Forecasting Intraday Volatility and Densities using Deep Learning

Bruno Morier[a] and Pedro L. Valls Pereira[b]

[a]Head of Quantitative Strategies - Banco Safra
[b]Sao Paulo School of Economics-FGV and CEQEF-FGV. Rua Dr Plínio Barreto 365, room 1319
ZIP code: 01313-020 Sao Paulo, SP, Brazil
Corresponding author, E-mail: pedro.valls@fgv.br

Abstract

In this paper we develop a new model for forecasting high-frequency intraday conditional discrete return densities and volatility by using deep learning. Specifically, we model the conditional distribution by a modified Skellam distribution with the mean following an auto-regressive specification and train feedforward neural networks in order to generate predictions for the underlying high-frequency volatility. Four different specifications are tested including different set of features and parameters. Then we conduct a comprehensive walk-forward forecasting experiment in order to compare the forecasting accuracy for the proposed models. All the proposed models beat the empirical non-parametric forecasting rules considered. The new forecasting procedure also provides better out-of-sample forecasts compared to all Space State Models considered in the thesis. We also conclude that new variables have predictive power for the volatility process: the bid-ask spread, high-low interval spread and the volume traded. According to our model estimates, all these variables appear to have a positive non-linear S shaped relation with volatility.

Keywords: volatility models; high frequency data; discrete price changes; deep learning; neural networks; Skellam; non-Gaussian time series models; dynamic discrete data

JEL Classification: C32, C45, G11

1 Introduction

As argued in Morier and Valls Pereira (2023), modelling and predicting asset returns distributions is a principal research goal in finance. In the last decades this topic received a growing attention, especially regarding the measurement of the second moment - the volatility. And in more recent years the topic expanded to intraday volatility measuring and forecasting. There are many studies using intraday prices to forecast daily volatility (see Andersen et al. (2001),Barndorff-Nielsen and Shephard (2002), and Hansen and Lunde (2006)) and, more recently, papers using this data to analyse and forecast the intraday price dynamics itself (see Engle and Sokalska (2012) and Koopman et al. (2017)).

In the first paper we modelled the intraday price dynamics by using the Modified Skellam distribution for the conditional density of the intraday returns, while using non-linear non-Gaussian state space models for modelling the volatility dynamics, in a similar way to Koopman et al. (2017). The characteristics that motived this choice are that volatility is unobservable, changes through time and possess a clustering pattern. This features are well-known for a long time for daily volatility (see Shephard (2005) for a review).

In this paper we also model intraday conditional return densities with focus on volatility dynamics modelling, but we emphasize the non-linearity of the volatility response to its own past and usual explanatory variables. In order to capture such features we model the volatility dynamics through deep neural networks. Previous works have shown evidence for non-linearities for the volatility process not captured by traditional GARCH processes which can be captured by such networks. They also show the ability for such networks in dealing with other non-traditional explanatory variables for predicting volatility. See Donaldson and Kamstra (1997); Hajizadeh et al. (2012); Kristjanpoller et al. (2014); Bucci (2019)).

Our work contributes to this literature first by extending the deep neural network modelling to the intraday high-frequency time frame. To the best of authors knowledge there is no previous comprehensive work analyzing the intraday volatility forecast at a time frame lower than 1 minute and considering Skellam-like distributions. We believe that actually this time frame is more suitable for neural networks than the daily time frame, as the distributions are non-Gaussian and the data is abundant and noisy.

Our approach is more flexible than the State-Space and GARCH approaches, in the sense that is easier to include new explanatory variables as we do not need to specify a 'correct' functional form for the input variable's impact on the volatility process. The neural network approach is motived by its ability to universally approximate non-linear functions, so we learn the functional form during the estimation. Building on this advantage we include new explanatory variables (bid-ask spread, high-low spread and volume) and test for their impact on the forecasts. We analyze the sensivity of changes of these input variables in the forecasts and analyzing the changes in the predictive power added for the inclusion of these variables.

Lastly, we use the neural network not only to forecast volatility, we actually forecast conditional densities in a similar way to Koopman et al. (2017, 2018). And we compare the results with models derived from from the approach of such papers, which are tougher benchmarks than the usual GARCH models. We show the ability for the networks for forecasting the conditional densities and volatilities, testing for the forecasting performance in a comprehensive walk-forward

forecasting study.

The rest of this paper is organized as follows. Section 2 gives a brief overview on the Deep Learning approach, describing neural networks, the optimization process and design issues. Section 3 describes the base model and its variants. Section 4 describes the data. Section 5 gives details on the Training (optimization) procedure. Section 6 and 7 describes the estimation results and results related to the forecasting exercise. Section 8 concludes.

2 Deep Learning Approach: an Overview

According to Goodfellow et al. (2016), machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions. Machine learning is more concerned about making models to forecast, rather than learning about parameters.

The central challenge in machine learning is that models must perform well on new, previously unseen inputs — not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalization. A model has its hyper-parameters chosen in a validation set, is typically trained (fitted) to a training set (in-sample) and tested on the test set (out-sample). The objective is thus designing the model that has the lowest test (generalization) error.

All the emphasis in machine learning is on making models that have low test error. There is no need to attain a maximum likelihood estimate, for instance, and for many machine learning algorithms there are no guaranties that any in-sample estimates are optimal in any sense. The terminology 'optimizing' or 'fitting' is substituted by 'learning' to emphasize such difference.

2.1 Neural Networks and Deep Learning

Deep learning is the part of machine learning that use models that rely on Deep Artificial Neural Networks. We now motivate the use of such procedure.

One might presume that learning a nonlinear function requires designing a specialized model family for the kind of nonlinearity this function presents. Our interest in feedforward networks with hidden layers for this paper is based on the fact that they provide a universal approximation framework. The universal approximation theorem (see Hornik et al. (1989) and Cybenko (1989)) states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (like the logistic sigmoid activation function) can approximate any Borel measurable function between finite-dimensional spaces with any non-zero amount of error, provided that the network has enough hidden units. The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well (see Hornik et al. (1990)).

The existence of such approximating neural network does not mean we can feasibly attain such network by training a network design with just one layer. The approximating network might involve a too large width to be feasible. Goodfellow et al. (2016) argue that for many problems increasing the depth of the network results in better approximating function, reducing

the width needed. More complex relations can be modeled this way with less network nodes. The authors compare this structure with a prior that the function being modeled is composed of chain composition of simpler transformations. And in practice deeper modern neural networks have shown a better generalization error.

2.2 Optimization

As mentioned, the objective of machine learning is different from classical statistics and this reflects on how optimization is done in machine learning, especially for Deep Learning. Finding the global minimum on the training set is unfeasible for most neural network designs, as the optimization problem involved is complex. And this is not actually the objective, which is in fact generating the model with the lower generalization error possible. So training a deep learning model is significantly reducing a loss function L_1 in the training sample with the hope that this procedure will reduce other loss function L_2 in the test sample.

The typical optimization procedure in Deep Learning is described in the algorithm below. The typical loss function consists of a sum through samples. At each step a 'mini-batch' of samples is taken from the training set (possibly at random) and the gradient for the loss is calculated using only these samples. If all the training sample is used on the 'mini-batch', we call the optimization method 'batch' gradient descent (otherwise it is called 'minibatch' gradient descent).

Then the model parameters are updated using the gradient times some 'Learning Rate', which might be a fixed parameter (in case of the Stochastic Gradient Descent) or may be adaptive (in the case of Adam or RMSprop, for instance). This procedure is repeated for a number of times, until the training set is passed by the procedure for a maximum number of times. The 'number of epochs' is such number of passages. Some procedures also specify some early stopping criteria, often related to the loss function applied in some other test set.

Algorithm 1: Basic Deep Learning Optimization Algorithm

Result: Trained Network parameters θ
input : learning rate ϵ_k (or adaptive method for generating it)
input : Method for calculating gradients from minibatchs, θ and previous gradients
input : Maximum number of epochs: maxEpochs
input : Starting θ: θ_0
$k \leftarrow 0$;
$\theta \leftarrow \theta_0$;
while $k < maxEpochs$ **do**
  while *can sample from training set* **do**
    Sample m minibatch samples from training set (without replacement);
    Compute gradient g ;
    apply update: $\theta \leftarrow \theta - \epsilon_k g$;
    if *early stopping criteria met* **then**
      return θ;
    end
  end
  $k \leftarrow k + 1$;
  reset training set returning all samples to it;
end

3 Model

In this section we develop the base model used in the rest of this paper, its variants, the features and the network design.

Let $y_t \in \mathbb{Z}$ be the integer-valued variable representing the price changes for the asset at time t. We represent the price changes by integers as we are focused in high frequency price changes. At high frequency the price changes are discrete as stocks are traded in multiples of a tick size. So we represent again the price changes as integer multiples of the tick size for each stock.

Let $p(y_t|\mathcal{F}_{t-1};\theta_t)$ be the time-varying conditional probability mass function's (pmf's), where $\mathcal{F}_t := \{X_1, \ldots, X_t\}$ is the filtration (information set) at time t and θ_t collects all the pmf's parameters. The X_t is a vector of variables that might include y_t. Our main goal in this paper is to find $p(\cdot)$, X_t and θ_t that provide best out of sample forecasts for the distribution of y_t. That is, we are searching for the best conditional density forecasts, which are also functions of the volatility forecasts.

3.1 Base Model

Like in Koopman et al. (2017) and Koopman et al. (2018), we consider only the case where $p(\cdot)$ is given by the Modified Skellam Distribution. As shown in the first paper, this distribution has a good match for the intraday return's distribution and have a parameterization that turns easier to study volatility. So we have $p(y_t|\mathcal{F}_{t-1};\theta_t) = p_{II}(y_t; i, j, k, \mu_t, \sigma_t^2, \gamma)$, where p_{II} is given

by

$$p_{II}(y_t; i, j, k, \mu, \sigma^2, \gamma) := \begin{cases} p_s(y_t; \mu, \sigma^2) \quad \text{for} \quad y_t \notin \{i, j, k\} \\ p_s(y_t; \mu, \sigma^2) - \gamma \frac{\Delta}{2} \quad \text{for} \quad y_t \in \{i, j\} \\ p_s(y_t; \mu, \sigma^2) + \gamma\Delta \quad \text{for} \quad y_t = k, \end{cases} \tag{1}$$

where γ satisfies $2\min(p_s(i, \mu, \sigma^2), p_s(j, \mu, \sigma^2)) > \gamma\Delta > -p_s(k, \mu, \sigma^2)$ and p_s denotes the Skellam pmf. As in the paper 1 we set $(i, j, k) = (-1, 1, 0)$. The Skellam pmf is given by

$$p_s(y; \mu, \sigma^2) := \exp(-\sigma^2) \left(\frac{\sigma^2 + \mu}{\sigma^2 - \mu} \right) I_{|y|}(\sqrt{\sigma^4 - \mu^2}), \tag{2}$$

where $I_{|y|}$ is the Bessel I Function with $|y|$ degrees of freedom.

