Quantitative Finance

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/rquf20

Learning a functional control for high-frequency finance

L. Leal, M. Lauriere & C.-A. Lehalle

To cite this article: L. Leal, M. Lauriere & C.-A. Lehalle (2022) Learning a functional control for high-frequency finance, Quantitative Finance, 22:11, 1973-1987, DOI: 10.1080/14697688.2022.2106885

To link to this article: https://doi.org/10.1080/14697688.2022.2106885

Published online: 23 Aug 2022.

Quantitative Finance, 2022
Vol. 22, No. 11, 1973–1987, https://doi.org/10.1080/14697688.2022.2106885

Learning a functional control for high-frequency finance

L. LEAL *†, M. LAURIERE† and C.-A. LEHALLE‡

†Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, USA
‡Capital Fund Management, Paris, and Imperial College London, 23 rue de l'Université, Paris 75007, France

(*Received 18 February 2021; accepted 29 June 2022; published online 23 August 2022*)

We use a deep neural network to generate controllers for optimal trading on high-frequency data. For the first time, a neural network learns the mapping between the preferences of the trader, i.e. risk aversion parameters, and the optimal controls. An important challenge in learning this mapping is that, in intra-day trading, trader's actions influence price dynamics in closed loop via the market impact. The exploration–exploitation tradeoff generated by the efficient execution is addressed by tuning the trader's preferences to ensure long enough trajectories are produced during the learning phase. The issue of scarcity of financial data is solved by transfer learning: the neural network is first trained on trajectories generated thanks to a Monte-Carlo scheme, leading to a good initialization before training on historical trajectories. Moreover, to answer to genuine requests of financial regulators on the explainability of machine learning generated controls, we project the obtained 'blackbox controls' on the space usually spanned by the closed-form solution of the stylized optimal trading problem, leading to a transparent structure. For more realistic loss functions that have no closed-form solution, we show that the average distance between the generated controls and their explainable version remains small. This opens the door to the acceptance of ML-generated controls by financial regulators.

Keywords: Market microstructure; Optimal execution; Neural networks; Quantitative finance; Stochastic optimization; Optimal control

JEL Classification: C450, C580, C610, G170, C630

1. Introduction

Financial mathematics are using stochastic control to ensure that market participants are operating as intermediaries and not as unilateral risk takers: investment banks have to design risk replicating strategies, systemic banks have to ensure they have plans in case of fire sales triggered by economic uncertainty, asset managers have to balance risk and returns in their portfolios and brokers have to be sure investors' large buy or sale orders are executed without distorting the prices. The latter took a primary role in post-2008 markets since participants understood preservation of liquidity is of primary importance.

Our paper addresses this last case, belonging to the academic field of optimal trading, initially introduced by Bertsimas and Lo (1998) and Almgren and Chriss (2001), and then extended in a lot of ways, from sophisticated stochastic control (Bouchard *et al.* 2011) to Gaussian-quadratic approximations allowing to obtain closed-form solutions like (Cartea and Jaimungal 2016) or (Cartea *et al.* 2015), or under a self-financing equation context, as in Carmona and Webster (2019). Learning was introduced in this field either via on-line stochastic approximation (Laruelle *et al.* 2013) or in the context of games with partial information (Cardaliaguet and Lehalle 2018). More recently, Reinforcement Learning (RL) approaches have been proposed, like in Guéant and Manziuk (2019) to face the high dimensionality of optimal trading of portfolios, or in Mounjid and Lehalle (2019) to adapt the controls in an online way; see also (Charpentier *et al.* 2020) for an overview.

For optimal control problems in continuous time, the traditional approach starts with the cost function and derives, through dynamic programming, a Hamilton-Jacobi-Bellman (HJB) equation involving the optimal control and the value function (that is the optimum of the Q-function in RL). This Partial Differential Equation (PDE) can be explicitly written only once the dynamics to be controlled are stylized enough. When it does not have a closed form solution, this PDE can be solved either by a deterministic method (like a finite difference scheme) that is limited by the dimension of the problem,

*Corresponding author. Email: lleal@princeton.edu or by RL approaches, like in Buehler *et al.* (2019) for deep hedging or in Tian *et al.* (2019) and Guéant *et al.* (2020) for optimal trading. In contrast with Tian *et al.* (2019) mentioned above, we focus less on the use of signals in algorithmic trading and more on the risk control dimension, going into neural network *explainability* aspects that, to the best of our knowledge, have not been explored before.

In the broader literature, deep learning techniques for PDEs and Backward Stochastic Differential Equations (BSDEs) have recently attracted a lot of interest (E *et al.* 2017, Han and Jentzen 2018, Sirignano and Spiliopoulos 2018, Huré *et al.* 2019, Raissi *et al.* 2019) and found numerous applications such as price formation (Sirignano and Cont 2019), option pricing (Liu *et al.* 2019) or financial systems with a continuum of agents (Fouque and Zhang 2019, Carmona and Laurière 2019a, 2019b).

In this paper, we propose to skip most of the previous steps: we directly learn the optimal control on the discrete version of the dynamics and of the cost function that are usually giving birth to the HJB, which we no longer have to derive. This approach, also used e.g. in Gobet and Munos (2005) and Han and E (2016) for optimal control, gives us the freedom to address dynamics for which deriving explicitly a PDE is not possible, and to learn directly on trajectories coming from real data. In optimal trading, the control has a feedback loop with the dynamics via price impact: the faster you trade, the more you move the price in a detrimental way (Bacry *et al.* 2015). In our setup the controller sets the trading speed of an algorithm in charge of buying or selling a large amount of stock shares, solving the trade-off between trading fast to get rid of the exposure to uncertainty of future prices and trading slow to not pay too much in market impact. At a high level, we would like to underline that our contribution is methodological rather than focused on a specific example. The 1-dimensional application with a closed-form solution is used as an illustration.

For numerical applications we use high frequency (HF) data from the Toronto Exchange (TSX) on 19 stocks over two years, each of them generating around 6500 trades (i.e. data points) per day. We compare different controls: controls generated by a widely known stylized version of the problem that has a closed form formula (Cartea *et al.* 2015), controls learned on simulated dynamics whose parameters are estimated on the data in different ways (with or without stationary assumptions), and controls learned on real data. For the latter case, we transfer the learning from simulated data to real data, to make sure we start with a good initialization of the controller.

In our setup, a controller maps the state space describing price dynamics and the trader's inventory to a trading speed that leads to an updated version of the state, and so on until the end of the trading day. The same controller is used at every decision step, which corresponds to one decision every 5 min. When the controller is a neural net, it is used multiple times before the loss function can be computed and improved via back propagation, in the spirit of Lehalle and Azencott (1998).

Our first main contribution is that we not only train a neural net for given values of the end-user's preferences, but we also train a neural net having two more inputs that are the trader's preferences (i.e. risk aversion parameters), so that this neural net is learning the mapping between the preferences (i.e. hyper-parameters of the control setup) and the optimal controls. Hence, for a different day or a different trader, if risk aversion parameters have changed, we don't need to re-train the network, we can just input the new parameters to the control and start trading. To the best of our knowledge, it is the first time that a neural net is performing this kind of 'functional learning' of risk aversion parameters: the neural net learns the optimal control for a range of cost functions that are parametrized by the risk preferences. In the paper, we call it a '*multi-preferences neural net*'.

The second major contribution of this paper is the way we compare the generated controls in a meaningful way through *model distillation*. Model distillation and, more precisely, model translation methods seek to replicate the DNN behavior across an entire dataset (see Tree-based Frosst and Hinton 2017, Tan *et al.* 2018, Zhang *et al.* 2019, Graph-based Zhang *et al.* 2016, 2017 and Rule-based Murdoch and Szlam 2017, Harradon *et al.* 2018 methods). Our paper introduces a model translation method focused on the idea of global approximation (see Tan *et al.* 2018 for a discussion on global additive explanations).

Our global distillation approach paves the way to methods which satisfy the request of financial regulators about the *explainability of learned controls*. We start by the functional space of controls spanned by the closed-form solution of the stylized problem: they are non-linear in the remaining time to trade and affine in the remaining quantity to trade (see Cardaliaguet and Lehalle 2018 for a description of the relationship between the optimal controls and the space generated by the $h_1(t)$ and $h_2(t)$ defined later in the paper). Hence, we project the learned controls on an affine basis of functions, for each slice of remaining time to trade $T-t$. In doing so, we provide evidence of the distance between the black-box controls learned by a neural network, and the baseline control used by banks and regulators. We can compare the controls in this functional space and, thanks to the R^2 of the linear regressions, we know the average distance between this very transparent representation of the controls and the ones generated by the neural controllers. In practice, we show that when the loss function is mainly quadratic, the learned controls, even when they are trained on real data, are spanning roughly the same functional space as the closed-form ones. End-users and regulators can consider them with the same monitoring tools and analytics than more standard (but sub-optimal) controls. When the loss function is more realistic but not quadratic anymore, taking into account the mean reverting nature of intra-day price dynamics (Labadie and Lehalle 2014), the generated controls may or may not belong to the same functional space. We underline that, once the risk is taken into account and trading begins, the control being used is coming from the neural network setup, not the projection.

The structure of the paper is as follows: Section 2 presents the optimal execution problem setup, focusing on the loss function and on the closed-form formula associated with the classic optimal execution problem. It also introduces the architecture of the neural networks we use and our learning strategies. Section 3 describes the dataset, stylized facts of intra-day price dynamics that should be taken into account, and motivates learning in a data-driven environment. Section 3 also presents the numerical results and our way to tackle explainability of the generated controls. Conclusions and perspectives are provided in Section 4.

2. The optimal execution model

2.1. *Dynamics and payoff*

Optimal trading is dealing with an agent who would like to execute a large buy or sell order in the market before a time horizon T. Here, their control is a trading speed $\nu_t \in \mathbb{R}$. The center of the problem is to find the balance between trading too fast (and hence move the price a detrimental way and pay trading costs, as modeled here in (1) and (3)) and trading too slow (and being exposed to not finishing the order and be exposed to uncertainty of future prices, reflected in (2) and (4)). They maximize their wealth while taking into account a running cost and a final cost of holding inventory, and they are constrained by the dynamics of the system, which is described by the evolution of the price S_t, their inventory Q_t (corresponding to a number of stocks), and their wealth X_t (corresponding to a cash amount). In the market, the price process for the asset they want to trade evolves according to

$$\Delta S_t := S_{t+1} - S_t = \alpha_t \nu_t \Delta t + \sigma \sqrt{\Delta t} \epsilon_t, \qquad (1)$$

for $t = 0, \ldots, T-1$, where $\alpha_t > 0$ is a parameter that accounts for the drift, Δt is the size of a time step, $\sigma > 0$ is a constant volatility term, taken to be the historical volatility of the stock being traded. For transfer learning on historical data, the database contains real price increments and hence the realized volatility is (implicitly) of the same order of magnitude. $\epsilon \sim \mathcal{N}(0, 1)$ is a noise term.

