III. TRADING WITH MACHINE LEARNING AND BIG DATA

7. MACHINE LEARNING AND BIG DATA TRADE EXECUTION SUPPORT

Erin Stanton
Global Head of Analytics Client Support, Virtu Financial

Introduction

As a result of recent advances in computing power, data abundance, and cloud services, asset managers are increasingly eager to use artificial intelligence (AI) and machine learning (ML) solutions to unearth meaningful insights from their data. However, despite their willingness and investment, many buy-side firms are struggling to establish an efficient and programmatic way to do ML-based analytics at scale. Why are these efforts failing to meet expectations?

Making Sense of the Data

As asset managers push their competitive edge into the digital sphere, AI/ML solutions are growing out of the necessity for better and faster processes, calculated decisions, and data-driven insights. In the TRADE's October 2020 article "Buy-Side Lags behind Sell-Side in Adoption of Machine Learning and Artificial Intelligence," a Refinitiv study found that only 28% of buy-side firms were leveraging ML and AI (Smith 2020). This lag was recently confirmed by a Q1 2022 Quant Strats survey of 100 hedge funds, asset managers, and investment banks across the United States and Canada that pegged regular use of machine learning in decision making at 22% (Finadium 2022). Anecdotally, a different picture is emerging, one fueled by significant buy-side interest and active ML skill building and development.

Like the chicken and egg paradox, the buy side needs advanced technology and established efficient processes to capture and store data in sufficient granularity and requires ML expertise to help sift through the reams of structured and unstructured data. Though buy-side traders still consider sourcing liquidity their biggest challenge—one that statistical algorithms cannot easily solve—they recognize the advantages of ML-assisted approaches. Based on our observations, data delay and capture capabilities are improving and there is enthusiasm for putting existing data to work. ML's abilities to uncover patterns in large datasets and surface correlations that humans cannot detect have buy-side traders and portfolio managers learning about and turning to open-source ML libraries, such as scikit-learn, tools similar to those used by traditional data scientists. In turn, these budding quasi-data scientists are developing such ML-based solutions as trigger algorithms that can help inform a direction to take when unforeseeable events occur or ML-powered solutions that overcome limitations by using data gathered from other industries.

In this chapter, five use cases detail real-life examples of ML's application in the analytics and trading spaces, while noting both the effective combination of data and ML-based techniques used to enrich the decision-support processes and some of the current barriers facing buy-side firms as the technology matures. Here is a synopsis:

- **Use Case 1: Feature Importance.** Designed to reduce the number of inputs a trader needs to consider when selecting the optimal trading strategy, feature importance calculates a score or rank for each input in a model. It is common for trading blotters to contain dozens, if not hundreds, of different security and market metrics, so using an ML-based approach can help traders focus on the most meaningful factors.
- **Use Cases 2 and 3: Transaction Cost Analysis (TCA).** Portfolio managers and traders use TCA to learn from post-trade data and to improve future performance. In searching for trends about which strategies work, data are typically broken down by components, such as the size of the order or the market cap of the stock. In Use Cases 2 and 3, different ML-based approaches help extract additional insights from existing trade data to further enhance TCA analysis.
- **Use Case 4: Normalized and Unbiased Algo Trading.** The automated, randomized testing performed by algo wheels helps traders find the best trading algorithm across a variety of brokers; it is like the A/B testing an e-commerce site would run to find the best product. Just like a retailer needs to ensure consistency of service for its customers, a data analyst must be aware that, aside from the testing component itself, all testers receive the same experience. In comparing algo wheel orders, a trader must account for the different order characteristics and market characteristics that each broker receives. Traditional market impact models are built statistically, but now, based on the size and scope of available data, it is possible to handicap brokers using ML-based models.
- **Use Case 5: Trading Strategy Recommendation Model.** In Use Cases 1–4, traders had to select the optimal trading strategy. In Use Case 5, an ML-based

trading strategy recommendation model is offered with an option to override should the recommendation not align with the trader's human intuition and experience. Incidentally, when a trader chooses to override, this too is captured and can be analyzed to better enhance the model.

Not All Data Are Created Equal

Computers cannot make decisions as humans do. Human decisions are based on heuristics and cognitive biases against a broad field of attention to fully comprehend an event. The use of ML-based models depends heavily on data, and more specifically on granular data, as well as on the humans that build them. It is important to acknowledge that all ML analyses begin with data, but not all data are created equal. In addition to the variance in data quality, humans also add variability via the multiple decisions they make throughout the ML model construction process.

Advanced models, such as neural networks, make such firms as Google successful, but they are like black boxes; it is hard to explain why they come up with the results they do. In trading, this approach is not optimal. In accordance with data science best practices, model builders should consider the following questions carefully during and after model construction. As a matter of transparency, their responses should also be included for every model submitted into production.

- What data were included versus excluded, and why?
- Assuming the data were available, what would improve the model?
- What consistency checks and quality assurances were performed on the input datasets?
- What features were included and excluded, and why?
- What model was chosen, and why?

Use Case 1: The Role of Feature Importance in TCA for Auto-Order Routing

Each ML-based model created is an experiment since they fail as often as they succeed. Sometimes, one may be able to make a model operational, but the predictive accuracy is low or the model does not work owing to a technical problem. Nevertheless, ML techniques are evolving, reducing the amount of time spent on manual analysis.

The feature importance approach is often where most buy-side firms start their ML implementation journeys. When evaluating the performance of a portfolio, a portfolio manager typically performs an attribution analysis to determine which stocks contributed positively and negatively. The output information can be used to explain performance to an end client and/or to predict the viability of future investment decisions.

The process of TCA is a data-driven way traders and portfolio managers can benefit by reviewing past results. Historically, TCA has primarily been observational, dissecting the changes in trading costs into factors, such as liquidity, volatility, and spread. Ideally, the trader may leverage the enhanced post-trade information when execution planning and when demonstrating or confirming execution quality/best execution.

