



Degree Project in Optimization and Systems Theory

Second Cycle, 30 credits

Attention-Based Time Series Forecasting in Foreign Exchange Trading

Intraday Market Spread Prediction with Self-Attention Networks

VIKTOR GJUROVSKI

Author

Viktor Gjurovski

KTH Royal Institute of Technology

External Project Partner

Skandinaviska Enskilda Banken

Examiner

Xiaoming Hu

Numerik, Optimering & System

KTH Royal Institute of Technology

Supervisors

Simon Österberg, Skandinaviska Enskilda Banken

Xiaoming Hu, KTH Royal Institute of Technology

Abstract

Financial market prediction is a critical area of research with significant implications for market participants such as traders and market makers. In this thesis, we investigate the effectiveness of attention-based time-series forecasting models in the context of foreign exchange (FX) trading. The forecasting of a multi-step spread prediction is done by introducing a model that draws its inspiration from, and utilizes the self-attention mechanism in the seminal work "Attention is All You Need".

This model is trained and tested using mean squared error (MSE) on one week of intraday market data in the EUR/SEK currency pair and evaluated using root mean square error (RMSE), mean absolute error (MAE), and confusion matrices. Comparative analysis is conducted against traditional models including recurrent neural networks (RNN) and long short-term memory (LSTM) architectures, as well as two simplistic models.

The results demonstrate that our attention-based model outperforms all baseline models, providing some evidence of the effectiveness of attention mechanisms in capturing complex temporal dependencies in financial data. However, the model faces challenges in accurately predicting periods of no movement, which are crucial for market makers. Future work is primarily focused on addressing this shortfall through various proposed enhancements.

In conclusion, the attention-based model shows promise in enhancing predictive performance in FX trading. By addressing identified limitations and implementing proposed enhancements, the model has the potential to become a valuable tool for market participants, facilitating informed decision-making and risk management strategies for market makers.

Keywords

Foreign Exchange Trading, Financial Time Series, Forecasting, Market Making, Spread, Liquidity, Self-Attention, Transformer, Orderbook Data.

Sammanfattning

Finansiell marknadsprognos är ett kritiskt forskningsområde med betydande implikationer för marknadsdeltagare så som handlare och marknadsbildare. I denna uppsatts undersöker vi effektiviteten hos självuppmärksamhetsbaserade tidsserieprognosmodeller inom ramen för valutahandel (FX). Prognosering av skillnad mellan köp och säljpris i flera steg görs genom att introducera en modell som hämtar sin inspiration från och använder sig av självuppmärksamhetsmekanismen i rapporten "Attention is All You Need".

Modellen är tränad och testad med medelvärdet av kvadratfelet på prediktionen (MSE) på en veckas intradagsdata i valutaparet EUR/SEK, och evalueras med roten ur kvadratfelet på prediktionen (RMSE), medelvärdet på absoluta felet (MAE), samt förvirringsmatriser. En jämförande analys mot mer traditionella modeller i sekvensmodellering, rekursiva nätverk (RNN) och långt-och kortsiktigt minne (LSTM), samt två enklare modeller görs.

Resultaten visar att vår framtagna modell överträffar alla jämförelsemodeller och framför viss evidens för effektiviteten hos uppmärksamhetsmodellen i att fånga komplexa tidsberoenden i finansiell data. Dock står modellen inför utmaningar gällande att förutsäga perioder utan rörelse, vilket är kritiskt för en marknadsbildare. Framtida arbete adresserar primärt denna brist i modellen med flertalet förslag på förbättring.

Sammanfattningsvis så visar den uppmärksamhetsbaserade modellen lovande resultat i sin predektiva presetanda inom FX handel. Genom att adressera och identifiera begränsningarna, samt implementera de förslagna förbättringarna så har modellen potential att bli ett värdefullt verktyg för marknadsdeltagare. Modellen har potential att underlätta beslutsfattande och riskhanteringsstrategier för en marknadsbildare.

Nyckelord

Valutahandel, finansiella tidsserier, prognos, marknadsskapande, spread, likviditet, Self-Attention, transformer, orderboksdata.

Acknowledgements

I would like to thank my employer, SEB for granting me the opportunity to write this thesis. I am thankful to my supervisors Simon Österberg and Xiaoming Hu for their expertise, insights and constructive feedback which have been instrumental in shaping this thesis. Last but certainly not least, I would like to extend my heartfelt appreciation to my family for their encouragement, understanding and support.

Stockholm, June 2024

Viktor Gjurovski

Contents

1	Introduction	1
1.1	Background and Problem Formulation	1
1.2	Purpose	3
1.3	Aim & Research Questions	3
1.4	Research Methodology	3
1.5	Delimitations	4
1.6	Structure of thesis	4
2	Background	7
2.1	Foreign Exchange Spot Trading	7
2.1.1	Market Structure	7
2.1.2	Inventory Risk	9
2.1.3	Price Construction	10
2.2	Sequence Models	13
2.2.1	Introduction	13
2.2.2	Encoder-Decoder	14
2.2.3	Attention Model	14
2.3	The Transformer Neural Network	15
2.3.1	Scaled Dot-Product Attention	16
2.3.2	Multi-headed Attention	17
2.3.3	The Transformer Architecture	18
2.4	Related Work	19
2.4.1	FX Spot Trading	19
2.4.2	Sequence Models for Time Series	20
3	Methodology	23
3.1	Data Collection	23
3.1.1	Targeted Dataset	23
3.1.2	Sampling	23
3.1.3	Features Construction	24
3.1.4	Target Variables	27
3.1.5	Data Normalization	27
3.2	Model Design	28

3.2.1	Test Environment	28
3.2.2	Baseline models	28
3.2.3	The Transformer Implementation	29
3.2.4	Validation & Reliability	29
4	Results	31
4.1	Hyperparameter Outcome	31
4.2	Prediction results	31
5	Discussion	35
5.1	Results	35
5.1.1	Predictive power of the model	35
5.1.2	Applicability of the proposed model in market making	36
5.2	The Architecture	37
5.3	Improvements	37
5.4	Conclusions	38
	Bibliography	41
A	Dataset Features	45
A.1	Feature Visualisations	45
A.2	Full Featureset	46
B	Results	47
B.1	Hyperparameters	47
B.2	Prediction variable distribution	48
B.3	Prediction Error Distribution	48
B.3.1	Linear y-axis	48
B.3.2	Logarithmic y-axis	50
B.4	Confusion Matrices	51

List of Figures

2.1	Cumulative volume of example orderbook in table 2.2 as a function of price. VWAP Price of the cumulative amount is seen as dashed lines, with markers at even 200k steps up until 800k.	11
2.2	Comparison between RNN, GRU and LSTM cells. Source: dprogrammer	12
2.3	The Self-Attention mechanism in the transformer neural network. Source: [1]	15
2.4	Schematic view of the transformer network. Source: [1]	16
3.1	Reference bid and ask prices constructed for level volumes of (2, 4, 6, 8, 10) million Euros over a 1 minute frame with $\tau = 100ms$	24
3.2	Corresponding mid and spread representation of figure 3.1.	25
3.3	Liquidity changes over each time bucket and level, for the prices in figure A.1.	26
3.4	Liquidity withdrawals and additions over each time bucket and volume level, for the prices in figure 3.3. The levels are constructed using even 2M EUR steps.	26
3.5	Number of orderbook updates by side for the prices in figure 3.1	27
3.6	Flowchart of baseline prediction models based on the keras.layers API.	28
3.7	Time Series Cross Validation Schematic.	29
3.8	Flowchart of the Convolution Transformer (CT) prediction model based on the Keras.layers API.	30
A.1	Reference bid, ask, mid, spread prices zoomed in from figure 3.1 and 3.2.	45
B.1	The distribution of the prediction variable returns y_t^l for each bin $l \in (1, 2, 4, 6, 8)$	48
B.2	The CT model error distribution compared to a simple one step martingale. The y-axis is a log scale.	48
B.3	The CT model error distribution compared to the RNN model.	49
B.4	The CT model error distribution compared to the LSTM model.	49
B.5	The CT model error distribution compared to a simple one step martingale. The y-axis is a log scale.	50
B.6	The CT model error distribution compared to the RNN model. The y-axis is a log scale.	50

B.7 The CT model error distribution compared to the LSTM model. The y-axis is a log scale..	51
---	----

List of Tables

2.1	Example of an limit order book updates from 3 different liquidity providers.	8
2.2	Consolidated Order Book for data in table 2.1	9
2.3	Reference prices for LP i receiving example prices according to table 2.2. .	12
2.4	VWAP Prices for dotted lines in figure 2.1	12
3.1	Liquidity Features per side and volume bucket.	25
4.1	RMSE for all tested folds for the baseline models RNN, LSTM, the proposed Convolutional Transformer (CT) as well as the two simplistic predictions models $M(1)$, $M(\emptyset)$	32
4.2	MAE for all tested folds for the baseline models RNN, LSTM, the proposed Convolutional Transformer (CT) as well as the two simplistic predictions models $M(1)$, $M(\emptyset)$	32
4.3	Comparison of RNN, LSTM, and the Convolutional Transformer (CT) models using RMSE and MAE metrics, where the model was trained on fold 1-4, and consequently tested on fold 2-5 respectively, for each volume bin.	33
A.1	Complete list of all features. Bin features are flattened in ascending order to create the feature vector of size $d = 76$ at each time step.	46
B.1	Hyperparams for the baseline models found by Keras Tuner.	47
B.2	Hyperparams for the CT model found by Keras Tuner.	47
B.3	CT Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.	51
B.4	RNN Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.	52
B.5	LSTM Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.	52

Abbreviations

CLOB	Central Limit Order Book
CNN	Convolutional Neural Network
CT	Convolutional Transformer
CV	Cross Validation
ECN	Electronic Communication Network
FX	Foreign Exchange
GRU	Gated Recurrent Unit
LP	Liquidity Provider
LLM	Large Language Model
LSTM	Long-Short term Memory
MAE	Mean Absolute Error
MHA	Multi-Head Attention
OHLC	Open-High-Low-Close
OTC	Over The Counter
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
TNN	Transformer Neural Network

Chapter 1

Introduction

1.1 Background and Problem Formulation

This thesis proposal targets the foreign exchange (FX) spot market, more specifically the price construction in an FX spot market, and the implication of short-time liquidity changes on such a price construction. The beneficiary and host of this thesis is the Scandinavian bank SEB, which acts as a market maker in the FX market.

Market making in foreign exchange trading is a crucial function within financial markets that ensures liquidity, efficiency, and stability. Market makers are entities—usually banks or financial institutions—that continuously quote both the buy (bid) and sell (ask) prices for a currency pair. By doing so, they facilitate smoother trading operations for other market participants, including retail traders, institutional investors, and corporations engaging in international trade.

The primary role of a market maker is to provide liquidity. This means they are always ready to buy or sell a currency pair at publicly quoted prices, ensuring that there is always a counterparty available for a trade. This function is vital in the FX market, the largest and most liquid financial market in the world, with daily trading volumes exceeding \$6 trillion. Without market makers, traders might struggle to execute orders quickly, especially for less commonly traded currency pairs or during periods of market stress.

In addition to providing liquidity, market makers also contribute to price discovery. By continuously updating their quotes based on supply and demand dynamics, they help reflect the current market sentiment and underlying economic conditions. This constant pricing adjustment ensures that currency prices remain fair and competitive, enabling more efficient and transparent trading.

Market makers earn their profit from the spread—the difference between the bid and ask prices. For example, if a market maker quotes a bid (buy) price of 1.1000 and an ask (sell) price of 1.1002 for the EUR/USD pair, the spread is the difference between the bid-ask, in

this case 2 pips. Consider the simplest of scenarios where a trader buys €1M at 1.1002 from the market maker, and another trader at the same time sells the same amount (€1M) at 1.1000 to the market maker, the market maker pockets the 2-pip difference. This spread compensates market makers for the risk they take on, including price fluctuations and holding inventory. When this situation occurs, i.e., when buy and sell orders net each other out, the situation is referred to as an *internalization*.

However, market making is not without risks. Market makers must manage the risk of holding large positions in currencies, which can be subject to significant price movements. Market makers employ sophisticated algorithms and high-frequency trading (HFT) systems to manage their operations. These technologies allow them to update prices in real time and manage their risk exposure effectively. One part of the equation is to analyze vast amounts of market data to anticipate trends and adjust their strategies accordingly. This technological edge is essential in the fast-paced FX market, where conditions can change rapidly due to economic news, geopolitical events, or sudden shifts in market sentiment.

As opposed to an equity exchange market which in the context could be considered "centralized" to a small amount of exchange venues with National Best Bid and Offers (NBBO), the FX market is highly fragmented. There exist somewhat centralized exchanges called electronic communication network (ECN); however, a large amount of the flow is never visible on these ECNs [2] [3], and is instead executed over-the-counter (OTC) directly between counterparties. One implication of this is that the price discovery of a currency pair is not as straightforward as looking at a stock exchange where there is almost complete transparency concerning price and liquidity. Given the nature of the FX market, contracts must often be established directly between participants. There exists no regulation directive such as ESMA's MiFID II, instead, contracts for streams of pricing are established based on initiatives such as the [FX Global Code](#).

A participant in the market who electronically receives prices from other participants will more or less have to use some technology to aggregate incoming prices. The price aggregation model that this report will use is simply referred to as "the aggregator", where all definitions are taken from [4]. In short, an aggregator is a technology that consolidates liquidity in the form of bid and ask prices and amounts, from multiple sources into a consolidated order book to create a view of the security the market participant wishes to trade. Given this consolidated orderbook, the market participant has an approximate view of what liquidity is available in the market at the current time, however the participant does not know anything about the near future. Consider the case where a market maker has a risk position that it wants to get rid of; the participant can either hold on to the risk and provide prices in a way that allows for internalisation, or hedge it externally in the market. In both cases, having a view of where the market is heading is crucial. As mentioned and exemplified above, the *spread* is a central parameter to understand and handle for a market maker. Predicting this parameter in a near, intraday horizon, utilizing price- and liquidity data is the purpose of this report.

