THE SHORT-TERM PREDICTABILITY OF RETURNS IN ORDER BOOK MARKETS: A DEEP LEARNING PERSPECTIVE

Lorenzo Lucchese[*]
Department of Mathematics
Imperial College London
lorenzo.lucchese17@imperial.ac.uk

Mikko S. Pakkanen
Department of Statistics and Actuarial Science
University of Waterloo
and
Department of Mathematics
Imperial College London
mikko.pakkanen@uwaterloo.ca

Almut E. D. Veraart
Department of Mathematics
Imperial College London
a.veraart@imperial.ac.uk

November 29, 2022

ABSTRACT

In this paper, we conduct a systematic large-scale analysis of order book-driven predictability in high-frequency returns by leveraging deep learning techniques. First, we introduce a new and robust representation of the order book, the volume representation. Next, we carry out an extensive empirical experiment to address various questions regarding predictability. We investigate if and how far ahead there is predictability, the importance of a robust data representation, the advantages of multi-horizon modeling, and the presence of universal trading patterns. We use model confidence sets, which provide a formalized statistical inference framework particularly well suited to answer these questions. Our findings show that at high frequencies predictability in mid-price returns is not just present, but ubiquitous. The performance of the deep learning models is strongly dependent on the choice of order book representation, and in this respect, the volume representation appears to have multiple practical advantages.

1 Introduction

1.1 Financial markets and exchanges: the rise of High-Frequency traders

A financial market is an ensemble of market agents willing to buy or sell a certain financial security, such as a stock, bond, or derivative. Today most trades take place on electronic exchanges, virtual places that bring together buyers and sellers, facilitating the occurrence of transactions. At the time of writing, the two largest U.S. equity exchanges by market capitalization are the NYSE, a hybrid (floor and electronic) auction market accounting for 20% of the U.S. equities market transactions, and the Nasdaq, a fully electronic dealer market executing about 16% of such trades (source: Cboe Exchange, Inc.). Both of these markets allow traders to access live order book information, i.e. the collection of all standing orders for a given security. In theory, this allows for symmetric information across traders, which should all have access to the same data regarding market depth, liquidity, and price discovery dynamics.

In practice, traders have access to different technology, receiving market data and submitting orders at different latencies. In the quest to exploit the advantages gained by faster access to markets, almost two decades ago, a new market

participant emerged. These market players are today known as High-Frequency Traders (HFTs) and, over the years, have rapidly grown to represent a significant share of the market [Abergel et al., 2014]. Their role has since been the object of a fierce debate: their critics claim HFTs engage in predatory – and sometimes illegal – behavior, while their supporters believe HFTs to be overall beneficial to the market by providing liquidity, reducing spreads, and helping price discovery dynamics. In this paper, we will not delve into questions regarding the legitimacy of HFT practices, but we will instead aim to independently analyze the value of faster access to order book information. Over the past few decades, HFT companies have engaged in a fierce race to zero latency, making vast economic efforts to reduce their latency by just a few microseconds. What we aim to explore in this research is one of the possible reasons why such a race happened in the first place. Specifically, we will be analyzing the predictive value of order book data, i.e. to what extent can a trader with immediate access to the order book predict the future direction of the market?

Remark 1.1. In general, over the past couple of decades, the trading process has become increasingly complex due to market fragmentation, availability of new technologies such as smart order routers (SORs), and – sometimes controversial – regulation, e.g. RegNMS [US Securities and Exchange Commission, 2005] and MiFID [European Parliament and Council, 2004]. While practices that exploit arbitrage between competing trading venues exist, in this research, we will assume the trader has access to a single electronic order book-based exchange, namely the Nasdaq.

1.2 Order book predictability: asking the right questions

As discussed in the opening paragraph, we would like to explore whether, contrary to low-frequency returns, ultra-high-frequency returns tend to display predictability. Empirical studies [Sirignano and Cont, 2019] have shown that price formation dynamics, i.e. next mid-price moves, are predictable. In this paper, we will try to understand whether such predictability persists at longer horizons. Intuitively, predictability in high-frequency returns may be understood to arise simply from the way an order book market is structured, i.e. the side of the order book with less liquidity is more likely to erode faster, resulting in a price increase/decrease or as the product of recurring trading patterns in response to liquidity information. The approaches considered in the literature for forecasting high-frequency returns from order book data can be roughly divided into two categories: relatively simple models built on carefully handcrafted features [Aït-Sahalia et al., 2022] and more sophisticated architectures applied directly to raw order book data [Zhang et al., 2019, Zhang and Zohren, 2021, Kolm et al., 2021]. In this research, we will focus on the latter class of models, leveraging the ability of deep learning techniques to learn complex dependence structures. We will consider a specific class of deep learning models, introduced by Zhang et al. [2019], designed to extract features from order book data. There is empirical evidence [Bengio et al., 2013] which suggests that, although deep learning models can extract complex features, the way data is represented may have a significant impact on model performance. We will hence explore how model performance varies when changing the way the order book data is arranged. Equipped with this class of models, we will aim to investigate the following questions, which naturally arise from our preceding discussion:

1. Do high-frequency returns display order book-driven predictability? If so, how far ahead can we predict?
2. Which order book representations perform best?
3. Can we use a single model across multiple horizons?
4. Can we use a single model across multiple stocks?

We aim to answer all these questions in a formalized statistical inference framework based on model confidence sets [Hansen et al., 2011].

1.3 Related work and contributions

The work presented in this paper can be seen as contributing to two parallel research streams. On one hand, we expand on the deep learning ideas discussed in Zhang et al. [2019], Zhang and Zohren [2021], Kolm et al. [2021] by introducing new data representations and carrying out a disciplined comparison between the specifications. On the other, we explore a set of questions related to short-term price predictability in a similar spirit to Aït-Sahalia et al. [2022] but under a different class of models.

In the broader context of the questions addressed in this paper, related works are those of Sirignano and Cont [2019], exploring the universality of order book dynamics, and Wu et al. [2021], advocating for robust representations of order books. We base our experimental procedure on model confidence sets, introduced by Hansen et al. [2011]. We believe this formalized statistical inference framework perfectly suits our aim of addressing questions that require comparisons between multiple models and benchmarks.

The two main contributions of this paper are summarized as follows. First, we introduce a deep learning model for mid-price forecasting based on a more robust representation of the order book, which we will refer to as deepVOL. This

representation allows us to easily adapt the model to the setting where more granular L3 data[2] is available. Second, we provide new empirical results addressing essential questions regarding short-term price predictability in a disciplined experimental framework.

2   The space of models under consideration

2.1   The order book

At a given point in time, an order book contains all the (visible) buy and sell orders placed for a given security on a specific exchange. The lowest ask price, resp. the highest bid price, is known as the first ask order book level, resp. the first bid order book level. Subsequent levels are defined accordingly. An example of a 10-level order book snapshot is displayed in Figure 1. We note that in electronic exchanges, orders can be submitted on an evenly spaced discrete set of prices, known as ticks. The smallest price increment is known as the tick size, for most U.S. traded stocks, it corresponds to $0.01. It is important to note that not all tick prices may have standing orders; therefore, the first 10-level ask/bid prices may not coincide with the first 10 ask/bid tick prices.
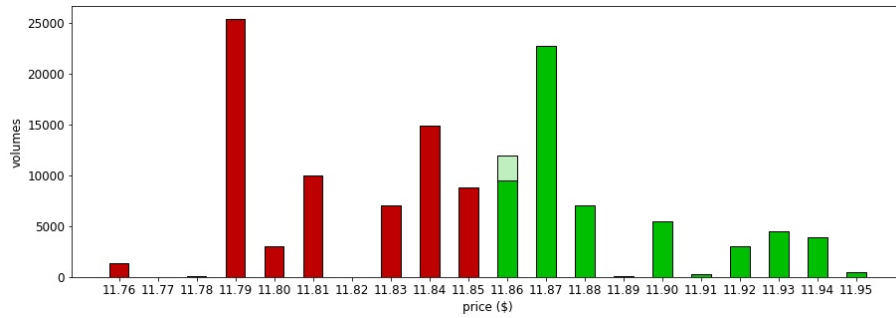

Figure 1: A sample snapshot of an order book. Ask (resp. bid) volumes are denoted by red (resp. green) bars. The lighter green shaded area represents the change in the order book shape when some of the liquidity at the best bid price is removed.

There are three main types of actions that traders can request on an exchange:

- a limit order: an order to buy a given quantity of the security at a given price;
- a market order: an order to buy a given quantity of the security at the best available price;
- a cancellation or partial deletion: an order to fully or partially delete a standing limit order.

Note that while market orders are always immediately executed once posted, passive limit orders, i.e. orders which do not cross the spread, will sit in the order book until they are matched. A limit order entering the market at a tick price where other limit orders are already present will be added at the end of the standing queue. Trades occur each time a market order or aggressive limit order is posted, the requested volume is matched to the standing limit orders according to price-time priority. We will not delve into the detailed characteristics of all the different order types which exist, but it suffices to point out that the three actions described above represent the fundamental drivers of the evolution of order books. For example, in Figure 1, the change in order book shape is due to either a (partial) deletion of a first-level buy limit order or the execution of a sell order (either a market order or an aggressive limit order).

During trading hours, electronic exchanges operate continuously, and for this reason, order books are sometimes referred to as continuous books. Exchanges such as the Nasdaq time stamp each event at nanosecond precision. Between different stocks, the level of trading activity may vary significantly, and the time elapsed between consecutive events may differ by orders of magnitude. For this reason, we define an alternative time clock: a discrete order book clock, which increments by one each time an action is executed on the order book. Throughout our discussion, we will explore questions of predictability with respect to this order book clock, which is the same as the one considered in Ntakaris et al. [2018]. Some authors, for example, Aït-Sahalia et al. [2022] consider alternative order book-driven time clocks, such as transaction clocks and volume clocks.

We understand that models based on order book-specific clocks might be challenging to use in practical trading applications. However, we believe order book-based clocks provide a more natural measure of time for exploring

---

[2]See Section 2.1 for the definitions of L1, L2 and L3 order book data.

predictability and are more suitable for comparing results across stocks than physical time: a 100ms time horizon has a significantly different meaning for stocks with different levels of trading activity.

Another important observation is that order book data might not be accessed by all market participants at the same level of granularity. The Nasdaq Quotation Dissemination Service makes the following distinctions:

- L1 data: the best bid and ask prices and corresponding volumes;
- L2 data: all available bid and ask prices and corresponding volumes;
- L3 data: all available bid and ask prices and corresponding volumes split among the orders in the queue.

We will be comparing the performance of models when different levels of data granularity are available. Figure 2 provides a visual comparison of L1, L2, and L3 data.
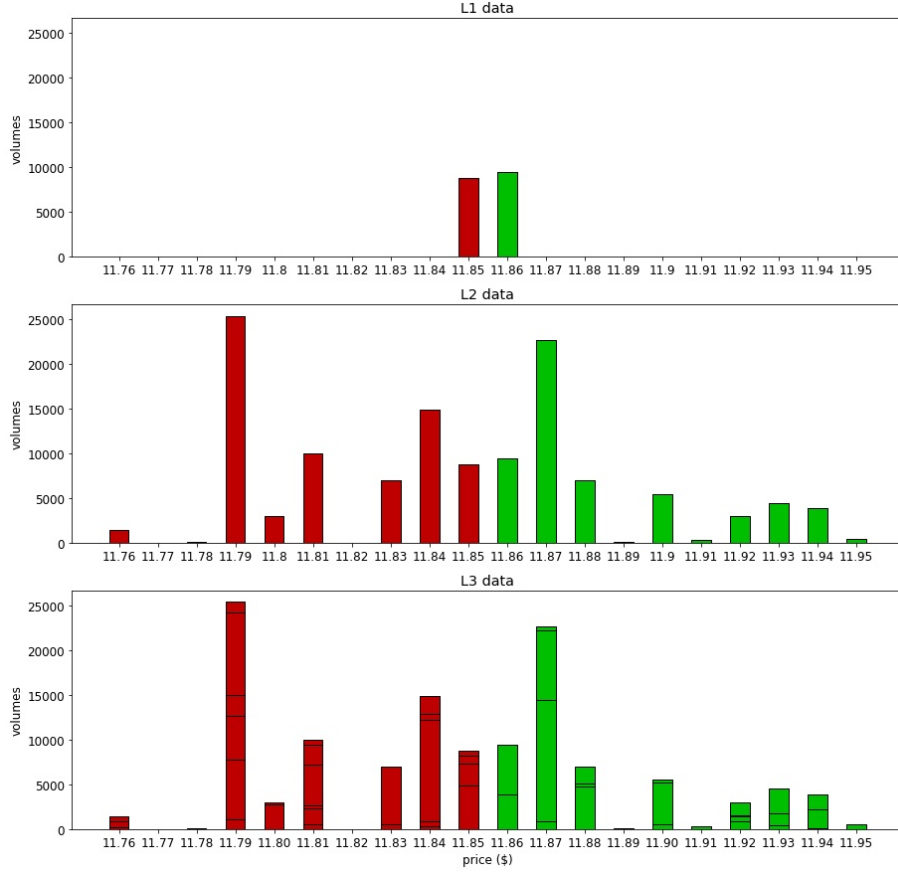
Figure 2: L1, L2 and L3 representations of the order book in Figure 1.

We introduce the following notation: at order book time $t \in \mathbb{Z}$ we denote by

- $p_{x,t}^{(l)}, v_{x,t}^{(l)}$ for $l = 1, 2, \ldots$ the l-th level ask/bid price and volume, for $x \in \{a, b\}$;

- $m_t = \dfrac{p_{b,t}^{(1)} + p_{a,t}^{(1)}}{2}$ the mid-price;

- $\pi_{x,t}^{(j)}, s_{x,t}^{(j)}$ for $j = 1, 2, \ldots$ the j-th ask/bid tick price from the mid and corresponding volume for $x \in \{a, b\}$, i.e. this is defined recursively as

$$\pi_{x,t}^{(1)} = \begin{cases} m_t & \text{if } m_t \in \mathcal{T}, \\ m_t \pm \frac{\vartheta}{2} & \text{if } m_t \notin \mathcal{T}, \end{cases}$$

$$\pi_{x,t}^{(j)} = \pi_{x,t}^{(j-1)} \pm \vartheta \text{ for } j = 2, 3, \ldots,$$

where ϑ is the tick size, $\mathcal{T} = \{k\vartheta : k \in \mathbb{N}\}$ is the set of possible tick prices and $\pm$ depends on $x \in \{a, b\}$.

- $q_{x,t}^{(i,k)}$ for $k = 1, 2, \ldots$ the queue corresponding to volume $v_{x,t}^{(i)}$ or $s_{x,t}^{(i)}$ and ordered by time priority.

2.2 Deep learning predictions

2.2.1 Predictability of returns

We will explore models that aim to identify short-term predictability in returns. We first introduce the familiar regression framework for return predictions. We then rephrase the task in terms of a classification problem, extending the definition of predictability to this setting.

We will be exploring predictability arising from past order book data and thus we define the information σ-algebra $\mathcal{F}_t$ to be

$$\mathcal{F}_t = \sigma(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

where $\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}$ are order book derived features at times $t, \ldots, t - T + 1$ for some look-back window of length T.