The state of the investor at time t is described by the tuple $(T-t, Q_t, X_t)$. Q_t is their inventory, or number of stocks, and X_t is their wealth at time t, taken as a cash amount. To isolate the inventory execution problem from other portfolio considerations, the wealth of the agent at time 0 is considered to be 0, i.e. $X_0 = 0$. Suppose the agent is buying inventory (the selling problem is symmetric). Then, $Q_0 < 0$ and the trading speed ν will be mostly positive throughout the trajectory. The evolution of the inventory is given by:

$$\Delta Q_t := Q_{t+1} - Q_t = \nu_t \Delta t, \quad \text{for } t = 0, \ldots, T-1. \qquad (2)$$

The wealth of the investor evolves according to

$$\Delta X_t := X_{t+1} - X_t = -\nu_t (S_t + \kappa \cdot \nu_t) \Delta t, \qquad (3)$$

for $t = 0, \ldots, T-1$, where $\kappa > 0$ is a constant, and the term $\kappa \cdot \nu_t$ represents the temporary market impact from trading at time t. As we can see, this is a linear function of ν_t that acts as an increment to the mid-price S_t. It can also be seen as the cost of 'crossing the spread', or a transaction cost.

The cost function of the investor fits the framework considered e.g. in Cartea *et al.* (2015). The agent seeks to maximize over the trading strategy ν the reward function given by:

$$J_{A,\phi}(\nu) := \mathbb{E}\left[X_T + Q_T S_T - A|Q_T|^\gamma - \phi \sum_{t=0}^{T} |Q_t|^\gamma \right], \qquad (4)$$

with $\gamma > 0$. We mostly focus on the standard case $\gamma = 2$, but we will also consider the case where $\gamma = 3/2$, which better takes into account the sub-diffusive nature of intra-day price dynamics, see Labadie and Lehalle (2014). X_T, S_T, and Q_T are the random variables parameterizing the terminal value of the stochastic processes described in the dynamics (1)–(3). $A > 0$ and $\phi > 0$ are constants representing the risk aversion of the agent. A penalizes holding inventory at the end of the time period, and ϕ penalizes holding inventory throughout the trading day. Together, they parameterize the control of the optimal execution model and stand for the agent's preferences.

The trading agent wants to find the optimal control ν for the cost (4) subject to the dynamics (1)–(3). In the sequel, we solve this problem using a neural network approximation for the optimal control. As a benchmark for comparison, we will use the closed-form solution for the corresponding continuous time problem, which is derived through a PDE approach. The closed-form solution of the PDE is well-defined only when the data satisfies suitable assumptions. The neural net learns the control ν_t directly from the data, while finding a closed-form solution for the PDE is not always possible.

When $\gamma = 2$, it can be shown, see Cartea and Jaimungal (2016), that the optimal control ν^* is obtained as a linear function of the inventory, which can be written explicitly as:

$$\nu^*(t, q) = \frac{h_1(t)}{2\kappa} + \frac{\alpha + h_2(t)}{2\kappa} q, \qquad (5)$$

where h_2 and h_1 are the solutions of a system of Ordinary Differential Equations (ODEs).

2.2. *Non-parametric solution*

2.2.1. The neural network setup. The neural network implementation we propose precludes all the usual derivations in the Cartea and Jaimungal (2016) framework. We no longer need to find the PDE corresponding to the stochastic optimal control problem, we no longer need to break it down into ODEs and solve the ODE system, and we can try to directly approximate the optimal control. The deep neural network approximation looks for a control maximizing the agent's objective function (4) while being constrained by the dynamics (1)–(3), without any further derivations.

We define *one single neural network $f_\theta(\cdot)$ to be trained for all the time steps*. Each iteration of the stochastic gradient descent (SGD) proceeds as follows. Starting from an initial point (S_0, X_0, Q_0), we simulate a trajectory using the control $\nu_t = f_\theta(t, Q_t)$. Based on this trajectory, we compute the gradient of associated cost with respect to the neural network's parameters θ. Finally, the network's parameters are updated based on this gradient. In our implementation, the learning rate is updated using Adaptive Moment Estimation

(Adam) (Kingma and Ba 2014), which is well suited for situations with a large amount of data, and also for non-stationary, noisy problems like the one under consideration.

In the Monte-Carlo simulation mode, we generate random increments of the Brownian motion through the term $\sigma\sqrt{\Delta t}\epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$ comes from the standard Gaussian distribution. Using the Gaussian increments, along with the ν_t obtained from the neural network, and the three process updates $(\Delta S_t, \Delta X_t, \Delta Q_t)$, we can compute the new state variables in discrete time:

$$S_{t+1} = S_t + \alpha_t f_\theta(t, Q_t)\Delta t + \sigma\sqrt{\Delta t}\epsilon, \tag{6}$$

$$X_{t+1} = X_t - f_\theta(t, Q_t)(S_t + \kappa_t f_\theta(t, Q_t))\Delta t, \tag{7}$$

$$Q_{t+1} = Q_t + f_\theta(t, Q_t)\Delta t. \tag{8}$$

The new state variable Q_{t+1}, along with the time variable, is then going to serve as input to help us learn the control at the next time step. This cycle continues until we have reached the time horizon T. Since we are using the same neural network f_θ for all the time steps, we expect it to learn according to both time and inventory. Note that each SGD iteration requires T steps because we need the neural net to have controlled the T steps before being able to compute the value function for the full trajectory. It then backpropagates the error on the terminal value function across the weights of the same neural net we use at each of the T iterations.

2.2.2. Neural network architecture and parameters. The architecture consists of three dense hidden layers containing five nodes each, using the hyperbolic tangent as activation function. Each hidden layer has a dropout rate of 0.2. We add one last layer, without activation function, that returns the output. The learning rate is $\eta = 5e^{-4}$; the mini-batch size is 64; the tile[†] size is 3; and the number of SDG iterations is 100,000. Every 100 SGD iterations, we perform a validation step in order to check the generalization error, instead of evaluating just the mini-batch error. We used Tensorflow in our implementation.

Although the neural network's basic architecture is not very deep, each SGD step involves T loops over the same neural network. This is because of the closed loop which the problem entails. The number of layers is thus artificially multiplied by the number of time steps considered.

For the inputs, we have two setups. In the basic setup (as discussed above), the input is the pair (t, Q_t). In the second case, which we call '*multi-preferences neural network*', the input is the tuple (t, Q_t, A, ϕ) and the neural network learns to maximize the $J_{A,\phi}(\cdot)$ cost functions for all (A, ϕ). Each time we need to solve a given system, we set A and ϕ to the desired value in the multi-preferences network and we do not need to relearn anything to obtain the optimal controls. For both neural nets, the output is the speed of trade ν_t for each time step t, intercalating with the variable updates, thus learning a controller for the full length of the trading day.

In order to learn from historical data, we first train the network on data simulated by Monte Carlo and then perform transfer learning on real data.

[†] The tile size (ts) stands for how many samples of the inventory we combine with each sampled pair. For $ts = 3$, each pair $(S_0, \{\Delta W_t\}_{t=0}^T)$ becomes three samples: $(Q_0^j, S_0, \{\Delta W_t\}_{t=0}^T)$, for $j \in \{1, 2, 3\}$, where j is the tile index. This is useful when using real data, because we can generate more scenarios than we would ordinarily be able to.

2.3. Explainability of the learned controls using projection on a well-known manifold

We would like to compare the control obtained using the PDE solution with the control obtained using the neural network approximation. As stated in equation (5), when $\gamma = 2$, the optimal control can be written as an affine function of the inventory q_t at any point in time, for $t \in [0, T]$. The shape of this closed-form optimal control belongs to the manifold of functions of t and q spanned by $[0,T] \times \mathbb{R} \ni (t,q) \mapsto h_1(t)/(2\kappa) + (\alpha + h_2(t))/(2\kappa) \cdot q \in \mathbb{R}$, where the $h_i(t)$ are non-linear smooth functions of t. To provide *explainability of our neural controller* we project its effective controls on this manifold. We obtain two non-linear functions $\tilde{h}_1(t)$ an $\tilde{h}_2(t)$ and a $R^2(t)$ measuring the distance between the effective control and the projected one at each time step t.

2.3.1. Procedure of projection on the 'closed-form manifold'. For each t, we form a database of all the learned controls ν_t mapped by the neural net to the remaining quantity q_t. It enables us to project this ν_t on q_t using an Ordinary Least Squares (OLS) regression whose loss function is given by:

$$L(\beta_1(t), \beta_2(t)) = \|\nu(q(t)) - (\beta_1(t) + \beta_2(t)q(t))\|_2^2, \tag{9}$$

and do a global translation of our neural network control.

The coefficients $\beta_1(t)$ and $\beta_2(t)$ of this regression $\nu_t = \beta_1(t) + \beta_2(t)q_t + \varepsilon_t$ can be easily inverted to give

$$\tilde{h}_1(t) := 2\kappa\, \beta_1(t), \quad \tilde{h}_2(t) := 2\kappa\, \beta_2(t) - \alpha \tag{10}$$

for each $t \in [0, T]$. The $R^2(t)$ of each projection associated to t quantifies the distance between the effective neural 'black box' control and the explained one $\tilde{\nu}_t := \tilde{h}_1(t)/(2\kappa) + (\alpha + \tilde{h}_2(t))/(2\kappa) \cdot q_t$. The curve of $R^2(t)$ represents how much of the non-linear functional can be projected onto the manifold of closed-form controls at each t. In practice we perform $T = 77$ OLS projections to obtain the curves of h_1, h_2 and R^2 (see figure 2) which provide explainability of our learned controls.

2.3.2. Users' preferences and exploration-exploitation issues. The rate at which the agent executes the whole inventory strongly depends on the agent's preferences A and ϕ. When they are both large, the optimal control tends to finish to trade earlier than T, the end of the trading day. Because of that, the trajectories of Q_t approach zero in just a few time steps and after that large values of Q_t are not visited during the training phase. As a consequence, in such a regime, it will be very difficult for the neural net to learn anything around the end of the time interval since it will no longer *explore* its interactions with price dynamics to be able to emulate a better control. For the case of the multi-preferences controller with 4 inputs, including A and ϕ, taking (A, ϕ) in a certain range ensures that the neural net will witness long enough trajectories to *exploit* them. To this end, we took profit of the closed-form solution of the stylized dynamics and scanned the duration of trajectories for each pair (A, ϕ). In order to avoid the exploration—exploitation trade-off, we restricted the domain to values of ϕ smaller or equal to 0.007, and values of A smaller or equal to 0.01. This enabled the regression to learn the functional produced by the neural network. There will be some inventory left to execute at the end of the trading day for these parameters. For the pair $(A, \phi) = (0.01, 0.007)$, we have less than 1% of the initial inventory left to execute, yet it is enough for us to estimate the regression accurately.

