Deep Learning, Jumps, and Volatility Bursts[*]

Oksana Bashchenko[†]and Alexis Marchal[‡]

October 19, 2019

We develop a new method that detects jumps nonparametrically in financial time series and significantly outperforms the current benchmark on simulated data. We use a long short-term memory (LSTM) neural network that is trained on labelled data generated by a process that experiences both jumps and volatility bursts. As a result, the network learns how to disentangle the two. Then it is applied to out-of-sample simulated data and delivers results that considerably differ from the benchmark: we obtain fewer spurious detection and identify a larger number of true jumps. When applied to real data, our approach for jump screening allows to extract a more precise signal about future volatility.

Keywords: Jumps, Volatility burst, High-frequency data, Deep learning, LSTM.

[*]We are thankful to Pierre Collin-Dufresne and Eric Jondeau for their helpful comments.

[†]UNIL and Swiss Finance Institute. Email: oksana.bashchenko@unil.ch

[‡]EPFL and Swiss Finance Institute. Email: alexis.marchal@epfl.ch

1. INTRODUCTION

A popular stochastic process used to describe the evolution of prices is the so called jump-diffusion model. Under this specification the price is modeled as combination of a drift, a Brownian term and a jump process. We are interested in classifying returns into innovations coming from the continuous Brownian or the discontinuous jump process. However, data being inherently observed at discrete time points the decomposition is not straightforward. At a given frequency, a large Brownian increment will be indistinguishable from a small size jump. This is why jumps are often mistaken for bursts of volatility (i.e. volatility jumps) at "reasonable" frequencies (around 5 minutes). One solution to this problem is to select the highest frequency possible (use so called ultra high-frequency data). Given the continuity of the Brownian component, diffusion-driven innovations will become smaller as the sampling frequency increases, letting the true jumps emerge and making them easier to be detected. The drawback of this approach is that for the highest frequencies the financial data is prone to be contaminated with microstructure noise.

This paper presents a new method that disentangles jumps nonparametrically and is able to separate them from bursts of volatility. Our approach detects the exact time of a jump within a day and is therefore benchmarked to the fundamental test of Lee and Mykland (2008) (LM henceforth). Our neural network largely outperforms the benchmark in presence of jumps inside the volatility of the price process, achieving smaller type I and type II errors, and performs equally good with continuous volatility. In order to achieve these results we use a long short-term memory network. It is first trained on labelled data that were generated from Monte-Carlo simulations. The network is then applied to classify every single return on out-of-sample simulated data.

On real data, since labels are not available we assess the performance of our method via an event study and a volatility forecasting exercise. Both of them confirm the higher accuracy of our approach.

The rest of the paper is organized as follows. Section 2 describes the literature. Section 3 compares our method with the "classical" statistical tests and presents the main benchmark. Section 4 explains the main idea behind the architecture and the training of the network. Section 5 assesses the performance of our method on out-of-sample simulated data. Section 6 applies the algorithm to classify real data. Finally section 7 concludes.

2. LITERATURE REVIEW

The importance of differentiating between the two sources of risk, the jump component and the continuous Brownian part, is outlined in Aït-Sahalia (2004). The existence of jumps impacts option prices, risk management and asset allocation. Early papers dedicated to the testing for jumps presence proposed a parametric approach using jump-diffusion models with constant volatility. More recent papers have added a stochastic volatility component and state-dependent jump parameters. The estimation for those models, however, is complex and subject to model errors.

A separate branch of literature has focused on nonparametric methods working with intraday data instead of the previously used low-frequency observations. Ole E Barndorff-Nielsen and Shephard (2004) introduced the notion of bipower variation (BPV) which is a nonparametric estimator of integrated variance that is consistent in presence of jumps. It established the foundations for multiple nonparametric tests that were proposed afterwards.[1] However most of the tests only allow to assess the continuity of the sample path during a given time period, they do not aim at detecting the exact jump times.

Lee and Mykland (2008) develop a model-free test that identifies the precise moment a jump occurred within a day. For now, it (with some adjustments like in Boudt, Croux, and Laurent (2011) that takes into account seasonality) remains the workhorse for this type of task (see the comprehensive reviews of Theodosiou and Zikes (2012) and Mukherjee et al. (2019)).[2] Dumitru and Urga (2012) offer a Monte-Carlo comparison of nine jump tests, concluding that LM demonstrates the best performance. Consequently, we consider LM as the most widely used benchmark to which we compare our results.

Lee and Mykland (2008) suggest to use data sampled not more frequently than at 2 minutes since above that the data would likely be contaminated with market microstructure noise. Christensen, R. C. Oomen, and Mark Podolskij (2014) is the first paper to test for the presence of jumps exploiting tick-by-tick data. They conclude that jumps are in fact less common than previously thought and that the tests at lower frequencies spuriously identify bursts of volatility as jumps. We are able to confirm this finding, though working with 2 minutes frequency data and avoiding microstructure noise filtering concerns.

Machine learning algorithms in finance are gaining an overwhelming popularity. Nonetheless, the exploration of jumps with those algorithms did not receive much attention. To the best of our knowledge, the only two papers in this area are the following. Mäkinen et al. (2018) use a recurrent neural network to predict future jumps in prices. However, the training of the network takes as input the jumps already classified by the LM test on real data. They therefore do not develop a new classification tool. The work of Au Yeung et al. (2019) aims to test for jumps (defined by them as regime switching points) using machine learning. We differ from them both in terms of research question and methodology. First, we are able to separate jumps from volatility bursts, which is impossible in their framework. Second, unlike them we train our network on simulated data, instead of using (necessarily misclassified) real data. This allows for clear pattern recognition and thus better performance.

3. Machine learning vs. statistical test

We start by motivating why the existing statistical methods might fail to classify jumps as such.

The most popular nonparametric individual jump test developed in Lee and Mykland (2008) is based on the estimator of spot volatility, that is consistent in the presence of jumps. Unlike

[1]The first test using bipower variation was developed by Ole E Barndorff-Nielsen and Shephard (2006). A generalized concept (multipower variation) is studied in Ole E. Barndorff-Nielsen, Shephard, and Winkel (2006) Building on the previous findings, Corsi, Pirino, and Renò (2010) and Podolskij and Ziggel (2010) introduced new jumps tests. Other well-known jump tests include Torben G Andersen, Dobrev, and Schaumburg (2009) (using median realized volatility) and Jiang and R. C. A. Oomen (2007) (using swap variance).

[2]Torben G. Andersen, Bollerslev, and Dobrev (2007) also develop a jump test for each individual return similar to Lee and Mykland (2008) but assume constant intraday volatility.

the realized variance (RV), that measures the total variance of the process and is unable to separate the continuous and jump variations, the bipower variation (BPV) is capable of estimating solely the diffusion variance, ignoring variation of the jump part.[3] Thus, it can be used to construct a statistical test to detect jumps.

Loosely speaking, the LM test classifies a data point as a jump if the return size standardized by estimation of the instantaneous volatility [4] is too high in absolute value. Though model-free, this test requires the data generating process to satisfy some regularity conditions. The crucial assumption is that the price is represented as a jump-diffusion process

$$d\log S_t = \mu_t dt + \sigma_t dB_t + Y_t dq_t \tag{3.1}$$

with the drift μ_t and the diffusion coefficient σ_t not changing dramatically over a short period of time.[5] From now on we will use interchangeably the notions of diffusion coefficient and (spot) volatility.