Predictions in the regression framework Let us first consider the regression setting. At time t we denote by $r_{t+h} \in \mathbb{R}$ the h-step ahead mid-price return, as defined in A.2. Given order book information at time t, $\mathcal{F}_t$, there exists a measurable function g such that

$$\mathbb{E}[r_{t+h}|\mathcal{F}_t] = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

or, equivalently,

$$r_{t+h} = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}) + \epsilon_t,$$

for some mean-zero noise variable ϵ_t orthogonal to the space of $\mathcal{F}_t$-measurable random variables. A prediction is defined to be any $\mathcal{F}_t$-measurable random variable, the best prediction under the model is the $\mathcal{F}_t$-measurable random variable $r_{t+h}^{(t)}$ which minimizes the expected cost

$$\mathbb{E}[C(r_{t+h}^{(t)}, r_{t+h})],$$

for some appropriate cost function $C : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. In the case of $C(r_1, r_2) = (r_1 - r_2)^2$, we have that

$$r_{t+h}^{(t)} = \mathbb{E}[r_{t+h}|\mathcal{F}_t] = g(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}).$$

Given a (probabilistic) model and an observed data set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}, r_{t+h})\}_{t \in \mathcal{I}_{\text{train}}}$ one can learn a function $g_{\hat{\theta}}$ approximating g and produce the return predictions

$$\hat{r}_{t+h}^{(t)} = g_{\hat{\theta}}(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

for test data points $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}, r_{t+h})\}_{t \in \mathcal{I}_{\text{test}}}$. Assuming returns to be stationary, we say that there is order book-driven predictability if the learned predictions outperform a benchmark prediction on the testing set with respect to the chosen cost function $C(\cdot, \cdot)$.

Remark 2.1. Different choices of cost function are possible, for example one may choose $C(r_1, r_2) = |r_1 - r_2|$ in which case the best prediction is given by the conditional median of r_{t+h} given $\mathcal{F}_t$.

Predictions in the classification framework In this paper, we will discretize the space of returns by grouping mid-price movements as downward, no-change, and upward[3]. We hence introduce the discretized return random variable

$$c_{t+h} = \begin{cases} \downarrow & \text{if } r_{t+h} \in (-\infty, -\alpha), \\ = & \text{if } r_{t+h} \in [-\alpha, +\alpha], \\ \uparrow & \text{if } r_{t+h} \in (+\alpha, +\infty), \end{cases}$$

for an appropriate choice of $\alpha > 0$. Instead of modeling only the expected conditional return, in the classification setting, one aims to approximate the whole conditional distribution, i.e. find measurable functions $p_\downarrow, p_=, p_\uparrow$ such that

$$\mathbb{P}(c_{t+h} = * | \mathcal{F}_t) = p_*(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

---

[3]We believe that, for high-frequency returns, a classification framework is more natural: mid-prices move in (half)-tick increments, and hence the corresponding returns live on a discrete lattice (see Appendix A.2). Moreover, grouping returns in three classes (downward, no-change, and upward) significantly reduces the effects of idiosyncratic noise while preserving the ability to investigate questions regarding predictability.

for $* \in \{\downarrow, =, \uparrow\}$. The discretized return prediction for an unobserved sample is then given by the minimizer of the expected misclassification cost:

$$\hat{c}_{t+h}^{(t)} = \operatorname*{argmin}_{* \in \{\downarrow, =, \uparrow\}} \sum_{\star \in \{\downarrow, =, \uparrow\}} c_{*|\star} p_{\star, \hat{\theta}}(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

where the conditional probabilities $p_{\downarrow, \hat{\theta}}, p_{=, \hat{\theta}}, p_{\uparrow, \hat{\theta}}$ are learnt from a training data set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}, c_{t+h})\}_{t \in \mathcal{I}_{\text{train}}}$ and $c_{*|\star}$ is the misclassification cost of a $*$ observation classified as a $\star$. Assuming the return process is stationary, we say that there is order book-driven predictability if the learned predictions outperform a benchmark prediction on a test set in terms of total misclassification cost.

Specifying an appropriate cost function is task-dependent. For example, a trader using the predictions as trading signals might be more interested in the correctness of up and down predictions than no-change ones. On the other hand, a market maker might prioritize the correctness of no-change predictions when using them to decide whether to tighten posted spreads. In both cases the consequences of different types of errors are asymmetric and heavily impact the choice of the cost matrix $C = \{c_{*|\star}\}_{*, \star \in \{\downarrow, =, \uparrow\}}$. In the classification framework an alternative approach is to compare the predicted conditional distributions, $\{p_*(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1})\}_{* \in \{\downarrow, =, \uparrow\}}$ to the realized outcomes directly. This approach does not require specifying a cost matrix C nor a return prediction. The predicted conditional distribution (the output of our model) is directly compared with the observed data via a suitable "distance" on the space of probability measures $\mathcal{P}(\{\downarrow, =, \uparrow\})$, where realized returns are encoded as delta functions. A natural choice of such "distance" is given by the categorical cross-entropy, as this can be interpreted (under appropriate assumptions) as the (log)-likelihood of the test set:

$$-\log \mathbb{P}(\mathcal{D}_{\text{test}}|\theta) \propto -\frac{1}{|\mathcal{I}_{\text{test}}|} \sum_{t \in \mathcal{I}_{\text{test}}} \sum_{* \in \{\downarrow, =, \uparrow\}} \mathbb{1}_{\{c_{t+h}=*\}} \log p_{*, \hat{\theta}}(\mathbf{x}_t, \ldots, \mathbf{x}_{t-T+1}),$$

where the model parameters $\hat{\theta}$ are learnt from the training set $\mathcal{D}_{\text{train}}$. As this paper does not target a specific trading strategy but is more concerned with general predictability questions, we will be evaluating our models based on the categorical cross-entropy loss. In the following, we will thus say that there is order book-driven predictability if the learned conditional distributions outperform a benchmark conditional distribution on a test set relative to categorical cross-entropy loss. As we will see in Section 2.2.2, the natural choice for the loss used in training will also be the categorical cross-entropy.

Note that in both the regression and classification framework, we compare the learned predictions with those obtained from a benchmark model to determine whether there is predictability by using a score/cost function. Simply comparing scores does not provide a sound statistical argument answering the question of whether there is predictability: the difference in score may simply be due to statistical variability. This is where the model confidence set [Hansen et al., 2011] procedure comes in. As we will discuss in Section 3 this provides a statistical testing framework to determine whether the learned models statistically outperform the benchmark predictions, i.e. whether there is predictability.

Remark 2.2. It is clear that the definition of predictability is intrinsically tied to that of benchmark prediction. Different choices for benchmark models are possible, for example, in the regression framework, a natural choice is given by (a version of) the efficient market hypothesis (EMH) [Fama, 1970]. In this case, the benchmark hypothesis assumes the conditional expected return to be 0. Under the classification framework, the simple EMH does not translate into a clear-cut model for the conditional distribution of returns. In this setting, we consider as a natural benchmark hypothesis a slightly stronger version of the EMH: under the benchmark, returns are assumed to be IID and independent of any information up to time t. The benchmark conditional distribution will therefore be given by the empirical distribution of the training set. As a side remark, we believe it is important to note the EMH was originally proposed in a very different market environment and at significantly higher latencies than the ones considered in this paper. Nevertheless, we believe the EMH provides a natural benchmark hypothesis when testing for predictability.

As mentioned in Section 1.2, there are two main approaches considered in the literature for exploring the short-term predictability of returns in order book markets: the first considers carefully handcrafted features $x_t, \ldots, x_{t-T}$ and relatively simple specifications for the prediction functions g or $(p_{\downarrow}, p_{=}, p_{\uparrow})$, i.e. linear, decision tree, etc, the second – which we will explore in this paper – uses raw order book features $x_t, \ldots, x_{t-T}$ as input to deep neural network architectures as prediction functions. We will assume the reader is familiar with deep learning techniques and thus give a brief overview of the relevant concepts in the next Section, more detailed expositions can be found in Goodfellow et al. [2016].

2.2.2 Deep Learning

The main idea behind deep neural networks is to lift the input into a higher dimensional space to extract relevant latent features before projecting it into the output space. Mathematically a neural network consists of a composition of

functions, known as layers, $f_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ for $l = 1, \ldots, N$ where d_0 is the dimensionality of input space and d_L is the dimensionality of the output space, for example $d_L = 3$ in our three-class classification task. A neural network is considered deep if the number of layers L is large.

In the simplest case the layers f_l's are "activated" affine transformations, i.e. for input $z_{l-1} \in \mathbb{R}^{d_{l-1}}$

$$f_l(z_{l-1}) = \sigma_l(W_l z_{l-1} + b_l),$$

where $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}$ are learnable weights and σ_l is a non-linear activation function applied element-wise. Neural networks of the form $f = f_L \circ \cdots \circ f_1$ such that each f_l is of this form have been shown to be universal for the class of continuous functions [Cybenko, 1989, Hornik et al., 1990], i.e. any continuous function (on a bounded domain) can be approximated arbitrarily well by a large enough network in the supremum norm[4]. Related works have shown the representational benefits of depth: there are functions that deep networks can construct with polynomially many parameters, which instead require exponentially many parameters when considering shallow networks [Telgarsky, 2015]. More complex and domain-specific forms of layers f_l exist; we briefly introduce convolutional and recurrent/LSTM layers.

Convolutional layers. Convolutional layers are a natural choice when the input is an image, i.e.

$$z_{l-1} \in \mathbb{R}^{d_{l-1}} = \mathbb{R}^{h_{l-1} \times w_{l-1} \times c_{l-1}},$$

where h_{l-1} is the height in pixels, w_{l-1} is the width in pixels and c_{l-1} is the number of channels, for example $c_{l-1} = 3$ if the image is in RGB. A convolutional layer is a specific case of a generic dense layer with parameter restrictions to account for adjacencies in the input's structure. It consists of k_l weight kernels, also known as filters, which are convoluted with the input image to produce the output. Mathematically a 2-dimensional convolution with k_l $(n_l \times m_l)$ filters and stride $(s_l \times t_l)$ is described by

$$[f_l(z_{l-1})]_{i,j,k} = \sum_{n=1}^{n_l} \sum_{m=1}^{m_l} \sum_{c=1}^{c_{l-1}} \left\{ [W_l^{(k)}]_{n,m,c} [z_{l-1}]_{s_l(i-1)+n, t_l(j-1)+m, c} + b_l^{(k)} \right\},$$

where $W_l^{(k)} \in \mathbb{R}^{m_l \times n_l \times c_{l-1}}$, for $k = 1, \ldots, k_l$, are the k_l filters, $b_l^{(k)} \in \mathbb{R}$ are bias terms and $[\cdot]_{i,j,k}$ denotes the i, j, k-th entry of a three dimensional tensor. The output then lives in $\mathbb{R}^{h_l \times w_l \times k_l}$, where $h_l = (h_{l-1} - n_l)/s_l + 1$ and $w_l = (w_{l-1} - m_l)/t_l + 1$, assuming that $s_l | (h_{l-1} - n_l)$ and $t_l | (w_{l-1} - m_l)$. While this equation might look a bit daunting, convolutional layers are often easily understood via a graphical depiction, as in Figure 3. Many empirical studies [Krizhevsky et al., 2012, He et al., 2016] have investigated the ability of convolutional layers to extract relevant features from input with grid-like topologies. In Section 2.2, we will discuss how this ability might be leveraged to extract relevant features from order books.
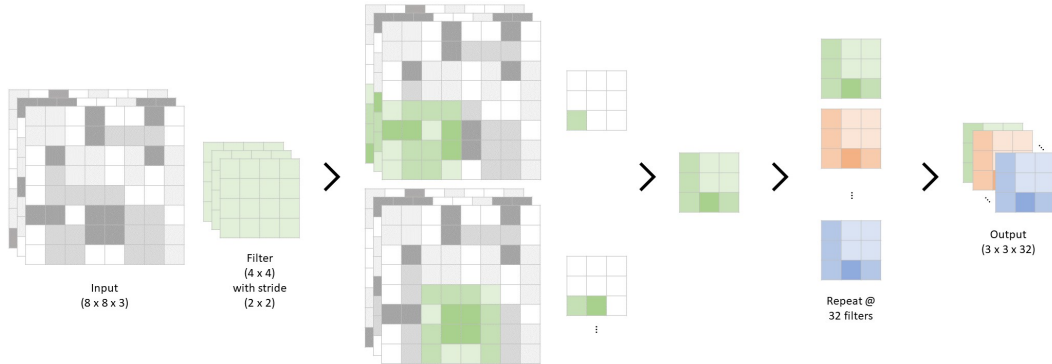
Figure 3: Convolutional layer, input $z_{l-1} \in \mathbb{R}^{8 \times 8 \times 3}$, filter size $(n_l \times m_l) = (4 \times 4)$, stride $(s_l \times t_l) = (2 \times 2)$ and number of filters $k_l = 32$.

Recurrent/LSTM layers. When the input instead has a temporal structure, a natural choice of layer is a recurrent layer. This type of layer retains information over time, discovering temporal dependencies in the data. Recurrent layers

---

[4]It follows directly that the class of neural networks is dense in $L^p(\mathcal{X})$ space, for bounded $\mathcal{X} \subseteq \mathbb{R}^d$.

may be used with streaming data to obtain online predictions or applied to a whole time series yielding a single output. We focus on the latter case, i.e. assuming the input $z_{l-1} \in \mathbb{R}^{d_{l-1}}$ has a temporal structure of the form

$$z_{l-1} = \left(z_{l-1}^{(1)}, \ldots, z_{l-1}^{(T_{l-1})} \right)^T \in \mathbb{R}^{T_{l-1} \times n_{l-1}},$$

the recurrent layer $z_l = f_l(z_{l-1})$ is given by

$$h_{l-1}^{(0)} \in \mathbb{R}^{m_{l-1}}, \quad h_{l-1}^{(t)} = \phi_l \left(h_{l-1}^{(t-1)}, z_{l-1}^{(t)} \right) \text{ for } t = 1, \ldots, T_{l-1}, \quad z_l = h_{l-1}^{(T_{l-1})},$$

where $\phi_l : \mathbb{R}^{m_{l-1}} \times \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{m_{l-1}}$ is a parameterized recurrent function and the $h_{l-1}^{(t)}$'s are known as hidden states. The most widely used type of recurrent layer is Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. While other types of recurrent layers suffer from vanishing and exploding gradients during training, the specific structure of LSTMs largely prevents such problems [Hochreiter et al., 2001]. In an LSTM layer each hidden state h is augmented with a memory state s, and hence the recurrence becomes

$$\left(h_{l-1}^{(t)}, s_{l-1}^{(t)} \right) = \phi_l \left(h_{l-1}^{(t-1)}, s_{l-1}^{(t-1)}, z_{l-1}^{(t)} \right).$$

The equations governing the LSTM recurrence are given by

$$h_{l-1}^{(t)} = o_{l-1}^{(t)} \odot \tanh(s_{l-1}^{(t)}),$$
$$s_{l-1}^{(t)} = g_{l-1}^{(t)} \odot \sigma \left(b_l + U_l z_{l-1}^{(t)} + W_l h_{l-1}^{(t-1)} \right) + f_{l-1}^{(t)} \odot s_{l-1}^{(t-1)},$$

where the input, forget and output gates depend on the context, i.e. on $h_{l-1}^{(t-1)}, z_{l-1}^{(t)}$, via:

$$g_{l-1}^{(t)} = \sigma \left(b_l^g + U_l^g z_{l-1}^{(t)} + W_l^g h_{l-1}^{(t-1)} \right),$$
$$f_{l-1}^{(t)} = \sigma \left(b_l^f + U_l^f z_{l-1}^{(t)} + W_l^f h_{l-1}^{(t-1)} \right),$$
$$o_{l-1}^{(t)} = \sigma \left(b_l^o + U_l^o z_{l-1}^{(t)} + W_l^o h_{l-1}^{(t-1)} \right),$$

for sigmoid activation function σ, element-wise product $\odot$ and parameters $b_l, b_l^g, b_l^f, b_l^o \in \mathbb{R}^{m_{l-1}}$, $U_l, U_l^g, U_l^f, U_l^o \in \mathbb{R}^{m_{l-1} \times n_{l-1}}$ and $W_l, W_l^g, W_l^f, W_l^o \in \mathbb{R}^{m_{l-1} \times m_{l-1}}$. The network diagram of an LSTM layer is given in Figure 4. LSTMs summarize relevant context information in the memory cell $s_{l-1}^{(t)}$, performing exceptionally well on tasks with a strong temporal dependency [Melis et al., 2017].
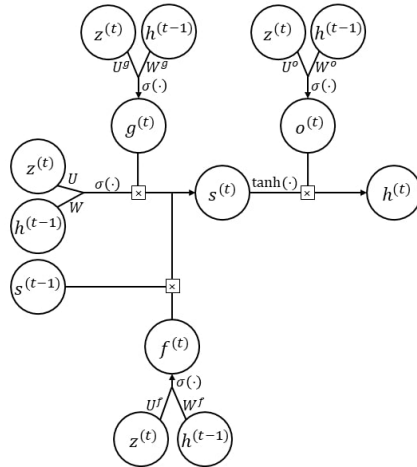

Figure 4: LSTM layer network diagram.