3. Optimal execution with data

3.1. Data description

We use ticker data for trades and quotes for equities traded on the Toronto Stock Exchange for the period that ranges from Jan/2008 until Dec/2009, which results in 503 trading days. Both trades and quotes are available for Level I (no Level II available) for 19 stocks. They have a diverse range of industries, and daily traded volume. Our method can be directly used for all the stocks, either individually or simultaneously.† Table 1 summarizes the database information.

We partition both trade and quote files into smaller pickled files organized by date. We merge them day by day to calculate other interesting variables, such as: trade sign, volume weighted average price (VWAP), and the bid-ask spread at time of trade. We drop odd-lots from the dataset, as they in fact belong to a separate order book. Moreover, we only keep the quotes that precede a trade, or a sequence of trades. It saves us memory usage and processing times.

The trade data, including the trade sign variable, is further processed into five-minute bins. We have picked five-minute intervals because, while the interval could be changed, here we want to make sure the data is not too sparse, and to provide additional risk control to the agent over a time horizon of one full day. TAQ data is extremely asynchronous and a situation might happen when there is not enough data in a given time interval. To avoid this problem, we aggregate the data into bins and focus our analysis on liquid stocks. This bin size is big enough for us to have data at each bin, and small enough to allow the agent to adjust the trading speed several times throughout the day. Since the market is open from 9:30 until 16:00, five-minute bin intervals, we have 78 time steps, which gives us 77 bins for which to estimate the control on each trading day. We have 91.5% of bins containing data (see table 1).

When training the model, we set the initial value for the wealth $X_0 = 0$ to isolate the execution problem from portfolio considerations. The initial price S_0 can either be set *ad hoc* to some reasonable value, such as \$100 for simulations, or, in the case of real data, to the initial price of the stock for a given day. σ is taken to be the historical volatility of the stock price, calculated on 5-minute bins. For the initial inventory Q_0, we train the network to execute large proportions of the average daily volume. We therefore take Q_0 to be uniform in $[-.2, -.04] \cup [.04, .2]$, and multiply it by the average daily volume for the stock.

† For the market impact parameters (α, κ), we estimate the same value for all the stocks through a normalization, then we denormalize the parameters using the seasonality profile of the stock used in the model.

Table 1. Descriptive statistics of the TSX stocks used.

Average # Trades per Day:	124,988.4
Average # Trades per Stock:	3,308,902.2
Permanent Market Impact α:	$0.16 \cdot \frac{\text{avg.binspread}}{\text{avg.binvolume}} \cdot \frac{1}{dt}$
Temporary Market Impact κ:	$0.24 \cdot \frac{\text{avg.binspread}}{\text{avg.binvolume}} \cdot \frac{1}{dt}$
% of 5-min Bins w/ Data:	91.5%

3.2. Important aspects of financial data

The stylized dynamics allowing a PDE formulation of the problem and a closed-form solution of the control make a lot of assumptions on the randomness of price increments, market impact and trading costs. Typically they assume independent and normally distributed, stationary, non-correlated returns, with no seasonality. However, these properties typically do not hold true for financial time series. See Campbell *et al.* (1993) and Bouchaud *et al.* (2018) for some interesting discussions.

Figure 1 shows the specific characteristics of financial data for the stock MRU. The left plot is a qq-plot, which tells us that the stock returns have a heavy tailed distribution. The same is confirmed for all stocks: we observe high kurtosis (ranging from 163 to 11302) for all the distributions of stock returns (which is a common metric for saying that a distribution has heavy tails with respect to a Gaussian distribution). The middle plot presents the five-minute lag auto-correlation profile for one particular day of MRU, namely January 2nd, 2008. For this stock, we can see a mean-reversion tendency for five minutes and one hour. The values outside the blue band are statistically significant at a 95% confidence level. When considering the full range of days available, and each of the stocks, the first lag of the five-minute auto-correlation (ranging from -0.017 to -0.118) indicates that this feature is present across the board. The right plot shows intra-day seasonality profiles: the bid-ask spread (blue, dashed curve), and the intra-day volume curve (full, gray curve).

As mentioned in Section 2.3, the first step is to compare the output of our neural network model to the PDE solution, both using Monte Carlo simulated data. The second step is to understand how seasonality in the data might affect the training and the output. Finally, we continue our learning process on real data, in order to learn its peculiarities in terms of heavy tails, seasonality and auto-correlation.

3.3. PDE vs DNNs: from Monte Carlo to real data

The results and our benchmark are summarized in figure 2. As stated in equation (5), the optimal control for the stylized

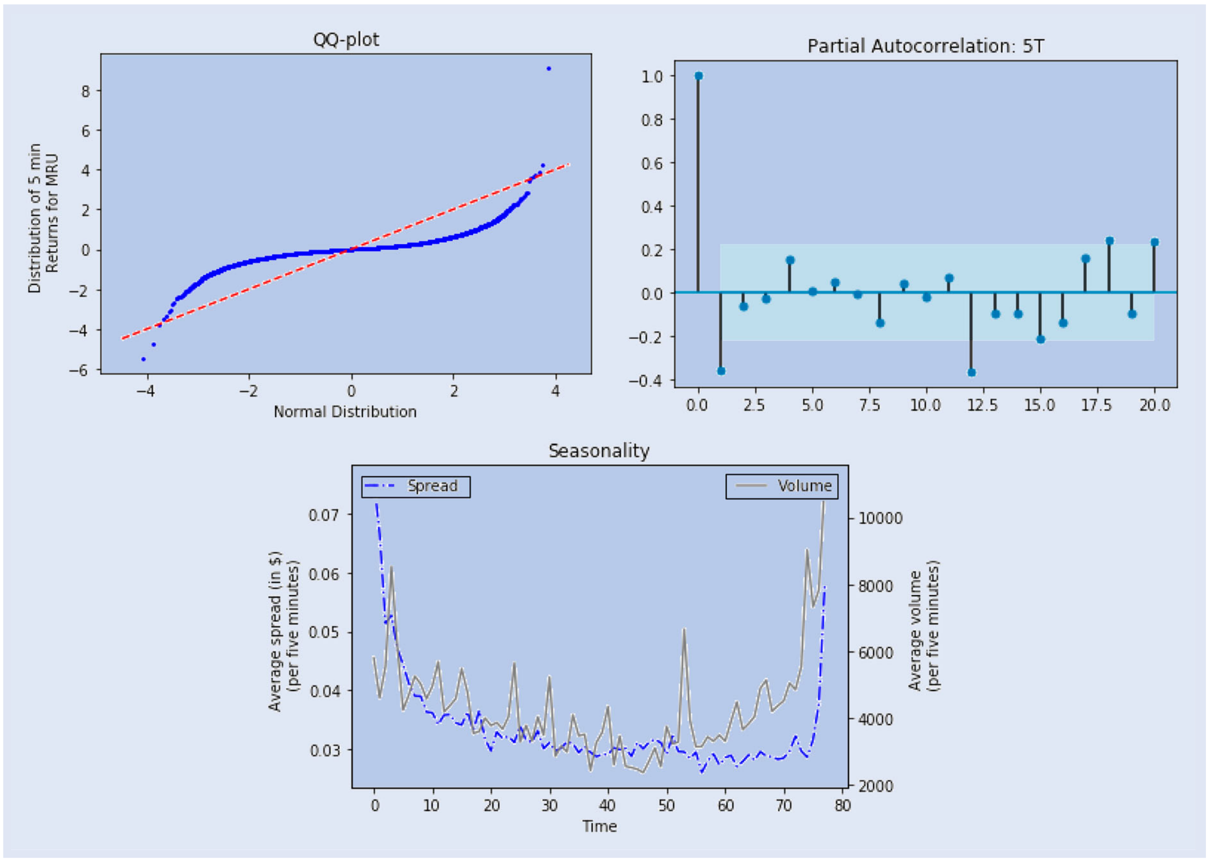

Figure 1. QQ-plot (top left), auto-correlation function (top-right), intra-day seasonalities (bottom) of price dynamics.

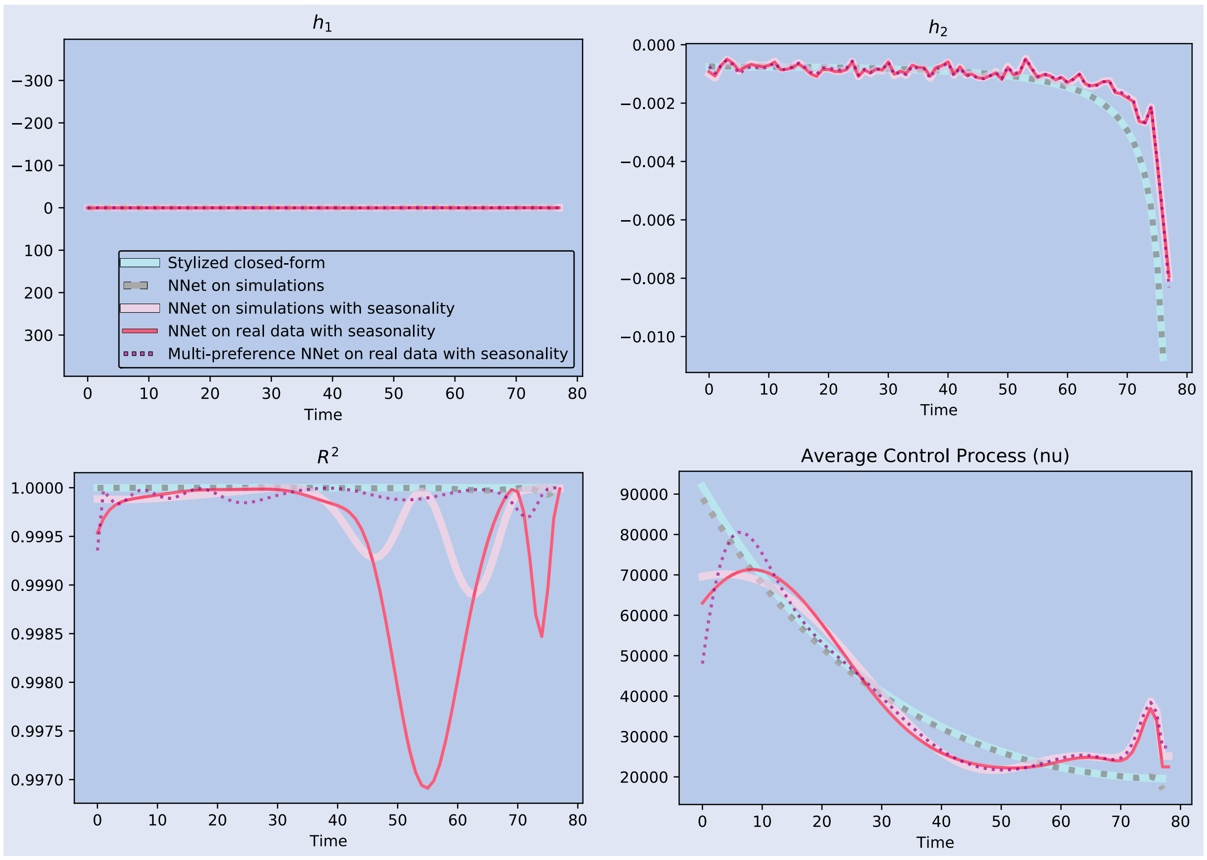

Figure 2. Explainable parts of the control: $h_1(t)$, $h_2(t)$; fraction of the explained control $R^2(t)$, and average control given it is positive $\mathbb{E}(\nu(t) \mid \nu(t) > 0)$ for different controllers.

dynamics and the PDE is linear in the inventory, hence the associated R^2 is obviously 1 for any t. For the different neural nets, we perform the projections on the (h_1, h_2) manifold and keep track of the R^2 curve.