1.2 Purpose

This project will aim to use intraday price- and liquidity features to predict market spread in a near time horizon. The benefits of the project host, SEB, are many. SEB acts as a market maker, especially in the realm of Scandinavian currencies where they make a substantial part of the market. This means that SEB acts as a liquidity provider which puts a lot of effort on managing risk since they are quoting prices to the market around the clock, all year around. Understanding and successfully predicting the market spread of an security in the market allows SEB to manage their risk better. An increased quality of managing risk consequently means that they can manage their pricing better, which would positively impact the market as a whole.

While recent developments in sequence models for natural language processing [1] have had extraordinary breakthroughs in the field, the application of these models on financial time series data is rather limited even though some very promising models have been proposed as in [5]. By exploring these new sequence models with attention mechanisms on intraday market data, we hope to contribute to the understanding of their architecture, and if the attention mechanism can be utilized to learn important (sequential) contextual properties on time series data that will increase the predictive power over the more common neural network architectures with recurrent neurons.

1.3 Aim & Research Questions

Given the background 1.1 and purpose 1.2, this thesis aims to answer the following research question:

1. Using a consolidated orderbook view, how well can we predict a multi-step market spread using liquidity changes as features, for some discrete time horizon T using a transformer-like neural network architecture that utilizes the self-attention mechanism as in [1]?
2. How does a transformer-like architecture perform compared to recurrent neural network architectures using vanilla RNN and LSTM cells in that multi-step spread prediction?

1.4 Research Methodology

The thesis aims at performing a multi-step regression prediction, no inference or actual trading will take place in real life. The initial dataset construction will be performed on SEB's market data from ECN's on a `kdb+` infrastructure which uses the coding language `Q`. `kdb+` stores its data in a column-major format and is highly optimized on handling large amounts of time sorted data. After this pre-process of data, python will be used with the `TensorFlow` package via the `Keras` API for the prediction. The prediction will be evaluated using various metrics on the precision and accuracy of the model, which will be evaluated against base models such as; a one-step martingale, RNN and LSTM neural networks. The implementation of the transformer neural network will be as closely

replicated to the publication in [1], where the main changes applied are those that make the network work for time series features instead of word embeddings.

1.5 Delimitations

There will be some limitations on the market data, namely:

- The only market data available will be limit orderbooks from a preselected list of ECN's available at SEB, liquidity providers that have not agreed to their prices being used for price discovery are not included.
- No other types of prices will be available, e.g. price banded orderbooks as this would introduce some complexity that would not help the purpose of the thesis, further these price mechanisms are not as common as limit orderbooks.
- No information on market orders or trades are available for the model, only the orderbook dynamics are used.

For creating and tuning the prediction model, there will be some limitations in hyperparameter tuning and size, due to the lack of time and infrastructure. The aim of the thesis is not to deliver a production-ready signal generator, but rather to explore the field of sequence models and see if they could be utilized in the future, so no exhaustive parameter search will be done. The transformer implementation together with the feature engineering will be limited in exploration. An as close replication of the architecture, suitable for orderbook data will be implemented.

1.6 Structure of thesis

The thesis is outlined as following:

- The background theory for the thesis is presented in Chapter 2 is divided into two main sections: FX Spot Trading 2.1 and Sequence Models 2.2. It first presents the FX spot market and how it functions with regard to how price and liquidity discovery can be established from the view of a market maker. Here work from [4] and [6] will be used as a base. In the chapter Sequence Models 2.2 we will briefly introduce neural networks suitable for sequence modelling tasks, then show how [1] is implemented and describe its differences from other sequence models such as [7]. Finally, related work is presented in 2.4.
- As the transformer network is the main character in this report, it has its own background section 2.3.3.
- The Methodology Chapter includes the data collection and processing, this is presented in Section 3.1. Further in this chapter the model implementation as well as validation methods are found in Section 3.2, where the transformer implementation is seen in Section 3.2.3.

- Results are found Chapter 4 which presents the outcome of the hyperparameter tuning, then the prediction performance using MSE and MAE as metrics.
- Finally, the discussion of the results, together with shortfalls, improvements and conclusions of the prediction model are found in Chapter 5.
- Appendix A provides a list with a description of all input features used for the prediction, as well as some visualizing aids of some of the features.
- Appendix B contains prediction- and error distributions, to aid the discussion of the results.

Chapter 2

Background

2.1 Foreign Exchange Spot Trading

When trading FX spot, participants buy and sell currencies at the current market rate. FX prices are always expressed as their value in another currency. A spot price of 10 EUR/SEK implies that 1 EUR is exchanged for 10 SEK. A sell order's price in a market is called a *ask*, and respectively, the price of an order to buy a currency pair is called a *bid*. If not specifically mentioned, consider all orders to be so-called *limit orders*; a limit order is an order to buy/sell a quantity, with a restriction (limit) on the maximum/minimum price respectively.

The FX market is traded over the counter (OTC) which means that there is no central exchange where an instrument is listed, instead the instrument is traded via a broker-dealer network. The market participant can hence connect directly to other participants, or indirectly via an Electronic Communication Network (ECN) which connects multiple participants to one platform and in a sense creates a centralized market. Given this, the FX market can be considered fragmented which imposes challenges on the price discovery of an asset. This section will introduce the structure and relevant definitions of the FX spot market for the thesis.

2.1.1 Market Structure

The initial definitions and notations in this sub-section are based on the paper *Execution in an aggregator* by Oomen, Roel [4]. Consider an unobserved true logarithmic price process of an asset that follows a random walk: $p_t^* = p_{t-1}^* + \epsilon_t$ where $\epsilon \sim \text{i.i.d. } \mathcal{N}(0, \sigma^2)$. We denote p_t^* as the *mid price* of an asset.

For a participant acting as a market maker, there is a finite amount of competing liquidity providers (LP) that offer liquidity to the market, as well as to that participant. Each LP quotes a bid price (*b*) and ask price (*a*), together with an amount at which they are willing to trade. The *spread* is defined as $s = a - b$. The bid- and ask prices the LP's can

LP	Bid Price	Bid Quantity	Ask Price	Ask Quantity
LP1	10.45	100,000	10.46	120,000
	10.43	80,000	10.47	90,000
	10.42	120,000	10.48	150,000
LP2	10.45	120,000	10.48	100,000
	10.43	200,000	10.49	130,000
LP3	10.45	150,000	10.50	80,000
	10.44	150,000	10.51	130,000

Table 2.1: Example of an limit order book updates from 3 different liquidity providers.

be modeled as:

$$b_t^i = p_t^i - \frac{s_t^i}{2}, \quad a_t^i = p_t^i + \frac{s_t^i}{2} \quad (2.1.1)$$

where the price for each LP may be modeled as:

$$p_t^i = p_t^* + m_t^i \quad (2.1.2)$$

where p_t^* can be viewed as the the *best* estimate of the mid price p_t for LP i . The term m_t^i allows for interpretation and will be covered more in detail in Section 2.1.3. For now, only consider it to be a price change of LP- i 's quoted mid price, but for more details see Chapter 2 in [4]. Each LP does not necessarily need to quote only one bid and one offer. An LP could e.g. quote 1M EUR/USD, at their best bid, then another 1M EUR/USD at a lower price, and so on. How many bids and asks that are quoted from one LP is denoted as a *level*. We proceed to hence extend the notation in equation 2.1.1 and 2.1.2 with a level index as following:

$$b_t^{(i,l)} = p_t^{(i,l)} - \frac{s_t^{(i,l)}}{2}, \quad a_t^{(i,l)} = p_t^{(i,l)} + \frac{s_t^{(i,l)}}{2} \quad (2.1.3)$$

$$p_t^{(i,l)} = p_t^* + m_t^{(i,l)} \quad (2.1.4)$$

The aggregator's view of incoming prices could look something like in table 2.1. The aggregator can then consolidate all orders and sort them according to price, this consolidation is denoted as a *central limit order book* (CLOB). An example of this is seen in table 2.2. At this point, it should be noted that if the market participant wishes to trade at these prices, the execution takes place at the LP that sent out the price, and depending on the circumstances a request to deal at a quoted price may in some cases lead to a rejection. This rejection might lead to extra costs for the trading entity dealing in the aggregator. In the case where a trader wishes to execute an order in the aggregator, for a small amount it can just execute at the best price and the severity of getting a reject is relatively small. However, in the case of larger amounts, there might not be sufficient liquidity on the best price, so the trader must face a choice of either aggressing throughout the stack of prices (potentially hitting multiple liquidity providers, and hence increasing the risk of getting a reject), or perform a so-called *full amount* trade with only one liquidity provider.

Bid Price	Ask Price	Bid Quantity	Ask Quantity	Liquidity Provider
10.42	-	120,000	-	LP1
10.43	-	80,000	-	LP1
10.43	-	200,000	-	LP2
10.44	-	150,000	-	LP3
10.45	-	100,000	-	LP1
10.45	-	120,000	-	LP2
10.45	-	150,000	-	LP3
-	10.46	-	120,000	LP3
-	10.47	-	90,000	LP1
-	10.48	-	150,000	LP1
-	10.48	-	100,000	LP2
-	10.49	-	130,000	LP2
-	10.50	-	80,000	LP3
-	10.51	-	130,000	LP3

Table 2.2: Consolidated Order Book for data in table 2.1

The case presented above considers the situation when one aggregator has a direct connection to the liquidity providers. In the case of the aggregator also receiving market data from an ECN, the model becomes more complex. Recall from the introduction that an ECN itself aggregates multiple liquidity providers into one consolidated orderbook and in a sense acts as a centralized market. Assume that at least one of the LPs in the example is connected to the same ECN as the aggregator, and posts the same liquidity at that ECN - the available liquidity as seen from the aggregator is now overlapping over multiple sources.

2.1.2 Inventory Risk

In the previous subsection (2.1) we described the structure of streamed prices in the FX spot market and how they may be interpreted from an aggregator's point of view. In this subsection, we will consider the case where the holder of the aggregator itself also is a liquidity provider.

As described in the introduction, the FX spot market primarily consists of bilateral agreements between counterparts participating in the same market. Using another liquidity provider's price for price discovery, i.e. for determining the parameters in equation 2.1.4 and the spread in equation 2.1.3, is something that is not always allowed, in the scope of this thesis we assume all liquidity providers prices are allowed to be used for price discovery.

Recall the parameter $m_t^{(i,l)}$ in equation 2.1.4 and $s_t^{(i,l)}$ in equation 2.1.3, i.e. the skew parameter for mid price $p_t^{(i,l)}$ and the spread. To understand how these parameters are used we need to first establish some basic understanding of risk from the perspective of the liquidity provider.

When trades arrive at the liquidity provider's prices, their inventory (or *position*) in the dealt security will increase/decrease; e.g. if a trade of buy 1M EUR/SEK at 10.0 in the liquidity pool is dealt, the liquidity provider's EUR position will increase by 1M, whereas the SEK position will decrease by 10M. Assuming that the LPs inventory of all currencies was 0 before this trade, the LP would then be *long* in EUR and *short* in SEK. When this happens the LP has two ways of managing the risk of these two EUR and SEK positions: the LP can either *externalize* or *internalize* its position. Externalise means that the liquidity provider goes to the market and *hedges* its positions. Internalization is the opposite; the LP instead warehouses the risk until an offsetting trade arrives which reduces the risk. The two methods of dealing with inventory risk have their pros and cons. When externalizing risk, the cost is imminent as described in subsection 2.1.1. Internalisation on the other hand will result in longer market risk, i.e., exposure to the price movements on the market. As discussed in [6] the time the dealer is exposed depends heavily on what currency pair is dealt with, as well as what counterparties are connected to the LP's prices. Recall that an externalisation for smaller amounts could be executed immediately however, for larger amounts the trader might not want to execute the amount directly but might want to run an execution algorithm as described in [2].

Whether the trader should internalise or externalise its risk depends not only on the potential cost of the risk of holding vs the cost of executing, but the trader could also take into account the price movement to try and make a profit out of its long or short positions; in this situation, intently holding a short position implies that the trader has a belief that the security's price will decrease, and vice versa.

2.1.3 Price Construction

Now, let's turn our focus toward how an aggregator may construct the mid, $p_t^{(i,l)}$ in equation 2.1.4 and skew, $s_t^{(i,l)}$ in equation 2.1.3, we begin by some slight change in notation; let \tilde{p}_t^l be the *reference mid*, and \tilde{s}_t^l be the *reference spread* for the aggregator at level l . These reference price can be viewed as the cost of externalising at the current time t and are created with standard amounts based on received bids and asks from the list of external liquidity providers. Given these definitions, we approach equation 2.1.3 and 2.1.4 as following:

$$p_t^l = \tilde{p}_t^l + m_t^l \quad (2.1.5a)$$

$$s_t^l = \tilde{s}_t^l + n_t^l \quad (2.1.5b)$$

where m_t^l is the aggregators internal skew for each level l , and n_t^l is a spread corrector. The skew can be seen as a factor that keeps the spread constant, but moves the bid and ask prices up or down. The spread correcting term, n_t^l widens/tightens the spread and is the research subject of this report, for now assume them to be equal to 0, and let's turn our focus towards how one may calculate \tilde{p}_t^l and \tilde{s}_t^l .