As discussed in Section 2.2.1, once we have specified a parametric model for approximating the prediction function $f \approx f_\theta$ we wish to learn the best approximation to f from a training set of observed input-output data $\mathcal{D}_{\text{train}} =$

$\{(x_i, y_i)\}_{i \in \mathcal{I}_{\text{train}}}$. To do so, we aim to find the parameter combination θ which yields predictions closest to the observations:

$$\hat{\theta} = \underset{\theta \in \Theta}{\text{argmin}}\, L(\theta | \mathcal{D}_{\text{train}}) = \underset{\theta \in \Theta}{\text{argmin}}\Big\{ \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \mathcal{L}\big(f_\theta(x_i), y_i\big) \Big\},$$

where $\mathcal{L}(\cdot, \cdot)$ is a loss function quantifying the distance between the prediction and the observed value, and θ contains all network parameters (weights matrices, bias terms, CNN filters...). The form of $\mathcal{L}$ may be chosen using probabilistic arguments or according to some other task-specific criteria.

When the model f_θ is a deep neural network the loss landscape $L(\theta) = L(\theta | \mathcal{D}_{\text{train}})$ may be very complex and finding a "good" minimizer is often a difficult task. The most widely used approach is to use (some variant of) gradient descent in which the model parameters are iteratively updated by

$$\theta^{(n+1)} = \theta^{(n)} - \eta \nabla_\theta L\big(\theta^{(n)}\big),$$

until a convergence criterion is met. Here $\eta > 0$ is a fixed learning rate, and the weights $\theta^{(0)}$ are chosen according to an appropriate initialization rule. When considering deep neural network architectures, the number of parameters is often very large, and therefore second-order methods are often intractable or computationally infeasible. Many variants of the first order gradient descent algorithm exist though, for example, stochastic gradient descent updates parameter values based on estimates of the gradient $\nabla_\theta L(\theta^{(n)})$ computed from a random subset (known as a batch) of the training data set. More advanced first-order optimizers consider momentum and/or use adaptive learning rates. In our experiments, we will be using the Adam optimizer [Kingma and Ba, 2015] which updates the parameters based on adaptive estimates of first and second-order moments:

$$m^{(n+1)} = \frac{1}{1 - \beta_1^n}\Big\{ \beta_1 m^{(n)} + (1 - \beta_1)\nabla_\theta L\big(\theta^{(n)}\big) \Big\},$$

$$v^{(n+1)} = \frac{1}{1 - \beta_2^n}\Big\{ \beta_2 v^{(n)} + (1 - \beta_2)\big[\nabla_\theta L\big(\theta^{(n)}\big)\big]^2 \Big\},$$

$$\theta^{(n+1)} = \theta^{(n)} - \eta \frac{m^{(n+1)}}{\sqrt{v^{(n+1)}} + \epsilon},$$

for parameters $\eta, \beta_1, \beta_2, \epsilon > 0$.

Remark 2.3. Note that, contrary to other model specifications, such as linear models, the fitting procedure of deep neural networks is not closed form. This adds a further layer of uncertainty. Any model is subject to the following two sources of uncertainty: one due to model specification (i.e. $f \approx f_\theta$) and one due to parameter estimation uncertainty (i.e. statistical error arising from estimating θ from a finite sample). Models which do not have closed forms to get $\hat{\theta}$ from observed data are further subject to numerical optimization error in the parameter estimation phase. Sometimes, the optimization procedure may not even converge to a (global) minimum. With deep neural networks, we thus trade off smaller model uncertainty (at least theoretically due to the universality property) for larger parameter estimation error.

When the task is a classification problem f_θ is usually chosen so that it maps to $\{p \in [0, 1]^C : \|p\|_1 = 1\}$, where C is the number of classes in the set $\mathcal{C}$. This allows to give a probabilistic interpretation of the deep neural network outputs, i.e.

$$f_\theta(x) = \{p_{c,\theta}(x)\}_{c \in \mathcal{C}}, \text{ where } p_{c,\theta}(x) = \mathbb{P}(y = c | x, \theta).$$

In this case, a natural choice of loss function to use during training is the cross-entropy loss, which can be understood by maximum likelihood arguments: assuming the responses are independent given the features and the distribution of the features is independent of the parameters θ the negative log-likelihood is

$$L(\theta | \mathcal{D}_{\text{train}}) = -\log \mathbb{P}(\mathcal{D}_{\text{train}} | \theta) \propto -\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \sum_{c \in \mathcal{C}} \mathbb{1}_{\{y_i = c\}} \log p_{c,\theta}(x_i).$$

When training samples for the classes are unbalanced, say class $\bar{c} \in \mathcal{C}$ has many more samples than all other classes, the optimization algorithm tends to get stuck in the trivial minimum given by the delta function at $\bar{c} \in \mathcal{C}$. One way to mitigate this effect is to re-weight the categorical cross entropy loss such that the network is less incentivized to move towards the trivial minimum. With appropriate weighting, $\bar{c} \in \mathcal{C}$ observations will influence the direction of the gradient less, thus reducing the strength of the attraction towards the trivial minimum. Mathematically one uses the following weighted categorical cross-entropy loss

$$L(\theta | \mathcal{D}_{\text{train}}) = -\frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{i \in \mathcal{I}_{\text{train}}} \sum_{c \in \mathcal{C}} w_c \mathbb{1}_{\{y_i = c\}} \log p_{c,\theta}(x_i),$$

where the weights w_c are chosen to be inversely proportional to the class's training set density. For example, in our experiments, we will set:

$$w_c = w_c(\mathcal{D}_{\text{train}}) = \frac{|\mathcal{I}_{\text{train}}|}{\#\{c_i \in \mathcal{D}_{\text{train}} : c_i = c\}}.$$

2.3 Deep learning models for short-term return predictions in order book markets

In the previous Section, we briefly outlined how deep learning methods can be used for prediction tasks. We now discuss how such techniques can be applied to order books to understand price formation mechanisms and predict h-step ahead high-frequency returns. We will consider a specific class of deep learning models based on the deepLOB architecture [Zhang et al., 2019]. Such specification is well suited for the grid-like topology of order books.

2.3.1 deepLOB, [Zhang et al., 2019]

This network acts on raw order book input. A CNN module and an inception module feed into an LSTM layer which produces the final classification output. The CNN and inception module aim to extract short-term spatio-temporal features in the data, while the LSTM module works on longer-term dependencies.
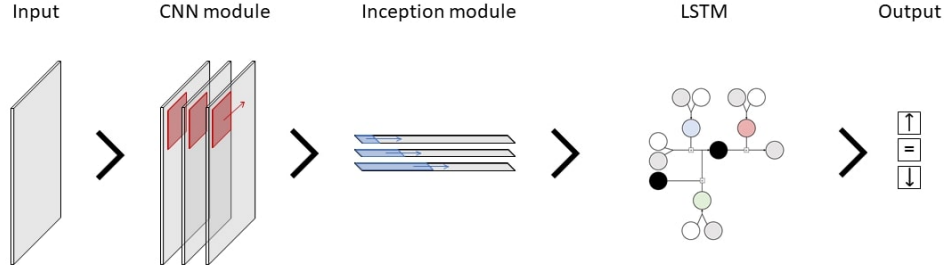
Figure 5: Core modules of network architectures.

The deepLOB architecture is summarized in Figure 5 and is made up of the following components:

- **Input**. The first L level raw order book information - price and volume - with a look-back window of length T is used as input. The input at time t is thus a $(T \times 4L)$ array given by

$$\{\mathbf{x}_{t-\tau}\}_{\tau=0,\ldots,T-1} = \{(p_{a,t-\tau}^{(l)}, v_{a,t-\tau}^{(l)}, p_{b,t-\tau}^{(l)}, v_{b,t-\tau}^{(l)})\}_{\tau=0,\ldots,T-1,l=1,\ldots,L} \in \mathbb{R}^{T \times 4L}.$$

  A feature-wise rolling window z-score standardization is applied to the input, i.e. $v_{a,t-\tau}^{(1)}$ is standardized using the mean and standard deviation of the first level ask volumes over the previous five days.

- **CNN module**. Convolutions are applied to the data in both the spatial and temporal dimensions. The spatial convolutions aim to aggregate information across order book levels, and the temporal convolutions can be understood as smoothing operations. The CNN module is summarized in Figure 6.

- **Inception module**. This module up-samples the convoluted data by applying various temporal convolutions with different filter lengths (time-window). Each temporal convolution can be interpreted as a (weighted) moving average. It is similar in spirit to approaches used in technical analysis by some day traders, but the frequencies at which this is applied are substantially different.

- **LSTM**. The Long Short-Term Memory layer takes the multidimensional time series produced by the inception module and feeds it through a recurrent network structure aimed at extracting longer-term dependencies among the data. The last hidden state of the LSTM is passed through a dense layer with a softmax activation function to produce the h-step ahead return prediction $\downarrow, =$ or $\uparrow$.

The exact details of the architecture can be found in Table 8.
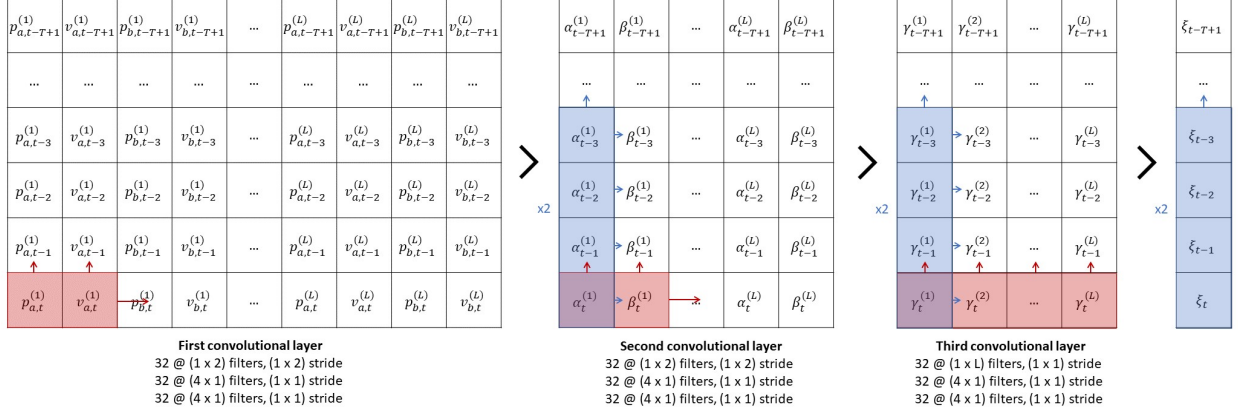
Figure 6: CNN module deepLOB.
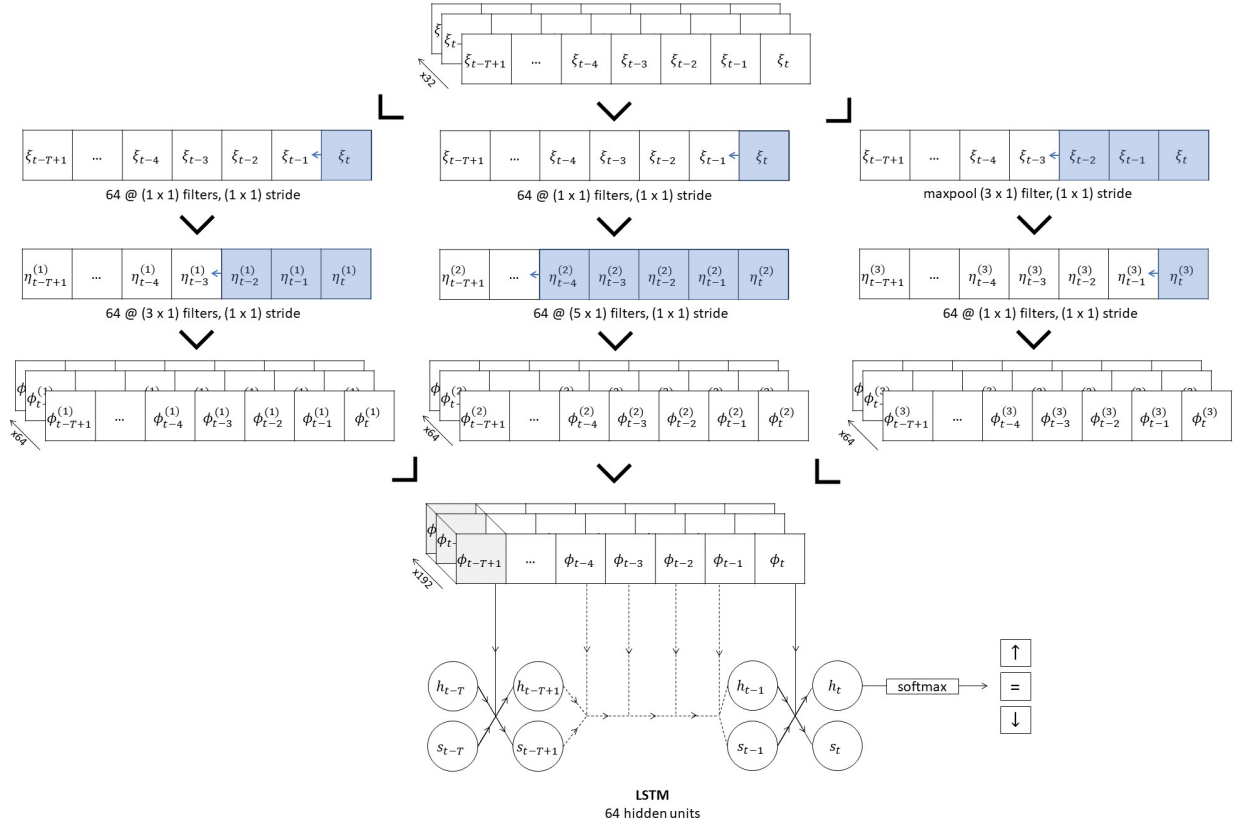
Figure 7: Inception module and LSTM layer.

