EPFL

Thèse n° 8498

# Essays in Empirical Asset Pricing

Présentée le 26 avril 2022

Collège du management de la technologie
Chaire du Prof. ordinaire Collin-Dufresne
Programme doctoral en finance

pour l'obtention du grade de Docteur ès Sciences

par

**Alexis Arilès MARCHAL**

Acceptée sur proposition du jury

Prof. E. Morellec, président du jury
Prof. P. Collin Dufresne, Prof. J. Hugonnier, directeurs de thèse
Prof. S. Scheidegger, rapporteur
Prof. D. Challet, rapporteur
Prof. D. Filipovic, rapporteur

École
polytechnique
fédérale
de Lausanne

2022

# Acknowledgements

I would like to thank my advisors Pierre Collin-Dufresne and Julien Hugonnier for their guidance. I feel lucky that they always supported the research directions I was interested in. Having freedom in my research process helped me select the most interesting problems and kept me motivated. I also want to thank Damir Filipovic for his valuable discussions during the course of my PhD. I am grateful to the other members of my thesis committee: Damien Challet, Erwan Morellec, and Simon Scheidegger. I also appreciate the administrative support of Lorraine and Sophie who always welcome me with a smile. I thank the Swiss Finance Institute for its support.

I also want to express my gratitude towards Oksana for the multitude of engaging discussions, some of which contributed to parts of this thesis (and for being a great co-author). I appreciate the numerous conversations I had over the years with Benoit as well as my officemates: Java, Oliver, Philippe, Thomas, and Yuan. I will always remember all the skiing, snowshoeing, and hiking days that I shared with my colleagues in the Swiss alps.

A special thank goes to my friend Jacob who consistently supported me, especially during the (many) lockdowns of these particular times. Finally, I would like to thank my parents for their continuous support and always believing in me. They gave me the intellectual curiosity that motivated me in pursuing this work.

*Lausanne, December 22, 2021* A. M.

# Abstract

This thesis consists of three applications of machine learning techniques to empirical asset pricing.

In the first part, which is co-authored work with Oksana Bashchenko, 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.

In the second part, which is co-authored work with Oksana Bashchenko, we develop a methodology for detecting asset bubbles using a neural network. We rely on the theory of local martingales in continuous-time and use a deep network to estimate the diffusion coefficient of the price process more accurately than the current estimator, obtaining an improved detection of bubbles. We show the outperformance of our algorithm over the existing statistical method in a laboratory created with simulated data. We then apply the network classification to real data and build a zero net exposure trading strategy that exploits the risky arbitrage emanating from the presence of bubbles in the US equity market from 2006 to 2008. The profitability of the strategy provides an estimation of the economical magnitude of bubbles as well as support for the theoretical assumptions relied on.

In the third part, I propose a new tool to characterize the resolution of uncertainty around FOMC press conferences. It relies on the construction of a measure capturing the level of discussion complexity between the Fed Chair and reporters during the Q&A sessions. I show that complex discussions are associated with higher equity returns and a drop in realized volatility. The method creates an attention score by quantifying how much the Chair needs to rely on reading internal documents to be able to answer a question. This is accomplished by building a novel dataset of video images of the press conferences and leveraging recent deep learning algorithms from computer vision. This alternative data provides new information on nonverbal communication that cannot be extracted from the widely analyzed FOMC transcripts. This chapter can be seen as a proof of concept that certain videos contain valuable information for the study of financial markets.

**Keywords:** Asset pricing, machine learning, alternative data, volatility modelling, LSTM

# Résumé

Cette thèse est composée de trois applications des techniques d'apprentissage automatique à l'évaluation empirique d'actifs.

Dans la première partie, qui est un travail en collaboration avec Oksana Bashchenko, nous développons une nouvelle méthode qui détecte les sauts de manière non paramétrique dans les séries chronologiques financières et qui surpasse de manière significative la référence actuelle sur des données simulées. Nous utilisons un réseau neuronal à mémoire à long terme (LSTM) qui est entraîné sur des données étiquetées générées par un processus qui connaît à la fois des sauts et des explosions de volatilité. En conséquence, le réseau apprend à distinguer les deux. Il est ensuite appliqué à des données simulées hors échantillon et donne des résultats qui diffèrent considérablement de ceux de la literature : nous obtenons moins de détections fallacieuse et identifions un plus grand nombre de véritables sauts. Lorsqu'elle est appliquée à des données réelles, notre approche permet d'extraire un signal plus précis qui prédit la volatilité future.

Dans la deuxième partie, qui est un travail en collaboration avec Oksana Bashchenko, nous développons une méthodologie pour détecter les bulles d'actifs à l'aide d'un réseau neuronal. Nous nous appuyons sur la théorie des martingales locales en temps continu et utilisons un réseau profond pour estimer le coefficient de diffusion du processus de prix plus précisément que l'estimateur actuel, obtenant ainsi une meilleure détection des bulles. Nous montrons la performance supérieure de notre algorithme par rapport à la méthode statistique existante dans un laboratoire créé avec des données simulées. Nous appliquons ensuite la classification du réseau à des données réelles et construisons une stratégie de trading à exposition nette nulle qui exploite l'arbitrage risqué émanant de la présence de bulles sur le marché des actions américaines de 2006 à 2008. La rentabilité de la stratégie fournit une estimation de l'ampleur économique des bulles ainsi qu'un soutien aux hypothèses théoriques sur lesquelles elle s'appuie.

Dans la troisième partie, je propose un nouvel outil pour caractériser la résolution de l'incertitude autour des conférences de presse du FOMC. Il s'appuie sur la construction d'une mesure capturant le niveau de complexité des discussions entre le président de la Fed et les journalistes pendant les séances de questions-réponses. Je montre que les discussions complexes sont associées à des rendements boursiers plus élevés et à une baisse de la volatilité réalisée. La méthode crée un score d'attention en quantifiant dans quelle mesure le président doit s'appuyer sur la lecture de documents internes pour pouvoir répondre à une question. Pour ce faire, je construis un nouvel ensemble de données d'images vidéo des conférences de

presse et exploite les récents algorithmes d'apprentissage profond de la vision par ordinateur. Ces données alternatives fournissent de nouvelles informations sur la communication non verbale qui ne peuvent être extraites des transcriptions du FOMC largement analysées. Ce chapitre peut être considéré comme une preuve de concept que certaines vidéos contiennent des informations précieuses pour l'étude des marchés financiers.
**Mots-clés :** Evaluation d'actifs, apprentissage automatique, données alternatives, modélisation de volatilité, LSTM

# Contents

# List of Figures

# List of Tables

# Introduction

The three chapters of this thesis are connected by the fact that each one is an application of deep learning to an empirical asset pricing problem.

The first two parts use a long short-term memory network to improve the estimation of volatility of continuous time stochastic processes in high-frequency data. This specific architecture is chosen because of its ability to handle temporal sequences of data. The core concept linking these two papers is that we solve a problem of rolling window estimation in the context of a regime changing model. When estimating a parameter at every point in time in a regime shifting environment, it is natural to use a rolling window and process the time series data in chunks. However, selecting the optimal size for the window is a nontrivial task, especially when the regimes are not uniformly spaced in time. Using a fixed sized window, it is inevitable that at some point the estimator will be using data points from an old regime as well as observations belonging to a new regime at the same time. This implies that it will fail to correctly estimate parameters around times of a regime shift. Choosing a large window will lead to more stable coefficients however it will fail to identify the exact time of a regime change (or even miss some short lived regimes). A small window leads to unstable estimations. Partial solutions like decaying weights exist however they do not resolve the fundamental problem described above. The neural network solves this rolling window estimation problem. Intuitively, it is first able to detect the times of regime changes before estimating the parameters and optimally adjusting the window size. In a given identified regime, it includes all the data points from this regime for the estimation, maximising the number of observations being used but without using data from another regime.

This methodology is applied to detect jumps in the first chapter. In this case, a precise estimate of the continuous volatility at every point in time is crucial. In the second chapter, we estimate the functional form of the diffusion coefficient at every instant. The function detected is then used to determine if asset prices are experiencing a bubble or not by relying on a theory from mathematical finance. The outperformance of our technique over statistical methods for both applications is shown in a laboratory created with simulated data.

The third chapter employs deep learning models to extract information from alternative data (images) in order to track the nonverbal communication of Fed officials and its consequences on various market quantities. I am the first to analyze the behaviour of the Chair of the Federal

Open Market Committee (FOMC) using video content.

A variety of machine learning algorithms is used to perform a multitude of tasks like face detection, identity verification, facial landmarks extraction, etc. The flagship architecture used is a convolutional neural network as it is well known for its superior performance to analyze visual imagery. The key variable constructed tracks the reliance on hard information by the Chair (i.e. the need to read documents) when answering questions. The signal extracted from facial movements is then shown to explain contemporaneous equity returns and the change in volatility around the FOMC press conferences.

# 1 Deep Learning, Jumps, and Volatility Bursts (co-authored with Oksana Bashchenko)

## 1.1 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 1.2 describes the literature. Section 1.3 compares our method with the "classical" statistical tests and presents the main benchmark. Section 1.4 explains the main idea behind the architecture and the training of the network. Section 1.5 assesses the performance of our method on out-of-sample simulated data. Section 1.6 applies the algorithm to classify real data. Finally section 1.7 concludes.

## 1.2 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. 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 et al. (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] Such test can be useful for insider trading detection, studying the time-series of jumps (arrival in the morning/afternoon, self-exciting, presence of systematic/co-jumps risk) to validate assumptions of asset pricing models, or reduce the number of parameters (regarding stochastic intensity, correlation, and jump size distribution) for estimations . 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.

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

[2] Andersen et al. (2007b) also develop a jump test for each individual return similar to Lee and Mykland (2008) but assume constant intraday volatility.

Christensen et al. (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.

## 1.3 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 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

$$d\log S_t = \mu_t dt + \sigma_t dB_t + Y_t dq_t \tag{1.1}$$

with the drift $\mu_t$ and the diffusion coefficient $\sigma_t$ not changing dramatically over a short period

[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 $t$.

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 $\sigma_t$ 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

$$\mathscr{L}(i) = \frac{\log \frac{S(t_i)}{S(t_{i-1})}}{\hat{\sigma}(t_i)} \tag{1.2}$$

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 $\sigma_t$, 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 rate. When $\sigma_t$ 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 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

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

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

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

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.

## 1.4 Network architecture and training

The long short-term memory network we choose for jump classification is a special type of 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. We simply select a plain vanilla structure which is detailed in appendix A. We have a total of 82 002 parameters that are trained on 175 years of high-frequency data at 2min which implies that we use roughly 100 data points per parameter to avoid overfitting. Given the very large imbalance of the classes in this dataset, the accuracy has to be extremely high for the network to be considered properly trained.

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].

[6] Apart from the necessary computational time.

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 $\sigma_t$:

$$dS_t = S_t(\mu_t dt + \sigma_t dB_{1t} + dJ_{1t}), \tag{1.3}$$

$$d\sigma_t = \alpha(\overline{\sigma} - \sigma_t)dt + \nu\sqrt{\sigma_t}dB_{2t} + dJ_{2t}. \tag{1.4}$$

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 [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 (1.3) and (1.4). The procedure would work using any data-generating process to create labelled data and then following the same steps.

## 1.5 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

[7] For further details see Cont and Tankov (2004).

data satisfies their assumption about "local" changes in spot volatility. Table 1.1 presents the 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 1.2 and 1.3.

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

Table 1.1 – 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 1.2 – 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 1.3 – 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 1.1 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 jump. We describe the data points by discussing four categories: (i) both methods agree and

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

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

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 1.2 and 1.3 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 $\sigma_t$ 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 1.2 and 1.3. 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 $\sigma_t$ 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

$$dS_t = S_t(\mu_t dt + \sigma_{1t} dB_{1t} + \sigma_{2t} dB_{2t} + dJ_{1t}), \tag{1.5}$$
$$d\sigma_{1t} = \alpha_1(\overline{\sigma}_1 - \sigma_{1t})dt + dJ_{2t}, \tag{1.6}$$
$$d\sigma_{2t} = \alpha_2(\overline{\sigma}_2 - \sigma_{2t})dt + dJ_{3t}. \tag{1.7}$$

The results stemming from this specification are displayed in table 1.4. As before our method outperforms the benchmark.

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

Table 1.4 – 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 (1.5)-(1.7).

As a final robustness check, we simulate the price by using (1.5) but the paths for $\sigma_{1t}$ and $\sigma_{2t}$ 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 $(\sigma_{1t})_{t=0}^{T}$ and $(\sigma_{2t})_{t=0}^{T}$. 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 1.5 and are in line with all previous findings.

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

Table 1.5 – 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 (1.5) but the paths for $\sigma_{1t}$ and $\sigma_{2t}$ are estimated from real data.

Confidence intervals can be build empirically for the LSTM network by constructing a histogram of the accuracy on each independent simulation. For each simulated path, we compute the % detection as defined previously for the network and for the Lee & Mykland estimator and

we subtract them. The result is displayed on figure 1.2. Most of the mass is over the positive region of the real line meaning that most of the time, our network has a higher percentage of correct detections than Lee & Mykland for the *same* simulated path.

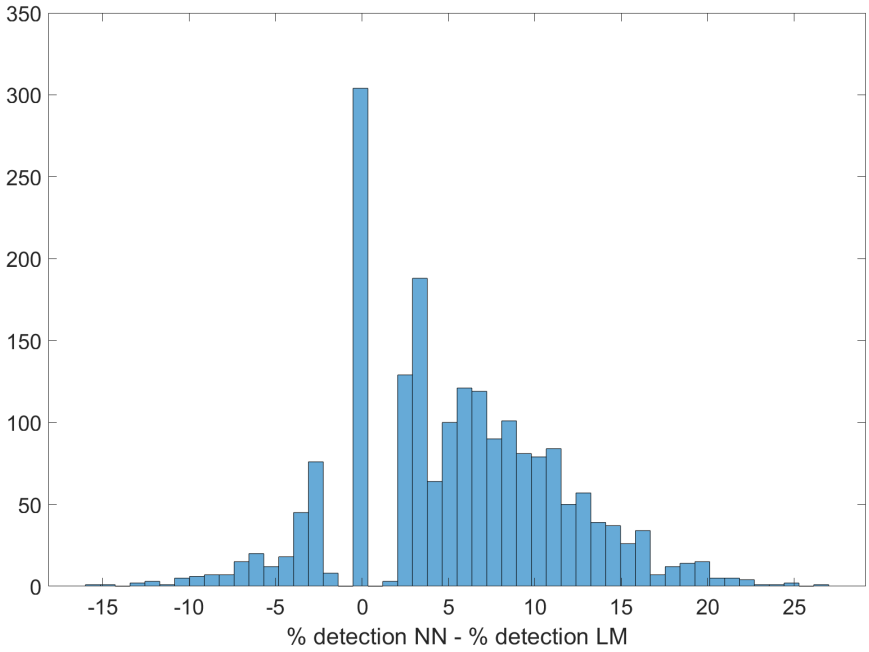



Figure 1.2 – Histogram of the percentage of correct detection by the LSTM network minus the percentage of correct detection by the estimator of Lee & Mykland over the same simulated path, repeated multiple times on independent draws.

We also build a similar histogram but for % spurious. The result is on figure 1.3 and this time most of the mass lays over the negative region. This means that on the *same* path, our network is much less often mistaken.

The estimator of Lee and Mykland (2008) only relies on past and current information when classifying a datapoint. However if the application is simply to analyze historical data there is no reason to impose such restriction. This is why whenever we use an LSTM network we construct it with a bidirectional layer which also accesses future data in order to improve the performance. But in the interest of a fairer comparison, in this paragraph we will discuss the relative performance of the statistical estimator and a unidirectional LSTM. Both of them will therefore rely on the same information set (the past and the present). The results are presented in table 1.6. We observe that even thought the performance of the network decreases if we use a unidirectional layer, it remains superior to the statistical test. Histograms are provided in figures 1.4 and 1.5.

| | Bi-Network | Lee & Mykland |
|---|---|---|
| % detection | 90.94 | 85.50 |
| % spurious | 10.09 | 114.96 |
| | Uni-Network | Lee & Mykland |
| % detection | 89.94 | 85.47 |
| % spurious | 17.96 | 113.73 |

Table 1.6 – Comparison of bidirectional and unidirectional LSTM networks with the statistical test of Lee and Mykland (2008).

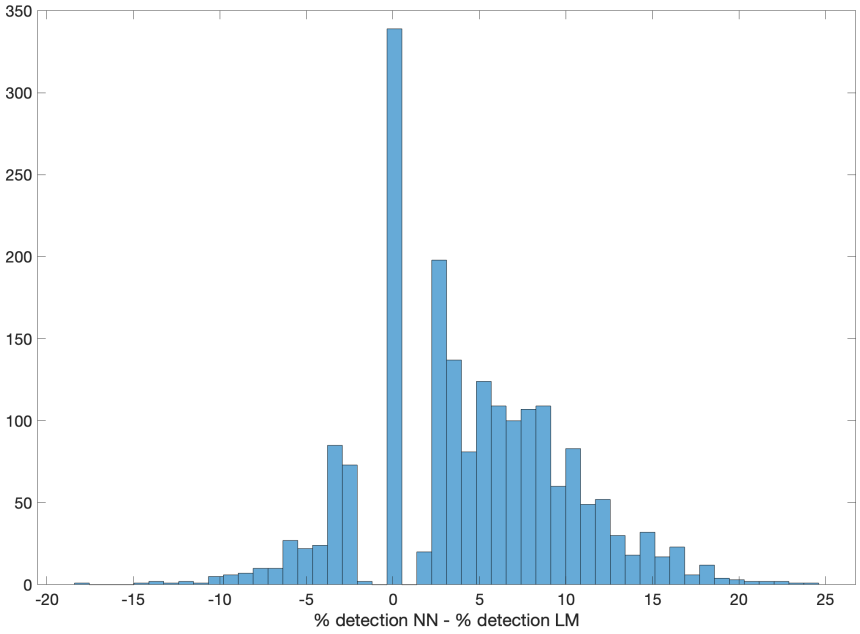



Figure 1.4 – Histogram of the percentage of correct detection by the unidirectional LSTM network minus the percentage of correct detection by the estimator of Lee & Mykland over the same simulated path, repeated multiple times on independent draws.