Given the example in table 2.2, with even incremental orderbook levels of 200k, the external price at time t can be visualised in figure 2.1 where the *volume weighted average price* (VWAP) of the orderbook is included in the graph, i.e., average price of execution the order on the y-axis. There are many ways one could construct price levels given the

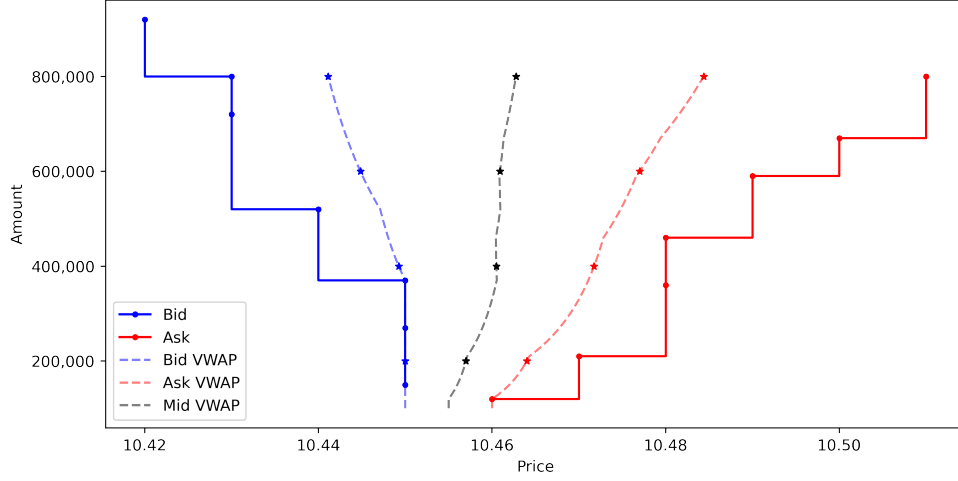


Figure 2.1: Cumulative volume of example orderbook in table 2.2 as a function of price. VWAP Price of the cumulative amount is seen as dashed lines, with markers at even 200k steps up until 800k.

increment, we do so by letting b_t^l, a_t^l be the prices at level l , and consequently by letting $\bar{b}_t(v(l)) = \bar{b}_t^l, \bar{a}_t(v(l)) = \bar{a}_t^l$ denote the average bid and ask prices. Then let P_b and P_a be permutations that sort bid and ask prices accordingly, i.e.:

$$P_b(\{b_t^{(i,l)}\}) = \text{sorted}_{desc}(\{b_t^{(i,l)}\}),$$

$$P_a(\{a_t^{(i,l)}\}) = \text{sorted}_{asc}(\{a_t^{(i,l)}\}).$$

Now, we omit the time index t , and let $b' = P_b(\{b_t^{(i,l)}\}), a' = P_a(\{a_t^{(i,l)}\})$, further, we let $\beta = P_b(\{\beta_t^{(i,l)}\}), \alpha = P_a(\{\alpha_t^{(i,l)}\})$ be the corresponding sorted volumes. The corresponding cumulative sums are denoted as S^β, S^α . Let the incremental steps be denoted as the array v , the prices may then be constructed as follows:

$$b^l = \max_i(b'_i | S_i^\beta \leq v_l) \quad (2.1.6a)$$

$$a^l = \min_i(a'_i | S_i^\alpha \leq v_l) \quad (2.1.6b)$$

$$\bar{b}^l = \frac{1}{\|\beta\|} \sum_i^J b'_i \beta_i, \quad J = \arg \max_j (S_j^\beta \leq v_l) \quad (2.1.6c)$$

$$\bar{a}^l = \frac{1}{\|\alpha\|} \sum_i^J a'_i \alpha_i, \quad J = \arg \max_j (S_j^\alpha \leq v_l) \quad (2.1.6d)$$

$$\tilde{p}^l = \frac{1}{2}(b^l + a^l) \quad (2.1.6e)$$

$$\tilde{s}_t^l = a^l - b^l \quad (2.1.6f)$$

l	b_t^l	a_t^l	\tilde{p}_t^l	\tilde{s}_t^l
1	10.450	10.464	10.457	0.014
2	10.448	10.479	10.464	0.031
3	10.436	10.488	10.462	0.051
4	10.430	10.507	10.468	0.077

Table 2.3: Reference prices for LP i receiving example prices according to table 2.2.

$v(l)$	$\bar{b}_t(v)$	$\bar{a}_t(v)$	$\bar{p}_t(v)$	$\bar{s}_t(v)$ (bps)
200k	10.450	10.464	10.457	140
400k	10.449	10.472	10.460	225
600k	10.445	10.477	10.461	322
800k	10.441	10.484	10.463	433

Table 2.4: VWAP Prices for dotted lines in figure 2.1

The resulting price levels given equation 2.1.6 are seen in table 2.3, and the vwap prices in table 2.4. Given the parameters \tilde{p}_t^l and \tilde{s}_t^l , what remains to complete equation 2.1.5 is to determine m_t^l and n_t^l . Given the current time t and some finite horizon of interest h , the purpose of the thesis is to predict, at time $t + h$ the change of spread, n_{t+h}^l . E.g. assume the prediction says that there will be a big liquidity withdrawal at the ask side, but no change at the bid side; this would imply that the mid skew $m_{t+h}^1 > 0$, and the spread would widen, i.e. $n_{t+h}^1 > 0$. In the case where the predicted liquidity withdrawal of the ask is the same as the predicted liquidity addition to the bid side, $m_{t+h}^1 > 0$, and the spread would remain the same, i.e. $n_{t+h}^1 = 0$.

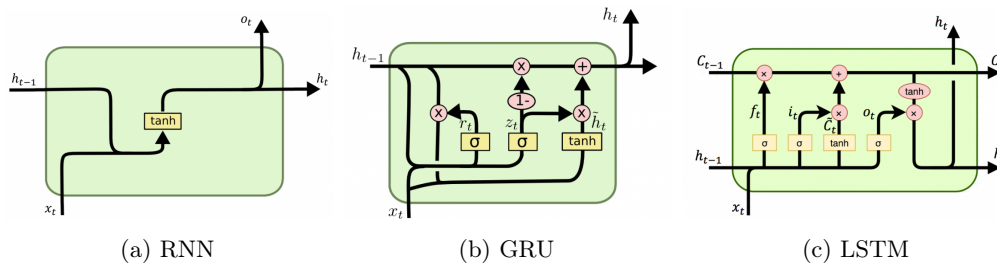


Figure 2.2: Comparison between RNN, GRU and LSTM cells. Source: [dprogrammer](#).

2.2 Sequence Models

2.2.1 Introduction

For sequential dependencies, the first established extension to a conventional feed-forward neural network was introduced by the name of a recurrent neural network (RNN). The RNN can handle variable-length input sequences that have hidden units that depend on the previous time step. To formalize, we introduce the input feature sequence of max length T_x as $\mathbf{x} = \{x_t\}_{t=1}^{T_x} \in \mathbf{R}^{T_x \times D}$ where D is the dimension of each sequential input $x_t \in \mathbf{R}^D$. For a shallow RNN with one hidden layer, where a schematic flow of the information can be seen in figure 2.2 (a), the hidden states h_t are then conventionally computed as follows:

$$h_t = \begin{cases} 0, & t = 0 \\ \phi(h_{t-1}, x_t), & t > 0 \end{cases} \quad (2.2.1)$$

where ϕ is some nonlinear function. The traditional way of implementing ϕ is similar to the likes of a conventional feed-forward network; by introducing weights for the input parameters to the hidden state:

$$h_t = g(Wx_t + Uh_{t-1}), \quad t > 0 \quad (2.2.2)$$

where g could e.g. be a sigmoid function or a hyperbolic tangent. In the case of an RNN with more than one hidden layer, the hidden state $h_t^{(l)}$ at level l simply extends equation 2.2.2 to:

$$h_t^{(l)} = g(W^{(l)}h_t^{(l-1)} + U^{(l)}h_{t-1}^{(l)}), \quad l > 1 \quad (2.2.3)$$

The output from the last layer of the RNN will be denoted as $y = \{y_u\}_{u=1}^{T_y}$, where T_y does not have to be equal to the max sequence size T_x . The learning phase of updating the weights W and U for an RNN is computed using a so-called back propagation through time (BPTT) algorithm which has seen various implementation attempts such as in [8]. The details of BPTT will not be covered here, however here we must point out that it has been shown that the RNN architecture struggles to learn long-term dependencies (e.g. the vanishing gradient problem) which has resulted in limited success of its application due to the fact that most optimization methods are gradient based [9]. Many attempts to solve this has been attempted, such as introducing other optimization methods e.g. as proposed in [10]. Another way to assess the issue has been by proposing new architectures. We also note that an RNN can be bi-directional, which implies that the hidden state calculation in equation 2.2.1 can be extended to:

$$h_t = \begin{cases} 0 & , & t = 0 \\ \phi(h_{t-1}, x_t) & , & t = T_x \\ \phi(h_{t-1}, h_{t+1}, x_t) & , & 0 < t < T_x \end{cases} \quad (2.2.4)$$

Two of the most known architectures that has been used in recent years are the Long-Term-Short-Memory (LSTM) [7] and Gated Recurrent Unit (GRU) [11]. Both these architectures utilize a gating mechanism attempting to solve the vanishing gradient problem. Each neuron has a gate that can either let through or forget information from previous sequence

steps. The key difference between these two networks is that a GRU cell lacks complexity compared to an LSTM cell such as an output gate. As a consequence, the network using a GRU architecture has fewer parameters to calculate. A comparison of the architecture between the different cells can be seen in figure 2.2. As examined in [12], for many applications, GRU's performance is similar to LSTM's, and both outperform a vanilla RNN. In recent years, for the application of LLM's architectures, more complex models have been introduced. These utilize a combination of different cells including CNN cells [13]. In the extent of developing architectures capturing longer sequence dependencies, one of the largest issues with sequential networks is their computational time. Models have been proposed to solve this, as in [14].

2.2.2 Encoder-Decoder

One common pre-processing step for machine learning approaches is the so-called Principal Component Analysis (PCA), which is a dimensionality reduction technique that can be either a linear or non-linear transformation. What the PCA does, lies in its name; perform a data transformation that projects the initial dataset onto its principal axes. The purpose of this is to capture the largest variations of features in the data, which could help the machine learning algorithm extract important information and increase the performance of a prediction. The same approach to data could be achieved using neural networks.

The Encoder-Decoder is a technique that has seen many successful applications in NLP in recent years [15]. The idea is to first feed the input data through a neural network denoted as an encoder, with the aim to, similar to a PCA, extract a fixed-size representation from the input data. The output of the encoder is then used as input to the decoder, where the purpose of the decoder is to use this new feature representation to perform predictions. The encoder-decoder approach is standard in most novel NLP approaches but has also seen light in applications of time series data. The encoder-decoder is a central part of the transformer networks which will be discussed later in this section.

2.2.3 Attention Model

The attention model was introduced in [16] for the purpose of machine translation for longer sentences, and later in [17] for image captioning where the attention is based on the image content. Opposed to previously popular search algorithms such as Beam Search, the attention model is designed to learn what other input positions in the input sequence contribute to the "context" of the input at position t . The initial attention model utilizes sequential neural networks, as introduced in the previous sub-section.

The model is implemented by introducing a set of so-called attention weights on top of the hidden state weights. To formalize the attention model, again consider the input sequence to be $x_t = (x_1, \dots, x_{T_x})$. Consider the encoder to be a shallow bi-directional RNN where the hidden states are called attention states and are defined as $a_t = (\vec{a}_t, \overleftarrow{a}_t)$, denoting the forward and backward recurrence respectively; a_t is simply a concatenation of these. The decoder will then be a simple (forward only) RNN with states s_t and output y_t . We denote The input vector to the decoder as the context, c_t . The encoder-decoder structure then implies that the context c_t for the decoder is dependent on the attention states a_t .

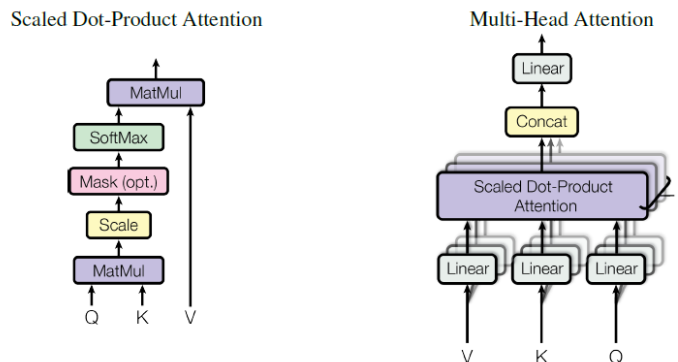


Figure 2.3: The Self-Attention mechanism in the transformer neural network. Source: [1]

These are then mapped together with attention weights $\alpha_{u,t}$ from the encoder to produce the decoder input:

$$c_t = \sum_u \alpha_{t,u} \times a_u \quad (2.2.5)$$

The attention weights $\alpha(t, u)$ can hence be interpreted as "how much should the output y_t at step t pay attention to the attention state a_u at step u ". The attention states $\alpha_{t,u}$ can be calculated using a softmax function as in [16]:

$$\alpha_{t,u} = \frac{\exp(e_{t,u})}{\sum_{t=1}^{T_x} \exp(e_{t,u})} \quad (2.2.6)$$

where $e_{t,u}$ commonly is calculated by a small neural network:

$$e_{t,u} = \phi(a_t, s_{t-1}) \quad (2.2.7)$$

The reasoning behind why it is implemented like this is that we do not know what the attention mapping from one feature step to another looks like, hence we use a small neural network to learn this dependency and trust back-propagation and gradient descent to learn these dependencies; which as shown in [16] [17] works. What still remains to be understood is if the attention mechanism works for time series data. As a final remark we note that one downside of this model is that, for the input size of T_x , and output size T_y , the model is costly in the aspect of that it has $T_x \cdot T_y$ attention parameters to compute.