2.3.2   deepOF, [Kolm et al., 2021]

In the deepLOB model order book states, which are non-stationary, are mapped to a stationary quantity, returns. While in theory, this shouldn't be a problem (due to the idea of universality in deep neural networks), the authors in Kolm et al. [2021] argue that using some form of stationary input might improve model performance by facilitating the training procedure.

A stationary order book quantity is order flow. First-level order flow describes the net flow of orders at the best bid and ask. This was introduced in Cont et al. [2013] to explore the price impact of order book events. This single quantity parsimoniously models the instantaneous effect of order book events on prices. The first-level bid and ask order flows

corresponding to the order book event occurring at time t (in order book time) are defined by:

$$bOF_t^{(1)} = \begin{cases} v_{b,t}^{(1)} & \text{if } p_{b,t}^{(1)} > p_{b,t-1}^{(1)}, \\ v_{b,t}^{(1)} - v_{b,t-1}^{(1)} & \text{if } p_{b,t}^{(1)} = p_{b,t-1}^{(1)}, \\ -v_{b,t-1}^{(1)} & \text{if } p_{b,t}^{(1)} < p_{b,t-1}^{(1)}, \end{cases} \quad \text{and} \quad aOF_t^{(1)} = \begin{cases} -v_{a,t-1}^{(1)} & \text{if } p_{a,t}^{(1)} > p_{a,t-1}^{(1)}, \\ v_{a,t}^{(1)} - v_{a,t-1}^{(1)} & \text{if } p_{a,t}^{(1)} = p_{a,t-1}^{(1)}, \\ v_{b,t}^{(1)} & \text{if } p_{a,t}^{(1)} < p_{a,t-1}^{(1)}. \end{cases}$$

The difference between the two is known as order flow imbalance:

$$OFI_t^{(1)} = bOF_t^{(1)} - aOF_t^{(1)}.$$

Bid order flow corresponds to the net change in volume at the best bid level; the three cases in the definition can be understood as:

- a new bid order being placed at a higher price than the current best bid;
- the order volume at the best bid price increasing or decreasing;
- the entire volume at the best bid price being consumed, thus decreasing the best bid price.

Note that, as discussed in Section 2.1, volumes can increase or decrease due to orders being placed, executed, or canceled. A similar description holds for ask order flow.

In Xu et al. [2019] the authors explore flows of volumes at deeper levels in the order book, by introducing multi-level order flow. The definitions are exactly as above with superscript (1) replaced by general (l). We note that in both Cont et al. [2013] and Xu et al. [2019] the authors investigate the explanatory power of (multi-level) order flow imbalance for price changes, i.e. the relationship between contemporaneous order flow imbalance and price changes.

In our setting, as in Kolm et al. [2021], we are instead interested in exploring the predictive power of order flow, i.e. the relationship between past order flow and future price changes. We will hence refer to deepOF as the deepLOB architecture with stationary order flow input, i.e. a $(T \times 2L)$ array given by

$$\{\mathbf{x}_{t-\tau}\}_{\tau=0,\ldots,T-1} = \{(aOF_{t-\tau}^{(l)}, bOF_{t-\tau}^{(l)})\}_{\tau=0,\ldots,T-1,l=1,\ldots,L} \in \mathbb{R}^{T \times 2L}.$$

The order flow input enters the CNN module in Figure 6 at the second convolutional layer. The rest of the architecture is exactly the same as deepLOB, as detailed in Table 8. We note that the first convolutions applied to the order flow input, i.e. the second convolutional layer in Figure 6, aggregate information across bid and ask order flows, essentially computing a weighted order flow imbalance.

Remark 2.4. The original deepOF specification in Kolm et al. [2021] was structured as a (multi-horizon) regression task. In their setting the last layer of the deep neural network maps each prediction to $\mathbb{R}$ instead of $[0, 1]^3$. Moreover, it is important to note that the way multi-horizon predictions were obtained in the original work is different from the encoder-decoder structure we will be using for our multi-horizon models. Another slight difference with the experiments in Kolm et al. [2021] is in the standardization procedure, instead of standardizing each feature by its training mean and standard deviation, we will be using a rolling window approach.

2.4 deepVOL

2.4.1 Motivation for robust representation

As discussed in the previous Section the main difference between deepLOB and deepOF is in the way the data is fed into the model. While deepLOB uses raw order book data as input, deepOF uses a derived quantity, order flow. In general, the success of deep learning tasks is highly dependent on the way the data is represented. The task of predicting returns in order book markets is no exception. In order to achieve the best possible results, one should adopt a robust representation of the data. Wu et al. [2021] identify five main desiderata for a robust representation of order book data:

- Region of interest: the entire order book may contain a wide range of prices, the data representation should select a region of interest based on a complexity-performance trade-off.
- Efficiency: the data representation should avoid excessive dimensionality.
- Validity: the data representation should have a simple definition of valid manipulations.
- Smoothness: the data representation should be robust to small perturbations.
- Compatibility: the data representation should be compatible with the deep learning architecture.

We note the order book representations used in deepLOB and deepOF do not conform to these desiderata. Order book states organized by 'level' do not have a simple validity (price and volume information are intrinsically tangled and would loose their significance if treated separately in a black box algorithm), are not robust to small perturbation (small orders added at empty ticks completely change the order book feature vector), and are incompatible with deepLOB's CNN module (the spatial structure is not homogeneous as there is no fixed interval between levels). It turns out that while the 'level' representation of the order book may be easily understandable by humans it is less so for computers. In addition to not satisfying the desiderata, this representation does not respect the following fundamental but implicit assumption of deep learning models: signals at the same entry of the input should come from the same source. In the 'level' representation, as new order book events happen, the same signal (i.e. a posted order) may move between levels.

We, therefore, introduce volume features, which provide a robust representation of order book data. Fixing a window of size $W > 0$ we define:

$$\{\mathbf{x}_{t-\tau}\}_{\tau=0,\ldots,T-1} = \{(s_{b,t-\tau}^{(W)}, \ldots, s_{b,t-\tau}^{(1)}, s_{a,t-\tau}^{(1)}, \ldots, s_{a,t-\tau}^{(W)})\}_{\tau=0,\ldots,T-1},$$

where $s_{x,t-\tau}^{(j)}$ for $x \in \{a, b\}$ are the bid/ask volumes corresponding to the j-th price from the mid $\pi_{x,t-\tau}^{(j)}$, as defined in Section 2.1. We note that the volume representation indeed satisfies the five desiderata: a region of interest is identified (via the window $W > 0$), it is efficient (for the same dimension of input it may convey more or less information than the 'level' representation, depending on how sparse the orders are placed in the order book), it has a simple validity (all entries are in the same units), it is robust to small perturbations (new orders at empty levels minimally affect the feature) and is compatible with the CNN architecture (the spatial structure of the volumes is homogeneous). An intuitive visualization of this representation as a one-dimensional gray-scale strip is given in Figure 8, when including a time dimension this naturally becomes a two-dimensional gray-scale image. The main drawback of the volume representation is that when orders are placed far apart in the order book it is sparse and a larger window $W > 0$ may be required.

1400	0	100	25400	3000	10000	0	7000	14930	8800	12000	22700	7000	100	5490	300	3000	4500	3900	500
1400	0	100	25400	3000	10000	0	7000	14930	8800	9484	22700	7000	100	5490	300	3000	4500	3900	500

Figure 8: Gray-scale visualization of the volume representation of the order book in Figure 1.

Remark 2.5. Our definition of volume features is similar to the mid-price-centered moving window representation of Wu et al. [2021], with the latter living in $\mathbb{R}^{2W+1}$ instead of $\mathbb{R}^{2W}$ and using $\pm$ signs to distinguish between bid and ask volumes. The need for a new, more robust representation of the order book was reached independently.

In order to use the deepLOB architecture with this new data representation, we must adapt the CNN module. In particular, we fold the volume input into a three-dimensional array

$$\{s_{x,t-\tau}^{(j)}\}_{\tau=0,\ldots,T-1,j=1,\ldots,W,x\in\{a,b\}} \in \mathbb{R}^{T \times W \times 2}$$

and feed this into a three-dimensional convolutional layer with a $(2 \times 2 \times 1)$ filter and $(1 \times 1 \times 1)$ stride. This layer aims to extract imbalances in the order book by comparing volumes on the two sides of the mid-price. The CNN module with the appropriate changes is depicted in Figure 9. The rest of the deepVOL architecture is exactly the same as deepLOB with one slight difference in the way the data is normalized. Thinking of the volume representation as a gray-scale image a natural choice of normalization is

$$\{s_{x,t-\tau}^{(j)}\}_{\tau,j,x} \mapsto \frac{1}{\max\limits_{\tau,j,x} s_{x,t-\tau}^{(j)}} \{s_{x,t-\tau}^{(j)}\}_{\tau,j,x}$$

instead of the rolling window standardization used in deepLOB and deepOF.

Remark 2.6. One could define a volume flow quantity based on the distance from the mid, in the same way order flow describes the flow of orders based on the level. Following the same motivation for considering order flow in deepOF presented in Section 2.3.2, one could consider using a volume flow quantity as input to the deep learning architectures with the desired robust representation properties.

2.4.2 Going to L3 data

All models considered so far, deepLOB, deepOF, and deepVOL use L2 data as input. If one has access to more granular data, i.e. L3 data breaking down each volume queue into single orders, one might be interested in trying to leverage this
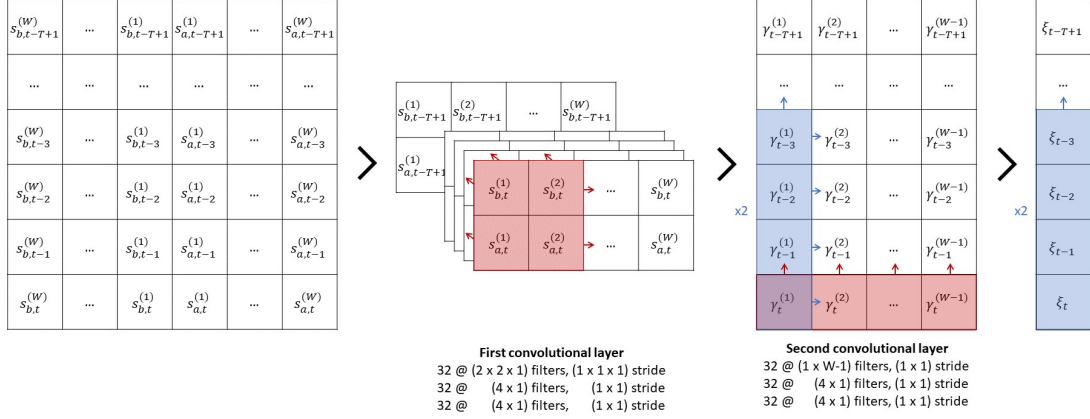
Figure 9: CNN module deepVOL.

information to obtain higher predictive performance. We thus define a natural extension of the volume representation considered in the previous Section.

Let us denote by $(q_{x,t}^{(j,1)}, q_{x,t}^{(j,2)}, \dots) \in \mathbb{R}^{\mathbb{N}}$ for $x \in \{a, b\}$ the queue at the j-th bid/ask price from the mid $\pi_{x,t}^{(j)}$ as introduced in Section 2.1. Here $q_{x,t}^{(j,k)}$ denotes the volume of the k-th order in the queue sorted by time priority and is set to zero if there is no such order. The aggregated volume at $\pi_{x,t}^{(j)}$ is given by:

$$s_{x,t}^{(j)} = \sum_{k \geq 1} q_{x,t}^{(j,k)}.$$

A natural extension of, the volume representation would therefore be to consider

$$\{(q_{x,t-\tau}^{(j,1)}, q_{x,t-\tau}^{(j,2)}, \dots)\}_{\tau=0,\dots,T-1, j=1,\dots,W, x \in \{a,b\}} \in \mathbb{R}^{T \times W \times 2 \times \mathbb{N}}.$$

Unfortunately, this is an infinite-dimensional array that cannot be directly fed into machine learning models. We, therefore, require to cut off the queue at a given depth level. In order to avoid discarding precious information, we aggregate all orders sitting past the maximum depth level at the end of the queue. For a given depth level $D > 0$, we thus consider

$$\left\{ \left(q_{x,t-\tau}^{(j,1)}, \dots, q_{x,t-\tau}^{(j,D-1)}, \sum_{k \geq D} q_{x,t-\tau}^{(j,k)} \right) \right\}_{\tau=0,\dots,T-1, j=1,\dots,W, x \in \{a,b\}} \in \mathbb{R}^{T \times W \times 2 \times D}.$$

Remark 2.7. A similar approach to the one used for cutting off the queue might be applied to cut off volumes on the bid and ask sides in level 2 representations. This would give a better idea of the remaining liquidity at deeper levels in the order book.

In order to extract relevant information from the queue, we add an initial convolutional layer, which maps each queue to a weighted sum of the order sizes. The weighted aggregated volumes are then fed into the deepVOL architecture as above. The resulting CNN module is summarized in Figure 10.

The full architectures for deepVOL and deepVOL L3 are detailed in Table 9.

2.5 Multi-horizon models

So far we considered single horizon models, i.e. order book input x_t is mapped to a three class distribution $(p_\downarrow, p_=, p_\uparrow)$ corresponding to the discretized return c_{t+h} at fixed horizon h. In the following, we consider a generalization of the architectures considered thus far to the setting where the forecasting horizon is a vector $h = (h_1, \dots, h_K)$. In this case the modelling task consists in predicting the distributions of the discretized returns $c_{t+h} = (c_{t+h_1}, \dots, c_{t+h_K})$.

The simplest way of adjusting the current models to the multi-horizon framework would be to replace the last softmax layer in Figure 7 with K parallel dense softmax layers. The last hidden state of the LSTM module would hence be mapped to an array of size $3 \times K$, corresponding to K distributions over three classes. A similar architecture – though in the regression framework – is considered in Kolm et al. [2021]. While this approach is perfectly valid it does not make use of the sequential nature of the task, potentially neglecting an important structural feature of the data.
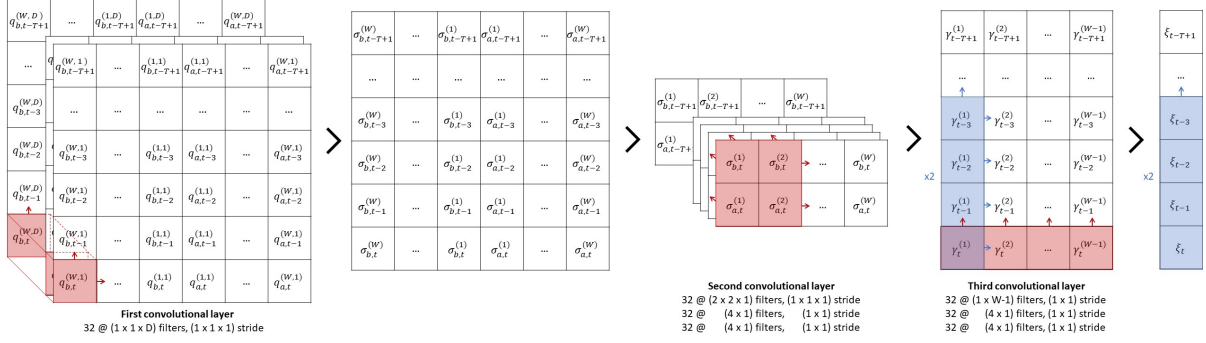
Figure 10: CNN module deepVOL L3. Abusing notation slightly we denote $\sum_{k \geq D} q_{x,t-\tau}^{(j,k)}$ by $q_{x,t-\tau}^{(j,D)}$.