Now we need only to specify the functional forms of σ_t and μ_t. All models considered in this paper, including the ones used for comparison follow the structure described so far. And this includes the models in paper Morier and Valls Pereira (2023). The functional for for μ_t is also the same of that paper. Thus μ_t is given by:

$$\mu_t = \delta y_{t-1} \tag{3}$$

which is different from Koopman et al. (2017) that uses $\mu_t = constant$.

We model σ_t by a nonlinear function of X_t that will be approximated by a deep Feed Forward Neural Network. So the functional form of σ_t is simply

$$\sigma_t = F(X_{t-1}) \tag{4}$$

where the F function represents the Neural Network. In the Machine Learning nomenclature, X_t represents the features we consider in our model. The features themselves might be generated by another model, making a Hybrid model.

3.2 The Features

We follow Donaldson and Kamstra (1997); Hajizadeh et al. (2012); Kristjanpoller et al. (2014) and use a Hybrid model, using outputs from another model as a feature for the neural network. In particular we used the output of a simple EWMA model, with its λ parameter estimated in-sample. The EWMA model is given by

$$\hat{\sigma}_t^2 = \lambda y_{t-1}^2 + (1 - \lambda)\hat{\sigma}_{t-1}^2 \tag{5}$$

we also denote EWMA forecast for time t by ew_t, that is, $ew_t := \hat{\sigma}_t^2$.

We also include y_t and y_t^2 as features. The idea for including y_t^2 is that the simple EWMA process might be unable to capture all the (possibly nonlinear) impact of the innovation y_t^2, even if it does already include such input. The y_t is included to capture asymmetric reactions of the volatility process regarding the sign of the return that have long being documented in the daily volatility forecasting literature (see Glosten et al. (1993) and Engle and Ng (1993)).

A salient feature of intraday volatility is seasonality. This aspect has been documented in many studies (e.g Andersen and Bollerslev (1997)). There are several reasons for deterministic

seasonality, including the usual schedule for economic and company news releases, lunch-time, the market opening and market closing. A common pattern is a high volatility on the market opening, which falls through the lunch-time and rises again over day afternoon until the makert close. As shown in Andersen (1996), the intraday volatility is also related to the volume of trading activity, which is itself influenced by the events just described.

We model the seasonality pattern using a parsimonious specification through cubic splines. The treatment is similar to the one used in Harvey and Koopman (1993) and Koopman et al. (2017). Specifically, we write s_t as a zero-sum cubic spline function

$$s_t = \beta' W_t. \tag{6}$$

W_t is calculated as described in Poirier (1973). β is a $K \times 1$ vector of parameters associated with $K + 1$ spline knots. In this paper instead of using fixed coefficients β like in equation (6), the splines join the features of the neural network. So they might interact in a non-linear way with all other features in determining the volatility forecast. As in the case of Morier and Valls Pereira (2023) we considered $K = 3$ for generating W_t, and the four resulting spline knots are fixed at the times 10:00, 12:00, 13:30 and 17:00, representing the market open, lunch time and market close. So W_t has dimension 3 and we denote the three resulting features as W_1, W_2, W_3.

Last set of features is composed of 3 new variables not considered, at the same time, in previous papers, which are not traditionally considered in volatility models. The first is the closing bid-ask spread, denoted by ba. When the volatility rises market liquidity providers including market makers, traders and HFT algorithms tend to be less aggressive in attending the market final orders. In the case of high frequency data, a large bid-ask spread also mechanically affect the volatility by increasing the bid-ask bounce effect. In any case this variable is expected to affect the volatility and is included for this reason. See Wyart et al. (2008) for a reference on this effect.

The second variable form this set is the High-Low interval spread for the last 10 seconds, denoted by hl. There is a vast literature relating the daily high-low interval to the volatility, and though our time frame is short (10 seconds), this variable is also expected to have some predictive value for the volatility. The intuition is simple: the more volatile the market, the wider this range should be. This is true for a continuous Brownian Motion (i.e. by the Reflection Principle) and is also expected for the discrete setting we are interested. See Yang and Zhang (2000), for instance, for the usage of this data for volatility forecasting in the daily time frame.

The last variable we consider is the volume of trade for the last 10 seconds, denoted by v. The volume of trade is related to the intensity of the market activity and is reported to rise/fall leading and lagging volatility rises and falls (see Xu et al. (2006)). The relation between volume and price changes is so relevant that some authors argue that intraday bar time series (like ones considered in this paper) are better analyzed by making regular intervals in terms of volume traded instead of time in order to make the return distributions more close do iid (see de Prado (2018)). There are also data vendors that offer the data formed this way.

All the features above are z-score normalized before entering the Neural Network input. The parameter for such normalization (mean and standard deviation) are estimated in-sample and kept unchanged for the test-sample.

3.3 Network Design

As the objective is to approximate an arbitrary non-linear function on the features, and there are just a few of them, we use a fully connected feedforward neural network. This type of network is characterized by the fact that information flows from the input layer, passing to hidden layers and arriving at the output layer without any cycles. Fully connected means that between any two layers all nodes are connected.

Following the modern deep learning practice, we choose a large depth (number of layers) and width (neurons per layer). We use 20 layers, each with 10 neurons. As the other hyper-parameters for the model, these parameters were tested on a validation, set prior to the data used for estimating the models. They are not necessarily the best hyper-parameters for a forecasting network, but in our tests the forecasting capabilities do not change much for large networks.

As the network is fully connected, this design means that we have 2000 weight parameters to estimate and 20 bias parameters. Although it might seem that are too many parameters, each training sample we use for this paper consists of about 20.000 data points.

Networks with a larger depth are usually harder to train, but are able to capture more complex relations between the input variables. Depending on the activation function, the depth can make problems associated with vanishing gradients worse (Goodfellow et al. (2016)) and are more computationally expensive - which was a problem in the past, but not nowadays.

In order to compensate for the difficulty in training these deep networks, we make three design choices:

1. A relatively high width (10 neurons)
2. Use the Leaky ReLU activation function
3. Use a customized initiation procedure

The first choice, picking a high width, alleviates the problem of vanishing gradients by simply adding more terms to the output expression, adding more paths so that the information can flow from the input layer to the output layer. So this choice makes less likely that the dead neurons (neurons with too low gradient) affect the entire network. See Zagoruyko and Komodakis (2016) for a discussion on the effects of network width on training.

The second choice tackles the vanishing gradients problem directly, by using an activation function whose gradient never vanishes. The LeakyReLU activation function (Maas et al. (2013)) is defined as:

$$\text{LeakyReLU}(x) := \{[l]x \quad x \in (0, \infty), 0.01x \quad \text{otherwise} \tag{7}$$

The gradient of LeakyReLU never vanishes, attaining a minimum value that is strictly greater than zero. This is different than the more standard activation functions tanh, sigmoid and ReLU. Worth mentioning that even tough the gradient never vanishes completely, it can be as low as 1e-40 on the composition of the 20 layers, so that although this choice alleviates the problem, we can still have dead neurons in practice.

The third choice refers to the initiation of the weights for the network, which will be described in greater detail in section 5. In short we initiated an entire path of weights from the input

feature ew to the output as ones, setting all bias coefficients as zero, except for the last one, which was set to a positive number. This choice of initiation made the starting parameters of the network resemble ew, making the gradients not vanishing and the computations for the log-likelihood loss derived from (1) more stable.

Let $I = 8$ be the number of features, $L = 20$ the number of layers and $H = 10$ the width of the network. The final definition for the neural network, that is, for the function F is given by the following equations:

$$\begin{aligned} h_t^{(0)} &= \text{leakyReLU}(W^{(in)} X_t + b^{(in)}) \\ h_t^{(i)} &= \text{leakyReLU}(W^{(i)} h_t^{(i-1)} + b^{(i)}), \text{ for } i \in \{1, \ldots, L\} \\ \sigma_{t+1}^2 &= F(X_t) = |W^{(out)} h_t^{(L)} + b^{(out)}| \end{aligned} \tag{8}$$

where $W^{(in)}$ is $(I \times H)$, $W^{(i)}$ is $(H \times H)$,$W^{(out)}$ is $(H \times 1)$, $b^{(in)}$ is $(H \times 1)$, $b^{(i)}$ is $(H \times 1)$ and $b^{(out)}$ is (1×1).

3.4 Model Variants

In order to measure the impact of different components of our model, we estimate a total of 4 versions of the model, including the final model and 3 simplified versions.

The table 1 resumes the characteristics of the models. The most basic is NN_0, in which we force γ and δ to zero and we use the same features used in paper 1 (that is, we exclude ba, hl and v). So this model use the standard Skellam distribution, with no model for the mean ($\mu_t = 0$).

The next is NN_v, which is like NN_0, except that we include the variables ba, hl and v. In $\text{NN}_{v,\gamma}$ we allow γ to be different than zero. The final model is $\text{NN}_{v,\gamma,\mu}$, in which we also model the mean using equation (3).

Table 1: Model Variant description

model	Features	γ	δ
NN_0	all except ba,hl, and v	$\gamma = 0$	$\delta = 0$
NN_v	all	$\gamma = 0$	$\delta = 0$
$\text{NN}_{v,\gamma}$	all	estimated	$\delta = 0$
$\text{NN}_{v,\gamma,\mu}$	all	estimated	estimated

4 Data

Like in Morier and Valls Pereira (2023), our dataset is formed by intraday prices for four Brazilian Stocks: Petrobras (PETR4), Vale (VALE3), Itau-Unibanco (ITUB4) and Bradesco (BBDC4) for the entire year of 2018. These stocks are among the most liquid Brazilian stocks. The data used in this paper consists of the closing trading prices for the intraday interval of 10

seconds obtained from B3 exchange by using Thomson Reuters Datascope service. Taking all four stocks together, our dataset consists of more than 2.3 billion prices.

It is worth noting that even using the 10 seconds time frame (instead of the 1 second) we still have to deal with a lot of missing values, as the stocks do not necessarily trade every 10 seconds interval. Regarding this issue we also take the same approach of Koopman et al. (2018), by only updating the model when trading activity occurs. So we do not pad missing price changes with 0 price changes as done in Koopman et al. (2017). We consider not trading as different event than trading at the last price and we also want to make the analysis in this paper comparable to the analysis on the rest of the thesis.

We consider all prices ranging from the market opening at 10:00 to the market closing at 17:00. We model the intraday price changes, so we discard the overnight price changes in the cases where we estimate the model through more than one day. We apply a data-cleaning procedure before the analysis, in order to clean for exchange reporting errors, as recommended by Brownlees and Gallo (2006).

For descriptive statistics on the dataset refer to the data section of the paper Morier and Valls Pereira (2023), and reproduced in the section, Part I: Data, of the supplementary material of this paper.

5 Training Procedure

In this section we describe in more detail the training procedure applied for the networks, along with the estimation of other parameters. The network parameters and the parameters γ and δ are estimated together using the procedure described in this section.

5.1 Loss Function and Regularization

As the objective is to forecast a conditional density along with the volatility, we choose the base loss function to be the log-likelihood loss for both the training and testing procedures. In order to avoid over-fitting we add L^2 norm penalization to the loss function, commonly know as weight decay. So the loss function expression is given by

$$L(y) = -\sum_t \log\left(p(y_t|\mathcal{F}_{t-1};\theta_t)\right) - \frac{\alpha}{2}\left\|W\right\|_2^2 \tag{9}$$

where $W = \{W^{(in)}, W^{(out)}\} \cup \{W^{(i)}\}_{i\in\{1,\dots,L\}}$ is the vector containing all weights in the model.