Although observational TCA is less statistical, it has proven to be an invaluable tool for bridging the consumer trust gap. Experience has made clear that it is essential to prove to your end audience how well their data and processes are understood before they can be convinced to adopt emerging technologies, such as AI and ML; TCA has been an effective catalyst toward ML-based analytics. Usually, once a client gains confidence in traditional TCA capabilities, their thinking leads them to ask whether anything is missing. For Harris Associates, their search led them to the feature importance approach for auto-routing execution, according to Jason Siegendorf, the firm's head of trading analytics:

> Harris Associates had previously set up a workflow within our execution management system to auto-route or send a subset of trades directly to a pre-defined algorithmic trading strategy, bypassing the trading desk. This enabled cash flows and other low risk orders to be traded very consistently and without trader bias in broker selection and allowed our traders to focus on the orders that they can have the most impact on by sourcing liquidity and deciding on the optimal execution strategy. We know that order size is an important consideration when deciding if an order is auto-routing eligible, but we also wondered if there might be other contributing factors we were missing out on. Applying the feature importance function confirmed our intuition and helped spotlight other meaningful order characteristics we typically pay less attention to.[1]

Typically, a human trader's execution decision making involves the synthesis and weighing of dozens of inputs to make instant decisions regarding how to best execute an order. However, brain processing studies show that interactions are limited to three, sometimes four, inputs (Cowan 2001). The feature importance approach assists human traders by statistically confirming observational

[1]Jason Siegendorf, personal communication with author.

TCA and human intuition by explaining subtle post-trade transaction cost patterns that may have otherwise gone undetected. It can also be used as a guide for pre-trade and auto-routing strategies.

The recommended ML-based approach for this use case was a supervised random forest model[2] (feature importance is built in natively); however, some buy-side clients may prefer the straightforward linear regression[3] approach. Most traders are already familiar with the linear regression coefficients, which can be used to determine the direction and impact of a specific feature.

The following steps taken in this use case can be implemented for any ML-based model.

- **Step 1: Cleansing and Normalizing Data**

  Any ML-based process must first identify the right data for the intended objective and determine whether any data normalization is required. Although not explored in this article, most of a model builder's time is spent on the data—data cleansing, normalizing, and removing outliers and blank values—in addition to selecting and normalizing features before they can be passed into the model.

- **Step 2: Chunking (Breaking Down) Data into Subsets for Easier Interpretation**

  The following examples illustrate how chunking (breaking down) data into smaller subsets enhances understanding:

  - **Have the drivers of trading costs changed over time?** Ideally, the data should be broken down into time periods (e.g., month, quarter) and a separate model should be run for each. When completed, the model builder can compare the feature importance for each distinct period and determine whether drivers of costs have shifted or stayed the same.
  - **What are the cost drivers between two types of order flow?** A trader may want to know the difference in transaction costs between single-stock versus program trading flow, so the data would be chunked into two respective datasets and modeled separately.
  - **Do transaction costs differ by fund type?** To understand the cost impacts on small-cap cap growth and large-cap cap value, for instance, a trader would need to segment the data into relevant attribute datasets and then run a model for each attribute to pinpoint the most important drivers of the transaction costs.

- **Step 3: Data Labeling and Testing and Training Datasets**

  As inputs, labeled datasets are required for the supervised model. Following the selection of which labeled dataset(s) to use, it is necessary to further separate the data into training and testing subsets. Model builders must reserve a portion of the data for testing to determine how well the model predicts with data it has not previously interacted with, also known as generalization.

- **Step 4: Selecting the Model's Features**

  When selecting the model's inputs, avoid using a 'kitchen sink' approach that includes every possible driver of transaction costs since this type of model does not perform well with highly correlated inputs. Several techniques exist to identify and remove the highly correlated features; it is a step in the process.

- **Step 5: Selecting the ML Model's Library and Training a Supervised Model**

  Among the open-source ML libraries, scikit-learn is a popular option, and since the use case involved predicting continuous trading costs rather than discrete ones, the RandomForestRegressor was chosen. Despite spending less time tweaking the model to improve accuracy, its predictive power remains a significant factor when analyzing the feature importance.

- **Step 6: Selecting a Feature Importance Approach**

  Finally, the model builder must identify the most influential inputs for model prediction. Here are a few techniques:

  - **Default scikit-learn's feature importance.** Informally looks at how much each feature is used in each tree within the training forest. **Exhibit 1** shows an example output from the scikit-learn feature importance module.
    - *Pros*: Single command to retrieve and fast
    - *Cons*: Can be biased to continuous features and high-cardinality categorical features; can only be run on the user's training dataset
  - **Permutation feature importance.** Shuffles a feature to see how much the model changes its prediction accuracy.
    - *Pros*: Can be run on both testing and training datasets
    - *Cons*: More computationally expensive and can overestimate the importance of correlated features

[2]A random forest grows multiple decision trees that are merged for more accurate predictions. The reasoning behind these models is that multiple uncorrelated models (i.e., individual decision trees) achieve much better performance as a group than individually.

[3]The simple linear regression forecasting method can be used to plot a trend line based on relationships between dependent and independent variables. In linear regression analysis, changes in a dependent variable are shown on the y-axis and changes in the explanatory variable are shown on the x-axis.

Exhibit 1. Selecting Feature Importance

Weight	Feature	
0.2904 ± 0.0302	tradeHorizon	
01733 ± 0.0100	tradedValue	**Expected Results Sample**
0.1663 ± 0.0131	historicalVolatility	The trading horizon is the most important factor influencing transaction costs, followed by traded value and historic volatility. Traders can use this to explain transaction costs post-trade and to inform them of the costs before placing an order in the market (pre-trade).
0.1217 ± 0.0137	participationRate	
0.0560 ± 0.0111	percentMDV	
0.0523 ± 0.0094	spread	
0.0310 ± 0.0042	marketCap	
0.0230 ± 0.0041	sector	
0.0064 ± 0.0053	securityType	
0.0022 ± 0.0013	side	
0 ± 0.0000	country	

Source: Virtu Analytics.

- **Drop column feature importance.** Manually drop one feature and examine model prediction accuracy.
  - *Pros*: Highly intuitive and accurate
  - *Cons*: Need to retrain the model each time, which can be resource-intensive
- **Feature importance computed with Shapley additive explanation (SHAP) values.** Uses the SHAP values from game theory to estimate how each feature contributes to the prediction.
  - *Pros*: Provides more information through decision and dependence plots and impact directionality
  - *Cons*: Can be computationally expensive and requires a separate library

Use Case 2: Semisupervised Learning to Cluster Similar Orders Where Clear Tags Are Missing

The lack of clear and consistent data tags is one of the biggest hurdles ML must overcome. Currently, vast amounts of data are either untagged, in free-form text, or hard to integrate into the core dataset. In this use case, unsupervised and semisupervised approaches are explored for populating tags where data are missing, and in the next use case, I examine these approaches for parsing free-form text.