In this paper, we will be leveraging architectures inspired by machine translation which are naturally suited for sequential forecasting. Specifically, we will be considering encoder-decoder models: an encoder maps the input data to a latent summary state (also known as context vector) and then a decoder rolls forward predictions sequentially. In this context let $z_{t-T+1}, \ldots, z_t$ denote the last T hidden states of an encoder at time t, then a decoder rolls forward predictions by:

$$z_0' = z_t, p_0 = \hat{p}_0,$$
$$c_k = h(z_{t-T+1}, \ldots, z_t), z_k' = f([z_{k-1}', c_k], p_{k-1}), p_k = g(z_{k-1}', c_k),$$

for $k = 1, \ldots, K$. Here $h(\cdot)$ is a function acting on the hidden states of the encoder to extract the context vector c_k (this may possibly depend on other inputs as well, such as previous hidden states of the decoder), $f(r, p)$ is a recurrent layer with recurrent input r and exogenous input p, $g(\cdot)$ is an output layer depending on both the decoder hidden state and the context. The general mechanism of such encoder-decoder architecture is visualized in Figure 11.
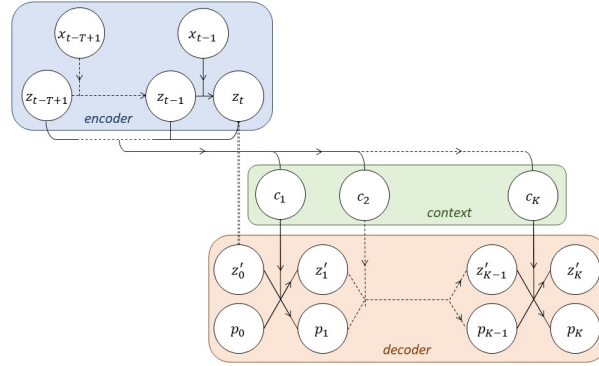
Figure 11: A general encoder-decoder architecture.

In our experiments, we will consider a sequence-to-sequence decoder [Cho et al., 2014], the simplest example of such architecture. In this setting we set $c_k \equiv z_t$, implicitly assuming that at every forecasting horizon the last hidden state of the encoder summarizes all the relevant information required to make the prediction. More complex architectures exist, for example, Luong et al. [2015] introduces an attention-based decoder that uses a weighted combination of all the hidden states of the encoder as context vector. In this setting different weights are used at different forecasting horizons, selectively accessing different hidden states of the encoder during decoding.

In the experiments we set $f(\cdot)$ to be an LSTM and $g(\cdot)$ to be a dense layer with softmax activation. Moreover p_0 is initialized at $\hat{p}_0 = (0, 1, 0)$.

Using multi-horizon forecasting was first proposed for deepLOB in Zhang and Zohren [2021]: the LSTM module of the deepLOB architecture discussed in Section 2.3.1 acts as an encoder, mapping order book states to the latent vector $z_t = [h_t, s_t]$. A seq2seq decoder then rolls forward the prediction producing distributional forecasts at horizons $h = (h_1, \ldots, h_K)$. The LSTM module with seq2seq decoder is illustrated in Figure 12. Clearly, there is nothing

stopping us from applying the same structure for multi-horizon forecasting with deepOF or deepVOL convolutional modules before the LSTM encoder.

Full details of the multi-horizon architectures can be found in Tables 8 and 9.

Remark 2.8. We adopt the same multi-horizon framework as in Zhang and Zohren [2021], where the response c_{t+h_k} at horizon $h_k \in \{h_1, \ldots, h_K\}$ corresponds to the return from time t to time $t + h_k$. Alternatively, one might wish to consider as multi-horizon responses the subsequent incremental returns $c_{(t+h_{k-1})+h_k}$, i.e. the returns between time $t+h_{k-1}$ and time $t+h_k$. By choosing evenly spaced h_k's one would obtain more consistent responses across prediction time steps $\{1, \ldots, K\}$ and possibly improve the performance of the multi-horizon models.
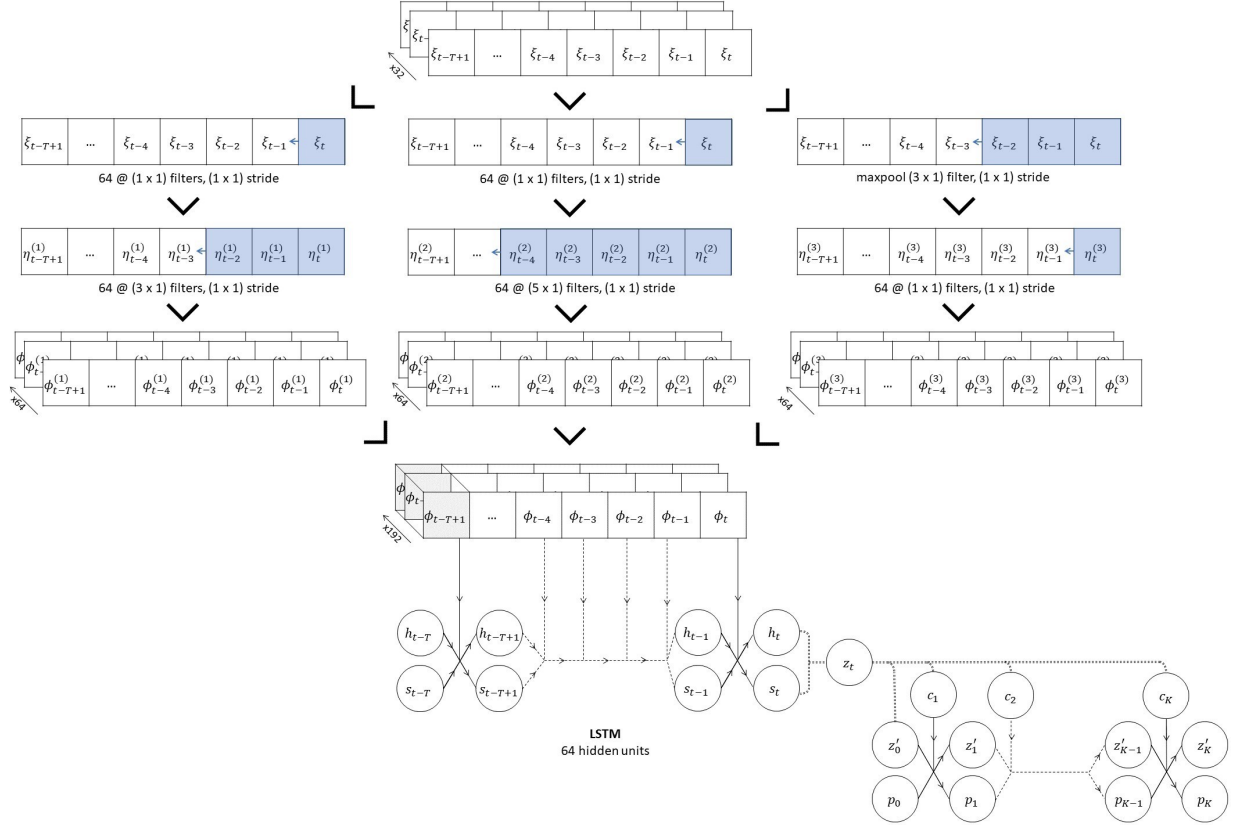
Figure 12: Inception module and LSTM layer with seq2seq decoder for multi-horizon forecasting.

3 Experiments

Model confidence set   The model confidence set procedure introduced by Hansen et al. [2011] provides a formalized statistical inference framework in which to compare competing models. It does not assume that any of the models is the true data generating process but simply aims to identify a set of models that will contain the best model with a given level of confidence, known as model confidence set, MCS. A model confidence set is, therefore, analogous to a confidence interval for a parameter.

To identify the best models, one must compare the model outputs (either as estimated class probabilities or as forecasts) by selecting an appropriate loss/score function. Since we are not considering a specific application, we will be comparing models using the cross-entropy loss, i.e. we will be evaluating the quality of the estimated probabilities. Alternatively, depending on the application, one could consider a specific cost function and modify both the training and testing procedures accordingly. We will be considering a rolling window testing scheme where the first 4 weeks are used for training and validation while the last week is used for testing. The period under consideration is then shifted forward by 5 weeks, so there are no overlapping windows. Let us denote by $\mathcal{M}_0$ the set of models under consideration. In the MCS framework for models $i, j \in \mathcal{M}_0$ and window $w \in \{1, \ldots, W\}$ we define:

$$d_{ij,w} = L_{i,w} - L_{j,w},$$

where

$$L_{i,w} = -\mathrm{cce}(\hat{\mathbf{p}}^i_{w,\text{test}}, \mathbf{c}_{w,\text{test}}) = -\frac{1}{|\mathcal{I}_{w,\text{test}}|} \sum_{t \in \mathcal{I}_{w,\text{test}}} \sum_{* \in \{\downarrow,=,\uparrow\}} \mathbb{1}_{\{c_{t,w,\text{test}}=*\}} \log \hat{p}^i_{*,t,w,\text{test}}$$

is the categorical cross-entropy loss corresponding to the estimated probabilities $\hat{\mathbf{p}}^i_{w,\text{test}}$ when compared with the observations $\mathbf{c}_{w,\text{test}}$. $\hat{\mathbf{p}}^i_{w,\text{test}}$ are the class probabilities produced by model i on the testing set $\mathcal{D}_{w,\text{test}}$ after being trained on the training set $\mathcal{D}_{w,\text{train}}$. The set of best models among those in $\mathcal{M}_0$ is defined as

$$\mathcal{M}^* = \{i \in \mathcal{M}_0 : \mathbb{E}[d_{ij}] \leq 0 \; \forall j \in \mathcal{M}_0\}.$$

The MCS procedure is defined as follows.

Definition 3.1. Let $\delta_{\mathcal{M}}$ and $e_{\mathcal{M}}$ be an equivalence test and an elimination rule.

Step 0. Initially set $\mathcal{M} = \mathcal{M}_0$.

Step 1. Test $H_{0,\mathcal{M}} : \mathbb{E}[d_{ij}] = 0, \; \forall i,j \in \mathcal{M}$ using $\delta_{\mathcal{M}}$ at level α.

Step 2. If $H_{0,\mathcal{M}}$ is accepted, set $\hat{\mathcal{M}}^*_{1-\alpha} = \mathcal{M}$; otherwise, use $e_{\mathcal{M}}$ to eliminate an object from $\mathcal{M}$ and repeat the procedure from Step 1.

Under appropriate assumptions on the equivalence test and elimination rule[5], the model confidence set $\hat{\mathcal{M}}^*_{1-\alpha}$ has the following asymptotic properties:

(i) $\displaystyle\liminf_{W \to \infty} \mathbb{P}(\mathcal{M}^* \subset \hat{\mathcal{M}}^*_{1-\alpha}) \geq 1 - \alpha$,

(ii) $\displaystyle\lim_{W \to \infty} \mathbb{P}(i \in \hat{\mathcal{M}}^*_{1-\alpha}) = 0$ for $i \notin \mathcal{M}^*$,

where $W \to \infty$ denotes the number of sample periods. If furthermore, the equivalence test and elimination rule satisfy a coherency condition[6], we have the following finite sample property:

(iii) $\mathbb{P}(\mathcal{M}^* \subset \hat{\mathcal{M}}^*_{1-\alpha}) \geq 1 - \alpha$.

In Hansen et al. [2011], the authors give multiple practical examples of equivalence tests and elimination rules. We focus on implementing the MCS procedure with tests constructed from t-statistics and bootstrap estimators. This choice of equivalence test and elimination rule has two practical advantages: it does not require estimating a variance-covariance matrix for the time series $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, w \geq 1\}$ which might be difficult when $|\mathcal{M}_0| \approx W$ and it satisfies the coherency condition required for the finite sample property (iii) of the MCS procedure. Let us assume the following holds:

Assumption 3.1. For some $r > 2$ and $\gamma > 0$, it holds that that $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}, w \geq 1\}$ is strictly stationary, α-mixing of order $-r/(r-2)$ and $\mathrm{Var}(d_{ij,w}) > 0$, $\mathbb{E}|d_{ij,w}|^{r+\gamma} < \infty$ for all $i,j \in \mathcal{M}_0$.

For a set of models $\mathcal{M} \subset \mathcal{M}_0$ with time series of relative performance variables $\{(d_{ij,w})_{i,j \in \mathcal{M}}, w \geq 1\}$ define the following quantities $\forall i,j \in \mathcal{M}$:

$$\bar{d}_{ij} = \frac{1}{W} \sum_{w=1}^{W} d_{ij,w}, \quad \bar{d}_{i\cdot} = \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} \bar{d}_{ij},$$

respectively measuring the sample relative performance of the models i and j and the sample relative performance of model i and all other models in $\mathcal{M}$. We can then introduce the following t-statistics $\forall i \in \mathcal{M}$:

$$t_{i\cdot} = \frac{\bar{d}_{i\cdot}}{\sqrt{\widehat{\mathrm{Var}}(\bar{d}_{i\cdot})}},$$

where $\widehat{\mathrm{Var}}(\bar{d}_{i\cdot})$ are appropriate estimators of the variances of the $\mathrm{Var}(\bar{d}_{i\cdot})$. The associated test statistic is $T_{\max,\mathcal{M}} = \max_{i \in \mathcal{M}} t_{i\cdot}$. The equivalence test and elimination rule are given by:

($\delta_{\mathcal{M}}$) Reject $H_{0,\mathcal{M}}$ if $T_{\max,\mathcal{M}} > F_\varrho^{-1}(1-\alpha)$ where $T_{\max} \sim F_\varrho$.

---

[5]We require the equivalence test $\delta_{\mathcal{M}}$ to have asymptotic level α and asymptotic power 1, while the elimination rule $e_{\mathcal{M}}$ needs to satisfy $\lim_{W \to 0} \mathbb{P}(e_{\mathcal{M}} \in \mathcal{M}^* | H_{1,\mathcal{M}}) = 0$.

[6]The required coherency condition is $\mathbb{P}(\delta_{\mathcal{M}} = 1, e_M \in \mathcal{M}^*) \leq \alpha$.

$(e_{\mathcal{M}})$ Eliminate $\mathrm{argmax}_{i\in\mathcal{M}}t_{i\cdot}$.

The asymptotic distribution of the test statistic $T_{\max,\mathcal{M}}$ is nonstandard because it depends on nuisance parameters (ϱ under the null). This does not pose a problem as we can consistently estimate the distribution of $T_{\max,\mathcal{M}}$ by using a bootstrapping procedure and then compare the observed value of $T_{\max,\mathcal{M}}$ with a bootstrap empirical quantile. The resulting procedure preserves both the asymptotic property (as both the sample size W and the number of bootstrap samples B increase) and the finite sample property (iii) under Assumption 3.1. For details of the bootstrapping procedure see the Appendix of Hansen et al. [2011].

Remark 3.1. The test based on the t-statistic discussed above relies on the fact that the null hypothesis $H_{0,\mathcal{M}}$ can be equivalently rewritten as

$$H_{0,\mathcal{M}} : \mathbb{E}[d_{i\cdot}] = 0, \ \forall i \in \mathcal{M}, \quad \text{where} \quad d_{i\cdot} = \frac{1}{|\mathcal{M}|}\sum_{j\in\mathcal{M}} d_{ij}.$$

In the following Sections, we will be presenting results using MCS p-values, which are defined as follows.