2.3 The Transformer Neural Network

The transformer network [1] was published in 2017, and with this new architecture a new age of sequence models began. The network utilises the encoder-decoder setup and relies on the so called self-attention mechanism, together with dense neural networks and some positional tricks. It increases longer sequence understanding, while it reduces

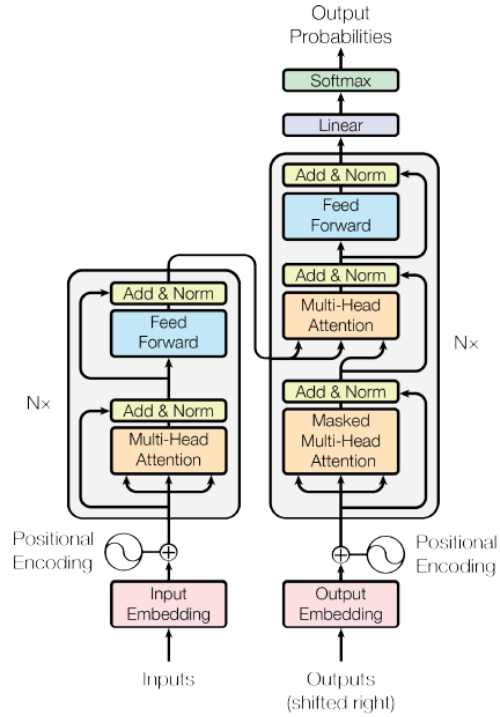


Figure 2.4: Schematic view of the transformer network. Source: [1].

computational time heavily. To understand the transformers, we start by exploring the self-attention and multi-headed attention mechanisms. The following parts will all be based on [1]. As a final note before we deep dive into the paper, as this paper will target time series, the word embedding that is covered in the paper will not be covered here.

2.3.1 Scaled Dot-Product Attention

The scaled Dot-Product Attention mechanism (also called self-attention) draws its inspiration from the attention model described in section 2.2.3, however with the important note that it does not build on an RNN-like architecture.

As previously, we denote the input sequence as $x = (x_1, \dots, x_{T_x})$. The purpose is to calculate an attention-based vector representation of each input x_t which is defined as $A_t = \text{Attention}(q_t, K, V)$, where the input arguments will be discussed later on. The purpose of calculating this representation is similar to the representation weights calculated in equation 2.2.6, described in section 2.2.3; we want to understand the input x_t as a function of the other values in the sequence and find the most "appropriate" representation

of x_t . The attention vectors are calculated similar to equation 2.2.6 as following:

$$\text{Attention}(q_t, K, V) = \sum_i \frac{\exp(q_t \cdot k_i)}{\sum_j \exp(q_t \cdot k_j)} v_i \quad (2.3.1)$$

where $Q = (q_1, \dots, q_{T_x})$ is called the *query*, $K = (k_1, \dots, k_{T_x})$ are called the *keys*, and $V = (v_1, \dots, v_{T_x})$ are called the *values*. In general, Q, K, V are packed into matrices, hence equation 2.3.1 can be expressed in the vectorised form in equation (1) in [1]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3.2)$$

where d_k is the dimension of the queries and keys. (q_t, k_t, v_t) are linearly weighted functions of the input:

$$\begin{aligned} q_t &= W^Q \cdot x_t \\ k_t &= W^K \cdot x_t \\ v_t &= W^V \cdot x_t \end{aligned} \quad (2.3.3)$$

where (W^Q, W^K, W^V) will be parameters of this learning algorithm. The intuition for these parameters lies in the analogy via their name. q_t can be seen as a query asking "what is happening at input x_t ". This query is then answered via a look-up of each of the key-value pair (k_u, v_u) to understand if x_u is an explanation of what is happening at x_t . The scaled dot-product attention mechanism from the paper can be seen to the left in figure 2.3.

2.3.2 Multi-headed Attention

Each time the attention is calculated for the input sequence x_t , it is called a *head*, hence the name multi-headed attention simply implies doing the scaled dot-product attention multiple times in parallel. Now, we let the number of heads be denoted by h , then each head will have its own learning parameters, this is defined by using equation 2.3.3 so that we get:

$$\text{head}_i = A(QW_i^Q, KW_i^K, VW_i^V) \quad (2.3.4)$$

where we note that new weights are introduced. As we introduced weights for each head, the weights in equation 2.3.3 are not necessary for some applications, the simplest implementation in this aspect of the multi-headed attention would be to e.g. set $Q = K = V = X$. How Q, K, V are implemented in the paper will be discussed in the next sub-section 2.3.3, however, once these are set equal, usually the name "self-attention" is used. We continue by noting that these h heads are then stacked together and multiplied by another set of learning weights yielding the multi-headed attention output:

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (2.3.5)$$

which can be seen to the right in figure 2.3. The multi-headed algorithm allows the algorithm to access information from different representation sub-spaces at different positions, allowing for a richer representation of the sequence.

2.3.3 The Transformer Architecture

Now once we have the multi-head attention mechanism in place, we can go through the architecture of the full transformer. In general, the architecture utilizes the encoder-decoder structure, where the input sequence $x = (x_1, \dots, x_{T_x})$ is mapped through the encoder which generates a sequence of continuous representation denoted as $z = (z_1, \dots, z_n)$. Given z the decoder then generates the output sequence denoted as $y = (y_1, \dots, y_{T_y})$. Important to note here is that the output sequence is generated one position at a time, using the previously generated outputs for the creation of the new. i.e, the output sequence is auto-regressive where the output of the network is not only conditioned on the input but also on all previous outputs in the sequence: $p(y_t|x, y_1, \dots, y_{t-1})$. The transformer is seen in figure 2.4.

Encoder

The encoder has two sub-layers as seen in the left of figure 2.4. The first sub layer is the MultiHead attention mechanism, from equation 2.3.5, and the second sub layer is a fully connected feed-forward neural network with ReLU activations, which is used for determining which features from the MultiHead sub layer are the most important ones. Around each sub layer, a residual connection [18] is applied together with a normalization layer, what this does will be covered later on with the positional encoding vectors. Finally, the encoder block is repeated $N_x = 6$ times in parallel in the paper.

Decoder

The decoder has three sub layers. The first part is a MultiHead layer with an input. The input will at the first iteration be empty, after one iteration it will reuse the first output it generates, then repeat this shift of its own outputs one step at a time until the complete output sequence is generated, thus making it auto-regressive. The purpose of this first sub layer is to generate the query, i.e. the Q parameter for the next sub layer. The second sub layer is again a MultiHead mechanism which takes the Q parameter from the previous step, but importantly, the key-value pair K and V arrive from the *encoder* output. This flow is visualised on the right in figure 2.4. The final sub layer in the decoder is again a fully connected feed forward network. As in the encoder, a residual connection is implemented around each sub layer. The decoder is also composed of $N_x = 6$ stacked layers as with the encoder. The output of all layers are finally combined together and passed through a linear and a softmax layer to generate output probabilities.

Positional Encoding

Since the transformer network does not have any recurrence or convolution, the model must contain some extra information in order to understand the position of the sequence. The transformer does this by applying positional encoding to the input of the encoder and decoder. The paper uses positional encoding from [19]. For each input x_t a positional vector p_t is generated and together with the input, the positional encoding vector is

generated by the following equations:

$$\text{PE}_{t,2i} = \sin\left(\frac{t}{10000^{2i/D}}\right) \quad (2.3.6)$$

$$\text{PE}_{t,2i+1} = \cos\left(\frac{t}{10000^{2i/D}}\right) \quad (2.3.7)$$

where we note that D is the dimension of x_t , and that also $p_t \in \mathbf{R}^D$. i will take the values up until of the integer division of $(D - 1)/2$, i.e. for $D = 5$, the index k of the vector p_t can be calculated as up until either $2i$ or $2i + 1$ reaches $D - 1$:

$$p_t = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}, \quad \begin{array}{ll} i = 0, & 2i = 0 \\ i = 0, & 2i + 1 = 1 \\ i = 1, & 2i = 2 \\ i = 1, & 2i + 1 = 3 \\ i = 2 & 2i = 4 \end{array}$$

Each p_t will then be a unique positional encoding vector.

Final Remarks

The residual connection mentioned in the description of the encoder and decoder sub layers, is intended for passing through the positional information from the p_t vector, this operation can be seen in figure 2.4 as the passing arrow around the MultiHead that ends in the "Add & Norm". The normalization layer used is not the common "batch normalization", but called "Layer Normalization" [20] which seems to be the most commonly used in sequence tasks. Regarding the network's ability to battle overfitting, the common dropout technique [21] is applied in multiple sub layers in both the encoder and decoder.

We finally note that in figure 2.4, the first MultiHead sub layer in the decoder block is "masked". This is a mechanism that is utilised during the training phase where the true output \hat{y} is partially used. Part of the true output sequence is passed through the decoder, and then used to check if the remaining sequence generated is accurately predicted. As mentioned in the beginning, the embedding used in the paper will not be necessary for the application of this thesis as the input to the network is time series, the "embedding size" in the paper will be this thesis dim D of the input vectors x_t . The paper uses the Adam optimizer [22].

2.4 Related Work

2.4.1 FX Spot Trading

Literature on studies and applications of high-frequency data in the FX spot market for price discovery is limited, especially with respect to short-term liquidity changes. The purpose of this chapter is to present some relevant and applicable literature for the construction of this thesis.

[23] studies short-term liquidity withdrawal in the FX spot market for eight different currency pairs, including the two Scandinavian currency pairs; EUR/SEK and EUR/NOK. Data used for the study comes from EBS and includes millisecond timestamps as well as if orders are limit- or market order. Liquidity withdrawal in short time windows are studied as well as different metrics. The paper presents some findings in features of the data. First, a wider bid-ask spread implies liquidity provision in all studied currency pairs except one. Secondly, large and aggressive limit orders has a large impact on changes in liquidity. Thirdly, order-splitting strategies seem to "successfully" avoid causing immediate liquidity withdrawal, which concurs well with the second finding. Fourth the paper argues that there exists some evidence for liquidity mirage in major currency pairs as new limit orders trigger more contributions to the different books in some of the major currency pairs.

[24] attempts to explain what impact the increasingly high turnover trend in the FX spot market for the last two decades has had on market liquidity. They point out that even though the bid-ask spreads have reduced, the fragmentation of the FX market has increased. Most importantly, the authors put forward that the greater activity might only provide an illusion of liquidity, i.e., a liquidity mirage. One possible explanation for this liquidity illusion is presented as the fact that HFT firms have shifted towards making markets rather than simply being takers of liquidity on multi-bank ECNs. It should be noted that the authors state that they lack enough evidence for this explanation to be accepted as a causal explanation.

[3] studies what implication the increasing share of algorithmic trading vs manual trading has had on the market liquidity. Their findings are based on EBS data between 2004 until 2010. They point out that the liquidity has changed towards being increasingly internalized for each year. Large FX banks internalize most of their flow and only turn to ECNs when they need to hedge their risk. Much like [24] they argue that the "newer" type of player, namely the market makers somewhat brings liquidity back to be more visible. All this again implies the fragmentation of the market and the challenges that creates with price discovery.

[25] studies price discovery and liquidity recovery in FX markets when the market reacts to macro announcements. Data used for the study comes from EBS from 1999-2017. The paper uses a rolling variance ratio measure as a quantitative indication of price discovery. For the liquidity recovery, the quantitative measure used is defined as the shrinkage of bid-ask spreads. The findings of the report (based on the proposed metrics) show that price discovery and liquidity recovery tend to be negatively correlated. The empirical findings suggest that over the time period the speed of price discovery is shown to improve and liquidity recovery has become slower over the years.

2.4.2 Sequence Models for Time Series

There are many publications on financial time series forecasting using different types of neural networks. In this sub section we present some findings that might prove useful for the thesis.

[26] attempts prediction of FX currency rates using a hybrid GRU-LSTM network, however only using 1-minute time series data of the mid price, for prediction of 10 and 30 min horizons. Their proposed model outperforms the vanilla GRU, and LSTM models respectively.

[27] introduces a model named CLSVSA which utilizes a hybrid approach of CNN's and LSTM units together with the Seq2Seq model [28] and a self-attention model in an encoder setup to capture features of commodity future prices. They compare the model to baseline models of each of the different architectures; LSTM, CNN and Seq2Seq and study the performance of predicting returns over the different assets. They evaluate the three different metrics, MAP (mean average precision), AAP (average annual return), and SP (sharpe ratio). The CLVSA outperforms all baseline models in all assets using these metrics.

[29] attempts at predicting short-term high dimensional data using a transformer neural network. Their proposed model is called Spatiotemporal Transformer Neural Network (STNN). The time series data is not of financial nature, however, the data sets used are gene expression data, solar, traffic flow, and many more. They utilise the transformer encoder-decoder structure, with similar information passing from the encoder to the decoder, however with the noticeable difference that there seems to not be any time embedding, and the self-attention models are replaced for the author's own attention model.

[30] attempts a prediction of crude oil price fluctuations using LSTM's and CNN's. Their findings suggest that the LSTM architecture outperforms the CNN architecture, at least when it comes to crude oil price predictions.

[5] is a new model that extends the transformer model to a new model called the *Informer*, which focuses on time series data, particularly long sequence time-series forecasting (LSTF). They point out that one of the main issues with the transformer model when it comes to time series is that the self-attention calculations are expensive compared to when one uses embedded text messages, as the text embedding is relatively sparse compared to a multi-dimensional time series. They attempt to overcome this performance issue by introducing a new self-attention mechanism called PropSparse. Further, The most prominent feature change compared to the vanilla transformer is that the informer models use 1-dimensional convolutional layers instead of dense layers on top of the PropSparse attention mechanism. The model is evaluated on both univariate and multivariate time series forecasting where it outperforms all compared models.