During the learning, it is important that the neural network can observe full trajectories. When the risk aversion parameters are very restrictive, the closed-form solution and neural net trades so fast that the order is fully executed (i.e. the control stops) far before $t = T$. Because of that it is impossible to learn after this stopping time corresponding to $q_t = 0$. Our workaround for this exploration—exploitation issue has been to use the closed-form solution to select a range of (A, ϕ) allowing the neural net to generate enough trajectories to observe the dynamics up to $t = T$. In figure 2, we use the pair $(A, \phi) = (0.01, 0.007)$ with $\gamma = 2$. We stress that, once the model has been trained on a variety of pairs (A, ϕ), we can then use it with new pairs of parameters.

In this figure 2, we mix closed-form controls on the stylized dynamics, neural controls on the same dynamics (using Monte Carlo simulations), neural controls in more realistic simulations (with an intra-day seasonality) and ultimately on real data. In particular, we use stock data (via transfer learning, see the thick purple and solid pink lines), and in order to make the model as close as possible to reality in a version that is in-between synthetic data and transfer learning (dotted pink curve). We include intra-day seasonality for both volume and bid-ask spread, and we emphasize how it is reflected in the learned control. We also superpose results for neural networks trained only on this regime of preferences and the controls of the multi-preference neural network, that is trained only once and then generates controls for any pair (A, ϕ) of preferences. The top panels represent the two components of the projection of the control on the 'closed-form manifold', to enable comparison. The plot on the upper left of figure 2 corresponds to the h_1 component. The y-axis scale is in the order of the median of the inventory path, to allow for a fair comparison to $h_2 \cdot Q$. It is naturally very close to zero, since there is no mean-field component in the market impact term. The upper right plot corresponds to the h_2 component. We notice that the stylized closed-form and the neural network simulation using Monte Carlo generated paths match perfectly, thus setting a great benchmark for the neural network. The improvements we add to the neural network model, simulations including seasonality, real data with seasonality and multi-preferences all overlap, indicating that the intra-day seasonality component is the most relevant when explaining differences in optimal execution speed when departing from the closed-form stylized model. The $R^2(t)$ curves (bottom-left panel) are flat at 1 for the closed form formula (since 100% of the control belongs to this manifold), whereas it can vary for the other controls (see figure 3, and discussion in Section 3.4). Thanks to this projection, it is straightforward to compare the behavior of all the controls. The $R^2(t)$ curves provide evidence that in this regime the neural controls are almost 100% explained by the closed-form manifold (notice that the y-axis has a very tight range close to 1). It does not say that they are similar to the closed-form solutions of the stylized problem, but that they are linear in q_t (but not in t) for each time step $t \in [0, T]$.

Saving the best for last, we present the optimal control for each different configuration in the bottom left plot of figure 2. We again stress the similarity between the neural network model on simulations versus the closed-form benchmark. We further observe the differences between the optimal controls for the improved setups. When we start taking into account the intra-day seasonality, the execution speed adjusts according to the volume traded and to the bid ask spread. It results in less trading when the spreads are large, and in more trading when there is more volumes. In adjusting to market conditions, we are able to improve on the existing execution benchmark.

3.3.1. Main differences between the learned controls. Going from stylized dynamics to simulations with intra-day seasonality, and then to real price dynamics does not significantly change the multiplicative term of q_t, but it does shift the trading speed. This has been already observed in the context of game theoretical frameworks (Cardaliaguet and Lehalle 2018). When the seasonality is added in the simulation (thick purple curve) the deformation of the control exhibits oscillations that most probably stem from the seasonalities of the bottom panel of figure 1. The shift learned on real price dynamics (solid pink) amplifies some oscillations that are now less smooth; this is probably due to autocorrelations of price innovations. Moreover figure 2 shows that the 'functional learning' worked: the mono-preference neural network trained only on this (A, ϕ) and the multi-preferences one have similar curves.

3.4. Nonlinear aspect of DNN controllers

For non quadratic costs (we took $\gamma = 3/2$ since it is a realistic case, see Labadie and Lehalle 2014) we take $(A, \phi) = (0.5, 0.1)$ to be in a comparable regime as the controls in figure 2, and $(A, \phi) = (0.01, 0.007)$ which results in a non-linear control. Notice that here we are fixing the risk aversion parameters, instead of activating the multi-preference controller. The solid blue $R^2(t)$ curve in the bottom left of figure 3 being again close to one shows that for this combination of parameters, the learned control remains extremely similar to the closed-form manifold. This kind of evidence can allow risk departments of banks and regulators to get more comfortable with this learned control and it can give guidelines to large institutional investors, such as pension funds, to execute their orders in the market with the view of controlling risk in an explicit fashion.

When $\gamma = 3/2$, the chosen set of risk parameters has more influence in the learned control, which can now either be linear or non-linear. For the pair $(A, \phi) = (0.01, 0.007)$, the absolute value of 'proportional term' h_2 is very close to zero when using the same scale as the h_2 when $(A, \phi) = (0.5, 0.1)$. Nevertheless, it is compensated by the part of the control that is outside of this space, as indicated by the R^2: at the start of the trading process 40% of the control is explained by the 'closed-form manifold'; it then decreases to 0%, meaning that the learned control takes part fully out of this manifold; but it then increases back to 80% at the end of the process. It seems that the best way to finish the trading is already well

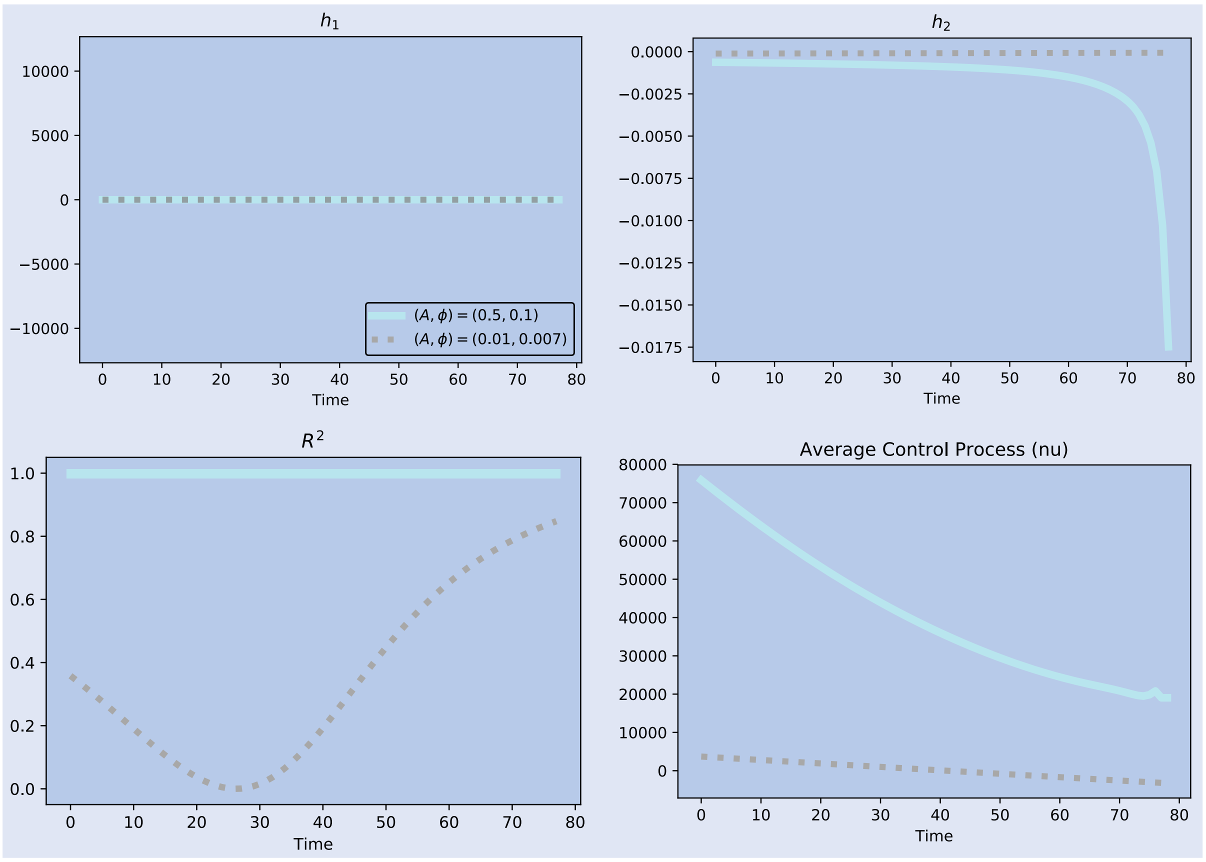

Figure 3. h_1, h_2, R^2, ν: for the case $\gamma = 3/2$, with fixed (A, ϕ) parameters indicated on the legend.

captured by the closed-form manifold. As we observe from the control plot in the bottom-right of figure 3, the parameter pair $(A, \phi) = (0.01, 0.007)$ does not enforce trading speeds as fast as the pair $(A, \phi) = (0.5, 0.1)$ when the loss function is sub-diffusive. Keeping the same preferences as before is thus making the trader less aggressive in a sub-diffusive environment. In order to have the same behavior in terms of control, they would need to behave in a more risk averse manner.

3.5. *Comparison to closed-form solution*

In figure 4, we compare the performance of the control learnt by the neural network with closed-form control. For the sake of illustration, we focus here on two cases: $\gamma = 2$ and $\gamma = 3/2$, using parameters $(A, \phi) = (0.01, 0.007)$. Given q_0 and a choice of control, using Monte Carlo samples we compute an approximation of the reward function (corresponding to (4) conditioned on $Q_0 = q_0$) and the marked-to-market final wealth in cash. We plot individual results stemming from point-wise simulation results starting at different Q_0.

