# Epochs	100	100	100	100
	# Search Trials	200	200	200	200
	Early Stopping	25	25	25	25
	Loss	MSE	Listwise	Sharpe	Sharpe
	Optimizer	Adam	Adam	Adam	Adam
	# Hidden Layers	2	2	1	1
Hyperparameters	**Batch Size**	100, 200, 400, 800	100, 200, 400, 800	8, 16, 32, 64, 128	8, 16, 32, 64, 128
	Hidden Width	16, 32, 64, 128, 256, 512, 1024	16, 32, 64, 128, 256, 512, 1024	-	-
	Activation Function	ReLU, Sigmoid, Linear, Tanh	ReLU	Tanh	Tanh
	Units	-	-	8, 16, 32, 64, 128, 256	8, 16, 32, 64, 128, 256
	Dropout Rate	0.1, 0.2, 0.3, 0.4, 0.5	0.1, 0.2, 0.3, 0.4, 0.5	0.1, 0.2, 0.3, 0.4, 0.5	-
	Learning Rate	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5

Table 17: Fixed & Hyperparameters - Benchmark CSMOM & TSMOM Models

While the parameters *portfolio size*, *re-balance frequency*, *target volatility*, *delta train start*, *train interval*, *train-validation* split, *test interval* and the *loss* function have already been explained previously, there are further (hyper) parameters that need to be derived. In addition, the *number of epochs* refers to the number of training-validation iterations over the corresponding underlying sample set (Goodfellow et al., 2016, p.246), whereupon the model parameters of the model-internal weights are learned over those iterations according to learning algorithms such as backpropagation or backpropagation through time. The number of epochs of 100 is based on research standards and benchmark research work (Wood et al., 2021, p.20). Furthermore, as part of the hyperparamter tuning, the *random search method* is used, whereby the combination

of the hyperparameter values is chosen randomly, and this is repeated according to the *number of search trials* (Goodfellow et al., 2016, p.434-435). The choice of 200 search trials is in line with other CSMOM and TSMOM research, see Poh et al. (2020, p.5) and Wood et al. (2021, p.20). In order to avoid overfitting during training-validation iterations, *early stopping* is applied such that training and thus the adjustment of the model-internal parameters are stopped as soon as there is no improvement in the validation performance for a number of consecutive epochs (Goodfellow et al., 2016, p.246-247). The early stopping rule is set to 25 epochs, which also improves the computational performance with regard to the relatively high choice of search trials. In the context to stochastic gradient descent methods, the Adam *optimizer* is chosen for all deep learning-based models, which allows for computational efficiency, little memory requirement, and invariance to diagonal rescaling of gradients, while being well suited for problems that are large in terms of data/parameters (Kingma and Ba, 2017, p.1). The last deterministically chosen parameter in both MLPs and the ListNet is set to two *hidden layers* since that configuration has already been empirically validated, see Poh et al. (2020, p.5) and Lim et al. (2020). After the fixed parameters have been explained, the hyperparameter sets need to be derived. Here, the *batch size* represents the number of samples according to which the model-internal parameters are updated based on the corresponding gradients (Goodfellow et al., 2016, p.152). It is necessary to define a set of batch sizes according to the condition $batchsize \in [1, Numberof Samples]$, while the concrete size is considered as a payoff instrument between computational speed and accuracy, whereby a batch size that is too large leads to a reduced generalization of the data structures and patterns, but is accompanied by a more efficient run-time (Keskar et al., 2017, p.2-3). In contrast, the opposite dynamic applies to small batch sizes (Keskar et al., 2017, p.2-3). Another hyperparameter is given the *hidden width* in the context of multi-layer perceptrons and ListNet. The choice of values within that hyperparameter set is based on existing research work and its empirical validity, see Lim et al. (2020, p.17) and Poh et al. (2022, p.10). This is further accompanied by the choice of the set for the *activation function*, whereby the MLP receives a set of possible activation functions according to (16), while ListNet model obeys the ReLU function according to the original architecture by Cao et al. (2007) and the vanilla RNN as well as the LSTM are constructed under tanh functions. In addition, it is important to define the number of *units* specifically for RNNs and LSTMs. Here as well, the hyperparameter sets are determined based on existing research work, see Lim et al. (2020, p.19) Wood et al. (2021, p.20). As an additional tool for overfitting prevention, additional dropout layers are added to the vanilla model architectures (Goodfellow et al., 2016, p.258-259). Based on research standards, *dropout rates* between 0.1 and 0.5 are used. The last hyperparameter is the *learning rate*, which controls the payoff between the efficiency and the achievement of a local or global minimum of the cost function during the gradient descent optimization, as already elaborated in section 3.1.1. A learning rate that is too large can lead to an oscillating learning structure and no convergence to a minimum, while a learning rate that is too small can tend to be local instead of achieving a global optimum. Since the learning rate has already been evaluated in many research papers in general and also with reference to momentum strategies, the hyperparameter sets are defined for all models using existing benchmarks, see Lim et al. (2020, p.19) or Wood et al. (2021, p.20).