The NN research on financial time series in recent years has seen many interesting hybrid approaches of architectures, almost all outperforming "pure" architectures such as GRUs and LSTMs; what seems to be evident is that the simpler architectures are not sufficient for handling noisy time series. Moreover, only a small fraction of publications has been done on full orderbook data, in most cases the dataset consists of aggregated, even time interval sampled data such as OHLC (open-high-low-close) time series of each interval.

Chapter 3

Methodology

3.1 Data Collection

3.1.1 Targeted Dataset

This report will target the currency pair EUR/SEK over the period 2023-11-06 until 2023-11-10. Each day will contain orderbook updates between the times 8:00 and 15:30 (CET). The motivation for excluding data outside these hours is that the market is most active over this time window for the currency pair EUR/SEK, introducing data samples outside this period would just introduce more complexity not necessary for the thesis.

The data used for constructing the dataset will use all orderbook information available, that is, data containing the same information as shown in table 2.2, that will then be heavily exploited with regards to liquidity and price dynamics, details on this is presented in subsection 3.1.3. The price level buckets are (semi)-arbitrarily chosen to be equal to (1, 2, 4, 6, 8) million EUR, hence the output target size will be $T_y = 5$. The targeted dataset will have around 600k samples, with a sampling step of $\tau = 0.1s$, number of timesteps, $T_x = 100$, and the dimensionality of the data $d = 76$. The forecasting horizon is $T = 1s$. This implies that we use a 10s data window to predict the next 1s average spread for all price level buckets.

3.1.2 Sampling

The time series will be evenly sampled with a fixed time step τ which will yield small time buckets. Each data point is created by using a rolling window, that will have a fixed number of time steps T_x , where each time step will have a fixed feature dimension d , we denote each data point input to the neural networks as $X_t \in \mathbf{R}^{T_x \times d}$. The target prediction size will also be of a fixed size T_y , where the one target point will be denoted as $y_t \in \mathbf{R}^{T_y}$. Note that the Transformer Network allows for variable size inputs, and this method is simply motivated by the purpose of comparability with baseline models. By allowing for different inputs for the baseline models and the transformer architecture it

will not be possible to understand whether the implemented architecture or the different shape yielded (possible) different results.

3.1.3 Features Construction

Price Features

Given raw orderbook data as the example in table 2.1, and a fixed sampling size of τ the reference bid and ask prices, and correspondingly the spread and mid values, for the different price levels will be constructed according to the equations in 2.1.5. A relatively longer example of such data is seen in figure 3.1 and the corresponding mid and spread representation is seen in figure 3.2.

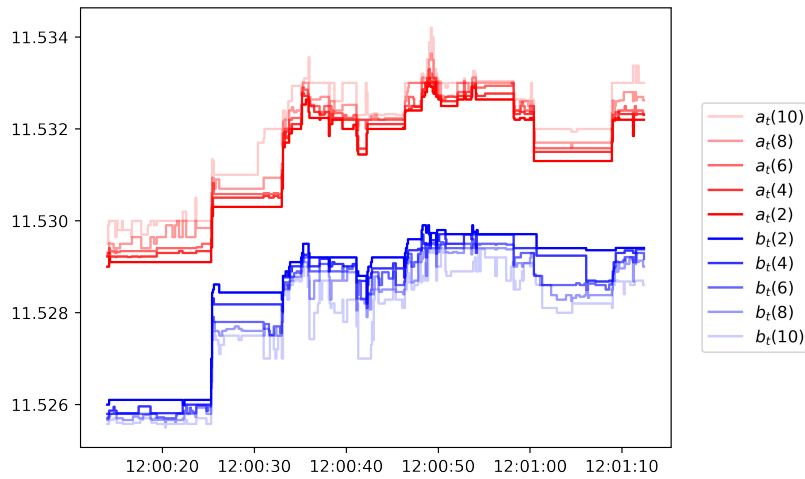


Figure 3.1: Reference bid and ask prices constructed for level volumes of (2, 4, 6, 8, 10) million Euros over a 1 minute frame with $\tau = 100ms$.

If we zoom in on the figures (3.1, 3.2), we see the even steps of size $\tau = 100ms$ in figure A.1.

The prices taken are those of the last price of each τ -time bucket. By sampling, some information may inevitably disappear in the case of many orderbook updates that are arriving at a rate faster than τ . To tackle this, aggregates over liquidity changes over each time bucket are applied.

Liquidity Features

Before the sampling is applied, some liquidity features are computed and then aggregated. What we wish to study is the liquidity addition and withdrawal from one-time step to another. Recall from section 2.1.3 the level volumes $v \in \mathbf{R}^l$, the constructed prices (a_t^l, b_t^l) , the sorted prices and volumes (a_t', b_t') , (α_t, β_t) .

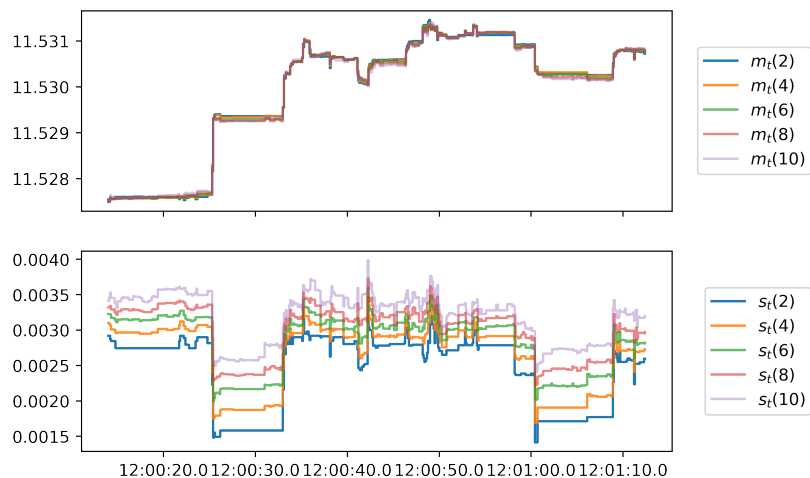


Figure 3.2: Corresponding mid and spread representation of figure 3.1.

	Name	Description
1	Bid/Ask size Δ	How much the absolute volume change.
2	Bid/Ask size Δ Addition	The total liquidity added.
3	Bid/Ask size Δ Withdrawal	The total liquidity withdrawn.
4	Bid/Ask size Δ given b_{t-1}^l	Given the previous price level, what is the absolute liquidity change at that price?
5	Bid/Ask size Δ Addition given b_{t-1}^l	Given the previous price level, what is the total liquidity addition?
6	Bid/Ask size Δ Withdrawal given b_{t-1}^l	Given the previous price level, what is the total liquidity withdrawal?

Table 3.1: Liquidity Features per side and volume bucket.

Consider two succeeding snapshots of an order book at time t_{i-1} and t_i , further for simplicity only consider one side, e.g. the *ask*. Given some previous price level a_{t-1}^l , as well as the snapshots a'_{t-1} , a'_t , has any liquidity changed between the two snapshots, i.e. given the newest snapshot and the older snapshot truncated by the previous price level, has any part of the orderbook changed, and if so, with how much liquidity? The complete list of liquidity features calculated is presented in table 3.1, where we point out that these are calculated for each volume level.

The liquidity feature (1) is seen in figure 3.3, where feature 4 would be visualized in the same way.

Another more informative way to aggregate this information is to look at exactly how much liquidity was added, and how much was withdrawn; in this way the total volume of liquidity changes during a time bucket will be captured. This is represented by feature

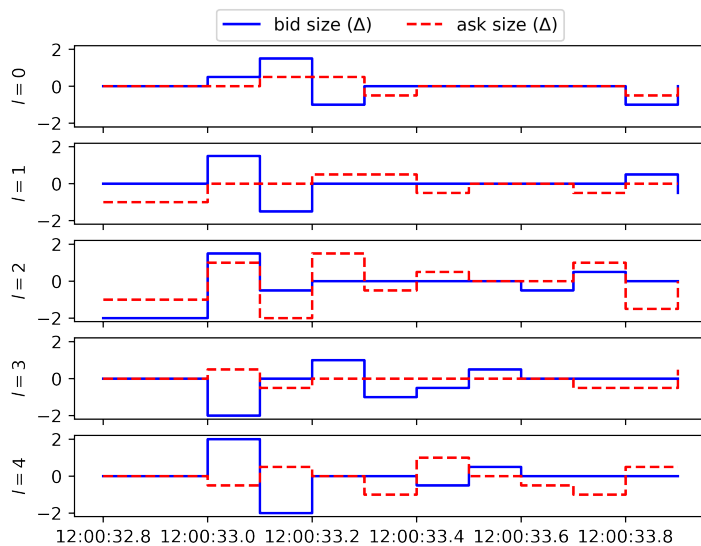


Figure 3.3: Liquidity changes over each time bucket and level, for the prices in figure A.1.

(2,3,5,6) in table 3.1. An visual example of this type of aggregation is shown in figure 3.4 where each subplot displays a time step, and the y-axis contains the bucket indication by bid and ask side.

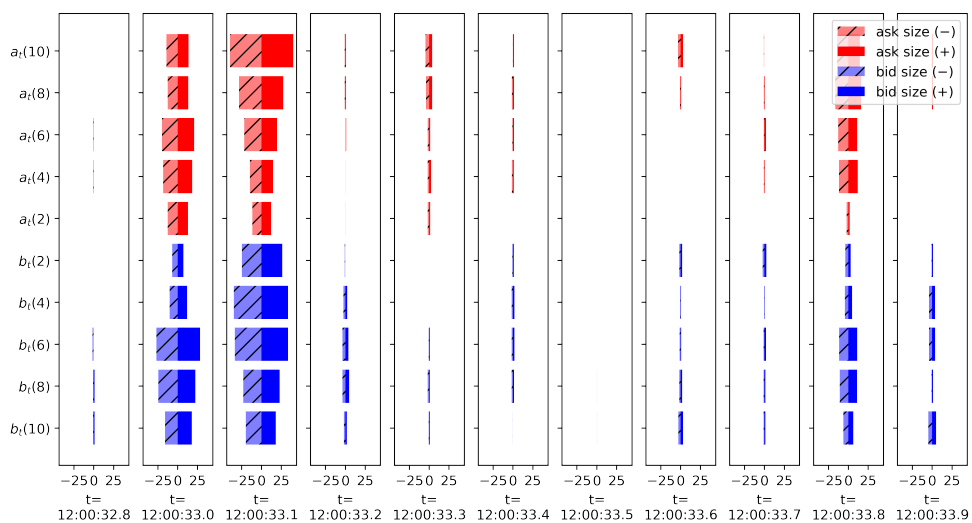


Figure 3.4: Liquidity withdrawals and additions over each time bucket and volume level, for the prices in figure 3.3. The levels are constructed using even 2M EUR steps.

Finally, the number of updates per side can be seen in figure 3.5.

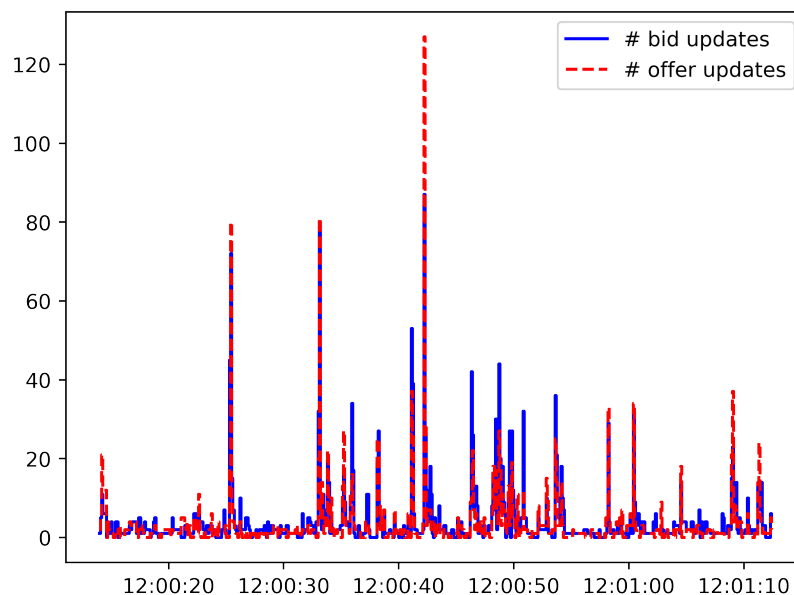


Figure 3.5: Number of orderbook updates by side for the prices in figure 3.1

3.1.4 Target Variables

As briefly noted in the introduction of the report, we attempt to predict the spread of the consolidated market for a certain finite step future horizon H , this is done by targeting the average spread for each volume bucket at that horizon H :

$$y_t^l = \frac{1}{H} \sum_{k=t+1}^{t+H} s_k^l, \quad \forall l \quad (3.1.1)$$

The resulting target variable will be seen as a return based on the latest observable spread in each input sample X_t .

3.1.5 Data Normalization

The price features will be transformed into logarithmic returns to make the time series stationary. Further, all price features will then be scaled by removing the mean and scaling to unit variance, this scaler will in the feature be referred to as a *Standard Scaler*. Since the liquidity features are static by nature, only the Standard Scaler will be applied to these features. Finally, we point out that the scaling will only be fitted to the training data in order to remove any look-ahead information from the testing set. The scaling will be applied to the flat training dataset before the complete dataset using a rolling window is created.

3.2 Model Design

3.2.1 Test Environment

Tensorflow version 2.16.1 via the Keras version 3.3.3 API is used for building and training the models. The hyperparameter tuning is performed using Keras Tuner [31] version 0.11.1. All training is done on the Tensorflow CPU version with 32GB of available RAM. The learning rate during the training phase of the models will be scheduled to decrease in rate with appropriate steps, inspired by the learning rate schedule in [1].