Definition 3.2. Let $(\delta_{\mathcal{M}}, e_{\mathcal{M}})$ be the equivalence test and elimination rule associated with a MCS procedure as defined in 3.1. The elimination rule defines a decreasing sequence of random sets $\mathcal{M}_0 \supset \mathcal{M}_1 \supset \ldots \supset \mathcal{M}_{|\mathcal{M}_0|}$ by successively eliminating models $e_{\mathcal{M}_0}, \ldots, e_{|\mathcal{M}_0|}$. Let $p_{H_{0,\mathcal{M}}}$ denote the p-value associated with the null hypothesis $H_{0,\mathcal{M}}$ under the equivalence test $\delta_{\mathcal{M}}$ with the convention that $p_{H_{0,\mathcal{M}_{|\mathcal{M}_0|}}} \equiv 1$. Then for model $i = e_{\mathcal{M}_k} \in \mathcal{M}_0$ the MCS p-value is defined as

$$p_i^{\mathrm{MCS}} = \max_{k' \leq k} \ p_{H_{0,\mathcal{M}_{k'}}}.$$

Note that the definition of MCS p-value is such that for $i \in \mathcal{M}_0$ $p_i^{\mathrm{MCS}} \geq \alpha \iff i \in \hat{\mathcal{M}}_{1-\alpha}^*$.

Remark 3.2. As we will see in the following Sections, the MCS procedure is quite flexible. It will allow us to compare models between each other and compare them to a benchmark model (by including the benchmark in $\mathcal{M}_0$). When only two models are considered, i.e. $|\mathcal{M}_0| = 2$, the MCS procedure reduces to testing the null hypothesis $H_0 : \mathbb{E}[d_{i_1 i_2}] = 0$.

The model confidence set procedure relies on the assumption that the loss differences time series $\{(d_{ij,w})_{i,j\in\mathcal{M}_0}\}_w$ is stationary and α-mixing. This means that even when the loss process $\{(L_{i,w})_{i\in\mathcal{M}_0}\}_w$ is non-stationary, e.g. model performance is tied to regime changes, the model confidence set procedure might still be valid. In the following, we will assume the time series $\{(d_{ij,w})_{i,j\in\mathcal{M}_0}\}_w$ to be stationary based on intuitive arguments, but we discuss some statistical checks which might be carried out to provide further evidence in this context. First of all, it is worth noting that general tests for stationarity are not available: most tests require assuming a parametric form on the time series. The simplest example would be to impose a linear structure on the time series, i.e. assume the difference time series $\{\boldsymbol{d}_w\}_w = \{(d_{ij,w})_{i,j\in\mathcal{M}_0}\}_w$ follows a Vector Auto-Regressive process of order 1:

$$\boldsymbol{d}_w = \Gamma \boldsymbol{d}_{w-1} + \boldsymbol{\eta}_w,$$

where $\boldsymbol{\eta}_w \sim N(0, \Sigma_\eta)$ are serially uncorrelated. In this setting, one can consider various tests to determine whether the process is stationary. For example, one may apply multivariate homogeneous Dickey-Fuller tests [Harvey and Bates, 2003] to test the null of a unit root against the alternative of stationarity or a multivariate KPSS test (short-memory) to test (short-memory) stationarity as the null and a unit root as the alternative [Nyblom and Harvey, 2000].

Experimental setup We aim to explore the five questions discussed in the introduction via the statistical framework provided by the model confidence set procedure. We consider the same universe of stocks and trading period as in Kolm et al. [2021]: through LOBSTER [Huang and Polak, 2011] we have access to one year of trading data for 115 Nasdaq stocks from January 14, 2019, to January 31, 2020.

To produce results in feasible computational time, we select a subset of 10 stocks, trying to preserve a sufficiently varied set of liquidity characteristics, see appendix Section A.3. To apply the model confidence set procedure, we divide the 55 weeks of data into $W = 11$ five-week periods (note that data from the week starting on January 7, 2019, is used to start the rolling window normalization required for deepLOB and deepOF architectures). Each window is divided into a joint training-validation set, the first four weeks, and a test set, the fifth week. The training-validation set is then further split into training and validation sets by randomly selecting 5 days out of the four weeks for validation. The training dataset is used to choose the α to define the return labels, as detailed in appendix Section A.2 (note that the choice of α is specific to the choice of window, horizon, and ticker). The joint training-validation dataset is then used to train the models. For the empirical benchmark model, this simply means determining the empirical distribution of returns in the training-validation set. For the deep neural network architectures, this amounts to finding the optimal parameters which minimize the weighted cross-entropy loss. To do so, we use Adam optimization with validation-based early stopping as described in Section 2.2.2.

Remark 3.3. Note that not all training procedures may converge to the optimal combination of parameters. This is a characteristic of any model learned via numerical optimization methods. We further note that in our experimental setup, no hyperparameter tuning is carried out for any of the models. When using these models in a production setting, one may obtain better results by using cross-validation based on a further split of the training-validation set to select the best hyperparameter combination. Parameters which one may investigate tuning include:

- Architecture hyperparameters:

  - number of filters in each convolutional layer (we fix 32 channels);
  - number of weighted averages and lengths of averaging windows;
  - number of LSTM hidden nodes (we fix 64 hidden nodes);
  - decoder type for multi-horizon models (we use seq2seq).

- Feature hyper-parameters:

  - number of levels in features (we fix 10 levels);
  - queue depth in deepVOL(L3) (we fix queue depth 10);
  - look back window length (we fix $T = 100$);

- Training/Optimization hyper-parameters:

  - Adam learning rate (we fix $\eta = 0.01$);
  - Adam parameters (we fix $\epsilon = 1, \beta_1 = 0.9, \beta_2 = 0.999$);
  - batch size (we fix batch size 256);
  - number of epochs (we fix 50 epochs);
  - early stopping patience (we fix 10 epochs validation patience).

Here we are not interested in obtaining the best fit possible for a specific model but in comparing different models/benchmarks on a level playing field. We thus leave questions related to hyper-parameter tuning for future work.

It is worth noting that in all our experiments, we subset the training dataset by a factor of 10, i.e. subsequent feature-response observations in the dataset are obtained by sliding forward by 10 order book events. This serves multiple purposes: on one hand, it significantly reduces the computational effort required to train the models, on the other, it helps to prevent the risk of overfitting to specific patterns arising in the training dataset. Down-sampling the dataset might also have adverse effects, for example, reducing the available training data of an illiquid stock by a factor of 10 might result in insufficient information to robustly train these data-intensive models.

Once the models are trained, we compare their out-of-sample performance on the test set using the cross-entropy loss, i.e. we have time series $\{L_{i,w}\}_{w=1}^{11}$ for each model $i \in \mathcal{M}_0$ where the model set $\mathcal{M}_0$ will depend on the question/hypothesis investigated. From these we can thus compute the differences in performance $\{d_{ij,w}\}_{w=1}^{11}$ such that $d_{ij,w} = L_{i,w} - L_{j,w}$, which form the basis for the MCS procedure. We will assume the loss difference time series $\{(d_{ij,w})_{i,j \in \mathcal{M}_0}\}_{w=0}^{11}$ to be stationary. We do not conduct multivariate DF or KPSS tests as these are based on asymptotic distributions, and thus, due to the low data regime ($W = 11$), they would not yield statistically robust results.

The strength of applying the model confidence set in this context is that it fully accounts for parameter estimation error, while classic methods for comparing deep learning models do not. In the literature, deep learning models are compared by fitting the same model starting with different seeds and reporting the out-of-sample average loss and standard error. This method accounts only for the uncertainty arising in the fitting procedure and not in the sampled training and testing data sets (cf. Remark 2.3). The model confidence set instead naturally considers overall parameter estimation uncertainty, comprised of both statistical and optimization errors, by comparing model losses as general stochastic processes. This provides a more robust comparison of model performance, i.e. specification error.

Remark 3.4. The reader with particular attention to detail might argue that losses are computed on different test sample sizes (depending on the number of order book updates occurring in the test set, i.e. $n_{\text{text},w}$) and thus the variance of losses and of their differences may change over time. It is important to note that the sample sizes are random, and there does not seem to be strong evidence against their stationary, so unconditional on sample size, the difference process can still be reasonably assumed to be stationary.

All code is developed in python with the `tensorflow` library and the `keras` API. Some procedures required specific changes for which custom `tensorflow` methods were developed. Due to the computationally intensive nature of the experiments, specialized infrastructure was required to store the data (5TB) and train the models (GPUs).

All computations were carried out on Imperial College's High-Performance Computing cluster [Imperial College Research Computing Service], which provides access to several RTX6000 GPUs. The code is available on GitHub at `https://github.com/lorenzolucchese/deepOBs` under the BSD 3-clause license.

The results discussed hereafter are specific to the experimental setup under consideration, i.e. they are specific to the selected stocks, time period, and models. Different experimental setups may lead to different results.

3.1 Do high frequency returns display predictability? If so, how far ahead can we predict?

In order to understand whether high-frequency returns display predictability we consider the following set of models:

$$\mathcal{M}_0 = \big\{\text{benchmark, deepLOB(L1), deepOF(L1), deepLOB(L2), deepOF(L2), deepVOL(L2), deepVOL(L3)}\big\},$$

where L1 models use only the first level of the order book, while L2 and L3 models use the first 10 levels. For each return horizon $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$ and TICKER $\in$ {LILAK, QRTEA, XRAY, CHTR, PCAR, EXC, AAL, WBA, ATVI, AAPL} we determine the MCS p-value of each model in $\mathcal{M}$. Fixing a $1 - \alpha$ confidence level we say that there is predictability if $p_{\text{benchmark}}^{\text{MCS}} < \alpha$. The MCS p-value for the benchmark model are reported in Table 1. Lighter green shading corresponds to $\alpha = 0.05$ while darker green is $\alpha = 0.01$.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK	0.00	0.00	0.42	1.00	1.00	1.00	1.00	1.00	1.00
QRTEA	1.00	0.01	0.01	0.00	0.00	0.00	0.08	0.77	0.54
XRAY	0.00	0.00	0.00	0.01	0.10	0.23	0.48	1.00	0.93
CHTR	0.00	0.00	0.04	0.00	0.42	1.00	1.00	1.00	0.97
PCAR	0.00	0.00	0.00	0.00	1.00	1.00	0.60	1.00	1.00
EXC	0.26	0.00	0.00	0.00	0.00	0.00	0.00	0.51	1.00
AAL	0.11	0.00	0.00	0.00	0.00	0.71	1.00	1.00	1.00
WBA	0.00	0.00	0.00	0.00	0.02	0.27	0.13	0.01	1.00
ATVI	0.00	0.00	0.00	0.00	0.00	0.78	1.00	1.00	1.00
AAPL	0.00	0.00	0.00	0.23	1.00	1.00	1.00	1.00	1.00

Table 1: Benchmark MCS p-values for the 10 tickers and 9 horizons under consideration.

From the results reported in Table 1 we see that at high frequencies predictability is systematically present. For most of the stocks under consideration, we were able to identify predictability up to 50 order book events ahead at the 99% confidence level.

Except for LILAK, which is the most illiquid stock, we observe a substantial correlation between the persistence in predictability and the average Updates to Price Changes ratio, cf. Table 7. Recalling that in our setting the horizon h is measured in order book updates, this can be interpreted as it being easier for the deep learning models to predict returns that are the result of fewer price changes. One might thus expect to obtain a more consistent maximum predictable horizon across stocks when using a price change-driven clock to measure time.

3.2 Which order book representations perform the best?

Having discussed the extent to which the class of models under consideration can identify predictability, it is now natural to ask which of these models performs best. In the MCS framework, this corresponds to determining the specifications which are consistently placed in the set of superior models. We restrict our attention to the horizons and stocks at which predictability is identified and, for each model in consideration, we count the number of times it is identified as a superior model. The results are reported in Table 2.

	$\alpha = 0.05$	$\alpha = 0.01$
benchmark	0%	0%
deepLOB(L1)	7%	11%
deepOF(L1)	23%	22%
deepLOB(L2)	5%	11%
deepOF(L2)	88%	89%
deepVOL(L2)	65%	84%
deepVOL(L3)	77%	86%

Table 2: % of times the model is in the α-MCS when predictability is identified at the corresponding level α.

From the results in Table 2 (at the 99% confidence level) we can make the following observations on the way order book representations influence model performance. When considering deep learning models for short-term return prediction having access to L2 data provides a significant advantage over L1 data. We see that models with L1 data are rarely placed in the set of superior models when predictability is identified. When going from L2 to L3 data instead, the increased granularity does not seem to provide a clear advantage: deepVOL(L2) and deepVOL(L3) display similar performance. Our experiment, therefore, suggests that L3 data might be excessively granular when predicting high-frequency returns from order books.

The choice of features used to represent the order book is also crucial when leveraging deep learning methods for return prediction. Using order flow or volume representations provides a significant improvement in performance: the basic deepLOB model ends up being included in the set of best models only 10% of the time and is outperformed even by the model with only first-level (L1) order flow. Volume- and order flow-based models (with L2/L3 data granularity) display comparable performance, being placed in the set of superior models in 85-90% of the predictable horizons.

3.3   Can we use a single model across multiple horizons?

In this Section, we will discuss whether using a seq2seq decoder to produce multi-horizon predictions is beneficial. Such models have the clear advantages of having a single set of weights (the network size is only slightly bigger than single horizon networks) and output multiple predictions in very similar run times. This significantly reduces both the memory required to store the models and the time needed to train them. But how do they perform when compared to their single-horizon counterparts?

To answer this question we run the same experiment as in 3.1 and 3.2 but enlarging the set of models with the seq2seq specifications. We focus only on prediction horizons $h \in \{10, 20, 30, 50\}$. The results are reported in Tables 3 and 4.

From Table 4 we note that, for each input type, the seq2seq specifications outperform their single horizon counterparts. We suggest this behavior might be due to the increased availability of information in a multi-horizon setting. For a given input-target pair, when targets are multi-horizon, more information is available on the "order book regime" the inputs should be mapped to. Multi-horizon models can thus learn a more granular map from the input variables to the latent space of "order book regimes", which might be beneficial for producing predictions.

Remark 3.5. Note that running the model confidence set procedure again with a larger set of models leads to a counter-intuitive situation where fewer prediction horizons are identified. One would expect that adding models could only increase the number of horizons at which predictability is identified. While this is true in the limit as the number oh horizons increases to infinity, it does not hold in the setting where we have access to a finite set of observations. With this observation in mind, we note that results in Table 3 are consistent with those in Table 1.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$
LILAK	0.00	0.00	0.31	0.55
QRTEA	1.00	0.10	0.06	0.00
XRAY	0.00	0.00	0.00	0.00
CHTR	0.00	0.00	0.01	0.01
PCAR	0.00	0.00	0.00	0.00
EXC	0.23	0.00	0.00	0.00
AAL	0.08	0.00	0.00	0.00
WBA	0.00	0.00	0.00	0.00
ATVI	0.01	0.00	0.00	0.00
AAPL	0.00	0.00	0.00	0.07

Table 3: Benchmark MCS p-values for the 10 tickers and 9 horizons under consideration when seq2seq models are also considered.

	$\alpha = 0.05$	$\alpha = 0.01$
benchmark	0%	0%
deepLOB(L1)	6%	10%
deepOF(L1)	13%	16%
deepLOB(L2)	6%	13%
deepOF(L2)	63%	71%
deepVOL(L2)	63%	74%
deepVOL(L3)	75%	84%
deepLOB(L1, seq2seq)	13%	26%
deepOF(L1, seq2seq)	16%	16%
deepLOB(L2, seq2seq)	16%	39%
deepOF(L2, seq2seq)	91%	97%
deepVOL(L2, seq2seq)	78%	84%
deepVOL(L3, seq2seq)	81%	87%

Table 4: % of times the model is in the α-MCS when predictability is identified at the corresponding level α.