This assumption is the main weak point of the test. Its violation (for example, if σ_t itself contains jumps) leads to a significant increase in the amount of spurious detection. Indeed, the test classifies a return as a jump if the absolute value of the test statistic

$$\mathcal{L}(i) = \frac{\log \frac{S(t_i)}{S(t_{i-1})}}{\hat{\sigma}(t_i)} \tag{3.2}$$

is above some threshold. If volatility is not allowed to change dramatically, a high test statistic would indeed reflect a jump. However, as soon as we allow for jumps in the volatility itself, it is not the case anymore. $\hat{\sigma}$ being computed over a moving window, if the diffusion coefficient changes dramatically, the bipower variation will take time to incorporate this change. Keeping this in mind, there are two different scenarios under which the LM test can fail. The first is a positive jump in σ_t, that may generate a high purely diffusion-driven return. Together with the bipower variation that did not have time to adjust for the new high level of volatility and thus stays relatively low, this results in a high test statistic value. LM is unable to distinguish between this scenario and a true jump in the price process, spuriously classifying both as jumps. The second scenario is a sudden volatility decrease. The LM test might not be able to find a real jump following this drop. For the same reason as above, the BPV is not able to incorporate the change immediately. So it may stay higher than the real volatility, lowering the test statistics and covering the true jump. This violation causes a significant misclassification

[3] The interested reader can find more details about volatility estimators in appendix B.

[4] Formally, the estimate of the spot volatility is a scaled version of the bipower variation and is computed as $\hat{\sigma}^2(t_i) = \frac{1}{K-2}\sum_{j=i-K+2}^{i-1} |\log S(t_j)/S(t_{j-1})||\log S(t_{j-1})/S(t_{j-2})|$ where K is the chosen window size and S_t is the spot price of the stock at time t.

[5] Strictly speaking, the assumption from Lee and Mykland (2008) is $\forall \epsilon > 0$

$$\sup_i \sup_{t_i \le u \le t_{i+1}} |\mu(u) - \mu(t_i)| = O_p(\Delta t^{\frac{1}{2}-\epsilon}),$$
$$\sup_i \sup_{t_i \le u \le t_{i+1}} |\sigma(u) - \sigma(t_i)| = O_p(\Delta t^{\frac{1}{2}-\epsilon}).$$

rate. When σ_t contains jumps, the spurious detection rate of the LM test in simulated data is above 150%.

Moreover, the assumption of no dramatic change in volatility is rejected by the real data. As shown in Tauchen and Todorov (2008), volatility of the asset price necessarily contains jumps and they happen much more frequently than just few times per year. So the core assumption of the LM test contradicts the inherent feature of the market data.

Another concern comes from consistency of the bipower variation. As stated in Ole E Barndorff-Nielsen and Shephard (2006), the BPV is a consistent estimator of the spot volatility only in absence of leverage effect. The authors acknowledge that this is an unfortunate but important restriction of their results, that confronts the stylized facts of equity data.

To overcome these significant limitations, we are the first to propose using an LSTM network for jump identification and distinguishing them from discontinuous changes in volatility. The main advantage of our approach is that we are not limited by any regularity conditions in contrast to the classical statistical tests. Instead, we train the algorithm on data generated by multiple processes and specifications of any desirable complexity, that incorporate stylized facts about price characteristics, such as leverage effect, volatility clustering, volatility jumps etc. As a result, we consistently outperform Lee and Mykland (2008) in the out-of-sample analysis when simulated data is more realistic, and perform comparably to them when their assumptions are met.

4. Network architecture and training

The long short-term memory network we choose for jump classification is a special type of the recurrent neural network, that was introduced in Hochreiter and Schmidhuber (1997). By construction, LSTM networks are well suited for remembering long-term dependencies and thus for handling time-series data. We use a standard LSTM, motivated by the main finding of Greff et al. (2016). They present the largest comparison study for different architectures of LSTM networks and show that improvements of different modifications over the plain vanilla LSTM are marginal. Our network consists of five layers: an input layer, a bidirectional LSTM layer with 200 hidden neurons, a fully connected layer, a softmax layer, and a classification output layer. An interested reader may refer to appendix A to find a brief intuition about neural networks in general and LSTM in particular, as well as a detailed layers description.

Now that the structure of the network is chosen, it has to be trained to detect jumps. Our methodology is conceptually simple. We first generate training data and since it is simulated, we know exactly when a jump occurred. Then we label each return as “jump” or “no jump” and train the network on them. Finally the network is ready to classify new time-series.

To create the training data, we simulate paths of the price process by discretizing the stochastic differential equation (SDE) governing it. An advantage of our method is that since we do not have to derive the distribution of a test statistic, there is no restriction on the data-generating process. Therefore the training data set as a whole can be composed by multiple time-series possibly generated from different processes of any complexity. Using a variety of them allows the network to concentrate only on the specific feature (jump or not) of the data point, preventing over-fitting. Another benefit of this method is that we can generate virtually an unlimited quantity of labelled data to train on, enhancing the network performance at no cost[6].

In order to incorporate many stylized facts of equity data, we have chosen the following specification for the stock price S_t and its diffusion coefficient σ_t:

$$dS_t = S_t(\mu_t dt + \sigma_t dB_{1t} + dJ_{1t}), \tag{4.1}$$
$$d\sigma_t = \alpha(\overline{\sigma} - \sigma_t)dt + \upsilon\sqrt{\sigma_t}dB_{2t} + dJ_{2t}. \tag{4.2}$$

The price follows a jump-diffusion model and we introduce a rich structure for the stochastic volatility which is governed by a mean-reverting process with jumps. The jump component J_{2t} introduces bursts of volatility, defined as a sudden change in the diffusion coefficient of the price. This specification for the stochastic volatility allows us to incorporate the conventional models with finite activity jumps and can serve (after suitable parameters adjustments) as a decent approximation for infinitely active jump processes [7]. The sources of volatility risk B_{2t} and J_{2t} can be (negatively) correlated with B_{1t} and J_{1t} to incorporate the leverage effect. In order to realistically reproduce real life patterns of securities, it is important to take all of that into account. Using this framework the network will learn how to identify jumps and disentangle them from bursts of volatility.

In order to simulate sample paths we need to select values for the parameters used in the data-generating process(es). One idea could be to take real financial data, estimate those parameters (for instance using maximum likelihood) and simulate our labelled data using them. This procedure has two drawbacks. First, there is an estimation risk. Second, financial markets are ever changing. This means that those parameters could have multiple regimes and change over time. To overcome these difficulties we train the network on a whole set of parameters. By constructing a realistic set, the network will learn what jumps look like under various environment. This gives us hope that whatever regime the market is in, the network will perform decently in the real data. In this way we also avoid re-training the network often.

Clearly, this method is not limited to using SDEs of the form of (4.1) and (4.2). The procedure would work using any data-generating process to create labelled data and then following the same steps.

5. Performance on simulated data

In this section we assess the performance of our method using out-of-sample simulated data. We start by generating new time-series, using parameters that the network was never trained on. For every number reported in the tables below, the network was tested using 2000 Monte-Carlo trials, each individual trial consisting of data generated at a 2 minutes frequency for 0.5 year.

First of all, we compare our network to the benchmark of Lee and Mykland (2008) when the data satisfies their assumption about "local" changes in spot volatility. Table 5.1 presents the

[6]Apart from the necessary computational time.
[7]For further details see Cont and Tankov (2004).

percentage of correct (% detection)[8] and spurious (% spurious)[9] classification of jumps by our network and the benchmark. We see that we perform roughly as good as the benchmark in this case. The good performance of LM is explained by the fact that all of their assumptions are met. But as pointed out by Tauchen and Todorov (2008) this is not a realistic framework to describe real world high-frequency equity data. Indeed, spot volatility should contain jumps.