After the fixed parameters and the hyperparameter grids have been derived in the course of the

CSMOM and TSMOM benchmark models, this needs to be done equivalently for the proposed transformer-based models. The strategy and model assumptions from the sections 4.2.3 and 4.4.3 are used and expanded to include the necessary (hyper)parameters. The following table contains the entire overview of all fixed parameters as well as the hyperparameter value sets:

		CSMOM		**TSMOM**	
		Transformer Ranker	**ListNet PR Transformer RR**	**Encoder-Decoder Transformer**	**Decoder-Only Transformer**
Fixed Parameters	**Portfolio Size**	10	10	50	50
	Re-balance Frequency	21	21	21	21
	look-back Window	-	-	12	12
	Target Volatility	15%	15%	15%	15%
	Δ Train Start	1	1	1	12
	Train Interval	5 (rolling)	5 (rolling)	5, 10, 15 (expanding)	5, 10, 15 (expanding)
	Train-Validation	80%-20%	80%-20%	80%-20%	80%-20%
	Test Interval	5	5	5	5
	# Epochs	100	100	100	100
	# Search Trials	200	200	200	200
	Early Stopping	25	25	25	25
	Optimizer	Adam	Adam	Adam	Adam
	Loss	Listwise	Listwise	Sharpe	Sharpe
Hyperparameters	**Batch Size**	4, 8, 16, 32	4, 8, 16, 32	100, 200, 400, 800	8, 16, 32, 64, 128
	$\mathbf{d}_{model}$	8, 16, 32, 64, 128, 256	8, 16, 32, 64, 128, 256	8, 16, 32, 64, 128, 256	8, 16, 32, 64, 128, 256
	# Encoder Layers	1, 2, 3	1, 2, 3	1, 2, 3	-
	# Decoder Layers	-	-	1, 2, 3	1, 2, 3
	# Heads	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4
	$\mathbf{d}_{ff}$	8, 16, 32, 64, 128	8, 16, 32, 64, 128	8, 16, 32, 64, 128	8, 16, 32, 64, 128
	Dropout Rate	0.1, 0.2, 0.3, 0.4, 0.5	0.1, 0.2, 0.3, 0.4, 0.5	0.1, 0.2, 0.3,0.4, 0.5	0.1, 0.2, 0.3, 0.4, 0.5
	Learning Rate	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5	1e-2, 1e-3, 1e-4, 1e-5

Table 18: Fixed & Hyperparameters - Proposed CSMOM & TSMOM Models

As can be seen from table 18, there are significant overlaps with the (hyper)parameters of the

benchmark models from table 17. For this reason, only the new parameters will be discussed below, while the previous section provides derivations of the overlapping parameters. With reference to the proposed model architectures, the hyperparameter d_{model} represents the size of the embedding vector, which, after the initial application of the embedding layer, defines the shape of the corresponding input vectors of the actual transformer model. Since there are only a few applications of transformer networks for numerical applications and in particular momentum strategies to date, a relatively large interval is chosen for the choice of value sets. In addition, in the core of the encoder and decoder blocks, the number of *encoder layers* and *decoder layers* must be defined as a hyperparameter set. Due to the relatively small number of samples for both the CSMOM and TSMOM data sets, only one, two and four layers are selected. In addition, the number of attention *heads* represents another transformer-specific hyperparameter as part of the multi-head self-attention, whereby, according to the theoretical model architecture from section 3.2, the embedding size d_{model} and the number of heads must be evenly divisible. The corresponding parameter set of the number of heads relates to existing research work, see Poh et al. (2022, p.10) and Wood et al. (2021, p.20). The last new hyperparameter in comparison to the benchmark models is given by d_{ff}, which represents the dimensionality of the hidden layer of the feed-forward neural networks as components of the transformer network according to section 3.2. Equivalent to the embedding size d_{model}, a broad parameter set is used in order to carry out the validation using a truly data-driven approach.

6.4 Model Back-Testing & Evaluation

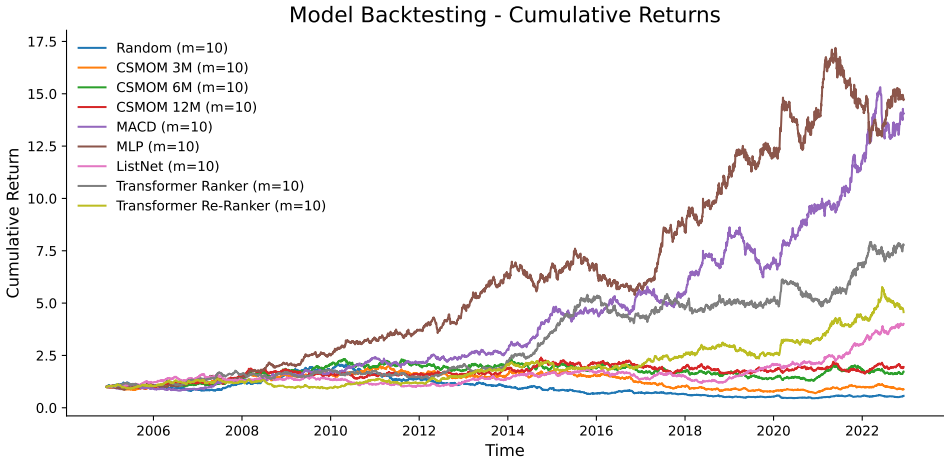
The empirical model back-testing of the proposed *CSMOM Transformer Ranker* and *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* as well as the *TSMOM Encoder-Decoder Transformer* and *TSMOM Decoder-Only Transformer* compared to the corresponding benchmark strategies is carried out using the back-testing framework derived in the previous sections. During the performance evaluation, the central focus is on risk-adjusted performance metrics in accordance with existing empirical strategy evaluations in the course of CSMOM and TSMOM strategies. In addition, an in-depth look at the risk metrics is also carried out in order to evaluate the volatility-related strategy risk. Nevertheless, all previously defined performance metrics of each benchmark and proposed strategy are reported and taken into account. The following back-testing results are achieved for the CSMOM benchmark and proposed strategies:

	Benchmark CSMOM					**Proposed CSMOM**	
	Random (m=10)	**Original CSMOM** (m=10)	**MACD** (m=10)	**MLP** (m=10)	**ListNet** (m=10)	**Transformer Ranker** (m=10)	**ListNet PR Transformer RR** (m=10)
Ann. Return	-0.0301	3M: -0.0068 6M: 0.0298 12M: 0.0367	0.1536	**0.1565***	0.0779	0.1173	0.0856
Ann. Volatility	0.1556	3M: 0.1531 6M: **0.1529*** 12M: **0.1529***	0.1548	0.1534	0.1531	0.1534	0.1549
Downside Risk	0.1137	3M: 0.1084 6M: 0.1073 12M: 0.1084	**0.1019***	0.1030	0.1057	0.1039	0.1070
Max Drawdown	0.7873	3M: 0.6550 6M: 0.4519 12M: 0.3278	0.2779	0.2917	0.3854	**0.2526***	0.3472
Sharpe Ratio	-0.1184	3M: 0.0319 6M: 0.2687 12M: 0.3121	1.0007	**1.0248***	0.5666	0.7997	0.6076
Calmar Ratio	-0.0382	3M: -0.0104 6M: 0.0660 12M: 0.1119	**0.5527***	0.5363	0.2021	0.4645	0.2465
Sortino Ratio	-0.1621	3M: 0.0451 6M: 0.3828 12M: 0.4400	1.5200	**1.5264***	0.8209	1.1815	0.8795
% Positive Returns	0.4996	3M: 0.4899 6M: 0.5019 12M: 0.5088	0.5176	**0.5262***	0.5107	0.5118	0.5157
Profit-Loss Ratio	0.9645	3M: 1.0162 6M: 1.0080 12M: 0.9899	**1.0953***	1.0529	1.0324	1.0717	1.0285

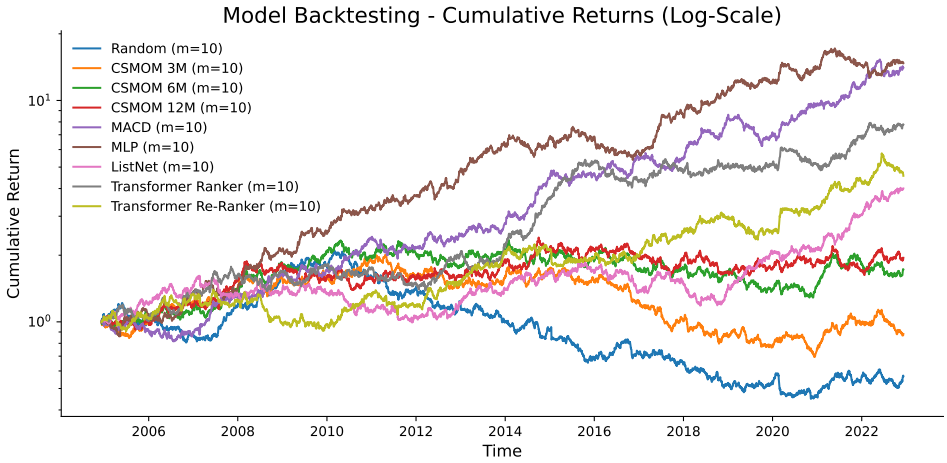
Table 19: Model Back-Testing - CSMOM Model Evaluation

As can be seen from table 19, the sharpe ratio as the central performance metric shows that the MLP model achieves the best performance with a sharpe ratio of 1.0248. It should also be noted that the MACD-based strategy, as a deterministic benchmark strategy, achieves the second-highest sharpe ratio of 1.007. The results of the proposed models of the CSMOM Transformer Ranker and the CSMOM ListNet Pre-Ranker Transformer Re-Ranker, on the other hand, lie significantly behind the MLP and the MACD-based strategy with sharpe ratios of 0.7997 and 0.6076, respectively, with the other benchmark models being significantly out-performed. With regard to the annualized volatility as the key risk measure, all strategies are close to each other, which is mainly due to the additional volatility scaling layer on the portfolio level in accordance with Poh et. al (2023, p.7) in order to ensure performance comparability, which in turn means that the MLP also obeys the largest annualized return when considering the sharpe ratio as a risk-adjusted return measure. With regard to the remaining evaluation metrics, there is no opposite picture to the results already derived, with both the MLP model and the MACD-based strategy dominating across all metrics and the proposed CSMOM models out-performing the remaining benchmark strategies. Beyond the metrics-based model evaluation, the following figures show the cumulative (log) returns of all strategies over the out-of-sample evaluation

period:



(a) Model Back-Testing - Cumulative Returns



(b) Model Back-Testing - Cumulative Log-Returns

Figure 25: Model Back-Testing - CSMOM Strategy Cumulative (Log) Returns - Own Illustration

Table 25 shows an equivalent picture of the evaluation metrics. Both the MLP model and MACD-based strategy have a significant out-performance compared to the proposed CSMOM strategies and the other benchmark strategies. In addition, there is a significant out-performance of the proposed CSMOM strategies compared to the other benchmarks. It should also be emphasized that the Covid-19 crash and the subsequent market regime have a particularly strong impact on the performance of the MLP model, while the MACD-based strategy has the most robust performance during this time and the other strategies are only slightly affected.