Due to the size of the dataset and limitations in memory, only parts of the flat training dataset will be stored in memory at the same time, and the data pipeline will include an iterator based on [tensorflow.data.Dataset](#) to create the input tensors.

3.2.2 Baseline models

To understand the performance of the transformer architecture we will evaluate it against two different baseline models, namely a vanilla RNN and a LSTM network. Recall that the input data $X_t \in \mathbf{R}^{T_x \times d}$, and target labels $y_t \in \mathbf{R}^{T_y}$ which implies that the input shape is (batch size, T_x , d) and the output shape is (batch size, T_y). A flowchart of the baseline model design can be seen in figure 3.6 where the recurrent layer will be comprised of either RNN or LSTM cells. The number of cells is equal to T_x in both cases, and the hidden states are to be hyperparameter-tuned. The models are designed using the Keras Functional API. On top of these models, we will also compare performance against two simplistic models. First, a one-step martingale, which will set the prediction to be the last observed value, this model is denoted as $M(1)$. The second simplistic model we will use is always to guess a zero return, denoted as $M(\emptyset)$.

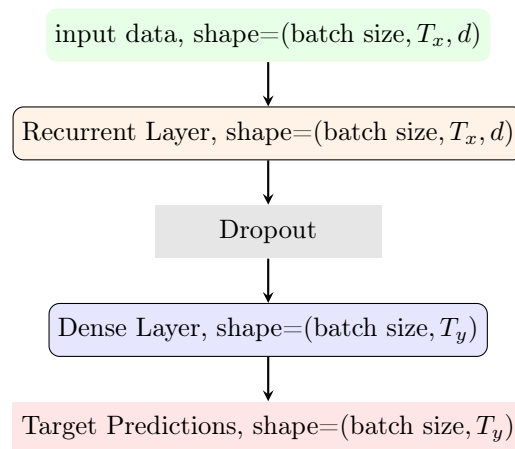


Figure 3.6: Flowchart of baseline prediction models based on the keras.layers API.

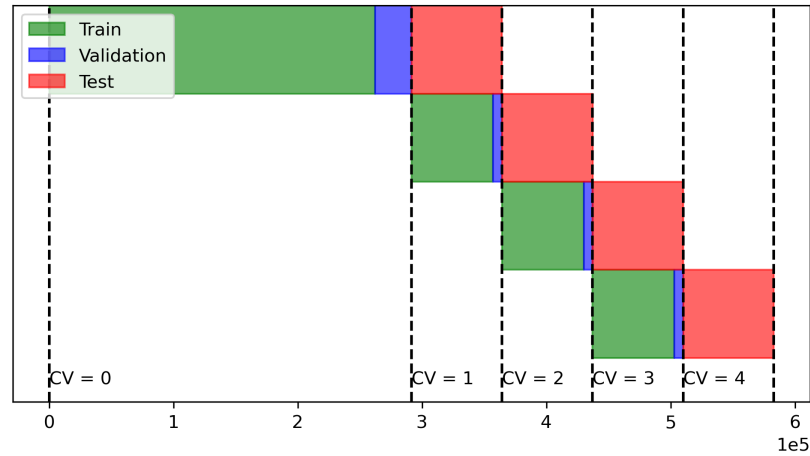


Figure 3.7: Time Series Cross Validation Schematic.

3.2.3 The Transformer Implementation

This implementation of the Vanilla Transformer (VT) uses the KerasNLP [32] library which extends the standard Keras library with some additional layers, the architecture is a replication of the one in figure 2.4 without the Embedding, and with a dense output layer with linear activations for the output instead of a softmax activation. Another transformer inspired by the Informer model in [5] has been created, the implementation can be seen in figure 3.8 and is referred to as the Convolutional Transformer (CT). The shape parameters (T_x, T_y, d) are all fixed and follow the same definition as for the baseline models, however, there are multiple parameters to be tuned here, primarily in the Self-Attention module. The hyperparameter tuning outcome will be presented in the Results sections. If one wishes to compare the VT and CT architecture, it can be seen that the first part of the CT model is almost identical to the encoder in VT, with the most prominent exception that all dense layers have been swapped for convolutional layers. The last part from the pooling layer until the output can be seen as a simplified decoder.

3.2.4 Validation & Reliability

To validate the model and understand its reliability and robustness, out-of-sample folded cross-validation for time series [33] is used for $N_{cv} = 5$ folds. The model will initially train using the first fold, and then test on the second fold. The network will then utilize the second fold for training, and then test on the third fold. This pattern is then repeated until the last N_{cv} fold has been tested. A visual representation of this is seen in figure 3.7. By utilizing the Train/Validation/Test split, the model training is to be finished as soon as the *validation* loss has converged.

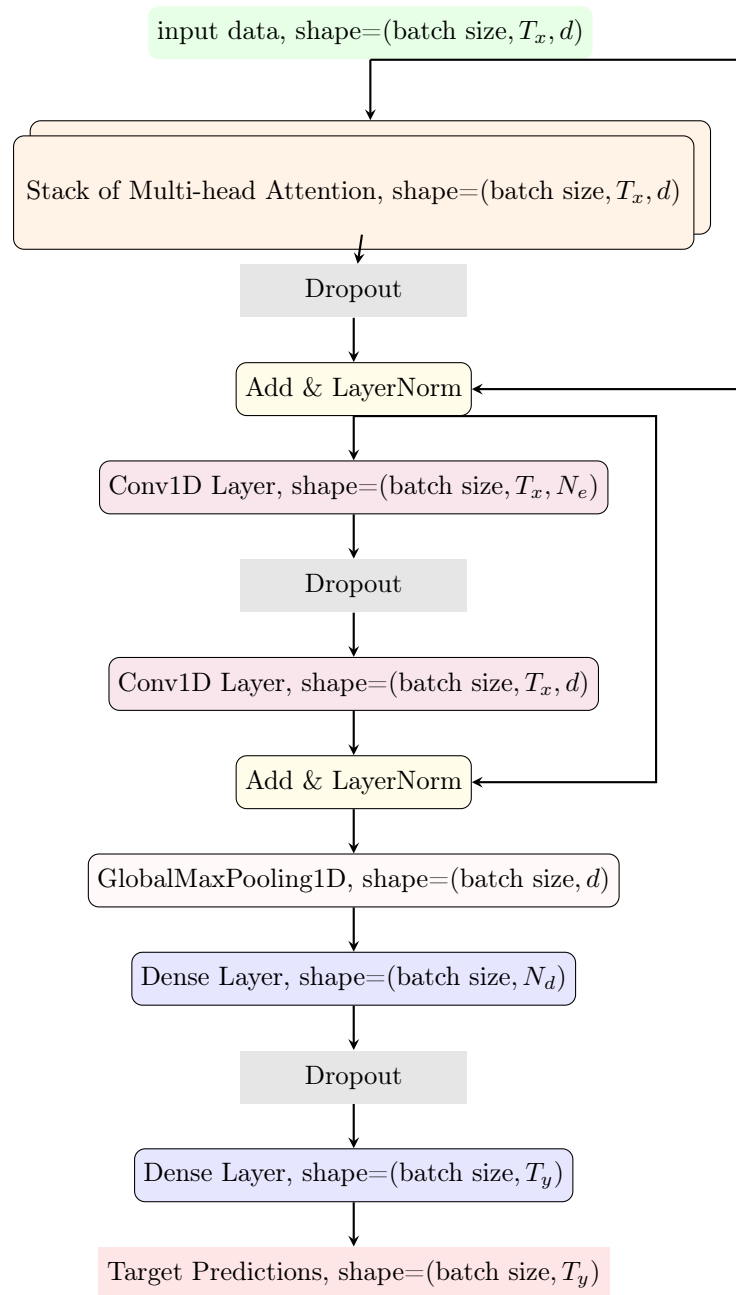


Figure 3.8: Flowchart of the Convolution Transformer (CT) prediction model based on the Keras.layers API.

Chapter 4

Results

4.1 Hyperparameter Outcome

The hyperparameter search for the two baseline models yielded an outcome where the primary factor with a sizeable impact on the prediction power of the validation set, was the batch size and the dense output layers dropout level. The hyperparameter search for the baseline models resulted in chosen parameters according to table B.1. The parameters for the CT model can be seen in table B.2. The hyperparameter search resulted in an RNN model with 9348 trainable parameters, an LSTM model with 36421 trainable parameters, and a CT model with 123925 trainable parameters. It should be noted that the search allowed for a larger trainable space for both the RNN and LSTM.

4.2 Prediction results

As an aggregated score of the models, we use the root mean squared error (RMSE), as the loss function of the optimization was the mean squared error. We also include mean absolute error (MAE) for each model. The RMSE averaged over all cv folds, for each volume bin is seen in table 4.1, and the corresponding MAE score if seen in table 4.2. The detailed results for each CV fold in the test and the average score for all folds can be seen in table 4.3. The Vanilla transformer will not be included in the results, why this is the case will be discussed in Chapter 5.

As can be seen in the results in table 4.3, the CT model outperforms the baseline models with some margin with respect to the RMSE score in all different cv folds and volume bins, however the MAE score is much closer between the models. The distribution of the errors per each model is presented in appendix B, where the error distribution is visualised as a comparison between the CT model and the baseline models. The figures B.3, B.4, B.5 represent this comparison using a linear scale. The same error distributions, but with a logarithmic y-axis is seen in figures B.6, B.4, B.5 to highlight the error tails.

bin	RNN	LSTM	CT	$M(1)$	$M(\emptyset)$
1	0.0916	0.0924	0.0755	0.1228	0.0918
2	0.0624	0.0631	0.0522	0.0837	0.0625
4	0.0409	0.0416	0.0363	0.0554	0.0410
6	0.0350	0.0356	0.0319	0.0473	0.0350
8	0.0335	0.0341	0.0313	0.0458	0.0335

Table 4.1: RMSE for all tested folds for the baseline models RNN, LSTM, the proposed Convolutional Transformer (CT) as well as the two simplistic predictions models $M(1)$, $M(\emptyset)$.

bin	RNN	LSTM	CT	$M(1)$	$M(\emptyset)$
1	0.0442	0.0469	0.0440	0.0645	0.0432
2	0.0320	0.0341	0.0316	0.0468	0.0317
4	0.0242	0.0253	0.0235	0.0351	0.0241
6	0.0212	0.0221	0.0209	0.0306	0.0211
8	0.0211	0.0219	0.0210	0.0306	0.0211

Table 4.2: MAE for all tested folds for the baseline models RNN, LSTM, the proposed Convolutional Transformer (CT) as well as the two simplistic predictions models $M(1)$, $M(\emptyset)$.

Finally, confusion matrices based on the predicted values have been created for the models and can be seen in appendix B.4. The predicted values have all been mapped to the labels to the return directions $(-1, 0, 1)$ where the return was based on the quantile mappings of:

$$\begin{aligned}
 -1 &\in (-\infty, Q_{0.4}) \\
 0 &\in [Q_{0.4}, Q_{0.6}] \\
 1 &\in (Q_{0.6}, \infty)
 \end{aligned}$$

where $(Q_{0.4})$ and $(Q_{0.6})$ denote the 0.4-quantile and 0.6-quantile of the target variables respectively. This range of zero guess is between 100 – 300% smaller than the standard deviation of the prediction values, i.e. the ratio $2\sigma/(Q_{0.6} - Q_{0.4}) \in (10, 30)$ where σ is the standard deviation of the $y_t^i, \forall t$ target variables.

CV Fold	bin	RNN		LSTM		CT	
		RMSE	MAE	RMSE	MAE	RMSE	MAE
2	1	0.0968	0.0448	0.0976	0.0475	0.0795	0.0445
	2	0.0642	0.0322	0.0649	0.0343	0.0535	0.0317
	4	0.0414	0.0242	0.0421	0.0253	0.0365	0.0236
	6	0.0352	0.0212	0.0359	0.0221	0.0319	0.0209
	8	0.0337	0.0211	0.0343	0.0219	0.0313	0.0211
	Avg.	0.0543	0.0287	0.0549	0.0302	0.0465	0.0283
3	1	0.1147	0.0541	0.1156	0.0567	0.1041	0.0585
	2	0.0832	0.0419	0.0839	0.0438	0.0759	0.0438
	4	0.0645	0.0334	0.0650	0.0344	0.0603	0.0342
	6	0.0547	0.0292	0.0551	0.0300	0.0509	0.0294
	8	0.0516	0.0292	0.0519	0.0299	0.0481	0.0289
	Avg.	0.0737	0.0376	0.0743	0.0389	0.0679	0.0390
4	1	0.1222	0.0574	0.1230	0.0604	0.1007	0.0573
	2	0.0734	0.0379	0.0744	0.0403	0.0631	0.0392
	4	0.0448	0.0259	0.0458	0.0275	0.0434	0.0286
	6	0.0393	0.0232	0.0402	0.0246	0.0379	0.0248
	8	0.0378	0.0229	0.0384	0.0240	0.0363	0.0242
	Avg.	0.0635	0.0334	0.0644	0.0354	0.0563	0.0348
5	1	0.0815	0.0404	0.0829	0.0429	0.0705	0.0441
	2	0.0528	0.0283	0.0540	0.0301	0.0470	0.0299
	4	0.0384	0.0228	0.0392	0.0237	0.0354	0.0231
	6	0.0343	0.0208	0.0351	0.0216	0.0317	0.0209
	8	0.0346	0.0216	0.0352	0.0223	0.0321	0.0216
	Avg.	0.0483	0.0268	0.0493	0.0281	0.0433	0.0279
Avg.		0.0600	0.0316	0.0607	0.0332	0.0535	0.0325

Table 4.3: Comparison of RNN, LSTM, and the Convolutional Transformer (CT) models using RMSE and MAE metrics, where the model was trained on fold 1-4, and consequently tested on fold 2-5 respectively, for each volume bin.