Our network starts to significantly outperform in all dimensions (type I & type II errors) as soon as we introduce jumps inside the diffusion coefficient. This violates the main assumption of LM and it explains the drastically different results described in tables 5.2 and 5.3.

	Network	Lee & Mykland (benchmark)
% detection	88.32	94.15
% spurious	2.1	0.38

Table 5.1: This table compares the performance of our method versus the benchmark. “%detection” stands for the percentage of correctly identified jumps. “%spurious” stands for the percentage of spurious jumps identified. The simulated data does not contain jumps inside the volatility (intensity of volatility jump arrival is 0).

	Network	Lee & Mykland (benchmark)
% detection	92.87	88.08
% spurious	17.52	200.75

Table 5.2: This table compares the performance of our method versus the benchmark. “%detection” stands for the percentage of correctly identified jumps. “%spurious” stands for the percentage of spurious jumps identified. The simulated data contains bursts of volatility (intensity of volatility jump arrival is 65 per year).

	Network	Lee & Mykland (benchmark)
% detection	88.60	82.40
% spurious	25.96	147.61

Table 5.3: This table compares the performance of our method versus the benchmark. “%detection” stands for the percentage of correctly identified jumps. “%spurious” stands for the percentage of spurious jumps identified. The simulated data contains bursts of volatility (intensity of volatility jump arrival is 6000 per year).

Furthermore, these results can be visualized in figure 5.1 which grasps the essence of this paper. We plot one time-series of simulated returns (standardized by the bipower variation) that are classified both by our network and the LM test. For readability, circles correspond to returns coming purely from the diffusion component and stars are returns truly containing a

[8] $\% \text{ detection} = \frac{\text{\# of correct jump detection}}{\text{total \# of true jumps}}$.

[9] $\% \text{ spurious} = \frac{\text{\# of no jumps, classified as jumps}}{\text{total \# of true jumps}}$.

jump. We describe the data points by discussing four categories: (i) both methods agree and are correct, (ii) both methods agree and are mistaken, (iii) LM outperforms the network, and (iv) the network outperforms the LM test.

(i) The major part of the returns are small (in absolute value) and come from the diffusion component. Those are correctly classified as “no jump” by both methods (blue circles). Similarly, the isolated large (in absolute value) returns are correctly classified as a jump by both tests (green stars).

(ii) There are a few true jumps that both methods fail to recognize as such (black stars). The reason is that those jumps are small in magnitude comparatively to the Brownian term. In the same spirit, both tests wrongly identify the data points as being a jump (green circles) when the diffusion component realization is unusually high.

The most interesting part of the analysis is where the methods disagree.

(iii) There is no true jump correctly classified only by the LM test. This is in line with our results from tables 5.2 and 5.3 that display a higher detection rate for our neural network. However, our method is not perfect: rarely, it is the only one to mistakenly categorize a point as a jump (yellow circles).

(iv) Since this plot depicts the returns standardized by the BPV, the LM test can be imagined as two horizontal lines symmetrically placed around zero on this plane. Every return between them is classified as no jump, while any outside of this region is identified as a jump. As we see, such approach results in significant amount of misidentification. This happens because the assumption of local changes in volatility (fundamental for the LM test) is violated now. There are two cases where our method outperforms the benchmark. First when a positive jump inside the spot volatility σ_t occurs. In this case, larger diffusion increments coupled with the bipower variation that does not adjust fast enough result in a test statistic that is too high and spurious jump identification by LM (red circles). This category is overwhelmingly big, in line with results presented in tables 5.2 and 5.3. The LM test spuriously identifies as a jump more points than the entire amount of true jumps in the sample. The second scenario is when σ_t experiences a fast drop. The BPV might stay too high, lowering the test statistic and masking the true jump (yellow stars).

To ensure the robustness of our method, we test the algorithm on a different data-generating model. Before, we conducted the out-of-sample analysis by solely changing the parameters. From now on we also modify the structure of the SDEs. The price S_t contains two Brownian terms and a jump component. On top of that, all the randomness in the diffusion coefficients is created by pure jump processes (as suggested by Tauchen and Todorov (2008)). The model is described by the following system of SDEs

$$dS_t = S_t(\mu_t dt + \sigma_{1t} dB_{1t} + \sigma_{2t} dB_{2t} + dJ_{1t}), \tag{5.1}$$

$$d\sigma_{1t} = \alpha_1(\overline{\sigma}_1 - \sigma_{1t})dt + dJ_{2t}, \tag{5.2}$$

$$d\sigma_{2t} = \alpha_2(\overline{\sigma}_2 - \sigma_{2t})dt + dJ_{3t}. \tag{5.3}$$

The results stemming from this specification are displayed in table 5.4. As before our method outperforms the benchmark.

	Network	Lee & Mykland (benchmark)
% detection	86.05	80.04
% spurious	14.96	59.33

Table 5.4: This table compares the performance of our method versus the benchmark. "%detection" stands for the percentage of correctly identified jumps. "%spurious" stands for the percentage of spurious jumps identified. The simulated data is governed by the model (5.1)-(5.3).

As a final robustness check, we simulate the price by using (5.1) but the paths for σ_{1t} and σ_{2t} are obtained from market data. This is a way to make our simulations closer to reality while being able to assess the performance of the test. For volatility estimation we decide not to use the bipower variation. The reason is that eliminating the impact of jumps requires the time window K to be large enough, over-smoothing volatility. Instead, we first take a time-series of real returns from which we remove the jumps detected by our network. Then the spot volatility is estimated using the realized volatility computed over a shorter window. Applying this procedure to two different stocks gives us two paths that are used as $\left(\sigma_{1t}\right)_{t=0}^{T}$ and $\left(\sigma_{2t}\right)_{t=0}^{T}$. This implies that when performing the 2000 trials, the paths of the diffusion coefficients stay fixed. Also, this technique is subject to errors both in jumps detection and in volatility estimation. However, we believe that this approach provides a meaningful complementary robustness check. The results are presented in table 5.5 and are in line with all previous findings.

	Network	Lee & Mykland (benchmark)
% detection	88.63	81.15
% spurious	3.92	64.43

Table 5.5: This table compares the performance of our method versus the benchmark. "%detection" stands for the percentage of correctly identified jumps. "%spurious" stands for the percentage of spurious jumps identified. The stock price comes from the process (5.1) but the paths for σ_{1t} and σ_{2t} are estimated from real data.