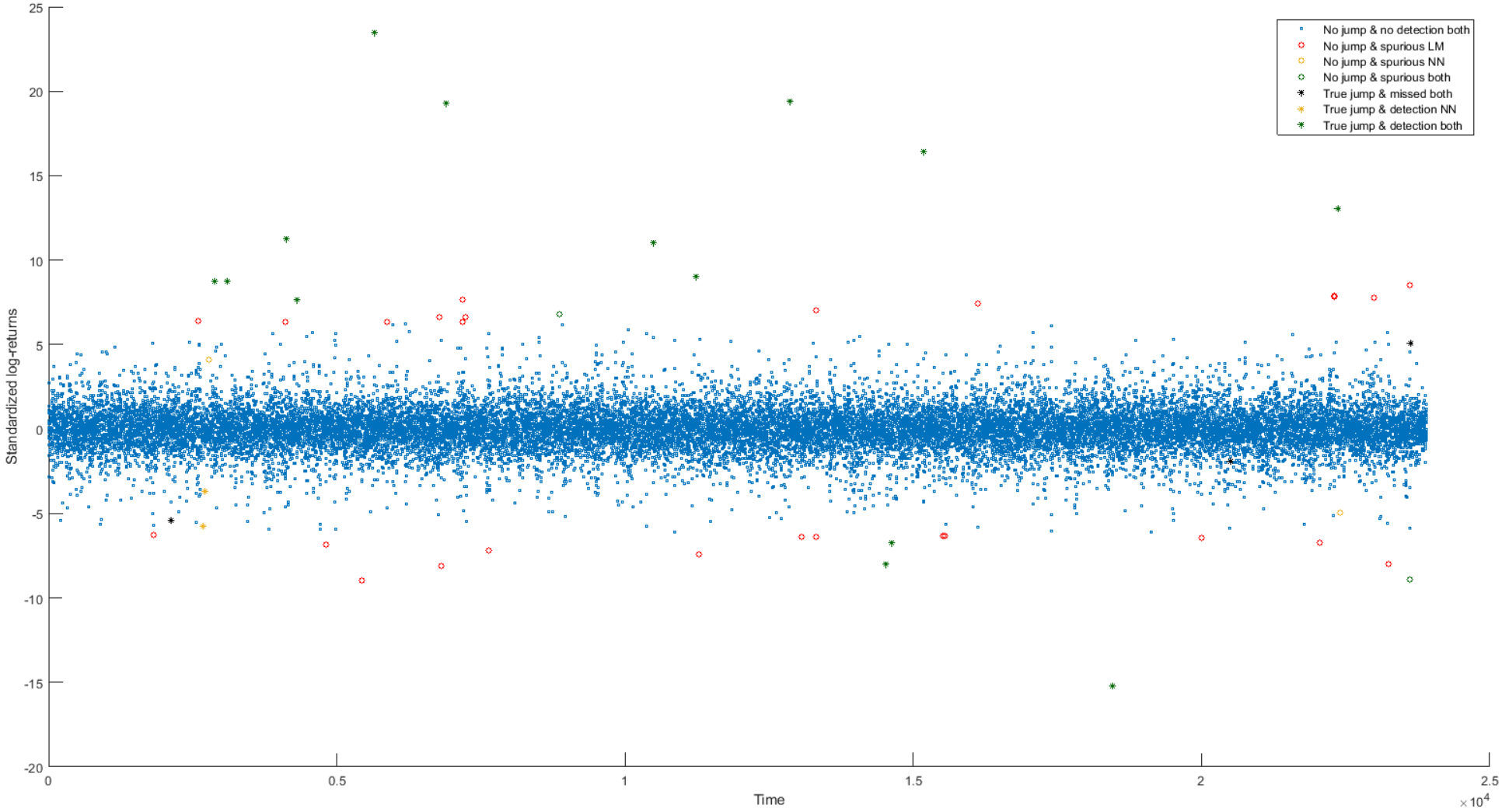



Figure 1.1 – Simulated data. Returns are standardized by the bipower variation and sampled at 2 min frequency for 0.5 year. The spot volatility itself jumps frequently.

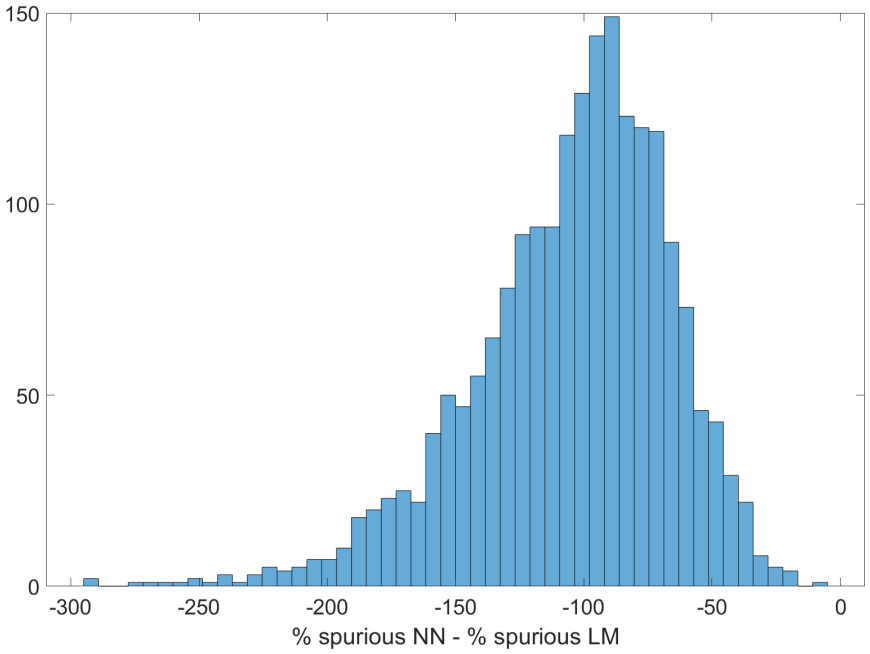



Figure 1.3 – Histogram of the percentage of spurious detection by the LSTM network minus the percentage of spurious detection by the estimator of Lee & Mykland over the same simulated path, repeated multiple times on independent draws.

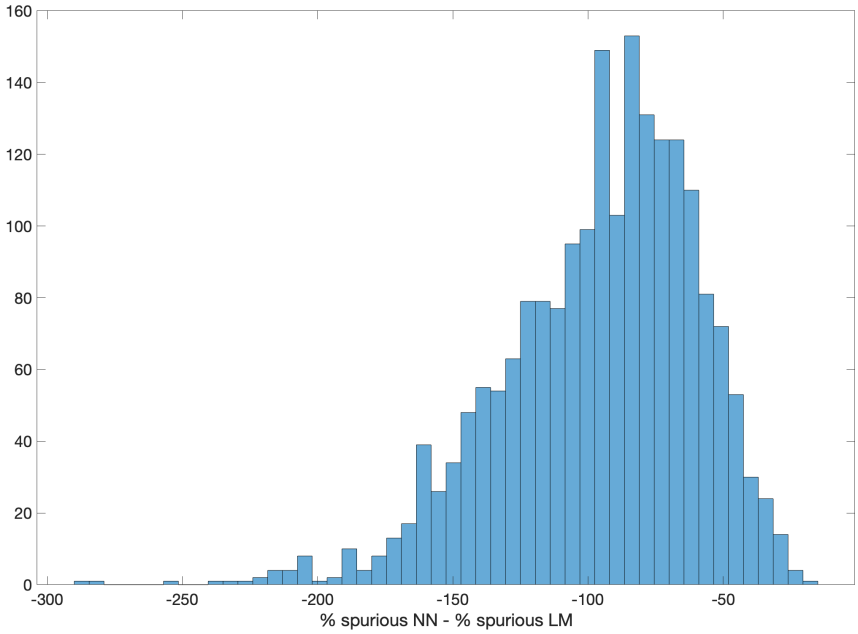



Figure 1.5 – Histogram of the percentage of spurious detection by the unidirectional LSTM network minus the percentage of spurious detection by the estimator of Lee & Mykland over the same simulated path, repeated multiple times on independent draws.

## 1.6 Application to real data

In this section we perform jump classification on real data sampled at 2 minutes obtained from the Trade and Quote (TAQ) database. 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 1.6 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 et al. (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.

### 1.6.1 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 1.6 in the center of the orange circle. In order to have a more granular view of this event, figure 1.7 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?

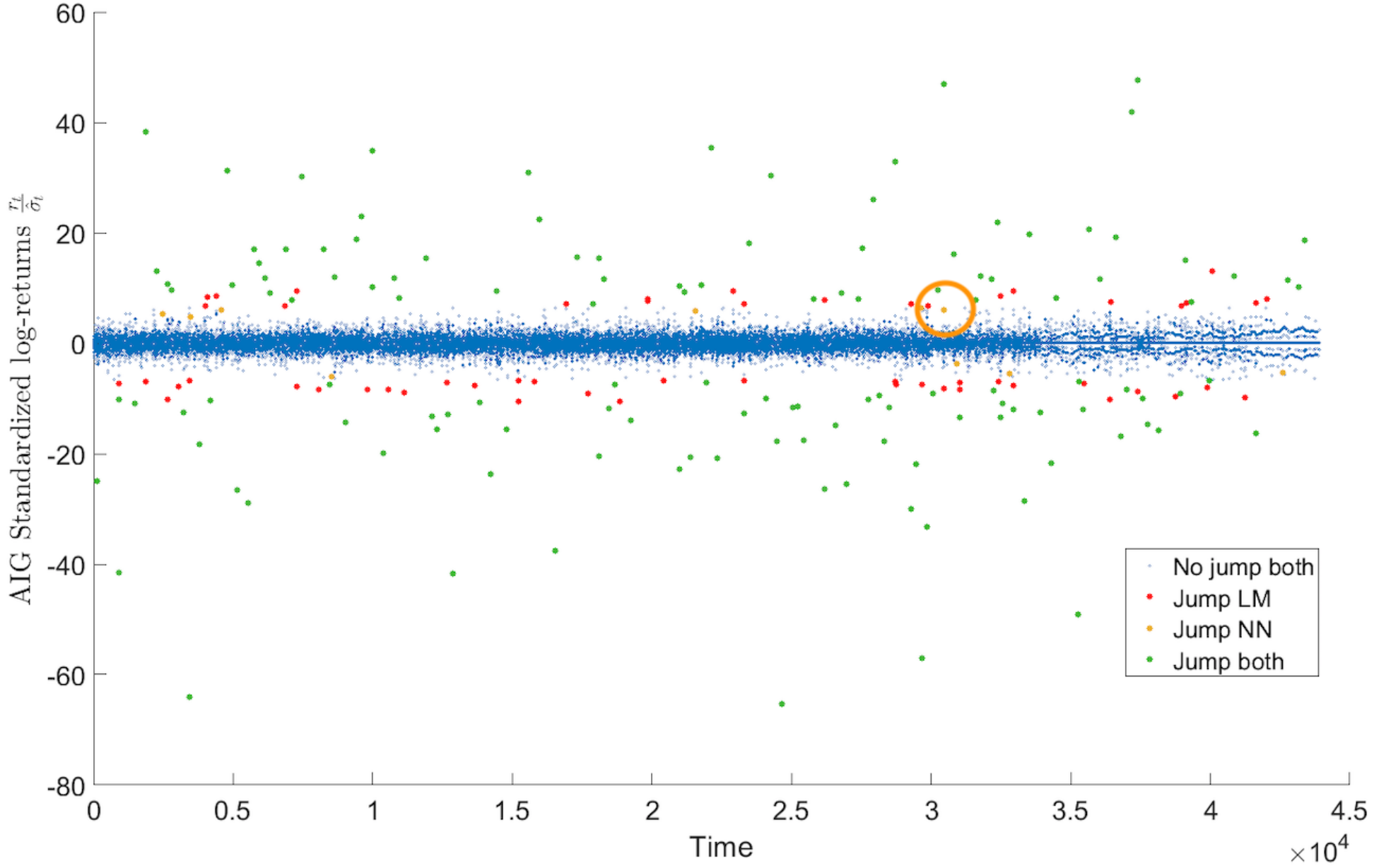



Figure 1.6 – Returns of American International Group (AIG) standardized by the bipower variation at 2 min frequency. Regarding the legend: "No jump both" means that none of the methods detected a jump, "Jump LM" means that a jump was detected only by the test of LM, "Jump NN" means that a jump was detected only by our network, and finally "Jump both" means that both LM and the network agree and detect a jump.

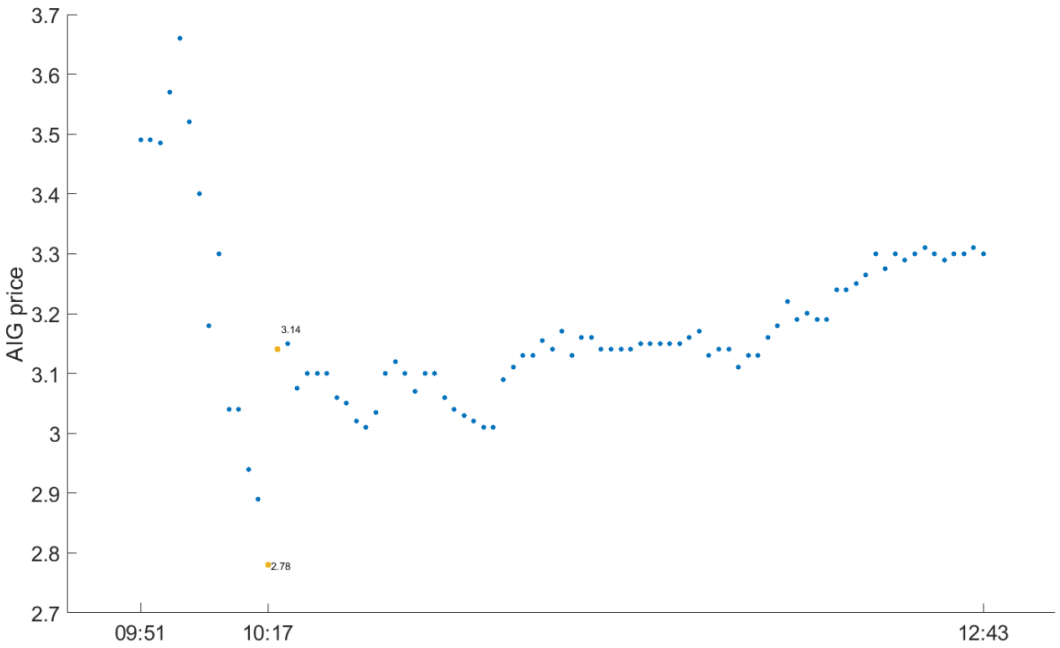



Figure 1.7 – AIG stock price in the morning of September 19, 2008 sampled at a 2 minutes frequency.

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.

### 1.6.2 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. Andersen et al. (2007a) 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 $\mathrm{RV}_t^d$ (see appendix B for the formula). The weekly volatility is simply an average of the daily quantities over five days

$$\mathrm{RV}_t^w = \frac{1}{5}\left(\mathrm{RV}_t^d + \mathrm{RV}_{t-1d}^d + ... + \mathrm{RV}_{t-4d}^d\right). \tag{1.8}$$

The monthly volatility $\mathrm{RV}_t^m$ 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

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

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 $\mathscr{BPV}$. A second approach ($\mathscr{LM}$) 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 ($\mathscr{NN}$) 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

$$\begin{aligned}\mathrm{RV}_{t+f}^f = \alpha_{f,i} + \beta_{f,i}^{d,C}\mathrm{C}_{t,i}^d + \beta_{f,i}^{w,C}\mathrm{C}_{t,i}^w + \beta_{f,i}^{m,C}\mathrm{C}_{t,i}^m \\ \beta_{f,i}^{d,J}\mathrm{J}_{t,i}^d + \beta_{f,i}^{w,J}\mathrm{J}_{t,i}^w + \beta_{f,i}^{m,J}\mathrm{J}_{t,i}^m + \mathrm{error}_{t+f},\end{aligned} \tag{1.9}$$

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

[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 comparison of the out-of-sample forecasting performance of the different methods is displayed in table 1.7. 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. 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 $\mathscr{LM}$ and $\mathscr{NN}$ 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 |
| $\mathscr{PV}$ | 62,41 | 1,12 | 69,83 | 0,88 | 64,56 | 0,89 |
| $\mathscr{BPV}$ | 64,69 | 1,09 | 71,75 | 0,85 | 65,78 | 0,88 |
| $\mathscr{LM}$ | 64,98 | 1,08 | 72,42 | 0,84 | 65,72 | 0,88 |
| $\mathscr{NN}$ | 65,38 | 1,07 | 72,53 | 0,83 | 65,99 | 0,87 |

Table 1.7 – 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.

## 1.7 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.

# 2 Deep Learning for Asset Bubbles Detection (co-authored with Oksana Bashchenko)

## 2.1 Introduction

Asset bubbles have preoccupied investors nearly since the beginning of financial markets. From the Tulip mania of 1636-1637 all the way to the more recent financial depression of 2008, including many others in between. Recently, bubble concerns have resurfaced with the rise of crypto assets experiencing rapid increase and violent crashes in prices. The concept of a bubble (defined as a deviation of the price from the fundamental value) might be well known for market participants but its detection remains a challenging task because the fundamental value is not observable.

The fundamental value is the expected sum of future cash-flows or benefits offered by the security, appropriately discounted for time and risk. The difficulty comes from the fact that one always needs some assumptions in order to make a statement about it. One approach is to build a theoretical model for the fundamental value but then the bubble detection procedure would suffer from a joint-hypothesis problem (you are simultaneously testing the presence of a bubble and the assumptions of the model). A second approach is to use almost model-free techniques from mathematical finance under the assumption of a continuous-time economy. Essentially, this boils down to classifying stock prices as being either true or strict local martingales. However to do so it is necessary to estimate the volatility function of a diffusion process given discrete time-series data and this method suffers from estimation errors.

This paper aims at providing a bubble detection methodology that outperforms the existing one under a continuous-time paradigm. To the best of our knowledge, we are the first to provide a certain number of findings to the literature. Our first contribution is to show that long short-term memory (LSTM) networks outperform current methods at detecting asset bubbles both in terms of accuracy and computational time. The second contribution is to provide an evaluation of the method proposed in Jarrow et al. (2011) and Obayashi et al. (2017) in a controlled environment. Indeed, by using simulated data we know exactly when an asset bubble occurs and are able to assess the accuracy of their method (and in a second step

compare it to ours). Finally, our last contribution is to assess the economic significance of these bubbles. We essentially detect them on real data and build a long-short trading strategy to assess how much a trader could potentially earn by exploiting this risky arbitrage.

The profitability of the strategy provides support for theoretical foundations that use strict local martingales to detect bubbles. It empirically shows that it is valid to consider the price as a continuous-time process that can only be observed at discrete random times (the trading times), as postulated in Jarrow and Protter (2012).

The rest of the paper is organized as follows. Section 2.2 describes the literature. Section 2.3 lays out the theoretical foundations for characterizing bubbles in continuous-time and presents the existing methods for detecting them. Section 2.4 explains the main idea behind the architecture and the training of the network. Section 2.5 assesses the performance of our method on out-of-sample simulated data. Section 2.6 applies the algorithm to detect bubbles using real data and builds a trading strategy that exploits this risky arbitrage. Section 2.7 discusses the implications of a discrete-time paradigm for our results. Finally section 2.8 concludes.