In the case $\gamma = 2$, using the neural network control (with or without seasonality, and also in the multi-preference setting) almost matches the optimal one obtained by the closed-form solution. When $\gamma = 3/2$, the control stemming from the neural network performs better than the commonly used control coming from the closed-form solution $\gamma = 2$, thus we advocate that a more realistic parameter can improve the execution of the agent (recall that the goal is to maximize the reward function). We stress that while the reward function is asymmetric due to the intrinsic asymmetry in equation (3), the marked-to-market wealth defined by $\mathrm{MTM}_T = \mathrm{sign}(Q_0)(Q_0 S_0 - X_T)$ is symmetric except for a sign adjustment.

3.6. *Empirical verification of DNN explainability*

Figure 5 displays the DNN control and relative error in the neural network regression, namely, if $(t, q) \mapsto f_\theta(t, q)$ and $(t, q) \mapsto \tilde{f}_\theta(t, q)$ denote respectively the trained neural network and approximate version obtained by regression, we compute: $(t, q) \mapsto |(f_\theta(t, q) - \tilde{f}_\theta(t, q))/\tilde{f}_\theta(t, q)|$. The maximum relative error for the case $\gamma = 2$ is 0.04, while for the case $\gamma = 3/2$ it is 0.35. We see that for most time steps, the relative error is small in the bulk of the distribution of q_t. However, around the terminal time = 78, the distribution is very concentrated around $q = 0$, which explains why the regression is less accurate.

4. Conclusion

In this paper, we succeed in using a neural network to learn the mapping between end-user preferences and the optimal trading speed, allowing to buy or sell a large number of shares or contracts on financial markets. Prior to this work, various proposals have been made but the learned controls have always been specialized for a given set of preferences. Here, our *multi-preferences neural network* learns the solution of a class of dynamical systems.

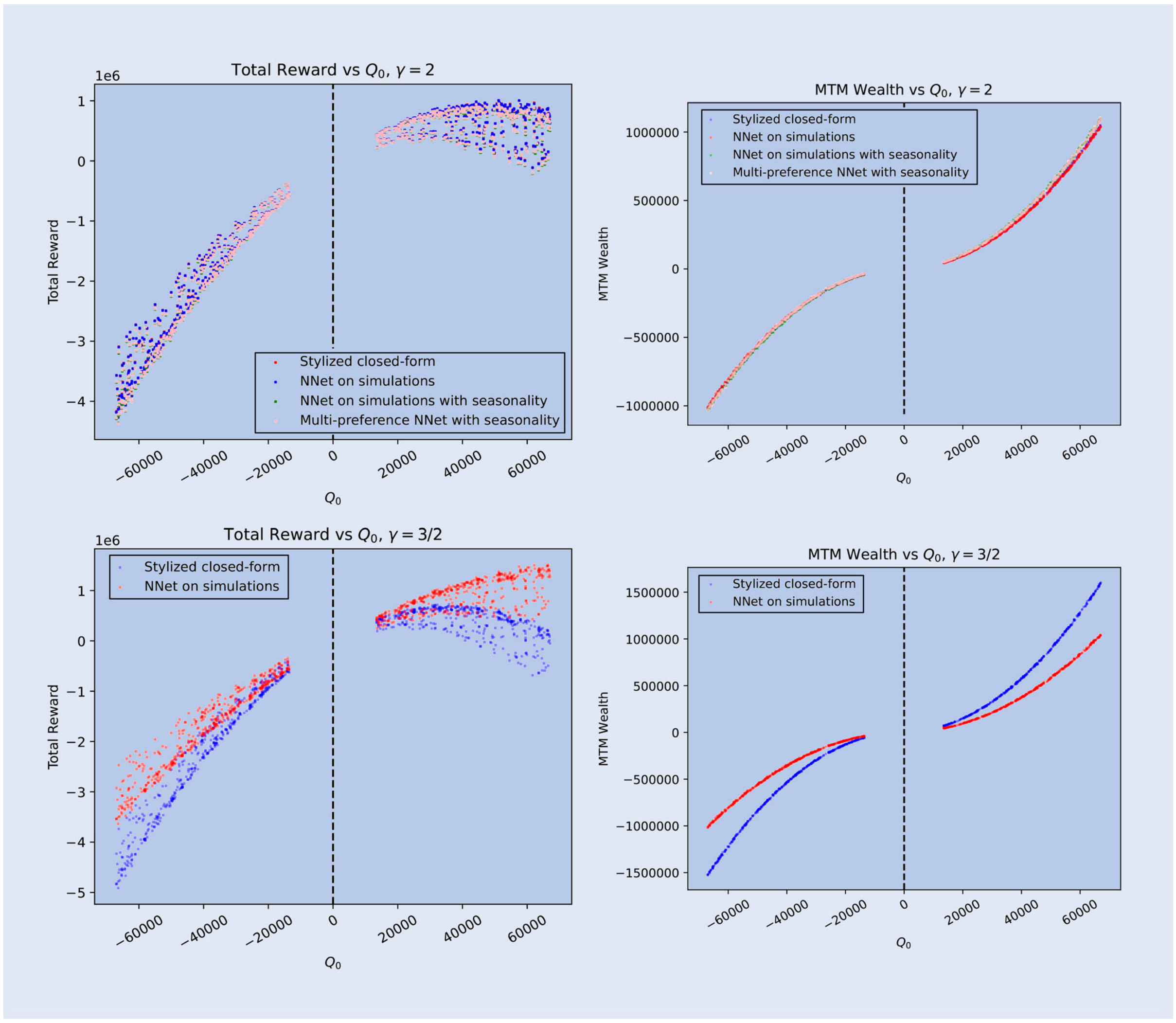

Figure 4. Value Functions (left) and Marked-to-Market Wealth (right): for $\gamma = 2$ (top) and $\gamma = 3/2$ (bottom), with respective neural network setup indicated in the legend.

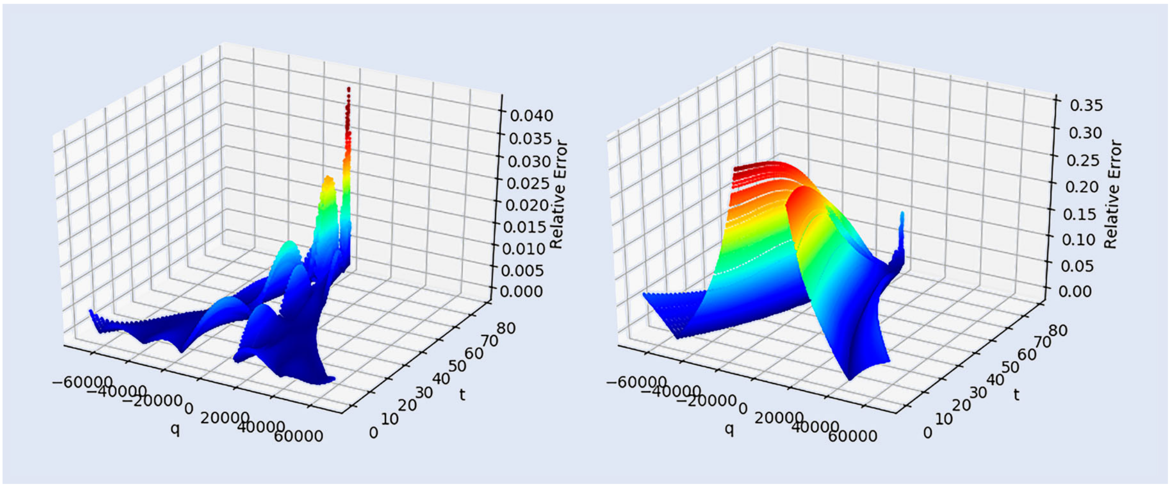

Figure 5. Relative error in the regression: $\gamma = 2$ (left) and $\gamma = 3/2$ (right).

Note that optimal execution dynamics are reacting to the applied control, via the price impact of the buying or selling pressure. The neural network hence has to learn this feedback. Our approach uses a deep neural network, whose inputs are user preferences and the state of the optimization problem that change after each decision. The loss function can only be computed at the end of the trading day, once the neural controller has been used 77 times. The backpropagation hence takes place across these 77 steps. We faced some exploration—exploitation issues and solved it by choosing suitable the range of users' preferences to ensure that long enough trajectories are observed during the learning.

Our setup leverages on transfer learning, starting on simulated data before switching to historical data. Since we want to understand how the learned controls are different from the closed-form solution on a stylized model that are largely used by practitioners, we learn on different versions of simulated data: from a model corresponding to the stylized one to one incorporating non-stationarities, and then to real data.

To ensure the explainability of our learned controls, we introduce a projection method on the functional space spanned by the closed-form formula. It allows us to show that most of the learned controls belong to this manifold. When we depart from the traditional model by incorporating a risk aversion term, the learned control is close to its projected version on the bulk of the distribution, supporting the idea that the projection explains a significant part of the neural network control. Furthermore, the source of adaptation to realistic dynamics focuses on a 'shift term' h_1 in the control. The most noticeable adaptation of h_1 exhibits slow oscillations that are most probably reflecting the seasonalities of intra-day price dynamics.

We then introduce a version of the loss function that reflects more the reality of intra-day price dynamics (that are sub-diffusive). Despite the fact that the associated HJB has no closed-form solution, we manage to learn its associated optimal control and show, using our projection technique, that it almost belongs to the same manifold. This approach delivers explainability of learned controls and can probably be extended to contexts other than optimal trading. It should help regulators to have a high level of trust in the learned controls, that are often considered as 'black boxes': our proposal exposes the fraction of the controls that belongs to the manifold practitioners and regulators are familiar with, allowing them to perform the usual 'stress tests' on it, and quantifies this fraction as a R^2 curve that is easy to interpret. More details on the ethical impact and broad societal implications are discussed in Appendix 3. The main components of the code are available at: https://github.com/arula10101/HFT_NN_paper.

Open Scholarship

This article has earned the Center for Open Science badge for Open Materials. The materials are openly accessible at https://github.com/arula10101/HFT_NN_paper.

Acknowledgments

The authors are extremely grateful for the fruitful discussions with professor René Carmona, who was also very kind to provide the data we used.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

The work of L. Leal and M. Laurière was supported by NSF [grant number DMS-1716673] and ARO [grant number W911NF-17-1-0578].