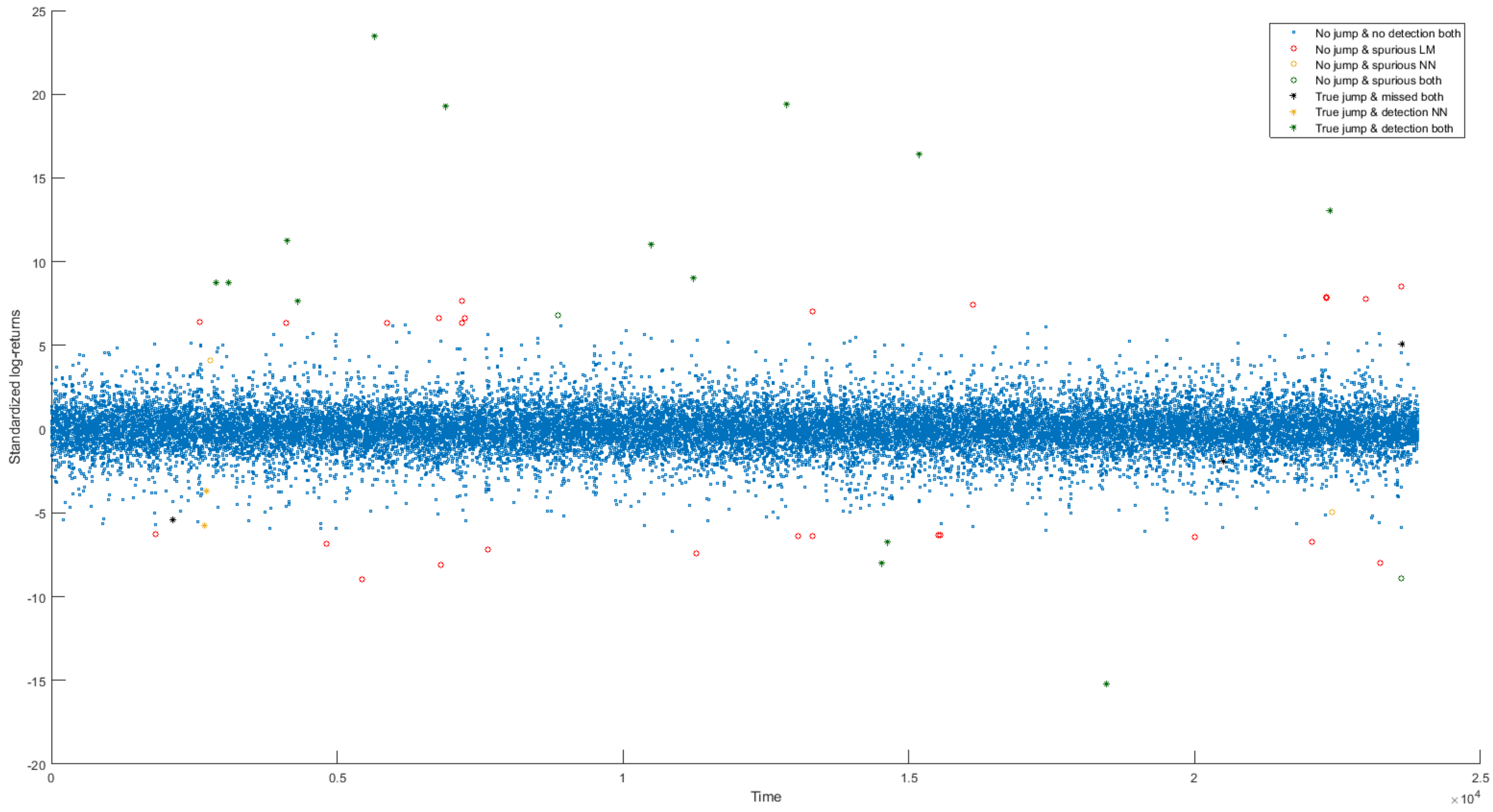

Figure 5.1: Simulated data. Returns are standardized by the bipower variation and sampled at 2 min frequency for 0.5 year. The spot volatility itself jumps frequently.

6. Application to real data

In this section we perform jump classification on real data sampled at 2 minutes. Here clearly we cannot exactly assess the performance of our test comparatively to the benchmark. However we provide evidence supporting our method through an event study and volatility forecasting.

As a first step we propose a visual inspection of classified returns of one US company and compare with the LM test. Figure 6.1 displays the returns of American International Group (AIG) standardized by the BPV. Similarly to simulated data, the core of the series that is constituted by a multitude of returns close to zero is classified as coming from the continuous Brownian by both methods (in blue). The isolated large (in absolute value) returns are also classified by both our network and LM as jumps (in green).

In this figure, some jumps identified only by LM (in red) cluster in time. The fact that a jump in the price is followed by other jumps over a very short period of time (usually within few minutes) goes against the very definition of what constitutes a jump in equity data. Finite activity jumps should be rare events and the probability of multiple occurring over a short time span is negligible. For this reason we believe that many of those jumps classified only by LM are in fact coming from the Brownian component during a burst of volatility, explaining their size. Also, these very points (in red) happen to locate close to a group of returns similar in magnitude, but identified as no jump. This suggests that the whole cluster of returns represents a volatility burst. It is unlikely to have a jump-generated return, that has the same magnitude as its diffusion-generated neighbors, so probably it is a misclassification. This finding matches the results of Christensen, R. C. Oomen, and Mark Podolskij (2014). The other dimension in which we differ are the jumps classified only by our network while they are considered as continuous returns by LM (in yellow). They happen after a sharp drop of spot volatility, hence the LM test appears to miss those jumps due to the reasons we explained in the previous section.

6.1. Event study

In a second step, we analyze the data points where our network and the LM test disagree and convey an event study to support the verdict of one of the two methods. This subsection presents an example.

On September, 19, 2008, the AIG stock experienced a log-return of 12.18% within 2 minutes. This event can be seen on figure 6.1 in the center of the orange circle. In order to have a more granular view of this event, figure 6.2 displays the stock price of AIG for the first half of this day. The price changed from \$2.78 at 10h17am to \$3.14 at 10h19am. Our network classifies this return as a jump, while the LM test disagrees with us and attributes this change to the continuous term. Why do we think that this return was indeed caused by the jump?