## 2.2 Literature review

With the focus being mostly on the martingale theory of bubbles, our paper is largely inspired by the insights of Jarrow et al. (2007) and Jarrow et al. (2010). In these studies the authors provide conditions for bubbles to exist in complete and incomplete markets respectively. Importantly, they show that a non-trivial bubble can be of three types. Two of them can exist only in infinite horizon economies, thus being less relevant for empirical applications. The third type of bubbles, that is the only type viable in the finite horizon framework, is proven to be represented as a strict local martingale under a risk-neutral measure. We are interested in detecting bubbles of that third type in data, and it explains our inclination to work with a continuous-time framework. The interested reader may find more details about the mathematical foundation of bubbles in continuous-time in the work of Protter (2013), while a general review of the asset price bubbles is proposed in Jarrow (2015). Hugonnier (2012) provides a theoretical framework for rational bubbles to appear in equilibrium asset pricing.

For diffusion processes with volatility being a function of the price solely, Delbaen and Shirakawa (2002) and Mijatović and Urusov (2012) provide a theoretical test based on the instantaneous volatility to distinguish between a true and a strict local martingale. The difficulty in applying this test for data is that the functional form of the volatility has to be known. Jarrow et al. (2011) try to solve this problem by proposing one parametric and multiple non-parametric methods for estimating it. One limitation of their paper is that the functional form of the volatility has to stay the same over the whole sample period. The recent work of Obayashi et al. (2017) mitigates this constraint by allowing the functional form to vary among different time periods. This results into adding more flexibility to the method, but

deteriorating the quality of each period estimation, leaving space for further improvement. We propose a new direction for this literature, departing from the classical statistical test by using deep learning techniques.

Machine learning methods are gaining a huge popularity in financial research, both among academicians and practitioners. With financial data being mostly represented as time-series, the family of recurrent neural networks is the logical choice for the majority of problems in the field. A prominent member of this class is the long short-term memory network (LSTM) proposed by Hochreiter and Schmidhuber (1997). It is able to remember important information that was faced in the far past and forget unimportant recent one. This type of network is so far the most popular for tasks involving complex historical patterns of data.

## 2.3 Bubbles in continuous-time

There are two paradigms on modeling asset prices: discrete or continuous-time. Often, both frameworks are equivalent in the sense that they describe the same phenomenon. For instance, for derivative pricing a discrete-time binomial model approximates the Black-Scholes formula which relies on a continuous-time framework. However this equivalence is in general not true.

Whether we adopt a discrete or continuous-time approach greatly changes the theoretical framework and assumptions required for bubbles detection. In a discrete-time economy, bubbles can only exist if the time horizon is infinite. Moreover, the only way to detect them is by relying on a model for the unobserved fundamental value, creating a joint-hypothesis problem. In continuous-time, bubbles can exist even in finite horizon economies and we only need to assume a certain stochastic differential equation (SDE) describing the evolution of the price. We can assess whether the SDE accurately describes the price changes independently from testing for the presence of bubbles (hence eliminating the joint-hypothesis problem). In both cases, we are forced to make a relatively strong assumption: either infinite time horizon or continuous-time. In this paper we have decided to make the latter assumption. This implies that we use objects (strict local martingales) which only exist in continuous-time models but not in their discrete-time counterparts.

It is therefore natural to wonder: how valid is this assumption? And how relevant are the results for the real world? In reality it is true that, despite the fact that high-frequency firms trade with incredibly short intervals, it still happens at discrete times. However, a trade can happen at any point on the half-line $[0,\infty)$ that represents time. At the tick level, the time between trades is not uniformly spaced as pointed out by Jarrow and Protter (2012). This contrasts with a discrete-time model where the time grid is fixed and trading is forced to happen at uniformly spaced dates. This is why it is more plausible to imagine that the price process evolves in continuous-time and we are only able to observe its values at random stopping times (when a trade takes place). Moreover, even though our objects of interest only exist in continuous-time, they can be approximated by discrete-time processes (see appendix C.0.3 for more details). A longer discussion on this topic is available in Protter (2013) in section 11.

Nevertheless, if the reader is a firm believer of the discrete-time framework, we can still give meaning to our results. We show that what we detect are true martingales but with significantly skewed distribution. We discuss in depth the link with a discrete-time economy in section 2.7.

### 2.3.1 Theoretical framework

We essentially use the set-up described in the annual review of Jarrow (2015) which is also common to most of the papers in this literature. We keep the theoretical framework to a minimum to obtain a concise paper but we still ensure that the reader can understand our main contributions.

Let $(\Omega, \mathscr{F}, \mathbb{F}, \mathbb{P})$ be a filtered probability space where the filtration satisfies the usual hypothesis (Protter (2001)). Time is continuous and the economy lives over a compact time interval $[0, T]$ with $T < \infty$. As discussed earlier, we only study bubbles that occur in finite horizon economies. There exist two assets. The risky asset price is denoted by $S_t$ and its real world dynamics is described by

$$dS_t = b(t, S_t)dt + \sigma(t, S_t)dB_t, \quad S_0 = s_0 \tag{2.1}$$

where $B_t$ is a one-dimensional $\mathbb{P}$-Brownian motion. We denote by $\tau \leq T$ the random terminal date[1] of this asset. The second asset is locally riskless and pays the risk-free rate $r_t$. Its value is given by the solution of the following differential equation

$$dS_{0t} = S_{0t} r_t dt, \quad S_{00} = 1. \tag{2.2}$$

The discounted price of the risky asset is defined in the classical way as

$$\hat{S}_t \triangleq \frac{S_t}{S_{0t}}. \tag{2.3}$$

As mentioned before, an asset is in a bubble when the spot price is trading above the fundamental value. So first we need to clarify what the fundamental value is. We assume that the cumulative dividends process $D = (D_t)_{0 \leq t \leq \tau}$ is an adapted càdlàg non-negative semimartingale. Then the fundamental value $S_t^\star$ is defined in a standard way as the expected sum of discounted cash-flows to be paid. Let $\mathscr{M}$ be a non-empty set of equivalent local martingale measures (ELMM). We fix[2] a measure $\mathbb{Q} \in \mathscr{M}$ and write

[1] This date can be thought as a bankruptcy or when the shareholders decide to liquidate the assets of the firm for instance.

[2] For concerns regarding the incompletness of the market we refer to the discussion in Jarrow (2015).

$$\hat{S}_t^\star \triangleq \mathbb{E}_t^{\mathbb{Q}}\left[\int_t^\tau \frac{dD_s}{S_{0s}} + \hat{L}_\tau\right] \tag{2.4}$$

where $\hat{L}_\tau \geq 0$ is the discounted liquidation value of the asset at the terminal date. We formally define a bubble process as

$$\beta_t = S_t - S_t^\star. \tag{2.5}$$

We say that an asset experiences a bubble whenever $\beta_t > 0$. The next theorem characterizes this phenomenon. For simplicity of exposition, we assume no dividends from now on.

**Theorem 2.3.1 (Obayashi et al. (2017))** *A risky asset price process $S$ is undergoing bubble pricing on the compact time interval $[0,T]$ if and only if under the chosen risk neutral measure the discounted bubble process $\hat{\beta}$ is not a martingale but is a strict local martingale. This is equivalent to the discounted price process $\hat{S}$ (plus its cumulative discounted dividends) being a strict local martingale since $\hat{S}^\star$ (plus its cumulative discounted dividends) is always a martingale and not a martingale.*

Theorem 2.3.1 provides us with a simple criterion for detecting bubbles by identifying strict local martingales. Now we can fully understand the implication of our assumptions since we aim at detecting strict local martingales which only exist in continuous-time. A benefit of this setting is that we do not need to model the fundamental value, all we need to know is whether the discounted stock price is a strict local martingale or not.

Under $\mathbb{Q}$ the process $\hat{S}_t$ has the following dynamics

$$d\hat{S}_t = \sigma(t, S_t) S_{0t}^{-1} dB_t^{\mathbb{Q}}. \tag{2.6}$$

By Girsanov theorem, the diffusion coefficient stays the same under any equivalent change of measure, so without loss of generality we can in fact work with any $\mathbb{Q}$. Now that we have the risk-neutral dynamics, we need to know how to differentiate between true and strict local martingales. From now on, we assume the interest rate to be constant and equal to zero for simplicity of exposition. This will not impact our results on simulated data. Regarding the application on real data where rates are not constant, analogous results will hold under some technical conditions. A discussion can be found in Protter (2013).

Thanks to the work of Delbaen and Shirakawa (2002) we have a simple condition to distinguish between true and strict local martingales. For a process of the form

$$dX_t = b(X_t)dB_t^{\mathbb{Q}}, \tag{2.7}$$

we first define an integral

$$I(\varepsilon) \triangleq \int_{\varepsilon}^{\infty} \frac{x}{b(x)^2} dx. \tag{2.8}$$

Then depending on whether this integral converges or not two cases can arise:

- $X$ is a strict local martingale (i.e. experiences a bubble) if and only if $I(\varepsilon) < \infty$ for all $\varepsilon > 0$.
- $X$ is a true martingale (i.e. does not experience a bubble) if and only if $I(\varepsilon) = \infty$ for all $\varepsilon > 0$.

At this point, we need to make assumptions about the diffusion coefficient $\sigma(t,s)$ in the SDE (2.6). The bubble testing procedure requires us to obtain this coefficient solely as a function of $s$, and not as a function of time. However assuming that the volatility function only depends on the stock price and that it never changes through time is unrealistic. This would imply for instance that if a stock is in a bubble, it will stay in this state forever. We therefore follow Obayashi et al. (2017) and assume a regime change model for the diffusion coefficient such that

$$\sigma(t,x) = \begin{cases} \sigma_1(x), \text{ if } MC_t = 1 \\ \sigma_2(x), \text{ if } MC_t = 2 \\ \dots \\ \sigma_n(x), \text{ if } MC_t = n, \end{cases} \tag{2.9}$$

where $MC_t$ is a Markov chain with $n$ regimes. Note that time intervals do not need to be equally spaced. In general, they are random and unknown. Now over a given time interval $(t_i, t_{i+1}]$ we will be able to estimate the diffusion coefficient as a function of $s$ only and perform the integral test locally using (2.8). In a theoretical model, the computation of $I(\varepsilon)$ is (easily) done. However, applying this test on real data is complicated by the fact that $\sigma(x)$ is not known. Its recovery becomes the cornerstone of the analysis. The tricky point of this estimation is that the functional form may vary over the ex-ante unknown time intervals. Another difficulty is that only data on a bounded interval is available for this estimation, while what matters for $I(\varepsilon)$ is the behavior of $\sigma(x)$ when $x \to \infty$.

For the readers familiar with the estimation of historical volatility, it is important to emphasize that the task is *not* to estimate the volatility through time but to estimate the functional form of volatility $\sigma(x)$ with respect to the price $x$ over some time interval. This implies that volatility estimators like realized variance (RV) are of no use for this task.

As another side note, it is worth noticing that $\beta_t \geq 0$ for any $t$ as shown in Jarrow (2015) in theorem 3 for instance. This means that in this framework, the stock price might be above the fundamental value but never below. It implies that when later we build a trading strategy using real data, we will always short stocks that are classified as being in a bubble.

### 2.3.2 Classical statistical method for bubble detection

In this section, our goal is to motivate the necessity of a neural network to detect bubbles by showing that it significantly improves the detection rate. We do so by constructing a simple example on simulated data where the stock volatility is assumed to only have two regimes. We show that already here the method described in the literature often underperforms. This example is unrealistically simple but sufficient to prove our point. Later we will use a more complex structure for the data-generating process when making the final comparison of both methods.

We now discuss the method presented in Obayashi et al. (2017). The authors decide to estimate $\sigma(x)$ over a rolling window of 21 days and propose two estimators. The first one being a parametric estimator and a second non-parametric one based on the local time of a Brownian motion. In that case, they can analyze when a bubble is born and when it bursts (and ultimately construct a distribution of the lifetime of a bubble).

In our paper we will only present and benchmark our network to the parametric estimator (PE). Given the fact that we will assess the performance of the two methods on simulated data, we always know the true data-generating process. We use the same family of functions for data simulations and for the parametric estimator. Since the PE uses a function from the true family, its performance will be higher than a non-parametric estimator. The latter might lead to better performance on real data where the functional family is unknown. However, with real data we cannot compare the methods because we do not know the true volatility regime.

In order to simulate data, we first assume a functional form for $\sigma(x)$. We choose the well-known power family[3]

$$\sigma(x) = \gamma_0 x^{\gamma_1} \tag{2.10}$$

where $\boldsymbol{\gamma} = (\gamma_0, \gamma_1)$ is constant over a given time interval $(t_i, t_{i+1}]$ but is allowed to change from

[3] This functional form implies that we model the stock price with a constant elasticity of variance (CEV) model.

one time interval to another. Given this function, the stock price $S_t$ is a strict local martingale for $\gamma_1 > 1$. In this simplified example we simulate price paths by using only two regimes:

**R1** When the stock has $\gamma_1 = 0.9$.

**R2** When the stock is in a bubble (i.e is a strict local martingale) and we set $\gamma_1 = 1.1$.

A Markov chain decides on the random times when the stock transits back and forth between regimes **R1** and **R2**.

It is important to emphasize that in a model like equation (2.9), even if today the stock is in a state with $\gamma_1 \leq 1$, we cannot conclude that there is no bubble. Because the existence of a positive probability to transition to a state with $\gamma_1 > 1$ would imply that there is a bubble today. The only thing that our methodology is able to achieve is to detect bubbles whenever we are in a regime with $\gamma_1 > 1$.

It is clear that the scaling parameter $\gamma_0$ does not impact the convergence of the integral in (2.8) so we set $\gamma_0 = 0.15$ for both regimes for simplicity. Once the data is simulated, we need to estimate $\boldsymbol{\gamma}$.

The above-mentioned paper relies on a parametric estimator originally introduced in Genon-Catalot and Jacod (1993) which is given by

$$\hat{\boldsymbol{\gamma}}^{\mathrm{PE}} = \operatorname{argmin} \frac{1}{n} \sum_{i=1}^{n} \left( \gamma_0^2 S_{t_{i-1}}^{2\gamma_1} - \frac{n}{T} \left( S_{t_i} - S_{t_{i-1}} \right)^2 \right)^2. \tag{2.11}$$

Since the data is simulated we know exactly when the stock was in a bubble and we compare this information to the one given by the estimators. To do so, we construct the following random variable

$$\xi_t \triangleq \mathbb{1}_{\{1/2 \leq \gamma_{1,t} \leq 1\}} \tag{2.12}$$

which takes the value one when the stock is in a regime that we do not identify as bubble and zero otherwise. Then we denote by $\hat{\xi}_t^{\mathrm{PE}}$ and $\hat{\xi}_t^{\mathrm{NN}}$ the parametric estimator[4] (PE) and the neural network estimator (NN) respectively.

The results for a single price path are displayed in figure 2.1 which contains four subplots. The first one simply shows the log-returns of the price through time during the different market

[4] In order to obtain $\hat{\xi}_t^{\mathrm{PE}}$ we follow the methodology described in Obayashi et al. (2017). More precisely, we first use the estimator described in equation (2.11) and then apply a Hidden Markov Model to smooth the signal and use the probability matrices from their paper.

regimes. The second subplot displays $\xi$ which takes the value 0 when the stock experiences a bubble and otherwise is equal to 1. Here we see that over the whole period, the stock experienced 3 bubbly regimes and 3 "normal" regimes. Then the last two subplots display the estimation of $\xi$ according to the parametric estimator available in the literature and the neural network respectively. A direct visual inspection reveals that the neural network largely outperforms at detecting bubbles (i.e $\hat{\xi}_t^{\text{NN}}$ is a better estimator of $\xi$).

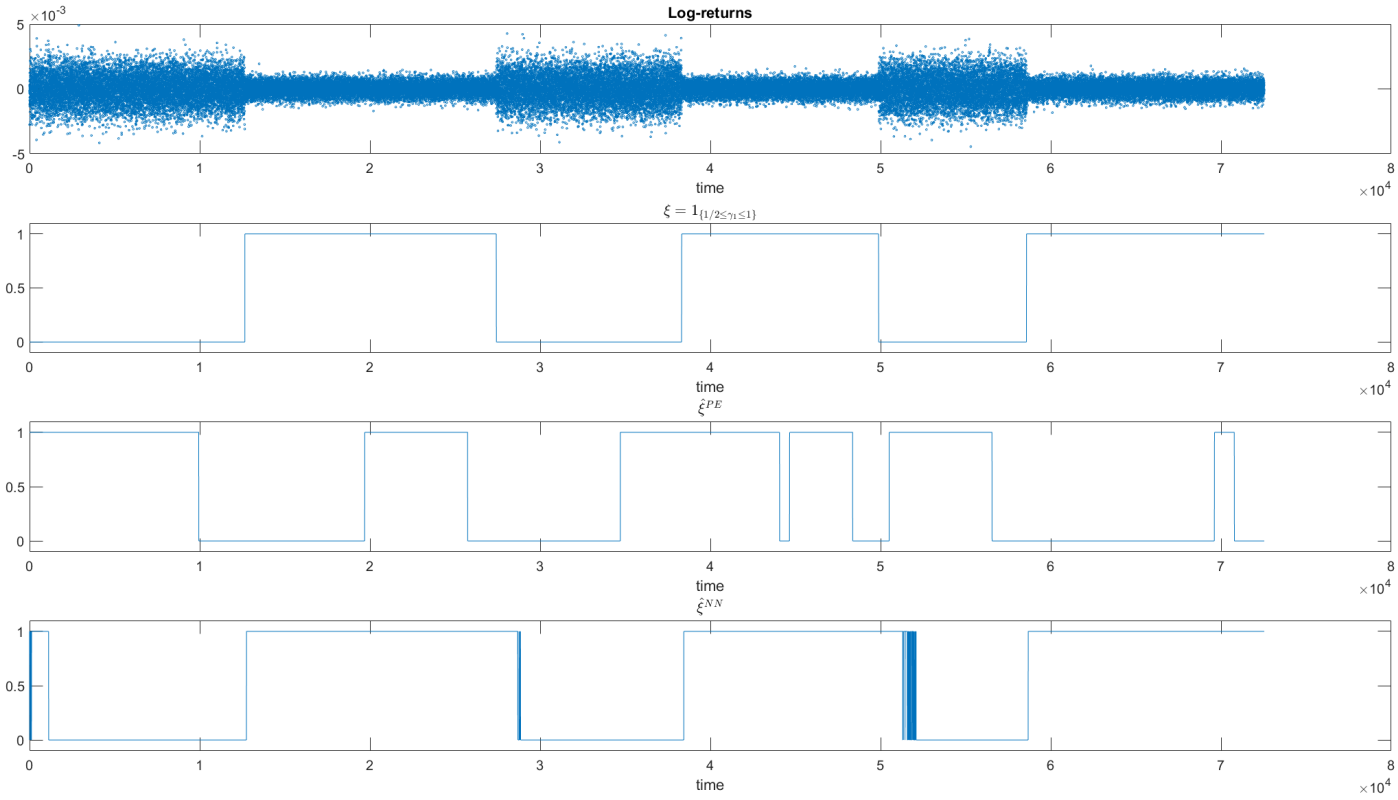



Figure 2.1 – Simulated data showing the drawback of the parametric estimator (PE)

Why is it the case? If $S_t$ stays in one specific regime (either true martingale or strict local martingale) over the whole sample path, then the statistical estimator performs quite well. However the statistical method starts to be challenged as soon as it has to perform the estimation of $\sigma(x)$ during a period when the regime changed. The reason is very intuitive. When using a rolling window around a regime change, the estimator will be using data points generated during the old regime as well as data points belonging to the new regime. This implies that it will fail to correctly estimate $(\gamma_0, \gamma_1)$ around these times. Choosing a large window will lead to more stable coefficients however it will fail to identify the exact time of a regime change (or even miss some short lived regime changes). A small window leads to unstable

estimations. Obayashi et al. (2017) mitigate this problem by combining their estimator with a Hidden Markov Model used to smooth the bubble signal. Nevertheless, as we see here it still underperforms a neural network.