ORCID

L. Leal http://orcid.org/0000-0002-9489-8354

References

Almgren, R. and Chriss, N., Optimal execution of portfolio transactions. *J. Risk*, 2001, **3**, 5–40.
Bacry, E., Iuga, A., Lasnier, M. and Lehalle, C.A., Market impacts and the life cycle of investors orders. *Market Microstructure Liquidity*, 2015, **1**, 1550009.
Bertsimas, D. and Lo, A.W., Optimal control of execution costs. *J. Financ. Markets*, 1998, **1**, 1–50.
Bouchard, B., Dang, N.M. and Lehalle, C.A., Optimal control of trading algorithms: A general impulse control approach. *SIAM J. Financ. Math.*, 2011, **2**, 404–438.
Bouchaud, J.P., Bonart, J., Donier, J. and Gould, M., *Trades, Quotes and Prices: Financial Markets Under the Microscope*, 2018 (Cambridge University Press: Cambridge).
Buehler, H., Gonon, L., Teichmann, J. and Wood, B., Deep hedging. *Quant. Finance*, 2019, **19**, 1271–1291.
Burgess, M., Australia pension funds fear early access may compound pressure. Bloomberg News, 2020.
Campbell, J.Y., Grossman, S.J. and Wang, J., Trading volume and serial correlation in stock returns. *Q. J. Econ.*, 1993, **108**, 905–939.
Cardaliaguet, P. and Lehalle, C.A., Mean field game of controls and an application to trade crowding. *Math. Financ. Econ.*, 2018, **12**, 335–363.
Carmona, R. and Laurière, M., Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: I – The ergodic case. arXiv preprint arXiv:1907.05980, 2019a.
Carmona, R. and Laurière, M., Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II – The finite Horizon case. arXiv preprint arXiv:1908.01613, 2019b.
Carmona, R. and Webster, K., The self-financing equation in limit order book markets. *Financ. Stoch.*, 2019, **23**, 729–759.
Cartea, Á. and Jaimungal, S., Incorporating order-flow into optimal execution. *Math. Financ. Econ.*, 2016, **10**, 339–364.
Cartea, Á., Jaimungal, S. and Penalva, J., *Algorithmic and High-frequency Trading*, 2015 (Cambridge University Press: Cambridge).
Charpentier, A., Elie, R. and Remlinger, C., Reinforcement learning in economics and finance. arXiv preprint arXiv:2003.10014, 2020.
Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P. and Yang, K. *et al.*, Large scale distributed deep networks. In *Proceedings of the Advances in Neural Information Processing Systems*, Denver, pp. 1223–1231, 2012.
E, W., Han, J. and Jentzen, A., Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Statist.*, 2017, **5**, 349–380.
Fouque, J.P. and Zhang, Z., Deep learning methods for mean field control problems with delay. arXiv preprint arXiv:1905.00358, 2019.
Frosst, N. and Hinton, G., Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784, 2017.

Gobet, E. and Munos, R., Sensitivity analysis using itô-Malliavin calculus and martingales, and application to stochastic optimal control. *SIAM J. Control Optim.*, 2005, **43**, 1676–1713.

Guéant, O. and Manziuk, I., Deep reinforcement learning for market making in corporate bonds: Beating the curse of dimensionality. arXiv preprint arXiv:1910.13205, 2019.

Guéant, O., Manziuk, I. and Pu, J., Accelerated share repurchase and other buyback programs: What neural networks can bring. *Quant. Finance*, 2020, **20**(8), 1389–1404.

Han, J. and E, W., Deep learning approximation for stochastic control problems. Deep Reinforcement Learning Workshop NIPS, 2016.

Han, J., Jentzen, A. and E, W., Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.*, 2018, **115**, 8505–8510.

Harradon, M., Druce, J. and Ruttenberg, B., Causal learning and explanation of deep neural networks via autoencoded activations. arXiv preprint arXiv:1802.00541, 2018.

Huré, C., Pham, H. and Warin, X., Some machine learning schemes for high-dimensional nonlinear PDEs. arXiv preprint arXiv:1902.01599, 2019.

Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P., On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836, 2016.

Kingma, D.P. and Ba, J., Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Labadie, M. and Lehalle, C.A., Optimal starting times, stopping times and risk measures for algorithmic trading. *J. Investment Strategies*, 2014, **3**.

Laruelle, S. and Lehalle, C.A., *Market Microstructure in Practice*, 2018 (World Scientific: Singapore).

Laruelle, S., Lehalle, C.A. and Pagès, G., Optimal posting price of limit orders: learning by trading. *Math. Financ. Econ.*, 2013, **7**, 359–403.

Lehalle, C.A. and Azencott, R., Piecewise affine neural networks and nonlinear control. In *Proceedings of the International Conference on Artificial Neural Networks*, London, pp. 633–638, 1998.

Liu, S., Oosterlee, C.W. and Bohte, S.M., Pricing options and computing implied volatilities using neural networks. *Risks*, 2019, **7**, 16.

Masters, D. and Luschi, C., Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612, 2018.

Mounjid, O. and Lehalle, C.A., Improving reinforcement learning algorithms: Towards optimal learning rate policies. arXiv preprint arXiv:1911.02319, 2019.

Murdoch, W.J. and Szlam, A., Automatic rule extraction from long short term memory networks. arXiv preprint arXiv:1702.02540, 2017.

Patzelt, F. and Bouchaud, J.P., Universal scaling and nonlinearity of aggregate price impact in financial markets. *Phys. Rev. E.*, 2018, **97**, 012304.

Raissi, M., Perdikaris, P. and Karniadakis, G.E., Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 2019, **378**, 686–707.

Sirignano, J. and Cont, R., Universal features of price formation in financial markets: Perspectives from deep learning. *Quant. Finance*, 2019, **19**, 1449–1459.

Sirignano, J. and Spiliopoulos, K., DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 2018, **375**, 1339–1364.

Tan, S., Caruana, R., Hooker, G., Koch, P. and Gordo, A., Learning global additive explanations for neural nets using model distillation. arXiv preprint arXiv:1801.08640, 2018.

Tian, C., Tao, S., Maarek, P. and Zheng, L., Learning to trade with market signals. Neurips Workshop on Robust AI in Financial Services, 2019.

Wilson, D.R. and Martinez, T.R., The general inefficiency of batch training for gradient descent learning. *Neural. Netw.*, 2003, **16**, 1429–1451.

Zhang, Q., Cao, R., Shi, F., Wu, Y.N. and Zhu, S.C., Interpreting cnn knowledge via an explanatory graph. arXiv preprint arXiv:1708.01785, 2017.

Zhang, Q., Cao, R., Wu, Y.N. and Zhu, S.C., Growing interpretable part graphs on convnets via multi-shot learning. arXiv preprint arXiv:1611.04246, 2016.

Zhang, Q., Yang, Y., Ma, H. and Wu, Y.N., Interpreting cnns via decision trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Long Beach, pp. 6261–6270, 2019.

Appendices

A. Appendix 1. Details on explicit solution and implementation

A.1. Details on the explicit solution

For the sake of completeness, we recall how the benchmark solution is obtained; see Cartea and Jaimungal (2016) for more details. The continuous form of the problem defined in equations (1) to (4) of the paper, when $\gamma = 2$, can be characterized by the value (we drop the subscripts A and ϕ to alleviate the notations)

$$\mathbb{E}_{X_0,S_0,Q_0}[V(0,X_0,S_0,Q_0)],$$

where V is the value function, defined as:

$$V(t_0,x,s,q) = \sup_{\nu} \mathbb{E}\left[X_T + Q_T S_T - A|Q_T|^2 - \phi\int_{t_0}^{T} |Q_t|^2\right]$$

$$\text{subjectto} \begin{cases} \mathrm{d}S_t = \alpha(\mu_t + \nu_t)\,\mathrm{d}t + \sigma\,\mathrm{d}W_t \\ \mathrm{d}Q_t = \nu_t\,\mathrm{d}t \\ \mathrm{d}X_t = -\nu_t(S_t + \kappa\nu_t)\,\mathrm{d}t \\ S_t > 0, \quad \forall\, t \\ X_{t_0} = x, Q_{t_0} = q, S_{t_0} = s. \end{cases} \tag{A1}$$

From dynamic programming, we obtain that the value function V satisfies the following Hamilton-Jacobi-Bellman (HJB): for $t \in [0,T), x, s, q \in \mathbb{R}$,

$$\partial_t V - \phi q^2 + \frac{1}{2}\sigma^2\partial_S^2 V + \alpha\mu\partial_S V + \sup_{\nu}\left\{\alpha\nu\partial_S V + \nu\partial_q V - \nu(s+\kappa\nu)\partial_X V\right\} = 0 \tag{A2}$$

with terminal condition $V(T,x,s,q) = x + q(s - Aq)$.

If we use the ansatz $V(t,x,s,q) = x + qs + u(t,q)$, with u of the form $u(t,q) = h_0(t) + h_1(t)q + h_2(t)\frac{q^2}{2}$, the optimal control resulting from solving this problem can be written as:

$$\nu^*(t,q) = \frac{\alpha q + \partial_q u(t,q)}{2\kappa} = \frac{h_1(t)}{2\kappa} + \frac{\alpha + h_2(t)}{2\kappa}q. \tag{A3}$$

Hence, $h_1(t)$ and $h_2(t)$ act over the control by influencing either the intercept of an affine function of the inventory, in the case of $h_1(t)$, or its slope, in the case of $h_2(t)$.

From the HJB equation, these coefficients are characterized by the following system of ordinary differential equations (ODEs):

$$\begin{cases} \dot{h_2}(t) = \left(2\phi - \dfrac{1}{2\kappa}\alpha^2\right) - \dfrac{\alpha}{\kappa}h_2(t) - \dfrac{1}{2\kappa}h_2^2(t), \\ \dot{h_1}(t) + \dfrac{1}{2\kappa}(\alpha + h_2(t))h_1(t) = -\alpha\mu(t), \\ \dot{h_0}(t) = -\dfrac{1}{4\kappa}h_1^2(t) \end{cases} \tag{A4}$$

with terminal conditions:

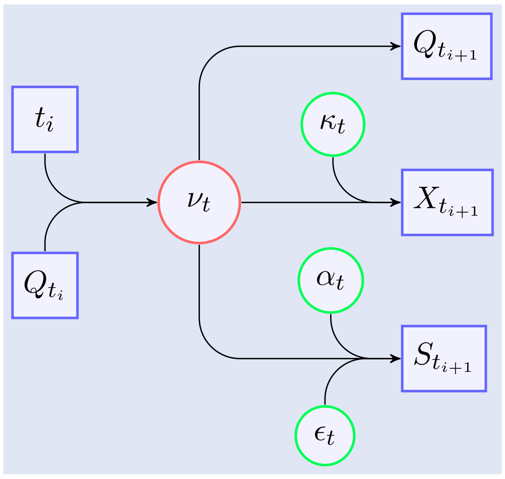

Figure A1. Structure of the state variables' simulation—One Step .

$$\begin{cases} h_0(T) = 0, \\ h_1(T) = 0, \\ h_2(T) = -2A. \end{cases} \tag{A5}$$

A.2. Details on the implementation

In this section, we provide more details on the implementation of the method based on neural network approximation. For the neural network, we used a fully connected architecture, with three hidden layers, five nodes each.