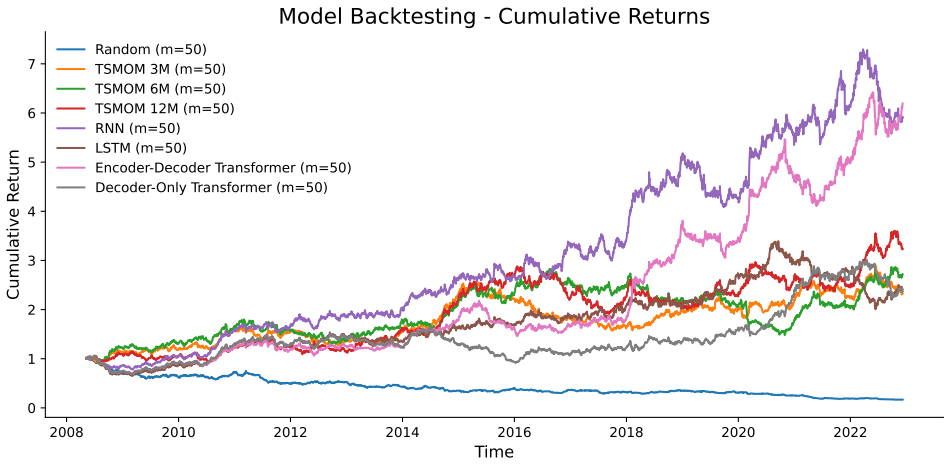
Before an in-depth discussion of possible causes of the under-performance of the proposed CSMOM strategies takes place, the back-testing of the TSMOM must be equivalently evaluated. The following table shows the previously defined evaluation metrics of all proposed and benchmark TSMOM strategies:

	Benchmark TSMOM				Proposed TSMOM	
	Random (m=50)	**Original TSMOM** (m=50)	**RNN** (m=50)	**LSTM** (m=50)	**Encoder-Decoder Transformer** (m=50)	**Decoder-Only Transformer** (m=50)
Ann. Return	-0.1118	3M: 0.0579 6M: 0.0688 12M: 0.0812	0.1258	0.0605	**0.1293***	0.0592
Ann. Volatility	0.1534	3M: 0.1546 6M: 0.1544 12M: 0.1547	0.1546	**0.1532***	0.1547	0.1546
Downside Risk	0.1141	3M: 0.1073 6M: 0.1083 12M: 0.1087	0.1053	0.1079	**0.1043***	0.1097
Max Drawdown	0.8436	3M: 0.3851 6M: 0.4764 12M: 0.3460	**0.2454***	0.4089	0.3405	0.4458
Sharpe Ratio	-0.6962	3M: 0.4416 6M: 0.5083 12M: 0.5824	0.8440	0.4599	**0.8633***	0.4497
Calmar Ratio	-0.1325	3M: 0.1504 6M: 0.1444 12M: 0.2347	**0.5126***	0.1479	0.3796	0.1329
Sortino Ratio	-0.9360	3M: 0.6363 6M: 0.7247 12M: 0.8285	1.2384	0.6527	**1.2797***	0.6335
% Positive Returns	0.4868	3M: 0.5190 6M: 0.5206 12M: 0.5196	0.5238	0.5193	**0.5304***	0.5183
Profit-Loss Ratio	0.9402	3M: 0.9979 6M: 1.0023 12M: 1.0199	**1.0470***	0.9992	1.0241	1.0016

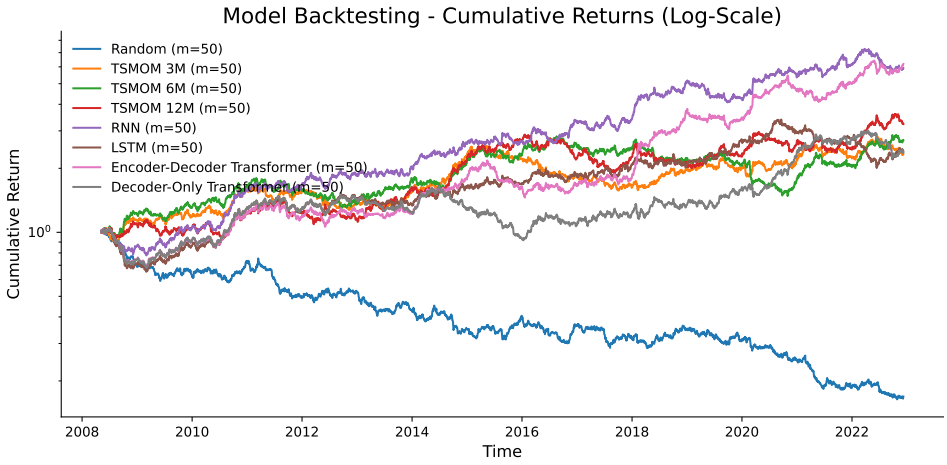
Table 20: Model Back-Testing - TSMOM Model Evaluation