The coefficient for the L^2 penalty is a model hyper-parameter and was obtained using a presample in 2017 by trial and error. We used $\alpha = 100$.

5.2 Optimization Algorithm

The Stochastic Gradient Descent (SGD) and its variants are probably the most used optimization algorithms for machine learning in general and for deep learning in particular. It consists in estimating the gradient using a mean of the minibatchs gradient estimates over a batch of data. The parameters are then updated using a hyper-parameter called learning rate

times the gradient estimate. So it is a first order gradient descent method where the estimates are updated using just a sub-sample of data.

The main difficulty in using SGD is appropriately setting the learning rate hyper-parameter as its choice greatly determines the predictive power of the trained model (according to Goodfellow et al. (2016)). If the learning rate is too high the algorithm makes the objective function to decay faster but becomes non-robust, failing to keep the objective function decaying. If it is too low, the optimization simply gets too much time to finish. And as one can see by the Newton's Method, it is better to have a lower learning rate when the curvature is higher and conversely.

In order to avoid such difficulties associated with the SGD method, we use the Adamax algorithm. It is an adaptive method introduced in Kingma and Ba (2014) in which the gradients are re-scaled by using the inverse of the weighted L^∞ norm applied to the previous calculated gradients. Adamax is a version of the Adam algorithm introduced in the same paper, which in its usual form uses the L^2 instead of L^∞.

Regarding the batch size, we used all the sample at once for each step of the optimization, so we actually used a 'batch gradient descent' method. As discussed in greater detail in the supplementary material of this paper, we used GPU for training the networks so having a greater batch size did not impact significantly the processing time. By using all data the gradients are calculated with more precision. We used 2000 epochs for optimizing each network, with no early stopping.

This above algorithm performed well in our pre-sample tests and was selected.

5.3 Initialization

Initialization is an important and not well understood part of the optimization process in deep learning. The starting values for the neural network heavily affects the results of the optimization process. In our case, the loss function used both for training and testing may not be numerically stable depending on the network outputs, which increases the concern regarding the initialization.

One standard initialization for the weights, which is default in many Deep Learning frameworks is given in equation (10)

$$W_{i,j}^{(k)} \sim U\left(-\frac{1}{\sqrt{m_k}}, \frac{1}{\sqrt{m_k}}\right) \tag{10}$$

where $k \in \{\text{in}, 1, \ldots, L, \text{out}\}$ is the layer, i, j are matrix coordinates and m_k is the number of columns for the matrix $W^{(k)}$.

We tested this initialization in our procedures and the results were unstable. For concluding the forecasting exercise in the next section we need to estimate more than a thousand different networks, so we need a robust optimization procedure.

So we changed this initialization scheme replacing the first column for matrices $W_{i,j}^{(k)}$ by the vector $(0, \ldots, 0, 1)^T$. This makes the output for the first neuron for all layers to be exactly the first feature, ew. We also set the bias b to zero for all layers except for the first one, which had a positive value in order to make sure that the value is positive. This is needed because ew is a z-score, and could be negative and we need to keep the value in domain where leakyRelu is the

identity function.

With this new initialization scheme the output for the network resembles the output for the EWMA model at the starting of the optimization and the optimization became much more robust, converging for the solutions presented in this paper with just one optimization trial.

Regarding the parameters outside the neural network, we initialized both γ and δ as zero.

6 Training and Forecasting Exercise

We applied the procedure in the previous section for training all the model variants in table 1. As done in paper 1, we estimated the models for all 5 consecutive business days periods of 2018, testing the model forecasts for the following 5 business days. The 5 days period was chosen so that the daily volatility seasonality can be better captured by the models.

In order to compare for the forecasts of the neural network models we used estimates for the models used in the paper Morier and Valls Pereira (2023):

SSM: State space model described in Morier and Valls Pereira (2023) with $\mu_t = \delta y_{t-1}$

SS: State space model described in Morier and Valls Pereira (2023) with $\mu_t = 0$

EW: σ_t is estimated by an EWMA process: $\hat{\sigma}_t^2 = \lambda y_{t-1}^2 + (1-\lambda)\hat{\sigma}_{t-1}^2$. λ is estimated using the fit sample (previous week). μ and γ are set to zero

M_1: σ_t is estimated non-parametrically by a moving average process using a rolling window of the past 90 returns. μ and γ are set to zero

M_2: σ_t is estimated non-parametrically by a moving average process using a rolling window of the past 900 returns. μ and γ are set to zero

E: the pmf of y_t is determined by the empirical probabilities of y_t on the fit sample.

The results for the exercise are described in the next two sections. The computational details of the implementation is described at the supplementary material of this paper.

7 Estimation Results

In this section we describe the main in-sample results for the training exercise described in last section. We compare the results obtained in Morier and Valls Pereira (2023)

7.1 Parameter Estimates

Now we describe the estimation results for the most complete model $\text{NN}_{v,\gamma,\mu}$. The model was estimated for 48 different 5 business days periods for all the four equities in our list.

We report the descriptive statistics for the parameter estimates at the table 2. For each equity / parameter we report the sample mean for the parameter estimates on the first row and the standard deviation in parenthesis on the second row. Notice that these statistics refer to estimates in different samples, so the mean and standard deviations are only indicative of

common estimated values through the year.They should not be interpreted as estimates and significance test statistics.

Table 2: Descriptive Statistics for the NN parameter estimates

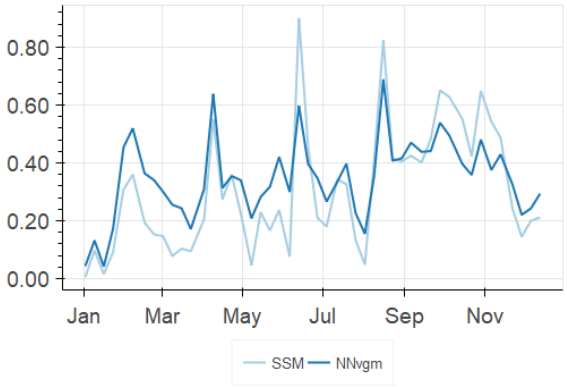
	Bradesco	Itau-Unibanco	Petrobras	Vale
δ	-0.200	-0.186	-0.225	-0.173
	(0.034)	(0.036)	(0.075)	(0.041)
γ	0.125	0.346	-0.103	0.307
	(0.111)	(0.136)	(0.089)	(0.174)

We notice that δ is negative for all stocks, exactly as in Morier and Valls Pereira (2023). This implies a negative dependency of the mean on previous returns, which is the expected value as noted in Morier and Valls Pereira (2023). We notice also that γ is only negative for Petrobras, also like in Morier and Valls Pereira (2023). In fact the estimates show different but similar values when compared to the estimates for SSM and SS models.

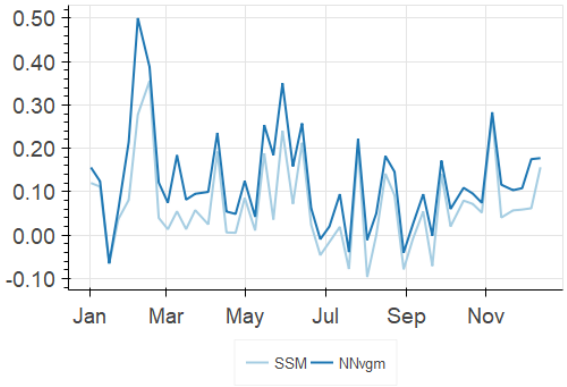
The similarities are more prominent when we compare the point estimates for each sample. Figures 1 and 2 shows the estimates for γ and δ respectively for both $\text{NN}_{v,\gamma,\mu}$ and SSM models.

Figure 1: Estimates for γ

(a) Itau-Unibanco

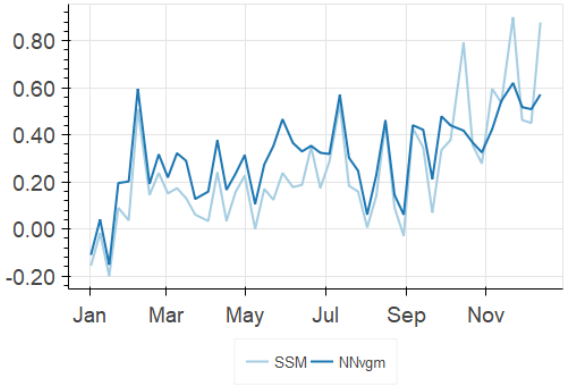


(b) Bradesco



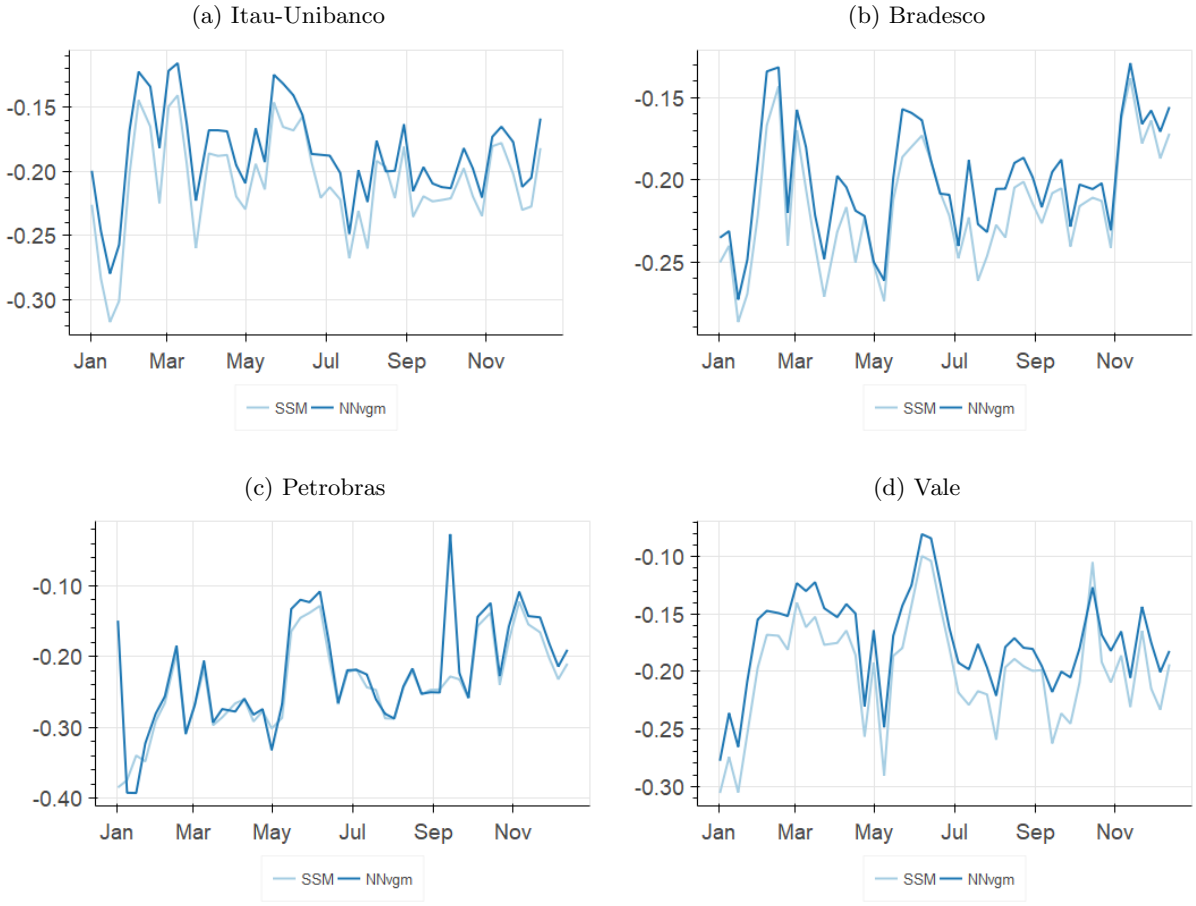
(c) Petrobras

(d) Vale



On both cases the lines plotted are similar for both models, although the level for the SSM model tended to be a little bit lower than for the neural network. This similarity shows the robustness of both estimation procedures, as the estimation procedures are totally different and the models are in fact also different.