Chapter 5

Discussion

5.1 Results

5.1.1 Predictive power of the model

As shown in Tables 4.1 and 4.3, the CT model consistently outperforms both the RNN and LSTM architectures across all volume buckets and cross-validation (CV) folds when evaluated using the RMSE metric. However, when considering the mean absolute error (MAE) metric, as seen in Table 4.2, the performance differences between the models are much smaller. Consequently, determining which model is superior depends on the specific type of error that is more critical for the application. To further this discussion, Appendix B presents the error distributions for the models, first on linear axes and then with a logarithmic y-axis to emphasize the tails of the distributions.

It is evident that the RNN and LSTM models have heavier tails compared to the CT model, despite having similar MAE values. This suggests that while the RNN and LSTM models often produce predictions with small errors, they also tend to make more severe prediction errors. In contrast, the CT model rarely makes severe errors but struggles to consistently achieve very low prediction errors.

Next, we consider the simplistic models $M(1)$ and $M(\emptyset)$. The RMSE and MAE errors for $M(\emptyset)$ represent the variation metrics of the prediction variables, essentially making it a "no-guess" model. The $M(1)$ model, which predicts based on the last observed movement, serves as an "extremely static guess". Although these simplistic models may appear trivial, they are crucial in the analytical process as they provide a baseline for comparing the relative RMSE and MAE errors, thereby adding context to the evaluation of the model's performance. As shown in the tables, the CT model outperforms both $M(1)$ and $M(\emptyset)$ when evaluated using the RMSE metric. However, for the MAE error, the analysis is similar to that of the RNN and LSTM models, with the performance differences being less pronounced.

To further enhance understanding of the model's ability to predict direction, confusion

matrices have been created as described in the results section 4. These matrices can be found in Appendix B.4. In summary, the zero label contains a distribution mass of 0.2, while the up and down labels each contain 0.4 of the mass. The issue with predicting no movement is evident; the CT model has a success rate of 53 – 58% in correctly predicting an upward price movement. For downward movement, the model has a prediction success rate of 50 – 60% for volume buckets 1, 2, and 4, but struggles with less than 50% success for volume buckets 6 and 8. This is where the differences compared to the RNN and LSTM models are notable: neither model achieves close to 50% accuracy in predicting upward or downward movements, although both models perform better than the CT model in predicting the zero label.

These observations highlight the strengths and weaknesses of the CT model in comparison to the RNN and LSTM models. While the CT model shows better performance in predicting directional movements, it faces challenges in accurately predicting no movement. This insight is crucial for understanding the model's behavior and potential areas for improvement, depending on the specific application requirements. Future work could explore methods to address the CT model's difficulty with zero predictions, potentially enhancing its overall predictive performance.

5.1.2 Applicability of the proposed model in market making

As discussed in the previous subsection, the primary limitation of the CT model is its inadequate performance in predicting zero movement. Despite outperforming all baseline models in predicting price movements, the model struggles with accurately forecasting periods of no movement, which constitute a significant portion of the returns. This deficiency poses a substantial risk for market makers, as incorrect predictions of movement when there is none can lead to unwanted risk exposure.

However, it is promising that the CT model significantly surpasses the baseline models in predicting movements. For the model to be truly effective for market makers, improving its precision in predicting no movement is essential. Several potential enhancements to the model could address this issue.

Firstly, increasing the amount of training data might improve the model's accuracy in predicting zero movement. A larger dataset could provide a more comprehensive understanding of market behavior, leading to better performance.

Secondly, during the training phase, returns are normalized to logarithmic returns, a common practice to stabilize prediction targets and prevent the model from always predicting zero. Since this does not seem to be problematic for the CT model, reconsidering the use of logarithmic normalization might be beneficial. Training on raw returns instead of logarithmic returns could potentially improve the model's ability to predict no movement accurately.

Thirdly, as noted in the background section 2.4.1, many studies focus on predicting labels rather than performing regression predictions with real values. Adopting a classification approach, where the model predicts discrete labels for upward, downward, and no movement, might enhance its performance in predicting no movement.

In conclusion, while the CT model shows substantial promise in predicting price movements, its suitability for market making depends on improving its accuracy in predicting zero movement. Future work should explore these proposed enhancements to optimize the model's performance, ensuring it meets the specific needs of market makers.

5.2 The Architecture

The architectural decision to use 1-dimensional convolutional layers, instead of the dense layers along with a sine positional encoding vector was heavily influenced by the Informer model presented in [5]. As mentioned in the methodology chapter, an attempt was made to utilize the transformer architecture described in [1]. However, this approach did not yield any convergent results in the training phase. The main issue appeared to be that the positional encoding vector rendered the original feature vectors ineffective. Given this outcome, we initially considered exploring alternative positional encodings, such as learnable positional encodings. However, after conducting further research, we opted to adopt an Informer-inspired architecture. This decision was further strengthened by several papers referenced in section 2.4 and supported by practices from the Keras and TensorFlow documentations. As depicted in figure 3.8, our model does not include an autoregressive decoder, unlike the original transformer architecture in [1].

The absence of an autoregressive decoder represents a significant area for potential improvement in the CT model. Incorporating such a component could enhance the model's performance in sequential prediction tasks. However, implementing an effective autoregressive decoder is a complex task. It might be beneficial to consider using the full Informer architecture initially, as it includes an autoregressive decoder designed for long-term time-series forecasting.

In conclusion, while the current CT model architecture has shown promise, further enhancements, particularly the inclusion of an autoregressive decoder, could substantially improve its predictive capabilities. Future work should explore these architectural adjustments, potentially leveraging the full Informer model, to optimize the model's performance.

5.3 Improvements

To enhance the robustness and applicability of the CT model, several areas for improvement have been identified. Here we summarize and conclude the alterations of the model design and architecture that were mentioned and discussed in the previous section 5.1.

First and foremost, the model requires more extensive backtesting. A single week of training and testing is insufficient to confidently determine the model's efficacy. Extending the backtesting period would provide a more comprehensive evaluation of the model's performance under varying market conditions. To facilitate this, it would be prudent to switch to the GPU version of TensorFlow, which would significantly reduce training times and allow for more extensive experimentation and hyperparameter tuning.

In this thesis, we used a fixed input sequence length, sampling length (τ) and a fixed

forecasting horizon. Future work should consider exploring multiple forecasting horizons (H) to understand the model's performance over different timeframes. Additionally, treating the fixed sequence size (T_x) and sampling length (τ) as a hyperparameter and tuning them could lead to better model optimization and potentially improve predictive accuracy.

The current model was trained on a single currency pair. For a market maker, it is crucial to understand how the model performs across different currency pairs, each with distinct market dynamics and liquidity profiles. Testing the model on various currency pairs would provide valuable insights into its generalizability and robustness in different market environments.

The most significant improvements involve addressing the model's difficulty with zero movement predictions, as discussed in 5.1.1. Changing the problem formulation from regression to classification could potentially enhance the model's performance in this area. By using a softmax activation function in the last layer, the model could predict price movements as belonging to different "classes" (e.g., ranging from "large downward movement" to "large upward movement"). This approach might improve its ability to accurately predict periods of no movement, thus making the model more suitable for real-world application.

Finally, we add the decoder as the most prominent improvement for the model as the spread on different liquidity levels is heavily cross-correlated, and an autoregressive decoder could better capture these dependencies.

In summary, extending the backtesting period, exploring different forecasting horizons, training on multiple currency pairs, reformulating the prediction task as a classification problem, and incorporating an autoregressive decoder should be considered as improvements in future work.

5.4 Conclusions

In this thesis, we introduced and evaluated the performance of the Convolutional Transformer (CT) model in predicting spread movements compared to RNN and LSTM architectures. The CT model demonstrated superior performance in terms of RMSE, consistently outperforming both RNN and LSTM models across various volume buckets and cross-validation folds. However, when evaluated using the MAE metric, the performance differences were less pronounced.

Despite the CT model's strong performance in predicting spread movements, it faced challenges in accurately predicting periods of no movement, a crucial aspect for application due to the high density of returns centered around zero. This limitation poses a risk of unwanted risk exposure for market makers if the model incorrectly predicts movement when there is none.

To address these challenges and enhance the CT model's applicability for market-making, several improvements were proposed. Including more extensive backtesting, changing

the prediction task from regression to classification, and incorporating an autoregressive decoder are identified as a possible significant enhancement.

Future work should focus on implementing these improvements to develop a more accurate, reliable, and versatile model that meets the specific needs of market makers. By addressing the identified limitations and enhancing the model's predictive capabilities, the CT model can become a valuable tool for SEB in their role as a market maker.

Bibliography

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [2] Andréa M Maechler. Fx execution algorithms and market functioning. Technical report, Bank for International Settlements, 2015.
- [3] Nina Karnaukh, Angelo Ranaldo, and Paul Söderlind. Understanding fx liquidity. *The Review of financial studies*, 28(11):3073–3108, 2015.
- [4] Roel Oomen. Execution in an aggregator. *Quantitative Finance*, 17(3):383–404, 2017.
- [5] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021.
- [6] M. Butz and R. Oomen. Internalisation by electronic fx spot dealers. *Quantitative Finance*, 19(1):35–56, 2019.
- [7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [8] George Bird and Maxim E. Polivoda. Backpropagation through time for networks with long-term dependencies, 2021.
- [9] Razvan Pascanu, Tomáš Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.
- [10] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks, 2012.
- [11] KyungHyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.
- [12] Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

- [13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [14] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.
- [15] KyungHyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.
- [16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv.org*, 2014.
- [17] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.
- [19] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv.org*, 2017.
- [20] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
- [21] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.
- [22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv.org*, 2014.
- [23] Alexis Stenfors and Masayuki Susai. Liquidity withdrawal in the fx spot market: A cross-country study using high-frequency data. *Journal of international financial markets, institutions money*, 59:36–57, 2019.
- [24] Michael King and Dagfinn Rime. Algorithmic trading and fx market liquidity. *CFA Magazine*, 22:15–17, 05 2011.
- [25] Masahiro Yamada and Takatoshi Ito. Price discovery and liquidity recovery: Forex market reactions to macro announcements. *Journal of International Money and Finance*, 120:102502, 2022.
- [26] M.S. Islam and E. Hossain. Foreign exchange currency rate prediction using a gru-lstm hybrid network. *Soft Computing Letters*, 3:100009, 2021.
- [27] Jia Wang, Tong Sun, Benyuan Liu, Yu Cao, and Hongwei Zhu. Clvsa: A convolutional lstm based variational sequence-to-sequence model with attention for predicting trends of financial markets. In *Proceedings of the Twenty-Eighth International Joint*

Conference on Artificial Intelligence, IJCAI-2019. International Joint Conferences on Artificial Intelligence Organization, August 2019.

- [28] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.
- [29] Yujie You, Le Zhang, Peng Tao, Suran Liu, and Luonan Chen. Spatiotemporal transformer neural network for time-series forecasting. *Entropy*, 24(11), 2022.
- [30] Rayan H. Assaad and Sara Fayek. Predicting the price of crude oil and its fluctuations using computational econometrics: Deep learning, lstm, and convolutional neural networks. *Econometric research in finance : ERFIN*, 6(2):119–137, 2021.
- [31] Tom O’Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. <https://github.com/keras-team/keras-tuner>, 2019.
- [32] Matthew Watson, Chen Qian, Jonathan Bischof, François Chollet, et al. Kerasnlp. <https://github.com/keras-team/keras-nlp>, 2022.
- [33] Vitor Cerqueira, Luis Torgo, and Igor Mozetič. Evaluating time series forecasting models: an empirical study on performance estimation methods. *Machine Learning*, 109(11):1997–2028, October 2020.

Appendix A

Dataset Features

A.1 Feature Visualisations

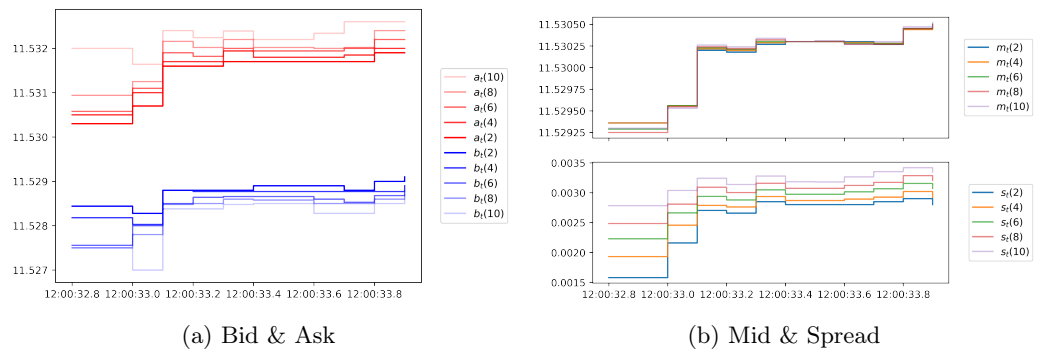


Figure A.1: Reference bid, ask, mid, spread prices zoomed in from figure 3.1 and 3.2.