The objective for this use case is to determine the type of commission rate when the data populating the commission-type tag are missing. The example is useful because it illustrates that model builders might have to get creative in how to use the data they have—solving the problem using a different tack.

ML-based techniques can be applied more broadly to identify segments of orders that are similarly structured across a range of partially or fully untagged attributes. Examples include the following:

- *In cases where the fund is not consistently tagged,* details such as the number of stocks, average market cap, average momentum, and volatility can be used to extrapolate the style of investment strategy.
- *In cases where a trading desk discretionary tag is not consistently tagged,* details around the order horizon, the type of algorithm used, the observed transaction costs, and the existence of a limit price can be used to extrapolate whether the buy-side trading desk had full discretion over the trading strategy of the order.

In contrast with the first use case, Use Case 2 has only a small segment of clearly labeled data, so model builders

Illustrative Example of Using Semisupervised Learning to Solve a Long-Running Data Tagging Issue

Any time data are sourced from multiple systems, such as from multiple trading systems, data tagging can become an issue. An example is that while some trading platforms easily capture the type of commission rate paid with a trade, many do not. Buy-side traders want to be able to compare their commission rates to a large peer-based database to get a sense of how much relative commission they are paying.

While not easily solvable with traditional observation-based analytics, semisupervised learning allows for the training of a model that can learn off clearly tagged examples and then predict the commission type for trades that do not have clear tagging. The buy-side trader can now compare her execution-only rate on an apples-to-apples basis to other investment managers.

could choose to implement an unsupervised or semisupervised approach.[4]

As previously noted, ML model construction requires significant data preparation, and in Use Case 2, this involved the parsing of the client–broker code as an extrapolated model feature. While the broker tag itself differs from firm to firm, many broker destinations indicate the category of commission paid—for instance, *brokerA_HT* and *brokerA_LT*—data that were parsed and tagged as inputs for the model (**Exhibit 2**).

If an unsupervised clustering approach is taken, then a model needs to be selected that allows for both continuous numerical information, such as the average commission rate paid, and discrete categorical information, such as the country the client traded in. Using a *k*-means algorithm for the model's purely numerical input and a *k*-modes algorithm for the model's purely categorical input is recommended. The model builder may also opt for Zhexue Huang's *k*-prototype clustering algorithm that combines both approaches into one (Huang 1998).

When the model is complete, the model builder will have an output cluster number that, in this example (**Exhibit 3**), indicates whether the record represents an execution-only or a non-execution-only commission rate.

The output cluster number can be used to report the execution-only commission rate; the use case goal was achieved. Nevertheless, good data science practices require model builders to review not only results but also the components of the model. In so doing, the modeler found that the dataset contained more labeled data than expected and re-ran the model, this time using a semisupervised approach (some labeled data and some unlabeled data).

In the semisupervised model approach, a small set of data labels was used to train a supervised classification model, and once trained, the model could be used to predict the

Exhibit 2. Broker Code Commission Parsed and Data Tagged

Client ID (categorical)	Country (categorical)	Turnover (numerical)	Avg. Commission Rate (cps) (numerical)	Broker Destination Code (categorical)	Parsed Broker Code Tag (categorical)
1	USA	$1,000,000	1	brokerA_LT	Execution Only
1	USA	$2,000,000	4	brokerA_HT	Not Execution Only
2	USA	$1,000,000	1.5	brokerabcd	Unknown

Source: Virtu Analytics.

[4]Semisupervised model learning involves a learning problem, as well as the algorithms created for the problem, that pertains to a small amount of labeled examples and numerous unlabeled examples from which the model learns and makes predictions on new examples.

Exhibit 3. Model Cluster Output Number Using Parsed and Data Tagged Broker Commission Code

Client ID (categorical)	Country (categorical)	Turnover (numerical)	Avg. Commission Rate (cps) (numerical)	Broker Destination Code (categorical)	Parsed Broker Code Tag (categorical)	Model Output Cluster #
1	USA	$1,000,000	1	brokerA_LT	Execution Only	1
1	USA	$2,000,000	4	brokerA_HT	Not Execution Only	2
2	USA	$1,000,000	1.5	brokerabcd	Unknown	1

Source: Virtu Analytics.

unlabeled portion of the dataset. Even though the model builder ultimately opted to use the semisupervised method, the fully unsupervised and semisupervised approaches yielded similar average execution-only rates. In ML-based data analysis, reproducibility is an important aspect of confirming that techniques and approaches are working as intended.

Use Case 3: Natural Language Processing to Parse Free-Form Text Fields into Data Tags

Frequently, the parsing of inputs is from free-form text, such as a portfolio manager's note to a buy-side trader that might include trading instructions and constraints that can impact a trade's outcome. The following are some examples:

- Target this order for the close
- Part of a prog, trade cash neutral
- Avoid impact, trade in line with volume

A human reader can easily parse this text into a few simple tags; however, to programmatically extract information from free-form text fields, model builders must be familiar with natural language processing (NLP). The goal in Use Case 3 is to automate what a human can understand on a more systematic and consistent basis, as follows:

- Close
- Program trade
- Go along

Useful in post-trade learning, parsed tags can provide a data-driven perspective on how the portfolio manager's instruction may have affected trading costs, and in execution planning, they can help the trader better understand what strategies work best based on the instructions received. NLP can be used against any free-form text field to help parse out additional information, even though our examination in this use case focuses on the portfolio manager's instructions.

Google's Bidirectional Encoder Representations from Transformers (BERT) model was chosen because it has the capability of enhanced context understanding because of its process of evaluating text in both left-to-right and right-to-left directions. As with other models, BERT is open source and has the advantage of being pretrained on an enormous dataset. Since its initial launch, BERT has been adapted into sub-language versions, such as FinBERT, which handles financial documents, and LEGAL-BERT, which handles legal documents. The Virtu Analytics team performs most of its NLP work on an internal JupyterHub, where they have installed the necessary libraries and built their codebase; however, AWS SageMaker and Comprehend can be much quicker to set up. BERT also offers the advantage of being fine-tuned using a relatively small number of labeled examples, which, due to the size requirements, can be labeled by humans manually.

At this point, BERT runs like any other supervised method, producing tagged predictions alongside probability.