Figure 2: Estimates for δ



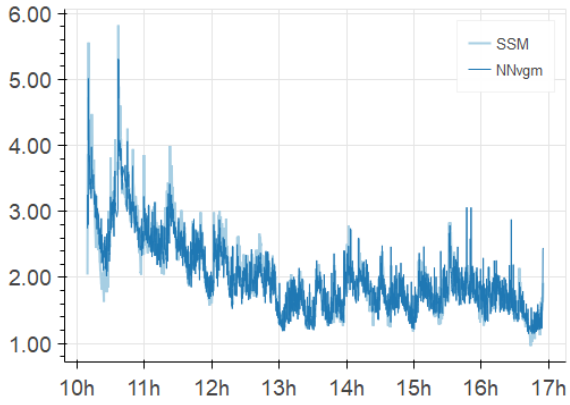
7.2 Filtered Volatility Example

We now compare volatility estimates from the Neural Network model $\text{NN}_{v,\gamma,\mu}$ and the SSM model. For the SSM model we used the Bootstrap Particle Filter in order to obtain the filtered forecasts. Figure 3 shows the path of the filtered volatility for the day July 6, 2018 for both models. Observe that the volatility decreases during the day as predicted by the mean seasonality in Morier and Valls Pereira (2023).

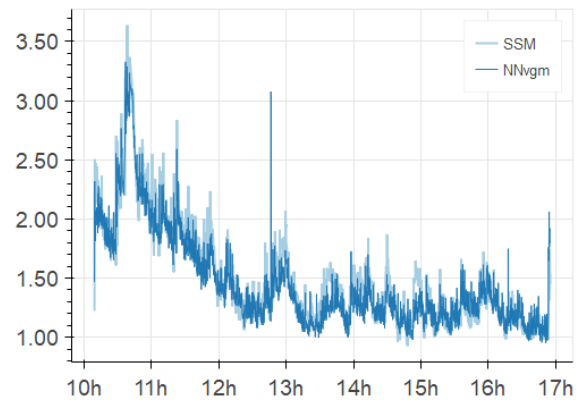
Note also that the volatility estimates for both models are strikingly similar, which is again a robustness check on the procedure of both models - as the models and procedures are totally different in both case. The volatility forecast for the Neural network appears to be little less volatile than the volatility forecast for the SSM model.

Figure 3: Filtered Forecast Volatility for July 6, 2018

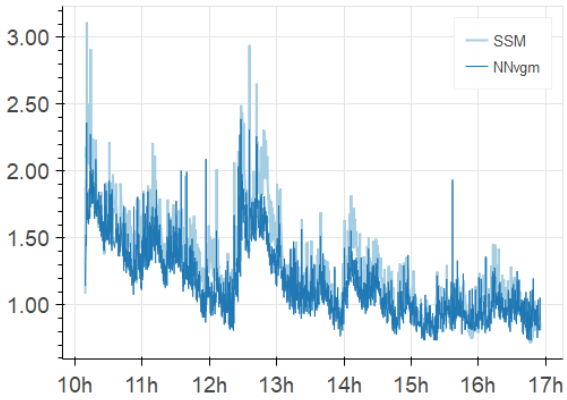
(a) Itau-Unibanco



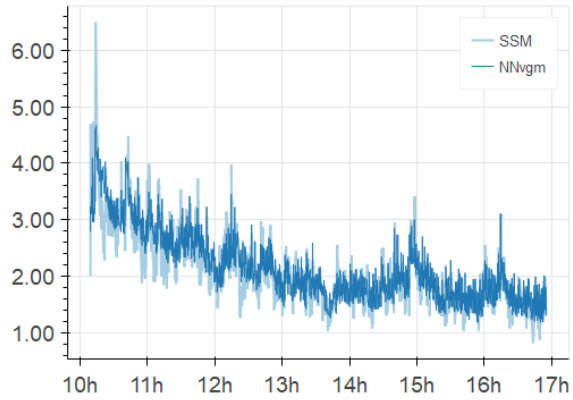
(b) Bradesco



(c) Petrobras



(d) Vale

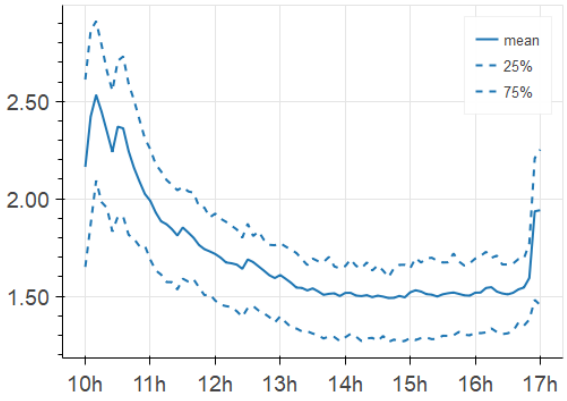


7.3 Intraday Seasonality

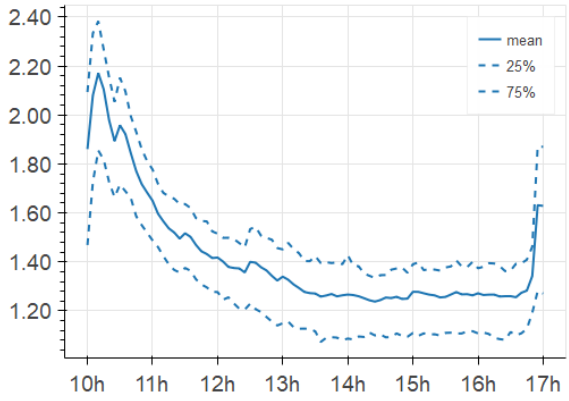
In this sub-section we analyze the descriptive statistics for the volatility seasonality. Figure 4 depicts the seasonality for the $\text{NN}_{v,\gamma,\mu}$ model, showing the mean and two percentiles for the volatility predicted by the model for each time of the day in the entire sample. Observe that for all stocks have a sharp decrease in the mean volatility predicted from the opening to the end of the day. This effect is also documented in Koopman et al. (2017). As stated in Morier and Valls Pereira (2023), one possible reason is that on the starting of the trading session there is more information for the markets to digest, including all the overnight news and economic releases, that are more common in the morning.

Figure 4: Average Seasonality for 2018

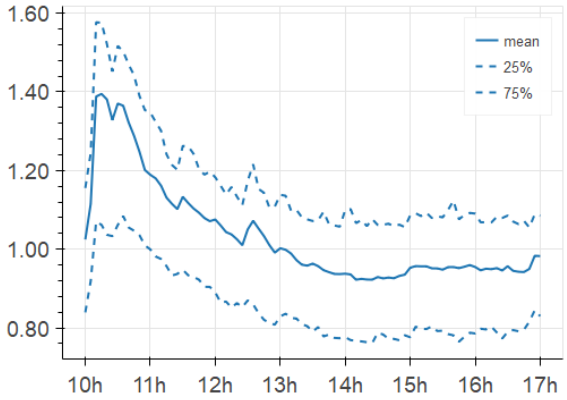
(a) Itau-Unibanco



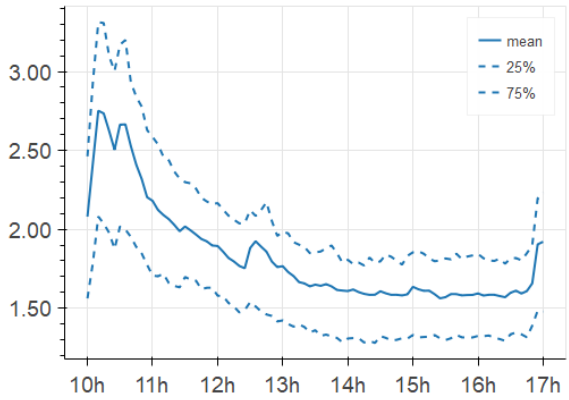
(b) Bradesco



(c) Petrobras



(d) Vale



The concept show in the figures is different however than the one show in figure 11 of Morier and Valls Pereira (2023). In that paper we plotted the theoretical seasonality arriving from the seasonality splines estimated in that model. Now we show percentiles for the model prediction directly. So we have more variability in the sample now, as the volatility varies not only because of the seasonality - the variability is also related to shocks. The curves are also less smooth beacuse there is no spline structure behind this estimates.

We can see in this figure an interesting pattern not captured by the splines: A spike in the volatility at the very ending of the trading session. This is actually the close auction, that happens on the last 5 minutes of the trading session for Brazilian equities. The splines are not able to capture this because of the smoothness imposed by such structure.

7.4 Non-linear Sensivity Analysis

As discussed in section 2, Deep Learning method is focused in building models that are good for forecasting. It is usually hard to get a deeper understanding on how the networks deal with each input feature, especially because the output of the network might be a highly non-linear function of the inputs. And the inputs interact with themselves inside the network, making interpretations harder.

Besides such difficulties, our objective in this section is to get a better understanding on how

the trained networks react to changes in the features, on average. We want to answer questions about what to expect in terms of volatility when the bid-ask spread rises, or if there is any detected relation between volume and volatility.

We make two exercises. First, we shock all input variables by adding one standard deviation, one at a time keeping the other constant, for the whole training sample. Then we compute the changes in the prediction for the volatility for each stock, for each sample time. Table 3 reports the descriptive statistics for these sensibilities, reporting means and standard deviations aggregating through time. The standard deviations are reported below the means, in parenthesis.