How does the network manage to overcome this problem? Intuitively, it is first able to detect the times of regime changes before estimating the parameters $(\gamma_0, \gamma_1)$. By doing so, it does not use data from another regime. In a given identified regime, it includes all the data points from this regime for the estimation, using potentially more appropriate data than with a rolling window and thus improving the estimation.

This simple example is easy to understand but is far from reality. For instance the time-series of returns in figure 2.1 does not resemble real returns. This is why a thorough comparison of both methods in realistic conditions will be done in section 2.5 after introducing the neural network and the training process. However already in this toy example the statistical method struggles. In a more sophisticated environment its performance will deteriorate even further.

## 2.4 Network architecture and training

The long short-term memory network we choose for bubble classification is a special type of 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 modifications of LSTM networks and show that improvements over the plain vanilla LSTM are marginal. Our network consists of six layers: an input layer, followed by two bidirectional LSTM layers with 100 hidden neurons each, 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. Here we train the network with around 50 data points per parameter to prevent overfitting.

Now that the structure of the network is chosen, it has to be trained to detect strict local martingales. Our methodology is conceptually simple. We first generate training data and since it is simulated, we know exactly when the stock was a true martingale (TM) or a strict local martingale (SLM). Then we label each regime as "TM" or "SLM" 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 (2.6) by discretizing the stochastic differential equation governing it. The training data set as a whole can be composed by multiple time-series possibly generated from different processes by assuming different functional forms for $\sigma(x)$. Using a variety of them allows the network to concentrate only on the specific feature (TM or SLM) 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.

In order to incorporate stylized facts of equity data, we have decided to model $\sigma(t,x)$ from (2.9) with a Markov chain. Locally, every $\sigma_i(x)$ is modeled using the power function (2.10). To each state of the chain corresponds a different value for $\boldsymbol{\gamma}$. For half of the states we impose $\gamma_1 \leq 1$ while for the other half we set $\gamma_1 > 1$. The Markov chain will control when to switch from $\sigma_i(x)$ to $\sigma_{i+1}(x)$. The transition matrix is made time-varying to prevent the network from learning the probabilities instead of estimating $\sigma(x)$.

Neural networks are considered by some people as a black box. One of our aims is to make the usage of a network as transparent as possible. In section 2.6 we build a trading strategy and try to predict future price changes. However we do not simply feed past price data to a network and ask for a prediction. By doing so, we would not know what factors does the network use in order to make predictions. This approach would be totally model-free but would also be a sort of black box. Instead, we impose some structure on what the network learns by combining it with a model for the stock price. First we assume that the price is governed by a stochastic differential equation. We impose that the network recognizes certain functional forms for the diffusion coefficient (see eq. (2.10) for instance) and we link these functions to the mathematical theory of asset bubbles. Given the prediction of the theory, we can build a trading strategy.

## 2.5 Performance on simulated data

In this section we assess the performance of the neural network using out-of-sample simulated data. We start by generating new time-series, using parameters[5] that the network was never trained on. Then we classify the data by using the parametric estimator (2.11) and the neural network. Both methods were tested using 150 Monte-Carlo paths, each individual path consisting of data generated at a 2 minutes frequency for 3 years. The results are displayed in table 2.1. The row "% detection" refers to the percentage of data points that were correctly classified. Those include classifying the generating process as being a TM when it was a TM and classifying the process as being an SLM when it was an SLM. "% spurious" refers to the percentage of points when a method was mistaken. Those include classifying the process as being a TM when in reality it was an SLM and classifying the process as being an SLM when in fact it was a TM.

| | Network | Obayashi et al. (2017) |
|---|---|---|
| % detection | 83.64 | 49.71 |
| % spurious | 16.36 | 50.29 |

Table 2.1 – Comparison of the accuracy of the neural network and the statistical estimator from the literature for bubbles detection.

The results show that the neural network outperforms the existing statistical estimator by

[5] The new parameters are the values for $(\gamma_0, \gamma_1)$ as well as transitions probabilities for the Markov chain governing the volatility regime changes.

displaying a higher rate of correct and a lower rate of spurious detections. We have tried multiple different network architectures and tuned the hyperparameters so that to maximize the performance on out-of-sample simulated data. Gu et al. (2019) find that more shallow networks outperform deeper ones at forecasting financial time-series due to the low signal-to-noise ratio. We obtain a similar result although we use a network for a classification problem.

At this point, it is worth emphasizing that once the network is trained, it is much faster at classifying data. The parametric estimation (since it relies on an optimization) took approximately 43 hours to accomplish the required task while the network did it in less than one hour.

Just like in the previous chapter, we assess the empirical performance of the LSTM network by constructing a histogram of the accuracy on each independent simulation. For each simulated path, we compute the % detection as defined previously for the network and for the parametric estimator and we subtract them. The result is displayed on figure 2.2. Most of the mass is over the positive region of the real line meaning that most of the time, our network has a higher percentage of correct detections than parametric estimator for the *same* simulated path.

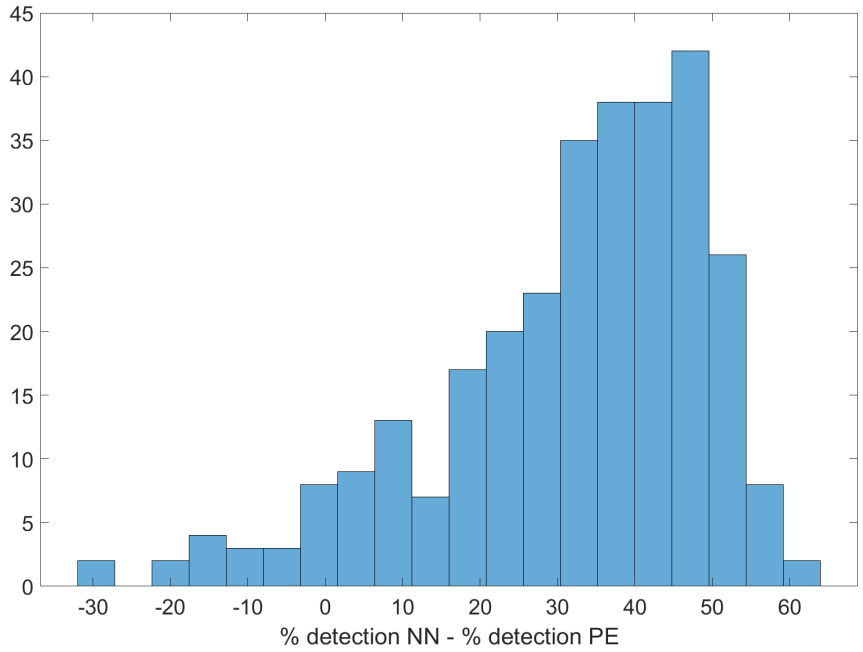



Figure 2.2 – Histogram of the percentage of correct detection by the LSTM network minus the percentage of correct detection by the parametric estimator over the same simulated path, repeated multiple times on independent draws.

We also build a similar histogram but for % spurious. The result is on figure 2.3 and this time most of the mass lays over the negative region. This means that on the *same* path, our network is much less often mistaken.

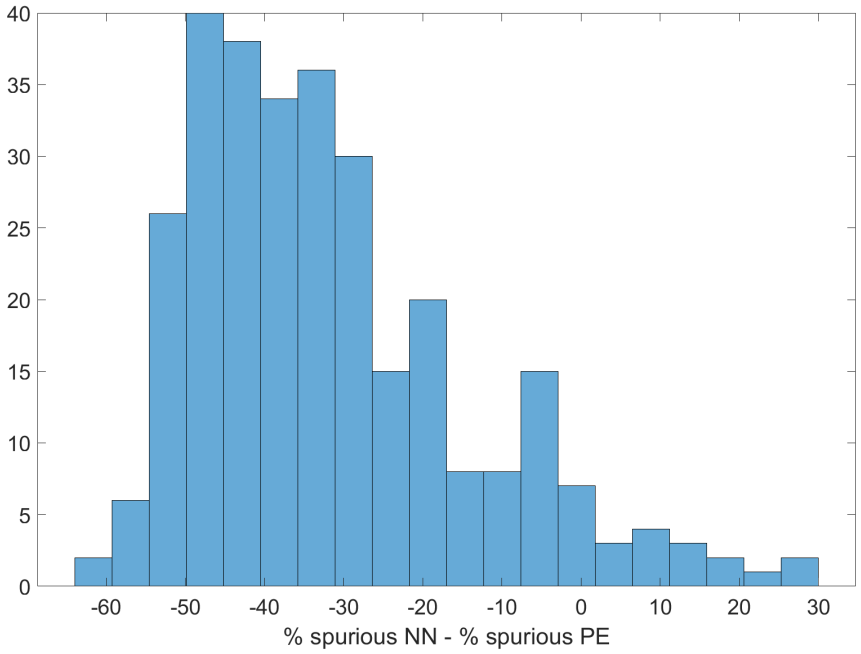



Figure 2.3 – Histogram of the percentage of spurious detection by the LSTM network minus the percentage of spurious detection by the parametric estimator over the same simulated path, repeated multiple times on independent draws.

In this chapter, using a unidirectional or bidirectional network will not significantly change the results of table 2.1. This is because the points living around regime changes constitute a small subset of all the data. This can also be seen by looking at the histograms in figures 2.4 and 2.5 that repeat the above procedure but with a unidirectional LSTM instead of bidirectional.

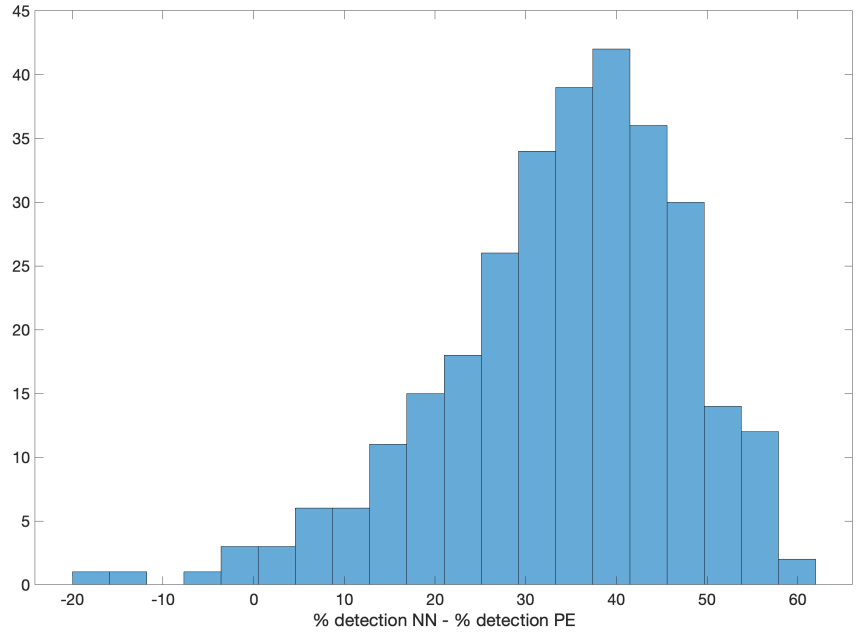



Figure 2.4 – Histogram of the percentage of correct detection by the unidirectional LSTM network minus the percentage of correct detection by the parametric estimator over the same simulated path, repeated multiple times on independent draws.

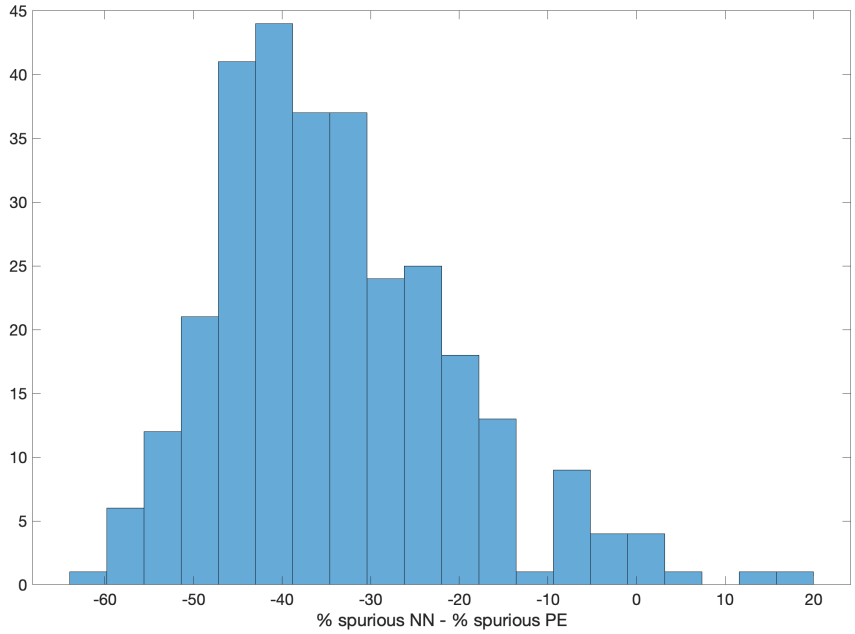



Figure 2.5 – Histogram of the percentage of spurious detection by the unidirectional LSTM network minus the percentage of spurious detection by the parametric estimator over the same simulated path, repeated multiple times on independent draws.

## 2.6 Application to real data

In this section we perform bubble classification on real data obtained from the Trade and Quote (TAQ) database using our neural network. The dataset consists of 30 individual stocks included in the Dow Jones index (plus the index itself) over a 3 years period (from 2006 to 2008). We select a sampling frequency of 2 minutes. This is high enough to have sufficiently many data points to precisely estimate the volatility function but yet low enough so that the data is not contaminated with microstructure noise. Here clearly we cannot exactly assess the performance of our test comparatively to the benchmark since the true data-generating process is unknown.

It is important to emphasize that up to now we only considered the dynamics of the discounted stock price $\hat{S}_t$ under the risk-neutral measure. By definition of the measure, this process has no drift. We did that because to determine if a stock is in a bubble all we need is the functional form of the diffusion coefficient in order to compute the integral (2.8). The theory tells us that when the stock price is in a bubble, its discounted value is a strict local martingale under $\mathbb{Q}$. Because it is bounded from below by 0, the discounted price is a $\mathbb{Q}$-supermartingale. This means that $\hat{S}_t$ is expected to decrease (under $\mathbb{Q}$). However, when trading stocks in the physical world, the processes describing their evolution likely have a non-zero drift under $\mathbb{P}$ (see equation (2.1) for the real world dynamics of the stock). Therefore to profit from the

existence of bubbles, it is necessary to go long and short in different assets in order to cancel any drift effect.

This leads us to a natural way to test our detection method and assess the economic magnitude of asset bubbles in the equity market. For that we build a risky arbitrage trading strategy that shorts an individual stock when in a bubble and at the same time goes long in an asset that replicates its fundamental value. Therefore the short-leg at any time $t$ of our strategy consists of all the individual stocks that are experiencing a bubble at $t$. The long-leg is the Dow Jones index (which is a proxy for the fundamental value of the short-leg). We invest the same dollar amount in the long and short legs so that we obtain a zero investment strategy. Given that our data covers the beginning of the financial crisis of 2008, the long-short strategy allows us to shield our portfolio from any market-wide shock (we have a zero net directional exposure).

Some classifications of time-series are are displayed in figure D.1 in the appendix. The performance of the trading strategy is presented in figure 2.6 where the vertical axis depicts the dollar value of the portfolio.

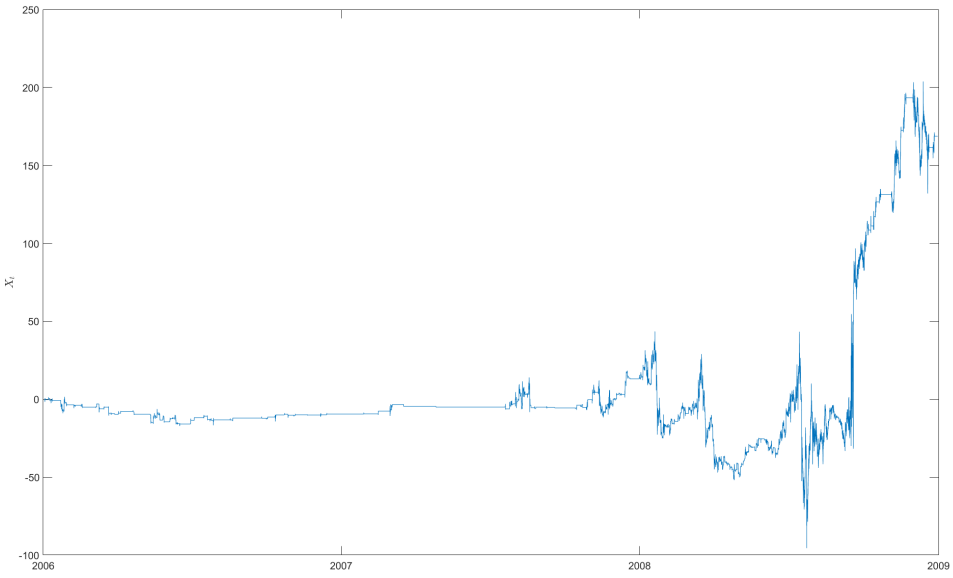



Figure 2.6 – Portfolio value of our long-short trading strategy with zero net exposure.

It is worth emphasizing that our trading strategy is not simply to short high volatility stocks. Suppose that a stock (call it A) has a high volatility but is a true martingale (if $\gamma_0$ is large but $\gamma_1 \leq 1$). It is possible that a second stock (call it B) has a lower volatility than stock A but is a strict local martingale (if $\gamma_0$ is low but $\gamma_1 > 1$). Our strategy would short stock B but not A.

## 2.7 Discrete-time paradigm

As discussed in section 2.3, strict local martingales are a continuous-time phenomenon and do not exist in discrete-time. Thus every method (not only our neural network) detecting bubbles by relying on the theory of SLM is subject to the assumption of time continuity. We have discussed previously why we consider such assumption as not only appropriate but in fact preferred over the discrete-time paradigm. However, if the reader is a strong believer in the latter, in this section we explain what would we detect if the true price process indeed evolved in discrete-time.