Use Case 4: Transaction Cost/Market Impact Prediction

Leveraging market impact models, TCA determines the quality of the execution versus the expected cost. Currently, various transaction cost outcome prediction approaches are based on a combination of order characteristics, stock-level market data inputs, and a model calibration process that incorporates the results of realized trades.

Exhibit 4. Precleansed Data Noise in a Typical Dataset

Training Example	Ticker	Trade Time	Shares	Volatility	Spread	Observed Transaction Cost Outcome (implementation shortfall)
1	AAPL	3/1/2022 10:00 a.m.	1,000	9 bps	10 bps	5 bps
2	AAPL	3/1/2022 10:05 a.m.	1,100	8 bps	9 bps	20 bps

Source: Virtu Analytics.

Building an ML-based model off observed transaction costs can be challenging because of the data noise in a typical dataset and the lack of data labeling that hampers the parsing of text. **Exhibit 4** illustrates the problem with data noise in the use case's dataset.

The training example in Exhibit 4 shows different observed transaction cost labels despite nearly identical inputs for ticker, trade time, shares, volatility, and spread, which can result in model confusion. Data analysts are advised that there may be other factors not captured in traditional TCA analysis that can affect a trade, which can similarly confuse a model. Attempts to estimate market impact using Virtu's Global Peer database in its entirety were unsuccessful; however, when filtering off more homogeneous order flows, such as those from the algo wheel, useful estimates were obtained.

An algo wheel provides automated, randomized routing to brokers according to user-defined allocations. When using an algo wheel, the trader selects the desired algo strategy and sends the order. Then, the algo wheel's broker-allocation engine routes the order to a user-defined set of normalized algo strategies. Post-trade data are accumulated over time (on average, 300 orders per broker represent a useful dataset) to assess performance and guide future broker allocations.

An ML-based market impact prediction model can be applied to cross-firm flows (giving insight into how brokers perform relative to each other) or to firm-specific flows to account for specific nuances in their proprietary investment processes. Enrico Cacciatore, senior quantitative trader and head of market structure and trading analytics at Voya Financial, explains why this capability matters: "Performance is what determines which brokers receive more flow on our algo wheel, and we need to account for differences in order difficulty across the flow that each of our counterparties receives. The Machine Learning Market Impact model we subscribe to allows us to handicap our brokers while at the same time giving us a sense of what transaction costs competing firms are achieving."[5]

The random forest model was chosen to accomplish the use case goal for market impact prediction. Several other model types were tested, including forest-related variations, such as gradient-boosted trees, but a simple random forest implementation performed the best in this case. We have found that random forest models provide high accuracy for our use cases and off our specific dataset, which includes both numeric and categorical data, and the results are also quite easy to explain and troubleshoot.

Every model builder is advised to incorporate a third-party review as an integral step in the construction process. Non-model builders with knowledge of the dataset, ML, and subject matter should review the approach with the model builder(s)—with all participants being strongly encouraged to challenge the model based on the data, features, and model type selected.

Use case in point, during our Transaction Cost Impact Algo Wheel Model peer review, someone raised a potential endogeneity issue related to a few features that included order horizons. Even though the order horizon greatly improved the model's prediction accuracy, it is directly controlled by the broker. The purpose of a peer review is for users to have a common understanding of the model's objective so they can assess whether it performed as intended. In this use case, the peer review team agreed that the broker had too much control over the order horizon and the feature was removed from the model.

Use Case 5: Trading Strategy Recommendation

In Use Cases 1–4, ML-based approaches provided data-driven trade information but left optimal trading strategy selection to the buy-side trader. In this use case, the review closes with an ML-based trading strategy recommendation model. Even though this example invokes greater risk—a suboptimal trading strategy can result in a higher-cost trade—the reader should now have a better understanding of ML-based alternative approaches to

[5]Enrico Cacciatore, personal communication with author.

segment order flow when data tags are unclear, how to parse free-form text from an ML perspective, and how market impact estimates can be constructed solely from observed trades.

This use case's data represented all algorithmic trades captured across all brokers from Virtu's execution management system. Leveraging the model at a firm-specific level is possible; however, data requirements are more stringent as accuracy is emphasized when a recommendation is being submitted.

The feature importance approach was applied to identify the key data inputs for the model, which include the following:

- *Order attributes*, such as the ticker, side, sector, size, and market cap
- *Stock clustering and NLP techniques*, used to fill in the gaps, such as whether a limit price was set and whether an urgency setting was used for the algo trade
- *Real-time market condition metrics*, such as relative volume, volatility, and spread compared with historical distributions

Random forest models were then trained using historical data for each of the algorithm trading styles available to a buy-side trader, including implementation shortfall, VWAP, liquidity seeking, and dark.

Finally, the buy-side trader receives a prediction of the transaction cost outcome based on all available algorithms. Though it is possible to display only the winning, low-cost strategy, it is also useful to show the cost and variability of outputs across all strategies as part of the trader's implementation display. Separately, some trades will be marked as not-low-touch-eligible because of the high strategy estimates provided across all models.

Effects of Training with Biased Data

While not discussed previously, if the datasets used to train machine learning models contain biased data, then the model predictions will most likely be biased as well. If most of the training data for the volume-weighted average price (VWAP) strategy represented large-cap stocks, which inherently have lower transaction costs, this strategy could look incorrectly cheap when compared to other models that have a more representative training sample. There are several ways to deal with bias, and it is an important consideration for any model that is used directly in the trading space.

Conclusion

Although the use cases presented in this chapter are successful, many experiments do not perform as expected. To provide some balance, an example of a failed experiment would be the effort to build a stock clustering model that could inform a trading strategy. Though the experiment was successful in clustering stocks by characteristics, it could not tie them to a model that consistently informed how to trade them.

ML-based model building can sometimes be creative, should be collaborative, and is always iterative.

References

Cowan, Nelson. 2001. "The Magical Number 4 in Short-Term Memory: A Reconsideration of Mental Storage Capacity." *Behavioral and Brain Sciences* 24 (1): 87–114. www.researchgate.net/publication/11830840_The_Magical_Number_4_in_Short-Term_Memory_A_Reconsideration_of_Mental_Storage_Capacity.

Finadium. 2022. "Quant Strats Survey Shows ML Adoption Lagging, Third Party Spend Drops, Quantum Computing Tops Future Tech" (17 March). https://finadium.com/quant-strats-survey-shows-ml-adoption-lagging-third-party-spend-drops-quantum-computing-tops-future-tech.