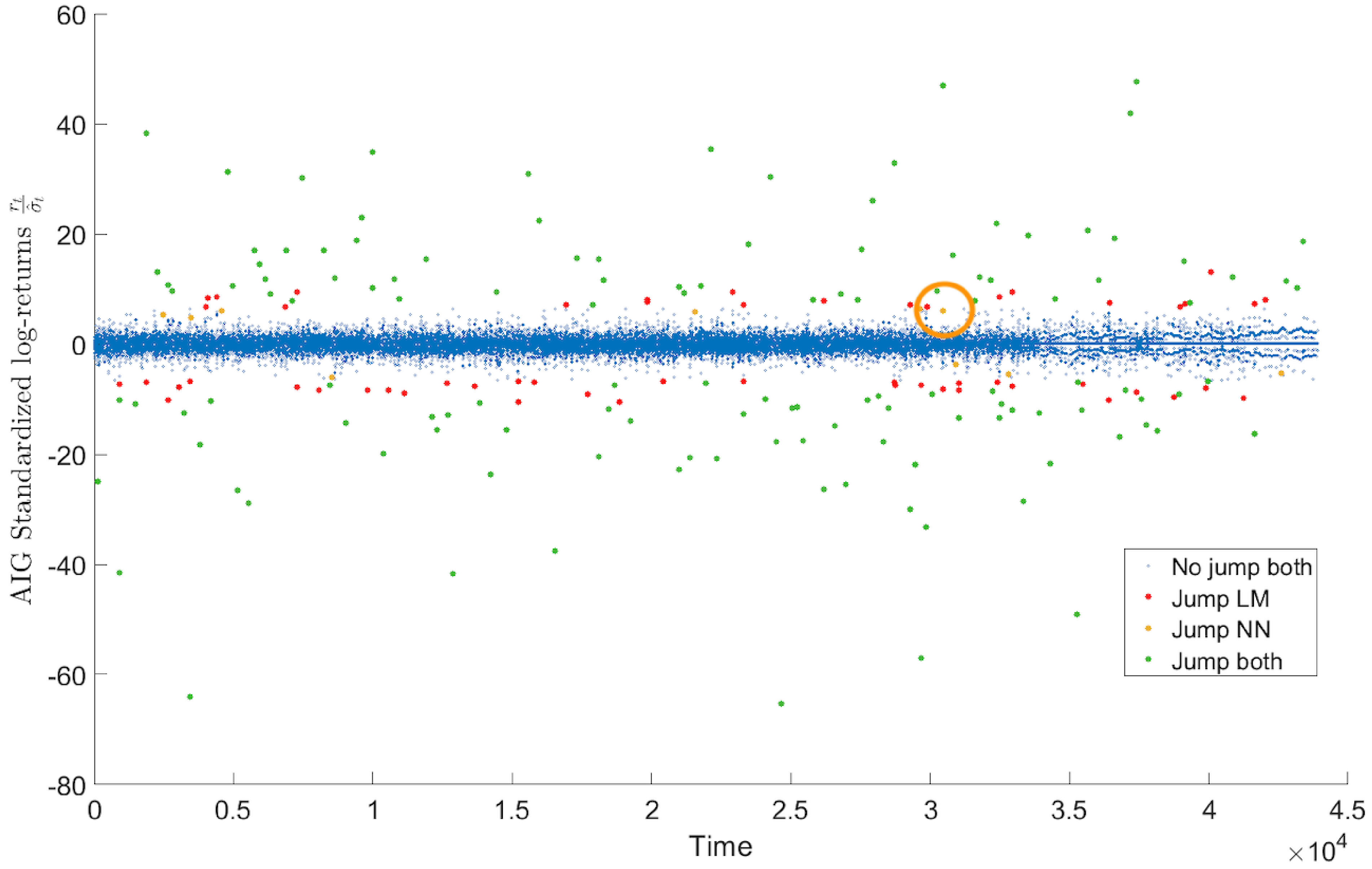

Figure 6.1: Returns of American International Group (AIG) standardized by the bipower variation at 2 min frequency. Regarding the legend: “No jump both” means that none of the methods detected a jump, “Jump LM” means that a jump was detected only by the test of LM, “Jump NN” means that a jump was detected only by our network, and finally “Jump both” means that both LM and the network agree and detect a jump.

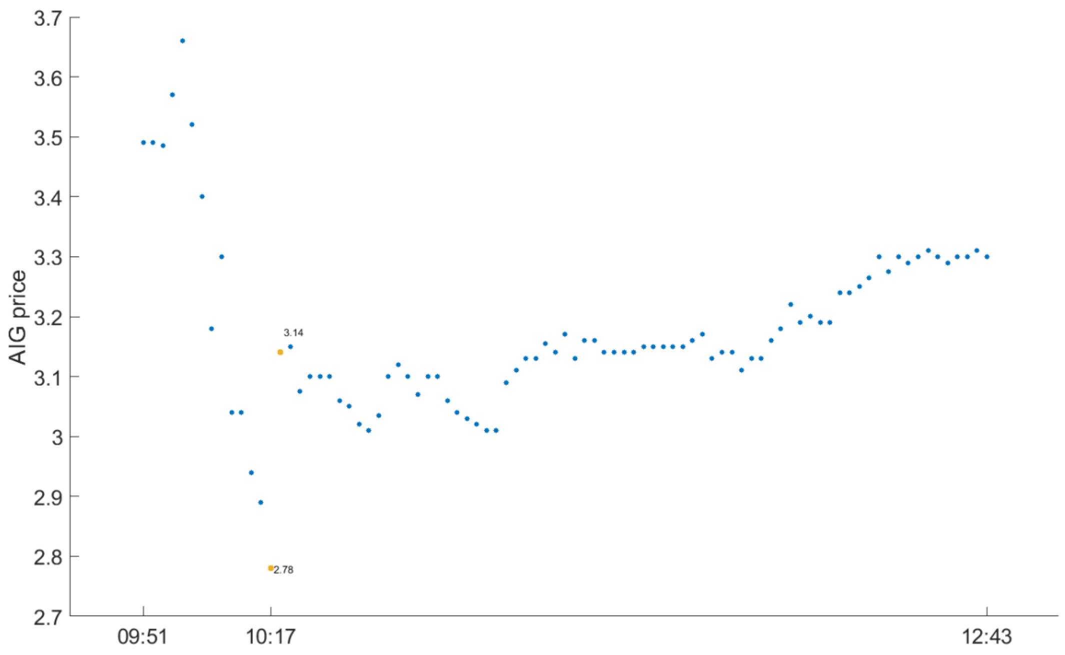

Figure 6.2: AIG stock price in the morning of September 19, 2008 sampled at a 2 minutes frequency.

The week of September 15, 2008 - September 19, 2008 was in the heart of the financial crisis. On Monday Lehman Brothers filed for bankruptcy, constituting the largest one in US history. That was a period of market instability, featuring high volatility. But on Friday September 19, at 10:05, the Treasury Secretary Henry Paulson announced a number of actions that the state would take to stabilize the financial system. In particular, the money market funds guarantee program was announced. This was perceived as positive news, and pushed markets up. The length on the Secretary's talk was approximately 9 minutes, and it ended at 10h14am. Then, three minutes later, we observe a sudden and big upward change in the price of AIG. Most interestingly, after this big change a calm period begins. Volatility decreases significantly and price balances on the relatively stable level. This refutes the hypothesis of the abovementioned return being generated by the diffusion part and supports the jump nature of this return. This is an example of our previous discussion about the failure of the LM test. In this case we suppose that a sudden drop in volatility violates the LM assumption and thus does not allow their test to detect the actual jump, causing a misclassification.

6.2. Volatility forecasting

Finally, we propose an exercise of volatility prediction. The intuition for this exercise is as follows. It is known that the HAR-RV (heterogeneous autoregressive model of realized volatility) of Corsi (2009) does a decent job in forecasting the variance. Torben G. Andersen, Bollerslev, and Diebold (2007) have developed a HAR-RV-CJ model that disentangles RV into a continuous (C) and jump (J) components and showed that it may improve the prediction. Thus a more accurate jump detection method allows for a better modelling of these two components and in turn would be reflected in an increased performance. This allows us to benchmark our method to others.

The first step is to construct an estimate of the daily, weekly and monthly total volatility using high-frequency data. For this purpose we use the realized variance and denote the daily component by RV_t^d (see appendix B for the formula). The weekly volatility is simply an average of the daily quantities over five days

$$\mathrm{RV}_t^w = \frac{1}{5}\left(\mathrm{RV}_t^d + \mathrm{RV}_{t-1d}^d + \ldots + \mathrm{RV}_{t-4d}^d\right). \tag{6.1}$$

The monthly volatility RV_t^m is constructed in a similar fashion using twenty two days. The plain vanilla regression (in spirit of Corsi (2009)) to forecast one day/week/month ahead is

$$\mathrm{RV}_{t+f}^f = \alpha_f + \beta_f^d \mathrm{RV}_t^d + \beta_f^w \mathrm{RV}_t^w + \beta_f^m \mathrm{RV}_t^m + \mathrm{error}_{t+f} \tag{$\mathcal{PV}$}$$

where $f \in \{d, w, m\}$.

In order to split RV into a continuous variance C and a jump variance J we compare three methods. The first one simply uses the bipower variation to estimate C and then computes the jump volatility as $J_t = RV_t - C_t$. We call this method $\mathcal{BPV}$. A second approach ($\mathcal{LM}$) is to use the jumps detected by the LM test and remove them from the time-series of stock returns. Assuming the method works perfectly, we obtain a series of returns generated by a pure diffusion process. Then we can compute the realized variance of this newly created series of returns in order to obtain C. A third method ($\mathcal{NN}$) is to perform the same steps but using our neural network instead. Of course the last two methods produce different results since LM and NN disagree about certain jumps.