In this case we assume that the stock price follows the equivalent of our continuous-time process. For this purpose, let us first time-discretize the stock price (2.6)[6] as

$$S_{t+\Delta t} = S_t + \gamma_0 S_t^{\gamma_1} \sqrt{\Delta t}\, Z \tag{2.13}$$

where the process innovations $Z \sim \mathrm{N}(0,1)$ are iid under $\mathbb{Q}$. In discrete-time, $S_t$ is trivially a $\mathbb{Q}$-martingale for any $\boldsymbol{\gamma}$. However for $\gamma_1 > 1$ the mass of its transition density shifts to the left making it positively skewed. This implies that when $\gamma_1 > 1$, the stock price will often decrease (by a small amount) and rarely increase (but by a large amount), so that the conditional expectation is kept equal to the current value of $S_t$.

In this discrete-time setting since the time horizon of the economy is finite, bubbles cannot exist. Therefore, instead of detecting asset bubbles, our network will detect periods when the stock price frequently decreases by a small amount and rarely increases by a large amount. However, since $S_t = S_t^\star$ at all times, the long-short trading strategy previously explained would not generate any profits. The fact that the strategy is profitable (as seen in figure 2.6) provides additional support for the continuous-time paradigm.

## 2.8 Conclusion

Relying on the martingale theory of asset bubbles, we show that a long short-term memory (LSTM) network is able to detect them and outperforms the current statistical estimator. On a technical level, the network performs the detection by being able to determine if a given time-series has more likely been generated by a strict local or a true martingale. The algorithm does so by estimating the functional form of the diffusion coefficient of a stochastic differential equation and identifying market regime changes. We then deploy our methodology to US equity data and show that there were multiple bubbles between 2006 and 2008. Finally, using this information we construct a zero net exposure trading strategy that shorts assets experiencing a bubble to assess the economic significance of this phenomenon.

[6] Recall that we assumed a risk-free rate constant and equal to zero for simplicity.

Time continuity is a necessary assumption for the type of bubbles we detect. However this hypothesis is a source of debate among researchers. The profitability of our strategy is therefore an indirect support for the validity of using continuous-time processes to model asset prices.

# 3 Risk & returns around FOMC press conferences: a novel perspective from computer vision

## 3.1 Introduction

Most central banks actively try to shape expectations of market participants through forward guidance. Some of the main objectives being to impact the price of various securities which in turn influences the financing cost of companies or to reduce market volatility during turbulent times. Over the last few years, we have witnessed an explosion of research papers employing machine learning to analyze various documents produced by central banks. The goal is to measure quantitatively how they communicate. This is usually realized by assigning a sentiment score (positive/negative) to the language employed by the bankers using Natural Language Processing (NLP) techniques.

The contribution of this paper is to provide a new method to characterise the complexity of the discussion between reporters and the Chair of the Fed. Instead of analyzing the text documents, I use the video recordings of the FOMC press conferences and introduce a measure of attention exploiting computer vision algorithms. This is based on the simple premise that complex questions from journalists are followed by complex answers from the Chair, which often creates the need to consult internal documents in order to reply. The main idea is to differentiate between two questions asked by reporters, not by studying their text content, but rather by analyzing how does the Chair behave on the video when answering each question. This way, I am able to identify complex discussions by quantifying how often the Chair needs to look at internal documents. This is the key variable that video images are able to provide over other sources of data. I identify the events that involve more complex discussions and show that they have the largest (positive) impact on equity returns and reduce realized volatility. This highlights a mechanism of uncertainty resolution that works as follows. Answers to complex

This chapter is the preprint version of the following published article. Marchal A. (2022) Risk and Returns Around FOMC Press Conferences: A Novel Perspective from Computer Vision. In: Arai K. (eds) Intelligent Systems and Applications. IntelliSys 2021. Lecture Notes in Networks and Systems, vol 295. Springer, Cham. https://doi.org/10.1007/978-3-030-82196-8_54

questions resolve more uncertainty than answers to simple questions and this ultimately impacts stock returns, volatility and the equity risk premium around the press conferences.

Macroeconomic announcement days have been substantially discussed in the asset pricing literature which studies how much of the equity risk premium is earned around these events. Savor and Wilson (2013), Lucca and Moench (2015), Cieslak et al. (2019), and Hu et al. (2019) all find that a significant risk premium is earned around macroeconomic announcements. Ernst et al. (2019) argue that if you account for sample selection and day-of-the-month fixed effects, these days are not special and the risk premium is not that concentrated around macroeconomic announcement days. Regardless of the fraction of the equity premium that is earned on those days, there is *some* risk premium that is earned around these events and they reduce *some* uncertainty by disclosing important information to market participants. This alone makes these events an important object of interest for researchers. Together with Beckmeyer et al. (2019) and Kurov et al. (2020), all of the above mentioned papers revolve around studying the build-up and resolution of uncertainty around macroeconomic announcements. My addition with respect to this literature is to identify *why* some press conferences reduce more uncertainty than others. To this end, I compare stock returns on FOMC press conference days when reporters had a complex discussion with other days when the talks were arguably simpler according to a new measure of attention. This allows me to identify a channel through which the Fed reduces market uncertainty and affects asset prices: by discussing with financial reporters. This implies that the Chair reveals additional information during the Q&A sessions that is not redundant with the pre-written opening statements. My findings are consistent with Kroencke et al. (2018) who show that monetary policy affects the pricing of risk by identifying shocks to risky assets that are uncorrelated with changes in the risk-free rate (i.e. “FOMC risk shifts").

This paper also provides a contribution to the literature of machine learning methods used to analyze central banks communication. I quantify the degree of complexity of a discussion without relying on NLP techniques, hence avoiding their limitations.[1] This new alternative dataset of videos allows me to analyze the press events from a new angle. Indeed, I investigate the same events but leverage computer vision to extract a different signal which is the time spent by the Chair reading documents while answering questions. In other words, the NLP literature has focused on *what* is being said during the press conferences while I focus on *how* it is being said. This is accomplished by exploiting the images of the conferences and scrutinizing the human behavior. This information is not present in the transcripts and I argue that it is valuable for financial markets. However, it is likely that the signal I construct using videos could be augmented by sentiment measures extracted from text data. This is why I view my method as complementary to what has been done in the NLP literature. However, the combination of both approaches is left for future research. Another interesting use of machine learning to analyze FOMC conferences is present in Gomez Cram and Grotteria (2020). Their

[1] One common drawback of NLP methods in finance/economics is the need to create a dictionary of positive and negative words. The choice of which words belong to which set is somehow subjective. Another problem with more advanced methods is the necessity to label the data.

dataset is closely related to mine in the sense that they also use the videos from FOMC press conferences but only analyze the audio in order to match sentences of the Chair with market reactions in real time. In comparison, my paper is the first to use the images from these videos. Overall, I present a proof of concept that FOMC videos actually provide useful information for financial economists.

In accounting, papers like Elliott et al. (2012), Blankespoor et al. (2017) and Cade et al. (2020) have used video data to analyze the effects of disclosure through videos. However they do not use any systematic algorithm to extract the visual content which makes the approaches hardly scalable. Some authors like Akansu et al. (2017), Gong et al. (2019) or Hu and Ma (2020) use machines to process the videos but they focus on extracting emotions either from CEOs or entrepreneurs. None of their methods are suited to analyze FOMC press conferences because central bankers exerce an effort to appear as neutral as possible when they speak. In contrast to this literature, I develop a simpler tool that systematically computes the reading time of a person appearing in a video. This fully objective measure does not rely at all on emotions.

The rest of the paper is organized as follows. Section 3.2 establishes the methodology to construct the attention measure. Section 3.3 presents the main results and finally section 3.4 concludes. A technical discussion on computer vision algorithms can be found in appendix E.

## 3.2 Dataset and methodology

I use the video of each FOMC press conference (available on the Fed website) from their start in April 2011 to September 2020.[2] The market data consists of the time-series of the S&P500 index sampled at a frequency of 1-min. Each press conference can be decomposed into two parts. (i) The first one is an introductory statement in which the Chair reads a pre-written speech, detailing the recent decisions of the Fed. (ii) The second part is a Q&A session between financial reporters and the Fed Chair. I focus solely on the Q&A for the following reasons. Most of the literature analyzing press conferences has focused on the $1^{st}$ part (with a few rare exceptions) even though the Q&A occupies around 82% of the time of the press conference. Moreover, the unprepared character of the Q&A session means that the behavior of the Chair, when answering questions (whether he needs to read documents to answer questions or not for instance), does bring valuable information that has never been analyzed. Indeed, the Q&A is spontaneous and the Chair did not prepare answers to the reporters' questions. Using this data, the main problems I try to solve are

**H1:** How can we measure the complexity of a question and its associated answer?

**H2:** Do complex discussions contribute more to reduce uncertainty?

In order to answer these questions, I need to characterise the content of the press conferences.

[2] I remove the conference from the $15^{th}$ of March 2020 simply because there is no video available (it is only audio).

As previously explained, the existing literature has done so by assigning a sentiment score to the verbal content by combining text transcripts with some NLP algorithm. The new idea in my paper is to characterise a discussion between a reporter and the Chair of the FOMC, not by analyzing the language but rather by considering how the Chair reacts after being asked a question. To this end, I decide to focus on the following dimension: Does the Chair reply directly or does he read some internal documents in order to provide an answer? This information is available in the videos provided by the Fed but it needs to be extracted and converted into a numerical quantity that can serve as input for statistical inference tools. This is done by employing various computer vision algorithms that are new in finance but have been applied for years to solve engineering problems. In this paper, I focus on the economic mechanisms and the value of the information that can be extracted from this alternative data. Therefore I will keep the discussion of the methodology on a high (non-technical) level and invite the reader to consult appendix E for more details. The need for a technical discussion on computer vision can be (partially) avoided because every image processing can actually be easily illustrated. I will simply visually present the result of every computation by showing an image and what kind of information I extract from it. Given that a video is nothing but a collection of still images, I will use these two words interchangeably.

The first step is to construct facial landmarks $\boldsymbol{l} \in \mathbb{R}^2$ which are certain key points on a human face used to localize some specific regions like the eyes, the mouth, the jaw, etc. They can be visualized in figure E.1 in the appendix. In this paper, they will help me track certain movements of the Fed Chair during the press conferences when he is answering a question. Basically, I want to know every time the Chair is looking at some documents. This is accomplished in two steps. (i) First I extract the identity of the people in every frame. This is done via a technique called deep metric learning that is briefly explained in the appendix. I do not linger on this method because it does not add any economic intuition. This is solely used to filter out images where the Chair appears and disregard the others. (ii) Once I have isolated the frames with the Chair, I will only use the landmarks associated with the eyes. In figure 3.1 we can observe the facial landmarks (black dots) that are specific to them. Each eye is localized by 6 vectors, the so-called landmarks, $(\boldsymbol{l}_1, ..., \boldsymbol{l}_6)$ on the 2D plane that is the picture. When the eyes close and open the landmarks will move, effectively tracking their movements. These points are created using an ensemble of regression trees that is also explained in the appendix. From

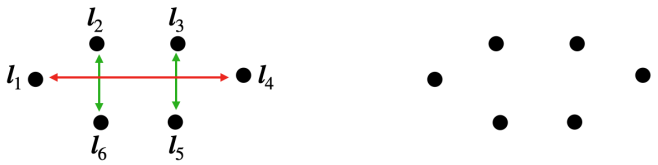



Figure 3.1 – Facial landmarks for the left eye and associated distances (colored arrows).

there I compute a measure of eyes openness. I want to compute a scalar value indicating how open or closed are the eyes at every point in time. For this purpose I use the eye aspect ratio

(EAR) developed in Cech and Soukupova (2016). On each still image (i.e. at one instant in time), I compute the EAR for eye $j$ by calculating the $L^2$ norm between the eye landmarks

$$\text{EAR}^j = \frac{\|\boldsymbol{l}_2^j - \boldsymbol{l}_6^j\| + \|\boldsymbol{l}_3^j - \boldsymbol{l}_5^j\|}{2\,\|\boldsymbol{l}_1^j - \boldsymbol{l}_4^j\|} \qquad j \in \{\text{left eye, right eye}\} \tag{3.1}$$

where $\boldsymbol{l}_1^j, \boldsymbol{l}_2^j, ..., \boldsymbol{l}_6^j$ are the facial landmarks characterizing one eye and depicted on the diagram in figure 3.1. The vertical distances are represented by green arrows and appear at the numerator of the EAR. The horizontal distance (red arrow) serves to normalize. The final EAR is a simple average for both eyes

$$\text{EAR} = \frac{\text{EAR}^{\text{left eye}} + \text{EAR}^{\text{right eye}}}{2}. \tag{3.2}$$

Computing this variable for each frame of the videos means that an EAR scalar value is associated with every single image in my dataset. I denote by $\text{EAR}_{i,t}$ the eye aspect ratio, during the press conference $i$ at instant (frame) $t$. For each FOMC press conference, I obtain a time series of eye aspect ratio for the Chair. An example is provided in the plot of figure 3.2. The blank spaces correspond to times when reporters ask questions and appear on camera. Given that the Chair is not visible during these periods I do not compute an EAR. The scalar

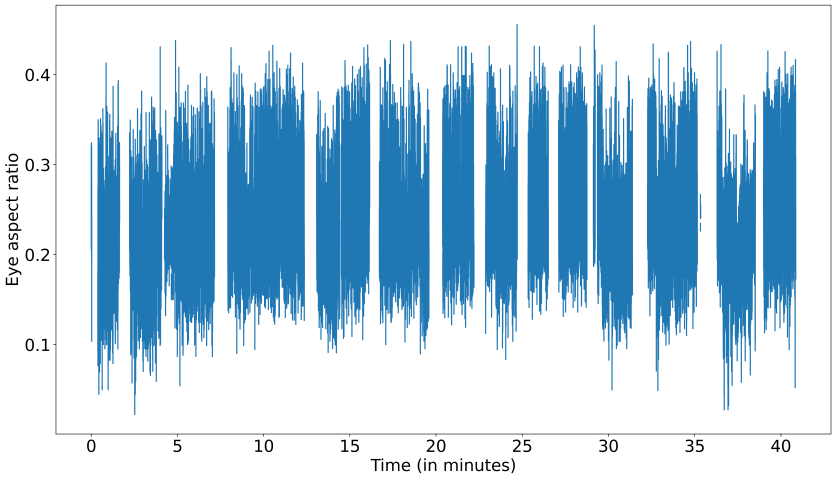



Figure 3.2 – Time series of the eye aspect ratio (EAR) of the Fed Chair during the Q&A session of the FOMC press conference (April, 29 2020). The blank spaces correspond to times when reporters ask questions and appear on camera. I therefore do not calculate an EAR during these periods.

quantity EAR provides a simple way to measure if the eyes are open or closed at every point in

time. The EAR takes high values when the eyes are wide open and approches 0 as they close. To convince the reader that this variable indeed captures what I intend, I provide an example of two video frames in figure 3.3 in order to compare the EAR in different situations. In figure 3.3a, the Chair Janet Yellen is not looking at the documents on the desk and the associated EAR is 0.33 (relatively high value). The convex hull connecting all the landmarks $\boldsymbol{l}_1, ..., \boldsymbol{l}_6$ (drawn in green around the eyes) creates a relatively large set. In the other picture 3.3b she is clearly reading and the EAR drops to 0.16. Here the convex envelope generates a smaller set. When the Chair spends a substantial amount of time reading, it will produce a series of $\text{EAR}_{i,t}$ that will be lower during this time period.


(a) EAR when the Chair is not reading.

(b) EAR when the Chair is reading a document.

Figure 3.3 – Convex hulls created by the landmarks $\boldsymbol{l}_1, ..., \boldsymbol{l}_6$ and associated eye aspect ratios (EARs).

It is natural to wonder what type of questions will cause some reading by the Chair. To clarify this, I report below a comparison of two questions asked by reporters. They are copied from the transcript of the press conference of September 21, 2016.

Q*: **Question from a reporter that does not lead to the consultation of internal documents by the Chair:** "Chair Yellen, at a time when the public is losing faith in many institutions, did the FOMC discuss the importance of today as an opportunity to dispel the thinking that the Fed is politically compromised or beholden to markets?"

Q': **Question from a reporter that does trigger substential reading from the Chair:** "Madam Chair, critics of the Federal Reserve have said that you look for any excuse not to hike, that the goalposts constantly move. And it looks, indeed, like there are new goalposts now when you say looking for further evidence and-and you suggest that it's evidence that labor-labor market slack is being taken up. Could you explain what for the time being means, in terms of a time frame, and what that further evidence you would look for in order to hike interest rates? And also, this notion that the goalposts seem to move, and that you've indeed introduced a new goalpost with this statement."

The whole idea of this paper is to differentiate between Q* and Q', not by studying the text content, but by analyzing how does the Chair behave when answering each question. The question Q' will be associated with a complex discussion because my measure of attention EAR will be low due to the reading from the Chair. On the other hand, the EAR stays relatively high when Janet Yellen answers question Q*. For simplicity, I focus solely on where the Chair looks while answering reporters' questions. More sophisticated measures incorporating extra facial landmarks on top of the ones locating the eyes could produce a more precise signal.

So far, for each press conference $i$ I have a time series of EAR. In order to compare the macroeconomic announcements, I decide to summarize the time series information into a variable $\Lambda_i$ that will take one single value per conference. This is done by integrating the reading time of the Chair for each FOMC meeting. The attention measure is therefore defined as

$$\Lambda_i = \int_0^{T_i} \mathbb{1}_{\{\text{EAR}_{i,t}<c\}} dt \tag{3.3}$$

where $T_i$ is the time at which the press conference finishes. The constant $c$ helps discriminate situations in which the person is consulting internal documents or not. When the EAR is low enough, the speaker is classified as looking down. The variable $\Lambda_i$ aggregates all the necessary information by measuring how much did the Chair look at his documents in a given press conference. The interpretation of $\Lambda_i$ is as follows. If the value is small, it means that the Chair did not spend much time looking at his documents during conference $i$. If on the other hand the value is large, the Chair spent a significant amount of time looking down. I argue

(and show later) that $\Lambda$ is directly proportional to the quantity of uncertainty that has been resolved during a Q&A session. Indeed, the Chair is more likely to look at documents when providing a complex answer. This in turn provides more relevant information to the market and thus reduces uncertainty. It is worth emphasizing that the spontaneity of the questions and answers is important for this analysis. Had the speaker received the questions in advance, this methodology would probably not work.

To conclude the methodology section, it is important to notice that even though I use machine learning methods to extract facial landmarks, the analysis is totally transparent. I would obtain approximately the same variables and results if I had personally watched with great attention all the press conferences and timed manually whenever the Chair is paying attention to the documents in his possession. Machine learning is being used only to automatize this procedure. The variable extracted from the computer vision algorithm $\text{EAR}_{i,t}$ is easily interpretable as an attention measure (i.e. where the person is looking). In the next sections, I will use this data as an input in a linear regression in order to explain the behavior of returns and uncertainty around the FOMC press conferences.

## 3.3 Main results

In this section I explore how the attention measure of the Chair $\Lambda$ can explain equity returns and quantify the uncertainty that has been resolved due to the conference. It is based on the premise that Q&A sessions involving complex discussions (associated with higher values of $\Lambda_i$) will further reduce uncertainty for financial markets, leading to higher stock returns and lower volatility.