Table 3: Standardized Sensivity to shock in variables

	ew	y	y^2	ba	hl	v	W_0	W_1	W_2
Itau-Unibanco	0.30	-0.0023	0.078	0.13	0.090	0.036	0.082	0.028	-0.012
	(0.12)	(0.021)	(0.036)	(0.044)	(0.028)	(0.033)	(0.060)	(0.033)	(0.027)
Bradesco	0.26	-0.0039	0.060	0.095	0.061	0.022	0.067	0.021	-0.011
	(0.083)	(0.022)	(0.033)	(0.051)	(0.025)	(0.025)	(0.042)	(0.022)	(0.026)
Vale	0.35	-0.0094	0.083	0.11	0.092	0.076	0.12	0.051	-0.0016
	(0.13)	(0.024)	(0.037)	(0.059)	(0.033)	(0.054)	(0.088)	(0.038)	(0.030)
Petrobras	0.20	-0.0092	0.032	0.033	0.049	0.053	0.036	0.0098	-0.0023
	(0.11)	(0.015)	(0.021)	(0.019)	(0.031)	(0.036)	(0.038)	(0.026)	(0.014)

We observe that, on average, the models estimated point to a positive relation between the ew variable and the volatility output of the network, with a one standard deviation shock in that variable changing the volatility forecast between 0.20 and 0.35 depending on the stock.

The relation with y is negative on average, which is the expected sign for this sensibility, but with a low average value and a high dispersion. So the networks do not appear to be highly sensible to this variable, at least for a shock keeping all other variables constant. But as the relations modelled are non-linear, this input feature can interact with others leveraging its effects.

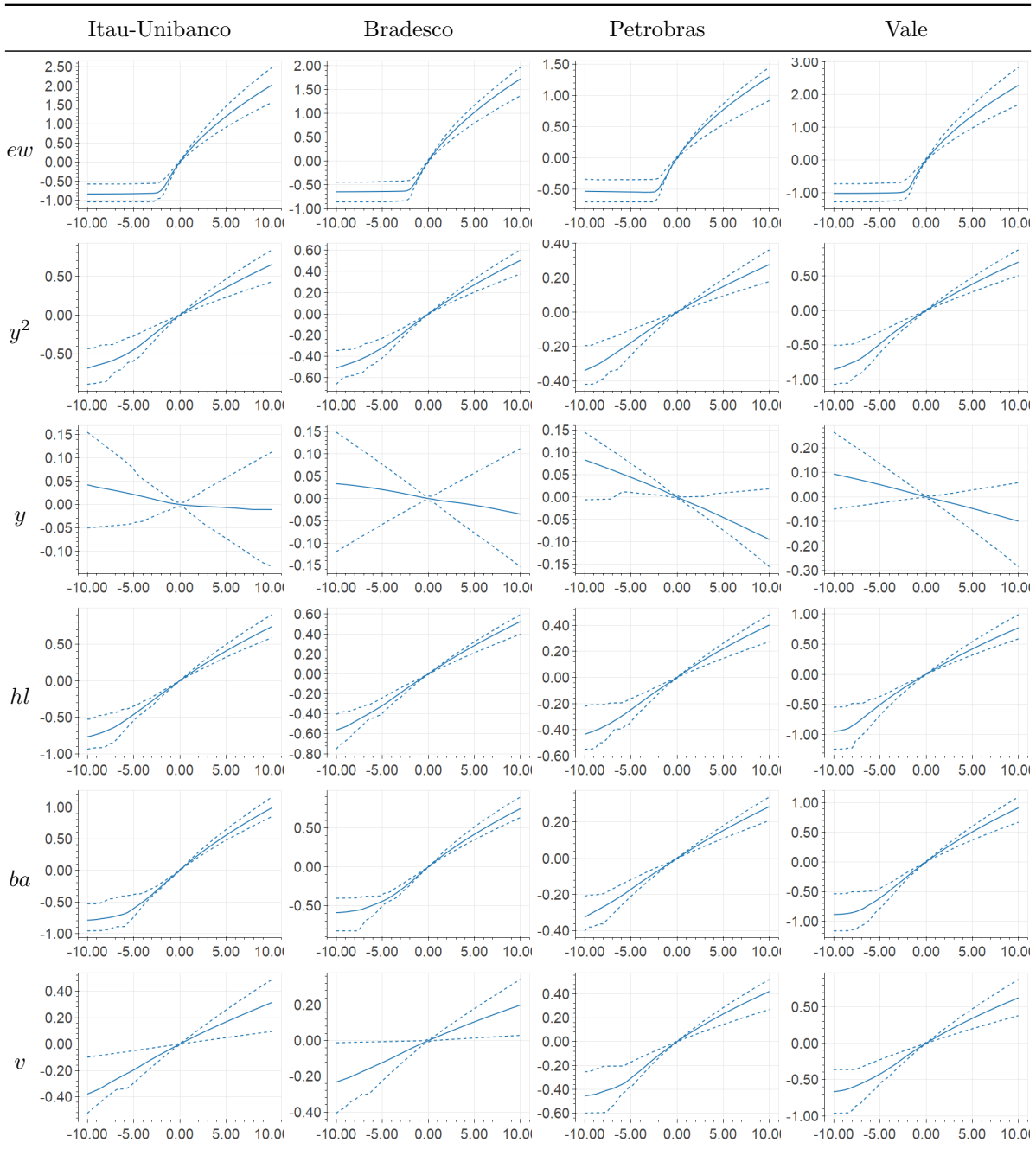
The relation with y^2 is clearly positive for all stocks, but with a impact below ew. The variables ba, hl and v all have positive effects in the output volatility, on average, for all stocks. As discussed before, this is the expected sign for the impact of these variables according to the literature.

The last three features, W_0, W_1 and W_2 are the seasonality splines. The first two tend to have positive values, with a higher values for the first, while the latter small negative values (with high relative dispersion). This is similar to the estimates of the coefficients β_0, β_1 and β_2 for the model SSM in the table 4 of Morier and Valls Pereira (2023), although in that table coefficients for β_1 are small negative on average.

In the second exercise we shock the input features by values ranging from -10 standard deviations and 10 standard deviations, keeping all other features constant. We shock all features, except for the seasonality splines. These shocks were again computed for the whole training sample, like in the previous exercise. The objective is to get an understanding on the non-linearities in the impact of each variable, holding all else constant. Observe that most input

variables are fat tailed, so that shocks with 10 standard deviations actually happen frequently in the sample. The results are reported on the figures collected a table 4.

Table 4: Non-linear sensivity to standardized shocks of selected variables



The solid line reflects the mean effect for the shock for each variable, while the dashed lines represent the 25% and 75% percentiles for the shocks. Looking at the figures in table 4 we can readily see that the higher relative dispersion of the impact is attained at the variable y. The other variables have a relatively lower dispersion.

Regarding the non-linearity of the effects, most variables have a S shaped impact format, where the effect for the impact is diminished for high absolute valued shocks. Perhaps the most salient effect is for the ew feature, where the impact seems to have a floor for negative values.

The concavity for negative values is also salient for hl, ba and v, for most stocks.

8 Forecasting Exercise Results

We now report the results for the walk-forward forecasting exercise described at section 6. We follow Koopman et al. (2017) and compare the forecasting performance of the models using log-likelihood loss. This measure is used because it takes into account the entire conditional density forecast into account, being able to compare models regarding not only the volatility forecasts but also forecasts for the mean and the format of the conditional distributions, including the tails and the frequency of zeros. So we are evaluating the capacity for the models to forecast not only the volatility, but their ability to forecast the conditional densities. For the forecast of time t and for all models, we use only data up to $t-1$.

Also following Koopman et al. (2017) and the methodology applied in paper 1, we compare the forecast performance by using Diebold-Mariano (DM) Statistics. In tables 5 and 6 we compute the mean log-likelihood Loss (column Log Loss) for each 10 seconds interval and the Diebold Mariano statistics comparing each model with the models $\text{NN}_{v,\gamma}$ and $\text{NN}_{v,\gamma,\mu}$ (columns $\text{DM}_{v,\gamma}$ and $\text{DM}_{v,\gamma,\mu}$). We used all the 2018 sample for computing this statistic, using the density prediction and the realized return y_t for every 10 seconds period - so this is a huge sample (more than 500,000 points) for each comparison. A positive number means that the model in the row outperforms the model in the column, while a negative number means the opposite.

Table 5: Forecasting Results - Itau-Unibanco and Bradesco

	Bradesco			Itau-Unibanco		
	Log Loss	$\text{DM}_{v,\gamma}$	$\text{DM}_{v,\gamma,\mu}$	Log Loss	$\text{DM}_{v,\gamma}$	$\text{DM}_{v,\gamma,\mu}$
NN_0	1.761	-30.13	-69.49	1.945	-46.35	-73.06
NN_v	1.755	-6.379	-63.01	1.936	-32.07	-63.85
$\text{NN}_{v,\gamma}$	1.754	-	-62.03	1.933	-	-53.62
$\text{NN}_{v,\gamma,\mu}$	1.737	62.03	-	1.919	53.62	-
EW	1.764	-28.19	-61.14	1.949	-41.11	-64.28
M_1	1.772	-37.02	-63.15	1.959	-49.34	-68.00
M_2	1.801	-78.92	-98.27	1.987	-83.51	-97.81
E	1.789	-53.56	-71.86	1.968	-50.97	-65.34
SS	1.761	-19.09	-53.49	1.948	-37.34	-61.49
SSM	1.742	23.71	-11.68	1.932	1.640	-28.56

Looking at the columns Log Loss and $\text{DM}_{v,\gamma,\mu}$ for both tables 5 and 6, we can see that the model $\text{NN}_{v,\gamma,\mu}$ outperforms all other neural network specifications and also outperform all other comparing models, including the SS and SSM models. This out-performance is highly statistically significant as the DM statistic closest to zero is -11.68 and the this statistic has a standard normal distribution asymptotically.

Looking at the columns Log Loss and $\text{DM}_{v,\gamma}$ we also see that the final NN version with

no mean modelled $NN_{v,\gamma}$ outperforms all models except for the models with mean modelled ($NN_{v,\gamma,\mu}$ and SSM). This shows the importance of modelling the mean for this kind of models, which is a novelty of the present work - regarding discrete high-frequency densities.

It also worth noting that this consistency in outperforming this set of models were not obtained by the SS model. This model actually did not outperform the EWMA model for Vale at the same sample, as pointed by the table 6 in Morier and Valls Pereira (2023). Another interesting point to note is that although $NN_{v,\gamma}$ did not outperform SSM for all stocks, it did outperform for one of them (Vale).

Table 6: Forecasting Results - Petrobras and Vale

	Petrobras			Vale		
	Log Loss	$DM_{v,\gamma}$	$DM_{v,\gamma,\mu}$	Log Loss	$DM_{v,\gamma}$	$DM_{v,\gamma,\mu}$
NN_0	1.476	-32.03	-75.98	2.018	-35.09	-62.28
NN_v	1.473	-17.88	-71.08	2.012	-24.47	-57.65
$NN_{v,\gamma}$	1.471	-	-68.67	2.010	-	-52.60
$NN_{v,\gamma,\mu}$	1.448	68.67	-	1.997	52.60	-
EW	1.480	-35.90	-75.36	2.023	-31.29	-53.89
M_1	1.486	-46.21	-80.39	2.033	-41.99	-59.74
M_2	1.500	-71.67	-98.33	2.065	-82.16	-94.76
E	1.517	-98.75	-119.1	2.046	-45.70	-58.24
SS	1.477	-26.17	-71.23	2.027	-37.60	-57.88
SSM	1.453	31.53	-12.58	2.015	-9.123	-36.13

We arrange the comparing DM Statistic for comparing all the models individually in table 7. The calculation is exactly the same as the tables 5 and 6, but now we can compare all pair of models. A positive number means (again) that the model in the row outperforms the model in the columns, where a negative number means the opposite.