For each of the three approaches we run the following regression

$$\begin{aligned}\mathrm{RV}_{t+f}^f = \alpha_{f,i} + \beta_{f,i}^{d,C} \mathrm{C}_{t,i}^d + \beta_{f,i}^{w,C} \mathrm{C}_{t,i}^w + \beta_{f,i}^{m,C} \mathrm{C}_{t,i}^m \\ \beta_{f,i}^{d,J} \mathrm{J}_{t,i}^d + \beta_{f,i}^{w,J} \mathrm{J}_{t,i}^w + \beta_{f,i}^{m,J} \mathrm{J}_{t,i}^m + \mathrm{error}_{t+f},\end{aligned} \tag{6.2}$$

$i \in \{\mathcal{BPV}, \mathcal{LM}, \mathcal{NN}\}$. In order to prevent over-fitting we set the coefficients in front of the jump variance to zero.[10]

The comparison of the out-of-sample forecasting performance of the different methods is displayed in table 6.1. We conduct the analysis on the Dow Jones constituents (excluding the financial firms) and the index-tracking ETF (DIA) from 2006 to 2008 included, observed at a 2 minutes frequency. The reported results are the cross-sectional average of the forecasting performance for each method. We can see that at all horizons the neural network reaches the lowest root mean-square error (RMSE) and the highest R^2, beating all the other methods.

[10]Adding the jump variance terms J^f, $f \in \{d, w, m\}$ improves the in-sample performance but in general deteriorates the out-of-sample results. Jump coefficients being typically insignificant we have decided to remove them.

The differences in performance between the methods might seem small at first glance but we have to remember that jumps are actually rare events and account only for a small portion of the returns. Moreover, the two methods $\mathcal{LM}$ and $\mathcal{NN}$ only disagree on a strict subset of the already small number of jumps. It is therefore normal to obtain performance metrics that are relatively close to one another. Yet, the higher performance of the network hints at the fact that even on real data it is better able to detect jumps and extract a more precise signal of future total volatility.

	Daily		Weekly		Monthly	
	R^2	RMSE	R^2	RMSE	R^2	RMSE
$\mathcal{PV}$	62,41	1,12	69,83	0,88	64,56	0,89
$\mathcal{BPV}$	64,69	1,09	71,75	0,85	65,78	0,88
$\mathcal{LM}$	64,98	1,08	72,42	0,84	65,72	0,88
$\mathcal{NN}$	65,38	1,07	72,53	0,83	65,99	0,87

Table 6.1: Comparison of the out-of-sample performance for one day ahead daily, one week ahead weekly and one month ahead monthly volatility forecast by four methods on the Dow Jones stocks (financial firms excluded) from 2006 to 2008. All the regressions are reestimated daily. R^2 and root mean square error (RMSE) are used as performance metrics.

7. Conclusion

We present a new method that uses an LSTM neural network to detect jumps nonparametrically in high-frequency data. The network is able to distinguish between jumps and bursts of volatility without using ultra high-frequency data, avoiding microstructure noise. The network is trained on labelled data generated at virtually zero cost using Monte-Carlo. Using out-of-sample simulated data, our approach significantly outperforms the benchmark of Lee and Mykland (2008) in realistic market conditions (i.e. in presence of jumps inside the spot volatility of the price process). On real data, we provide supportive evidence of the higher accuracy in jump detection of the neural network through an event study and improved volatility forecasts.

References

Aït-Sahalia, Yacine (2004). “Disentangling diffusion from jumps”. *Journal of Financial Economics* 74.3, pp. 487–528.

Andersen, Torben G., Tim Bollerslev, and Francis X. Diebold (2007). “Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility”. *Review of Economics and Statistics* 89.4, pp. 701–720.

Andersen, Torben G., Tim Bollerslev, and Dobrislav Dobrev (2007). “No-arbitrage semi-martingale restrictions for continuous-time volatility models subject to leverage effects, jumps and i.i.d. noise: Theory and testable distributional implications”. *Journal of Econometrics* 138.1, pp. 125–180.

Andersen, Torben G, Dobrislav Dobrev, and Ernst Schaumburg (2009). “Jump-robust volatility estimation using nearest neighbor truncation”.

Au Yeung, Jay F.K. et al. (2019). “Jump detection in financial time series using machine learning algorithms”. *Soft Computing*.

Barndorff-Nielsen, Ole E and Neil Shephard (2004). “Power and Bipower Variation with Stochastic Volatility and Jumps”. *Journal of Financial Econometrics* 2.1, pp. 1–37.

– (2006). “Econometrics of testing for jumps in financial economics using bipower variation”. *Journal of Financial Econometrics* 4.1, pp. 1–30.

Barndorff-Nielsen, Ole E., Neil Shephard, and Matthias Winkel (2006). “Limit theorems for multipower variation in the presence of jumps”. *Stochastic Processes and their Applications* 116.5, pp. 796–806.

Boudt, Kris, Christophe Croux, and Sébastien Laurent (2011). “Robust estimation of intraweek periodicity in volatility and jump detection”. *Journal of Empirical Finance* 18.2, pp. 353–367.

Christensen, Kim, Roel C.A. Oomen, and Mark Podolskij (2014). “Fact or friction: Jumps at ultra high frequency”. *Journal of Financial Economics* 114.3, pp. 576–599.

Cont, Rama and Peter Tankov (2004). *Financial Modelling With Jump Processes*. Chapman & Hall/CRC Financial Mathematics Series.

Corsi, Fulvio (2009). “A simple approximate long-memory model of realized volatility”. *Journal of Financial Econometrics* 7.2, pp. 174–196.

Corsi, Fulvio, Davide Pirino, and Roberto Renò (2010). “Threshold bipower variation and the impact of jumps on volatility forecasting”. *Journal of Econometrics* 159.2, pp. 276–288.

Dumitru, Ana Maria and Giovanni Urga (2012). “Identifying jumps in financial assets: A comparison between nonparametric jump tests”. *Journal of Business and Economic Statistics* 30.2, pp. 242–255.

Greff, Klaus et al. (2016). “LSTM: A Search Space Odyssey”. *Transactions on Neural Networks and Learning Systems*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. *Neural Computation* 9(8), pp. 1735–1780.

Jiang, George J and Roel C A Oomen (2007). “Testing for Jumps When Asset Prices are Observed with Noise - A “Swap Variance” Approach”.

Lee, Suzanne S. and Per A. Mykland (2008). “Jumps in financial markets: A new nonparametric test and jump dynamics”. *Review of Financial Studies* 21.6, pp. 2535–2563.

Mäkinen, Milla et al. (2018). “Forecasting of Jump Arrivals in Stock Prices: New Attention-based Network Architecture using Limit Order Book Data”.

Mukherjee, Arpita et al. (2019). “Financial econometrics and big data: A survey of volatility estimators and tests for the presence of jumps and co-jumps”.

Podolskij, M and D. Ziggel (2010). “New tests for jumps in semimartingale models”. *Statistical Inference for Stochastic Processes* 13.1, pp. 15–41.

Tauchen, George and Viktor Todorov (2008). “Volatility Jumps”.

Theodosiou, Marina G and Filip Zikes (2012). “A Comprehensive Comparison of Nonparametric Tests for Jumps in Asset Prices”. *SSRN Electronic Journal* 44.0.

APPENDIX A

This section provides a brief intuition about neural networks and an overview of the structure of the specific network we use.