As can be seen from the table 20, especially with regard to the central performance metric of the sharpe ratio, the proposed TSMOM encoder-decoder strategy delivers the best performance, with a sharpe ratio of 0.8633. However, this is closely followed by the RNN benchmark model with a sharpe ratio of 0.8440. The proposed TSMOM decoder-only strategy has a sharpe ratio of 0.4497, which is only marginally underperformed compared to the LSTM benchmark model with 0.4599, but especially compared to the original TSMOM strategies with a six- and twelfth-month look-back window and sharpe ratios of 0.5083 and 0.5524, respectively, significantly underperformed. With regard to the annualized volatility as a central risk measure, a similar picture as during the CSMOM back-testing emerges, with all strategies having very similar volatilities due to the additional volatility scaling on the portfolio level in accordance with Poh et. al (2023, p.7) in order to ensure performance comparability, and therefore the TSMOM Encoder-Decoder Transformer also has the highest annualized return. This result and, above all, the significant out-performance of the proposed encoder-decoder transformer compared to the decoder-only transformer underlines, under the back-testing framework used here, the central argument of Zhou et al. (2021) that transformer neural networks were inherently designed as encoder-decoder architectures and should also be applied to numerical time series use cases in

that setup (Wood et al., 2021, p.17). In addition to the evaluation metrics, the following figures show the cumulative (log) returns of all TSMOM strategies:



(a) Model Back-Testing - Cumulative Returns



(b) Model Back-Testing - Cumulative Log-Returns

Figure 26: Model Back-Testing - TSMOM Strategy Cumulative (Log) Returns - Own Illustration

As shown in figure 26, the cumulative (log-)returns underline the previously elaborated back-testing results. It can be seen that the RNN benchmark model and the TSMOM Encoder-Decoder Transformer have a clear out-performance compared to the other strategies, whereas the TSMOM Decoder-Only Transformer has a similar performance compared to the remaining strategies. However, it should also be noted that the TSMOM Encoder-Decoder Transformer only shows significant out-performance from 2018.

Based on the previously explained results of the CSMOM and TSMOM back-testing, the question arises as to how the performance of the proposed deep learning-enhanced multi-asset momentum strategies should be evaluated. In this context, it is worth taking a look at previously applied transformer models on momentum strategies, in particular the research findings by Poh et al. (2022) and Wood et al. (2021). While the empirical back-testing of those research works was partly applied to other data universes, and especially not to multi-asset portfolios, as well

as to significantly lower rebalance frequencies, they show different results. Compared to similar benchmark strategies, deep learning and especially transformer-based strategies achieve significantly more significant out-performances against benchmarks (Poh et al., 2022, p.7; Wood et al., 2021, p.7). A likely explanation lies in problems surrounding the *data scarcity*, which is associated with monthly versus daily re-balance frequencies and is explained and examined in more detail in the following section.

Data scarcity is a well-known problem in deep learning and therefore highly parameterized models that utilize deep neural networks. It describes the often-existing situation where there is too little training data available to train neural networks (Bansal et al., 2022, p.9 -10), while there is a central condition for the training of neural networks given by a sufficiently large amount of available data (Bansal et al., 2022, p.9-10). This condition goes hand in hand with the size of the models, with neural networks often having a significantly larger number of model-internal parameters compared to other machine learning models, which in turn requires a higher amount of underlying training data. In addition, this is particularly relevant for transformer neural networks, which, due to their many different components and the possibility of further stacking several encoder and decoder blocks, have significantly more model-internal parameters than comparable models such as MLPs, RNNs or LSTMs. With regard to the data scarcity issue, problems arise primarily with regard to the generalizability of the model. If it has been trained on a limited training data set, it will be unable to accurately generalize the underlying data dynamics and patterns in order to perform well on a new set of data, such as the testing set, which leads to an overfitting bias (Bansal et al., 2022, p.9-10). Looking at the model inputs in the tables 14, 15 and 16, it is evident that the monthly re-balance frequency and the testing intervals of five years lead to comparatively little available training data, whereby a daily re-balance frequency alone would increase the training set by almost a factor of twenty. In addition, the training scheme of all proposed and benchmark strategies shows that the in-sample performance of the sharpe-optimizing models is significantly better with sharpe ratios between 1.5 and 3.0, while the out-of-sample performance according to tables 19 and 20 is significantly below 1.0. According to Alzubaidi et al. (2023, p.13-14), such a discrepancy between in-sample and out-of-sample performance, which is available directly from the loss output, represents a clear indicator of too few training samples and implies a significant overfitting bias.

While the focus of this thesis is explicitly on the empirical testing of the proposed CSMOM and TSMOM for multi-asset portfolio construction with relatively high rebalance frequencies of one month, a small asset universe of only 50 continuous futures contracts and the momentum-specific adaptation of vanilla deep learning architectures, which ensures industry-relevant findings, there are several possible approaches for overcoming issues related to data scarcity. One of the state-of-the-art approaches is given by *transfer learning*, by focusing on developing an effective model for a target task with scarce training samples by transferring knowledge across different but related data source domains (Poh et al., 2023, p.2). A concrete example with regard to momentum strategies is the training of a source deep learning model based on assets with similar properties, whereupon the pre-trained model shares the adjusted model parameters with the target model, which is then re-trained on the target data set (Alzubaidi et al., 2023, p.15). There are already isolated research results from Poh et al. (2023), whereby a transfer ranking approach, which

represents a specific application of transfer learning for LTR use cases, has already been applied to CSMOM strategies with weekly re-balance frequencies for cryptocurrencies and the source model was trained on asset returns of real currency pairs. The empirical results show that the transfer ranking approach achieves significant improvements in terms of common evaluation metrics and overcomes the issues of data scarcity.