Huang, Zhexue. 1998. "Extensions to the *k*-Means Algorithm for Clustering Large Data Sets with Categorical Values." *Data Mining and Knowledge Discovery* 2: 283–304. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.4028&rep=rep1&type=pdf.

Smith, Annabel. 2020. "Buy-Side Lags behind Sell-Side in Adoption of Machine Learning and Artificial Intelligence." The TRADE (27 October). www.thetradenews.com/buy-side-lags-behind-sell-side-in-adoption-of-machine-learning-and-artificial-intelligence/.

8. MACHINE LEARNING FOR MICROSTRUCTURE DATA-DRIVEN EXECUTION ALGORITHMS

Peer Nagy
ML Intern, Man Group
DPhil Student, Oxford-Man Institute, University of Oxford

James Powrie, PhD
Principal Quant, Man Group

Stefan Zohren, PhD
Principal Quant, Man Group
Faculty Member, Oxford-Man Institute, University of Oxford

Introduction

A central task in implementing any trading strategy is executing the trades required to reach a desired portfolio position. Typically, larger trades are required to change positions for funds with higher assets under management, and the larger a trade is, the more impact it tends to have on market prices. This impact can be measured as slippage (i.e., the difference between a reference price before the start of the trade and the prices at which trades are executed). To minimize this slippage cost, which can lead to a significant performance degradation over time, machine learning (ML) methods can be deployed to improve execution algorithms in various ways.

An execution problem is usually framed as follows. The execution algorithm is presented with a block of shares to buy or sell within a required time frame, which typically ranges from seconds to hours. To minimize the adverse impact of trades on the market price, this large order is split into smaller slices that are then executed over the available time horizon. The role of the algorithm is to choose an execution schedule that reduces the slippage bill as much as possible. If the order was not split up this way and distributed over time but instead was executed as a market order at the moment the trade instruction was presented, then large buy orders would push prices up, sell orders would push them down, or there might simply not be enough liquidity in the market at the time to complete the trade, leading to a less favorable slippage cost or an incomplete execution.

In any execution problem, there is a trade-off between the price impact and the risk of the price moving unfavorably over the execution horizon. If we are risk neutral and have no information on the direction the price is likely to move or on future trading activity, the optimal execution schedule involves partitioning all trades into smaller slices spaced evenly over the execution horizon. This strategy is referred to as TWAP (time-weighted average price) execution, which serves as a benchmark for more advanced execution algorithms to surpass. Using methods from stochastic optimal control, TWAP can even formally be shown to be optimal under these assumptions, so it constitutes a valid baseline (Almgren and Chriss 2000).

However, it might appear obvious that we can further improve execution if we have useful predictions of where prices might move in the short term. Another commonly used execution strategy makes use of information on trading volume, or market turnover, because a higher volume allows larger trades to be executed for the same amount of price impact. This type of strategy targets execution of a block of shares at the volume-weighted average price (VWAP) by splitting up execution over time proportionately to the expected volume profile. While expected volume is inherently a forward-looking variable, it shows patterns depending on the time of day and can be predicted reasonably accurately using ML models. The mathematical finance literature also shows that VWAP is an optimal execution strategy when volume information is available under the assumptions of risk neutrality and permanent market impact (Kato 2015; Cartea and Jaimungal 2016). In practice, we can enhance our execution performance in contrast to TWAP by adjusting trade sizes proportionately to expected volume. Estimating trading volume can be as simple as observing a volume profile by time of day or as complex as using deep learning models with a plethora of market features.

One role for ML algorithms in execution is therefore to compute forecasts of short-term price movements and expected volume that an execution algorithm can use to front- or back-load the execution schedule—in effect, locally speeding up or slowing down trading activity. For example, given a signal forecasting decreasing prices over the next few seconds, the algorithm would place a sell trade now rather than wait for the price to fall. This supervised learning—or more specifically, regression problem—has been approached using a variety of methods, ranging from statistical modeling to deep learning. It is possible to apply advanced ML techniques, such as

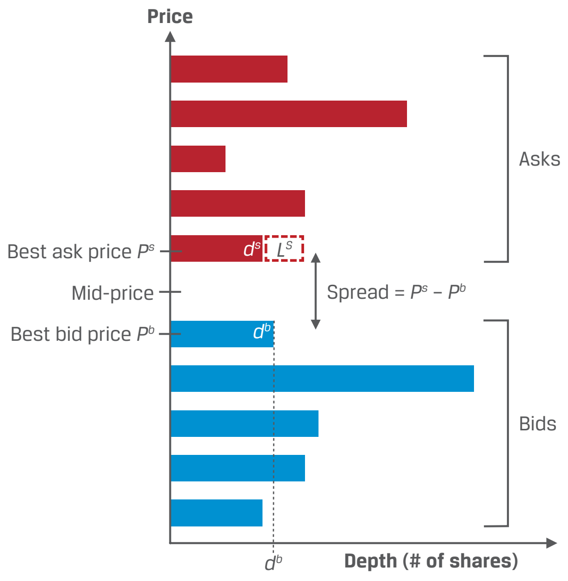
Jamuna deep learning, only because of the abundance of high-frequency data, which are essential for their training.

Predicting price movements directly is only one task of the many that ML can be applied to. For example, ML can also be used to forecast other relevant variables for optimizing the execution problem, such as market spreads, available future liquidity in the market, or volatility of the returns. Indeed, one can even have reinforcement learning algorithms directly choose concrete execution actions, such as sizing, timing, and pricing of slices, to solve the execution problem.

Microstructure Data and Analysis

Most electronic exchanges involved in the trading of cash equities, futures, or options use limit order books (LOBs) to match buy and sell orders using a price–time priority matching mechanism. In this prioritization scheme, orders are first matched by their price levels, with orders at the same price on a first-come, first-served basis. Every limit order to buy with a price lower than any sell order in the LOB is added to the bid side of the LOB, where levels are ordered from best (highest price and earliest arrival) to worst (lowest price and latest arrival). Similarly, if an order to sell is posted at a price higher than any order to buy in the LOB, it is added to the ask side of the LOB. Together, the bid and ask levels of the LOB constitute the visible supply and demand for any instrument and any moment in time (see **Exhibit 1**). The bid level with the highest price is called the best bid, and the ask level

Exhibit 1. Illustration of an LOB with Five Bid and Ask Levels



Note: The arrival of a new limit sell order of size L^s at the best ask is added to the book by increasing the depth at that level.

with the lowest price is called the best ask. The difference in price between the best ask and best bid is called the spread. If a market order is placed to buy (sell), it is first executed against the best price level of the ask (bid) side, then executed in order of arrival time of the corresponding limit order, and finally executed against the next level if the order is larger than the number of shares available at the best price. Rather than market orders, practitioners often use limit orders that are targeted against the best bid or ask. A refinement of this tactic is to use immediate-or-cancel (IOC) orders, which are automatically canceled if they cannot be filled at the limit price.