Assuming that markets are rational, the relevant explanatory variable is the deviation from what is expected by participants given their information set $\mathscr{F}$. This is why I adopt the following model when trying to explain a generic market quantity $y$ (either returns or volatility)

$$y_i = \alpha + \beta \left(\Lambda_i - \mathbb{E}\left[\Lambda_i | \mathscr{F}_{i-1}\right]\right) + \boldsymbol{X}_i \boldsymbol{\gamma} + \varepsilon_i \tag{3.4}$$

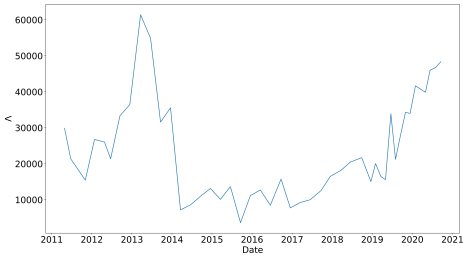
where $\boldsymbol{X}$ is a vector of control variables. One control will be called "market conditions" and consists of the return of the S&P500 over the last two months prior to the FOMC meeting. Another variable considered is the return of the S&P500 just before the beginning of the Q&A session denoted by $r^b$ and defined in equation (3.6). This is supposed to capture any pre-announcement effect and any information that was revealed during the introductory statement. The limited number of observations ($< 50$) prevents me from adding any more controls if I want to preserve some statistical power.

In order to optimally forecast $\Lambda_i$, we first need to find the best time series model for this variable (displayed in figure 3.4a). I run an augmented Dickey Fuller test (ADF) on $\Lambda_i$ and find

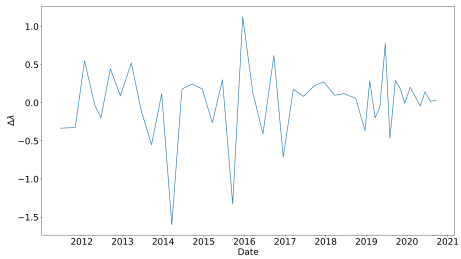
that the variable is neither constant nor trend stationary. The details are presented in table F.1 in the appendix. The results point toward the fact that $\Lambda_i$ follows a random walk. This implies that $\mathbb{E}[\Lambda_i|\mathcal{F}_{i-1}] = \Lambda_{i-1}$ and the main covariate of interest will be

$$\Delta\lambda_i = \lambda_i - \lambda_{i-1} \tag{3.5}$$

where $\lambda_i = \log(\Lambda_i)$. The new object $\Delta\lambda$ is stationary (see figure 3.4b).




(a) The variable $\Lambda$ is not stationary and appears to follow a random walk.




(b) The log difference of $\Lambda$ is stationary.

Figure 3.4 – Comparison of the level and log difference of the attention measure $\Lambda$.

I also construct various benchmark variables that might proxy the complexity of the discussion between reporters and the Fed Chair. These benchmarks are potential substitutes to $\Lambda$. They are constructed in a simpler way and I include them to show that my attention measure based on computer vision is not trivially redundant with easily accessible information. For this purpose, the benchmarks I consider are: # questions, Duration Q&A and Duration speech Chair. The exact construction of these variables is described in appendix F.

Each press conference takes place in two stages: (i) an introductory statement and (ii) a Q&A session. Given that I do not analyze the first part, I define four times $(\tau_i)_{i=1}^4$ that are relevant for the analysis. They are illustrated on the timeline in figure 3.5 and are constructed as follows.

- $\tau_1$ is 2 hours before the beginning of the Q&A session,
- $\tau_2$ is the start of the Q&A session,
- $\tau_3$ is the end of the press conference,
- $\tau_4$ is the end of the trading day.

These times will be used to construct returns and volatility measures around the FOMC announcements.

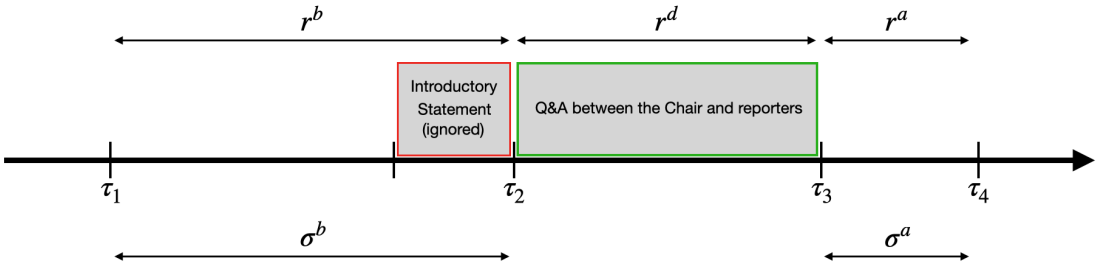



Figure 3.5 – Timeline of a typical FOMC press conference.

### 3.3.1 Explaining contemporaneous stock returns

For each press conference $i$, I define the following intraday returns which capture the S&P500 price ($P$) changes respectively before, during, and after the press conference

$$r_i^{\mathrm{b}} = \log\left(\frac{P_{i,\tau_2}}{P_{i,\tau_1}}\right), \tag{3.6}$$

$$r_i^{\mathrm{d}} = \log\left(\frac{P_{i,\tau_3}}{P_{i,\tau_2}}\right), \tag{3.7}$$

$$r_i^{\mathrm{a}} = \log\left(\frac{P_{i,\tau_4}}{P_{i,\tau_3}}\right). \tag{3.8}$$

I then run a linear regression

$$r_i^d = \alpha + \beta\Delta\lambda_i + \boldsymbol{X}_i\boldsymbol{\gamma} + \varepsilon_i \tag{3.9}$$

of the returns during the Q&A session onto the change in the discussion complexity measure. For comparison I also run similar regressions where I replace $\Delta\lambda$ with the log difference of the benchmark variables. The results are reported in table 3.3.1. The first observation is that all the betas of the attention measure and the benchmark variables are positive and of the same order of magnitude. The fact that all variables agree on the sign of the effect is reassuring since they are all measuring the same quantity to some extent. The two most significant variables are $\Delta\lambda$ and the duration of the speech of the Chair (with p-values below 1%). Interestingly, the variable constructed using the video data $\Delta\lambda$ is "better" at explaining equity returns in the sense that its associated t-stat is the highest (around 5) and has by far the highest $R^2$ that is 28.6%. This is not surprising since the duration of the Chair speech is simply measuring how long did the Chair speak during the Q&A session, disregarding anything else. While $\Lambda$ is also correlated to the length of the speech but contains the additional information that the Chair was paying close attention to important documents while answering questions. Hence $\Lambda$ gauges the intricacy of a Q&A session, which is arguably difficult to capture using

NLP techniques without making subjective choices (like choosing a dictionary). An interesting consequence of these results is that the Chair reveals additional information during the Q&A sessions that is not present in the pre-written opening statements. Looking at the last two columns of table 3.3.1 we see that controlling for the market conditions or the pre-Q&A return does not impact the value of the coefficient associated to $\Delta\lambda$ nor its significance.

Table 3.1 – Explaining contemporaneous stock returns

| | $r^d$ | | | | | |
|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) | (6) |
| const | -0.000 | -0.000 | -0.000 | -0.000 | -0.001 | -0.001 |
| | (0.001) | (0.001) | (0.001) | (0.001) | (0.001) | (0.001) |
| $\Delta\lambda$ | 0.005*** | | | | 0.005*** | 0.005*** |
| | (0.001) | | | | (0.001) | (0.001) |
| $\Delta$# questions | | 0.004 | | | | |
| | | (0.002) | | | | |
| $\Delta$Duration Q&A | | | 0.008** | | | |
| | | | (0.004) | | | |
| $\Delta$Duration speech chair | | | | 0.006*** | | |
| | | | | (0.002) | | |
| Market conditions | | | | | 0.012 | |
| | | | | | (0.013) | |
| $r^b$ | | | | | | 0.110 |
| | | | | | | (0.118) |
| Observations | 44 | 44 | 44 | 44 | 44 | 44 |
| $R^2$ | 0.286 | 0.057 | 0.093 | 0.179 | 0.302 | 0.301 |
| Adjusted $R^2$ | 0.269 | 0.035 | 0.072 | 0.160 | 0.268 | 0.267 |
| Residual Std. Error | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 | 0.004 |
| F Statistic | 16.815*** | 2.554 | 4.322** | 9.161*** | 8.856*** | 8.818*** |

*Note:* *p<0.1; **p<0.05; ***p<0.01

This table reports the regression statistics for the four different models, each analyzing the explanatory power of a different covariate. The dependent variable $r^d$ is the log return of the S&P500 during the Q&A session (between the beginning until the end). Standard errors are in parenthesis.

I also run a similar regression to (3.9) but replacing the dependent variable with the return after the conference $r^a$ and find no significant coefficients. This means that the information is incorporated immediately in the stock price over the time of the Q&A.

### 3.3.2 Impact on uncertainty

The previous section showed that when the change in $\lambda$ is high, stock returns tend to be higher. Given that this variable is independent from any sentiment measure, it is natural to expect that positive returns are caused by a reduction in uncertainty. This is what I argue in this section by

showing that $\Delta\lambda$ is negatively correlated with stock market volatility. For this purpose, I simply compute the realized variances before and after each press conference which are denoted by $\sigma_i^b$ and $\sigma_i^a$ respectively. The timeline is illustrated in figure 3.5. The formal construction of these measures is done as follows

$$\sigma_i^b = \sqrt{\frac{1}{\#\mathbb{S}_{\tau_1,\tau_2}} \sum_{\tau=\tau_1}^{\tau_2} r_{i,\tau}^2}, \tag{3.10}$$

$$\sigma_i^a = \sqrt{\frac{1}{\#\mathbb{S}_{\tau_3,\tau_4}} \sum_{\tau=\tau_3}^{\tau_4} r_{i,\tau}^2} \tag{3.11}$$

where $\mathbb{S}_{t,t'}$ is defined as the set containing all the returns between the times $t$ and $t'$ included. Equipped with this measure of change in market uncertainty, I run the main regression of interest

$$\sigma_i^a - \sigma_i^b = \alpha + \beta\Delta\lambda_i + \boldsymbol{X}_i\boldsymbol{\gamma} + \varepsilon_i. \tag{3.12}$$

Again, for comparison purposes I also replace $\Delta\lambda$ with all the log difference of the benchmark variables and report the results in table 3.3.2. The two most significant variables (with p-values below 5%) are the same than in the previous section: $\Delta\lambda$ and the duration of the Chair's speech. They again both agree on the sign of the effect in the sense that complex discussions reduce market volatility. And consistently with the previous results, $\Delta\lambda$ is the most precise signal in the sense that its associated t-stat is the highest (in absolute) with a value of 2.5. The $R^2$ is also the highest and around 12.7%.

As in the previous section, the control variables do not impact the sign nor the significance of my attention variable. However it is worth noticing that in column (6) of table 3.3.2, the pre-Q&A return $r^b$ is negative, highly significant, and boosts the adjusted $R^2$ to 31%.

Table 3.2 – Explaining the change in volatility before and after the Q&A

| | $(\sigma^a - \sigma^b) * 100$ | | | | | |
|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) | (6) |
| const | -0.002 | -0.002 | -0.002 | -0.002 | -0.002 | 0.003 |
| | (0.003) | (0.003) | (0.003) | (0.003) | (0.003) | (0.003) |
| $\Delta\lambda$ | -0.015** | | | | -0.016** | -0.013** |
| | (0.006) | | | | (0.006) | (0.005) |
| Δ# questions | | 0.002 | | | | |
| | | (0.011) | | | | |
| ΔDuration Q&A | | | -0.016 | | | |
| | | | (0.019) | | | |
| ΔDuration speech chair | | | | -0.019** | | |
| | | | | (0.009) | | |
| Market conditions | | | | | 0.026 | |
| | | | | | (0.064) | |
| $r^b$ | | | | | | -1.865*** |
| | | | | | | (0.510) |
| Observations | 44 | 44 | 44 | 44 | 44 | 44 |
| $R^2$ | 0.127 | 0.001 | 0.017 | 0.091 | 0.131 | 0.342 |
| Adjusted $R^2$ | 0.106 | -0.023 | -0.006 | 0.069 | 0.088 | 0.310 |
| Residual Std. Error | 0.019 | 0.020 | 0.020 | 0.019 | 0.019 | 0.017 |
| F Statistic | 6.108** | 0.032 | 0.731 | 4.197** | 3.080* | 10.645*** |

*Note:* *p<0.1; **p<0.05; ***p<0.01

This table reports the regression statistics for the four different models, each analyzing the explanatory power of a different covariate. The dependent variable $(\sigma^a - \sigma^b)$ is the difference in volatility (realized variation) of the S&P500 before and after the press conference. The time windows over which the realized variation is measured are illustrated in the timeline of figure 3.5. Standard errors are in parenthesis.

## 3.4 Conclusion

This paper develops a new measure of discussion complexity between the Fed Chair and reporters during the Q&A sessions of FOMC press conferences. It is accomplished by analyzing a new dataset of videos and taking advantage of tools from computer vision in order to measure how often the Chair needs to consult internal documents when answering questions. This variable is then showed to explain both contemporaneous equity returns and the change in volatility before and after the conference, even after controlling for general market conditions or pre-Q&A drift effects. On average, complex discussions lead to higher returns and lower volatility. This is consistent with recent findings in the literature that central banks impact the pricing of risk. My work allows to pin down a new mechanism through which press conferences impact the expectations of market participants. A by-product of this result is that there is additional information being revealed during the Q&A sessions that is not redundant with the opening statements. I am currently working on incorporating more data into the analysis by including the videos of the press conferences of the European Central Bank.

In general, my methodology is not constrained to macroeconomic events and can be useful to analyze the nonverbal communication of CEOs or politicians for instance.

# Conclusion

This thesis contributes to research on machine learning in empirical asset pricing. The focus is in particular on modelling temporal sequences and alternative data.

The first two chapters show the superior performance of LSTM networks at estimating volatility in an environment with sudden changes and regime shifts over statistical methods. One contribution is to show how LSTM networks help at detecting jumps or bubbles by solving a rolling window estimation problem. It would be interesting to pursue this direction and show that our methodology works in general better for any rolling window estimation. Another venue for future research would be to theoretically understand why certain networks outperform statistical estimators. The method presented in this thesis works better than statistical methods, but at the cost of the mathematical certainty of convergence.

The third chapter presents the first analysis of nonverbal communication of Fed officials and empirically demonstrates that this information impacts asset prices. The systematic study of nonverbal communication (either via audio files or images) in economics is still in its infancy. Future research could expand the set of methodologies, signals or the datasets by studying different central banks, CEOs or even politicians.

Finally, the recent rise of Transformer models for both time series modelling and computer vision means that they are potential challengers to LSTM networks and a thorough comparison of both architectures for financial applications would be relevant.

# A Appendix: LSTM networks

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[1] 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 $\phi$ 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

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

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:[2] 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.

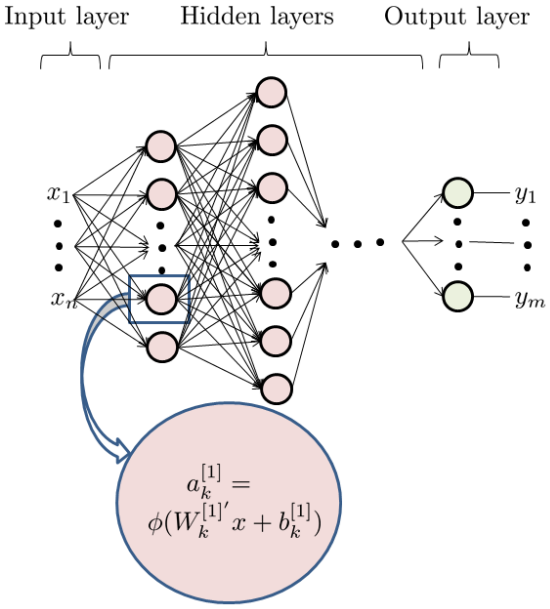



Figure A.1 – Deep Neural Network

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

$$h_t = \phi(W x_t + U h_{t-1} + b) \tag{A.2}$$

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

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

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. [3]

*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 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 $\sigma$ 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

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

[3] 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.

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 follows

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

$$h_t = f_t \otimes \tanh(c_t). \tag{A.8}$$

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

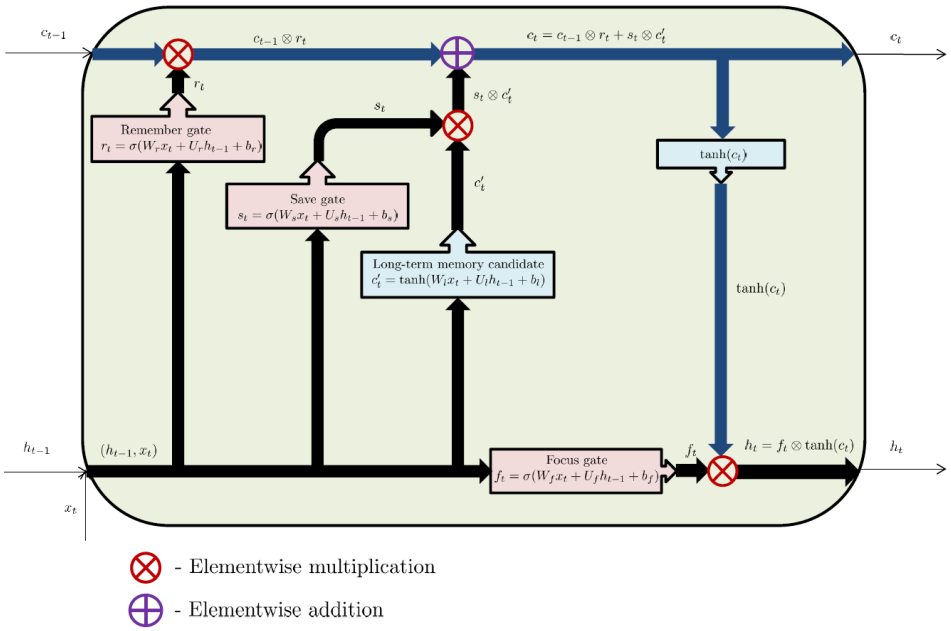



Figure A.2 – Transformation of the input inside the LSTM unit

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 [4] 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[5] 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.

[4] 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 $i = 1, ..., n$.