7 Conclusion & Outlook

The central purpose of this thesis is given by two sequential and inter-dependent sub-goals. On the one hand, deep-learning enhanced multi-asset CSMOM and TSMOM strategies are derived and constructed based on the fundamental vanilla architectures of existing deep learning models, which are adapted from the ground up according to generalized CSMOM and TSMOM strategy frameworks. On the other hand, these proposed CSMOM and TSMOM models are empirically back-tested on a multi-asset data set against state-of-the-art benchmark strategies. The examination of existing vanilla deep learning architectures shows that transformer neural networks, which were initially used primarily in the area of text- and language-based use cases, also achieve empirically significant results with regard to numerical use cases, in particular with respect to learning-to-rank applications (Pobrotyn et al., 2021) as the foundation for CSMOM approaches as well as quantitative portfolio strategies and time series forecasting (Kisiel and Gorse, 2022) as the foundation for TSMOM approaches. As part of the derivation of deep learning-enhanced multi-asset CSMOM and TSMOM strategies, this thesis proposes the *CSMOM Transformer Ranker* as well as the *CSMOM ListNet Pre-Ranker Transformer Re-Ranker* for optimized cross-sectional asset ranking as well as the *TSMOM Encoder-Decoder Transformer* and the *TSMOM Decoder-Only Transformer* for simultaneous trend estimation and position sizing, in accordance with the generalized strategy frameworks and adapted from vanilla deep learning architectures. With respect to the vanilla transformer neural network according to Vaswani et al. (2017), it can be seen that the core components of the encoder and decoder blocks, especially with regard to the (masked) multi-head self-attention, are also applicable and beneficial for numerical use cases, while input and output-related adjustments are needed with regard to the general CSMOM and TSMOM strategy frameworks. Key model enhancements are given by encoder-only applications for asset ranking prediction with listwise loss functions for CSMOM and either encoder-decoder or decoder-only applications with direct position size outputs through tanh layers with sharpe loss function for direct performance metric optimization of TSMOM strategies.

After the proposed deep learning-enhanced multi-asset CSMOM and TSMOM strategies have been derived from the ground up, they are empirically tested against baseline benchmark strategies in the form of the random model, deterministic benchmark strategies in the form of the original CSMOM and TSMOM strategies according to Jegadeesh & Titman (1993) and Moskowitz et al. (2012), respectively, as well as empirically applied deep learning-based strategies in the form of multi-layer perceptrons, ListNet, RNNs and LSTMs. The back-testing of the proposed CSMOM and TSMOM strategies against the predefined benchmark models is carried out using a multi-asset data set of 50 future contracts, which are transformed into continuous price series, with equity, fixed income, foreign exchange and commodity instruments over a period from 1999 to 2022 . A 5-year rolling window approach is used for CSMOM strategies, whereby the models are trained on an interval of 5 years and validated with regard to the hyperparameters under an in-sample training-validation split of 80%-20%, and tested out-of-sample on the following interval of 5 years. As part of the TSMOM strategies, a 5-year expanding window approach is applied, whereby, compared to the CSMOM training-validation split, the training interval is increasing by 5 years with each interval under the equivalent in-sample 80%-20%

training-validation split. Hence, the training start timestamp is fixed, while the end timestamp is increasing, and the out-of-sample testing is still applied to a rolling 5-year window approach. For comparability and inference purposes, this train-validation-test procedure is in accordance with existing research (Poh et al., 2022; Wood et al., 2021).

Based on the predefined back-testing framework, the following empirical results emerge: In the course of the CSMOM strategies, the MLP model and the deterministic MACD-based strategy lead to the best performances, especially with regard to the sharpe ratios of 1.0248 and 1.0007, respectively. Furthermore, with regard to the *CSMOM Transformer Ranker* and the *CSMOM ListNet Pre-Ranker Transformer Re-Ranker*, it can be seen that both strategies with sharpe ratios of 0.7997 and 0.6076 lag behind, but still significantly out-perform against the other benchmark strategies. In the course of back-testing the TSMOM strategies, it is shown that the proposed *TSMOM Encoder-Decoder Transformer* out-performs all other strategies with a sharpe ratio of 0.8633, whereby the *TSMOM Decoder-Only Transformer* with a sharpe ratio of 0.6076 does not achieve any significant out-performance compared to the majority of benchmark strategies. The latter underlines the central argument of Zhou et al. (2021) that transformer neural networks were inherently designed as encoder-decoder architectures and should also be applied to numerical time series use cases in that setup (Wood et al., 2021, p.17). In comparison with existing transformer-based CSMOM and TSMOM strategies by Poh et al. (2022) and Wood et al. (2021), however, it is evident that those transformer-based strategies achieve a more significant out-performance compared to equivalent benchmark strategies, especially under shorter rebalance frequencies of one day. This is likely related to data scarcity issues, whereby deep learning models, and especially transformer networks with a comparatively high number of model-internal parameters, require sufficiently large training data sets in order to ensure generalizability of the underlying data dynamics and patterns in the course of training develop and thus create a balance between in- and out-of-sample performance. However, it turns out that the in-sample performance, which is directly observable through the sharp loss functions during training, is significantly better than the final out-of-sample performance, which results in overfitting and a lack of generalizability due to corresponding data scarcity.