Given this order matching mechanism, LOBs are often described as double-sided continuous auctions, since a continuous order flow changes the book dynamically over time. Limit orders that cannot be matched immediately at the time of arrival, because either the bid is too low for a buy order or the ask is too high for a sell order, enter the book and provide liquidity to the market. Market orders, IOCs, or executable limit orders—buy (sell) limit orders with prices higher (lower) than the best ask (bid)—however, take away liquidity from the market, do not enter the LOB, and are executed immediately. In the case of executable limit orders, we can also have a situation where part of the limit order is executed, which thus removes liquidity, while a remainder stays on the book and provides liquidity.

To train ML models for execution tasks, LOB data are required in some form. These data can be represented at different levels of granularity. For most tasks in this space, a univariate price time series is insufficient, and at the least, a dataset with best bid and ask prices and volumes over time is required to learn useful relationships. Some ML models even train on data including multiple or all levels of the LOB to forecast variables of interest. Such datasets can be represented either as a stream of individual orders—so-called market-by-order (MBO) data—or as a sequence of snapshots of the state of the LOB over time. MBO data contain different order types (limit, market, cancel, and modify), which allow a dynamic reconstruction of the LOB in full fidelity. For modeling purposes, however, time series of features of the LOB state are usually more amenable to be used as model inputs.

As mentioned in the previous section, supervised ML techniques can be fruitfully applied to several prediction targets in the execution domain. Models that can predict future values of these variables over even part of the execution horizon can help reduce slippage. One such target is the market spread, because lower spreads imply more favorable execution prices for marketable orders crossing the spread. Similarly, high trading volume makes it more likely that a passively placed limit order will be executed at each time step. Conversely, forecasting return volatility can be useful for gauging the risk of large price moves over the execution horizon. In periods of high volatility, for example, it might pay to front-load the execution schedule to limit exposure to the downside risk of adverse price shocks.

Over the short term, one of the best indicators of immediate price moves in the LOB is the *order flow imbalance* (Cont, Kukanov, and Stoikov 2014). The definition of the order flow imbalance (OFI) is the order flow on the buy side: incoming limit buy orders at the best bid, L^b, net of order cancellations, C^b, and market orders, M^b, minus the opposing sell-side flow, within a period of time:

$$OFI = (L^b - C^b - M^s) - (L^s - C^s - M^b).$$

This measure captures an imbalance between demand and supply in the market, which is an essential determinant of the market price. A high positive order flow imbalance indicates excessive buy pressure at current prices, thereby making it more likely that prices will rise imminently. Cont et al. (2014) describe an empirical linear relationship between OFI and price changes.

A related LOB-derived measure of supply–demand imbalance, which can be used as a predictive signal of short-term price changes, is the *order book imbalance* (OBI):

$$OBI = \frac{d^b - d^s}{d^b + d^s}.$$

Here, d^b and d^s are the depths of the best bid and best ask, respectively (see Exhibit 1). The OBI calculates the normalized difference between the depth (number of available shares) on the buy side at the best bid, d^b, and the number of shares posted on the sell side at the best ask, d^s. This measure is limited between −1 and 1, ranging from a strong downward price pressure to a strong upward price pressure.

The order book imbalance is also often used by practitioners to calculate a *micro-price*, p^{micro}, which more closely reflects the microstructure effects than the mid-price. The micro-price simply weighs the bid and ask prices, p^b and p^s, respectively, by the imbalance, I:

$$P^{micro} = IP^s + (1 - I)P^b,$$

where

$$I = \frac{d^b}{d^b + d^s} = \frac{OBI + 1}{2}.$$

ML-Based Predictive Signals for Execution Algorithms

A classical approach in statistical modeling is to start out with simple, perhaps linear, models and a small set of variables, or features, that are likely to have some predictive power for the quantity of interest. Over the modeling process, model complexity is gradually increased and features are further engineered and refined to extract more information from the raw data. Driven by the domination of the ML literature by deep learning and artificial neural network (ANN) models, most recent approaches, however, have moved away from handcrafted feature engineering and instead approached prediction problems using raw data directly. This trend has also taken hold in financial ML and quantitative trading.

A recent exception to this rule is the deep order flow imbalance model (Kolm, Turiel, and Westray 2021), which uses order flow imbalance, as described in the previous section, to predict a vector of returns over multiple short-term horizons. The authors show that extracting order flow features from the top 10 levels of the order book is sufficient to achieve state-of-the-art predictive performance with relatively simple conventional neural network models, such as LSTM (long short-term memory) networks, or combinations of LSTMs with multilayer perceptrons (MLPs). This implies that practitioners might be able to get away with simpler models in some cases by performing an input data transformation from raw data to order flows. These results contrast with those of the same simple neural network models that instead use raw order book features, which cannot achieve any predictive power on the same task, implying that the data transformation is essential. However, another result of this study is that linear models and simple MLP feed-forward networks alone are not useful for forecasting such short-term alpha signals.

Notwithstanding deep order flow imbalance models, the trend in the current ML literature on forecasting short-term price signals points in the direction of using raw order book states directly, using more advanced ANN architectures to automatically extract feature representations amenable to forward return regression or classification tasks. One such model is DeepLOB (Zhang, Zohren, and Roberts 2019a), which uses a deep neural network architecture with convolutional layers and an inception module (Szegedy, Liu, Jia, Sermanet, Reed, Anguelov, Erhan, Vanhoucke, and Rabinovich 2015) also based on convolutional layers. Convolutional neural networks (CNNs) were originally developed for visual classification tasks, such as handwritten digit recognition or classifying images based on their content. The start of the current popularity of CNNs came with AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and its superior performance on the ImageNet challenge, a classification problem of a large database of images with more than 1,000 prediction classes. Convolutional layers act as local filters on an image, aggregating local information in every special region. During learning, the weights of many such filters are updated as the overall system learns to recognize distinct features in the data, such as horizontal or vertical lines, corners, and regions of similar contrast. LOBs can be likened to images because they also contain local information; for example, price and volume information at each level of the book can be combined with adjacent levels to automatically extract new features using convolutional layers. Similarly, CNNs can learn local temporal information by convolving order book states over the time dimension.