3.4 Can we use a single model across multiple stocks?

In a similar spirit to Sirignano and Cont [2019], we investigate questions regarding the universality of order book dynamics. Intuitively, at a microstructural level, securities which are traded by market participants with similar characteristics may be subject to the same trading patterns, independently of the underlying stock's properties. In Sirignano and Cont [2019], the authors observe that price formation dynamics driven by past order book information, i.e. next mid-price moves, display common patterns across different stocks. In this paper, we explore whether similar results hold for the short-term predictability of mid-prices, i.e. returns at fixed horizons.

We run the same experiment as in Section 3.1 but train the models on multiple stocks simultaneously. Specifically, we split the set of 10 stocks under consideration into two; a first "in-sample" set of stocks given by {QRTEA, CHTR, EXC, WBA, AAPL}, and a second "out-of-sample" set {LILAK, XRAY, PCAR, AAL, ATVI}. For each window w, we use all the training-validation data for the "in-sample" stocks to train (and validate) the models. We then evaluate the trained models on the test data of both the "in-sample" and the "out-of-sample" stocks. The results are reported in Table 5.

	$h = 10$	$h = 20$	$h = 30$	$h = 50$	$h = 100$	$h = 200$	$h = 300$	$h = 500$	$h = 1000$
LILAK	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
QRTEA ($*$)	0.00	0.00	0.01	0.06	0.00	0.13	0.59	0.34	1.00
XRAY	0.03	1.00	1.00	1.00	1.00	1.00	0.62	0.81	1.00
CHTR ($*$)	1.00	1.00	1.00	1.00	1.00	1.00	0.94	1.00	1.00
PCAR	0.08	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
EXC ($*$)	0.00	0.00	0.00	0.00	0.00	0.39	0.69	1.00	1.00
AAL	0.00	0.00	0.00	0.00	0.00	0.58	0.60	1.00	1.00
WBA ($*$)	0.00	0.00	0.00	0.00	0.02	0.58	0.36	0.14	1.00
ATVI	0.00	0.00	0.00	0.00	0.60	0.74	0.88	1.00	1.00
AAPL ($*$)	0.07	1.00	1.00	1.00	1.00	0.59	1.00	0.65	1.00

Table 5: Benchmark MCS p-values for the 10 tickers and 9 horizons under consideration, universal models only. ($*$) are "in-sample" stocks.

Focusing on "in-sample" stocks, we note that using universal models leads to results that are partially inconsistent with those in Table 1. A possible interpretation for this is that universal models may be picking up different order book dynamics from those identified by stock-specific models. In this sense, relatively illiquid stocks with overall "standard" trading behavior might benefit from universal models thanks to the greater availability of data. We suggest this might be the case for QRTEA and EXC at horizon $h = 10$ (where we remark that at such a short horizon the three classes $\{\downarrow, =, \uparrow\}$ are significantly unbalanced for these two specific tickers). When, instead, the ticker is mainly subject to

stock-specific trading patterns, universal models have a hard time detecting predictability – we believe this might be the case for CHTR and AAPL. These results are intrinsically tied to the observations of Remark 3.6.

It is rather remarkable that universal models can identify predictability for "out-of-sample" stocks. This provides evidence of the presence of common trading patterns in the order books of different tickers. For AAL and ATVI, universal models can consistently outperform benchmark predictions without ever learning from the stock's past order book data.

	$\alpha = 0.05$	$\alpha = 0.01$
benchmark	0%	0%
deepLOB(L1, universal)	0%	0%
deepOF(L1, universal)	0%	0%
deepLOB(L2, universal)	8%	24%
deepOF(L2, universal)	42%	62%
deepVOL(L2, universal)	71%	81%
deepVOL(L3, universal)	92%	100%

Table 6: % of times each model is in the α-MCS when predictability is identified at the corresponding level α.

Since we believe universal and stock-specific models may be picking up different trading patterns, a natural question to ask is whether the conclusions from Section 3.2 carry over to the universal setting. The results in Table 6 suggest the superior power of L2 over L1 data is retained for universal models. In this case, though, the increased granularity of L3 data might be beneficial when looking for universal trading patterns. Moreover, when considering universal models, the volume representation appears to outperform both order book and order flow inputs.

Remark 3.6. As discussed in Section A.2, the choice of α used to define the return classes is stock-specific. This means that up/down labels for different securities may correspond to different numbers of mid-price changes. This is in line with our choice of an order book-driven clock, which is by definition also stock-specific. It is important to note that the choices of α and time t might not entirely account for structural trading differences of stocks, making it harder to identify universal trading patterns. This contrasts with Sirignano and Cont [2019], where the next mid-price move has a straightforward universal structural interpretation.

Remark 3.7. There are a couple more differences with Sirignano and Cont [2019] we want to point out:

- First, in that paper, the authors also investigate questions regarding the stationarity of price formation dynamics. This entails using a single long training window: the greater availability of data leads to more stable results. To employ the model confidence set procedure and allow the models to capture patterns specific to different economic regimes, we did not adopt this approach in our work.

- The LSTM model considered in that paper differs from the deepLOB/deepOF/deepVOL architectures. In Sirignano and Cont [2019], the LSTM specification is an online model, i.e. inputs at time t represent the order book state at time t only and are fed through an LSTM-based architecture updating the stored hidden state and producing output predictions. In our models, instead, there is no storage of hidden units between one prediction and the next, i.e. at each time step t, we input the order book history between $t - T + 1$ and t (in the form of raw order books, order flow or volumes) and, after appropriate convolutional feature extraction, apply an LSTM to the processed sequential data to produce the prediction. When studying dependence on order book history T, Sirignano and Cont [2019] refer to the cut-off horizon used in the backpropagation through time computation of gradients during the training procedure.

4 Conclusions and Outlook

In this paper, we explored empirical questions regarding the predictability of mid-price returns driven by order book data in centralized exchanges. The predictability in price formation dynamics, already considered in the sense of next mid-price change in Sirignano and Cont [2019], was found to persist up to 50-300 order book updates ahead, horizons over which multiple mid-price changes may occur. Such predictable horizons might vary from a few milliseconds to nearly half a second, depending on the stock under consideration. These results contrast with low-frequency returns, where predictability is much harder to identify but easier to trade once discovered. In fact, in the high-frequency context, predictability is not always exploitable due to technological limitations and market microstructure issues. A related

and more challenging question is, therefore, to understand whether the predictability identified in this paper is actually tradeable.

The experiments were carried out using specific deep learning architectures. Other than identifying predictable horizons, we aimed to answer questions related to the models under consideration. In particular, we found strong empirical evidence for using L2 data over L1 data. But, from the results presented in this paper, using even more granular order book information (L3) seems to benefit only universal models. The experiments also highlight the importance of carefully choosing a data representation: models based on the basic order book level representation are considerably outperformed by those with order flow or volume inputs. Finally, we found empirical evidence for the presence of universal trading patterns in order book dynamics. Our preliminary results suggest that deep learning architectures may pick up different predictable order book patterns when applied to a pool of stocks instead of a single one. The volume representation has some considerable theoretical advantages, in particular, it is robust to small perturbations, which might explain its superior predictive ability when considering universal models.

Many further theoretical questions regarding predictability which we believe to be of research interest remain unanswered. First, it would be natural to explore whether predictability in returns can be entirely explained by order book structure or if recurring trading patterns play a relevant role. Next, one could compare the predictive performance of the deep learning architectures discussed in this paper to that of the models based on carefully engineered features considered in Aït-Sahalia et al. [2022]. Finally, only Nasdaq-traded stocks were considered in this paper. This is a centralized electronic dealer market on which relatively big companies are listed. It would be interesting to explore whether similar results are obtained for securities traded on exchanges with different market structures. For example, one could consider auction markets (e.g. NYSE) or foreign exchange markets. With appropriate experimental setups, we believe the model confidence set procedure used in this paper provides a solid framework to tackle all these questions.

On the practical side, there are some relevant issues one should consider. First, it is essential to note that in this paper, we focus on mid-to-mid returns, which are not tradeable in practice. When considering specific trading applications, one should thus define the return labels appropriately, for example, working with ask-to-bid returns. Second, in this paper, we use an order book-driven clock. In practice, this means the prediction horizon is intrinsically stochastic, which might be a problem in execution. Another critical aspect that would need to be accounted for in practical high-frequency trading is total execution speed. This consists of order book information latency, model prediction run time, and order submission speed. Zhang et al. [2019] and Kolm et al. [2021] argue that their models (which are also considered in this paper) are sufficiently fast to be used by traders with good connections.

As with any trading application, one must also account for the impact his own trade will have on the order book. Moreover, if many traders exploit the same inefficiencies, this might erode all predictive power above the lowest tradeable latencies. Overall we believe that while the predictability identified in this paper might not be directly tradeable through a standalone strategy, it might still help some market players, such as market makers, to gauge the direction of the market and adjust their quotes accordingly.

A Appendix

A.1 Data Processing

We obtain data from LOBSTER in the form of order book and message book files. These files are jointly processed to obtain the desired order book features: raw order books, order flow and volumes. The data is treated in the following way:

1. order book states with crossed quotes are removed;
2. states in the order book are time-stamped to nanosecond precision, with states occurring at the same time collapsed onto the last state (e.g. an aggressive order executing against multiple resting limit orders);
3. the first and last 10 minutes of market activity are dropped;
4. the smoothed returns at the requested horizons – as defined in Section A.2 – are computed and matched to the corresponding features.

The data provided by LOBSTER is easily accessible and does not exhibit significant corruption. The only data inconsistencies we found were on 2019-11-05 for the WBA ticker after a trading halt occurring between 48770.233001415 and 49070.2335639 seconds from midnight.

Volume data processing   LOBSTER provides level-based data in a form that can be easily manipulated to derive order book and order flow features (we access the first $L = 10$ levels). Deriving the volume features from LOBSTER data instead is a non-trivial task. If we set a window $W > 0$ for the volume representation, we may encounter the following cases at timestamp t:

- some prices in the first $L = 10$ levels fall out of the volume window, i.e. $p_{x,t}^{(l)} \notin \{\pi_{x,t}^{(1)}, \ldots, \pi_{x,t}^{(W)}\}$, and thus some information is discarded;
- some levels past $L = 10$ have prices which fall in the volume window, i.e. $p_{x,t}^{(L+1)} \in \{\pi_{x,t}^{(1)}, \ldots, \pi_{x,t}^{(W)}\}$, but we won't have the required information (this may happen only if $W > L$).

These issues apply to both L2 and L3 data. Reconstruction of L3 queue data has some further issues: every time a quoting price leaves the 10-level range, all orders submitted/canceled at this price are not recorded. Therefore there is no way of reconstructing what happens at such a price level while it is outside of the 10-level range. To minimize the effect of this shortcoming, we adopt the following systematic approach: each time a quoting price (re-)enters the 10-level range, we initialize the corresponding queue with a single order of size given by the total liquidity available at that price level (even if it may be made up of multiple independent orders).

All of these issues could be mitigated by accessing data deeper in the order book from LOBSTER, say $L = 50$ or 100, while keeping a relatively small window $W << L$.

A.2 Return definition

The way returns are defined is often an overlooked topic which inevitably has a profound effect on model performance. In all cases, we treat the mid-price as the "true" price and define returns relative to it, but it is important to note that, by definition, this is not a tradeable price. Here we discuss three different return definitions used in the literature and their properties.

- In Ntakaris et al. [2018] and Kolm et al. [2021] the authors define the return at horizon h as

$$r_{t+h} = \frac{\overline{m}_{t,t+h} - m_t}{m_t}, \ \overline{m}_{t,t+h} = \frac{1}{h} \sum_{i=1}^{h} m_{t+i},$$

  where m_t denotes the mid-price at time t. This definition of return compares the smoothed mid-price over the next h time steps to the current mid-price. In this case, the return can be equivalently understood as the average mid-to-mid return over the next h time steps. As the horizon h becomes longer, so does the window over which we smooth the returns. This has a couple of practical disadvantages in terms of the predictability questions we wish to explore. First, predictability over $h = 100$ time steps might be due to changes of the mid-price over the first, say, $h = 50$ time steps, and thus it becomes difficult to discuss questions of persistency in predictability. Second, the definition produces correlation between returns at different horizons, r_{t+h_1} and r_{t+h_2}, inducing a hidden bias in multi-horizon models.

- In Zhang et al. [2019] and Zhang and Zohren [2021] the authors define the return at horizon h by

$$r_{t+h} = \frac{\overline{m}_{t,t+h} - \overline{m}_{t-h,t}}{\overline{m}_{t-h,t}}, \ \overline{m}_{t,t+h} = \frac{1}{h} \sum_{i=1}^{h} m_{t+i}, \ \overline{m}_{t-h,t} = \frac{1}{h} \sum_{i=0}^{h-1} m_{t-i},$$

  where m_t denotes the mid-price at time t. This definition has similar drawbacks to the previous one. But, additionally, we believe it may suffer from look-ahead bias: for example, if the mid-price has been going up in the past h steps (an information which is included in our covariates), then it is more likely the return r_{t+h} will be positive.

- In this paper, instead, we define the return at horizon h as

$$r_{t+h} = \frac{\overline{m}_{t+h}^{(k)} - m_t}{m_t}, \ \overline{m}_{t+h}^{(k)} = \frac{1}{2k+1} \sum_{i=-k}^{k} m_{t+h+i},$$

  where m_t denotes the mid-price at time t and k is a fixed smoothing window. This definition of returns does not suffer from the issues identified for the other two. It is subject to two possible interpretations. First, treating the smoothed mid-price as a de-noised estimate of the true (latent) price h steps ahead, we can understand the return as the percentage change of the true (latent) price relative to the current mid. Second, the return can be understood as the average return one would experience by entering a position at the current mid and exiting it roughly h steps ahead (assuming mid-mid trading). For all the horizons $h \in \{10, 20, 30, 50, 100, 200, 300, 500, 1000\}$ we fix $k = 5$.

Remark A.1. All these definitions intrinsically depend on the clock used to measure time. As discussed in Section 2.1, we use an order book-based clock which ticks every time the order book changes. We do not use raw mid-price changes as we believe these have a couple of practical disadvantages. First, they are subject to stronger idiosyncratic noise effects. Second, even if we assume mid-mid trading, there is no way of placing a trade exactly h order book updates ahead, i.e. the prediction horizon is random. Smoothing the exit price can thus be understood as averaging out the uncertainty in the execution time.

All our experiments are carried out in a classification framework where the discretized returns are defined as

$$c_{t+h} = \begin{cases} \downarrow & \text{if } r_{t+h} \in (-\infty, -\alpha), \\ = & \text{if } r_{t+h} \in [-\alpha, +\alpha], \\ \uparrow & \text{if } r_{t+h} \in (+\alpha, +\infty), \end{cases}$$

for some $\alpha > 0$. We empirically choose α from the training set $\mathcal{D}_{w,\text{train}}$ by

$$\hat{\alpha}_{h,w} = \frac{|\hat{Q}_{w,h}(0.33)| + \hat{Q}_{w,h}(0.66)}{2},$$

where $\hat{Q}_{w,h}$ is the empirical quantile function of the training set returns $\{r_{t+h}\}_{t \in \mathcal{I}_{w,\text{train}}}$. The choice of α is, therefore, window w and horizon h specific.

Remark A.2. In multi-horizon models, the choice of α will therefore vary across horizons $h_1, \ldots, h_K$ and, similarly, in universal models, it will vary across stocks.