Based on the empirical back-testing results, it is recommended to create specific model architectures for monthly re-balanced deep learning-enhanced multi-asset CSMOM and TSMOM with a focus on the handling of data scarcity issues. While there are already isolated research findings with empirically significant results, including those by Poh et al. (2023) on transfer ranking approaches on CSMOM, these need to be validated in the course of multi-asset portfolios and monthly rebalance frequencies or further developed in a strategy and data-specific manner as well as with regard to time series momentum applications.

List of References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. http://download.tensorflow.org/paper/whitepaper2015.pdf

Alsmadi, M. k., Omar, K. B., Noah, S. A., & Almarashdah, I. (2009). Performance comparison of multi-layer perceptron (back propagation, delta rule and perceptron) algorithms in neural networks. *2009 IEEE International Advance Computing Conference*, 296–299.

Alzubaidi, L., Bai, J., Al-Sabaawi, A., Santamaría, J., Albahri, A., Al-dabbagh, B., Fadhel, M., Manoufali, M., Zhang, J., Al-Timemy, A., Duan, Y., Abdullah, A., Farhan, L., Lu, Y., Gupta, A., Albu, F., Abbosh, A., & Gu, Y. (2023). A survey on deep learning tools dealing with data scarcity: Definitions, challenges, solutions, tips, and applications. *Journal of Big Data*, *10*. https://doi.org/10.1186/s40537-023-00727-2

Bansal, M. A., Sharma, D. R., & Kathuria, D. M. (2022). A systematic review on data scarcity problem in deep learning: Solution and applications. *54*(10s). https://doi.org/10.1145/3502287

Barberis, N., Shleifer, A., & Vishny, R. (1998). A model of investor sentiment1. *Journal of Financial Economics*, *49*(3), 307–343. https://www.sciencedirect.com/science/article/pii/S0304405X98000270

Baz, J., Granger, N., Harvey, C., Roux, N., & Rattray, S. (2015). Dissecting investment strategies in the cross section and time series. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2695101

Bird, R., Gao, X., & Yeung, D. (2017). Time-series and cross-sectional momentum strategies under alternative implementation strategies. *Australian Journal of Management*, *42*(2), 230–251. https://www.uts.edu.au/sites/default/files/FDG_Seminar_150408.pdf

Bondt, W. F. M. D., & Thaler, R. (1985). Does the stock market overreact? *The Journal of Finance*, *40*(3), 793–805. http://www.jstor.org/stable/2327804

Campbell, H. R., Hoyle, E., Korgaonkar, R., Rattray, S., Sargaison, M., & van Hemert, O. (2018). The impact of volatility targeting. https://doi.org/http://dx.doi.org/10.2139/ssrn.3175538

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. *Proceedings of the 24th International Conference on Machine Learning*, *227*, 129–136. https://doi.org/10.1145/1273496.1273513

Corp, P. D. (2023). *Continuously linked commodity contracts*. Retrieved October 31, 2023, from https://pinnacledata2.com/clc.html

Daniel, K., Hirshleifer, D., & Subrahmanyam, A. (1998). Investor psychology and security market under- and overreactions. *The Journal of Finance*, *53*(6), 1839–1885. https://onlinelibrary.wiley.com/doi/abs/10.1111/0022-1082.00077

De Bondt, W. F. M., & Thaler, R. H. (1987). Further evidence on investor overreaction and stock market seasonality. *The Journal of Finance*, *42*(3), 557–581. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1987.tb04569.x

de Long, J. B., Shleifer, A., Summers, L. H., & Waldmann, R. J. (1990). Positive feedback investment strategies and destabilizing rational speculation. *The Journal of Finance*, *45*(2), 379–395. http://www.jstor.org/stable/2328662

Ernst, O. K. (2014). Stochastic gradient descent learning and the backpropagation algorithm.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, *25*(2), 383–417. http://www.jstor.org/stable/2325486

Fama, E. F. (1998). Market efficiency, long-term returns, and behavioral finance. *Journal of Financial Economics*, *49*(3), 283–306. https://www.sciencedirect.com/science/article/pii/S0304405X98000269

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, *12*, 2451–2471.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org

Grosse, R. (2019). Lecture 5: Multilayer perceptrons. *inf. téc.* https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/readings/L05%20Multilayer%20Perceptrons.pdf

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., & Graves, A. (2016). Memory-efficient backpropagation through time. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/a501bebf79d570651ff601788ea9d16d-Paper.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80.

Hong, H., & Stein, J. C. (1999). A unified theory of underreaction, momentum trading, and overreaction in asset markets. *The Journal of Finance*, *54*(6), 2143–2184. https://onlinelibrary.wiley.com/doi/abs/10.1111/0022-1082.00184

IBM. (2023). What are recurrent neural networks? https://www.ibm.com/topics/recurrent-neural-networks

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91. http://www.jstor.org/stable/2328882

Johnson, T. C. (2002). Rational momentum effects. *The Journal of Finance*, *57*(2), 585–608. https://onlinelibrary.wiley.com/doi/abs/10.1111/1540-6261.00435

Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., & Brubaker, M. (2019). Time2vec: Learning a vector representation of time.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima.