Artificial Neural Networks (ANN or simply NN) are for now one of the most powerful and widely used tool to tackle complex machine learning problems. In essence, every NN is a sequence of non-linear data transformations. A network consists of units called *neurons*, which are hierarchically organized into *layers*. Each neuron in the network is associated with its own weighting vector W and bias b, which are the parameters that will be updated during the learning process. The neuron performs an affine data transformation (multiplying the input by its weighting vector and adding the bias), and then applies a predetermined activation function[11] to the result. All neurons of layer l take as input the previous $l-1$ layer's output, and in turn pass their own output as an input for the following layer $l+1$. Neurons within one layer use the same activation function and operate independently from each other. Formally, the output of neuron number k in layer l is

$$a_k^{[l]} = \phi(W_k^{[l]'} x + b_k^{[l]}) \tag{A.1}$$

where ϕ is the activation function that is applied elementwise and x is the output of all neurons of the previous layer. Passing the data through the network and obtaining the output is called *forward propagation.*

The goal of network learning is to find parameter values that will result in a minimal error between the network output and the desired output (in our case this is the labelled output). It is done iteratively. First, the input data is fed to the network and the output is obtained. Then an error function is computed, that shows how far is the network result from the target. The chosen activation functions being piecewise differentiable, a gradient descend method is used to update the parameters. This process is called *backpropagation.* Repeating forward and backpropagation allows to adjust parameters in the way that results in increasing network performance (i.e. lower error function).

An example of a simple network is the logit regression. It has two layers:[12] The first one is the input layer, with the amount of neurons being equal to the number of regressors. The second layer is the output layer with one single neuron, that has the *sigmoid* activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. Parameters of this neuron are just the regression coefficients. This network has no hidden layers (that is, layers other than input and output). Networks that have one or no hidden layers are called shallow, while networks with multiple hidden layers are called *deep.* Figure A.1 provides a visualization of a deep neural network.

[11] Theoretically, any function can be used as an activation. However, a piecewise differentiable function is usually preferred in practice.

[12] Due to conventions, it is called a one layer network since the input layer is usually not counted.

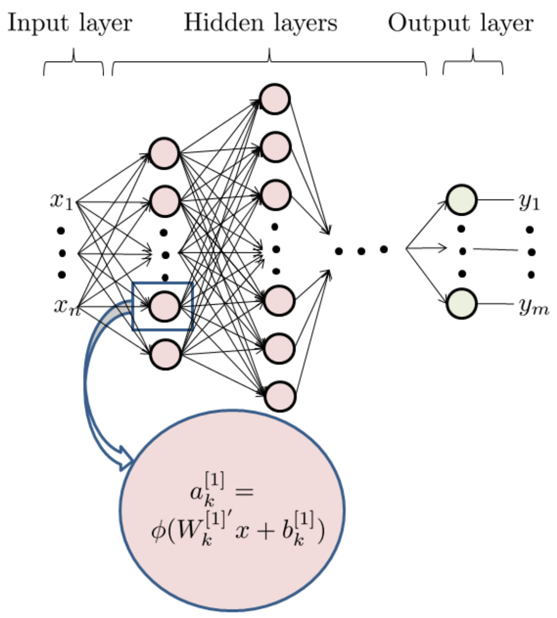

Figure A.1: Deep Neural Network

The drawback of using a general network as described above is that it is incapable of learning temporal dependencies from time-series data. To address this task a *recurrent neural network* (RNN) could be used. The recursive layer of such network has an analogue of memory, called *hidden state*. It carries the information from the previous time step and is used as additional input for the recursive layer. Formally, it processes time t observation according to

$$h_t = \phi(Wx_t + Uh_{t-1} + b) \tag{A.2}$$

where x_t is the input, h_{t-1} is the hidden state from the previous time step and W and U are the corresponding weighting matrix for the input and hidden state respectively. h_t is the updated hidden state, that is kept to treat the next observation $t+1$ and also it serves as the output. Two major issues arise for such plain vanilla RNN. First, the memory is short-lived and not elective. There is no way to keep for a long time the important information and quickly forget the irrelevant one. The second issue, technical in nature, is the exploding/vanishing gradient. [13]

Long short-term memory network (LSTM) is a specific type of RNN, that mitigates both of these problems. An LSTM block has two states. One state corresponds to the working memory and is analogous to the RNN hidden state h_t. It is also the output of a block to the following network layer. The second one is the long-term memory mechanism, called the cell state and

[13] Intuitively, each RNN could be unfolded into a non-recurrent network of the same length than the data series. During backpropagation, the derivative of the error function with respect to weights should be computed for every node. Due to the chain rule, it results in iterative multiplication and thus the derivative may become unstable.

denoted by c_t. The block also has three gates (non-linear input transformation), that regulate the information flow inside:

(r) The forget/remember gate coordinates which information from the long-term memory should be kept and which one should be discarded.

(s) The input gate (or sometimes called save gate) decides which information from the input should be saved in the long-term memory.

(f) The output gate (sometimes called focus) controls the updates of the hidden state.

Each gate is in essence just a shallow neural network itself. The parameters associated with the **r**emember, **s**ave, **f**ocus gates respectively are given by the triplet (W_i, U_i, b_i) where $i \in \{r, s, f\}$.

To better grasp the intuition, let us walk along the transformation of the input x_t within one LSTM block. From the previous time step the cell state c_{t-1} and the hidden state h_{t-1} are passed.

1. This first step is dedicated to learning which information of the existing long-term memory will be kept or forgotten. To do so, the remember gate (r) uses x_t and the working memory h_{t-1} to obtain the "remember" vector r_t that is computed as

   $$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r). \tag{A.3}$$

   Here σ defines the sigmoid activation function, that ensures values of the remember vector are between 0 (fully forget) and 1 (fully remember). r_t will later be elementwise multiplied by c_{t-1} to keep only the relevant information.

2. Now we decide which information should potentially be added to the long-term memory. The candidate is formed as

   $$c'_t = \tanh(W_l x_t + U_l h_{t-1} + b_l) \tag{A.4}$$

   where (W_l, U_l, b_l) are the parameters of the **l**ong-term memory candidate formation.

3. Before it enters the long-term memory, the save gate (s) decides which part of this candidate is worth saving by computing the following quantity

   $$s_t = \sigma(W_s x_t + U_s h_{t-1} + b_s). \tag{A.5}$$

   As before, the activation function here is sigmoid, that ensures values between 0 and 1 and thus by elementwise multiplication allows to regulate the information flow.

4. We are ready to update the long-term memory by performing the following operation

   $$c_t = c_{t-1} \otimes r_t + s_t \otimes c'_t \tag{A.6}$$

   where $\otimes$ denotes an elementwise multiplication. The updated value of the long-term memory c_t consists of the information remembered from the past (first term) and the newly added component (second term).

5. Finally, we can update the hidden state h_t. The focus gate (f) allows to concentrate on the relevant information from the long-term memory. This is done as follow

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{A.7}$$
$$h_t = f_t \otimes \tanh(c_t). \tag{A.8}$$

The figure A.2 allows to represent those transformations visually.