One forward step in this setup is described by figure A1, while one round of the SGD is represented by figure A2. The *same* neural network learns from the state variables obtained in the previous step. It thus learns the influence of its own output through many time steps. We have experimented with using one neural network per time step. However, this method implies more memory usage, and did not provide any clear advantage with respect to the one presented here.

From figures A1 and A2, we can clearly see the idea that the control influences the dynamics. We are, in fact, optimizing in a closed-loop learning environment, where the trader's actions have both permanent and temporary market impact on the price.

The mini-batch size we used, namely 64, is relatively small. While papers like (Dean *et al.* 2012) defend the use of larger mini-batch sizes to take advantage of parallelism, smaller mini-batch sizes such as the ones we use are shown to improve accuracy in Wilson and Martinez (2003), Keskar *et al.* (2016), and Masters and Luschi (2018).

Simulations are run on a MacOS Mojave laptop with 2.5 GHz Intel Core i7 and 16G of RAM, without GPU acceleration. Available GPU cluster did not increase the average speed of the simulations. Tests done on CPU clusters composed of Dell 2.4 GHz Skylake nodes also did not indicate relevant speed improvements.

A.3. Kurtosis and auto-correlation of stock returns

Table A1 shows the values of kurtosis and auto-correlation for all the stocks. They respectively indicate that the returns have heavy tails and are auto-correlated. Neither of these characteristics of real stock returns are taken into account in the baseline model, but they are easily accounted for in the deep neural network setup presented in the paper.

A. Appendix 2. Learning the mapping to the optimal control in a closed loop

A.1. Details on market impact parameters estimation

Permanent market impact. Let $\mathcal{S}$ be the space of stocks; and $\mathcal{D}$ be the space of trading days. If we take five-minute intervals (indicated by the superscript notation), we can write Equation (1) for each stock $s \in \mathcal{S}$, for each day $d \in \mathcal{D}$, and for each five-minute bin indexed by t as:

$$\Delta S^{5\text{min}}_{s,d,t} = \alpha_{s,t}\mu^{5\text{min}}_{s,d,t}\Delta t + \sigma_s\sqrt{\Delta t}\epsilon^{5\text{min}}_{s,d,t}, \tag{A6}$$

where the subscripts s, d, t respectively indicate the stock, the date, and the five-minute interval to which the variables refer, and $\Delta t = 5$ min, by construction. We have $\alpha_{s,t}$ independent of d, which assumes that for any given day the permanent market impact multiplier the agent may have on the price of a particular stock s, for a given time bin t is the same. And we have σ_s independent of the day d and time bin t, which means the volatility related to the noise term is constant for a given stock.

As an empirical proxy for the theoretical value $\mu^{5\text{min}}_{s,d,t}$ representing the net order flow of all the agents, we use the order flow imbalance observed in the data: $\text{Imb}^{5\text{min}}_{s,d,t}$. We define this quantity as:

$$\text{Imb}^{5\text{min}}_{s,d,t} = \sum_{j=t}^{t+5\text{min}} v^{\text{buy}}_{s,d,j} - \sum_{j=t}^{t+5\text{min}} v^{\text{sell}}_{s,d,j}, \tag{A7}$$

where $v^{\text{buy}}_{s,d,t}$ is the volume of a buy trade, and $v^{\text{sell}}_{s,d,t}$ is the volume of a sell trade, and we aggregate their net amounts over five minute bins.

However, since we estimate the permanent market impact parameter α using data from different stocks, and we would like to perform a single regression for all of them, we re-scale the data used in the estimation. In order to make the data from different stocks comparable, we do the following:

(i) Divide the trade price difference ($\Delta S^{5\text{min}}_{s,d,t}$) by the average bin spread over all days for the given stock, also calculated in its respective five minute bin:

$$\overline{\Delta S}^{5\text{min}}_{s,d,t} = \frac{\Delta S^{5\text{min}}_{s,d,t}}{\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}\psi^{5\text{min}}_{s,d,t}}, \tag{A8}$$

where $\psi^{5\text{min}}_{s,d,t}$ is the bid-ask spread for a given (stock, date, bin) tuple.

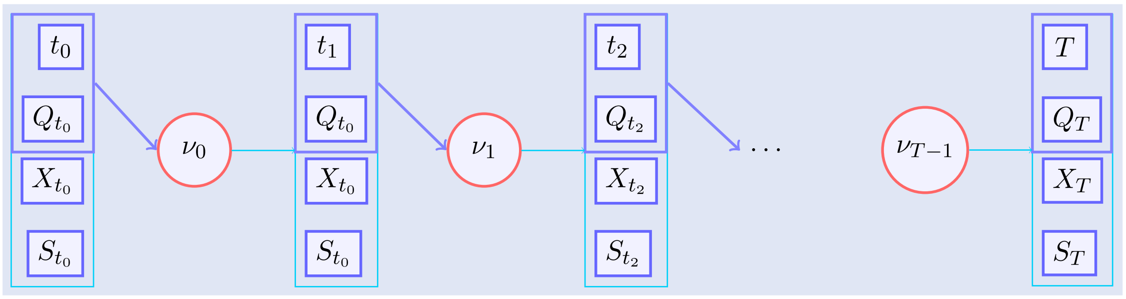

Figure A2. Structure of the State Variables' Simulation—All Steps.

Table A1. Kurtosis and auto-correlation of intra-day returns.

Kurtosis	ABX	AEM	AGU	BB	BMO	BNS	COS
	222	264	1825	2344	1796	2087	2528
	DOL	GIL	GWO	HSE	MRU	PPL	RCI.B
	163	7746	507	2975	319	7006	2188
	RY	SLF	TRI	TRP	VRX		
	2648	11302	403	7074	213		
Auto-correlation	ABX	AEM	AGU	BB	BMO	BNS	COS
(lag = 5 min)	− .025	− .026	− .03	− .031	− .026	− .03	− .041
	DOL	GIL	GWO	HSE	MRU	PPL	RCI.B
	− .118	− .047	− .094	− .02	− .105	− .15	− .062
	RY	SLF	TRI	TRP	VRX		
	− .017	− .07	− .04	− .066	− .075		

(ii) Divide the trade imbalance ($\text{Imb}^{5\text{min}}_{s,d,t}$) by the average bin volume over all days for the given stock, calculated on the respective five minute bin:

$$\overline{\text{Imb}}^{5\text{min}}_{s,d,t} = \frac{\text{Imb}^{5\text{min}}_{s,d,t}}{\frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}(\text{TotalVolume}^{5\text{min}}_{s,d,t})}. \tag{A9}$$

where $\text{TotalVolume}^{5\text{min}}_{s,d,t}$ stands for the total traded volume for both buys and sells for a given (stock, date, bin) tuple.

This ensures that both the volume and the price are normalized to quantities that can be compared. Each (s, d, t) now gives rise to one data point, and we can thus run the following regression instead, using comparable variables, to find $\bar{\alpha}$:

$$\overline{\Delta S}^{5\text{min}}_{s,d,t} = \bar{\alpha}\cdot\overline{\text{Imb}}^{5\text{min}}_{s,d,t} + \bar{\epsilon}^{5\text{min}}_{s,d,t}, \tag{A10}$$

where $\bar{\alpha}$ is the new slope parameter we would like to estimate, and $\bar{\epsilon}^{5\text{min}}_{s,d,t}$ is the normalized version of the residual we had in equation (A6).

In order to use $\bar{\alpha}$ in a realistic way, we de-normalize the regression equation for each stock by doing:

$$\Delta S^{5\text{min}}_{s,d,t} = \left(\frac{\bar{\alpha}}{\Delta t}\frac{\bar{\psi}_{s,t}}{\bar{\mathcal{V}}_{s,t}}\right)\cdot\text{Imb}^{5\text{min}}_{s,d,t} + \sigma\sqrt{\Delta t}\epsilon^{5\text{min}}_{s,d,t}. \tag{A11}$$

where $\bar{\mathcal{V}}_{s,t}$ is the average bin volume for stock s at bin the time bin t, $\bar{\psi}_{s,t}$ is the average bin bid-ask spread for the same pair (s, t).

The derivation for the *temporary market impact* κ uses equation (2) in the paper, and follows similar steps.

Temporary market impact. The κ parameter represents the magnitude of the trading cost of an agent in the market. This is an indirect cost incurred by the trader due to the impact that their order has in the market mid-price. From the dynamics of the wealth process $\mathrm{d}X_t = -\nu_t(S_t + \kappa\nu_t)\,\mathrm{d}t$, we are assuming that the trading cost is linear in the speed of trading. Notice that the faster the agent needs to trade, the larger the cost incurred, and the opposite is true for slower trading. We can re-normalize (A6) by rewriting it as:

$$\delta x := (-1)\cdot\frac{\mathrm{d}X_t}{\nu\,\mathrm{d}t} = S_t + \kappa\nu_t. \tag{A12}$$

Then, for each stock $s\in\mathcal{S}$, for each day $d\in\mathcal{D}$, and for each five-minute bin indexed by t, we would like to estimate the following expression:

$$\delta x^{5\text{min}}_{s,d,t} = S^{5\text{min}}_{s,d,t,\text{start}} + \kappa\nu^{5\text{min}}_{s,d,t} + \epsilon^{5\text{min}}_{s,d,t}. \tag{A13}$$

where the subscript 'start' indicates that we are using the mid-price at the start of the five-minute bin indexed by t.

Since we do not have access to ν, as it is private information for each trader, we would like to compensate by restricting our statistical analysis to bins that are dominated by only one trader, but not so much as to have stagnant prices. With the goal of isolating one trader from the crowd in mind, we seek to restrict our dataset based on the imbalance and on the size of the dominating traded volume in a given bin.

We extract the trade volume for each bin as the maximum between the aggregate buy volume and the aggregate sell volume for that particular bin. With this criteria we simultaneously define whether we have a buy or a sell, and thus the sign of the trade (+ 1 for buys, and − 1 for sells). In mathematical terms, the dominant trading volume by bin, $D^{5\text{min}}_{s,d,t}$, can be expressed as:

$$D^{5\text{min}}_{s,d,t} = \frac{\max_{5\text{min}}\left\{\sum_{j=t}^{t+5\text{min}}\nu^{\text{buy}}_{s,d,j}, \sum_{j=t}^{t+5\text{min}}\nu^{\text{sell}}_{s,d,j}\right\}}{\frac{1}{|d|}\sum_{d\in\mathcal{D}}(\nu^{5\text{min}}_{s,d,t})}. \tag{A14}$$

where all variables have been defined in the paragraph about Permanent market impact.