We can see that even the simplest neural network model considered, NN_0, outperforms the *EW* model for all stocks - which does not happen even for the SS model, that underperforms *EW* for Vale. NN_0 also outperforms SS for all stocks, altough the outperformance for Bradesco is not statistically significant (DM=-0.5). Recall that this model does not even allow γ to be different from zero and uses the same input variables as SS.

Comparing the performance of NN_v and NN_0 we see that the newly added variables (*ba*, *hl* and v) significantly enhances the predictive power for the model. NN_v beats NN_0 by a high margin, with DM statitics above 20 in absolute value terms.

Comparing $NN_{v,\gamma}$ and NN_v we also see that including the γ the forecasts' loss becomes significantly lower, also by a large margin. Finally, comparing $NN_{v,\gamma,\mu}$ and $NN_{v,\gamma}$ we see again the importance of modelling the mean. This appear to be the single most important feature comparing the forecasting performance, as the DM statistics of this last comparison is even higher in absolute terms then the last comparisons.

Table 7: DM Statistics Results

Itau-Unibanco

	$NN_{v,\gamma,\mu}$	$NN_{v,\gamma}$	NN_v	NN_0	EW	SS
SSM	-28.6	1.6	7.9	24.8	27.5	42.6
SS	-61.5	-37.3	-27.5	-7.3	3.1	-
EW	-64.3	-41.1	-33.6	-14.7	-	-
NN_0	-73.1	-46.4	-35.0	-	-	-
NN_v	-63.9	-32.1	-	-	-	-
$NN_{v,\gamma}$	-53.6	-	-	-	-	-

Bradesco

	$NN_{v,\gamma,\mu}$	$NN_{v,\gamma}$	NN_v	NN_0	EW	SS
SSM	-11.7	23.7	24.5	37.3	41.4	46.0
SS	-53.5	-19.1	-17.3	-0.5	9.2	-
EW	-61.1	-28.2	-27.0	-12.4	-	-
NN_0	-69.5	-30.1	-28.9	-	-	-
NN_v	-63.0	-6.4	-	-	-	-
$NN_{v,\gamma}$	-62.0	-	-	-	-	-

Petrobras

	$NN_{v,\gamma,\mu}$	$NN_{v,\gamma}$	NN_v	NN_0	EW	SS
SSM	-12.6	31.5	34.7	40.7	47.4	45.6
SS	-71.2	-26.2	-15.3	-4.3	12.8	-
EW	-75.4	-35.9	-29.6	-20.9	-	-
NN_0	-76.0	-32.0	-24.2	-	-	-
NN_v	-71.1	-17.9	-	-	-	-
$NN_{v,\gamma}$	-68.7	-	-	-	-	-

Vale

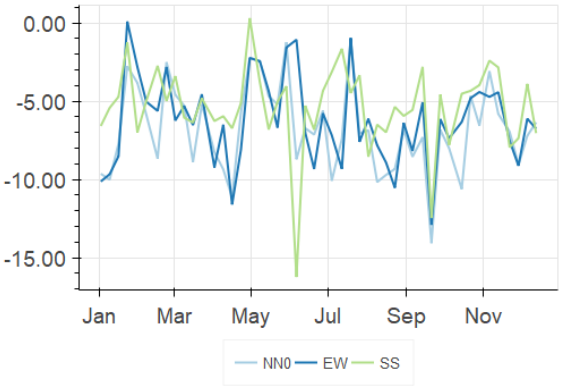
	$NN_{v,\gamma,\mu}$	$NN_{v,\gamma}$	NN_v	NN_0	EW	SS
SSM	-36.1	-9.1	-5.4	5.8	10.9	33.5
SS	-57.9	-37.6	-32.3	-19.6	-7.9	-
EW	-53.9	-31.3	-26.8	-13.4	-	-
NN_0	-62.3	-35.1	-27.2	-	-	-
NN_v	-57.7	-24.5	-	-	-	-
$NN_{v,\gamma}$	-52.6	-	-	-	-	-

We also computed the DM statistic individually for each 5-day period. We show the results for the comparison with the $NN_{v,\gamma}$ model in figure 5 and the comparison with the $NN_{v,\gamma,\mu}$ model in figure 6. We can see in the figures that all previously reported out-performances are consistent through most of the 5-days samples, with few points of exception.

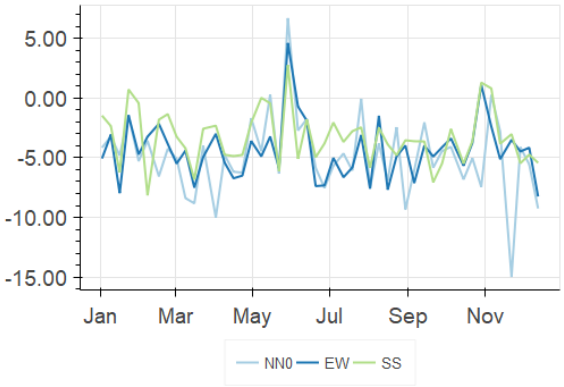
Each point in these figures represent the statistics applied to the 5-days period, which constitutes more than 20.000 data points. In figure 3 we see that the test statistics are similar for all models compared to $NN_{v,\gamma}$, and they oscillate around the level of -5.

Figure 5: Weekly DM Score for $NN_{v,g}$ Model

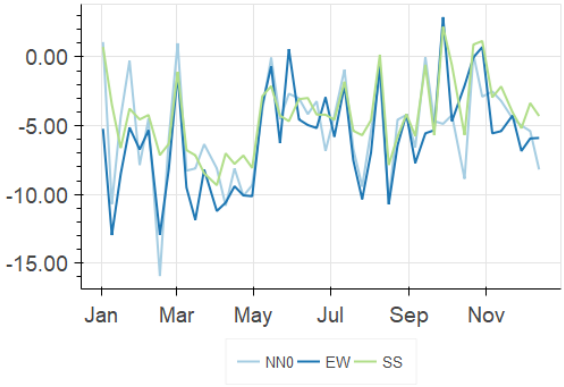
(a) Itau-Unibanco



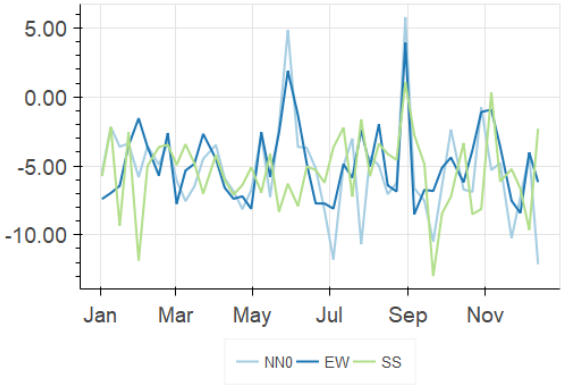
(b) Bradesco



(c) Petrobras



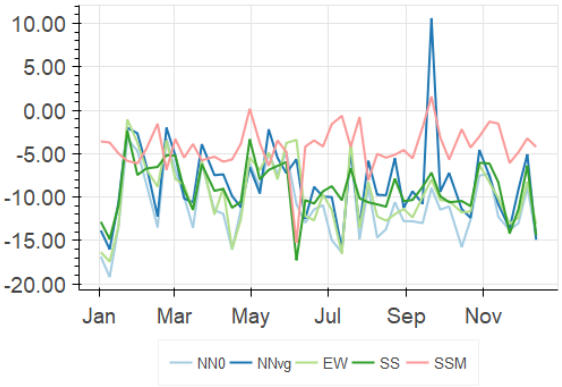
(d) Vale



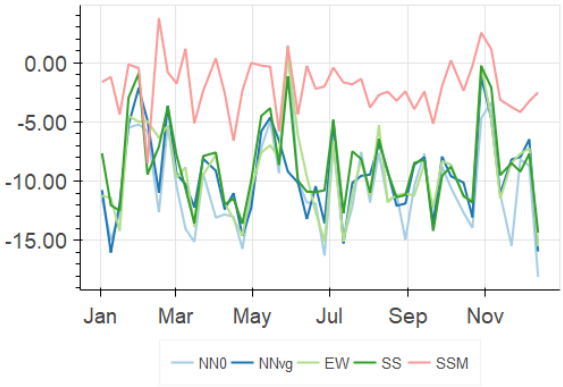
In figure 6 we see that the test statistics are similar for all models compared to $NN_{v,\gamma,\mu}$, except for the SSM model, and they oscillate around the level of -10. The comparison with the SSM yields a smaller statistic (in absolute value terms), mostly ranging from -2 to -5.

Figure 6: Weekly DM Score for $\text{NN}_{v,g,m}$ Model

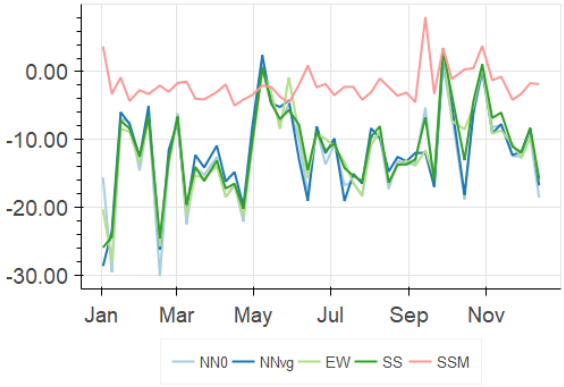
(a) Itau-Unibanco



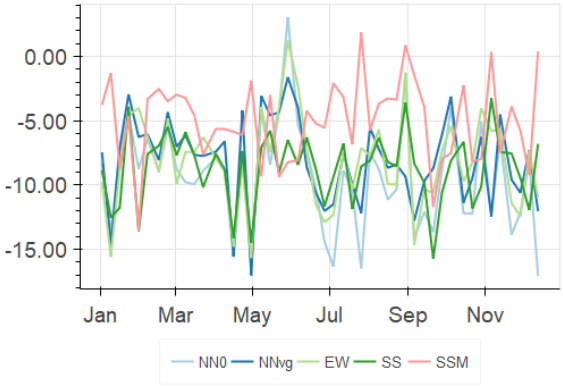
(b) Bradesco



(c) Petrobras



(d) Vale



9 Final Remarks

In this paper we developed a new model for forecasting discrete high-frequency densities and volatility. The model is composed of a deep feed-forward neural network to forecast volatility, an AR model for the mean and uses the Modified-Skellam distribution for computing densities. We are unaware of any other works using Deep Learning to predict volatility at this time-scale. Besides the forecasting performance, this model is easier to implement and computationally faster than the State Space counterparts.