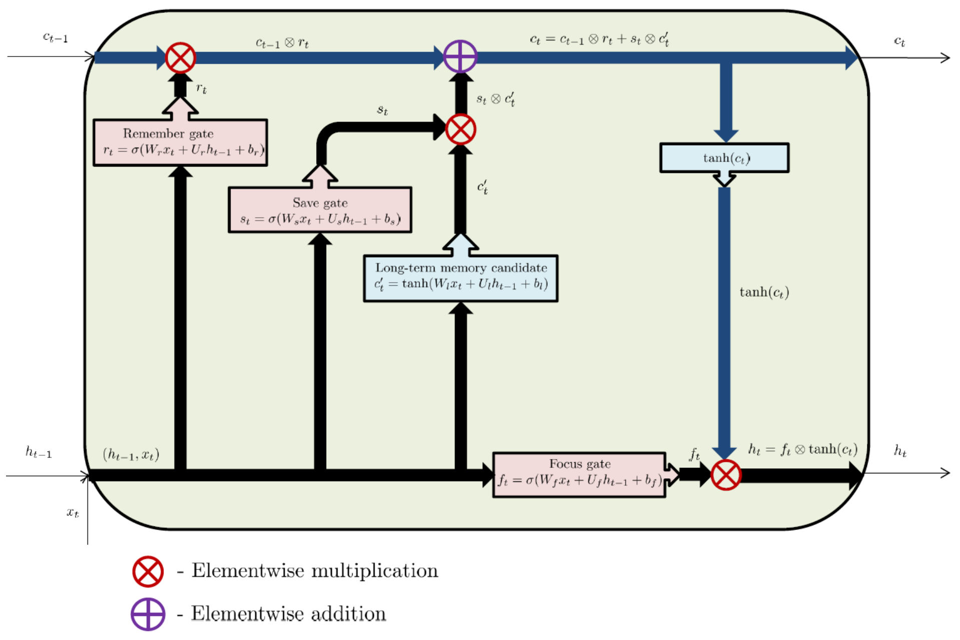
c_{t-1}
$c_{t-1} \otimes r_t$
$c_t = c_{t-1} \otimes r_t + s_t \otimes c'_t$
c_t
r_t
s_t
$s_t \otimes c'_t$
Remember gate
$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$
Save gate
$s_t = \sigma(W_s x_t + U_s h_{t-1} + b_s)$
c'_t
Long-term memory candidate
$c'_t = \tanh(W_l x_t + U_l h_{t-1} + b_l)$
$\tanh(c_t)$
$\tanh(c_t)$
h_{t-1}
(h_{t-1}, x_t)
Focus gate
$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$
f_t
$h_t = f_t \otimes \tanh(c_t)$
h_t
x_t
- Elementwise multiplication
- Elementwise addition

Figure A.2: Transformation of the input inside the LSTM unit

Now that we have a brief intuition about how the LSTM block works, this section concludes with a brief description of our neural network architecture. The network consists of the following layers:

1. Sequence input layer, that inputs the time-series into the network.

2. Bidirectional LSTM layer with 200 neurons. This layer is learning long-term dependencies from the complete sequence

   - An LSTM layer, as discussed before, allows the network to keep track of the valuable information, that was encountered long time ago, forgetting the more recent but unimportant.
   - A bidirectional layer duplicates the LSTM layer, creating two such layers one after the other. The first receives the actual time-series as an input, while the second one receives the reversed copy of the data. This allows the network to use the whole

dataset to classify points, including information that comes from the moments after.

3. Fully connected layer with two neurons, both of them being connected to all the neurons from the previous layer. Each neuron multiplies the input by the weight vector and adds the bias. This layer assembles all the features learned by the bidirectional LSTM layer to classify points into "jump"/"no jump" categories.

4. Softmax layer with two neurons, that applies softmax function [14] to the inputs, computing the probabilities of the point belonging to one of the two categories. If the outputs of the previous layer's two neurons are s_1 and s_2, the neurons of this layer will compute $p_i = \frac{e^{s_i}}{\sum_{j=1}^{2} e^{s_j}}$ for $i = 1,2$.

5. Classification output layer, that computes the cross-entropy[15] for the classification problem for multiple non-intersecting classes in order to construct the error function (that will be minimized).

Schematic representation of our network can be found on the figure A.3.

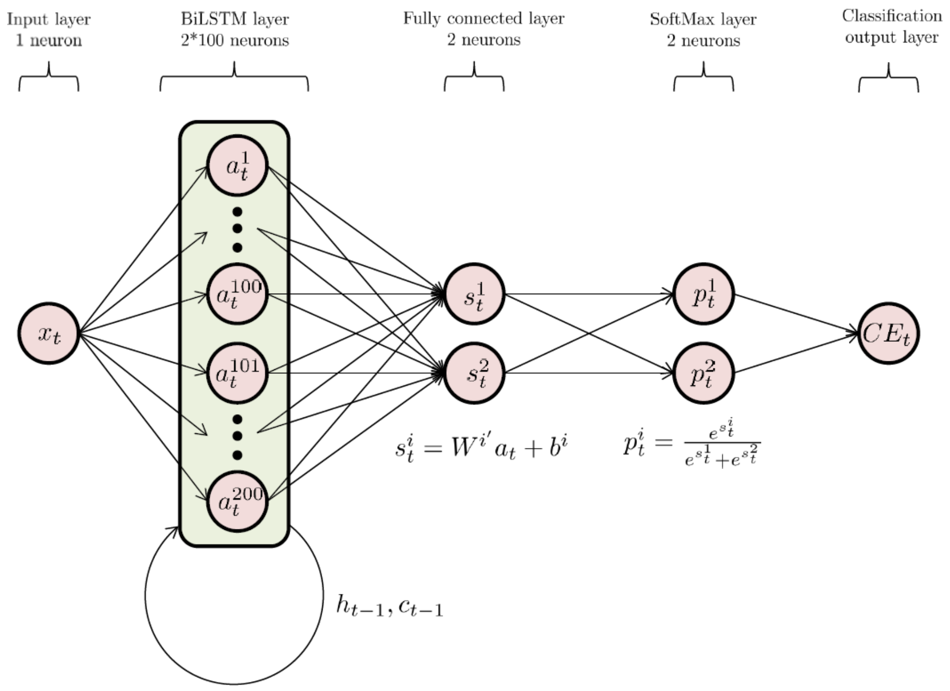

Figure A.3: Network used in this paper

[14]The softmax function (also called normalized exponential) takes as input a real vector and transforms it into a probability distribution. Formally, for a vector $v \in \mathbb{R}^n$, the softmax function $g : \mathbb{R}^n \to \mathbb{R}^n$ is defined as $g(v)_i \triangleq \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}}$ for $i = 1, ..., n$.

[15]The cross-entropy of two probability distributions $\mathbb{P}$ and $\mathbb{P}^*$ is defined as $H(\mathbb{P}, \mathbb{P}^*) = E^{\mathbb{P}}\left[-\log \mathbb{P}^*\right]$.

APPENDIX B

If the price process is represented by the jump-diffusion model (3.1), then the realized variation is defined as

$$\mathrm{RV}^2_{t+1}(\Delta) \triangleq \sum_{j=1}^{1/\Delta} r^2_{\Delta,t+j\cdot\Delta} \rightarrow \int_t^{t+1} \sigma^2_s ds + \sum_{t<s\leq t+1} Y^2_s \tag{B.1}$$

where Δ corresponds to the chosen frequency and $r_{\Delta,t+j\cdot\Delta}$ is the j^{th} log-return within day t. The realized variance converges (in probability) to the total variance composed by two terms. The first one being the continuous variance (generated by the diffusion term of the stochastic process) while the second is the jump variance.

In order to estimate only the integrated volatility, Ole E Barndorff-Nielsen and Shephard (2004) introduced the realized bipower variation which is defined as

$$\mathrm{BPV}^2_{t+1}(\Delta) \triangleq (\pi/2)^{-2} \sum_{j=2}^{1/\Delta} |r_{\Delta,t+j\cdot\Delta}||r_{\Delta,t+(j-1)\cdot\Delta}| \rightarrow \int_t^{t+1} \sigma^2_s ds. \tag{B.2}$$

BPV is a consistent estimator of integrated volatility in the presence of jumps. It is therefore useful to create a jump test.