Kim, A. Y., Tse, Y., & Wald, J. K. (2016). Time series momentum and volatility scaling. *Journal of Financial Markets*, *30*(100), 103–124. https://ideas.repec.org/a/eee/finmar/v30y2016icp103-124.html

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization.

Kisiel, D., & Gorse, D. (2022). Portfolio transformer for attention-based asset allocation.

Lakonishok, J., Shleifer, A., & Vishny, R. W. (1994). Contrarian investment, extrapolation, and risk. *The Journal of Finance*, *49*(5), 1541–1578. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1994.tb04772.x

Lewellen, J. (2002). Momentum and autocorrelation in stock returns. *The Review of Fionancial Studies*, *15*(2), 533–564. https://academic.oup.com/rfs/article-abstract/15/2/533/1588891?redirectedFrom=fulltext

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2020). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting.

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2020). Temporal fusion transformers for interpretable multi-horizon time series forecasting.

Lim, B., Zohren, S., & Roberts, S. (2020). Enhancing time series momentum strategies using deep neural networks.

Liu, L. X., & Zhang, L. (2008). Momentum profits, factor pricing, and macroeconomic risk. *The Review of Financial Studies*, *21*(6), 2417–2448. https://academic.oup.com/rfs/article-abstract/21/6/2417/1575021?redirectedFrom=fulltext

Lu, Y., & Lu, J. (2020). A universal approximation theorem of deep neural networks for expressing distributions. *CoRR*, *abs/2004.08867*. https://arxiv.org/abs/2004.08867

Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, *17*(1), 59–82. https://www.aeaweb.org/articles?id=10.1257/089533003321164958

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.

Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2012). Time series momentum [Special Issue on Investor Sentiment]. *Journal of Financial Economics*, *104*(2), 228–250. https://doi.org/https://doi.org/10.1016/j.jfineco.2011.11.003

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. https://arxiv.org/pdf/1211.5063.pdf

Pobrotyn, P., Bartczak, T., Synowiec, M., Białobrzeski, R., & Bojar, J. (2021). Context-aware learning to rank with self-attention.

Poh, D., Lim, B., Zohren, S., & Roberts, S. (2020). Building cross-sectional systematic strategies by learning to rank.

Poh, D., Lim, B., Zohren, S., & Roberts, S. (2022). Enhancing cross-sectional currency strategies by context-aware learning to rank with self-attention.

Poh, D., Roberts, S., & Zohren, S. (2023). Transfer ranking in finance: Applications to cross-sectional momentum with data scarcity.

Sazli, M. (2006). A brief review of feed-forward neural networks. *Communications Faculty Of Science University of Ankara*, *50*, 11–17.

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *CoRR*, *abs/1912.05911*. http://arxiv.org/abs/1912.05911

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, *04*, 310–316.

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. https://arxiv.org/abs/1803.09820

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, *abs/1909.09586*. http://arxiv.org/abs/1909.09586

Svozil, D., Kvasnicka, V., & Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, *39*(1), 43–62. https://www.sciencedirect.com/science/article/pii/S0169743997000610

Tan, W. L., Roberts, S., & Zohren, S. (2023). Spatio-temporal momentum: Jointly learning time-series and cross-sectional strategies.

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. *Artif. Intell. Rev.*, *53*(8), 5929–5955. https://doi.org/10.1007/s10462-020-09838-1

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. https://arxiv.org/abs/1706.03762

Vojtko, R., & Padysak, M. (2020). Continuous futures contracts methodology for backtesting. https://doi.org/https://dx.doi.org/10.2139/ssrn.3517736

Wood, K., Giegerich, S., Roberts, S., & Zohren, S. (2022). Trading with the momentum transformer: An intelligent and interpretable architecture.

Wood, K., Roberts, S., & Zohren, S. (2021). Slow momentum with fast reversion: A trading strategy using deep learning and changepoint detection. *The Journal of Financial Data Science*, *4*(1), 111–129. https://doi.org/10.3905/jfds.2021.1.081

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting.

Declaration of Authorship

I hereby declare,

- that I have written this thesis independently,
- that I have written the thesis using only the aids specified in the index;
- that all parts of the thesis produced with the help of aids have been precisely declared;
- that I have mentioned all sources used and cited them correctly according to established academic citation rules;
- that I have acquired all immaterial rights to any materials I may have used, such as images or graphics, or that these materials were created by me;
- that the topic, the thesis or parts of it have not already been the object of any work or examination of another course, unless this has been expressly agreed with the faculty member in advance and is stated as such in the thesis;
- that I am aware of the legal provisions regarding the publication and dissemination of parts or the entire thesis and that I comply with them accordingly;
- that I am aware that my thesis can be electronically checked for plagiarism and for third-party authorship of human or technical origin and that I hereby grant the University of St.Gallen the copyright according to the Examination Regulations as far as it is necessary for the administrative actions;
- that I am aware that the University will prosecute a violation of this Declaration of Authorship and that disciplinary as well as criminal consequences may result, which may lead to expulsion from the University or to the withdrawal of my title."

By submitting this thesis, I confirm through my conclusive action that I am submitting the Declaration of Authorship, that I have read and understood it, and that it is true.

Date and signature

November 20th 2023, ...

By submitting this academic term paper, I confirm through my conclusive action that I am submitting the Declaration of Authorship, that I have read and understood it, and that it is true.