We used this model to forecast conditional intraday discrete high-frequency densities and volatilities, contributing to the nascent literature on this subject. In this new model we added new variables not present in previous papers on high frequency volatility forecasting, including the bid-ask spread, the high-low interval spread and the volume traded. We conducted a sensibility analysis of the networks estimated and found that these variables affected the volatility forecasts, finding a positive relation between all of them and the volatility. We also showed that this relation is not linear, appearing to have an S shape (with more concavity on for negative shocks). We also included the previous price change in the model, but sensibility analysis was not conclusive. We show that the inclusion of these variables increase the prediction power of the model.

We compare the in-sample filtered volatilities, seasonality, and common parameters obtained

with the Neural Network Model with the State Space models derived at paper 1 - and find many similarities. But, more interestingly, we conducted an extensive out-sample walk-forward forecasting exercise, involving the entire year of 2018 and about 2 billion data points of price changes of four distinct stocks. The new Neural Network model showed the best forecasting performance for all the stocks when compared to State Space Models and EWMA models. This outperformance was consistent between all the stocks, across time and with and without modeling the mean of the density.

References

Andersen, T. G. (1996). Return volatility and trading volume: An information flow interpretation of stochastic volatility. *The Journal of Finance*, 51(1):169–204.

Andersen, T. G. and Bollerslev, T. (1997). Intraday periodicity and volatility persistence in financial markets. *Journal of Empirical Finance*, 4(2-3):115–158.

Andersen, T. G., Bollerslev, T., Diebold, F. X., and Labys, P. (2001). The distribution of realized exchange rate volatility. *Journal of the American Statistical Association*, 96(453):42–55.

Barndorff-Nielsen, O. E. and Shephard, N. (2002). Econometric analysis of realized volatility and its use in estimating stochastic volatility models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(2):253–280.

Brownlees, C. T. and Gallo, G. M. (2006). Financial econometric analysis at ultra-high frequency: Data handling concerns. *Computational Statistics & Data Analysis*, 51(4):2232–2245.

Bucci, A. (2019). Realized volatility forecasting with neural networks.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

de Prado, M. L. (2018). *Advances in Financial Machine Learning*. Wiley Publishing, 1st edition.

Donaldson, R. G. and Kamstra, M. (1997). An artificial neural network-garch model for international stock return volatility. *Journal of Empirical Finance*, 4(1):17–46.

Engle, R. F. and Ng, V. K. (1993). Measuring and testing the impact of news on volatility. *The journal of finance*, 48(5):1749–1778.

Engle, R. F. and Sokalska, M. E. (2012). Forecasting intraday volatility in the us equity market. multiplicative component garch. *Journal of Financial Econometrics*, 10(1):54–83.

Glosten, L. R., Jagannathan, R., and Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. *The journal of finance*, 48(5):1779–1801.

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

Hajizadeh, E., Seifi, A., Zarandi, M. F., and Turksen, I. (2012). A hybrid modeling approach for forecasting the volatility of s&p 500 index return. *Expert Systems with Applications*, 39(1):431–436.

Hansen, P. R. and Lunde, A. (2006). Realized variance and market microstructure noise. *Journal of Business & Economic Statistics*, 24(2):127–161.

Harvey, A. and Koopman, S. J. (1993). Forecasting hourly electricity demand using time-varying splines. *Journal of the American Statistical Association*, 88(424):1228–1236.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560.

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

Koopman, S. J., Lit, R., and Lucas, A. (2017). intraday stochastic volatility in discrete price changes: the dynamic skellam model. *Journal of the American Statistical Association*, 112(520):1490–1503.

Koopman, S. J., Lit, R., Lucas, A., and Opschoor, A. (2018). Dynamic discrete copula models for high-frequency stock price changes. *Journal of Applied Econometrics*, 33(7):966–985.

Koopman, S. J., Lucas, A., and Scharth, M. (2015). Numerically accelerated importance sampling for nonlinear non-gaussian state-space models. *Journal of Business & Economic Statistics*, 33(1):114–127.

Kristjanpoller, W., Fadic, A., and Minutolo, M. C. (2014). Volatility forecast using hybrid neural network models. *Expert Systems with Applications*, 41(5):2437–2442.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3.

Morier, B. and Valls Pereira, P. L. (2023). Modeling high frequency intraday returns by non-linear state space models. Technical report, CEQEF Working papers.

Poirier, D. J. (1973). Piecewise regression using cubic splines. *Journal of the American Statistical Association*, 68(343):515–524.

Shephard, N. (2005). *Stochastic volatility: selected readings*. Oxford University Press on Demand.

Wyart, M., Bouchaud, J.-P., Kockelkoren, J., Potters, M., and Vettorazzo, M. (2008). Relation between bid–ask spread, impact and volatility in order-driven markets. *Quantitative Finance*, 8(1):41–57.

Xu, X. E., Chen, P., and Wu, C. (2006). Time and dynamic volume–volatility relation. *Journal of Banking & Finance*, 30(5):1535–1558.

Yang, D. and Zhang, Q. (2000). Drift-independent volatility estimation based on high, low, open, and close prices. *The Journal of Business*, 73(3):477–492.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Supplentary Material for the paper Forecasting Intraday Volatility and Densities using Deep Learning

In the supplemntary material for the paper Forecasting Intraday Volatility and Densities using Deep Learning it is presented to parts. This first part is a description of the data used in this paper and also in Morier and Valls Pereira (2023). The second part is the computational aspects for this paper.

Part I: Data

Our dataset is formed by intraday prices for four Brazilian Stocks: Petrobras (PETR4), Vale (VALE3), Itau-Unibanco (ITUB4) and Bradesco (BBDC4) for the entire year of 2018. These stocks are among the most liquid Brazilian stocks. The data used in this paper consists of the closing trading prices for the intraday interval of 10 seconds obtained from B3 exchange by using Thomson Reuters Datascope service. Taking all four stocks together, our dataset consists of more than 2.3 billion prices.

In the literature we have studies with approaches similar to this paper that uses both 1 second (Koopman et al. (2017)) and 10 seconds time interval (Koopman et al. (2018)). The first one is concerned with the volatility dynamics, analyzing univariate time series while the second makes a multivariate analysis. The 10 seconds time frame is more convenient for the multivariate analysis as it raises the probability of the joint event of simultaneous trading for the assets considered in the analysis, as argued in Koopman et al. (2018). A reason for using 10 seconds interval even in the univariate case is that although we chose the most liquid Brazilian stocks, these stocks are less liquid than the American counterparts used in Koopman et al. (2017). For these two reasons we chose the 10 seconds interval for the three papers in this thesis. There is a last advantage in this choice: it makes the whole thesis comparable and based in the same dataset, which is described in this section.

It is worth noting that even using the 10 seconds time frame (instead of the 1 second) we still have to deal with a lot of missing values, as the stocks do not necessarily trade every 10 seconds interval. Regarding this issue we also take the same approach of Koopman et al. (2018), by only updating the model when trading activity occurs. So we do not pad missing prices as done in Koopman et al. (2017). Repeating prices when trading activity do not occur has the effect of equating the event of no trade in a time t to the (different) event of a trade happening in time t with the same price as $t-1$, which is not desirable. This procedure would inflate the number of zero price changes.

We consider all prices ranging from the market opening to the market closing. We model the intraday price changes, so we discard the overnight price changes in the cases where we estimate the model through more than one day. We apply a data-cleaning procedure before the analysis, in order to clean for exchange reporting errors, as recommended by Brownlees and Gallo (2006).

In table 8 we show descriptive statistics for the entire sample. We notice that there is high occurrence of no change in price (0's) and changes of magnitude 1 (± 1). The distribution for all stocks is also fat tailed, as the minimum and maximum price changes are more than 30 times

the standard deviation. This last fact can also be checked by looking at the 0.1% and 99.9% percentiles that are more than 5 times the standard deviation. These occurrences shows not only a fatter than normal distribution, but also fatter than the Skellam or Modified Skellam distributions.

Table 8: Descriptive Statistics for the 2018 Sample

	# prices	%0	%±1	std	min	max	0.1%	99.9%
Itau-Unibanco	585,142	33	36	1.90	-58	38	-9	9
Bradesco	583,730	36	40	1.57	-63	35	-8	7
Petrobras	587,000	43	43	1.24	-33	26	-6	5
Vale	567,534	31	36	2.09	-49	53	-11	11

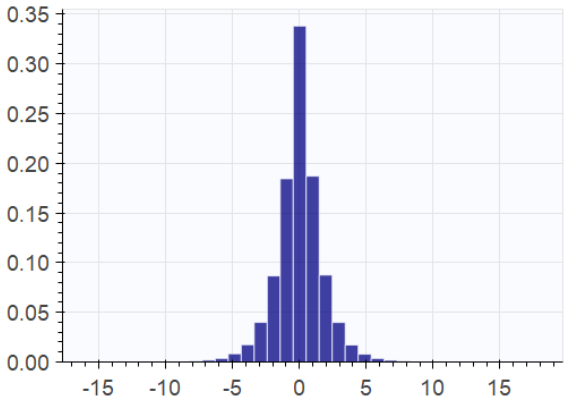
Note: We report prices as the number of 10 seconds closing prices considered in the sample, %0 as the percentage of 0 price changes, $\% \pm 1$ as the percentage of $\% \pm 1$ price changes, std as the standard deviation, min as the minimum price change in ticks, max as the maximum price change in ticks, 0.1% as the 0.1% percentile of price changes in ticks and 99.9% as the 99.9% percentile of price changes in ticks the data, including whatever notes are needed.

In figure 7 we plot the histograms for the price changes for the selected stocks in the entire year of 2018. In order to make this plot we discard the most severe price changes, that is, we only consider the percentiles 0.1% to 99.9%, as the minimum and maximum values exceed the range of the graphs. We can see from this plots that the price changes takes few values most of the time emphasizing the need for a discrete distribution for this data.

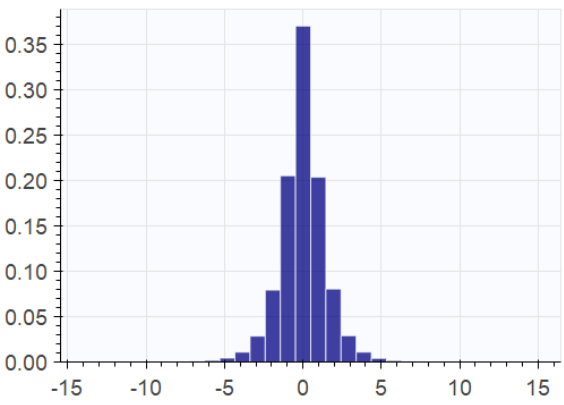
We also note from figure 7 that the least volatile stock in terms of tick volatility, Petrobras, attains less values most of the time for its price changes, while compared with the others. This does not mean that Petrobras stock is less volatile than the others considering usual price log returns . In fact, Petrobras stock is more volatile than the other in this sample. The histogram dispersion for each stock is actually determined by the stock volatility divided by the ratio of the tick size to its price. And for our sample this ratio is higher for Petrobras than for the others. This fact forces the price changes to attain less distinct values most of the time.