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

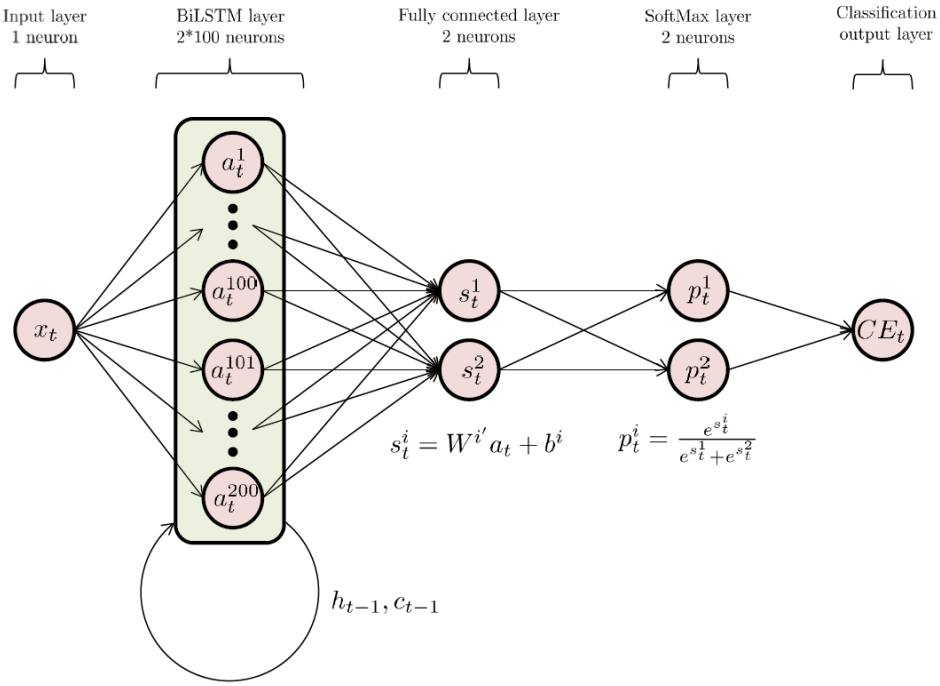



Figure A.3 – Network used in this paper

# B Appendix

If the price process is represented by the jump-diffusion model (1.1), then the realized variation is defined as

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

where $\Delta$ corresponds to the chosen frequency and $r_{\Delta,t+j\cdot\Delta}$ is the $j^{\text{th}}$ 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, Barndorff-Nielsen and Shephard (2004) introduced the realized bipower variation which is defined as

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

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

# C Appendix: Discussion on local martingales

In this section we aim at building intuition about true and strict local martingales. We will do so through the famous example of the doubling strategy. Imagine that a gambler is betting on the outcome of a coin toss. If the coin comes out head, he wins his bet. If it comes out tail he looses his bet.

Now assume that he chooses the following strategy: if he looses round $n$, he doubles his previous bet for the next round. He does so until he wins. As soon as he wins for the first time, he takes his gain and stops playing. Clearly, if the player is allowed to take infinite credit, he can bet endlessly until the coin comes out head and thus is guaranteed to walk away with a net gain of 1 dollar. However, for every finite number of trials it is a fair game: with high probability the player wins 1, and with very small probability he loses all his previous bets, such that the expected net gain is 0. Therefore his personal wealth is a true martingale.

However if the gambler is allowed to bet infinitely fast in a finite time interval, his wealth process turns into a strict local martingale and is expected to increase.

### C.0.1 Betting in discrete-time

Let $X_t^\pi$ denote the dollar value at time $t$ of the portfolio that invests in the strategy $\pi$ described above. We set $X_0^\pi = 0$ such that the gambler starts with zero initial wealth. Let $Z$ be the random variable that represents the outcome of the coin toss. It takes the value 1 if the coin lands on head and -1 otherwise. The coin is unbiased such that $\mathbb{P}(Z_n = 1) = \mathbb{P}(Z_n = -1) = 1/2$. It follows that

$$X_n^\pi = \begin{cases} 1 & \text{if } X_{n-1}^\pi = 1, \\ X_{n-1}^\pi + (1 - X_{n-1}^\pi) Z_n & \text{otherwise.} \end{cases}$$

As we see, until the first time the coin comes out heads, the value of the strategy $X_n^\pi$ is negative and the debt grows exponentially. As soon as the coin lands on head, the winning bet covers the previously occurred losses and provides the net gain of 1 dollar. It is easy to see that this is

a martingale.

For every finite number of tosses $N$ the probability of loss (i.e the probability of coin landing only on the tails all N times) is $\frac{1}{2^N}$ and the corresponding loss in this case is the sum of all the lost bets, including the current one, so it is $(-2^N + 1)$. The probability of winning is then $1 - \frac{1}{2^N}$ and the net gain in case of victory is 1.

Even though the expected gain stays zero in discrete-time independently of $N$, it is insightful to analyze what happens to the distribution as the number of bets rises to understand the continuous-time framework in the next section. For that, we simulate $10^7$ sample paths of such game, each representing one different gambler. We present the histograms of the players' wealth after $n$ rounds below. In line with the reasoning above, after the first coin toss in half of the cases the player leaves as a winner with \$1 and in the other half the player looses \$1 as displayed in figure C.1. Increasing the number of tosses to 4, we see in figure C.2 a shift in the distribution: most of the mass is concentrated on the winners side, with approximately $\frac{15}{16}$ of players being the winners of \$1 and $\frac{1}{16}$ of gamblers loosing \$15. This phenomenon accentuates as we increase further $n$: the probability of loosing decreases but the potential loss increases, thus decreasing the skewness of the distribution. The loss probability will disappear when we work in continuous-time, transforming the gambler's wealth into a strict local martingale.

While in theory the expectation of this discrete-time process is always 0, when increasing the number of trials (for example up to 25) we run into the *finite-sample problem*. The probability of loss becomes so small that our number of samples is just not high enough: simulations will tell us that among $10^7$ players after 25 rounds there was no losers and everyone eventually walked away with \$1. In this case the average terminal wealth is clearly 1. Here we already approximate a strict local martingale behavior that should only exist in continuous-time. But this picture is misleading and arises solely due to the computational limitations. We should remember that this is still a fair game! The probability of the loss is, however small, still nonzero, and the potential loss is so huge that it balances the mass of winners in expectation.

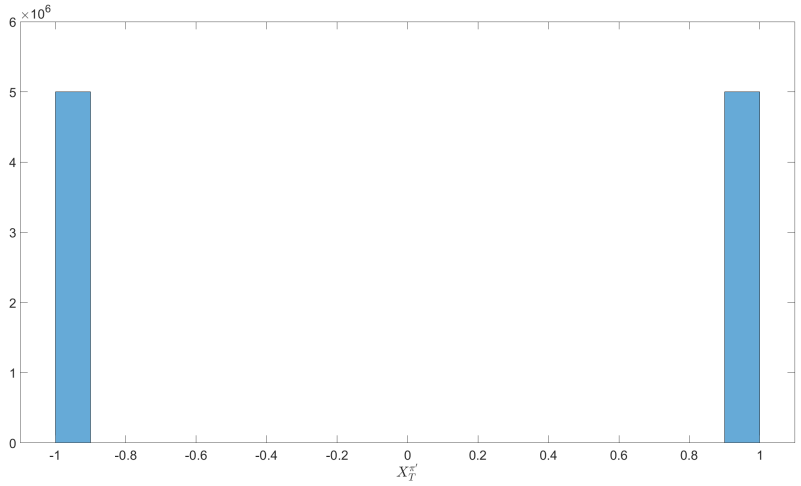



Figure C.1 – Distribution of the terminal wealth of the gamblers after 1 trial.

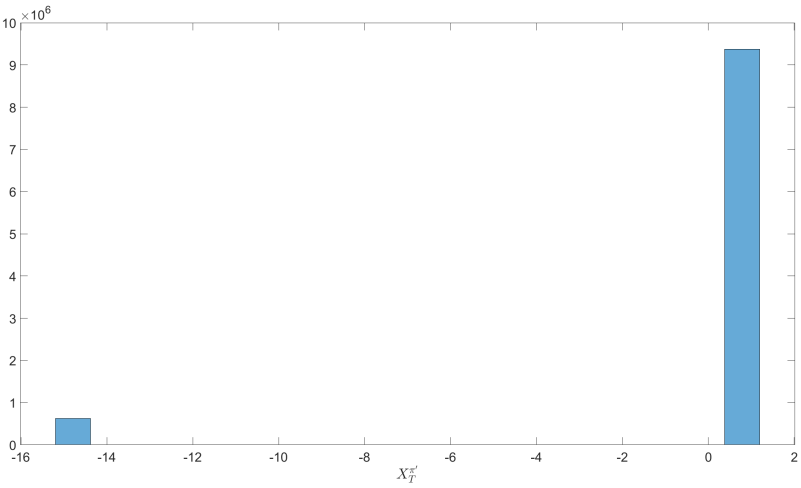



Figure C.2 – Distribution of the terminal wealth of the gamblers after 4 trials.

### C.0.2 Betting in continuous-time

The previously described strategy is pathological in the sense that even though for every finite number of trial it is fair, it stops being fair as soon as we allow for infinitely many trials. This pathology is translated into the strict local martingale property of the continuous-time process. Imagine that the player can speed up the time, such that after each lost trial he can bet faster and faster. In such a fantastic world the player can place infinitely many bets (thus eventually win $1) in a finite time interval (say 1 hour). Mathematically, we construct the following continuous-time process:

$$Y_t = \begin{cases} X_n^\pi & \text{if } 1 - \frac{1}{n} \leq t < 1 - \frac{1}{n+1}, \\ 1 & t \geq 1. \end{cases}$$

This process represents the dollar value of a portfolio which employs this doubling strategy. As a result, the player always ends up with the net gain of 1. This process is clearly not a martingale, since $\mathbb{E}[Y_0] = 0 \neq \mathbb{E}[Y_1] = 1$. However, it is a local martingale (under the localizing sequence of stopping times chosen as, for example, $\tau_n = \inf\{t : |X_t| \geq n\}$).

It is important to point that not all strict local martingales arise due to a doubling strategy. As pointed out in Dumas and Luciano (2017), in this example the strict local martingale arises due to the behavior of the agent (which bets faster and faster) while the underlying stochastic process $Z$ (the coin) itself has always the same distribution after each round since the variable is iid. In the bubble detection part of this paper, the process (2.6) itself might be a strict local martingale irrespective of the betting behavior of the investor. Therefore it is important to understand that when we detect strict local martingales (i.e. bubbles) and implement a long-short trading strategy, we do not rely on a doubling mechanism.

### C.0.3 Approximation of strict local martingales by discrete-time processes

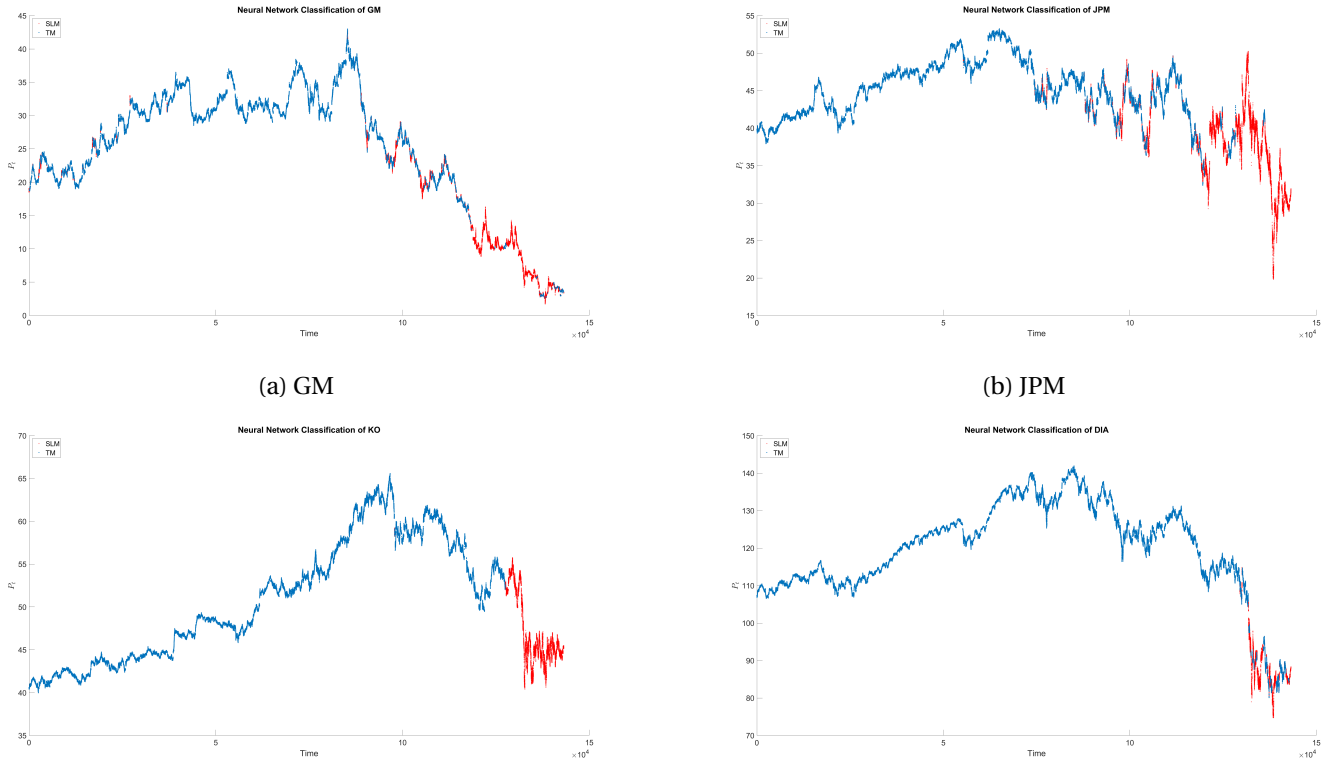
The final point of our discussion concerns the simulation of strict local martingales, a phenomenon that exists solely in continuous-time. Since computers cannot simulate continuous variables, we have to rely on a discretization scheme. So in fact, we are left with a discrete-time process, trying to study a phenomenon, that exists purely in continuous-time. However, this appears to cause no complication for our purpose. As we have seen in subsection C.0.1, the probability mass of the gambler's wealth is sparsely distributed. As $n$ rises, a higher probability is concentrated on the winning point, while a very small and further decreasing probability is shifting to the left, corresponding to loosing higher and higher amounts. At some point the probability becomes so small, that among the whole simulation set there is not enough samples for this probability to realise and create at least one loosing path. As a consequence, we end up with a process $X^\pi$ that approximates a strict local martingale behavior. Even though we still work in discrete-time and theoretically this process should be a martingale (staying at 0 in expectation), due to the absence of at least one loosing path the simulated process in fact increases on average.

The same logic applies to our simulations of the stock price $S_t$. We use Monte-Carlo to generate paths of the discretized version of the price in equation (2.13) where $\gamma_1 > 1$. Its continuous-time counterpart is a strict local martingale which is a supermartingale (since is bounded below by 0) and so has to go down in expectation. However even the simulated discrete-time process decreases on average while in theory it should not. This happens exactly because of an extremely sparse probability distribution as discussed in subsection C.0.1. With very high probability the process goes down (resembling an SLM) while with very small probability it goes extremely high and thus balances the average to stay a true martingale. Since we simulate

a finite number of paths, this highly unlikely exploding path just does not appear, leaving us with paths that resemble the SLM.

If we choose a time interval $\Delta t$ of $10^{-3}$ with maturity $T = 5$ for the simulations, we observe a decreasing average even if we simulate over $10^8$ different paths. Decreasing $\Delta t$ further for the same $T$ will accentuate this phenomenon.

# D Appendix




(a) GM

(b) JPM

(c) KO

(d) DIA

Figure D.1 – Bubbles detection from 2006 to 2008 included (3 years) using our LSTM network. The blue line represents the price under a non bubbly regime. The red line represents periods when the asset is in a bubble regime.

# E Appendix: Computer Vision

This appendix provides a brief overview of the machine learning techniques used in order to compute an eye aspect ratio (EAR) defined in equation (3.1). This paper uses videos as data, however, a video is simply a collection of still images indexed by time and therefore I will always talk about images when referring to the variables used as inputs. Each image $\boldsymbol{I}$ (also called frame) can be represented numerically by a tensor of dimension $\mathscr{D} = x \times y \times 3$. The $x$ and $y$ dimensions respectively correspond to the length and height of the image. Each pixel being an entry in the $x \times y$ matrix that corresponds to its horizontal and vertical location. The 3 represents the three primitive components of any color. Extracting the behavior of only a subset of people (the Chairs of the FOMC) present in the videos requires using a series of different algorithms one after another. They are all described below in the order used to process the data. My paper employs popular computer vision algorithms for which more details can for instance be found in the book Rosebrock (2017) or the associated online blog[1].

### E.0.1 Identity detection via deep metric learning

The first task when processing the data is to apply a face recognition algorithm. Meaning that on every single image, I want to know if the Chair of the Fed appears on it or not. For that I need a method that will take as input a frame $\boldsymbol{I}$ and output the identity of the person on it.[2] A powerful tool available is known as deep metric learning.

Let us first begin with a simplified example. Suppose you want to perform a classification of human pictures. That is you want to figure out if there is a human in a given picture or not (without being interested in the identity of the person). This is a simpler task than my original goal but I will build on it later. When one wants to classify labelled images, it is common to train a neural network (typically a convolutional network) that accepts an image as input and outputs a scalar value. For instance the output could be 1 if there is a human face in the

[1] https://www.pyimagesearch.com

[2] To simplify the explanations I will assume that in each picture there is only one face. The methods in this appendix do not need this assumption and I do not use it since it does not hold for my dataset.

picture and 0 otherwise. However, this classification is too simple for me because financial reporters also appear in the FOMC press conference videos and I am not interested in their behavior. This is why instead of using this algorithm, I use a slight modification that will help me filter out the images that do not contain the Fed Chair.

Deep metric learning is different in the sense that the output will be a vector $\boldsymbol{e} \in \mathbb{R}^n$ of embeddings where $n$ is the number of points used to characterize the human face.[3] Formally, our neural network will be a non-linear function $f(.|\boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$ that takes as input a still image $\boldsymbol{I} \in \mathbb{R}^{\mathscr{D}}$ and outputs a vector $\boldsymbol{e}$:

$$f(\boldsymbol{I}|\boldsymbol{\theta}) = \boldsymbol{e}. \tag{E.1}$$

The vector $\boldsymbol{e}$ is describing the face in the picture $\boldsymbol{I}$ in a mathematical way. This step is also called *encoding* the face into a vector. Training the network boils down to making sure the output vectors are close to each other when two pictures of the same person are used as inputs, and far when the persons are different. Suppose that we have two different images $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ that both contain the same person. We want to train the network such that the associated embedding vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are "close". If on the other hand we had two images of different persons, we would like the output vectors to be "far" from each other. In other words, the goal is to twist the network parameters $\boldsymbol{\theta}$ such that two pictures of the same person are classified as having an identical face. For instance any twenty different pictures of the same person should approximatively give the same output vector $\boldsymbol{e}$. In order to perform the training[4], I need to build a new database (different from the FOMC press conferences videos) of $M$ pictures including multiple images of each person I want to detect. Each image $\boldsymbol{I}$ in this set is indexed by $m \in \{1, 2, ..., M\}$. The label is simply the identity of the person in the picture (in my case it is the name of the Fed Chair: Ben Bernanke, Janet Yellen, Jerome Powell). It also helps to include pictures from random people in order to increase the performance of the neural network.