We note that, since stocks trade on a discrete grid of prices determined by the tick size ϑ also the mid-price m_t evolves on a discrete price (with steps of size $\vartheta/2$). One could thus define the return from t to $t + h$ by the number of (half) ticks the mid-price moves, i.e.

$$R_{t+h} = m_{t+h} - m_t \in \{\ldots, -\vartheta, -\vartheta/2, 0, \vartheta/2, \vartheta, \ldots\} \equiv \{\ldots, -2, -1, 0, +1, +2, \ldots\}.$$

This return is by definition discretized, and thus one could directly apply classification models (grouping large negative and positive returns to obtain a finite number of classes). A similar approach is used in Sirignano and Cont [2019] when predicting the next change in mid-price. In our work, we consider the estimate for the "true" mid-price to be

$$\overline{m}_{t+h}^{(k)} = \frac{1}{2k+1} \sum_{i=-k}^{k} m_{t+h+i},$$

which lives on a much finer grid than m_{t+h}: over the, possibly quite long, time horizon h multiple changes to the mid-price might occur. In this case the smallest change has little meaning and so we group the returns

$$R_{t+h} = \overline{m}_{t+h}^{(k)} - m_t \in \{\ldots, -\frac{\vartheta}{2k+1}, -\frac{\vartheta}{4k+2}, 0, \frac{\vartheta}{4k+2}, \frac{\vartheta}{2k+1}, \ldots\}$$

into "larger" classes given by

$$(-\infty, -m_t\alpha), [-m_t\alpha, +m_t\alpha], (+m_t\alpha, +\infty).$$

A.3   Stock selection

Ideally, we would like to work with the same set of stocks as in Kolm et al. [2021]. Due to computational limitations, we were able to conduct experiments only on a subset of 10 tickers from the total 115 NASDAQ stocks considered in the original paper. We selected 10 stocks with diverse liquidity characteristics, hopefully providing a sufficiently representative sub-sample of the whole set.

To choose the sub-sample, we use the stock characteristics provided in Kolm et al. [2021], Table 6. For each of the liquidity characteristics – Updates, Trades, Price Changes, Spread – we compute a sub-score based on the characteristic's rank. For example, the stock with the most updates is assigned an updates score of 1, while the stock with the least number of updates is assigned an updates score of 0. The characteristic-specific scores are then averaged to obtain a general "liquidity score" for each stock. The 10 chosen stocks correspond to the 10 evenly spaced quantiles of the "liquidity score". The liquidity characteristics of the 10 chosen stocks are reported in Table 7.

Ticker	Updates (000)	Trades (000)	Price Changes (000)	Price (USD)	Spread (bps)	Volume (USD MM)
LILAK	49.89	1.81	3.77	18.09	15.92	2.63
QRTEA	121.32	5.23	2.79	13.54	9.75	9.53
XRAY	83.07	5.41	7.11	52.32	4.35	21.67
CHTR	80.97	6.75	19.21	400.68	6.22	111.71
PCAR	131.38	6.85	13.03	70.63	3.82	34.43
EXC	298.32	7.58	7.21	47.35	2.45	37.55
AAL	398.13	13.15	10.99	30.65	3.88	50.29
WBA	328.5	13.2	15.18	57.81	2.54	75.19
ATVI	423.01	17.97	20.25	49.87	2.81	91.82
AAPL	1137.36	64.19	127.86	215.8	0.99	1156.3

Table 7: Selected stocks' characteristics, daily averages.

A.4   Model details

In the experiments we use the model architectures reported in Tables 8 and 9. To speed up the training procedure, we use batch normalization [Ioffe and Szegedy, 2015] (with momentum = 0.6) after every convolutional layer and inception block. To prevent overfitting, we use a dropout layer [Srivastava et al., 2014] (with noise = 0.2) positioned after the inception module.

All the models are implemented with float32 precision policy: at every layer float32 is used as computation and variable data type. Loss and gradient computations are also carried out with float32 precision.

Model Architectures — Table 8: DeepLOB and DeepOF network description.

Model		DeepLOB Architecture	DeepLOB Description	DeepLOB Output Size	DeepOF Description	DeepOF Architecture	DeepOF Output Size
Input Layer			Raw order book states, $\{(p_{a,t-\tau}^{(l)}, v_{a,t-\tau}^{(l)}, p_{b,t-\tau}^{(l)}, v_{b,t-\tau}^{(l)})\}_{\tau=0,\ldots,T-1}^{l=1,\ldots,L}$	$(T, 4L, 1)$	Multi-level order flow, $\{(aOF_{t-\tau}^{(l)}, bOF_{t-\tau}^{(l)})\}_{\tau=0,\ldots,T-1,\, l=1,\ldots,L}$		$(T, 2L, 1)$
First Convolutional layer	Spatial convolution	32 (1×2) filters, (1×2) stride	Convolve price and volume information	$(T, 2L, 32)$	Convolve order flow information	32 (1×2) filters, (1×2) stride	$(T, L, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 2L, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, L, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 2L, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, L, 32)$
Second Convolutional layer	Spatial convolution	32 (1×2) filters, (1×2) stride	Convolve bid and ask price-volume information	$(T, L, 32)$			
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$			
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, L, 32)$			
Third Convolutional layer	Spatial convolution	32 $(1\times L)$ filters, (1×1) stride	Convolve information through the whole order book	$(T, 1, 32)$	Convolve information through the whole order book	32 $(1\times L)$ filters, (1×1) stride	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, 1, 32)$
Inception module (parallel blocks)	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (1×1) filters	$(T, 1, 64)$
	Temporal convolution	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$	Convolve through three time-steps, "MA(3)"	64 (3×1) filters, padding	$(T, 1, 64)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (1×1) filters	$(T, 1, 64)$
	Temporal convolution	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$	Convolve through five time-steps, "MA(5)"	64 (5×1) filters, padding	$(T, 1, 64)$
	Temporal maxpooling	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$	Rolling maximum, three time-steps	(3×1) pool size, (1×1) stride & padding	$(T, 1, 32)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (4×1) filters	$(T, 1, 64)$
Concatenate and reshape			Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$	Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T		$(T, 192)$
LSTM module	LSTM	64 hidden units	Capture longer term dependencies	(64)	Capture longer term dependencies	64 hidden units	(64)
(a) Output layer	Dense layer	Softmax activation	Produce classification output	(3)	Produce classification output	Softmax activation	(3)
(b) Decoder module	LSTM decoder (seq2seq/attention)	64 hidden units, softmax activation	Produce sequential classification output	$(5, 3)$	Produce sequential classification output	64 hidden units, softmax activation	$(5, 3)$

Table 8: DeepLOB and DeepOF network description.
(a) Single horizon, (b) Multi-horizon.

Model Architectures — Table 9: DeepVOL and DeepVOL L3 network description.

Model		DeepVOL Architecture	DeepVOL Description	DeepVOL Output Size	DeepVOL L3 Description	DeepVOL L3 Architecture	DeepVOL L3 Output Size
Input Layer			Volume representation, $\{s_{x,t-\tau}^{(j)}\}_{\tau=0,\ldots,T-1,\, j=1,\ldots,W,\, x\in\{a,b\}}$	$(T, W, 2, 1)$	L3 volume representation, $\{(q_{x,t-\tau}^{(j,k)})\}_{\tau=0,\ldots,T-1,\, j=1,\ldots,W,\, k=1,\ldots,D,\, x\in\{a,b\}}$		$(T, W, 2, D, 1)$
First Convolutional layer	Spatial convolution	32 $(1\times2\times2)$ filters, $(1\times1\times1)$ stride	Convolve volumes on opposite sides of the mid	$(T, W-1, 32)$	At each price, convolve the queue volumes	32 $(1\times1\times D)$ filters, $(1\times1\times1)$ stride	$(T, W, 2, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$	Convolve volumes on opposite sides of the mid	32 $(1\times2\times2)$ filters, $(1\times1\times1)$ stride	$(T, W-1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, W-1, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, W-1, 32)$
Second Convolutional layer	Spatial convolution	32 $(1\times W{-}1)$ filters, (1×1) stride	Convolve information through the whole order book	$(T, 1, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, W-1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	Convolve information through the whole order book	32 $(1\times W{-}1)$ filters, (1×1) stride	$(T, 1, 32)$
	Temporal convolution	32 (4×1) filters, padding	Convolve through four time-steps	$(T, 1, 32)$	Convolve through four time-steps	32 (4×1) filters, padding	$(T, 1, 32)$
Third Convolutional layer	Spatial convolution				Convolve through four time-steps	32 (4×1) filters, padding	$(T, 1, 32)$
	Temporal convolution						
	Temporal convolution						
Inception module (parallel blocks)	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (1×1) filters	$(T, 1, 64)$
	Temporal convolution	64 (3×1) filters, padding	Convolve through three time-steps, "MA(3)"	$(T, 1, 64)$	Convolve through three time-steps, "MA(3)"	64 (3×1) filters, padding	$(T, 1, 64)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (1×1) filters	$(T, 1, 64)$
	Temporal convolution	64 (5×1) filters, padding	Convolve through five time-steps, "MA(5)"	$(T, 1, 64)$	Convolve through five time-steps, "MA(5)"	64 (5×1) filters, padding	$(T, 1, 64)$
	Temporal maxpooling	(3×1) pool size, (1×1) stride & padding	Rolling maximum, three time-steps	$(T, 1, 32)$	Rolling maximum, three time-steps	(3×1) pool size, (1×1) stride & padding	$(T, 1, 32)$
	Temporal convolution	64 (1×1) filters	Increase dimensionality (Network-in-Network)	$(T, 1, 64)$	Increase dimensionality (Network-in-Network)	64 (4×1) filters	$(T, 1, 64)$
Concatenate and reshape			Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T	$(T, 192)$	Concatenate and reshape the outputs of the inception module to obtain an $\mathbb{R}^{192}$-valued time series of length T		$(T, 192)$
LSTM module	LSTM	64 hidden units	Capture longer term dependencies	(64)	Capture longer term dependencies	64 hidden units	(64)
(a) Output layer	Dense layer	Softmax activation	Produce classification output	(3)	Produce classification output	Softmax activation	(3)
(b) Decoder module	LSTM decoder (seq2seq/attention)	64 hidden units, softmax activation	Produce sequential classification output	$(5, 3)$	Produce sequential classification output	64 hidden units, softmax activation	$(5, 3)$

Table 9: DeepVOL and DeepVOL L3 network description.
(a) Single horizon, (b) Multi-horizon.

References

F. Abergel, C.-A. Lehalle, and M. Rosenbaum. Understanding the Stakes of High-Frequency Trading. *The Journal of Trading*, 9(4):49–73, 2014. ISSN 1559-3967. doi:10.3905/jot.2014.9.4.049. URL https://jot.pm-research.com/content/9/4/49.

Y. Aït-Sahalia, J. Fan, L. Xue, and Y. Zhou. How and When are High-Frequency Stock Returns Predictable? NBER Working Papers 30366, National Bureau of Economic Research, Inc, Aug. 2022. URL https://ideas.repec.org/p/nbr/nberwo/30366.html.

Y. Bengio, A. C. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8):1798–1828, Aug. 2013. ISSN 0162-8828. doi:10.1109/TPAMI.2013.50. URL https://ieeexplore.ieee.org/document/6472238.

K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, 2014. doi:10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

R. Cont, A. Kukanov, and S. Stoikov. The Price Impact of Order Book Events. *Journal of Financial Econometrics*, 12(1):47–88, 06 2013. ISSN 1479-8409. doi:10.1093/jjfinec/nbt003. URL https://doi.org/10.1093/jjfinec/nbt003.

G. V. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

European Parliament and Council. Directive 2004/39/EC on markets in financial instruments amending Council Directives 85/611/EEC and 93/6/EEC and Directive 2000/12/EC of the European Parliament and of the Council and repealing Council Directive 93/22/EEC, 2004.

E. F. Fama. Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, 25 (2):383–417, 1970. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/2325486.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

P. R. Hansen, A. Lunde, and J. M. Nason. The Model Confidence Set. *Econometrica*, 79(2):453–497, 2011. doi:https://doi.org/10.3982/ECTA5771. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA5771.

A. Harvey and D. Bates. Multivariate Unit Root Tests and Testing for Convergence. Cambridge Working Papers in Economics 0301, Faculty of Economics, University of Cambridge, Jan 2003. URL https://ideas.repec.org/p/cam/camdae/0301.html.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Los Alamitos, CA, USA, jun 2016. IEEE Computer Society. doi:10.1109/CVPR.2016.90. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi:10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies, 2001.

K. Hornik, M. Stinchcombe, and H. White. Universal Approximation of an Unknown Mapping and its Derivatives using Multilayer Feedforward Networks. *Neural Networks*, 3(5):551–560, 1990. ISSN 0893-6080. doi:https://doi.org/10.1016/0893-6080(90)90005-6. URL `https://www.sciencedirect.com/science/article/pii/0893608090900056`.

R. Huang and T. Polak. LOBSTER: Limit Order Book Reconstruction System. *Information Systems & Economics eJournal*, 2011.

Imperial College Research Computing Service. High-Performance Computing Cluster. doi:10.14469/hpc/2232.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/ioffe15.html`.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2015.

P. N. Kolm, J. D. Turiel, and N. Westray. Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book. *Econometric Modeling: Capital Markets - Portfolio Theory eJournal*, 2021.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, sep 2015. Association for Computational Linguistics. doi:10.18653/v1/D15-1166. URL `https://aclanthology.org/D15-1166`.

G. Melis, C. Dyer, and P. Blunsom. On the State of the Art of Evaluation in Neural Language Models. *CoRR*, abs/1707.05589, 2017. URL `http://arxiv.org/abs/1707.05589`.

A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37(8):852–866, 2018. doi:https://doi.org/10.1002/for.2543. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/for.2543`.

J. Nyblom and A. Harvey. Tests of Common Stochastic Trends. *Econometric Theory*, 16(2):176–199, 2000. ISSN 02664666, 14694360. URL `http://www.jstor.org/stable/3533193`.

J. Sirignano and R. Cont. Universal Features of Price Formation in Financial Markets: Perspectives from Deep Learning. *Quantitative Finance*, 19(9):1449–1459, 2019. doi:10.1080/14697688.2019.1622295. URL `https://doi.org/10.1080/14697688.2019.1622295`.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

M. Telgarsky. Representation Benefits of Deep Feedforward Networks. *arXiv e-prints*, art. arXiv:1509.08101, 2015.

US Securities and Exchange Commission. Regulation NMS - Proposed rules and amendments to joint industry plans, 2005.

Y. Wu, M. Mahfouz, D. Magazzeni, and M. Veloso. Towards Robust Representation of Limit Orders Books for Deep Learning Models. Papers 2110.05479, arXiv.org, Oct. 2021. URL `https://ideas.repec.org/p/arx/papers/2110.05479.html`.

K. Xu, M. D. Gould, and S. D. Howison. Multi-level Order-flow Imbalance in a Limit Order Book. *Market Microstructure and Liquidity*, 4, 2019.

Z. Zhang and S. Zohren. Multi-Horizon Forecasting for Limit Order Books: Novel Deep Learning Approaches and Hardware Acceleration using Intelligent Processing Units. Papers 2105.10430, arXiv.org, May 2021. URL `https://ideas.repec.org/p/arx/papers/2105.10430.html`.

Z. Zhang, S. Zohren, and S. Roberts. DeepLOB: Deep Convolutional Neural Networks for Limit Order Books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019. doi:10.1109/TSP.2019.2907260.