Figure 7: Histogram for price changes in 2018

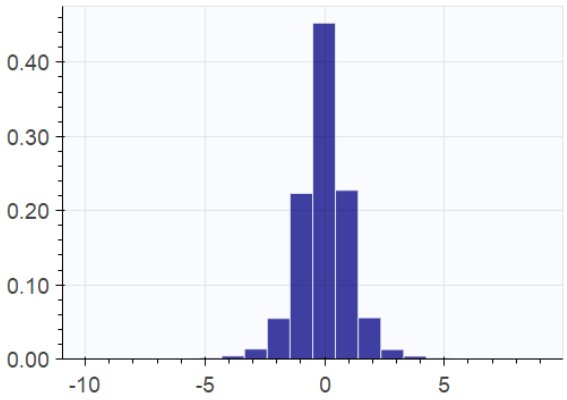
(a) Itau-Unibanco



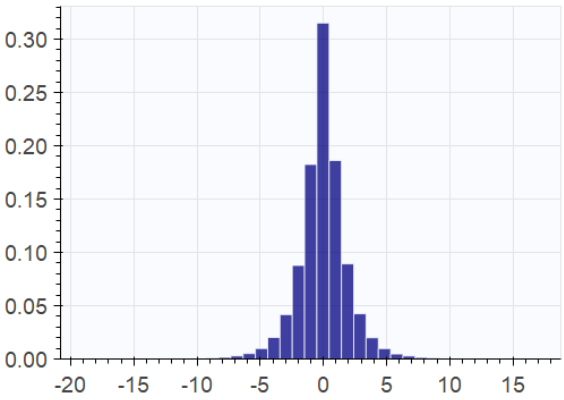
(b) Bradesco



(c) Petrobras



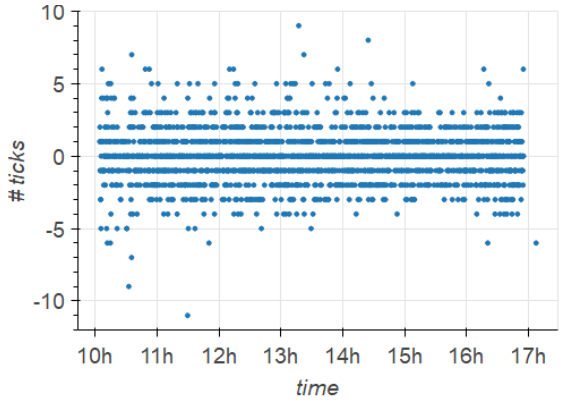
(d) Vale



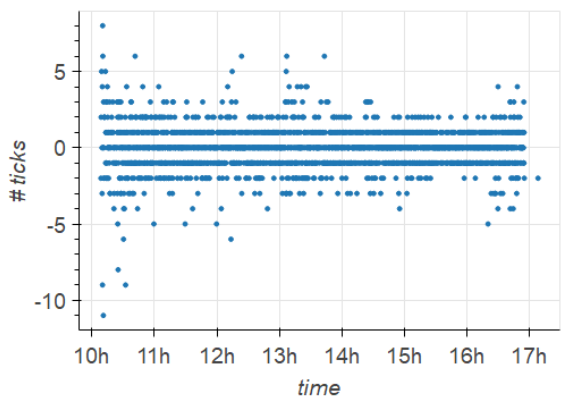
Note: In figure 8 we plot the price changes for the selected stocks in number of ticks for a selected date (June 20, 2018). We can see again that Petrobras price changes attain less distinct values than the other stocks.

Figure 8: Price changes for assets on June 20, 2018

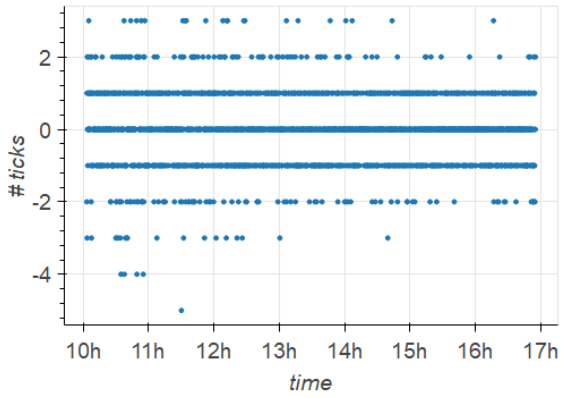
(a) Itau-Unibanco



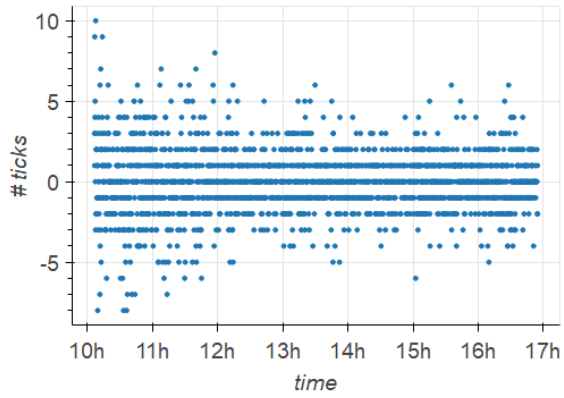
(b) Bradesco



(c) Petrobras



(d) Vale



Part II: Computational Aspects

The computations needed for the exercises in this article would be very hard to carry a few years ago. Recent advances in IT, both for hardware and software made the tasks in this work less difficult. In this appendix we describe the technologies used in this work.

A Working in Parallel with GPU and CUDA

One of the main advances in hardware for the last decade is the cheap availability of parallel processing. First, we now have access to GPU units, that were popularized in along the 2000's decade. And they were specialized for deep learning tasks over the last decade.

A graphics processing unit (GPU) is a specialized processor unit initially designed to accelerate the creation of images for output to a display device. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Their highly parallel structure makes them more efficient than general-purpose central processing units (CPUs) for algorithms that process large blocks of data in parallel. In a personal computer, a GPU can be present on a video card or embedded on the motherboard.

The chip maker NVIDIA created the CUDA (Compute Unified Device Architecture), which

is a parallel computing platform and application programming interface (API) model. It allows software developers and software engineers to use a graphics processing unit (GPU) for general purpose processing. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels. In top of CUDA, nVidia maintains a set of specialized libraries. Three of them are of great interest for the computations involved in this work: cuBLAS, cuSOLVER and cuDNN.

The NVIDIA cuBLAS library is a fast GPU-accelerated implementation of the standard basic linear algebra subroutines (BLAS). NVIDIA cuSOLVER library provides a collection of dense and sparse direct solvers with the intent of providing useful LAPACK-like features, such as common matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver. Lastly, the NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. The cuDNN library provides highly efficient implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

The CUDA framework became the standard framework for general processing in GPU's. Fortunately end users do not need to deal with this infrastructure directly. The availability of cuBLAS, cuSOLVER and cuDNN made easy for software developers to develop high level packages that build in the CUDA architecture in order to make scientific computing. As in the case of BLAS and LAPACK for the CPU's, most users actually access the libraries transparently, using high level wrappers. This is the case for Pytorch, Tensorflow and MXNet, which are deep learning frameworks backed by Google, Facebook and Apache Foundation, respectively. They provide generic access to classes implementing Deep Learning features that can run either in CPU or GPU, with almost the same code. They also provide generic matrix / tensor libraries that make the use of BLAS, LAPACK, cuBLAS and cuSOLVER transparent to the user, computing the transformations on both CPU and GPU with the same interface.

For the present work we computed all the Neural Network training procedures on the GPU, running four training problems at the same time. We used a gaming notebook with a CUDA enabled NVIDIA GeForce GPU. The networks were coded using Pytorch framework. We trained four models, for four stocks and 48 periods (768 models) for 2000 epochs. The training took less then 1 day to complete. We estimate that using the CPU the computation wold be about 20 times slower.

B Distributed Computing: AWS EC2 Cluster

Parallelizing computations over the GPU is a great and economical way of processing many tasks in a short period of time. However, this approach is no panacea: there are limitations and practical difficulties in using GPU.

First, not all problems are well suited for processing in parallel. A great example is time series analysis. Most time series models contain some time dependency, which likely involves a recurrence that usually must be computed sequentially. The task of computing an AR model for 20.000 timestamps or running a Kalman filter for the same lenght cannot be parallelized. And these applications were ubiquitous in papers 1 and 2 computations. When the task cannot

be parallelized, it makes absolutely no sense to use GPU's, as each processor in the GPU has limited capabilities when compared with the CPU. In these cases computing in GPU is actually slower. The GPU main advantage is the number of processors and this advantage is not relevant in these scenarios.

Second, some times it is hard to write a code that runs entirely on the GPU even when the problem can be parallelized. Taking the example of the Kalman filtering, one cannot parallelize the filtering procedure from one time series, but could make the filtering of several time-series at the same time. And this would be the case of the Kalman Simulator Smoothing algorithm used to sample for Gaussian distribution obtained by the NAIS procedure. So that part of the algorithm could be parallelized. But in order to do this on the GPU, one would need to write extremely efficient compiled functions for the Kalman Filtering over the GPU, with custom Kernels, which is hard and time consuming. Another difficulty faced is the possible unavailability of certain specialized mathematical functions over the GPU.

So we opted to compute the models of the first two papers on the CPU. The computations involved the optimization of 2 models for 4 stocks over 48 periods in the first paper, and 2 models for 2 pairs of stocks over 48 periods in the second paper. The forecasting part was also computationally expensive for the State Space models in both papers, as the Bootstrap Particle Filter was used. These two papers involved the computation of the estimation procedure of 960 models and the computation of the particle filtering for 768 models. Computing all these models over a single desktop using one CPU would take many days to complete, possibly some weeks. So we opted to use a cluster on Amazon AWS EC2 to make such computations, distributing the computing process through 5 instances, each with 72 processors each. we used the c5.18xlarge instance type. The computations took about 1 day to complete.

C Approximating Functions

Another difficulty faced is the unavailability of certain specialized mathematical functions over the GPU. This is the case for the modified Bessel I function, needed for the Skellam pmf computation. Actually, even the computation of this function on the CPU is too slow, as it involves the expansion of series for computing each value. The function might not be numerically stable also, even when using the exponential scaled version.

So we opted to approximate this function by splines. The same was done with the 2 dimensional Gaussian CDF function, needed for the Gaussian copula computation. The implementations became faster and more numerically stable, maintaining the differentiability needed for the optimizations.

D Compiling Python Code

A final important topic concern how to generate low level efficient code for implementing specialized algorithms needed for the paper. The language of choice for this paper was python because of its easy of use, it does have large scientific and generic computing libraries and because it is open source. But the language is not particularly fast. This is not a problem if the

problem at hand is not computationally intensive or if most of the computation can be done by calling compiled code already available in python libraries. Unfortunately the computations for this work did not fell into any of these two categories.

Using python to write the time-series loops needed for the algorithms in this work would lead to implementations 5 to 10 times slower than low level code implementations, making the computations unfeasible in practice. Writing this code directly in a low level language would be too time consuming for the purposes of the current work. Our solution involved generating low level compiled code using python package Numba. Numba allows on-the-fly/transparent compilation of python code, approaching low level C/Fortran Speed. Only a subset of the Python language is supported, so some effort compatible code is needed. But it involves much less effort than writing C/Fortran specialized functions.