Building on this analogy between images and LOBs, the state-of-the-art deep learning model DeepLOB (Zhang et al. 2019a), which uses raw order book states directly as inputs to the network, constitutes a new tool in the execution toolbox. This type of model currently provides the most accurate short-term price signals, which can be used to improve execution trajectories. To improve the robustness of forecasts, DeepLOB can also be extended to perform quantile regression on the forward return distribution (Zhang, Zohren, and Roberts 2019b). To do this, the final LSTM layer of the network is split into multiple separate parallel parts for as many quantiles as should be forecast, and each network branch uses a corresponding quantile loss function. Quantile predictions can then be combined to compute more robust point estimates and add a measure of uncertainty and risk for the practitioner. Training separate models for the bid and ask sides has the additional advantage of producing estimates of the market spread, which can help in deciding the prices at which the order should be placed and whether the spread should be crossed.

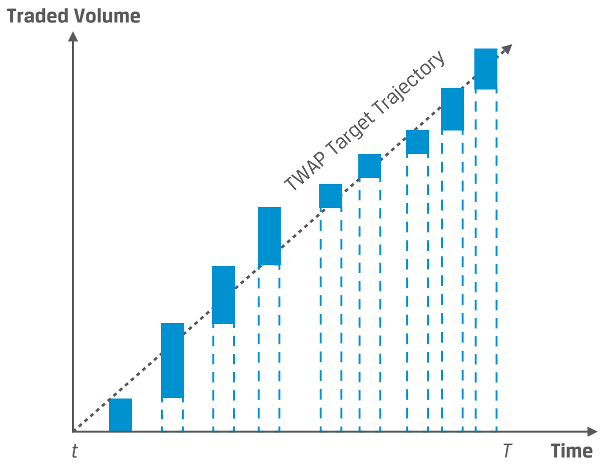
To improve the planning of an execution schedule, a point-in-time price forecast can be useful; ultimately, however, a price path over the execution horizon would be more beneficial for concrete planning. Using ideas from natural language processing and machine translation, a sequence-to-sequence model with an attention mechanism can be used to achieve such multihorizon return forecasts using LOB data (Zhang and Zohren 2021). To translate written text from one language to another, the idea of the sequence-to-sequence model (Sutskever, Vinyals, and Le 2014) is to use an LSTM encoder to learn a representation of a sentence as a fixed-length vector and then use a separate LSTM-based decoder to again translate this vector representation into the target language. Adapting this idea to predict return trajectories, the model in Zhang and Zohren (2021) uses the DeepLOB network (Zhang et al. 2019a) as an encoder, while an *attention mechanism* (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin 2017) allows using selected hidden states of the encoder layers in the decoding step, producing the forecast time series. This way, the model drastically improves forecasting performance over longer time horizons as forecasts sequentially build on each other in an autoregressive fashion.

Because deep learning models benefit from large amounts of data, in the extreme case, models could even be trained on market-by-order messages directly. Given that deep order flow imbalance models already showed the advantage of this more stationary data source, using all the raw data in a deep learning framework is a logical next step.

From Integrating Predictive Signals into Execution Algorithms to Automatically Learning Policies Using Reinforcement Learning

An execution strategy in an LOB market can be viewed as essentially two dimensional. One dimension is how very large order volumes are sliced up over time, while the other dimension describes the prices at which slices are placed in the order book. A simple TWAP strategy might, for instance, split a large order uniformly over time and always place marketable limit orders by crossing the spread (see **Exhibit 2**). This would imply, however, that half the spread is lost with every trade relative to the mid-price, incurring slippage costs even if the mid-price stays constant over the entire execution time. In contrast, placing trades passively by using limit orders at the near touch (i.e., without crossing the spread) has the opposite effect of earning half a spread whenever a marketable order is placed in the opposing direction. The downside of passive orders is that execution is uncertain and depends on other market participants' order flow and future price moves. Thus, if one had a directional signal of future prices, it would be sensible to deviate from the simple TWAP strategy. For example, if the execution task is to sell a block of shares, a favorable price signal might indicate that the mid-price is expected to rise; hence, a slice of the order could be placed more passively deeper in the book (at a higher price), anticipating that a price swing

Exhibit 2. Simple TWAP Strategy



Note: This exhibit shows an illustration of an execution schedule with a linear TWAP target trajectory and executed discrete slices as blocks over time around the target trajectory.

could complete the trade. If price forecasts point downward, however, a market order could still make use of the higher price by executing a slice before the move happens.

Using a combination of ML models, we can thus engineer a complete execution strategy. An example algorithm might work as follows. Volatility forecasts, obtained using any method from historical averaging over generalized autoregressive conditional heteroskedasticity (GARCH) models to deep learning, can be used to schedule either the time allowed for an execution ticket or the amount of front-loading over a fixed time horizon by controlling how execution slice sizes decrease over time. Using ML forecasts of trade volume, we can further modulate execution schedules by proportionately changing future slice sizes with expected volume forecasts. Predicted price paths can then be used to fine-tune the strategy by varying placement levels (prices) dynamically. Should we expect a favorable price move, we would place a passive limit order in the book at the first level—or even deeper into the book if the expected price move is sufficiently large. Such a strategy could be refined in various ways—for example, by using a probabilistic level placement to increase the probability of passive placements with more favorable price moves and wider spread forecasts. A probabilistic strategy also has the benefit of being less predictable by other market participants, who might want to exploit the effect that large orders have in moving the market.

Another ML approach encompassing the execution problem uses reinforcement learning (RL) algorithms to plan the execution trajectory. RL—and especially deep RL using deep neural networks—has been tremendously successful in solving complex problems, from learning to play games (Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, and Riedmiller 2013) to predicting three-dimensional protein structures from genomic data (Jumper, Evans, Pritzel, Green, Figurnov, Ronneberger, Tunyasuvunakool, Bates, Žídek, Potapenko, et al. 2021). In the execution context, Ning, Lin, and Jaimungal (2021) describe a good example using deep RL for optimal execution problems using double Q-learning (Hasselt 2010), employing a modern algorithm previously used to train autonomous agents to play Atari games at a superhuman level from raw pixel input.