As Patzelt and Bouchaud (2018) find, aggregate-volume impact saturates for large imbalances, on all time scales, and highly biased order flows are associated with very small price changes. They find that the probability for an order to change the price decreases with the local imbalance, and vanishes when the order signs are locally strongly biased in one direction. However, if we first restrict the data to having the dominant trading volume by bin $D^{5\text{min}}_{s,d,t}$, the imbalances are naturally restricted. Hence, we do not need to further restrict the upper bound of the imbalance. However, we do want to restrict its lower bound, since we want to analyze bins that isolate either the buying or the selling activity of only one trader.

In short, to estimate κ we restrict our dataset in the following way:

(i) $2 < D^{5\text{min}}_{s,d,t} < 10$,

(ii) $|\text{Imb}^{5\text{min}}_{s,d,t}| > |\text{Imb}^{5\text{min}}_{s,d,t}|_{40\%}$.

For the trading cost κ estimation, we would like to track agents who traded more than usual during five minutes. Hence, if the volume traded in the direction that had most trades is not minimally significant, we would not like to use it in the estimation. We have defined in equation (A14) the dominant trading volume as being the volume in the direction that had more trades during a given bin (either buy or sell).

We exclude from the sample those trades that had a dominant trading volume lower than twice the average dominant volume for a given bin. Since we only want to keep track of one individual agent, we also restrict this same quantity to be below ten times the average dominant trade volume for the bin. We believe that if the volume is larger than this, then it is very likely that other agents are also involved in the trade.

The third and last filter we make is for the trade imbalance, as we would like to isolate the effect of buys and sells. We keep in the sample only those observations on which the absolute value of the imbalance is greater than the 40% percentile of the signed imbalances.

Once these filters have been put into place, we are almost ready to estimate the agent's transaction costs. For our new subset of (s, d, t) only, we would like to establish:

$$\delta x^{5\text{min}}_{s,d,t} - S^{5\text{min}}_{s,d,t,\text{start}} = \kappa\nu^{5\text{min}}_{s,d,t}. \tag{A15}$$

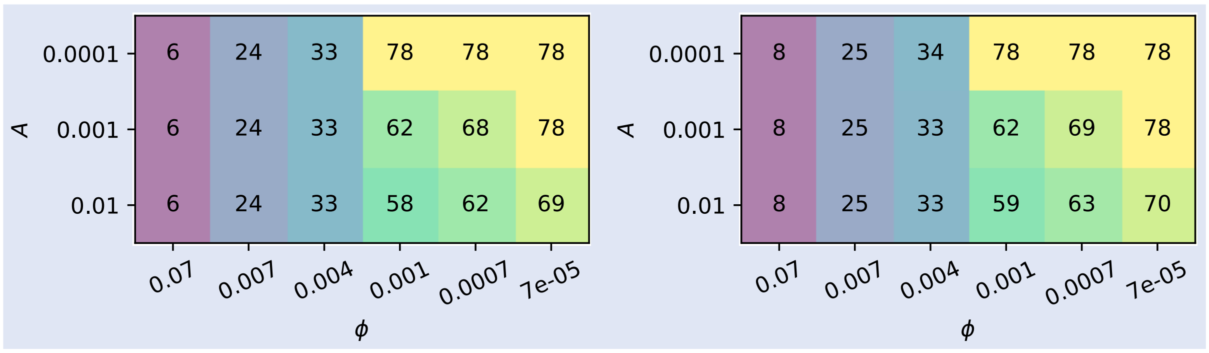

Figure A3. Time steps until at least 90% of the order is executed, for PDE closed-form (left) and for neural network multi-preference controller (right).

Nevertheless, we would like to do it for all stocks at the same time, just like we did for the parameter α. Hence, we need to re-normalize both sides of this equation to comparable values. For the left-hand side, we re-normalize price moves by the average bid-ask spread of a given stock on a given bin. For the right-hand side, we re-normalize $v^{5\min}_{s,d,t}$ by the average trading volume at the bin. Notice that this is precisely $D^{5\min}_{s,d,t}$, as defined in equation (A14). Hence, we have:

$$\frac{\delta x^{5\min}_{s,d,t} - S^{5\min}_{s,d,t,\text{start}}}{\frac{1}{d}\sum_{d\in\mathcal{D}}\psi^{5\min}_{s,d,t}} = \kappa \frac{v^{5\min}_{s,d,t}}{\frac{1}{|d|}\sum_{d\in\mathcal{D}} v^{5\min}_{s,d,t}}. \tag{A16}$$

Having re-normalized our variables, we can now run the regression on equation (A6) to find κ and estimate individual trading costs. In practice, we will be using the VWAP defined in Section (2.1), item 4 as the input for δx.

The regression we estimate in order to find κ is, thus:

$$\frac{\text{sign}^{5\min}_{s,d,t}\cdot(\text{VWAP}^{5\min}_{s,d,t} - S^{5\min}_{s,d,t,\text{start}})}{\frac{1}{|d|}\sum_{d\in\mathcal{D}}\psi^{5\min}_{s,d,t}} = \kappa \frac{v^{5\min}_{s,d,t}}{\frac{1}{|d|}\sum_{d\in\mathcal{D}} v^{5\min}_{s,d,t}}. \tag{A17}$$

A.2. *Details on the multi-preferences controller*

In Section 2.3 of the paper, we discussed the exploration—exploitation trade-off we encountered when projecting the neural network controller onto the manifold of closed-form solutions when $\gamma = 2$.

In figure A3, we compare the number of time steps it takes the controller to trade at least 90% of the initial inventory in the market. Rows stand for values of the risk parameter $A \in \{0.01, 0.001, 0.0001\}$, while columns represent the risk parameter $\phi \in \{0.07, 0.007, 0.004, 0.001, 7e^{-4}, 7e^{-5}\}$.

When the trader's preferences over risk aversion gives them incentive to trade faster, the inventory is executed sooner. The controller is 'exploiting' the available trading choices, and the projection on the closed-form manifold becomes harder to estimate. On the other hand, if the pair (A, ϕ) defines less restrictive preferences towards execution, there will be some (although not a lot of) inventory left at the end of the trading day that allows us to learn the projection accurately.

A. Appendix 3. Ethical impact and broad societal implications

A.1. *Explainability of learned control for financial markets*

Attempts of using machine learning for financial markets are booming the last 5 years. It spans a wide spectrum of applications: from client profiling to nowcasting using databases of texts and satellite images. One area where the acceptance of ML goes slowly is the automation of hedging strategies. These strategies are part of the functioning of financial markets in the post 2008-crisis environment due to genuine regulations demand market participants to compute and compensate their exposure to as many risk scenarios as possible. Moreover, and for obvious reasons, regulators ask for a better understanding of algorithms that look like 'black boxes' before allowing the use of ML in production to improve the efficiency of hedging. It is related to the very well-known topics of '*explainability of AI*' and '*ethics of algorithms*'.

Optimal trading addresses part of these hedging strategies. A framework like the one in this paper corresponds to a 'large investor', typically a pension fund, who decided to rebalance its portfolio for good reasons (think about the market moves following the spread of COVID-19: the optimal response of pension funds is clearly to rebalance their portfolio to get less exposure to factors like airplane companies, to prevent their pensioners to suffer from losses in the coming year), but they have to do it at a slow enough pace to not push the price too much at their own detriment (Laruelle and Lehalle 2018). The reason for the big COVID-19 drop of markets followed by a rebound 10 days later is due to non-optimal trading strategies: the institutions sold too fast, paying too much for their deleveraging and perturbing the prices with the downward pressure of accelerated sells. Being able to adapt the trading strategies to realistic features of intra-day price dynamics (like seasonality and auto-correlations) decreases both the trading costs for large institutions like pension funds (and consequently to their pensioners) and is profitable to the public price formation, that is otherwise does not reflect the 'fair value' of traded instruments.

In the introduction, we cite papers on improving optimal trading, via reinforcement learning or directly using deep learning. They do not attempt to *provide an explainable version of the obtained controls*. In this paper we do it in a very practical way that is compliant with the practices of the financial industry: we project the learned controls on the manifold spanning the controls currently in use by brokers and asset managers, and measure the distance between the obtained projection and the learned controls. In doing this, we provide evidences that the learned controls that we generated are largely contained in this regular manifold, and we allow practitioners and regulators to apply their usual stress testing and certification practices to the projected controls. It is the first proposal in this direction, and we hope that it will propagate in other areas of finance, like deep hedging. It may also be used in other industries when the regulatory demand in explainability is high.

A.2. *'Functional learning' for control*

The second 'broad impact' of this paper is what we called 'functional learning' in the scope optimal trading with the setup we call *multi-preferences neural network*. In other applications of optimal control, the loss function is fixed because it correspond to one universal use of the controlled object. On financial markets the loss function is parametrized by hyperparameters describing the risk aversion of the end-user. In game theory versions of the optimal trading framework, they are usually called 'agent preferences' (Cardaliaguet and Lehalle 2018). It is like allowing the driver of an autonomous car to choose the 'driving style' they want to use: far below the speed limit (may be for low carbon emission reasons) or just below, or a 'sport mode', etc. In financial markets, mainly because on the one hand models of price impact have confidence intervals and on the other hand asset owners can have a lot of reasons to rebalance their portfolio (think about an Australian pension fund that has to sell fast enough to face the exceptional pensioners' redemption authorized by the government following the COVID-19 crisis, see Burgess (2020); it is clear that the selling speed has to be fast enough to give the money to pensioners), these parameters are chosen at the start of trading ($t = 0$ in our framework) by the end-user. Today a stylized version of the control problem is numerically solved on the fly (either in closed-form, either by a numerical scheme) for the chosen preferences, but the dynamics have to be stylized, hence simplified, enough to allow to solve this very quickly. Learning the optimal control each time an end-user choose new preferences is not fast enough.

In this paper, the neural net is learning to solve the optimal control problem for any preference; in fact it learns the mapping between the preferences and the optimal solution. It means more flexibility to adapt to user's needs that 'one learning fits all model'.

It effectively learns to solve a whole parametrized family of Hamilton-Jacobi-Bellman equations. Hence it can provide the optimal trading strategy to a given choice of preferences in a flash. Keep in mind this optimal strategy is not one number but it is itself a mapping between the state space of the control problem and the optimal trading speed to be applied.

It had chances to work because it is straightforward to check (in the continuous setting) that the infinite-dimensional optimal control strategy is a smooth function of the two-dimensional vector of preferences (A, ϕ), hence we ask to the neural network to interpolate high dimensional functions in this 2-dimensional grid. Nevertheless it is far from easy (that's probably why other attempts did not succeed by now).

In the paper, we refer to what we suspect to be the main difficulty under the name of 'exploration—exploitation issue'. For some pairs of user preferences, the optimal trading speed is so fast that the full order is bought or sold far before T, and hence the neural network cannot learn from the feedback of its control on the dynamics, simply because it stopped to interact with them. To counter this we used the closed-form formula of the stylized problem to define a domain of user preferences that is compatible with our T, it helped the convergence of the 'functional learning'.