A.2 Full Featureset

	Feature Name	bin	Description
0	mid_lim	(1, 2, 4, 6, 8)	Mid Price at bin, as log return.
1	mid_offset	(2, 4, 6, 8)	Mid Price at bin, offset in price from bin=1M.
2	s_lim	(1, 2, 4, 6, 8)	Spread at bin, as log return.
3	s_lim_avg	(1, 2, 4, 6, 8)	1s avg. Spread at bin, as log return.
4	s_offset	(2, 4, 6, 8)	Spread at bin, offset in price from bin=1M.
5	bid_size_lim	(1, 2, 4, 6, 8)	Bid size at bin.
6	bid_size_plim	(1, 2, 4, 6, 8)	Bid size at bin, given previous price.
7	offer_size_lim	(1, 2, 4, 6, 8)	Offer size at bin.
8	offer_size_plim	(1, 2, 4, 6, 8)	Offer size at bin, given previous price.
9	bid_size_d_plim	(1, 2, 4, 6, 8)	Bid size absolute change at bin, given previous price.
10	offer_size_d_plim	(1, 2, 4, 6, 8)	Offer size absolute change at bin, given previous price.
11	bid_size_d_plim_wd	(1, 2, 4, 6, 8)	Bid size total added volume at bin, given previous price.
12	bid_size_d_plim_ad	(1, 2, 4, 6, 8)	Bid size total withdrawn volume at bin, given previous price.
13	offer_size_d_plim_wd	(1, 2, 4, 6, 8)	Offer size total added volume at bin, given previous price.
14	offer_size_d_plim_ad	(1, 2, 4, 6, 8)	Offer size total withdrawn volume at bin, given previous price.
15	n_bid_updates	-	Number of bid updates, not by bin.
16	n_offer_updates	-	Number of offer updates, not by bin.
17	n_updates	-	Number of orderbook updates, not by bin.

Table A.1: Complete list of all features. Bin features are flattened in ascending order to create the feature vector of size $d = 76$ at each time step.

Appendix B

Results

B.1 Hyperparameters

Parameter	RNN	LSTM
Recurrent Dropout	0.0	0.0
Dense Output Layer Dropout	0.1	0.2
Hidden units	64	128
Batch Size	128	32

Table B.1: Hyperparams for the baseline models found by Keras Tuner.

Parameter	CT
Batch Size	64
Head Size	64
Number of Heads	2
Number of MHA layers	2
Dense Hidden (Encoder) Dimension	128
Dense Hidden (Decoder) Dimension	64
Encoder Dropout	0.0
Decoder Dropout	0.2

Table B.2: Hyperparams for the CT model found by Keras Tuner.

B.2 Prediction variable distribution

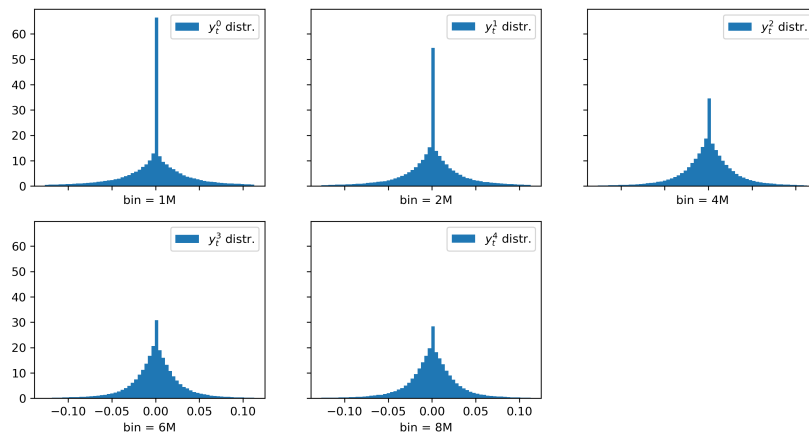


Figure B.1: The distribution of the prediction variable returns y_t^l for each bin $l \in (1, 2, 4, 6, 8)$.

B.3 Prediction Error Distribution

B.3.1 Linear y-axis

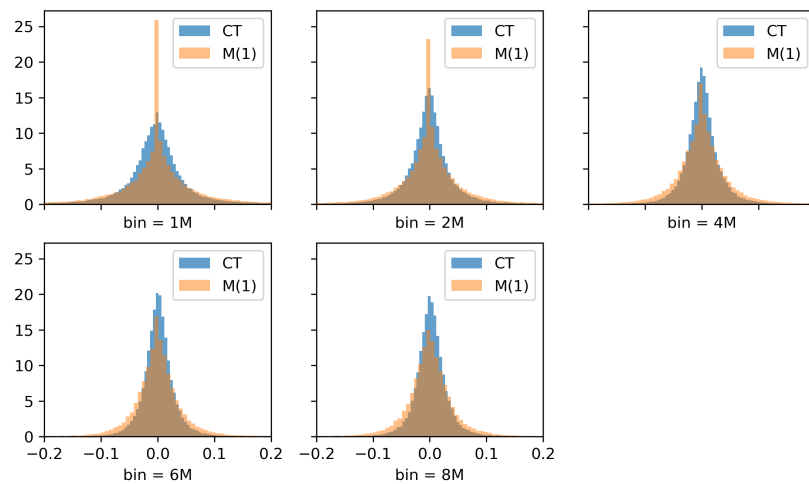


Figure B.2: The CT model error distribution compared to a simple one step martingale. The y-axis is a log scale.

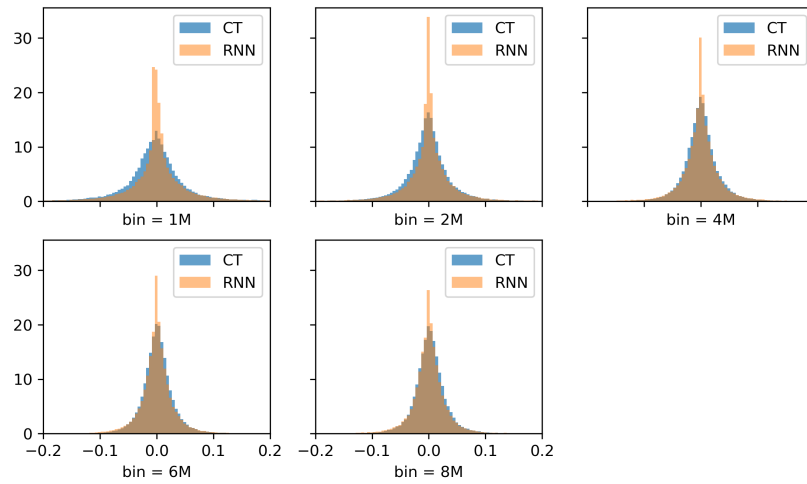


Figure B.3: The CT model error distribution compared to the RNN model.

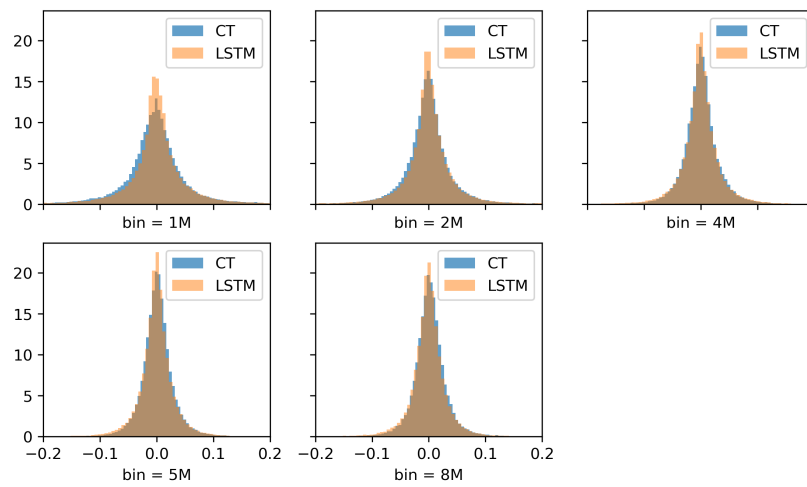


Figure B.4: The CT model error distribution compared to the LSTM model.

B.3.2 Logarithmic y-axis

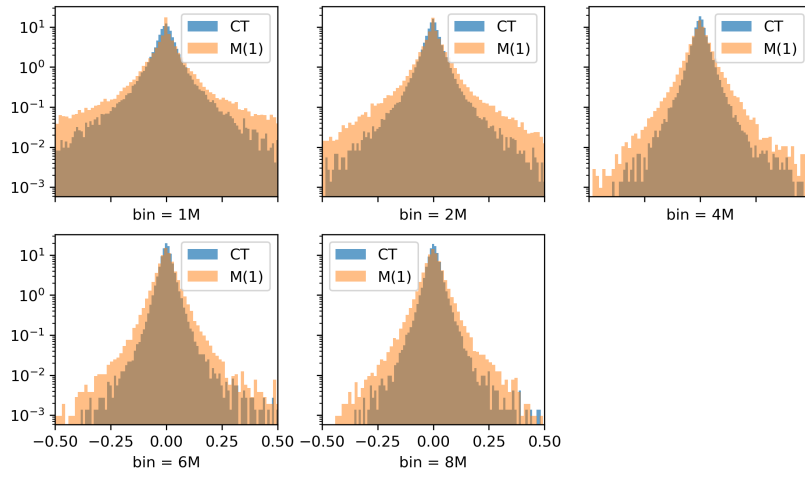


Figure B.5: The CT model error distribution compared to a simple one step martingale. The y-axis is a log scale.

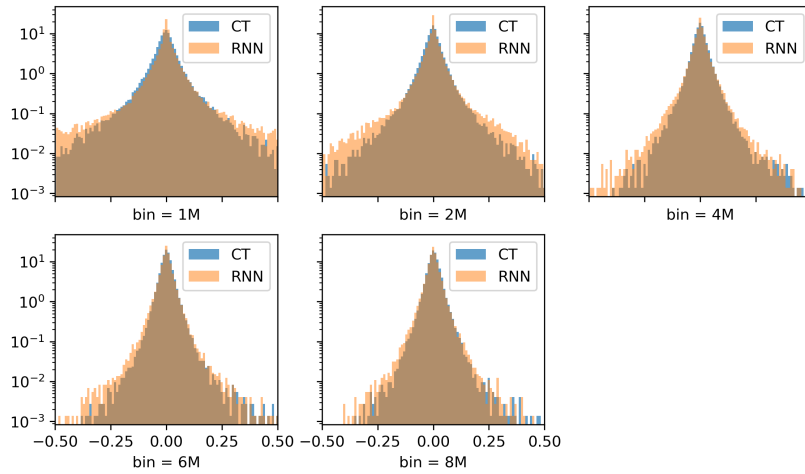


Figure B.6: The CT model error distribution compared to the RNN model. The y-axis is a log scale.

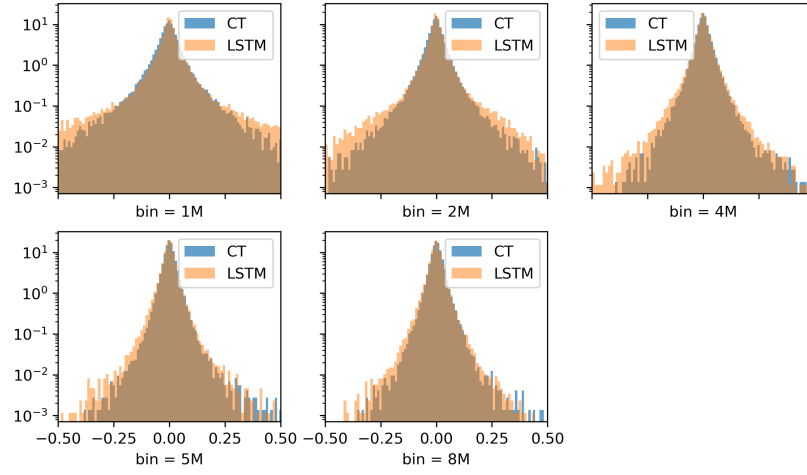


Figure B.7: The CT model error distribution compared to the LSTM model. The y-axis is a log scale..

B.4 Confusion Matrices

CT		Predicted Labels		
		-1	0	1
bin	True Labels			
1	-1	0.592	0.094	0.314
	0	0.480	0.123	0.397
	1	0.371	0.097	0.532
2	-1	0.553	0.139	0.308
	0	0.459	0.180	0.362
	1	0.339	0.138	0.524
4	-1	0.504	0.191	0.304
	0	0.385	0.230	0.385
	1	0.286	0.179	0.535
6	-1	0.462	0.217	0.321
	0	0.328	0.247	0.426
	1	0.255	0.206	0.538
8	-1	0.452	0.197	0.350
	0	0.322	0.220	0.458
	1	0.240	0.183	0.577

Table B.3: CT Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.

RNN		Predicted Labels		
		-1	0	1
bin	True Labels			
1	-1	0.316	0.373	0.310
	0	0.384	0.350	0.266
	1	0.285	0.371	0.344
2	-1	0.153	0.635	0.212
	0	0.208	0.639	0.153
	1	0.133	0.628	0.239
4	-1	0.034	0.882	0.084
	0	0.039	0.887	0.074
	1	0.031	0.867	0.103
6	-1	0.021	0.935	0.044
	0	0.026	0.937	0.037
	1	0.017	0.938	0.045
8	-1	0.016	0.955	0.029
	0	0.015	0.961	0.025
	1	0.013	0.957	0.030

Table B.4: RNN Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.

LSTM		Prediction Labels		
		-1	0	1
bin	True Labels			
1	-1	0.393	0.184	0.423
	0	0.446	0.184	0.369
	1	0.363	0.184	0.453
2	-1	0.389	0.249	0.362
	0	0.441	0.259	0.300
	1	0.358	0.249	0.392
4	-1	0.339	0.368	0.293
	0	0.350	0.374	0.276
	1	0.316	0.373	0.311
6	-1	0.312	0.424	0.264
	0	0.299	0.432	0.269
	1	0.292	0.434	0.274
8	-1	0.268	0.479	0.253
	0	0.263	0.480	0.257
	1	0.262	0.479	0.259

Table B.5: LSTM Confusion Matrices: The predicted values mapped to direction labels. The matrices are normalized on the true labels and are performed on all volume bins.

TRITA-SCI-GRU 2024-469
Stockholm, Sweden 2024

www.kth.se