On a theoretical level, the execution problem can be framed as a partially observable Markov decision process (POMDP), which can be amenable to being solved using RL algorithms. The RL learning paradigm works analogously to biological learning processes in animals and humans. The agent, our execution algorithm, interacts with a market environment in state s, which describes all information necessary to characterize the current state of the market, by performing an action, a, thereby transitioning the state to $s' = T(s,a)$. The environment state is further assumed to satisfy the Markov property, which means that past states do not add any further relevant information for the future. The agent, however, perceives not the entire state of the world but only an observation, $o' = obs(s')$, and hence does usually not know the underlying state exactly. In addition to the new observation o' at each step, the learner also receives a reward signal, r. Based on the reward signal, the RL algorithm learns over time which sequence of actions leads to the highest expected cumulative rewards.

Observational features for the RL algorithm might include various data from the LOB, including such handcrafted features as order flow imbalance and order book imbalance, or even specific model predictions, such as future price paths, spreads, or volatility. In the extreme case, using deep RL can help one even learn policies directly from raw order book data. Rewards in this scenario are usually based on incurred slippage during training.

A difficulty in real-world RL applications is that training the algorithm necessarily must be done in a simulated environment. The most basic kind of "simulator" simply uses historical market prices. This approach limits the action space to timing market orders, because past prices alone cannot determine whether a limit order would have been executed or not. Another shortcoming of relying solely on historical prices for simulation is that trades do not generate any market impact, because neither do they take away liquidity in the book nor can they cause any other market participant to react to the trade. To alleviate the latter problem, simulation environments are sometimes enhanced with stochastic models of price impact to represent more realistic costs of aggressive trading. Another approach to help with the former problem of handling limit orders is to model the environment as a complete market replay using market-by-order data. This way, new market or even limit orders can be injected into the historical order flow. This approach accurately handles how the order would have been executed at the time.

However, this alone does not solve the problem of counterfactual behavior by other agents. For example, if one of our orders *is* executed, it might imply that someone else's order was *not* executed. They then might have placed another order at a different price; however, this is not represented in the historical data. One approach to handle these counterfactual scenarios is agent-based modeling, which represents individual traders explicitly in a simulation. These simulated agents follow their own trading strategies and can react to changes in the market caused by our execution algorithm, as well as to actions and reactions of other agents. Capturing realistic trading behavior remains a challenging task, and building realistic LOB models is the subject of active research.

Conclusion

How large trades are optimally executed depends on the details of a market's microstructure environment. We have described price–time priority LOB markets, because this is the most common market design at major exchanges trading cash equities, futures, and options. Limit order books thus provide a wealth of high-frequency data that can be used for developing data-driven execution algorithms using ML methods. LOB data can be used to engineer informative features for the prediction of a range of relevant variables: spreads, trade volume, volatility, and even short-term price movements (fast alpha signals). Separate ML models, each predicting a different variable, can then be combined into a sophisticated execution algorithm that plans an execution trajectory in both the volume and placement level (price) dimension.

The trend in ML research more generally points in the direction of using growing computer resources to further automate feature extraction from raw data instead of relying on handcrafted features. The DeepLOB model from Zhang et al. (2019a) and further models building on it are good examples demonstrating that this trend also holds in finance and particularly for trade execution. Given expansive LOB datasets and current computer power, deep learning models are already outperforming simpler models in many tasks, such as generating fast alpha signals. Taking trade automation a step further, RL offers an appealing framework to reduce slippage costs by letting an execution algorithm learn to take optimal actions directly. Actions can be defined on varying levels of abstraction—from choosing parameters in an existing execution algorithm dynamically to placing trades outright. The research outlook in ML for execution shows a path toward more complete end-to-end execution systems using more advanced deep learning and (deep) RL algorithms and architectures, improving predictive performance while maintaining critical issues, such as model robustness.

References

Almgren, Robert, and Neil Chriss. 2000. "Optimal Execution of Portfolio Transactions." *Journal of Risk* 3 (2): 5–39.

Cartea, Álvaro, and Sebastian Jaimungal. 2016. "A Closed-Form Execution Strategy to Target Volume Weighted Average Price." *SIAM Journal on Financial Mathematics* 7 (1): 760–85.

Cont, Rama, Arseniy Kukanov, and Sasha Stoikov. 2014. "The Price Impact of Order Book Events." *Journal of Financial Econometrics* 12 (1): 47–88.

Hasselt, Hado. 2010. "Double Q-Learning." In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, edited by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. https://papers.nips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. 2021. "Highly Accurate Protein Structure Prediction with AlphaFold." *Nature* 596: 583–89.

Kato, Takashi. 2015. "VWAP Execution as an Optimal Strategy." *JSIAM Letters* 7: 33–36.

Kolm, Petter, Jeremy Turiel, and Nicholas Westray. 2021. "Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book." Working paper (5 August). Available at https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3900141.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning." NIPS Deep Learning Workshop 2013. Cornell University, arXiv:1312.5602 (19 December). https://arxiv.org/abs/1312.5602.

Ning, Brian, Franco Ho Ting Lin, and Sebastian Jaimungal. 2021. "Double Deep Q-Learning for Optimal Execution." *Applied Mathematical Finance* 28 (4): 361–80.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. "Sequence to Sequence Learning with Neural Networks." *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. https://papers.nips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. "Going Deeper with Convolutions." *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. "Attention Is All You Need." *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Zhang, Zihao, Stefan Zohren, and Stephen Roberts. 2019a. "DeepLOB: Deep Convolutional Neural Networks for Limit Order Books." *IEEE Transactions on Signal Processing* 67 (11): 3001–12.

Zhang, Zihao, Stefan Zohren, and Stephen Roberts. 2019b. "Extending Deep Learning Models for Limit Order Books to Quantile Regression." Time Series Workshop of the 36th International Conference on Machine Learning. Cornell University, arXiv:1906.04404 (11 June). https://arxiv.org/abs/1906.04404.

Zhang, Zihao, and Stefan Zohren. 2021. "Multi-Horizon Forecasting for Limit Order Books: Novel Deep Learning Approaches and Hardware Acceleration Using Intelligent Processing Units." Cornell University, arXiv:2105.10430 (27 August). https://arxiv.org/abs/2105.10430.