At this stage, one could wonder how does the network embed a face into a real valued vector $\boldsymbol{e}$? For humans it seems natural to compare features like the shape of the eyes, the mouth, the jaws, the length of the nose, etc in order to differentiate people. However, at this step of the process, I do not instruct the machine how to describe a face. I do not specifically constraint it to compare the attributes that seem natural to us humans. The network is learning by itself what are the characteristics that it should pick to properly encode a face. Later, for another task (extracting facial landmarks) I will ask the computer to detect features that are familiar to humans (I will especially be interested in the movement of the eyes).

Once the network is trained and has learnt to properly associate an output vector $\boldsymbol{e}$ to a specific

[3] It is standard to use $n = 128$.

[4] In order to save time, I use a pre-trained network that already knows how to encode human faces. I only re-train it on very few images to make sure that it is calibrated for my specific task.

person's face, I generate an embedding vector $\boldsymbol{e}_m$ for every labelled image $\boldsymbol{I}_m$ in my database. Every single face in my database will have an associated mathematical representation that "encodes" it in a vector $\boldsymbol{e}$. It will serve later to compare new unknown faces we want to classify to the labelled images in the database to tell how close they are to each others.

We are now ready to extract the identity of people in the new data. Let $\boldsymbol{I}'$ denote a picture the network has never seen before. By using the following mapping

$$f(\boldsymbol{I}'|\boldsymbol{\theta}) = \boldsymbol{e}' \tag{E.2}$$

we can characterize the face present on $\boldsymbol{I}'$ by using the encoding $\boldsymbol{e}'$. To find out the identity of the person in this new picture I use a simple $k$-NN model (with votes) to make the decision. I measure the distance between $\boldsymbol{e}'$ and all encodings associated with our known faces in our database $\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_M$. One way to do it is by using the $L^2$ norm

$$d_m = \|\boldsymbol{e}' - \boldsymbol{e}_m\| \tag{E.3}$$

where $d_m$ is the distance between the unknown face in image $\boldsymbol{I}'$ and the known face in image $\boldsymbol{I}_m$. If this distance is small enough (below some tolerance level $\epsilon$), I conclude that both pictures contain the same person. The problem here is that the distance could be small with multiple pictures, not necessarily of the same person (if the encoding is not good for some reason). This is where the voting mechanism enters. To deal with that I create a vector of binary classification $\boldsymbol{a} \in \{0,1\}^M$ whose $m^{\text{th}}$ element in denoted by $a_m$ and is constructed as follows

$$a_m = \begin{cases} 1 & \text{if } d_m < \epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

Basically, $a_m$ contains a yes/no answer to the question: is the person on image $\boldsymbol{I}'$ the same than on image $\boldsymbol{I}_m$? Then I can simply count the number of votes. For instance, for this new image $\boldsymbol{I}'$ I find that it is very close to 40 pictures of B. Bernanke, 2 pictures of Jerome Powell and 1 picture of a random person in my dataset. I will conclude that $\boldsymbol{I}'$ is an image of B. Bernanke. Repeating the above procedure for all the frames in my videos allows me to isolate the times when the Fed Chair appears and disregard moments when the reporters are on the screen.

### E.0.2 Facial landmarks

After applying the algorithm in the previous section, I now have filtered the parts of the videos that contain the Fed Chair since I was able to extract the identity of a person on an image.

Now I would like to know what is the Fed Chair looking at on every image. However the method used in section E.0.1 encodes a face in a vector $\boldsymbol{e}$ by selecting facial attributes that do not necessary make sense for us humans.[5] In fact, I do not know what are the exact features selected by the neural network to make the classification. For the previous task it did not matter but now it is a problem because ultimately I want to analyze the position of the eyes of a person. In this section I am going to use an algorithm that can detect face parts that are common to us when describing someone. In technical words I will identify facial landmarks.

A facial landmark is a point represented by a vector $\boldsymbol{l} \in \mathbb{R}^2$ on a 2D image. They are used to localize "important" regions of the face that are present on every human like the eyes, the eyebrows, the nose, the mouth as well as the jaws. An example is displayed in figure E.1. Facial landmarks (green dots) are in fact $(x, y)$-coordinates that map to important regions of the face on $\boldsymbol{I}$. To extract all the landmarks I use a pre-trained facial landmarks detector in the dlib Python library that is based on Kazemi and Sullivan (2014). They rely on an ensemble of regression trees that is trained on a set of human faces. Without entering too much into details, their ensemble method can be thought as a function $g(.|\boldsymbol{\Psi})$ that is parametrized by a vector $\boldsymbol{\Psi}$ and performs the following mapping

$$g(\boldsymbol{I}|\boldsymbol{\Psi}) = \boldsymbol{L}. \tag{E.4}$$

The matrix $\boldsymbol{L} \in \mathbb{R}^{2\times k}$ collects all the landmark vectors such that

$$\boldsymbol{L} = \begin{pmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \dots & \boldsymbol{l}_k \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \dots & x_k \\ y_1 & y_2 & \dots & y_k \end{pmatrix}. \tag{E.5}$$

Regarding the dimensions of $\boldsymbol{L}$, there are 2 rows because each $\boldsymbol{l}$ is a vector on a 2D image, and $k$ is the number of landmarks we want to extract.[6] Once $\boldsymbol{L}$ is known, I simply keep the landmarks characterizing the eyes and disregard the others. Each eye is is represented by six points (they are drawn in figure 3.1). In order to track the eye movements in my database of videos, I extract a matrix $\boldsymbol{L}$ for every frame $\boldsymbol{I}$ under the constraint that it contains the Chair of the Fed. Then I can easily compute the so called eye aspect ratio (EAR) according to equation (3.1).

[5] In order to distinguish two people a human would likely compare the color of the eyes, the length of the nose, the structure of the jaws, etc. However this is not (necessary) how a computer does it.

[6] I use $k = 68$.

Figure E.1 – Picture of Jerome Powell during an FOMC press conference along with all the 68 facial landmarks (green circles) obtained using the pre-trained detector from the dlib Python library.

In this appendix, I explain the main intuition behind the computer vision algorithms without discussing all the details. The practical implementation of the above algorithms in sections E.0.1 and E.0.2 requires some additional steps. One of them is that the user must pre-process the images by first detecting the position of the faces (finding the $(x, y)$-coordinates of a box surrounding the face on image $\boldsymbol{I}$). This can be done in a multitude of ways, including for instance using an other neural network specifically dedicated to that task. In my implementation I use a combination of Histogram of Oriented Gradients (HOG method) and linear Support Vector Machine (SVM). Other small steps are required in order to effectively implement the methods. They can all be found in the above mentioned resources.

# F Appendix

The benchmark variables are constructed as follows

- # questions: The log of the number of questions asked by reporters during a press conference. This is simply constructed by counting the number of interrogation marks in the transcripts.
- Duration Q&A: The log of the time length of the Q&A part of the conference.
- Duration speech Chair: The log of the total time period during which the Fed Chair was speaking. This is constructed using the video data but could also be closely approximated by counting the number of words in the Chair answers in the transcripts.

Whenever I use the notation $\Delta$ in front of a variable, it means that I take the difference between the FOMC meeting $i$ and $i-1$.

Table F.1 – Results of the ADF test performed on the total reading time of the FOMC Chair during a press conference $\Lambda_i$. The null hypothesis is that a unit root is present in a time series.

| Type | t-stat | p-value |
|---|---|---|
| Constant | -1.12 | 0.71 |
| Trend | -1.65 | 0.77 |

# Bibliography


Aït-Sahalia, Y. (2004). Disentangling diffusion from jumps. *Journal of Financial Economics*, 74(3):487–528.

Akansu, A., Cicon, J., Ferris, S. P., and Sun, Y. (2017). Firm Performance in the Face of Fear: How CEO Moods Affect Firm Performance. *Journal of Behavioral Finance*, 18(4):373–389.

Andersen, T. G., Bollerslev, T., and Diebold, F. X. (2007a). Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility. *Review of Economics and Statistics*, 89(4):701–720.

Andersen, T. G., Bollerslev, T., and Dobrev, D. (2007b). 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):125–180.

Andersen, T. G., Dobrev, D., and Schaumburg, E. (2009). Jump-robust volatility estimation using nearest neighbor truncation.

Au Yeung, J. F., kai Wei, Z., Chan, K. Y., Lau, H. Y., and Yiu, K. F. C. (2019). Jump detection in financial time series using machine learning algorithms. *Soft Computing*.

Barndorff-Nielsen, O. E. and Shephard, N. (2004). Power and Bipower Variation with Stochastic Volatility and Jumps. *Journal of Financial Econometrics*, 2(1):1–37.

Barndorff-Nielsen, O. E. and Shephard, N. (2006). Econometrics of testing for jumps in financial economics using bipower variation. *Journal of Financial Econometrics*, 4(1):1–30.

Barndorff-Nielsen, O. E., Shephard, N., and Winkel, M. (2006). Limit theorems for multipower variation in the presence of jumps. *Stochastic Processes and their Applications*, 116(5):796–806.

Beckmeyer, H., Grunthaler, T., and Branger, N. (2019). The Fed Call: FOMC Announcements and Stock Market Uncertainty.

Blankespoor, E., Hendricks, B. E., and Miller, G. S. (2017). Perceptions and Price: Evidence from CEO Presentations at IPO Roadshows. *Journal of Accounting Research*, 55(2):275–327.

Boudt, K., Croux, C., and Laurent, S. (2011). Robust estimation of intraweek periodicity in volatility and jump detection. *Journal of Empirical Finance*, 18(2):353–367.

Cade, N. L., Koonce, L., and Mendoza, K. I. (2020). Using video to disclose forward-looking information: the effect of nonverbal cues on investors' judgments. *Review of Accounting Studies*, 25(4):1444–1474.

Cech, J. and Soukupova, T. (2016). Real-Time Eye Blink Detection using Facial Landmarks.

Christensen, K., Oomen, R. C., and Podolskij, M. (2014). Fact or friction: Jumps at ultra high frequency. *Journal of Financial Economics*, 114(3):576–599.

Cieslak, A., Morse, A., and Vissing-Jorgensen, A. (2019). Stock Returns over the FOMC Cycle. *Journal of Finance*, 74(5):2201–2248.

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

Corsi, F. (2009). A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, 7(2):174–196.

Corsi, F., Pirino, D., and Renò, R. (2010). Threshold bipower variation and the impact of jumps on volatility forecasting. *Journal of Econometrics*, 159(2):276–288.

Delbaen, F. and Shirakawa, H. (2002). No arbitrage condition for positive diffusion price processes. *Asia-Pacific Financial Markets*, 9(3-4):159–168.

Dumas, B. and Luciano, E. (2017). *The Economics of Continuous-Time Finance*. MIT Press Books, edition 1 edition.

Dumitru, A. M. and Urga, G. (2012). Identifying jumps in financial assets: A comparison between nonparametric jump tests. *Journal of Business and Economic Statistics*, 30(2):242–255.

Elliott, W. B., Hodge, F. D., and Sedor, L. M. (2012). Using online video to announce a restatement: Influences on investment decisions and the mediating role of trust. *Accounting Review*, 87(2):513–535.

Ernst, R., Gilbert, T., and Hrdlicka, C. M. (2019). More Than 100% of the Equity Premium: How Much Is Really Earned on Macroeconomic Announcement Days? *SSRN Electronic Journal*.

Genon-Catalot, V. and Jacod, J. (1993). On the estimation of the diffusion coefficient for multi-dimensional diffusion processes. *Annales de l'I.H.P. Probabilités et statistiques*, 29(1):119–151.

Gomez Cram, R. and Grotteria, M. (2020). Real-time Price Discovery via Verbal Communication: Method and Application to Fedspeak. *SSRN Electronic Journal*, pages 1–44.

Gong, M., Zhang, Z., and Jia, M. (2019). Lie Detectors? How Entrepreneurs' Facial Expressions During IPO Roadshow Presentations Predict New Venture Misconduct Behaviors. *IEEE Transactions on Engineering Management*, pages 1–12.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). LSTM: A Search Space Odyssey. *Transactions on Neural Networks and Learning Systems*.

Gu, S., Kelly, B., and Xiu, D. (2019). Empirical Asset Pricing via Machine Learning.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hu, A. and Ma, S. (2020). Persuading Investors: A Video-Based Study.

Hu, G. X., Pan, J., Wang, J., and Zhu, H. (2019). Premium for Heightened Uncertainty: Solving the FOMC Puzzle. *SSRN Electronic Journal*.

Hugonnier, J. (2012). Rational asset pricing bubbles and portfolio constraints. *Journal of Economic Theory*, 147(6):2260–2302.

Jarrow, R., Kchia, Y., and Protter, P. (2011). How to detect an asset bubble. *SIAM Journal on Financial Mathematics*, 2(1):839–865.

Jarrow, R. and Protter, P. (2012). Discrete versus continuous time models: Local martingales and singular processes in asset pricing theory. *Finance Research Letters*, 9(2):58–62.

Jarrow, R. A. (2015). Asset Price Bubbles. *Annual Review of Financial Economics*, 7(1):201–218.

Jarrow, R. A., Protter, P., and Shimbo, K. (2007). *Asset price bubbles in complete markets*.

Jarrow, R. A., Protter, P., and Shimbo, K. (2010). Asset price bubbles in incomplete markets. *Mathematical Finance*, 20(2):145–185.

Jiang, G. J. and Oomen, R. C. A. (2007). Testing for Jumps When Asset Prices are Observed with Noise - A "Swap Variance" Approach.

Kazemi, V. and Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1867–1874.

Kroencke, T. A., Schmeling, M., and Schrimpf, A. (2018). The FOMC Risk Shift. *SSRN Electronic Journal*.

Kurov, A., Wolfe, M. H., and Gilbert, T. (2020). The disappearing pre-FOMC announcement drift. *Finance Research Letters*.

Lee, S. S. and Mykland, P. A. (2008). Jumps in financial markets: A new nonparametric test and jump dynamics. *Review of Financial Studies*, 21(6):2535–2563.

Lucca, D. O. and Moench, E. (2015). The Pre-FOMC announcement drift. *Journal of Finance*, 70(1):329–371.

Mäkinen, M., Kanniainen, J., Gabbouj, M., and Iosifidis, A. (2018). Forecasting of Jump Arrivals in Stock Prices: New Attention-based Network Architecture using Limit Order Book Data.

Mijatović, A. and Urusov, M. (2012). On the martingale property of certain local martingales. *Probability Theory and Related Fields*, 152(1-2):1–30.

Mukherjee, A., Peng, W., Swanson, N. R., and Yang, X. (2019). Financial econometrics and big data: A survey of volatility estimators and tests for the presence of jumps and co-jumps.

Obayashi, Y., Protter, P., and Yang, S. (2017). The lifetime of a financial bubble. *Mathematics and Financial Economics*, 11(1):45–62.

Podolskij, M. and Ziggel, D. (2010). New tests for jumps in semimartingale models. *Statistical Inference for Stochastic Processes*, 13(1):15–41.

Protter, P. (2001). A partial introduction to financial asset pricing theory. *Stochastic Processes and their Applications*, 91(2):169–203.

Protter, P. (2013). A mathematical theory of financial bubbles. *Paris-Princeton Lectures on Mathematical Finance*, pages 1–108.

Rosebrock, A. (2017). *Deep Learning for Computer Vision with Python*. PYIMAGESEARCH.

Savor, P. and Wilson, M. (2013). How Much Do Investors Care About Macroeconomic Risk ? Evidence from Scheduled Economic Announcements. *The Journal of Financial and Quantitative Analysis*, 48(2):343–375.

Tauchen, G. and Todorov, V. (2008). Volatility Jumps.

Theodosiou, M. G. and Zikes, F. (2012). A Comprehensive Comparison of Nonparametric Tests for Jumps in Asset Prices. *SSRN Electronic Journal*, 44(0).

# Alexis Marchal

Swiss Finance Institute at EPFL • Extranef 129 • Quartier UNIL-Chamberonne • CH-1015 Lausanne
+41 78 635 71 55 • alexis.marchal@epfl.ch • www.alexismarchal.com

## EDUCATION

**Ph.D. Candidate in Finance** *2016 - Aug. 2021 (expected)*
EPFL & Swiss Finance Institute
Advisors: Pierre Collin-Dufresne and Julien Hugonnier

**M.Sc. dual degree in Economics** *2014 - 2016*
University of Geneva & Economics School of Louvain

**B.Sc. Economics and Management** *2011 - 2014*
Université Catholique de Louvain

## WORKING PAPERS

**Risk & returns around FOMC press conferences: a novel perspective from computer vision**, 2020

Presentations: AFFI (2021), SFI Research Days (Gerzensee, 2021), Intelligent Systems (IntelliSys) (Amsterdam, 2021)

**Deep Learning for Asset Bubbles Detection**, 2020 with Oksana Bashchenko

Presentations (*incl. by co-authors): EPFL-UNIL Brown Bag (Lausanne, 2020), SFI Research Days* (Gerzensee, 2020), Webinar DISA-LNU: stochastic analysis, statistics and machine learning* (2020), Webinar B-TU Cottbus-Senftenberg on: Stochastics* (2020)

**Deep Learning, Jumps, and Volatility Bursts**, 2019 with Oksana Bashchenko

Presentations (*incl. by co-authors): EPFL-UNIL Brown Bag* (Lausanne, 2019), Workshop on the Systemic Impact of Digitalization on Finance (Zurich, 2019), Young Swiss Economists Meeting - YSEM* (Poster session, Zurich, 2020), SFI Research Days (Gerzensee, 2020), Webinar DISA-LNU: stochastic analysis, statistics and machine learning (2020), EFMA* (2021)

**An Equilibrium Model of Decentralized & Walrasian Markets**, 2019

Presentations: SFI Research Days (Gerzensee, 2019)

## WORK IN PROGRESS

**Research Team in the Finance Crowd Analysis Project (fincap)**

**News analysis with S-BERT**

## COMPUTER SKILLS

- Python:
  - Time-Series: Pandas, NumPy, Scikit Learn, Statsmodels
  - NLP: Hugging Face, PyTorch
  - Computer Vision: OpenCV, Dlib, Keras, TensorFlow
- Others: Matlab, Mathematica, Stata, Git, LaTeX, Web scraping (with Selenium)

## TECHNICAL SKILLS

- Deep Learning (CNN, LSTM, BERT)
- Time-Series Analysis, Stochastic Calculus, Option Theory

## RESEARCH INTERESTS

Asset Pricing, Machine Learning

## EXPERIENCE

**Teaching assistant**
Financial applications of blockchains and distributed ledgers, EPFL *2021 - present*

Derivatives (Prof. Julien Hugonnier), EPFL *2018 - present*

- Course content: Pricing theory for European and American options in discrete and continuous time (trees and PDEs)
- Responsibilities: Hold weekly exercise sessions, grade weekly assignments and exams

## AWARDS AND FELLOWSHIPS

Swiss Finance Institute Best Discussant Doctoral Award, 2021

Best teaching assistant of 2020 class, Master in Financial Engineering (MFE), EPFL

Swiss Finance Institute PhD student fellowship, 2016-2017

## LANGUAGES

French (native)
English (fluent)

## MISCELLANEOUS

**PhD Student Representative**, Doctoral Program in Finance, EPFL *Oct. 2019 - present*